Intel® Arria® 10 Hard Processor System Technical Reference Manual

Updated for Intel® Quartus® Prime Design Suite: **21.2**

Online Version
Send Feedback
a10_5v4
ID: 683711
Version: 2022.08.22

Contents

1. Intel® Arria® 10 Hard Processor System Technical Reference Manual Revision History....14

2. Introduction to the Hard Processor System....28
2.1. Features of the HPS....31
2.2. HPS Block Diagram and System Integration....32
2.2.1. HPS Block Diagram....32
2.2.2. Cortex-A9 MPCore....33
2.2.3. HPS Interfaces....33
2.2.4. System Interconnect....34
2.2.5. On-Chip Memory....35
2.2.6. Flash Memory Controllers....36
2.2.7. Support Peripherals....38
2.2.8. Interface Peripherals....41
2.2.9. CoreSight Debug and Trace....44
2.2.10. Hard Processor System I/O Pin Multiplexing....45
2.3. Endian Support....45
2.4. Introduction to the Hard Processor System Address Map....46
2.4.1. HPS Address Spaces....46
2.4.2. HPS Peripheral Region Address Map....48

3. Clock Manager....52
3.1. Features of the Clock Manager....52
3.2. Clock Manager Block Diagram and System Integration....54
3.2.1. L4 Peripheral Clocks....55
3.2.2. Boot Clock....56
3.3. Functional Description of the Clock Manager....57
3.3.1. Clock Manager Building Blocks....57
3.3.2. PLL Integration....59
3.3.3. Hardware-Managed and Software-Managed Clocks....61
3.3.4. Hardware Sequenced Clock Groups....61
3.3.5. Software Sequenced Clocks....63
3.3.6. Resets....65
3.3.7. Security....66
3.3.8. Interrupts....67
3.4. Clock Manager Address Map and Register Definitions....67
3.4.1. clkmgr_clkmgrp Address Map....67
3.4.2. clkmgr_mainpllgrp Address Map....82
3.4.3. clkmgr_perpllgrp Address Map....106
3.4.4. clkmgr_alteragrp Address Map....131

4. Reset Manager....134
4.1. Reset Manager Block Diagram and System Integration....136
4.1.1. Reset Controller....137
4.1.2. Security Reset Functions....140
4.1.3. Module Reset Signals....140
4.1.4. Slave Interface and Status Register....145
4.2. Functional Description of the Reset Manager....146

4.2.1. Reset Sequencing....147
4.2.2. Reset Pins....151
4.2.3. Reset Effects....152
4.2.4. Reset Handshaking....152
4.3. Reset Manager Address Map and Register Definitions....153

5. FPGA Manager....154
5.1. Features of the FPGA Manager....154
5.2. FPGA Manager Block Diagram and System Integration....155
5.3. Functional Description of the FPGA Manager....156
5.3.1. FPGA Configuration....156
5.3.2. FPGA Status....157
5.3.3. Error Message Extraction....157
5.3.4. Data AES Decryption....157
5.3.5. Boot Handshake....157
5.3.6. General Purpose I/O....158
5.3.7. Clock....158
5.3.8. Reset....158
5.4. FPGA Manager Address Map and Register Definitions....158

6. System Manager....159
6.1. Features of the System Manager....159
6.2. System Manager Block Diagram and System Integration....160
6.3. Functional Description of the System Manager....161
6.3.1. Boot Configuration and System Information....161
6.3.2. Additional Module Control....162
6.3.3. Boot ROM Code....164
6.3.4. FPGA Interface Enables....165
6.3.5. ECC and Parity Control....166
6.3.6. Preloader Handoff Information....167
6.3.7. Clocks....167
6.3.8. Resets....167
6.4. System Manager Address Map and Register Definitions....167

7. SoC Security....168
7.1. Security Manager....169
7.1.1. Security Manager Block Diagram....170
7.1.2. Functional Overview....171
7.1.3. Functional Description....171
7.2. Security System Extensions....187
7.2.1. TrustZone....187
7.2.2. Arria 10 HPS Secure Firewalls....189

8. System Interconnect....200
8.1. About the System Interconnect....201
8.1.1. Features of the System Interconnect....201
8.1.2. System Interconnect Block Diagram and System Integration....201
8.1.3. Arria 10 HPS Secure Firewalls....206
8.1.4. About the Rate Adapters....207
8.1.5. About the SDRAM L3 Interconnect....207
8.1.6. About Arbitration and Quality of Service....210
8.1.7. About the Service Network....210

8.1.8. About the Observation Network....211
8.2. Functional Description of the System Interconnect....211
8.2.1. System Interconnect Address Spaces....212
8.2.2. Secure Transaction Protection....220
8.2.3. System Interconnect Master Properties....220
8.2.4. System Interconnect Slave Properties....222
8.2.5. System Interconnect Clocks....223
8.2.6. System Interconnect Resets....224
8.2.7. Functional Description of the Rate Adapters....225
8.2.8. Functional Description of the Firewalls....225
8.2.9. Functional Description of the SDRAM L3 Interconnect....227
8.2.10. Functional Description of the Arbitration Logic....232
8.2.11. Functional Description of the QoS Generators....232
8.2.12. Functional Description of the Observation Network....236
8.3. Configuring the System Interconnect....238
8.3.1. Configuring the Rate Adapters....238
8.3.2. Configuring the SDRAM Scheduler....239
8.3.3. Configuring the Hard Memory Controller....240
8.3.4. Configuring the Quality of Service Logic....241
8.4. System Interconnect Address Map and Register Definitions....247

9. HPS-FPGA Bridges....248

9.1. Features of the HPS-FPGA Bridges....248
9.2. Arria 10 HPS-FPGA Bridges Block Diagram and System Integration....250
9.3. Functional Description of the HPS-FPGA Bridges....250
9.3.1. Functional Description of the FPGA-to-HPS Bridge....250
9.3.2. Functional Description of the HPS-to-FPGA Bridge....263
9.3.3. Functional Description of the Lightweight HPS-to-FPGA Bridge....275
9.3.4. Clocks and Resets....278
9.3.5. Data Width Sizing....280
9.4. HPS-FPGA Bridges Address Map and Register Definitions for Arria 10....281

10. Cortex-A9 Microprocessor Unit Subsystem....282

10.1. Features of the Cortex-A9 MPU Subsystem....282
10.2. Cortex-A9 MPU Subsystem Block Diagram and System Integration....283
10.2.1. Cortex-A9 MPU Subsystem with System Interconnect....283
10.2.2. Cortex-A9 MPU Subsystem Internals....284
10.3. Cortex-A9 MPCore....285
10.3.1. Functional Description....285
10.3.2. Implementation Details....286
10.3.3. Cortex-A9 Processor....286
10.3.4. Interactive Debugging Features....287
10.3.5. L1 Caches....288
10.3.6. Preload Engine....288
10.3.7. Floating Point Unit....289
10.3.8. NEON Multimedia Processing Engine....289
10.3.9. Memory Management Unit....290
10.3.10. Performance Monitoring Unit....293
10.3.11. Arm Cortex-A9 MPCore Timers....293
10.3.12. Generic Interrupt Controller....294
10.3.13. Global Timer....301

10.3.14. Snoop Control Unit....302
10.3.15. Accelerator Coherency Port....303
10.4. L2 Cache....310
10.4.1. Functional Description....310
10.5. CPU Prefetch....316
10.6. TrustZone....317
10.6.1. Secure Partitioning....317
10.6.2. Virtual Processor Operation....318
10.6.3. Secure Debug....319
10.7. Debugging Modules....319
10.7.1. Program Trace....320
10.7.2. Event Trace....320
10.7.3. Cross-Triggering....321
10.8. Clocks....321
10.9. Cortex-A9 MPU Subsystem Register Implementation....321
10.9.1. Cortex-A9 MPU Subsystem Address Map for Arria 10....321
10.9.2. L2 Cache Controller Address Map for Arria 10....321

11. CoreSight Debug and Trace....322
11.1. Features of CoreSight Debug and Trace....323
11.2. Arm CoreSight Documentation....323
11.3. CoreSight Debug and Trace Block Diagram and System Integration....324
11.4. Functional Description of CoreSight Debug and Trace....324
11.4.1. Debug Access Port....324
11.4.2. System Trace Macrocell....326
11.4.3. Trace Funnel....326
11.4.4. CoreSight Trace Memory Controller....327
11.4.5. AMBA Trace Bus Replicator....328
11.4.6. Trace Port Interface Unit....328
11.4.7. Embedded Cross Trigger System....328
11.4.8. Program Trace Macrocell....332
11.4.9. HPS Debug APB Interface....333
11.4.10. FPGA Interface....333
11.4.11. Debug Clocks....335
11.4.12. Debug Resets....336
11.5. CoreSight Debug and Trace Programming Model....337
11.5.1. Coresight Component Address....337
11.5.2. STM Channels....338
11.5.3. CTI Trigger Connections to Outside the Debug System....339
11.5.4. Configuring Embedded Cross-Trigger Connections....341
11.6. CoreSight Debug and Trace Address Map and Register Definitions....342
11.6.1. stm Address Map....342
11.6.2. dap Address Map....343

12. Error Checking and Correction Controller....345
12.1. ECC Controller Features....345
12.2. ECC Supported Memories....345
12.3. ECC Controller Block Diagram and System Integration....346
12.4. ECC Controller Functional Description....347
12.4.1. Overview....347
12.4.2. ECC Structure....347

12.4.3. Memory Data Initialization....349
12.4.4. Indirect Memory Access....350
12.4.5. Error Logging....355
12.4.6. ECC Controller Interrupts....357
12.4.7. ECC Controller Initialization and Configuration....360
12.4.8. ECC Controller Clocks....361
12.4.9. ECC Controller Reset....362
12.5. ECC Controller Address Map and Register Descriptions....362

13. On-Chip Memory....364
13.1. On-Chip RAM....364
13.1.1. Features of the On-Chip RAM....364
13.1.2. On-Chip RAM Block Diagram and System Integration....365
13.1.3. Functional Description of the On-Chip RAM....365
13.2. Boot ROM....367
13.2.1. Features of the Boot ROM....367
13.2.2. Boot ROM Block Diagram and System Integration....368
13.2.3. Functional Description of the Boot ROM....368
13.3. On-Chip Memory Address Map and Register Definitions....369

14. NAND Flash Controller....370
14.1. NAND Flash Controller Features....370
14.2. NAND Flash Controller Block Diagram and System Integration....371
14.3. NAND Flash Controller Signal Descriptions....371
14.4. Functional Description of the NAND Flash Controller....372
14.4.1. Discovery and Initialization....372
14.4.2. Bootstrap Interface....373
14.4.3. Configuration by Host....374
14.4.4. Local Memory Buffer....375
14.4.5. Clocks....375
14.4.6. Resets....376
14.4.7. Indexed Addressing....377
14.4.8. Command Mapping....378
14.4.9. Data DMA....383
14.4.10. ECC....387
14.5. NAND Flash Controller Programming Model....390
14.5.1. Basic Flash Programming....391
14.5.2. Flash-Related Special Function Operations....395
14.6. NAND Flash Controller Address Map and Register Definitions....404

15. SD/MMC Controller....405
15.1. Features of the SD/MMC Controller....405
15.1.1. SD Card Support Matrix....407
15.1.2. MMC Support Matrix....407
15.2. SD/MMC Controller Block Diagram and System Integration....408
15.3. SD/MMC Controller Signal Description....408
15.4. Functional Description of the SD/MMC Controller....409
15.4.1. SD/MMC/CE-ATA Protocol....409
15.4.2. BIU....410
15.4.3. CIU....422
15.4.4. Clocks....438
15.4.5. Resets....439

15.4.6. Voltage Switching....440
15.5. SD/MMC Controller Programming Model....442
15.5.1. Software and Hardware Restrictions†....442
15.5.2. Initialization†....444
15.5.3. Controller/DMA/FIFO Buffer Reset Usage....451
15.5.4. Non-Data Transfer Commands....451
15.5.5. Data Transfer Commands....453
15.5.6. Transfer Stop and Abort Commands....459
15.5.7. Internal DMA Controller Operations....461
15.5.8. Commands for SDIO Card Devices....463
15.5.9. CE-ATA Data Transfer Commands....466
15.5.10. Card Read Threshold....473
15.5.11. Interrupt and Error Handling....476
15.5.12. Booting Operation for eMMC and MMC....477
15.6. SD/MMC Controller Address Map and Register Definitions....489

16. Quad SPI Flash Controller....490
16.1. Features of the Quad SPI Flash Controller....490
16.2. Quad SPI Flash Controller Block Diagram and System Integration....491
16.3. Quad SPI Flash Controller Signal Description....492
16.4. Functional Description of the Quad SPI Flash Controller....493
16.4.1. Overview....493
16.4.2. Data Slave Interface....493
16.4.3. SPI Legacy Mode....497
16.4.4. Register Slave Interface....498
16.4.5. Local Memory Buffer....499
16.4.6. DMA Peripheral Request Controller....499
16.4.7. Arbitration between Direct/Indirect Access Controller and STIG....501
16.4.8. Configuring the Flash Device....501
16.4.9. XIP Mode....503
16.4.10. Write Protection....503
16.4.11. Data Slave Sequential Access Detection....503
16.4.12. Clocks....504
16.4.13. Resets....504
16.4.14. Interrupts....505
16.5. Quad SPI Flash Controller Programming Model....506
16.5.1. Setting Up the Quad SPI Flash Controller....506
16.5.2. Indirect Read Operation with DMA Disabled....507
16.5.3. Indirect Read Operation with DMA Enabled....508
16.5.4. Indirect Write Operation with DMA Disabled....508
16.5.5. Indirect Write Operation with DMA Enabled....509
16.5.6. XIP Mode Operations....509
16.6. Quad SPI Flash Controller Address Map and Register Definitions....511

17. DMA Controller....512
17.1. Features of the DMA Controller....512
17.2. DMA Controller Block Diagram and System Integration....514
17.3. Functional Description of the DMA Controller....515
17.3.1. Peripheral Request Interface....516
17.4. DMA Controller Address Map and Register Definitions....519
17.4.1. DMA Controller Address Map and Register Definitions....519

18. Ethernet Media Access Controller.....520
18.1. Features of the Ethernet MAC.....521
18.1.1. MAC.....521
18.1.2. DMA.....522
18.1.3. Management Interface.....522
18.1.4. Acceleration.....522
18.1.5. PHY Interface.....522
18.2. EMAC Block Diagram and System Integration.....523
18.3. EMAC Signal Description.....524
18.3.1. HPS EMAC I/O Signals.....525
18.3.2. FPGA EMAC I/O Signals.....527
18.3.3. PHY Management Interface.....529
18.4. EMAC Internal Interfaces.....530
18.4.1. DMA Master Interface.....530
18.4.2. Timestamp Interface.....530
18.4.3. System Manager Configuration Interface.....532
18.5. Functional Description of the EMAC.....532
18.5.1. Transmit and Receive Data FIFO Buffers.....533
18.5.2. DMA Controller.....535
18.5.3. Descriptor Overview.....548
18.5.4. IEEE 1588-2002 Timestamps.....560
18.5.5. IEEE 1588-2008 Advanced Timestamps.....566
18.5.6. IEEE 802.3az Energy Efficient Ethernet.....570
18.5.7. Checksum Offload.....571
18.5.8. Frame Filtering.....571
18.5.9. Clocks and Resets.....576
18.5.10. Interrupts.....579
18.6. Ethernet MAC Programming Model.....579
18.6.1. System Level EMAC Configuration Registers.....580
18.6.2. EMAC FPGA Interface Initialization.....581
18.6.3. EMAC HPS Interface Initialization.....583
18.6.4. DMA Initialization.....584
18.6.5. EMAC Initialization and Configuration.....585
18.6.6. Performing Normal Receive and Transmit Operation.....586
18.6.7. Stopping and Starting Transmission.....586
18.6.8. Programming Guidelines for Energy Efficient Ethernet.....586
18.6.9. Programming Guidelines for Flexible Pulse-Per-Second (PPS) Output.....588
18.7. Ethernet MAC Address Map and Register Definitions.....590

19. USB 2.0 OTG Controller.....591
19.1. Features of the USB OTG Controller.....592
19.1.1. Supported PHYS.....593
19.2. USB OTG Controller Block Diagram and System Integration.....594
19.3. USB 2.0 ULPI PHY Signal Description.....595
19.4. Functional Description of the USB OTG Controller.....596
19.4.1. USB OTG Controller Block Description.....596
19.4.2. Local Memory Buffer.....600
19.4.3. Clocks.....600
19.4.4. Resets.....600
19.4.5. Interrupts.....602

19.5. USB OTG Controller Programming Model....603
19.5.1. Enabling ECC....603
19.5.2. Enabling SPRAM ECCs....603
19.5.3. Host Operation....603
19.5.4. Device Operation....605
19.6. USB 2.0 OTG Controller Address Map and Register Definitions....607

20. SPI Controller....608
20.1. Features of the SPI Controller....608
20.2. SPI Block Diagram and System Integration....609
20.2.1. SPI Block Diagram....609
20.3. SPI Controller Signal Description....609
20.3.1. Interface to HPS I/O....610
20.3.2. FPGA Routing....610
20.4. Functional Description of the SPI Controller....611
20.4.1. Protocol Details and Standards Compliance....611
20.4.2. SPI Controller Overview....612
20.4.3. Transfer Modes....615
20.4.4. SPI Master....616
20.4.5. SPI Slave....620
20.4.6. Partner Connection Interfaces....624
20.4.7. DMA Controller Interface....629
20.4.8. Slave Interface....629
20.4.9. Clocks and Resets....630
20.5. SPI Programming Model....631
20.5.1. Master SPI and SSP Serial Transfers....632
20.5.2. Master Microwire Serial Transfers....634
20.5.3. Slave SPI and SSP Serial Transfers....636
20.5.4. Slave Microwire Serial Transfers....637
20.5.5. Software Control for Slave Selection....637
20.5.6. DMA Controller Operation....638
20.6. SPI Controller Address Map and Register Definitions....642

21. I^2C Controller....643
21.1. Features of the I^2C Controller....643
21.2. I^2C Controller Block Diagram and System Integration....644
21.3. I^2C Controller Signal Description....645
21.4. Functional Description of the I^2C Controller....647
21.4.1. Feature Usage....647
21.4.2. Behavior....647
21.4.3. Protocol Details....649
21.4.4. Multiple Master Arbitration....653
21.4.5. Clock Frequency Configuration....655
21.4.6. SDA Hold Time....656
21.4.7. DMA Controller Interface....656
21.4.8. Clocks....657
21.4.9. Resets....657
21.5. I^2C Controller Programming Model....657
21.5.1. Slave Mode Operation....658
21.5.2. Master Mode Operation....662
21.5.3. Disabling the I^2C Controller....664

21.5.4. Abort Transfer....665
21.5.5. DMA Controller Operation....665
21.6. I 2 C Controller Address Map and Register Definitions....669

22. UART Controller....670

22.1. UART Controller Features....670
22.2. UART Controller Block Diagram and System Integration....671
22.3. UART Controller Signal Description....672
22.3.1. HPS I/O Pins....672
22.3.2. FPGA Routing....672
22.4. Functional Description of the UART Controller....673
22.4.1. FIFO Buffer Support....673
22.4.2. UART(RS232) Serial Protocol....673
22.4.3. Automatic Flow Control....674
22.4.4. Clocks....676
22.4.5. Resets....676
22.4.6. Interrupts....677
22.5. DMA Controller Operation....679
22.5.1. Transmit FIFO Underflow....680
22.5.2. Transmit Watermark Level....680
22.5.3. Transmit FIFO Overflow....682
22.5.4. Receive FIFO Overflow....682
22.5.5. Receive Watermark Level....682
22.5.6. Receive FIFO Underflow....682
22.6. UART Controller Address Map and Register Definitions....683

23. General-Purpose I/O Interface....684

23.1. Features of the GPIO Interface....684
23.2. GPIO Interface Block Diagram and System Integration....685
23.3. Functional Description of the GPIO Interface....685
23.3.1. Debounce Operation....685
23.3.2. Pin Directions....686
23.3.3. Taking the GPIO Interface Out of Reset....686
23.4. GPIO Interface Programming Model....686
23.5. General-Purpose I/O Interface Address Map and Register Definitions....687

24. Timer....688

24.1. Features of the Timer....688
24.2. Timer Block Diagram and System Integration....688
24.3. Functional Description of the Timer....689
24.3.1. Clocks....690
24.3.2. Resets....690
24.3.3. Interrupts....690
24.4. Timer Programming Model....691
24.4.1. Initialization....691
24.4.2. Enabling the Timer....691
24.4.3. Disabling the Timer....691
24.4.4. Loading the Timer Countdown Value....691
24.4.5. Servicing Interrupts....692
24.5. Timer Address Map and Register Definitions....692

25. Watchdog Timer 693
25.1. Features of the Watchdog Timer 693
25.2. Watchdog Timer Block Diagram and System Integration 694
25.3. Functional Description of the Watchdog Timer 694
25.3.1. Watchdog Timer Counter 694
25.3.2. Watchdog Timer Pause Mode 695
25.3.3. Watchdog Timer Clocks 695
25.3.4. Watchdog Timer Resets 696
25.4. Watchdog Timer Programming Model 696
25.4.1. Setting the Timeout Period Values 696
25.4.2. Selecting the Output Response Mode 696
25.4.3. Enabling and Initially Starting a Watchdog Timer 697
25.4.4. Reloading a Watchdog Counter 697
25.4.5. Pausing a Watchdog Timer 697
25.4.6. Disabling and Stopping a Watchdog Timer 697
25.4.7. Watchdog Timer State Machine 697
25.5. Watchdog Timer Address Map and Register Definitions 699

26. Hard Processor System I/O Pin Multiplexing 700
26.1. Features of the HPS I/O Block 700
26.2. HPS I/O Block Diagram and System Integration 701
26.3. Functional Description of the HPS I/O 703
26.3.1. Dedicated I/O Pins 703
26.3.2. Shared I/O Pins 704
26.3.3. FPGA Access 705
26.3.4. Control Registers 705
26.3.5. Configuring HPS I/O Multiplexing 708
26.4. Test Considerations 709
26.5. I/O Pin MUX Address Map and Register Definitions for Arria 10 709

27. Introduction to the HPS Component 710
27.1. MPU Subsystem 711
27.2. Arm CoreSight Debug Components 711
27.3. Interconnect 711
27.4. HPS-to-FPGA Interfaces 711
27.5. Memory Controllers 712
27.5.1. HPS SDRAM Controller 712
27.6. Support Peripherals 714
27.7. Interface Peripherals 714
27.8. On-Chip Memories 715

28. Instantiating the HPS Component 716
28.1. Using the HPS Parameter Editor 716
28.2. FPGA Interfaces 717
28.2.1. General Interfaces 717
28.2.2. FPGA-to-HPS SDRAM Interface 719
28.2.3. DMA Peripheral Request 720
28.2.4. Security Manager 720
28.2.5. Interrupts 720
28.2.6. AXI Bridges 722
28.3. Configuring HPS Clocks and Resets 723

28.3.1. Alternate Clock Source from FPGA..... 723
28.3.2. User Clocks..... 723
28.3.3. Reset Interfaces..... 724
28.3.4. Peripheral FPGA Clocks..... 725
28.4. Configuring Peripheral Pin Multiplexing..... 725
28.4.1. Configuring Peripherals..... 726
28.4.2. Connecting Unassigned Pins to GPIO..... 727
28.4.3. Peripheral Signals Routed to FPGA 727
28.5. Configuring the External Memory Interface..... 728
28.6. Using the Address Span Extender Component..... 728
28.7. Generating and Compiling the HPS Component..... 729

29. HPS Component Interfaces..... 731
29.1. Memory-Mapped Interfaces..... 731
29.1.1. FPGA-to-HPS Bridge..... 731
29.1.2. HPS-to-FPGA and Lightweight HPS-to-FPGA Bridges..... 732
29.1.3. FPGA-to-HPS SDRAM Interface..... 733
29.1.4. EMIF Conduit..... 734
29.2. Clocks..... 734
29.2.1. Alternative Clock Inputs to HPS PLLs..... 734
29.2.2. User Clocks..... 734
29.2.3. AXI Bridge FPGA Interface Clocks..... 734
29.2.4. SDRAM Clocks..... 734
29.2.5. Peripheral FPGA Clocks..... 735
29.3. Resets..... 735
29.3.1. HPS-to-FPGA Reset Interfaces..... 736
29.3.2. HPS External Reset Request..... 736
29.3.3. Peripheral Reset Interfaces..... 736
29.4. Debug and Trace Interfaces..... 736
29.4.1. FPGA System Trace Macrocell Events Interface..... 736
29.4.2. FPGA Cross Trigger Interface..... 736
29.4.3. Debug APB Interface..... 737
29.5. Peripheral Signal Interfaces..... 737
29.5.1. Platform Designer (Standard) Peripheral Port Interface Mapping..... 737
29.5.2. DMA Controller Interface..... 744
29.6. Other Interfaces..... 745
29.6.1. MPU Standby and Event Interfaces..... 745
29.6.2. General Purpose Signals..... 746
29.6.3. FPGA-to-HPS Interrupts..... 746
29.6.4. Boot from FPGA Interface..... 746
29.6.5. Security Manager..... 746

30. Simulating the HPS Component..... 747
30.1. Simulation Flows..... 748
30.1.1. Setting Up the HPS Component for Simulation..... 749
30.1.2. Generating the HPS Simulation Model in Platform Designer (Standard)..... 751
30.1.3. Running the Simulations..... 751
30.2. Clock and Reset Interfaces..... 755
30.2.1. Clock Interface..... 755
30.2.2. Reset Interface..... 756
30.3. FPGA-to-HPS AXI Slave Interface..... 757

30.4. HPS-to-FPGA AXI Master Interface....757
30.5. Lightweight HPS-to-FPGA AXI Master Interface....758
30.6. HPS-to-FPGA MPU Event Interface....758
30.7. Interrupts Interface....758
30.8. HPS-to-FPGA Debug APB Interface....760
30.9. FPGA-to-HPS System Trace Macrocell Hardware Event Interface....760
30.10. HPS-to-FPGA Cross-Trigger Interface....760
30.11. FPGA-to-HPS DMA Handshake Interface....761
30.12. Boot from FPGA Interface....762
30.13. Security Manager Anti-Tamper Signals Interface....762
30.14. EMIF Conduit....762
30.15. Pin MUX and Peripherals....762
30.15.1. HPS Conduit Interfaces Connecting to the HPS I/O....763
30.15.2. HPS Conduit Interfaces Connecting to the FPGA....768

A. Booting and Configuration....771
A.1. Boot Overview....771
A.2. FPGA Configuration Overview....772
A.3. Booting and Configuration Options....772
A.4. Boot Definitions....775
A.4.1. Reset....775
A.4.2. Boot ROM....775
A.4.3. Boot Select....776
A.4.4. Flash Memory Devices for Booting....782
A.4.5. Clock Select....792
A.4.6. I/O Configuration....796
A.4.7. L4 Watchdog 0 Timer....797
A.4.8. Second-Stage Boot Loader....798
A.4.9. Secure Boot....799
A.5. Boot ROM Flow....800
A.6. Second-Stage Boot Flow....804
A.6.1. Typical Boot Flow (Non-Secure)....805
A.6.2. Secure Boot Flow....807
A.6.3. HPS State on Entry to the Second-Stage Boot Loader....810
A.6.4. Loading the Second-Stage Boot Loader Image....811
A.7. FPGA Configuration....813
A.7.1. Full FPGA Configuration Flow Through HPS....814
A.7.2. Early I/O Release FPGA Configuration Flow Through HPS....815
A.7.3. Arria 10 SoC FPGA Configuration Sequence Through FPGA Manager....817
A.8. FPGA Reconfiguration....819
A.8.1. Full FPGA Reconfiguration....819
A.8.2. Arria 10 SoC FPGA Partial Reconfiguration Sequence Through FPGA Manager. 821

1. Intel® Arria® 10 Hard Processor System Technical Reference Manual Revision History

Table 1. Intel® Arria® 10 Hard Processor System Technical Reference Manual Revision History Summary

Chapter	Date of Last Update
Introduction to the Hard Processor System	August 22, 2022
Clock Manager	July 22, 2017
Reset Manager	July 22, 2017
FPGA Manager	November 2, 2015
System Manager	May 27, 2016
SoC Security	February 23, 2018
System Interconnect	January 21, 2021
HPS-FPGA Bridges	July 22, 2017
Cortex*-A9 Microprocessor Unit Subsystem Revision History	May 26, 2021
CoreSight* Debug and Trace	July 29, 2017
Error Checking and Correction Controller	November 2, 2015
On-Chip Memory	August 18, 2014
NAND Flash Controller	May 27, 2016
SD/MMC Controller	July 8, 2021
Quad SPI Flash Controller	August 18, 2020
DMA Controller	July 22, 2017
Ethernet Media Access Controller	August 22, 2022
USB 2.0 OTG Controller	November 2, 2015
SPI Controller	November 2, 2015
I^2C Controller	November 2, 2015
UART Controller	November 2, 2015
General-Purpose I/O Interface	December 15, 2014
Timer	August 18, 2014
Watchdog Timer	November 2, 2015
Hard Processor System I/O Pin Multiplexing	June 14, 2019
Introduction to the HPS Component	May 3, 2016
Instantiating the HPS Component	May 27, 2016

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Chapter	Date of Last Update
HPS Component Interfaces	May 27, 2016
Simulating the HPS Component	May 27, 2016
Booting and Configuration	July 22, 2017

Document Version	Changes
2022.08.22	Removed RGMII because it does not support FPGA IO
2017.05.31	Removed HMCREGS row from HPS Peripheral Region Map table. Accesses to this address block are not supported.
2016.10.28	Renamed MPU Subsystem to Cortex-A9 MPCore*
2016.05.27	Corrected the HPS-FPGA powering scheme.
2016.05.03	Removed low-power double data rate 3 (LPDDR3) as a supported device.
2015.11.02	Updated the link to the Memory Maps.
2015.05.04	Corrected HPS-FPGA powering scheme.
2014.12.15	Maintenance release
2014.08.18	Initial release

Introduction to the Hard Processor System on page 28

Document Version	Changes
2017.07.22	• Replaced instances of EOSC1 pin with correct pin name, HPS_CLK1. • Clarified that the osc1_clk signal is sourced from the HPS_CLK1 input pin.
2016.10.28	• Added mpuclk register (offset 0x0) to i_clk_mgr_alteragrp
2016.05.27	• Removed references to f2h_emac*_ap_clk in the *Arria 10 Top Level Clocks* table. The Ethernet application interface (also called the switch interface) is not supported. • Removed clk9cntr (offset 0x44) and clk15cntr (offset 0x5C) registers from the i_clk_mgr_mainpllgrp • Removed clk9cntr register (offset 0x44) from the i_clk_mgr_perpllgrp • Clarified the src fields of the clk*cntr registers in the i_clk_mgr_mainpllgrp and i_clk_mgr_perpllgrp Removed references to PLL counter outputs C9-C15 from the following topics: • Boot Clock • PLLs • FREF, FVCO, and FOUT Equations
2016.05.03	Added a section titled "L4 Peripheral Clocks".
2015.11.02	Updated Sections: • Clock Manager Block Diagram and System Integration • PLL Integration • Software Sequenced Clocks • Clock Gating • Boot Clock Added Sections: • Updating PLL Settings without a System Reset • MPU Clock Scaling

continued...

Document Version	Changes
	Updates: • Removed references to PLL output C9 used as HMC PLL reference clock.
2015.05.04	• Updated Block Diagram with HMC block • Added Arria 10 Top Level Clocks table • Updated PLL Integration in Clock Manager figure • Added Address Map and Register Descriptions
2014.12.15	Clock Manager Block Diagram. Updated mux output route. NOC clock added. Peripheral Clocks block update C15 input for PLL1 has been removed throughout document.
2014.08.18	Initial release.

Clock Manager on page 52

Document Version	Changes
2017.07.22	• Removed references to EOSC1 and replaced them with the correct external oscillator pin name, HPS_CLK1. • Clarified that the osc1_clk signal is sourced from the HPS_CLK1 pin.
2016.10.28	Maintenance release.
2016.05.03	Maintenance release.
2015.11.02	Updated "Reset Pins" section.
2015.05.04	"Slave Interface" and "Status Register" sections updated.
2014.12.15	Maintenance release.
2014.08.18	Initial release.

Reset Manager on page 134

Document Version	Changes
2015.11.02	• Provided more information for the configuration schemes for the dedicated pins. • Added missing address maps and register definitions.
2015.05.04	Maintenance release
2014.12.15	Maintenance release
2014.08.18	Initial release

FPGA Manager on page 154

Document Version	Changes
2016.10.28	Maintenance release.
2016.05.27	Removed the following references to the Ethernet application interface: • app_clk_sel content in the *EMAC* section • emac_*_switch bits in the fpgaintf_en_3 register • app_clk_sel bits in the emac* registers The Ethernet application interface (also called the switch interface) is not supported.
2016.05.03	Maintenance release.

continued...

Document Version	Changes
2015.11.02	Maintenance release.
2015.05.04	Maintenance release.
2014.12.15	• Block Diagram updated: Slave port defined • "NAND Flash Controller" section updated • "USB 2.0 OTG" section updated • "EMAC" section updated
2014.08.18	Initial release.

System Manager on page 159

Table 2. SoC Security Revision History

Date	Version	Changes
July 2017	2017.07.21	• Removed references to EOSC1 and replaced them with the correct external oscillator pin name, HPS_CLK1. • Clarified that the osc1_clk signal is sourced from the HPS_CLK1 pin.
October 2016	2016.10.28	Maintenance release
May 2016	2016.05.27	Maintenance release
May 2016	2016.05.03	• Added note regarding blocked firewall transaction responses in the "Secure Firewall" section • Updated content and added figures to "JTAG" section
November 2015	2015.11.02	Clarified initialization steps in "Secure Initialization Overview" section
May 2015	2015.05.04	Added Address Map and Register information.
December 2014	2014.12.15	• Added "Secure Fuses" section under "Secure Initialization" section. • Added summary of "Security State" and "Security Check" and "Secure Serial Interface" features. • Added "MPU" and "JTAG" sub-sections to the "Secure Debug" section. • Added information regarding CSEL programming in the "Clock Configuration" section. • Added "FPGA Security Features" section
August 2014	2014.08.18	Initial Release

SoC Security on page 168

Table 3. System Interconnect Revision History

Document Version	Changes
2021.01.21	Updated *Arria 10 HPS Available Address Maps* to explain how to access the registers that are connected to the HPS-to-FPGA AXI* Master Bridge.
2019.01.01	Updated ddrConf bitfield description in the ddr_T_main_Scheduler_Ddrconf register to include encodings.
2018.09.24	Updated the following section: • *ECC Read Behavior*

continued...

Document Version	Changes
2017.05.31	• Clarify relationship between quality of service (QoS) and arbitration • Add QoS examples • Remove 32-bit bus from L3 interconnect to hard memory controller in the *SDRAM L3 Interconnect Block Diagram and System Integration* section. • Remove *Hard Memory Controller Memory Mapped Registers* section • Remove `io48_hmc_mmr` Address map and registers from *System Interconnect Address Map and Register Definitions* section
2016.10.28	• "System Interconnect Slave Interfaces" table: Corrected acceptance values for Lightweight HPS-to-FPGA Bridge and Lightweight HPS-to-FPGA Bridge • "Controlling Quality of Service from Software": Added note about register access • "Configuring SDRAM Burst Sizes": Refers to guidelines for selecting burst sizes • "Sharing I/O between the EMIF and the FPGA": New section, discusses constraints on sharing I/Os with EMIF
2016.05.27	Maintenance release
2016.05.03	Maintenance release
2015.11.02	• Correct size of HPS-to-FPGA region in "MPU Address Space" • Clarify power domains in "Functional Description of the SDRAM L3 Interconnect"
2015.05.04	• Added address maps and register definitions • Added information about the SDRAM scheduler • Added information about rate adapters • Added information about the observation network
2014.12.15	• Added register details for address remapping • Added information about quality of service • Added information about arbitration • Clarified block diagram of SDRAM L3 interconnect
2014.08.18	Initial release

System Interconnect on page 200

Document Version	Changes
2017.07.22	Added note about bridge transaction timeout to "HPS-to-FPGA Bridge Clocks and Resets" and "Lightweight HPS-to-FPGA Bridge Clocks and Resets".
2016.10.28	• Added note about AXI 4 KB boundary restriction • Clarified description of bridge master-slave connections
2016.05.27	Maintenance release
2015.11.02	Maintenance release
2015.05.04	Added address maps and register definitions
2014.12.15	Maintenance release
2014.08.18	Initial release.

HPS-FPGA Bridges on page 248

Document Version	Changes
2021.05.26	Added clarification for ACP usage requirements by adding the following chapters:

continued...

Document Version	Changes
	• *AxPROT Attributes* • *Configuring AxPROT[2:0] Sideband Signals for Coherent Accesses*
2020.08.18	Added information about maintaining cache coherency in the *Accelerator Coherency Port*
2019.06.14	Added details about arbitration behavior in the SCU when the ACP is not being used in the *Implementation Details* of the *Snoop Control Unit* section,
2016.10.28	• Added "Configuring AxCACHE[3:0] Sideband Signals" and "Configuring AxUser[4:0] Sideband Signals" subsections to the "AXI Master configuration for ACP Access" section
2016.05.27	Maintenance release
2016.05.03	Maintenance release
2015.11.02	• Reordered "L2 Cache" subsections • Renamed "ECC Support" L2 subsection to be "Single Event Upset Protection" • Added "L2 Cache Parity" subsection in "L2 Cache" section
2015.05.04	• Corrected allowed AxID values in "Accelerator Coherency Port" section • Added address maps for the Cortex-A9 MPU subsystem and the L2 cache controller
2014.12.15	• Added bus transaction scenarios in the "Accelerator Coherency Port" section • Added the "AxUSER and AxCACHE" subsection to the "Accelerator Coherency Port" section • Added the "Shared Requests on ACP" subsection to the "Accelerator Coherency Port" section • Added the "Configuration for ACP Use" subsection to the "Accelerator Coherency Port" section • Added parity error handling information to the "L1 Caches" section and the "Cache Controller Configuration" topic of the "L2 Cache" section.
2014.08.18	Initial Release

Cortex-A9 Microprocessor Unit Subsystem on page 282

Document Version	Changes
2017.07.29	Added reset requirement for BST
2016.10.28	Maintenance release
2016.05.03	Maintenance release
2015.11.02	Added a description on how the 2 TAP controllers are connected and supporting figures.
2014.05.04	Maintenance release.
2014.12.15	Maintenance release.
2014.08.18	Initial release.

CoreSight Debug and Trace on page 322

Table 4. Error Checking and Correction Controller Revision History

Date	Version	Changes
October 2016	2016.10.28	Maintenance Release
May 2016	2016.05.27	Maintenance Release
May 2016	2016.05.03	Maintenance Release

continued...

Date	Version	Changes
November 2015	2015.11.02	• Added "ECC Bits Required Based on Data Width" table to "Error Checking and Correction Algorithm" section • Added 136-bit Hamming Matrix figure to "Error Checking and Correction Algorithm" section
May 2015	2015.05.04	• Added 35-bit Hamming Matrix figure to "Error Checking and Correction Algorithm" section • Added "ECC Controller Address Map and Register Definitions" section
December 2014	2014.12.15	Added the "RAM and ECC Memory Organization Example" subsection to the "ECC Structure" section Added the following subsections in the "Indirect Memory Access" section: • "Watchdog Timer" • "Data Correction" • "Error Injection" • "Memory Testing" • "Error Checking and Correction Algorithm"
August 2014	2014.08.18	Initial Release

Error Checking and Correction Controller on page 345

Document Version	Changes
2014.08.18	Initial release

On-Chip Memory on page 364

Document Version	Changes
2016.05.27	Added a link to the *Supported Flash Devices for Arria 10 SoC* webpage.
2016.05.03	Added information about determining how many CE/RB signals are available based on the selected pins.
2015.11.02	• Updated the Interrupt and DMA Enabling section to recommend reading back a register to ensure clearing an interrupt status. • Removed the reference to a missing figure from the `cs_setup_cnt` description. • Documented the behavior of the `wp_n` bit for when it is or is not be available.
2015.05.04	Added information about clearing out the ECC before the feature is enabled
2014.12.15	Maintenance release
2014.08.18	Initial release

NAND Flash Controller on page 370

Document Version	Changes
2021.07.08	Changed the SD Card Clock Frequency in *Changing the Card Clock Frequency*.
2017.07.22	Corrected the MMC Support Matrix table in the "MMC Support Matrix" section.
2016.10.28	Removed SPI support in tables in the Features section.
2016.05.27	Added a link to the *Supported Flash Devices for Arria 10 SoC* webpage.
2016.05.03	Maintenance release

continued...

Document Version	Changes
2015.11.02	• Moved "Interface Signals" section below "SD/MMC Controller Block Diagram and System Integration" section and renamed to "SD/MMC Signal Description." Clarified signals in this section. • Removed the indication that the AV/CV HPS support 8-bit eMMC. • Added information that Card Detect is only supported on interfaces routed via the FPGA fabric.
2015.05.04	Added information about clearing out the ECC before the feature is enabled
2014.12.15	Maintenance release
2014.08.18	Initial release

SD/MMC Controller on page 405

Document Version	Changes
2020.08.18	Added clarification to the description of the QSPI register, `indaddrtrig`
2019.07.09	Added a new section, *Write Request*, with WREN and RDSR information
2019.06.14	Maintenance release
2016.10.28	Maintenance release
2016.05.27	• Changed the name of the internal QSPI reference clock from `qspi_clk` to `qspi_ref_clk`; and the external QSPI output clock, from `sclk_out` to `qspi_clk`. • Added a link to the *Supported Flash Devices for Arria 10 SoC* webpage. • Re-worded information about disabling the watermark feature in the "Indirect Read Operation" and "Indirect Write Operation" sections.
2016.05.03	Maintenance release
2015.11.02	• Renamed "Interface Pins" section to "Quad SPI Flash Controller Signal Description" and moved it below the "Quad SPI Flash Controller Block Diagram and System Integration" section • Corrected the link to the HPS Address Map. • Added the Intel® Arria® 10 register map. • Better defined `l4_main_clk` clock. • Added Clock Gating information.
2015.05.04	Added information about clearing out the ECC before the feature is enabled
2014.12.15	Maintenance release
2014.08.18	Initial release

Quad SPI Flash Controller on page 490

Document Version	Changes
2017.07.22	Added information about DMA requiring that caches need to be enabled
2016.10.28	Maintenance release
2016.05.27	Maintenance release
2016.05.03	Maintenance release
2015.11.02	• Updated link point to the HPS Address Map and Register Definitions • Added information about the instruction fetch cache properties
2015.05.04	• Added Synopsys* handshake rules.
2014.12.15	Maintenance release
2014.08.18	Initial release

DMA Controller on page 512

Document Version	Changes
2022.08.22	Removed RGMII because it does not support FPGA IO
2021.04.09	Added `emac_clk_tx_i` handling requirement for exported HPS EMAC GMII interface in the *EMAC FPGA Interface Initialization* section.
2020.08.18	Updated *EMAC HPS Interface Initialization* to clarify how to verify RX PHY clocks after bringing the Ethernet PHY out of reset.
2019.06.14	• Clarified the PCF bit description for encoding value 0x2 in the `MAC_Frame_Filter` register. • Clarified "Busy Bit" (`gb` bit of `GMII_Address` register) in `Flow_Control` register description. • Clarified that `ttc` bit resides in the `Operation_Mode` Register (Register 6). • Clarified that the `pcsancis` and the `pcslchgis` bits of the`Interrupt_Status` register can be ignored because they apply to TBI, RTBI, or SGMII interface only.
2016.10.28	• Bit 16 updated Transmit Descriptor table • Updated "Clock Structure" • Added "Clock Structure"
2016.05.27	Removed references to the Application Interface (also known as the switch interface). This feature is not supported in this device.
2016.05.03	Maintenance release.
2015.11.02	• Added emac_phy_mac_speed_o signals to "FPGA EMAC I/O Signals" section • Added the following subsections in the "Layer 3 and Layer 4 Filters" section: — Layer 3 and Layer 4 Filters Register Set — Layer 3 Filtering — Layer 4 Filtering • Added internal clock diagram to "Clock Structure" section
2015.05.04	• Corrected IEEE 1588 timestamp resolution in the "EMAC Block Diagram and System Integration" section and the "IEEE 1588-2002 Timestamps" section • Added clarification for `phy_txclk_o` and `phy_clk_rx_i` in the "HPS EMAC I/O Signals" • Added subsections "Ordinary Clock," "Boundary Clock," "End-to-End Transparent Clock" and "Peer-to-Peer Transparent Clock" in the "Clock Type" section • Added clarification in the "EMAC Module Clock Inputs and Outputs" table for the description of `phy_clk_rx_i` in the "Clock Structure" section • Added "EMAC ECC RAM Reset" subsection in "Taking the Ethernet Out of Reset" section • Added address map and register definitions
2014.12.06	• Added *Application Interface* sub-section to *Features* Section. • Updated *EMAC Block Diagram and System Integration* section with new diagram and information. • Added *Signal Descriptions* section. • Added *EMAC Internal Interfaces* section. • Added *TX FIFO* and *RX FIFO* subsection to the *Transmit and Receive Data FIFO Buffers* section. • Updated *Descriptor Overview* section to clarify support for only enhanced (alternate) descriptors. • Added *Destination and Source Address Filtering Summary* in *Frame Filtering* Section. • Added *Clock Structure* sub-section to *Clocks and Resets* section • Added *Application Interface to FPGA Fabric* section in *Functional Description of EMAC* • Added *System Level EMAC Configuration Registers* section in *Ethernet Programming Model* • Added *EMAC Interface Initialization for FPGA GMII/MII Mode* section in Ethernet Programming Model • Added *EMAC Interface Initialization for RGMII/RMII Mode* section in Ethernet Programming Model • Corrected *DMA Initialization* and *EMAC Initialization and Configuration* titles to appear on correct initialization information • Removed duplicate programming information for DMA • Added *Taking the Ethernet MAC Out of Reset* section.
2014.08.18	Initial release.

Ethernet Media Access Controller on page 520

Document Version	Changes
2018.01.26	Added steps for enabling ECC.
2016.10.28	Maintenance release.
2016.05.03	Maintenance release.
2015.11.02	• Renamed "ULPI PHY Interface" section to "USB 2.0 ULPI PHY Signal Description" and moved it after the "USB OTG Controller Block Diagram and System Integration" section. • Removed references to LPM mode in document, including "LPM Function" section • Added "DMA" section • Added "Clock Gating" section
2015.05.04	Maintenance release.
2014.12.15	• Maintenance release. • Added *Taking the USB OTG Out of Reset* section.
2014.08.18	Initial release.

USB 2.0 OTG Controller on page 591

Document Version	Changes
2015.11.02	• Renamed "Interface Pins" section to "Interface to HPS I/O" and moved it under the "SPI Controller Signal Description" section • Moved "FPGA Routing" section under "SPI Controller Signal Description" Section • Added Clock Gating information • Added Multi-Master mode to "Features of the SPI Controller" section • Updated "RXD Sample Delay" section • Updated "Glue Logic for Master Port ss_in_n" section
2015.05.04	Maintenance release.
2014.12.15	• Maintenance release. • Added *Taking the SPI Out of Reset* section.
2014.08.18	Initial release.

SPI Controller on page 608

Document Version	Changes
2015.11.02	• Renamed *Interface Pins* section to *I^2C Controller Signal Description* and moved section below *I^2C Controller Block Diagram and System Integration*
2015.05.04	• Added *Impact of SCL Rise Time and Fall Time On Generated SCL* figure to Clock Synchronization section • Updated Minimum High and Low Counts section
2014.12.15	• Maintenance release. • Added *Taking the I^2C Out of Reset* section.
2014.08.18	Initial release.

I2C Controller on page 643

Document Version	Changes
2015.11.02	Renamed *Interface Pins* section to *HPS I/O Pins* and moved this section and *FPGA Routing* under *UART Controller Signal Description*
2015.05.04	Maintenance release.
2014.12.15	• Maintenance release. • Added *Taking the UART Out of Reset* section.
2014.08.18	Initial release.

UART Controller on page 670

Document Version	Changes
2014.12.15	• Maintenance release. • Added *Taking the GPIO Out of Reset* section.
2014.08.18	Initial release.

General-Purpose I/O Interface on page 684

Version	Changes
2014.08.18	Initial release.

Timer on page 688

Document Version	Changes
2015.11.02	Added note to "Watchdog Timer Counter" section.
2015.05.04	Maintenance release.
2014.12.15	• Maintenance release. • Added *Taking the Watchdog Timer Out of Reset* section.
2014.08.18	Initial release

Watchdog Timer on page 693

Table 5. Hard Processor System I/O Pin Multiplexing Revision History

Document Version	Changes
2019.06.14	Updated the PU_DRV_STRG and PD_DRV_STRG fields in the `pinmux_dedicated_io_1` through `pinmux_dedicated_io_17` registers in the `io48_pin_mux_dedicated_io_grp`.
2018.09.24	Updated the following section: • *Features of the HPS I/O Block*
2016.10.28	Maintenance release
2016.05.27	Maintenance release.
2016.05.03	Maintenance release
2015.11.02	Maintenance release
2015.05.04	Added address maps and register definitions
2014.08.18	Initial release

Document Version	Changes
2016.05.03	Removed FPGA-to-HPS SDRAM interface
2015.11.02	Maintenance release
2015.05.04	Maintenance release
2014.12.15	Maintenance release
2014.08.18	Initial release

Introduction to the HPS Component on page 710

Document Version	Changes
2016.05.27	Removed *FPGA EMAC Switch Interface* section. The application interface (also called the switch interface) is not supported.
2016.05.03	Maintenance release.
2015.11.02	Added content regarding peripheral pin placement in HPS shared and dedicated I/O to the "Configuring Peripherals" section.
2015.05.04	• Updated "Reset Interfaces" section. • Updated "General Interfaces" section.
2014.12.15	Initial release.

Instantiating the HPS Component on page 716

Document Version	Changes
2016.05.27	Removed *FPGA EMAC Switch Interface* section. The application interface (also called the switch interface) is not supported.
2016.05.03	Maintenance release.
2015.11.02	• Added "Platform Designer (Standard) Port Interface Mapping" section and subsections to "Peripheral Signal Interfaces"
2015.05.04	Maintenance release.
2014.12.15	Initial release.

HPS Component Interfaces on page 731

Table 6. Simulating the HPS Component Revision History

Document Version	Changes
2016.05.27	Removed *FPGA EMAC Switch Interface* section. The application interface (also called the switch interface) is not supported.
2016.05.03	Removed references to FPGA to HPS SDRAM simulation.
2015.11.02	Added information about the signals on the "Advanced FPGA Placement" tab.
2015.05.04	Maintenance release.
2014.12.15	Maintenance release.
2014.08.15	Initial release.

Simulating the HPS Component on page 747

Document Version	Changes
2018.11.02	Added note regarding SD card image partitioning in the *SD/MMC Flash Devices* section.
2017.12.15	Removed `CM_PLL_CLK*` signals from "Boot Source MUX Selects" table in *Boot Source I/O Pins* section
2017.07.22	• Removed references to `EOSC1` and replaced them with the correct external oscillator pin name, `HPS_CLK1`. • Clarified that the `osc1_clk` signal is sourced from the `HPS_CLK1` pin.
2017.07.10	• Modified note in the *Booting and Configuration Options* and *Boot Select* sections to remove statement requiring HPS to hold in reset until after the FPGA has been fully programmed. • Added a note regarding the necessity to prevent the HPS from being held in cold or warm reset indefinitely in the *Reset*, *FPGA Configuration* and *Full FPGA Reconfiguration* sections.
2017.05.31	• Added note regarding SmartVID and early I/O release to the sections *FPGA Configuration* and *Early I/O Release FPGA Configuration Flow Through HPS*. • Added *I/O State* section.
2016.10.28	Modified the "Remapping the On-Chip RAM" diagram in the "Typical Boot Flow (Non-secure)" section
2016.05.27	• Added *Handling an FPGA Configuration Failure after Early I/O Release* section. • Updated "Second-stage Boot Loader Image Layout" figure in the *Loading the Second-Stage Boot Loader Image* section. • Changed the name of the internal QSPI reference clock from `qspi_clk` to `qspi_ref_clk`; and the external QSPI output clock, from `sclk_out` to `qspi_clk`.
2016.05.03	• Updated SD/MMC device clock values in the *CSEL Settings for SD/MMC Controller* section. • Updated configuration sequence steps in the *Arria 10 SoC FPGA Configuration Sequence Through FPGA Manager* section. • Updated the reconfiguration sequence steps in the *Arria 10 SoC FPGA Partial Reconfiguration Sequence Through FPGA Manager* section. • Added bus mode to the "SD/MMC Controller Default Settings" table in the *Default Settings of the SD/MMC Controller* section.
2015.12.11	Updated the Full FPGA Reconfiguration section with the supported reconfiguration options.
2015.11.02	Added Intel Quartus® Prime setting information to "Early I/O Release FPGA Configuration Through HPS" section
2015.06.12	• Updated "Typical Second-Stage Loader Flow (Non-Secure)" figure in "Typical Boot Flow (Non-Secure)" section • Added content to "FPGA Overview" • Described I/O types and added details to booting option figures in "Booting and Configuration Options" • Added table to "Boot Fuses" section • Updated the CSEL setting tables in the "CSEL Settings for NAND Controller" section • Updated the "Quad SPI Controller Default Settings" table in the "Quad SPI Controller Default Settings" section • Updated the CSEL setting tables in the "Quad SPI Controller CSEL settings" section • Updated "Low-Level Boot Flow" figure in the "Typical Boot Flow (Non-Secure)" section • Added content regarding configuration flows in "FPGA Configuration" section • Added "Full FPGA Configuration Flow Through HPS" section • Added "Early I/O Release FPGA Configuration Flow Through HPS" section • Added "FPGA Reconfiguration" section with "Full FPGA Reconfiguration Section"

continued...

Document Version	Changes
2015.05.04	• Added more detail in the "FPGA Configures First" table of the "Booting and Configuration Options" section. • Added alternate "Boot Source Mux Selects" table in "Boot Source I/O Pins" section and "I/O Configuration" section. • Added more detail regarding booting from FPGA in the "Boot Select" section. • Added note in the "Boot ROM Flow" section about caches being enabled by the Boot ROM.
2014.12.15	• Removed "Boot Stages" section. • The following sections were added: — "Boot ROM Flow" — "Typical Second-Stage Boot Flow" — "HPS State on Entry to the Preloader" — "Loading the Second-Stage Boot Image" — "FPGA Configuration" section with "Intel Arria 10 SoC FPGA Full Configuration" and "Intel Arria 10 SoC FPGA Partial Reconfiguration" • In the "Boot Definitions" section, "Boot Source I/O Pins", "Boot Fuses", and "Flash Memory Device" subsections were added.
2014.08.18	Initial release

Booting and Configuration on page 771

2. Introduction to the Hard Processor System

The Intel Arria 10 system-on-a-chip (SoC) is composed of two distinct portions- a dual-core Arm* Cortex-A9 hard processor system (HPS) and an FPGA. The HPS architecture integrates a wide set of peripherals that reduce board size and increase performance within a system. A dedicated security manager within the HPS supports secure boot with the ability to authenticate, decrypt and provide tamper event response.

Figure 1 on page 29 shows a high-level block diagram of the Intel Arria 10 SoC device. Blocks connected to device pins have symbols (square with an X) adjacent to them in the figure.

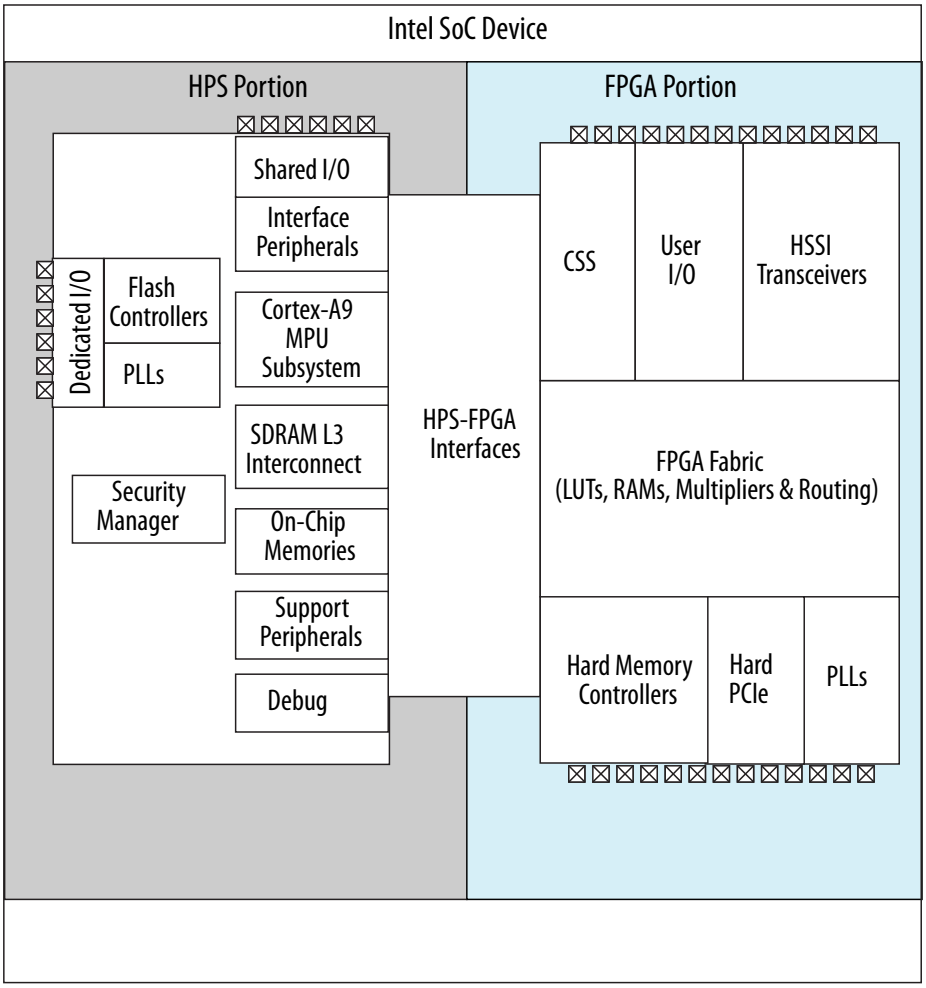
The SoC features the following types of I/O pins:

- Dedicated I/O - I/O that is dedicated to an external non-volatile storage device (for boot ROM), HPS clock and resets.
- Shared I/O - I/O that can be assigned to peripherals in the HPS or FPGA logic.
- FPGA I/O - I/O that is dedicated to the FPGA fabric.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 1. Intel SoC Device Block Diagram



The HPS consists of the following types of modules:

- Microprocessor unit (MPU) subsystem with a dual Arm Cortex-A9 MPCore processors
- Flash memory controllers
- SDRAM L3 interconnect
- System interconnect
- On-chip memories
- Support peripherals
- Interface peripherals
- Debug components
- Phase-locked loops (PLLs)

The HPS incorporates third-party intellectual property (IP) from several vendors.

The dual-processor HPS supports symmetric (SMP) and asymmetric (AMP) multiprocessing.

The FPGA portion of the device contains:

- FPGA fabric
- Configuration sub-system (CSS)
- PLLs
- High-speed serial interface (HSSI) transceivers, depending on the device variant
- Hard PCI Express* (PCI-e) controllers
- Hard memory controllers

The HPS and FPGA communicate with each other through bus interfaces that bridge the two distinct portions. On a power-on reset, the HPS can boot from multiple sources, including the FPGA fabric and external flash. The FPGA can be configured through the HPS or an externally supported device.

The HPS and FPGA portions of the device each have their own pins. The HPS has dedicated I/O pins and can also share some FPGA I/O pins. Pin assignments are configured when the HPS component is instantiated in Platform Designer (Standard). At boot time, software executing on the HPS assigns the I/O pins to the available HPS modules and configures the I/O pins through I/O control registers. For more information, refer to the "Hard Processor System I/O Pin Multiplexing" chapter. The FPGA I/O pins are configured by an FPGA configuration image through the HPS or any external source supported by the device.

The FPGA fabric and the HPS must be powered at the same time. Once powered, the FPGA fabric and the HPS can be configured independently thus providing you with more design flexibility:

- You can boot the HPS independently. After the HPS is running, the HPS can fully or partially reconfigure the FPGA fabric at any time under software control. The HPS can also configure other FPGAs on the board through the FPGA configuration controller.
- You can configure the FPGA fabric first, and then boot the HPS from the memory accessible to the FPGA fabric.

Related Information

- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter
- Intel Intel Arria 10 Device Datasheet

2.1. Features of the HPS

The main modules of the HPS are:

- MPU subsystem featuring a dual-core Arm Cortex-A9 MPCore processor
- System interconnect that includes three memory-mapped interfaces between the HPS and FPGA:
  - HPS-to-FPGA: 32-, 64-, or 128-bit wide AXI-4
  - Lightweight HPS-to-FPGA: 32-bit wide AXI-4
  - FPGA-to-HPS: 32-, 64-, or 128-bit wide ACE
- General-purpose direct memory access (DMA) controller
- Security manager
- Supports peripheral memories with single-error correction and double-error detection (SECDED)
- Three Ethernet media access controllers (EMACs)
- Two USB 2.0 on-the-go (OTG) controllers
- NAND flash controller
- Quad serial peripheral interface (QSPI) flash controller
- Secure digital/multimedia card (SD/MMC) controller
- Two serial peripheral interface (SPI) master controllers
- Two SPI slave controllers
- Five inter-integrated circuit (I^2C) controllers(1)
- 256 KB on-chip RAM
- 128 KB on-chip boot ROM
- Two UARTs
- Four system timers
- Two watchdog timers
- Three general-purpose I/O (GPIO) interfaces
- Arm CoreSight debug components:
  - Debug access port (DAP)
  - Trace port interface unit (TPIU)
  - System trace macrocell (STM)
  - Program trace macrocell (PTM)
  - Embedded trace router (ETR)
  - Embedded cross trigger (ECT)
- System manager

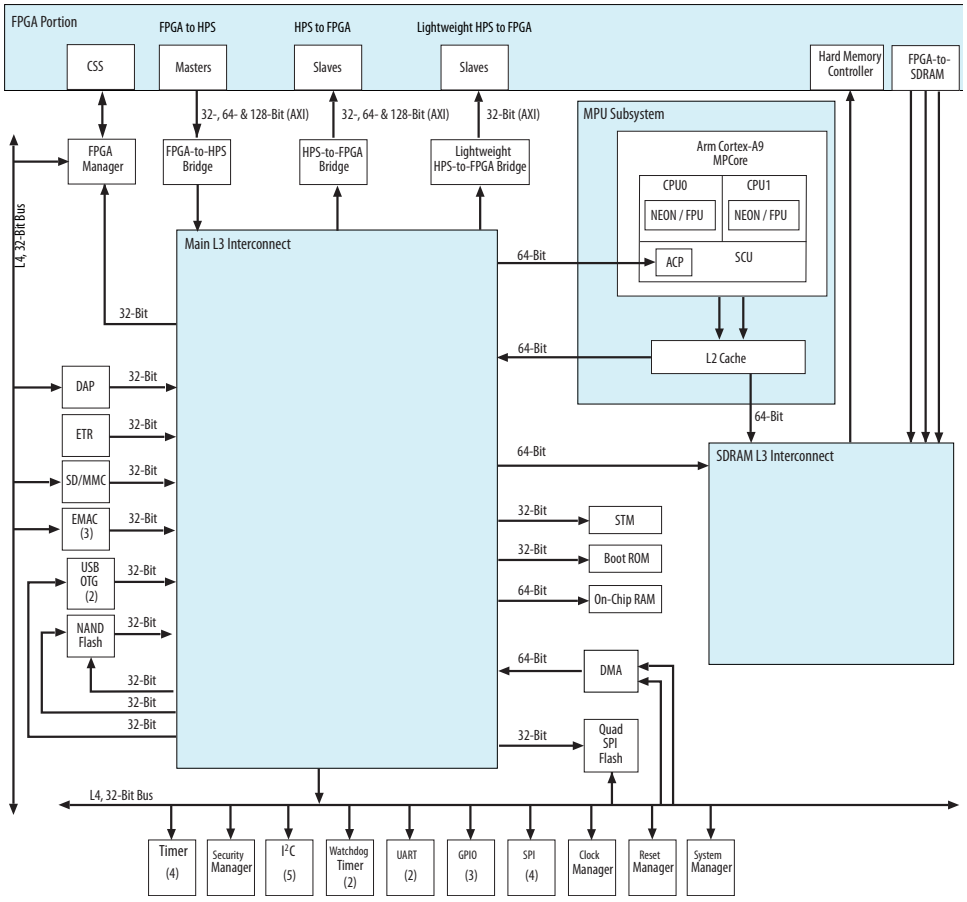
(1) Three of the five I^2Cs, can optionally be configured to provide PHY management support for each EMAC controller.

- Clock manager
- Reset manager
- FPGA manager

2.2. HPS Block Diagram and System Integration

2.2.1. HPS Block Diagram

Figure 2. HPS Block Diagram



2.2.2. Cortex-A9 MPCore

The MPU subsystem is a stand-alone, full-featured Arm Cortex-A9 MPCore dual-core 32-bit application processor. It provides the following functionality:

- Arm Cortex-A9 MPCore
  - Two Arm Cortex-A9 processors
  - NEON* single instruction, multiple data (SIMD) coprocessor and vector floating-point v3 (VFPv3) per processor
  - Snoop control unit (SCU) to ensure coherency between processors
  - Accelerator coherency port (ACP) that accepts coherency memory access requests
  - Interrupt controller
  - One general-purpose timer and one watchdog timer per processor
  - Debug and trace features
  - 32 KB instruction and 32 KB data level 1 (L1) caches per processor
  - Memory management unit (MMU) per processor
- Arm L2-310 level 2 (L2) cache
  - Shared 512 KB L2 cache

As shown in the "HPS Block Diagram", the L2 cache has one 64-bit master port that is connected to the main L3 interconnect and one 64-bit master port connected to the SDRAM L3 interconnect. A programmable address filter in the L2 cache controls which portions of the 32-bit physical address space can be accessed by each master.

Related Information

HPS Block Diagram on page 32

2.2.3. HPS Interfaces

The Arria 10 device family provides multiple communication channels to the HPS.

2.2.3.1. HPS–FPGA Memory-Mapped Interfaces

The HPS–FPGA memory-mapped interfaces provide the major communication channels between the HPS and the FPGA fabric. The HPS–FPGA memory-mapped interfaces include:

- FPGA–to–HPS bridge—a high–performance bus with a configurable data width of 32, 64, or 128 bits, allowing the FPGA fabric to master transactions to the slaves in the HPS. This interface allows the FPGA fabric to have full visibility into the HPS address space. This interface also provides access to the coherent memory interface
- HPS–to–FPGA bridge—a high–performance interface with a configurable data width of 32, 64, or 128 bits, allowing the HPS to master transactions to slaves in the FPGA fabric
- Lightweight HPS–to–FPGA bridge—an interface with a 32–bit fixed data width, allowing the HPS to master transactions to slaves in the FPGA fabric

Related Information

HPS-FPGA Bridges on page 248

2.2.3.2. Other HPS Interfaces

- TPIU trace—sends trace data created in the HPS to the FPGA fabric
- FPGA System Trace Macrocell (STM)—an interface that allows the FPGA fabric to send hardware events to be stored in the HPS trace data
- FPGA cross–trigger—an interface that allows the CoreSight trigger system to send triggers to IP cores in the FPGA, and vise versa
- DMA peripheral interface—multiple peripheral–request channels
- FPGA manager interface—signals that communicate with the FPGA fabric for boot and configuration
- Interrupts—allow soft IP cores to supply interrupts directly to the MPU interrupt controller
- MPU standby and events—signals that notify the FPGA fabric that the MPU is in standby mode and signals that wake up Cortex-A9 processors from a wait for event (WFE) state
- HPS debug interface – an interface that allows the HPS debug control domain (debug APB*) to extend into FPGA

Another HPS–FPGA communications channel is FPGA clocks and resets.

2.2.4. System Interconnect

The system interconnect consists of the main L3 interconnect, SDRAM L3 interconnect, and level 4 (L4) buses. The system interconnect is based on the Arteris*FlexNoC* network-on-chip (NoC) interconnect module. The system interconnect incorporates configurable secure firewalls protecting each peripheral.

The L4 buses are each connected to a master in the main L3 interconnect. Each L4 bus is 32 bits wide and is connected to multiple slaves. Each L4 bus operates on a separate clock source.

The SDRAM L3 interconnect consists of the SDRAM adapter and the SDRAM scheduler. SDRAM is protected by firewalls in the SDRAM L3 interconnect.

Related Information

System Interconnect on page 200

2.2.4.1. Arria 10 HPS SDRAM L3 Interconnect

The SDRAM L3 interconnect is part of the system interconnect and connects the HPS to the hard memory controller that is located in the FPGA fabric. The SDRAM L3 interconnect is composed of the SDRAM adapter and the SDRAM scheduler, which are secured by firewalls. It supports Arm Advanced Microcontroller Bus Architecture (AMBA*) Advanced eXtensible Interface (AXI) quality of service (QoS) for the fabric interfaces

The hard memory controller implements the following high-level features:

- Support for double data rate 3 (DDR3) and DDR4 devices
- Software-configurable priority scheduling on individual SDRAM bursts
- Error correction code (ECC) support, including calculation, single-bit error correction and write-back, and error counters
- Fully-programmable timing parameter support for all JEDEC-specified timing parameters
- All ports support memory protection and mutual-exclusive accesses
- FPGA-to-SDRAM interface—a configurable interface to the SDRAM scheduler in the SDRAM L3 interconnect You can configure the following parameters:
  - Up to three bridges
  - Up to 288 bits across all three ports

2.2.4.1.1. SDRAM Adapter

The SDRAM adapter is responsible for bridging the hard memory controller in the FPGA fabric to the SDRAM scheduler. The adapter is also responsible for error correction code (ECC) generation and checking.

2.2.4.1.2. SDRAM Scheduler

The SDRAM scheduler accepts read and write requests from the processor, HPS peripheral masters, and soft logic in FPGA through the FPGA-to-SDRAM interface. It is programmed with memory timings, allowing it to optimize memory access.

2.2.5. On-Chip Memory

2.2.5.1. On-Chip RAM

The on-chip RAM offers the following features:

- 256 KB size
- 64-bit slave interface
- High performance for all burst lengths
- ECC support provides detection of single–bit and double–bit errors and correction for single-bit errors
- Can clear the on-chip RAM on reset

Related Information

On-Chip Memory on page 364

2.2.5.2. Boot ROM

The boot ROM offers the following features:

- 32-bit interface
- Capable of data transfers for every clock cycle
- 128 KB size
- Contains the code required to support both secure and non-secure HPS boot from cold or warm reset
- Used exclusively for booting the HPS

Related Information

- On-Chip Memory on page 364
- Booting and Configuration on page 771

2.2.6. Flash Memory Controllers

2.2.6.1. NAND Flash Controller

The NAND flash controller is based on the Cadence® Design IP® NAND Flash Memory Controller and offers the following functionality and features:

- Supports up to four chip selects. One chip select is connected to an HPS I/O pin and the rest can be connected to the FPGA I/O pins.
- Integrated descriptor-based DMA controller
- 8-bit and 16-bit ONFI 1.0 NAND flash devices
- Programmable page sizes of 512 bytes, 2 KB, 4 KB, and 8 KB
- Supports 32, 64, 128, 256, 384, and 512 pages per block
- Programmable hardware ECC for single-level cell (SLC) and multi-level cell (MLC) devices
- 512-byte ECC sector size with 4-, 8-, or 16-bit correction
- 1 KB ECC sector size with 24-bit correction

Related Information

NAND Flash Controller on page 370

2.2.6.2. Quad SPI Flash Controller

The quad SPI flash controller is used for access to serial NOR flash devices. It supports standard SPI flash devices as well as high-performance dual and quad SPI flash devices. The Quad SPI flash controller is based on the Cadence Quad SPI Flash Controller and offers the following features:

- Supports SPIx1, SPIx2, or SPIx4 (Quad SPI) serial NOR flash devices
- Supports direct access and indirect access modes
- Supports single, dual, and quad I/O instructions
- Support up to four chip selects
- Programmable write-protected regions

- Programmable delays between transactions
- Programmable device sizes
- Support eXecute-In-Place (XIP) mode
- Programmable baud rate generator to generate device clocks

Related Information

Quad SPI Flash Controller on page 490

2.2.6.3. SD/MMC Controller

The Secure Digital (SD), Multimedia Card (MMC), (SD/MMC) and CE-ATA host controller is based on the Synopsys DesignWare* Mobile Storage Host controller and offers the following features:

- Integrated descriptor-based DMA
- Supports CE-ATA digital protocol commands
- Supports single card
- Single data rate (SDR) mode only
- Programmable card width: 1-, 4-, and 8-bit
- Programmable card types: SD, SDIO, or MMC
- Up to 64 KB programmable block size
- Supports the following standards and card types:
  - SD, including eSD—version 3.0(2)
  - SDIO, including embedded SDIO (eSDIO)—version 3.0(3)
  - CE-ATA—version 1.1
- Supports various types of multimedia cards, MMC version 4.41(4)
  - MMC: 1-bit data bus
  - Reduced-size MMC (RSMMC): 1-bit and 4-bit data bus
  - MMCMobile: 1-bit data bus
- Supports embedded MMC (eMMC) version 4.5(5)
  - 1-bit, 4-bit, and 8-bit data bus

Note: For an inclusive list of the programmable card types and versions supported, refer to the *SD/MMC Controller* chapter.

Related Information

SD/MMC Controller on page 405

(2) Does not support SDR50, SDR104, and DDR50 modes.

(3) Does not support SDR50, SDR104, and DDR50 modes.

(4) Does not support DDR mode.

(5) Does not support DDR and HS200 mode.

2.2.7. Support Peripherals

2.2.7.1. Clock Manager

The clock manager is responsible for providing software-programmable clock control to configure all clocks generated in the HPS. The clock manager offers the following features:

- Manages clocks for HPS
- Supports clock gating at the signal level
- Supports dynamic clock tuning

Related Information

Clock Manager on page 52

2.2.7.2. Reset Manager

The reset domains and sequences support several security features. The security manager brings the reset manager out of reset only when device security is confirmed. Afterwards, the reset manager brings the rest of the HPS system out of reset. The reset manager performs the following functions:

- Manages resets for HPS
- Controls the resets for cold, warm, debug, and TAP reset domains
- Controls the RAM clearing domain separately

Related Information

- Reset Manager on page 134
- SoC Security on page 168

2.2.7.3. Security Manager

The security manager module is integrated in the HPS and provides the overall management of security within the SoC, including:

- Recognition of secure fuse configuration, preventing non-secure device startup
- State control and check of the security features
- Secure boot options
- Secure debug options
- Anti-tamper support

Related Information

SoC Security on page 168

2.2.7.4. System Manager

The system manager contains logic to control system functions and logic to control other modules that need external control signals provided as part of system integration. The system manager offers the following features:

- Provides combined ECC status and interrupts from other HPS modules with ECC-protected RAM
- Low-level control of peripheral features not accessible through the control and status registers (CSRs)

Related Information

System Manager on page 159

2.2.7.5. System Timers

The HPS provides four 32-bit general-purpose timers connected to the level 4 (L4) peripheral bus. The four system timers are based on the Synopsys DesignWare Advanced Peripheral Bus (APB) Timers peripheral and offer the following features:

- 32-bit timer resolution
- Free-running timer mode
- Programmable time-out period up to approximately 86 seconds (assuming a 50 MHz input clock frequency)
- Interrupt generation

Related Information

Timer on page 688

2.2.7.6. System Watchdog Timers

The two system watchdog timers are based on the Synopsys DesignWare APB Watchdog Timer peripheral and offer the following features:

- 32-bit timer resolution
- Interrupt request
- Reset request
- Programmable time-out period up to approximately 86 seconds (assuming a 50 MHz input clock frequency)

  Note: Countdown can be paused when the MPU is in debug mode.

Related Information

Watchdog Timer on page 693

2.2.7.7. DMA Controller

The DMA controller provides high-bandwidth data transfers for modules without integrated DMA controllers. The DMA controller is based on the Arm Corelink* DMA Controller (DMA-330) and offers the following features:

- Micro-coded to support flexible transfer types
  - Memory-to-memory
  - Memory-to-peripheral
  - Peripheral-to-memory
  - Scatter-gather
- Supports up to eight channels
- Supports flow control with 32 peripheral handshake interfaces

Related Information

DMA Controller on page 512

2.2.7.8. FPGA Manager

The FPGA manager manages and monitors the FPGA portion of the SoC device. The FPGA manager can configure the FPGA fabric from the HPS, monitor the state of the FPGA, and drive or sample signals to or from the FPGA fabric.

Related Information

FPGA Manager on page 154

2.2.7.9. Error Checking and Correction Controller

ECC controllers provide single- and double-bit error memory protection for integrated on-chip RAM and peripheral RAMs within the HPS.

The following peripherals have integrated ECC-protected memories:

- USB OTG 0 - 1
- SD/MMC Controller
- EMAC 0 - 2
- DMA controller
- NAND flash controller
- QSPI flash controller

Features of the ECC controller:

- Single-bit error detection and correction
- Double-bit error detection
- Interrupts generated on single- and double-bit errors

Related Information

Error Checking and Correction Controller on page 345

2.2.8. Interface Peripherals

2.2.8.1. EMACs

The three EMACs are based on the Synopsys DesignWare 3504-0 Universal 10/100/1000 Ethernet MAC and offer the following features:

- Supports 10, 100, and 1000 Mbps standard
- PHY interfaces supported through the HPS I/O pins:
  - Reduced media independent interface (RMII) and Reduced gigabit media independent interface (RGMII) through the HPS I/O pins
- PHY interfaces supported using adapter logic to route signals to the FPGA I/O pins:
  - Media independent interface (MII), Gigabit media independent interface (GMII), RMII, and Serial gigabit media independent interface (SGMII) (with external conversion logic) through the FPGA pins

  Note: The SoC device does not support adapting the HPS EMAC signals to RGMII using FPGA I/O pins.
- Integrated DMA controller
- Supports IEEE 1588-2002 and IEEE 1588-2008 standards for precision networked clock synchronization
- IEEE 802.3-az, version D2.0 of Energy Efficient Ethernet
- Supports IEEE 802.1Q Virtual local area network (VLAN) tag detection for reception frames
- PHY Management control through Management data input/output (MDIO) interface or I^2C interface
- 4 KB TX FIFO and 16 KB RX FIFO RAM
- Supports a variety of address filtering modes

Related Information

Ethernet Media Access Controller on page 520

2.2.8.2. USB Controllers

The HPS provides two USB 2.0 Hi-Speed On-the-Go (OTG) controllers from Synopsys DesignWare. The USB controller signals cannot be routed to the FPGA like those of other peripherals; instead they are routed to the dedicated I/O.

Each of the USB controllers offers the following features:

- Complies with the following specifications:
  - USB OTG Revision 1.3
  - USB OTG Revision 2.0
  - Embedded Host Supplement to the USB Revision 2.0 Specification
- Supports software-configurable modes of operation between OTG 1.3 and OTG 2.0
- Supports all USB 2.0 speeds:
  - High speed (HS, 480-Mbps)
  - Full speed (FS, 12-Mbps)
  - Low speed (LS, 1.5-Mbps)

  Note: In host mode, all speeds are supported; however, in device mode, only high speed and full speed are supported.
- Local buffering with Error Correction Code (ECC) support

  Note: The USB 2.0 OTG controller does not support the following interface standards:
  - Enhanced Host Controller Interface (EHCI)
  - Open Host Controller Interface (OHCI)
  - Universal Host Controller Interface (UHCI)
- Supports USB 2.0 Transceiver Macrocell Interface Plus (UTMI+) Low Pin Interface (ULPI) PHYs (SDR mode only)
- Supports up to 16 bidirectional endpoints, including control endpoint 0

  Note: Only seven periodic device IN endpoints are supported.
- Supports up to 16 host channels

  Note: In host mode, when the number of device endpoints is greater than the number of host channels, software can reprogram the channels to support up to 127 devices, each having 32 endpoints (IN + OUT), for a maximum of 4,064 endpoints.
- Supports generic root hub
- Supports automatic ping capability

Related Information

- USB 2.0 OTG Controller on page 591
- Universal Serial Bus (USB) website

  Additional information is available in the On The Go and Embedded Host Supplement to the USB Revision 2.0 Specification, which you can download from the USB Implementers Forum website.

2.2.8.3. I²C Controllers

The five I²C controllers are based on Synopsys DesignWare APB I²C controller which offer the following features:

- Three controllers optionally can be used as a control interface for Ethernet PHY communication

  Note: When an I^2C controller is used for Ethernet, it takes the place of the MDIO pins.
- Support both 100 KBps and 400 KBps modes
- Support both 7-bit and 10-bit addressing modes
- Support master and slave operating mode
- Direct access for host processor
- DMA controller may be used for large transfers

Related Information

I2C Controller on page 643

2.2.8.4. UARTs

The HPS provides two UART controllers to provide asynchronous serial communications. The two UART modules are based on Synopsys DesignWare APB Universal Asynchronous Receiver/ Transmitter peripheral and offer the following features:

- 16550-compatible UART
- Support automatic flow control as specified in 16750 standard
- Programmable baud rate up to 6.25 MBaud (with 100MHz reference clock)
- Direct access for host processor
- DMA controller may be used for large transfers
- 128-byte transmit and receive FIFO buffers
- Separate thresholds for DMA request and handshake signals to maximize throughput

Related Information

UART Controller on page 670

2.2.8.5. SPI Master Controllers

The two SPI master controllers are based on Synopsys DesignWare Synchronous Serial Interface (SSI) controller and has a maximum bit rate of 60Mbps. The following features are offered:

- Programmable data frame size from 4 bits to 16 bits
- Supports full- and half-duplex modes
- Supports two chip selects connected to HPS I/O
- Supports four chip selects connected to the FPGA fabric
- Direct access for host processor
- DMA controller may be used for large transfers
- Programmable master serial bit rate

- Support for `rxd` sample delay
- Transmit and receive FIFO buffers are 256 words deep
- Choice of Motorola® SPI, Texas Instruments® Synchronous Serial Protocol or National Semiconductor® Microwire protocol

Related Information

SPI Controller on page 608

2.2.8.6. SPI Slave Controllers

The two SPI slave controllers are based on Synopsys DesignWare Synchronous Serial Interface (SSI) controller and has a maximum bit rate of 50 Mbps. The following features are offered:

- Programmable data frame size from 4 bits to 16 bits
- Supports full- and half-duplex modes
- Direct access for host processor
- DMA controller may be used for large transfers
- Transmit and receive FIFO buffers are 256 words deep

Related Information

SPI Controller on page 608

2.2.8.7. GPIO Interfaces

The HPS provides three GPIO interfaces that are based on Synopsys DesignWare APB General Purpose Programming I/O peripheral and offer the following features:

- Supports digital de-bounce
- Configurable interrupt mode
- Supports up to 17 dedicated 1.8 V and 3.0 V HPS I/O pins used for clock, reset, and external flash devices
- Supports up to 48 shared 3.0 V I/O pins that can be used by either the HPS or the FPGA. These pins are useful for Ethernet, USB, and other communication functions

Related Information

General-Purpose I/O Interface on page 684

2.2.9. CoreSight Debug and Trace

The CoreSight debug and trace system offers the following features:

- Real-time program flow instruction trace through a separate PTM for each processor
- Host debugger JTAG interface
- Connections for cross-trigger and STM-to-FPGA interfaces, which enable soft IP cores to generate of triggers and system trace messages
- Custom message injection through STM into trace stream for delivery to host debugger
- Capability to route trace data to any slave accessible to the ETR master, which is connected to the level 3 (L3) interconnect

Related Information

CoreSight Debug and Trace on page 322

2.2.10. Hard Processor System I/O Pin Multiplexing

The SoC has a total of 48 flexible I/O pins that are used for hard processor system (HPS) operation, external flash memories, and external peripheral communication. All of the 48 HPS I/O pins can also act as loaner I/O from the FPGA fabric. A pin multiplexing mechanism allows the SoC to use the flexible I/O pins in a wide range of configurations.

2.3. Endian Support

The HPS is natively a little–endian system. All HPS slaves are little endian.

The processor masters are software configurable to interpret data as little endian, big endian, or byte–invariant (BE8). All other masters, including the USB 2.0 interface, are little endian. Registers in the MPU and L2 cache are little endian regardless of the endian mode of the CPUs.

Note: Intel strongly recommends that you only use little endian.

The FPGA–to–HPS, HPS–to–FPGA, FPGA–to–SDRAM, and lightweight HPS–to–FPGA interfaces are little endian.

If a processor is set to BE8 mode, software must convert endianness for accesses to peripherals and DMA linked lists in memory. The processor provides instructions to swap byte lanes for various sizes of data.

The Arm Cortex-A9 MPU supports a single instruction to change the endianness of the processor and provides the REV and REV16 instructions to swap the endianness of bytes or half–words respectively. The MMU page tables are software configurable to be organized as little–endian or BE8.

The Arm DMA controller is software configurable to perform byte lane swapping during a transfer.

2.4. Introduction to the Hard Processor System Address Map

The address map specifies the addresses of slaves, such as memory and peripherals, as viewed by the MPU and other masters. The HPS has multiple address spaces, defined in the following section.

Related Information

System Interconnect on page 200

2.4.1. HPS Address Spaces

The following table shows the HPS address spaces and their sizes.

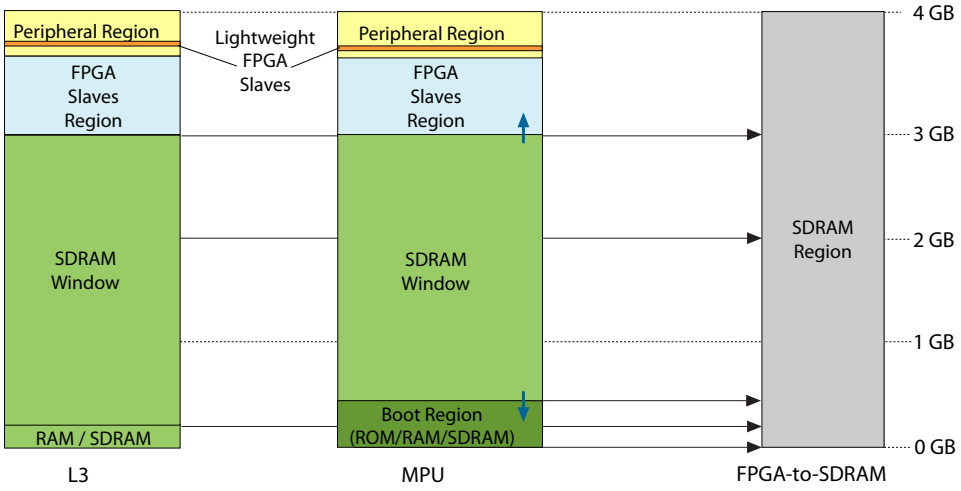
Table 7. HPS Address Spaces

Name	Description	Size
MPU	MPU subsystem	4 GB
L3	System interconnect	4 GB
SDRAM	SDRAM region	4 GB

Address spaces are divided into one or more nonoverlapping regions. For example, the MPU address space has the peripheral, FPGA slaves, SDRAM window, and boot regions.

The following figure shows the relationships between the HPS address spaces. The figure is not to scale.

Figure 3. HPS Address Space Relationships



The SDRAM window in the MPU address space can grow and shrink at the top and bottom (short, blue vertical arrows) at the expense of the FPGA slaves and boot regions. For specific details, refer to "MPU Address Space".

The following table shows the base address and size of each region that is common to the L3 and MPU address spaces.

Table 8. Common Address Space Regions

Region Name	Base Address	Size
FPGA slaves	0xC0000000	960 MB
Peripheral	0xFC000000	64 MB
Lightweight FPGA slaves	0xFF200000	2 MB

2.4.1.1. SDRAM Address Space

The SDRAM address space is up to 4 GB. The entire address space can be accessed through the FPGA-to-SDRAM interface from the FPGA fabric. The total amount of SDRAM addressable from the other address spaces can be configured at runtime.

There are cacheable and non-cacheable views into the SDRAM space. When a master of the L3 SDRAM interconnect performs a cacheable access to the SDRAM, the transaction is performed through the ACP port of the MPU subsystem. When a master of the SDRAM L3 interconnect performs a non-cacheable access to the SDRAM, the transaction is performed through the 64-bit L3 interconnect master of the SDRAM L3 interconnect.

Related Information

System Interconnect on page 200
For more information about how to configure SDRAM address space.

2.4.1.2. MPU Address Space

The MPU address space is 4 GB and applies to addresses generated inside the MPU.

The MPU address space contains the following regions:

- The SDRAM window region provides access to a large, configurable portion of the 4 GB SDRAM address space.

The address filtering start and end registers in the L2 cache controller define the SDRAM window boundaries. The boundaries are megabyte-aligned. Addresses within the boundaries route to the SDRAM master. Addresses outside the boundaries route to the system interconnect master.

Related Information

- HPS Address Spaces on page 46
  For more information, refer to the "HPS Address Spaces Relationships" table.
- System Interconnect on page 200
  For more information regarding SDRAM address mapping, refer to the System Interconnect chapter.
- Cortex-A9 Microprocessor Unit Subsystem on page 282

2.4.1.3. FPGA Slaves Address Space

The FPGA Slaves address space is located in the FPGA core and accessed from the HPS through the HPS2FPGA AXI Bridge. The FPGA Slaves address space in the FPGA core is of unlimited size (soft logic in the FPGA performs address decoding). The L3 and MPU

regions provide a window of nearly 1GB (actually 1GB less 64MB) into the FPGA Slaves address space. The base address of the FPGA Slaves Window is mapped to address 0x0 in the FPGA Slaves address space.

The HPS is able to boot from offset 0x0 into the FPGA Slaves address space (BSELselection).

2.4.1.4. LWFPGA Slaves Address Space

The Lightweight (LW) FPGA Slaves address space is located in the FPGA core and accessed from the HPS through the LWHPS2FPGA Bridge. The LWFPGA Slaves address space in the FPGA core is of unlimited size (soft logic in the FPGA performs address decoding). A portion of the Peripheral region provides a window of 2MB into the LWFPGA Slaves address space. The base address of the LWFPGA Slaves window is mapped to address 0x0 in the LWFPGA Slaves address space.

The HPS is not able to boot from the LWFPGA Slaves address space.

2.4.1.5. L3 Address Space

The L3 address space is 4 GB and applies to all L3 masters except the MPU. The L3 address space has more configuration options than the other address spaces.

Related Information

System Interconnect on page 200
For more information about configuring the L3 address space, refer to the System Interconnect chapter.

2.4.2. HPS Peripheral Region Address Map

Each peripheral slave interface has a dedicated address range in the peripheral region. The table below lists the base address and address range size for each slave.

Table 9. Peripheral Region Address Map

Slave Identifier	Description	Base Address	Size
STM	STM module	0xFC000000	48 MB
DAP	DAP module	0xFF000000	2 MB
LWFPGASLAVES	FPGA slaves accessed through lightweight HPS2FPGA bridge module	0xFF200000	2 MB
EMAC0	EMAC0 module	0xFF800000	8 KB
EMAC1	EMAC1 module	0xFF802000	8 KB
EMAC2	EMAC2 module	0xFF804000	8 KB
SDMMC	SD/MMC module	0xFF808000	4 KB
QSPIREGS	QSPI flash controller module registers	0xFF809000	4 KB
EMAC0RXECC	Receive ECC, Ethernet MAC0	0xFF8C0800	1 KB
EMAC0TXECC	Transmit ECC, Ethernet MAC0	0xFF8C0C00	1 KB

continued...

Slave Identifier	Description	Base Address	Size
EMAC1RXECC	Receive ECC, Ethernet MAC1	0xFF8C1000	1 KB
EMAC1TXECC	Transmit ECC, Ethernet MAC1	0xFF8C1400	1 KB
EMAC2RXECC	Receive ECC, Ethernet MAC2	0xFF8C1800	1 KB
EMAC2TXECC	Transmit ECC, Ethernet MAC2	0xFF8C1C00	1 KB
NANDECC	NAND ECC	0xFF8C2000	1 KB
NANDREADECC	NAND read ECC	0xFF8C2400	1 KB
NANDWRITEECC	NAND write ECC	0xFF8C2800	1 KB
SDMMCECC	SD/MMC ECC	0xFF8C2C00	1 KB
OCRAMECC	On-chip RAM ECC	0xFF8C3000	1 KB
DMAECC	DMA ECC	0xFF8C8000	1 KB
QSPIECC	QSPI ECC	0xFF8C8400	1 KB
USB0ECC	USB 2.0 OTG 0 ECC	0xFF8C8800	1 KB
USB1ECC	USB 2.0 OTG 1 ECC	0xFF8C8C00	1 KB
QSPIDATA	QSPI flash module data	0xFFA00000	1 MB
USB0	USB 2.0 OTG 0 controller module registers	0xFFB00000	256 KB
USB1	USB 2.0 OTG 1 controller module registers	0xFFB40000	256 KB
NANDREGS	NAND controller module registers	0xFFB80000	64 KB
NANDDATA	NAND controller module data	0xFFB90000	64 KB
UART0	UART0 module	0xFFC02000	256 B
UART1	UART1 module	0xFFC02100	256 B
I2C0	I^2C0 module	0xFFC02200	256 B
I2C1	I^2C1 module	0xFFC02300	256 B
I2C2	I^2C2 module (can be used with EMAC0)	0xFFC02400	256 B
I2C3	I^2C3 module (can be used with EMAC1)	0xFFC02500	256 B
I2C4	I^2C4 module (can be used with EMAC2)	0xFFC02600	256 B
SPTIMER0	SP Timer0 module	0xFFC02700	256 B
SPTIMER1	SP Timer1 module	0xFFC02800	256 B
GPIO0	GPIO0 module	0xFFC02900	256 B
GPIO1	GPIO1 module	0xFFC02A00	256 B
GPIO2	GPIO2 module	0xFFC02B00	256 B

continued...

Slave Identifier	Description	Base Address	Size
HMCAREGS	Hard memory controller adapter control registers	0xFFCFB000	4 KB
SECMGRDATA	Security manager module data	0xFFCFE000	1 KB
FPGAMGRDATA	FPGA manager module configuration data	0xFFCFE400	1 KB
OSC1TIMER0	OSC1 Timer0 module	0xFFD00000	256B
OSC1TIMER1	OSC1 Timer1 module	0xFFD00100	256B
L4WD0	Watchdog0 module	0xFFD00200	256B
L4WD1	Watchdog1 module	0xFFD00300	256B
SECMGRREGS	Security manager module control and status registers	0xFFD02000	4 KB
FPGAMGRREGS	FPGA manager module control and status registers	0xFFD03000	4 KB
CLKMGR	Clock manager module	0xFFD04000	4 KB
RSTMGR	Reset manager module	0xFFD05000	4 KB
SYSMGR	System manager module	0xFFD06000	4 KB
IOMGR	I/O manager module	0xFFD07000	4 KB
FWL4PRIV	L4 privilege firewall registers	0xFFD11000	256 B
MPURADAPTER	MPU rate adapter registers	0xFFD11100	3.84 KB
DDRPRB	DDR probe registers	0xFFD12000	1 KB
SCHREGS	DDR scheduler control registers	0xFFD12400	128 B
FWL4PER	L4 peripheral firewall registers	0xFFD13000	256 B
FWL4SYS	L4 system firewall registers	0xFFD13100	256 B
FWOCRAM	On-chip RAM firewall registers	0xFFD13200	256 B
FWFPGA2SDRAM	DDR firewall registers for FPGA-to-SDRAM	0xFFD13300	256 B
FWDDRL3	DDR L3 firewall registers	0xFFD13400	256 B
FWHPS2FPGA	HPS-to-FPGA firewall registers	0xFFD13500	256 B
L4PRB	L4 bus probe registers	0xFFD14000	4 KB
MPUPRB	MPU probe and test registers	0xFFD15000	4 KB
L4QOS	L4 bus QoS	0xFFD16000	4 KB (estimated)
EMACTSF	EMAC transaction status filter registers	0xFFD1 7080	44 B
DMANONSECURE	DMA non-secure module registers	0xFFDA0000	4 KB

continued...

Slave Identifier	Description	Base Address	Size
DMASECURE	DMA secure module registers	0xFFDA1000	4 KB
SPI0	SPI module 0 slave	0xFFDA2000	4 KB
SPI1	SPI module 1 slave	0xFFDA3000	4 KB
SPI2	SPI module 0 master	0xFFDA4000	4 KB
SPI3	SPI module 1 master	0xFFDA5000	4 KB
OCRAM	On-chip RAM module	0xFFE00000	1 MB (256 KB used)
ROM	Boot ROM Module	0xFFFC0000	128 KB
MPU	MPU Module Registers	0xFFFFC000	8 KB
MPUL2	MPU L2 Cache Controller Module Registers	0xFFFFF000	4 KB

3. Clock Manager

The hard processor system (HPS) clock generation is centralized in the clock manager. The clock manager is responsible for providing software-programmable clock control to configure all clocks generated in the HPS. Clocks are organized in clock groups. A clock group is a set of clock signals that originate from the same clock source. A phase-locked loop (PLL) clock group is a clock group where the clock source is a common PLL voltage-controlled oscillator (VCO).

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

3.1. Features of the Clock Manager

The *Clock Manager* offers the following features:

- Generates and manages clocks in the HPS
- Contains the following clock groups:
  - Main—contains clocks for the Cortex-A9 microprocessor unit (MPU) subsystem, level 3 (L3) interconnect, level 4 (L4) peripheral bus, and debug
  - Peripheral—contains clocks for PLL-driven peripherals
- Contains two identical 16-output PLL blocks:
  - PLL 0
  - PLL 1
- Generates clock gate controls for enabling and disabling most clocks
- Initializes and sequences clocks
- Allows software to program clock characteristics, such as the following items discussed later in this chapter:
  - Input clock source for the two PLLs
  - Multiplier range, divider range, and 16 post-scale counters for each PLL
  - VCO enable for each PLL
  - Bypass modes for each PLL
  - Gate of individual clocks in all PLL clock groups
  - Clear loss of lock status for each PLL
  - Boot mode for hardware-managed clocks
  - General-purpose I/O (GPIO) debounce clock divide

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Allows software to observe the status of all writable registers
- Supports interrupting the MPU subsystem on PLL-lock and loss-of-lock
- Supports clock gating at the signal level

The clock manager is ***not*** responsible for the following functional behaviors:

- Selection or management of the clocks for the FPGA-to-HPS and HPS-to-FPGA interfaces. The FPGA logic designer is responsible for selecting and managing these clocks.
- Software must not program the clock manager with illegal values. If it does, the behavior of the clock manager is undefined and could stop the operation of the HPS. The only guaranteed means for recovery from an illegal clock setting is a cold reset.
- When re-programming clock settings, there are no automatic glitch-free clock transitions. Software must follow a specific sequence to ensure glitch-free clock transitions. Refer to *Hardware-Managed and Software-Managed Clocks* section of this chapter.

Related Information

Hardware-Managed and Software-Managed Clocks on page 61

3.2. Clock Manager Block Diagram and System Integration

Figure 4. Clock Manager Block Diagram

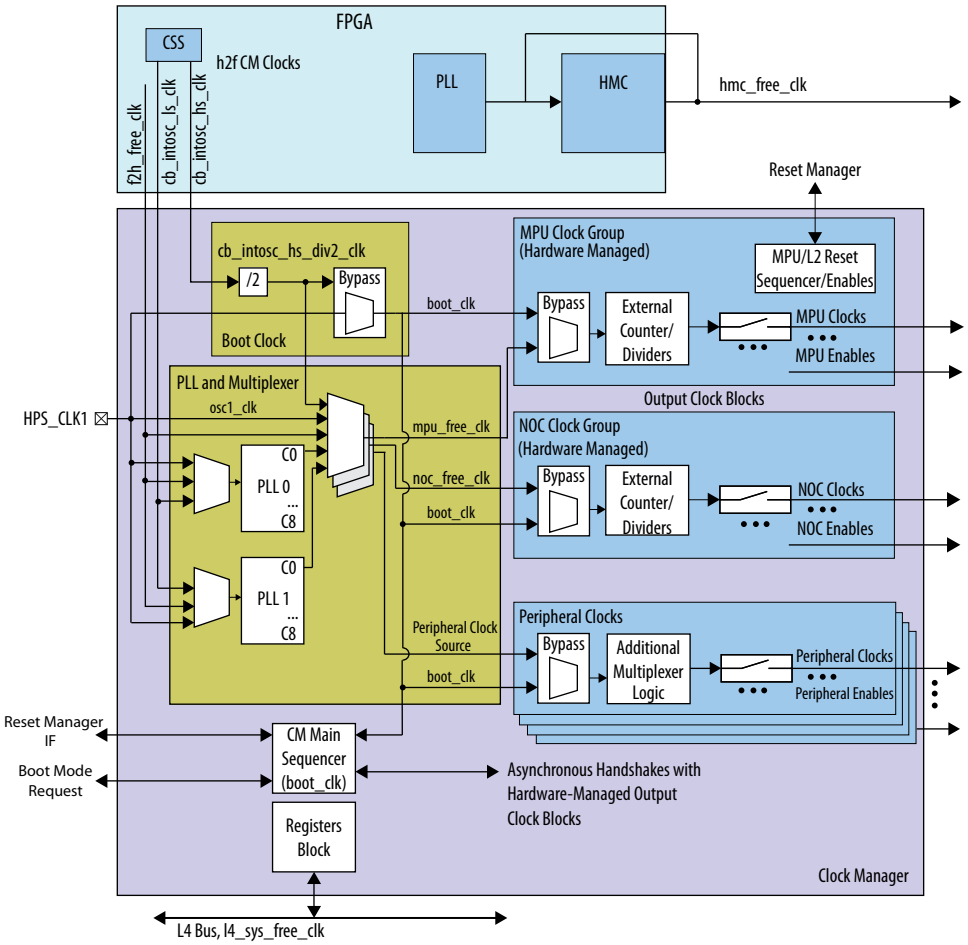


Table 10. Arria 10 Top Level Clocks

Clock Name	Source/Destination	Description
mpu_free_clk	Clock manager To MPU complex	Source clock from clock manager for both MPU clock groups.
mpu_clk	Internal to MPU complex	MPU main clock
mpu_l2ram_clk	Internal to MPU complex and HMC switch in NOC.	MPU L2 RAM clock and HMC switch in NOC. Fixed at ½ mpu_clk.
mpu_periph_clk	Internal to MPU complex	MPU peripherals clock for interrupts, timers, and watchdogs. Fixed at ¼ mpu_clk.
l3_main_free_clk	Clock manager to NOC/Peripherals	Interconnect L3 main switch clock. Always free running.
l4_sys_free_clk	Clock manager to NOC/Peripherals	Interconnect L4 system clock. Always free running.

continued...

Clock Name	Source/Destination	Description
l4_main_clk	Clock manager to NOC/Peripherals	L4 Interconnect clock for fast peripherals including DMA, SPIM, SPIS and TCM.
l4_mp_clk	Clock manager to NOC/Peripherals	Interconnect L4 peripheral clock.
l4_sp_clk	Clock manager to NOC/Peripherals	Interconnect L4 slow peripheral clock.
cs_at_clk	Clock manager to CoreSight/NOC	CoreSight Trace clock and debug time stamp clock.
cs_pdbg_clk	Clock manager to CoreSight	CoreSight bus clock
cs_trace_clk	Clock manager to CoreSight	CoreSight Trace I/O clock. This is independent and defaults to a low frequency (25 MHz) for lower speed debuggers. Not required to be sync. to other clocks (but keep in clock group as hardware managed)
cs_timer_clk	Clock manager to CoreSight/NOC	CoreSight timer clock. Same as cs_at_clk with different software enable to ensure MPU clock running. Not required to be sync. to other clocks (but still hardware managed).
fpga2soc_clk	FPGA fabric to NOC	FPGA to SoC interface clock from the FPGA.
soc2fpga_clk	FPGA fabric to NOC	SoC to FPGA interface clock from the FPGA.
lws2f_clk	FPGA fabric to NOC	LWHPS to FPGA interface clock from FPGA.
hmc_free_clk	From HMC to HMC switch in NOC	HMC interface clock from HMC (Hard Memory Controller) in FPGA.
f2h_sdram0_clk f2s_sdram1_clk f2h_sdram2_clk	FPGA fabric to HMC switch in NOC	FPGA fabric SDRAM write interfaces clocks.
f2h_pclkdbg	FPGA fabric to CoreSight bridge	CoreSight FPGA fabric APB debug port clock
usb[0,1]_ulpi_clk	I/O to USB	ULPI clock for PHY.

3.2.1. L4 Peripheral Clocks

The L4 peripheral clocks, denoted by l4_mp_clk, range up to 200 MHz.

Table 11. Clock List

Peripheral	Clock Name	Description
USB OTG 0/1(6)	hclk	AHB* clock
	pmu_hclk	PMU AHB clock. pmu_hclk is the scan clock for the PMU's AHB domain. *Note:* Select it as a test clock.
	utmi_clk	Always used as the PHY domain clock during DFT Scan mode.

continued...

Peripheral	Clock Name	Description
		Note: Select `utmi_clk` as a test clock even when the core is configured for a non-UTMI PHY.
Quad SPI Flash Controller[6]	`pclk`	APB clock
	`hclk`	AHB clock
NAND Flash Controller (Locally gated `nand_mp_clk`.)[6]	`ACLK`	AHB Data port clock
	`mACLK`	AXI Master port clock
	`regACLK`	AHB Register port clock
	`ecc_clk`	ECC circuitry clock
	`clk_x`	Bus Interface Clock
EMAC 0/1/2	`aclk`	Application clock for DMA AXI bus and CSR APB bus.
SD/MMC Controller	`sdmmc_clk`	All registers reside in the BIU clock domain.

For more information about the specific peripheral clocks, refer to their respective chapters.

Related Information

- SD/MMC Controller on page 405
- NAND Flash Controller on page 370
- Quad SPI Flash Controller on page 490
- Ethernet Media Access Controller on page 520
- USB 2.0 OTG Controller on page 591

3.2.2. Boot Clock

The Boot Clock (`boot_clk`) is used as the default clock for both cold or warm reset (Boot Mode), the Hardware Sequencer local clock and the external bypass clock reference.

The `boot_clk` is generated from the secure `cb_intosc_hs_div2_clk` or the unsecure external oscillator. `boot_clk` is only updated coming out of cold reset or a warm reset (boot mode request) and is not sampled at any other time.

The source of every output clock block comes from the following:

- Bypass source: `boot_clock` configured through fuses and security manager: registers at cold or warm reset
- Non-External Bypass: Each Clock may come from 1 of 5 sources:

(6) Clock manager provides CSR bits for software enables to some peripherals. These enables are defaulted to enable. In boot mode, these enables are automatically active to ensure all clocks are active if RAM is cleared for security.

Table 12. Non-External Bypass Sources of clocks

Source	Description
OSC1	Pin for external oscillator
f2h_free_clk	FPGA fabric PLL clock reference
cb_intosc_hs_div2_clk	FPGA CSS (Control Block) high speed internal oscillator divided by 2 (200 MHz maximum)
PLL0 Counter Output	Main PLL counter outputs 0 to 8
PLL1 Counter Output	Peripheral PLL counter outputs 0 to 8

Clock Output Blocks have the following features:

- Each clock output block contains an external bypass multiplexer
- Some clock output blocks contain additional dividers
- Clock gates controlled by either hardware or software are used to disable the clocks
- The MPU and NOC (includes debug clocks) blocks contain enable outputs to define clock frequency ratios to the MPU, NOC and CoreSight logic

The CSR Register logic uses an independent clock, `l4_sys_free_clk`, to allow the clock to be changed by software.

3.2.2.1. Updating PLL Settings without a System Reset

Updating the PLL settings without a system reset does not reset the clock manager. All clocks transitions gracefully to their boot safe dividers. You can go in and out of boot mode and force the PLLs into bypass without doing a system reset. Software may reset the PLLs and reconfigure the VCO and bring it out of reset. By doing this you can safely update the PLL settings.

3.2.2.2. MPU Clock Scaling

You can dynamically slow down the MPU clock without touching the PLL by modifying the MPU external counter register (`MAINPLLGRP.MPUCLK.CNT`) to slow down the MPU frequency.

Related Information

Clock Manager Address Map and Register Definitions on page 67

3.3. Functional Description of the Clock Manager

3.3.1. Clock Manager Building Blocks

3.3.1.1. PLLs

The two PLLs in the clock manager generate the majority of clocks in the HPS. There is no phase control between the clocks generated by the two PLLs.

Each PLL has the following features:

- Phase detector and output lock signal generation
- Registers to set VCO frequency
  - Multiplier range is 1 to 4096
  - Divider range is 1 to 64
- 16 post-scale counters (C0-C8) with a range of 1 to 2048 to further subdivide the clock
- A PLL can be enabled to bypass all outputs to the input clock for glitch-free transitions

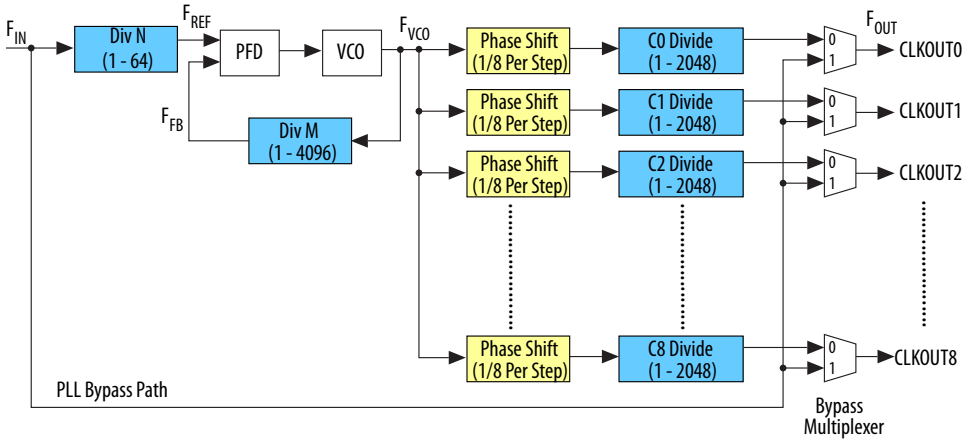
Related Information

- PLL Integration on page 59
- Hardware Sequenced Clock Groups on page 61
- Software Sequenced Clocks on page 63
- Intel Intel Arria 10 Device Datasheet

3.3.1.2. FREF, FVCO, and FOUT Equations

Figure 5. PLL Block Diagram

Values listed for M, N, and C are actually one greater than the values stored in the CSRs.



$$F_{REF} = F_{IN} / N$$
$$F_{VCO} = F_{REF} \times M = F_{IN} \times M/N$$
$$F_{OUT} = F_{VCO} / C_i = F_{REF} \times M/C_i = (F_{IN} \times M)/ (N \times C_i)$$

where:

- FVCO = VCO frequency
- FIN = input frequency
- FREF = reference frequency
- FOUT = output frequency

- M = numerator, part of the clock feedback path
- N = denominator, part of the input clock reference path
- Ci = post-scale counter, where *i* is 0-8

The Ci dividers are used to derive lower frequencies from the PLLs. The M and N dividers can be dynamically updated without losing the PLL lock if the VCO frequency changes less than 20%. If changes greater than 20% are needed, iteratively changing the frequency in increments of less than 20% allows a slow ramp of the VCO frequency without loss of clock.

To minimize jitter, use the following guidelines:

- The VCO should be as close to maximum as possible
- It is better to use the Ci dividers than the N divider. The N divider should be kept as small as possible.
- The M divider should be minimized, but should be greater than 32.

When making any changes that affect the VCO frequency, software must put the PLL into external bypass. After changing the VCO frequency, software must wait for the PLL lock, as indicated by the `intrs` register or enabled interrupt, before taking the PLL out of bypass.

As shown in the "Hardware Clock Groups" and "Peripheral Clocks" figures, every clock has five possible sources. When switching between clock sources:

- Put PLL into bypass mode
- Change the mux select
- Switch back out of bypass mode

  Note: The new clock source must be active and locked before exiting bypass mode.

As shown in the "PLL Integration in Clock Manager" figure, each PLL has multiple input sources. If the input source is changed, the PLL loses VCO lock and all output clocks are not reliable. Before switching the PLL input source, software must select the `boot_clk` bypass for all PLL0 and PLL1 clocks. After the PLL output clocks are bypassed, software can change the PLL source and reinitialize the PLL.

Unused clock outputs should be set to a safe frequency such as 50 MHz to reduce power consumption and improve system stability.

3.3.1.3. Clock Gating

Clock gating enables and disables clock signals. Refer to the Peripheral PLL Group Enable Register (`en`) for more information on what clocks can be gated.

Related Information

Clock Manager Address Map and Register Definitions on page 67

3.3.2. PLL Integration

The two PLLs contain exactly the same set of output clocks. PLL0 is intended to be used for the MPU and the NOC clocks. PLL1 is intended to be used as the 250 MHz Ethernet clock reference.

Figure 6. PLL Integration in Clock Manager

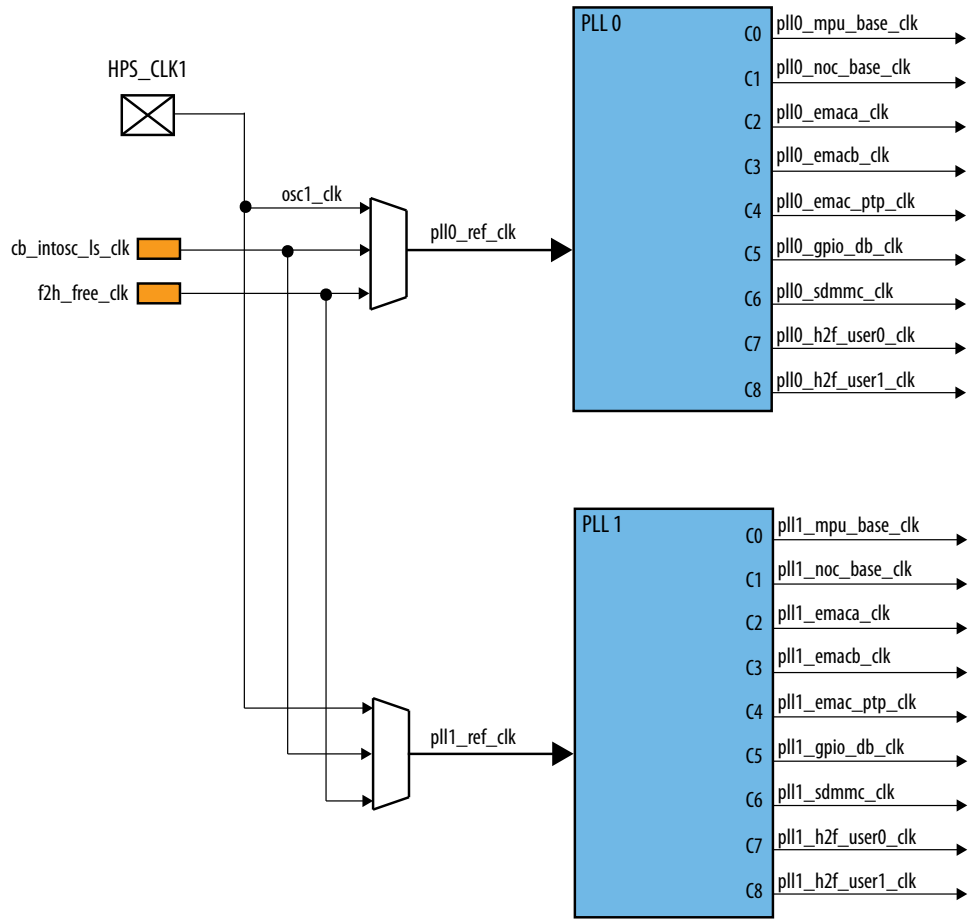


Table 13. PLL Output Clock Characteristics

Output Counter	Clock Name	Frequency	Boot Frequency
C0	mpu_base_clk	Up to varies	10 MHz to 200 MHz
C1	noc_base_clk	Up to C0/3	10 MHz to 200 MHz
C2	emaca_clk	50 to 250 MHz	10 MHz to 200 MHz
C3	emacb_clk	50 to 250 MHz	10 MHz to 200 MHz
C4	emac_ptp_clk	Up to 100 MHz	10 MHz to 200 MHz
C5	gpio_db_clk	Up to 200 MHz	10 MHz to 200 MHz

continued...

(7) Frequency depends on device, refer to device datasheet for more information.

Output Counter	Clock Name	Frequency	Boot Frequency
C6	sdmmc_clk	Up to 200 MHz	10 MHz to 200 MHz
C7	h2f_user0_clk	Up to 400 MHz	10 MHz to 200 MHz
C8	h2f_user1_clk	Up to 400 MHz	10 MHz to 200 MHz

Related Information

Intel Intel Arria 10 Device Datasheet

3.3.3. Hardware-Managed and Software-Managed Clocks

When changing values on clocks, the terms hardware-managed and software-managed define how clock transitions are implemented. When changing a software-managed clock, software is responsible for gating off the clock, waiting for a PLL lock if required, and gating the clock back on. Clocks that are hardware-managed are automatically transitioned by the hardware to ensure glitch-free operation.

- mpu_periph_clk
- mpu_l2_ram_clk
- mpu_clk
- l3_main_free_clk
- l4_sys_free_clk
- l4_sys_free_div4_clk
- l4_main_clk
- l4_mp_clk
- l4_sp_clk
- cs_timer_clk
- cs_at_clk
- cs_pdbg_clk
- cs_trace_clk

All other clocks in the HPS are software-managed clocks.

3.3.4. Hardware Sequenced Clock Groups

The hardware sequenced clock groups consists of the MPU clocks and the NOC clocks. The following diagram shows the external bypass muxes, hardware-managed external counters and dividers, and clock gates. For hardware-managed clocks, the group of clocks has only one software enable for the clock gate. As a result, the group of clocks are all enabled or disabled together. The slight exception is the NOC has two software enables, one for the l3/l4 clocks and one for the CoreSight clocks.

Figure 7. Hardware Clock Groups

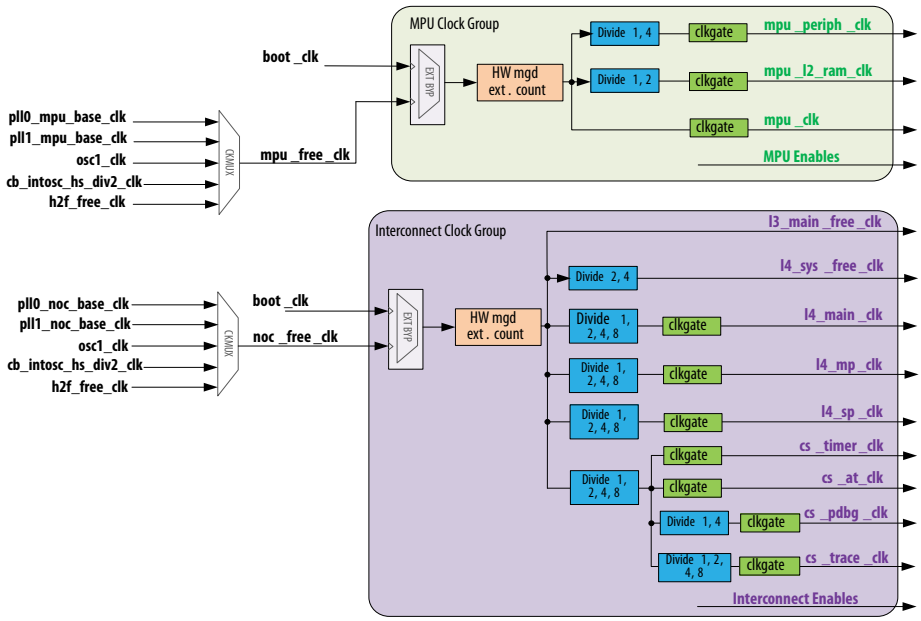


Table 14. The Hardware Sequenced Clocks Feature Summary

Clock Output Group	System Clock Name	Frequency	Boot Frequency	Uses
MPU	mpu_clk	PLL C0	boot_clk	MPU subsystem, including CPU0 and CPU1
	mpu_l2_ram_clk	mpu_clk/2	boot_clk	MPU level 2 (L2) RAM
	mpu_periph_clk	mpu_clk/4	boot_clk	MPU snoop control unit (SCU) peripherals, such as the general interrupt controller (GIC)
NOC	l3_main_free_clk	PLL C1	boot_clk	L3 interconnect
	l4_sys_free_clk	l3_main_free_clk/4	boot_clk/2	L4 interconnect
	l4_sys_free_div4_clk	l4_sys_free_clk/4	l4_sys_free_clk/4	L4 interconnect timer reference
	l4_main_clk	l3_main_free_clk/{1,2,4,8}	boot_clk	L4 main bus

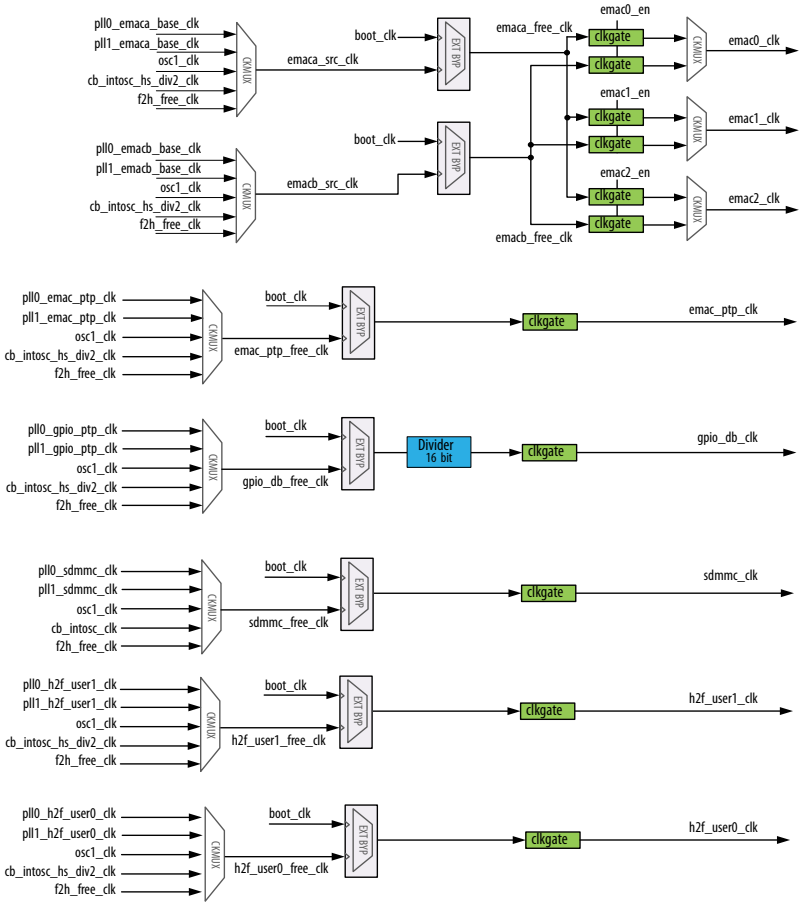
continued...

Clock Output Group	System Clock Name	Frequency	Boot Frequency	Uses
	l4_mp_clk	l3_main_free_clk/{1,2,4,8}	boot_clk	L4 MP bus
	l4_sp_clk	l3_main_free_clk/{1,4,8}	boot_clk/2	L4 SP bus
	cs_timer_clk	l3_main_free_clk/{1,2,4,8}	boot_clk	Trace timestamp generator
	cs_at_clk	l3_main_free_clk/{1,2,4,8}	boot_clk	CoreSight debug trace bus
	cs_pdbg_clk	cs_at_clk/{1,4}	cs_at_clk/2	Debug Access Port (DAP) and debug peripheral bus
	cs_trace_clk	cs_at_clk/{1,2,8}	cs_at_clk/4	Coresight debug Trace Port Interface Unit (TPIU)
Main FPGA Reference	h2f_user0_clk	h2f_user0_free_clk	boot_clk	FPGA

3.3.5. Software Sequenced Clocks

The software sequenced clock groups include additional clocks for peripherals not covered by the NOC clocks. The main purpose is to have a second PLL for the Ethernet 250 MHz clock reference. The following diagram shows the external bypass muxes, hardware-managed external counters and dividers, and clock gates.

Figure 8. Peripheral Clocks



There are 3 EMAC cores that have a very strict requirement of either a 250 MHz or 50 MHz clock reference. If the PLL0 frequency is a multiple of 250 MHz (for example 1.5 GHz), driving the EMAC clocks from PLL0 provides PLL1 with more flexibility in VCO clock frequency. In addition, to minimize the PLL clock outputs required, `emac_clka` can be 250 MHz and `emac_clkb` can be 50 MHz, allowing each EMAC core to be software configured to select 250 MHz or 50 MHz.

Table 15. Software Sequenced Clocks Feature Summary

System Clock Name	Frequency	Boot Frequency	Descriptions
`emac{0,1,2}_clk`	PLL C2 or PLL C3	`boot_clk`	Clock for EMAC. Fixed at 250 MHz or 250 MHz `emac_clk` and 50 MHz `emacb_clk`
`emac_ptp_clk`	PLL C4	`boot_clk`	Clock for EMAC PTP timestamp clock
`gpio_db_clk`	125 Hz to PLL C5	`boot_clk`	Clock for GPIO debounce clock

continued...

System Clock Name	Frequency	Boot Frequency	Descriptions
sdmmc_clk	PLL C6	boot_clk	Clock for SDMMC
h2f_user0_clk	PLL C7	boot_clk	Clock reference for FPGA
h2f_user1_clk	PLL C8	boot_clk	Clock reference for FPGA

3.3.6. Resets

Power-On-Reset (POR) Reset

The security manager handles clocking during POR. Depending on the status of one of the security fuses, the device is initially clocked using either the secure cb_intosc_hs_div2_clk or the unsecure external oscillator.

Cold Reset

The reset manager brings the clock manager out of cold reset first in order to provide clocks to the rest of the blocks. After POR is de-asserted, clock manager enables boot_clk to the rest of the system before the module resets are de-asserted.

Warm Reset

During a Warm Rest, the clock manager module is not reset. The following steps are taken during a warm reset:

1. Reset manager puts all modules affected by Warm Reset into reset. Reset manager issues a Boot Mode request to clock manager.
2. Based on the status of the hps_clk_f fuse during POR, security manager indicates if the boot clock should be secure.
   a. If secure clocks are enabled, boot_clk transitions gracefully to cb_intosc_hs_div2_clk.
   b. If secure clocks are not enabled, boot_clk transitions gracefully to the external oscillator input, HPS_CLK1.

   Note: The security fuse is only sampled during cold reset and warm reset. The security fuse hps_clk_f allows the user to enable secure clocks. If clearing RAM on a Cold or Warm reset, the user should enable secure clocks (cb_intosc_hs_clk divide by 2).
3. The *Clock Manager* gracefully transitions Hardware-Managed and Software Managed clocks into Boot Mode as follows:
   a. Disable all output clocks including Hardware and Software-Managed clocks.
   b. Wait for all clocks to be disabled, and do the following two things:

i. Bypass all external Hardware and Software-Managed clocks.
ii. Update Hardware-Managed external counters/dividers to Boot Mode settings.

c. Wait for all bypasses to switch, and then synchronously reset the CSR registers.
d. Enable all clocks.

4. After Hardware Managed Clocks have transitioned, the *Clock Manager* acknowledges the *Reset Manager*.
5. *Reset Manager* continues with Warm Reset de-assertion sequence.

Related Information

SoC Security on page 168

3.3.7. Security

The clock manager creates a boot clock as the clock reference for boot mode and external bypass. The clock configuration is based on security features in the security manager.

Security Input Clocks

The following table defines the boot clock sources.

Table 16. Security Input Clocks

Clock Name	Max	Min	Source/Dest	Description
HPS_CLK1	50	10	Pin	External Oscillator Clock Reference. Non-secure clock reference. Named osc1_clk signal inside device.
cb_intosc_hs_clk	400	120	FPGA Control Block	Control Block high speed internal ring oscillator. This clock has wide variation across process/ temperature. This clock is used as the secure reference for Boot Mode. Because the range/ jitter of the clock is low quality, the clock is divided by 2.
cb_intosc_ls_clk	100	30	FPGA Control Block	Control Block low speed internal ring oscillator. This clock has wide variation across process/ temperature. This clock is used by *Security Manager* as the Power on Reset Domain secure clock.

continued...

Clock Name	Max	Min	Source/Dest	Description
				This clock is also provided as an input to the *Clock Manager* PLLs. However, because of the low quality of the clock, it is not recommended to use this clock as the PLL reference.

Boot Clock

The status of the `hps_clk_f` security fuse in the *Security Manager* determines if `boot_clk` should be secure:

- If set, then `boot_clk` is `cb_intosc_hs_clk` divided by 2.
- If clear, then `boot_clk` comes from the external oscillator input pin `HPS_CLK1`.

The above setting is reported by the security manager before the chip is brought out of cold reset. Also the security status may be updated by security option registers in the security manager.

Related Information

SoC Security on page 168

3.3.8. Interrupts

The clock manager provides one interrupt output, which is enabled through the interrupt enable register (`intren`). The interrupt can be programmed to trigger twhen either the PLL achieves or loses its lock.

3.4. Clock Manager Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

3.4.1. clkmgr_clkmgrp Address Map

Module Instance	Base Address	End Address
`i_clk_mgr_clkmgr`	`0xFFD04000`	`0xFFD0403F`

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Register	Offset	Width	Access	Reset Value	Description
ctrl on page 70	0x0	32	RW	0x3	Control Register
intr on page 71	0x4	32	RW	0x0	Interrupt Status Register
intrs on page 72	0x8	32	RW	0x0	Interrupt Status Register Set
intrr on page 73	0xC	32	RW	0x0	Interrupt Status Register Reset
intren on page 75	0x10	32	RW	0x0	Interrupt Enable Register
intrens on page 76	0x14	32	RW	0x0	Interrupt Enable Register Set
intrenr on page 78	0x18	32	RW	0x0	Interrupt Enable Register Reset
stat on page 79	0x1C	32	RO	0x10000	Status Register
testioctrl on page 80	0x20	32	RW	0x100808	Test IO Control Register

3.4.1.1. clkmgr_clkmgrp Summary

Base Address: 0xFFD04000

Register Address Offset	Bit Fields															
i_clk_mgr_clkmgr																
ctrl 0x0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved						swctrlbtclksel RW 0x0	swctrlbtclken RW 0x0	Reserved							boot mode RW 0x1
intr 0x4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

continued...

Register Address Offset	Bit Fields															
	Reserved				perllfbslip RW 0x0	mainpllfbslip RW 0x0	perllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perlllost RW 0x0	mainplllost RW 0x0	perllachieved RW 0x0	mainpllachieved RW 0x0
intrs 0x8	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				perllfbslip RW 0x0	mainpllfbslip RW 0x0	perllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perlllost RW 0x0	mainplllost RW 0x0	perllachieved RW 0x0	mainpllachieved RW 0x0
intrr 0xC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				perllfbslip RW 0x0	mainpllfbslip RW 0x0	perllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perlllost RW 0x0	mainplllost RW 0x0	perllachieved RW 0x0	mainpllachieved RW 0x0
intren 0x10	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				perllfbslip RW 0x0	mainpllfbslip RW 0x0	perllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perlllost RW 0x0	mainplllost RW 0x0	perllachieved RW 0x0	mainpllachieved RW 0x0
intrens 0x14	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				perllfbslip RW 0x0	mainpllfbslip RW 0x0	perllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perlllost RW 0x0	mainplllost RW 0x0	perllachieved RW 0x0	mainpllachieved RW 0x0
intrenr 0x18	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				perllfbslip RW 0x0	mainpllfbslip RW 0x0	perllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perlllost RW 0x0	mainplllost RW 0x0	perllachieved RW 0x0	mainpllachieved RW 0x0

continued...

Register Address Offset	Bit Fields															
stat 0x1C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved														bootclksrc RO 0x0	bootmode RO 0x1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved						perpllllocked RO 0x0	mainpllllocked RO 0x0	Reserved							busy RO 0x0
testioctrl 0x20	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved											debugclksel RW 0x10				
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				periclksel RW 0x8				Reserved				mainclksel RW 0x8			

3.4.1.2. ctrl

Contains fields that control the entire Clock Manager.

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD04000

Offset: 0x0

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						swctrlbtclksel RW 0x0	swctrlbtclken RW 0x0	Reserved							bootmode RW 0x1

ctrl Fields

Bit	Name	Description	Access	Reset
9	swctrlbtclksel	This bit is only used if swctrlbtclken is set. If 1, boot_clk source will be from cb_intosc_hs_clk divided by 2. If 0, boot_clk source will be from the external oscillator (HPS_CLK1 input pin that becomes signal osc1_clk in the device). This bit is cleared on a cold reset. Warm reset has no affect on this bit.	RW	0x0
8	swctrlbtclken	If set, then Software will take control of the boot_clk mux select. If set, then swctrlbtclksel will determine the mux setting. If not set, the security features will determine the fuse settings. This bit is cleared on a cold reset. Warm reset has no affect on this bit.	RW	0x0
0	bootmode	When set the Clock Manager is in Boot Mode. In Boot Mode Clock Manager register settings defining clock behavior are ignored and clocks are set to their Boot Mode settings. All clocks will be bypassed and external HW managed counters and dividers will be set to divide by 1. This bit should only be cleared when clocks have been correctly configured. This field is set on a cold reset and optionally on a warm reset. SW may set this bit to force the clocks into Boot Mode. SW exits Boot Mode by clearing this bit.	RW	0x1

3.4.1.3. intr

Contains interrupt fields for Clock Manager

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD04004

Offset: 0x4

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				perpllfbslip RW 0x0	mainpllfbslip RW 0x0	perpllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perpllost RW 0x0	mainplllost RW 0x0	perpllachieved RW 0x0	mainpllachieved RW 0x0

intr Fields

Bit	Name	Description	Access	Reset
11	perpllfbslip	If 1, the Peripheral PLL feedback cycle has slipped (CLKOUT frequency too low). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
10	mainpllfbslip	If 1, the Main PLL feedback cycle has slipped (CLKOUT frequency too low). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
9	perpllrfslip	If 1, the Peripheral PLL reference cycle has slipped (CLKOUT frequency too high). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
8	mainpllrfslip	If 1, the Main PLL reference cycle has slipped (CLKOUT frequency too high). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
3	perplllost	If 1, the Peripheral PLL has lost lock at least once since this bit was cleared. If 0, the Peripheral PLL has not lost lock since this bit was cleared.	RW	0x0
2	mainplllost	If 1, the Main PLL has lost lock at least once since this bit was cleared. If 0, the Main PLL has not lost lock since this bit was cleared.	RW	0x0
1	perpllachieved	If 1, the Peripheral PLL has achieved lock at least once since this bit was cleared. If 0, the Peripheral PLL has not achieved lock since this bit was cleared.	RW	0x0
0	mainpllachieved	If 1, the Main PLL has achieved lock at least once since this bit was cleared. If 0, the Main PLL has not achieved lock since this bit was cleared.	RW	0x0

3.4.1.4. intrs

Contains fields that indicate the PLL lock status.

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD04008

Offset: 0x8

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				perpllfbslip RW 0x0	mainpllfbslip RW 0x0	perpllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perplllost RW 0x0	mainplllost RW 0x0	perpllachieved RW 0x0	mainpllachieved RW 0x0

intrs Fields

Bit	Name	Description	Access	Reset
11	perpllfbslip	If 1, the Peripheral PLL feedback cycle has slipped (CLKOUT frequency too low). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
10	mainpllfbslip	If 1, the Main PLL feedback cycle has slipped (CLKOUT frequency too low). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
9	perpllrfslip	If 1, the Peripheral PLL reference cycle has slipped (CLKOUT frequency too high). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
8	mainpllrfslip	If 1, the Main PLL reference cycle has slipped (CLKOUT frequency too high). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
3	perplllost	If 1, the Peripheral PLL has lost lock at least once since this bit was cleared. If 0, the Peripheral PLL has not lost lock since this bit was cleared.	RW	0x0
2	mainplllost	If 1, the Main PLL has lost lock at least once since this bit was cleared. If 0, the Main PLL has not lost lock since this bit was cleared.	RW	0x0
1	perpllachieved	If 1, the Peripheral PLL has achieved lock at least once since this bit was cleared. If 0, the Peripheral PLL has not achieved lock since this bit was cleared.	RW	0x0
0	mainpllachieved	If 1, the Main PLL has achieved lock at least once since this bit was cleared. If 0, the Main PLL has not achieved lock since this bit was cleared.	RW	0x0

3.4.1.5. intrr

Contains fields that indicate the PLL lock status.

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD0400C

Offset: 0xC

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				perpllfbslip RW 0x0	mainpllfbslip RW 0x0	perpllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perpllost RW 0x0	mainpllost RW 0x0	perplachieved RW 0x0	mainpllachieved RW 0x0

intrr Fields

Bit	Name	Description	Access	Reset
11	perpllfbslip	If 1, the Peripheral PLL feedback cycle has slipped (CLKOUT frequency too low). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
10	mainpllfbslip	If 1, the Main PLL feedback cycle has slipped (CLKOUT frequency too low). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
9	perpllrfslip	If 1, the Peripheral PLL reference cycle has slipped (CLKOUT frequency too high). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
8	mainpllrfslip	If 1, the Main PLL reference cycle has slipped (CLKOUT frequency too high). This does not mean the PLL has lost lock, but the quality of the clock has degraded.	RW	0x0
3	perpllost	If 1, the Peripheral PLL has lost lock at least once since this bit was cleared. If 0, the Peripheral PLL has not lost lock since this bit was cleared.	RW	0x0

continued...

Bit	Name	Description	Access	Reset
2	mainplllost	If 1, the Main PLL has lost lock at least once since this bit was cleared. If 0, the Main PLL has not lost lock since this bit was cleared.	RW	0x0
1	perpllachieved	If 1, the Peripheral PLL has achieved lock at least once since this bit was cleared. If 0, the Peripheral PLL has not achieved lock since this bit was cleared.	RW	0x0
0	mainpllachieved	If 1, the Main PLL has achieved lock at least once since this bit was cleared. If 0, the Main PLL has not achieved lock since this bit was cleared.	RW	0x0

3.4.1.6. intren

Contain fields that enable the interrupt

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD04010

Offset: 0x10

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				perpllfbslip RW 0x0	mainpllfbslip RW 0x0	perpllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perpllost RW 0x0	mainplllost RW 0x0	perpllachieved RW 0x0	mainpllachieved RW 0x0

intren Fields

Bit	Name	Description	Access	Reset
11	perpllfbslip	When set to 1,the Peripheral PLL feedback cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Peripheral PLL feedback cylce slipped bit is not ORed into the Clock Manager interrupt output.	RW	0x0
10	mainpllfbslip	When set to 1,the Main PLL feedback cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Main PLL feedback cylce slipped bit is not ORed into the Clock Manager interrupt output.	RW	0x0
9	perpllrfslip	When set to 1,the Peripheral PLL reference cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Peripheral PLL reference cylce slipped bit is not ORed into the Clock Manager interrupt output.	RW	0x0
8	mainpllrfslip	When set to 1,the Main PLL reference cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Main PLL reference cylce slipped bit is not ORed into the Clock Manager interrupt output.	RW	0x0
3	perplllost	When set to 1, the Peripheral PLL lost lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Peripheral PLL lost lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
2	mainplllost	When set to 1, the Main PLL lost lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Main PLL lost lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
1	perpllachieved	When set to 1, the Peripheral PLL achieved lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Peripheral PLL achieved lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
0	mainpllachieved	When set to 1, the Main PLL achieved lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Main PLL achieved lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0

3.4.1.7. intrens

Contain fields that enable the interrupt

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD04014

Offset: 0x14

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				perpllfbslip RW 0x0	mainpllfbslip RW 0x0	perpllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perplllost RW 0x0	mainplllost RW 0x0	perpllachieved RW 0x0	mainpllachieved RW 0x0

intrens Fields

Bit	Name	Description	Access	Reset
11	perpllfbslip	When set to 1,the Peripheral PLL feedback cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Peripheral PLL feedback cylce slipped bit is Ored into the Clock Manager interrupt output.	RW	0x0
10	mainpllfbslip	When set to 1,the Main PLL feedback cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Main PLL feedback cylce slipped bit is Ored into the Clock Manager interrupt output.	RW	0x0
9	perpllrfslip	When set to 1,the Peripheral PLL reference cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Peripheral PLL reference cylce slipped bit is Ored into the Clock Manager interrupt output.	RW	0x0
8	mainpllrfslip	When set to 1,the Main PLL reference cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Main PLL reference cylce slipped bit is Ored into the Clock Manager interrupt output.	RW	0x0
3	perplllost	When set to 1, the Peripheral PLL lost lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Peripheral PLL lost lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
2	mainplllost	When set to 1, the Main PLL lost lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Main PLL lost lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
1	perpllachieved	When set to 1, the Peripheral PLL achieved lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Peripheral PLL achieved lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
0	mainpllachieved	When set to 1, the Main PLL achieved lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Main PLL achieved lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0

3.4.1.8. intrenr

Contain fields that enable the interrupt

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD04018

Offset: 0x18

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				perpllfbslip RW 0x0	mainpllfbslip RW 0x0	perpllrfslip RW 0x0	mainpllrfslip RW 0x0	Reserved				perpllost RW 0x0	mainpllost RW 0x0	perpllachieved RW 0x0	mainpllachieved RW 0x0

intrenr Fields

Bit	Name	Description	Access	Reset
11	perpllfbslip	When set to 1,the Peripheral PLL feedback cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Peripheral PLL feedback cylce slipped bit is Ored into the Clock Manager interrupt output.	RW	0x0
10	mainpllfbslip	When set to 1,the Main PLL feedback cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Main PLL feedback cylce slipped bit is Ored into the Clock Manager interrupt output.	RW	0x0
9	perpllrfslip	When set to 1,the Peripheral PLL reference cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Peripheral PLL reference cylce slipped bit is Ored into the Clock Manager interrupt output.	RW	0x0
8	mainpllrfslip	When set to 1,the Main PLL reference cycle slipped bit is ORed into the Clock Manager interrupt output. When set to 0, the Main PLL reference cylce slipped bit is Ored into the Clock Manager interrupt output.	RW	0x0

continued...

Bit	Name	Description	Access	Reset
3	perplllost	When set to 1, the Peripheral PLL lost lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Peripheral PLL lost lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
2	mainplllost	When set to 1, the Main PLL lost lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Main PLL lost lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
1	perpllachieved	When set to 1, the Peripheral PLL achieved lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Peripheral PLL achieved lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0
0	mainpllachieved	When set to 1, the Main PLL achieved lock bit is ORed into the Clock Manager interrupt output. When set to 0 the Main PLL achieved lock bit is not ORed into the Clock Manager interrupt output.	RW	0x0

3.4.1.9. stat

Provides status for Clock Manager including PLL lock and HW Managed Clock State Machine busy.

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD0401C

Offset: 0x1C

Access: RO

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														bootclksrc RO 0x0	bootmode RO 0x1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						perpllocked RO 0x0	mainplllocked RO 0x0	Reserved							busy RO 0x0

stat Fields

Bit	Name	Description	Access	Reset
17	bootclksrc	If 1, the source of boot_clk is cb_intosc_hs_div2_clk. . If 0, the boot_clk source is the external oscillator (HPS_CLK1 input pin which becomes signal osc1_clk in the device). This is a read only status.	RO	0x0
16	bootmode	If 1, the clocks are currently in Boot Mode. If 0, the clocks are not in Boot Mode. This is a read only status. For SW to exit Boot Mode, SW must clear the RW bit CTRL.BOOTMODE.	RO	0x1
9	perplllocked	If 1, the Peripheral PLL is currently locked. If 0, the Peripheral PLL is currently not locked.	RO	0x0
8	mainplllocked	If 1, the Main PLL is currently locked. If 0, the Main PLL is currently not locked.	RO	0x0
0	busy	This read only bit indicates that the Hardware Managed clock's state machine is active. If the state machine is active, then the clocks are in transition. Software should poll this bit after changing the source of internal clocks when changing the state of CTRL.BOOTMODE, MAINPLLGRP.BYPASS.MPU or MAINPLLGRP.BYPASS.NOC register bits. Immediately following writes to any of these registers, SW should wait 0.5 usecs and then poll this BUSY bit until it is IDLE before proceeding with any other register writes in the Clock Manager. The reset value of this bit is applied on a cold reset; warm reset has no affect on this bit. Value: 0 — IDLE; 1 — BUSY	RO	0x0

3.4.1.10. testioctrl

Contains fields setting the IO output select for Test Clock and Debug outputs. The dedicated IO outputs includes two outputs for the Main PLL clock outputs (PLL_CLK0 and PLL_CLK1), two outputs for the Peripheral PLL clock outputs (PLL_CLK2 and PLL_CLK3), and one output for miscelaneous debug for the Main and Peripheral PLL (PLL_CLK4).

The Test Clock and Debug outputs will only propagate to the dedicated IO based on the IO pinmux configuration. If Test Clocks are selected in the pinmux, then the selects in this register determine which PLL clocks and PLL debug signals will propagate to the IOs.

Module Instance	Base Address	Register Address
i_clk_mgr_clkmgr	0xFFD04000	0xFFD04020

Offset: 0x20

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved											debugclksel RW 0x10				
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				periclksel RW 0x8				Reserved				mainclksel RW 0x8			

testioctrl Fields

Bit	Name	Description	Access	Reset
20:16	debugclksel	Selects the source of PLL_CLK4 for miscellaneous PLL signals. Bit[3] (p below) determines if from the debug output is from the Main or Peripheral PLL. If 0, the the output is from the Main PLL and if 1, the output is from the Peripheral PLL. The following table determines the PLL debug output select: sel PLL_CLK4 0p000 OUTRESETACK0 0p001 OUTRESETACK3 0p010 OUTRESETACK7 0p011 PLLRESET 0p100 OUTRESETACK15 0p101 FBSLIP 0p110 RFSLIP 0p111 LOCK 1xxxx VSS	RW	0x10
11:8	periclksel	Selects the source of PLL_CLK2 and PLL_CLK3 dedicated IO outputs if selected. All of the CLKOUT# counter outputs are from the Peripheral PLL. The following table determines the PLL counter output select: sel PLL_CLK2 PLL_CLK3 0000 CLKOUT0 CLKOUT8 0001 CLKOUT1 CLKOUT9 0010 CLKOUT2 CLKOUT10 0011 CLKOUT3 CLKOUT11 0100 CLKOUT4 CLKOUT13 0101 CLKOUT5 CLKOUT14 0110 CLKOUT6 CLKOUT15 0111 CLKOUT7 CLKOUT16 1xxx VSS VSS	RW	0x8
3:0	mainclksel	Selects the source of PLL_CLK0 and PLL_CLK1 dedicated IO outputs if selected. All of the CLKOUT# counter outputs are from the Main PLL. The following table determines the PLL counter output select: sel PLL_CLK0 PLL_CLK1 0000 CLKOUT0 CLKOUT8	RW	0x8

continued...

Bit	Name	Description	Access	Reset
		0001 CLKOUT1 CLKOUT9 0010 CLKOUT2 CLKOUT10 0011 CLKOUT3 CLKOUT11 0100 CLKOUT4 CLKOUT13 0101 CLKOUT5 CLKOUT14 0110 CLKOUT6 CLKOUT15 0111 CLKOUT7 CLKOUT16 1xxx VSS VSS		

3.4.2. clkmgr_mainpllgrp Address Map

Module Instance	Base Address	End Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD040BF

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Register	Offset	Width	Access	Reset Value	Description
vco0 on page 86	0x0	32	RW	0x10043	Main PLL VCO Control Register 0
vco1 on page 88	0x4	32	RW	0x10001	Main PLL VCO Control Register 1
en on page 89	0x8	32	RW	0xFF	Enable Register
ens on page 90	0xC	32	RW	0xFF	Enable Set Register
enr on page 91	0x10	32	RW	0xFF	Enable Reset Register
bypass on page 92	0x14	32	RW	0x3F	Bypass Register
bypasss on page 93	0x18	32	RW	0x3F	Bypass Set Register
bypassr on page 94	0x1C	32	RW	0x3F	Bypass Reset Register
mpuclk on page 95	0x20	32	RW	0x0	Main PLL Control Register for MPU Clock Group.
nocclk on page 96	0x24	32	RW	0x0	Main PLL Control Register for NOC Clock Group.

continued...

Register	Offset	Width	Access	Reset Value	Description
cntr2clk on page 97	0x28	32	RW	0x0	Main PLL Control Register for Counter 2 Clock
cntr3clk on page 98	0x2C	32	RW	0x0	Main PLL Control Register for Counter 3 Clock
cntr4clk on page 99	0x30	32	RW	0x0	Main PLL Control Register for Counter 4 Clock
cntr5clk on page 99	0x34	32	RW	0x0	Main PLL Control Register for Counter 5 Clock
cntr6clk on page 100	0x38	32	RW	0x0	Main PLL Control Register for Counter 6 Clock
cntr7clk on page 101	0x3C	32	RW	0x0	Main PLL Control Register for Counter 7 Clock
cntr8clk on page 102	0x40	32	RW	0x0	Main PLL Control Register for Counter 8 Clock
outrst on page 102	0x60	32	RW	0x0	Main PLL Output Counter Reset Register
outrststat on page 103	0x64	32	RO	0x0	Main PLL Output Counter Reset Ack Status Register
nocdiv on page 104	0x68	32	RW	0x18020100	NoC Divide Register

3.4.2.1. clkmgr_mainpllgrp Summary

Base Address: 0xFFD04040

Register Address Offset	Bit Fields															
i_clk_mgr_mainpllgrp																
vco0 0x0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved			bwadjen RW 0x0	bwadj RW 0x1											
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

continued...

Register Address Offset	Bit Fields															
	Reserved						psrc RW 0x0		Reserved	saten RW 0x1	fasten RW 0x0	regextsel RW 0x0	outreseteall RW 0x0	en RW 0x0	pwrdn RW 0x1	bgpwrdn RW 0x1
vco1 0x4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved										denom RW 0x1					
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved			numer RW 0x1												
en 0x8	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								hmcpllrefclken RW 0x1	s2fuser0clken RW 0x1	cstimerclken RW 0x1	csclken RW 0x1	l4spclken RW 0x1	l4mpclken RW 0x1	l4mainclken RW 0x1	mpuclken RW 0x1
ens 0xC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								hmcpllrefclken RW 0x1	s2fuser0clken RW 0x1	cstimerclken RW 0x1	csclken RW 0x1	l4spclken RW 0x1	l4mpclken RW 0x1	l4mainclken RW 0x1	mpuclken RW 0x1
enr 0x10	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								hmcpllrefclken RW 0x1	s2fuser0clken RW 0x1	cstimerclken RW 0x1	csclken RW 0x1	l4spclken RW 0x1	l4mpclken RW 0x1	l4mainclken RW 0x1	mpuclken RW 0x1
bypass 0x14	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved										fben RW 0x1	rfen RW 0x1	hmcpllref RW 0x1	s2fuser0 RW 0x1	noc RW 0x1	mpu RW 0x1
bypasss 0x18	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															

continued...

Register Address Offset	Bit Fields															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved										fben RW 0x1	rfen RW 0x1	hmcpllref RW 0x1	s2fuser0 RW 0x1	noc RW 0x1	mpu RW 0x1
bypassr 0x1C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved										fben RW 0x1	rfen RW 0x1	hmcpllref RW 0x1	s2fuser0 RW 0x1	noc RW 0x1	mpu RW 0x1
mpuclk 0x20	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved													src RW 0x0		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
nocclk 0x24	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved													src RW 0x0		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr2clk 0x28	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr3clk 0x2C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr4clk 0x30	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr5clk 0x34	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															

continued...

Register Address Offset	Bit Fields															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr6clk 0x38	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr7clk 0x3C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved													src RW 0x0		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr8clk 0x40	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
outrst 0x60	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	outreset RW 0x0															
outrststat 0x64	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	outresetack RO 0x0															
nocdiv 0x68	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved			cspdbgclk RW 0x1	cstraceclk RW 0x2		csatclk RW 0x0		Reserved						l4spclk RW 0x2	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved						l4mpclk RW 0x1		Reserved						l4mainclk RW 0x0	

3.4.2.2. vco0

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04040

Offset: 0x0

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			bwadjen RW 0x0	bwadj RW 0x1											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						psrc RW 0x0		Reserved	saten RW 0x1	fasten RW 0x0	regextsel RW 0x0	outresetall RW 0x0	en RW 0x0	pwrdn RW 0x1	bgpwrdn RW 0x1

vco0 Fields

Bit	Name	Description	Access	Reset
28	bwadjen	If set to 1, the Loop Bandwidth Adjust value comes from the Loop Bandwidth Adjust field. If set to 0, the Loop Bandwidth Adjust value equals the M field divided by 2 value of the VCO Control Register. The M divided by 2 is the upper 12 bits (12:1) of the M field in the VCO register.	RW	0x0
27:16	bwadj	Provides Loop Bandwidth Adjust value.	RW	0x1
9:8	psrc	Controls the VCO input clock source. Value: 0 — HPS_CLK1 (external oscillator input pin). Internal signal within device is named osc1_clk. Value: 1 — cb_intosc_ls_clk (internal low speed oscillator) Value: 2 — f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
6	saten	Enables saturation behavior.	RW	0x1
5	fasten	Enables fast locking circuit.	RW	0x0

continued...

Bit	Name	Description	Access	Reset
4	regextsel	If set to '1', the external regulator is selected for the PLL. If set to '0', the internal regulator is slected. It is strongly recommended to select the external regulator while the PLL is not enabled (in reset), and then disable the external regulater once the PLL becomes enabled. Software should simulateously update the 'Enable' bit and the 'External Regulator Input Select' in the same write access to the VCO register. When the 'Enable' bit is clear, the 'External Regulator Input Select' should be set, and vice versa. The reset value of this bit is applied on a cold reset; warm reset has no affect on this bit.	RW	0x0
3	outresetall	Before releasing Bypass, All Output Counter Reset must be set and cleared by software for correct clock operation. If '1', Reset phase multiplexer and all output counter state. So that after the assertion all the clocks output are start from rising edge align. If '0', phase multiplexer and output counter state not reset and no change to the phase of the clock outputs.	RW	0x0
2	en	If '1', VCO is enabled. If '0', VCO is in reset.	RW	0x0
1	pwrdn	If '1', power down analog circuitry. If '0', analog circuitry not powered down.	RW	0x1
0	bgpwrdn	If '1', powers down bandgap. If '0', bandgap is not power down.	RW	0x1

3.4.2.3. vco1

Contains settings that control the Main PLL VCO. The VCO1 register contains the numerator and denominator counter settings.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04044

Offset: 0x4

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										denom RW 0x1					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			numer RW 0x1												

vco1 Fields

Bit	Name	Description	Access	Reset
21:16	denom	Denominator in VCO output frequency equation. For incremental frequency change, if the new value lead to less than 20% of the frequency change, this value can be changed without resetting the PLL. The Numerator and Denominator can not be changed at the same time for incremental frequency changed.	RW	0x1
12:0	numer	Numerator in VCO output frequency equation. For incremental frequency change, if the new value lead to less than 20% of the frequency change, this value can be changed without resetting the PLL. The Numerator and Denominator can not be changed at the same time for incremental frequency changed.	RW	0x1

3.4.2.4. en

Contains fields that control clock enables for Main Clocks.
1: The clock is enabled.
0: The clock is disabled.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04048

Offset: 0x8

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								hmcpllrefclken RW 0x1	s2fuser0clken RW 0x1	cstimerclken RW 0x1	csclken RW 0x1	l4spclken RW 0x1	l4mpclken RW 0x1	l4mainclken RW 0x1	mpuclken RW 0x1

en Fields

Bit	Name	Description	Access	Reset
7	hmcpllrefclken	Enables clock hmc_pll_ref_clk output	RW	0x1
6	s2fuser0clken	Enables clock s2f_user0_clk output	RW	0x1
5	cstimerclken	Enables Debug Timer Clock output (cs_timer_clk)	RW	0x1
4	csclken	Enables Debug Clock outputs (cs_at_clk, cs_pdbg_clk, and cs_trace_clk)	RW	0x1
3	l4spclken	Enables clock l4_sp_clk output	RW	0x1
2	l4mpclken	Enables clock l4_mp_clk output	RW	0x1
1	l4mainclken	Enables clock l4_main_clk output	RW	0x1
0	mpuclken	Enable for MPU Clock Group (mpu_clk, mpu_l2ram_clk and mpu_periph_clk).	RW	0x1

3.4.2.5. ens

Write One to Set correspondng fields in the Enable Register.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD0404C

Offset: 0xC

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								hmcpllrefclken RW 0x1	s2fuser0clken RW 0x1	cstimerclken RW 0x1	csclken RW 0x1	l4spclken RW 0x1	l4mpclken RW 0x1	l4mainclken RW 0x1	mpuclken RW 0x1

ens Fields

Bit	Name	Description	Access	Reset
7	hmcpllrefclken	Enables clock hmc_pll_ref_clk output	RW	0x1
6	s2fuser0clken	Enables clock s2f_user0_clk output	RW	0x1
5	cstimerclken	Enables Debug Timer Clock output (cs_timer_clk)	RW	0x1
4	csclken	Enables Debug Clock outputs (cs_at_clk, cs_pdbg_clk and cs_trace_clk)	RW	0x1
3	l4spclken	Enables clock l4_sp_clk output	RW	0x1
2	l4mpclken	Enables clock l4_mp_clk output	RW	0x1
1	l4mainclken	Enables clock l4_main_clk output	RW	0x1
0	mpuclken	Enable for MPU Clock Group (mpu_clk, mpu_l2ram_clk and mpu_periph_clk).	RW	0x1

3.4.2.6. enr

Write One to Clear corresponding fields in Enable Register.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04050

Offset: 0x10

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								hmcpllrefclken RW 0x1	s2fuser0clken RW 0x1	cstimerclken RW 0x1	csclken RW 0x1	l4spclken RW 0x1	l4mpclken RW 0x1	l4mainclken RW 0x1	mpuclken RW 0x1

enr Fields

Bit	Name	Description	Access	Reset
7	hmcpllrefclken	Enables clock hmc_pll_ref_clk output	RW	0x1
6	s2fuser0clken	Enables clock s2f_user0_clk output	RW	0x1
5	cstimerclken	Enables Debug Timer Clock output (cs_timer_clk)	RW	0x1
4	csclken	Enables Debug Clock outputs (cs_at_clk, cs_pdbg_clk and cs_trace_clk)	RW	0x1
3	l4spclken	Enables clock l4_sp_clk output	RW	0x1
2	l4mpclken	Enables clock l4_mp_clk output	RW	0x1
1	l4mainclken	Enables clock l4_main_clk output	RW	0x1
0	mpuclken	Enable for MPU Clock Group (mpu_clk, mpu_l2ram_clk and mpu_periph_clk).	RW	0x1

3.4.2.7. bypass

```
        Contains fields that control bypass for clocks derived from the Main PLL.
1: The clock is bypassed to boot_clk.
0: The clock is derived from the 5:1 active mux.
```

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04054

Offset: 0x14

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										fben RW 0x1	rfen RW 0x1	hmcpllref RW 0x1	s2fuser0 RW 0x1	noc RW 0x1	mpu RW 0x1

bypass Fields

Bit	Name	Description	Access	Reset
5	fben	If set, the pll_main_fben_clk will be bypassed to the boot_clk. The pll_main_fben_clk is used to synchronously update the Numerator to the Main PLL.	RW	0x1
4	rfen	If set, the pll_main_rfen_clk will be bypassed to the boot_clk. The pll_main_rfen_clk is used to synchronously update the Denominator to the Main PLL.	RW	0x1
3	hmcpllref	If set, the hmc_pll_ref_clk will be bypassed to the boot_clk.	RW	0x1
2	s2fuser0	If set, the s2f_user0_clk will be bypassed to the boot_clk.	RW	0x1
1	noc	If set, the NOC clock group will be bypassed to boot_clk.	RW	0x1
0	mpu	If set, the MPU clock group will be bypassed to the boot_clk.	RW	0x1

3.4.2.8. bypasss

Write One to Set corresponding fields in Bypass Register.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04058

Offset: 0x18

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										fben RW 0x1	rfen RW 0x1	hmcpllref RW 0x1	s2fuser0 RW 0x1	noc RW 0x1	mpu RW 0x1

bypasss Fields

Bit	Name	Description	Access	Reset
5	fben	If set, the pll_main_fben_clk will be bypassed to the boot_clk. The pll_main_fben_clk is used to synchronously update the Numerator to the Main PLL.	RW	0x1
4	rfen	If set, the pll_main_rfen_clk will be bypassed to the boot_clk. The pll_main_rfen_clk is used to synchronously update the Denominator to the Main PLL.	RW	0x1
3	hmcpllref	If set, the hmc_pll_ref_clk will be bypassed to the boot_clk.	RW	0x1
2	s2fuser0	If set, the s2f_user0_clk will be bypassed to the boot_clk.	RW	0x1
1	noc	If set, the NOC clock group will be bypassed to the boot_clk.	RW	0x1
0	mpu	If set, the MPU clock group will be bypassed to the boot_clk.	RW	0x1

3.4.2.9. bypassr

Write One to Clear corresponding fields in Bypass Register.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD0405C

Offset: 0x1C

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										fben RW 0x1	rfen RW 0x1	hmcpllref RW 0x1	s2fuser0 RW 0x1	noc RW 0x1	mpu RW 0x1

bypassr Fields

Bit	Name	Description	Access	Reset
5	fben	If set, the pll_main_fben_clk will be bypassed to the boot_clk. The pll_main_fben_clk is used to synchronously update the Numerator to the Main PLL.	RW	0x1
4	rfen	If set, the pll_main_rfen_clk will be bypassed to the boot_clk. The pll_main_rfen_clk is used to synchronously update the Denominator to the Main PLL.	RW	0x1
3	hmcpllref	If set, the hmc_pll_ref_clk will be bypassed to the boot_clk.	RW	0x1
2	s2fuser0	If set, the s2f_user0_clk will be bypassed to the input clock reference of the Main PLL.	RW	0x1
1	noc	If set, the NOC clock group will be bypassed to the input clock reference of the Main PLL.	RW	0x1
0	mpu	If set, the MPU clock group will be bypassed to the input clock reference of the Main PLL.	RW	0x1

3.4.2.10. mpuclk

Contains settings that control clock mpu_clk generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04060

Offset: 0x20

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

mpuclk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value / Description: 0 — pll0_mpu_base_clk 1 — pll1_mpu_base_clk 2 — osc1_clk (external oscillator input pin HPS_CLK1) 3 — cb_intosc_hs_div2_clk (internal high speed oscillator divided by 2) 4 — f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO/2 frequency by the value+1 in this field.	RW	0x0

3.4.2.11. nocclk

Contains settings that control clock main_clk generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04064

Offset: 0x24

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

nocclk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value / Description: 0 — pll0_noc_base_clk 1 — pll1_noc_base_clk 2 — osc1_clk (external oscillator input pin, HPS_CLK1) 3 — cb_intosc_hs_div2_clk (high speed internal oscillator divided by 2) 4 — f2h_free_clk (FPGA Fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.2.12. cntr2clk

Contains settings that control Counter 2 clock generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04068

Offset: 0x28

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr2clk Fields

Bit	Name	Description	Access	Reset
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.2.13. cntr3clk

Contains settings that control Counter 3 clock generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD0406C

Offset: 0x2C

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr3clk Fields

Bit	Name	Description	Access	Reset
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.2.14. cntr4clk

Contains settings that control Counter 4 clock generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04070

Offset: 0x30

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr4clk Fields

Bit	Name	Description	Access	Reset
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.2.15. cntr5clk

Contains settings that control Counter 5 clock generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04074

Offset: 0x34

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr5clk Fields

Bit	Name	Description	Access	Reset
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.2.16. cntr6clk

Contains settings that control Counter 6 clock generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04078

Offset: 0x38

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr6clk Fields

Bit	Name	Description	Access	Reset
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.2.17. cntr7clk

Contains settings that control Counter 7 clock generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD0407C

Offset: 0x3C

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr7clk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value: 0 — pll0_h2f_user0_clk Value: 1 — pll1_h2f_user0_clk Value: 2 — osc1_clk (external oscillator input pin, HPS_CLK1) Value: 3 — cb_intosc_hs_div2_clk (internal high speed oscillator divided by 2) Value: 4 — f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.2.18. cntr8clk

Contains settings that control Counter 8 clock generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD04080

Offset: 0x40

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr8clk Fields

Bit	Name	Description	Access	Reset
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.2.19. outrst

Contains settings to assert individual Outreset for all Main PLL Counters.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD040A0

Offset: 0x60

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
outreset RW 0x0															

outrst Fields

Bit	Name	Description	Access	Reset
15:0	outreset	Resets the individual PLL output counter. For software to change the PLL output counter without producing glitches on the respective clock, SW must set the Output Counter Reset Register 'Output Counter Reset' bit. Software then polls the respective Output Counter Reset Acknowledge bit in the Output Counter Reset Ack Status Register. Software then writes the appropriate counter register, and then clears the respective Output Counter Reset bit. LSB 'outreset[0]' corresponds to PLL output clock C0, etc. If set to '1', reset output divider, no clock output from counter. If set to '0', counter is not reset. The reset value of this bit is applied on a cold reset; warm reset has no affect on this bit.	RW	0x0

3.4.2.20. outrststat

Contains Output Clock Counter Reset acknowledge status.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD040A4

Offset: 0x64

Access: RO

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
outresetack RO 0x0															

outrststat Fields

Bit	Name	Description	Access	Reset
15:0	outresetack	These read only bits per PLL output indicate that the PLL has received the Output Reset Counter request and has gracefully stopped the respective PLL output clock. For software to change the PLL output counter without producing glitches on the respective clock, SW must set the Output Counter Reset Register 'Output Counter Reset' bit. Software then polls the respective Output Counter Reset Acknowledge bit in the Output Counter Reset Ack Status Register. Software then writes the appropriate counter register, and then clears the respective Output Counter Reset bit. The reset value of this bit is applied on a cold reset; warm reset has no affect on this bit. **Value / Description** 0 — IDLE 1 — ACK_RECEIVED	RO	0x0

3.4.2.21. nocdiv

Contains fields that control clock dividers for NoC Clocks.

Module Instance	Base Address	Register Address
i_clk_mgr_mainpllgrp	0xFFD04040	0xFFD040A8

Offset: 0x68

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			cspdbgclk RW 0x1	cstraceclk RW 0x2		csatclk RW 0x0		Reserved						l4spclk RW 0x2	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						l4mpclk RW 0x1		Reserved						l4mainclk RW 0x0	

nocdiv Fields

Bit	Name	Description	Access	Reset
28	cspdbgclk	The external cs_pdbg_clk divider is specified in this field. This divider is cascaded after the cs_at_clk external divider. Value: 0 — Div1 Value: 1 — Div4	RW	0x1
27:26	cstraceclk	The external cs_trace_clk divider is specified in this field. The cs_trace_clk is used by the actual trace interface to the debugger. This divider is cascaded after the cs_at_clk external divider. Value: 0 — Div1 Value: 1 — Div2 Value: 2 — Div4 Value: 3 — Div8	RW	0x2
25:24	csatclk	The external cs_at_clk divider is specified in this field. Value: 0 — Div1 Value: 1 — Div2 Value: 2 — Div4 Value: 3 — Div8	RW	0x0
17:16	l4spclk	The external l4_sp_clk divider is specified in this field. Value: 0 — Div1 Value: 1 — Div2 Value: 2 — Div4 Value: 3 — Div8	RW	0x2
9:8	l4mpclk	The external l4_mp_clk divider is specified in this field.	RW	0x1

continued...

Bit	Name	Description	Access	Reset
		Value / Description 0 – Div1 1 – Div2 2 – Div4 3 – Div8		
1:0	l4mainclk	The external l4_main_clk divider is specified in this field. Value / Description 0 – Div1 1 – Div2 2 – Div4 3 – Div8	RW	0x0

3.4.3. clkmgr_perpllgrp Address Map

Module Instance	Base Address	End Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD0413F

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Register	Offset	Width	Access	Reset Value	Description
vco0 on page 111	0x0	32	RW	0x10043	Peripheral PLL VCO Control Register 0
vco1 on page 112	0x4	32	RW	0x10001	Main PLL VCO Control Register
en on page 113	0x8	32	RW	0xF7F	Enable Register
ens on page 114	0xC	32	RW	0xF7F	Enable Set Register
enr on page 116	0x10	32	RW	0xF7F	Enable Reset Register

continued...

Register	Offset	Width	Access	Reset Value	Description
bypass on page 117	0x14	32	RW	0xFF	Bypass Register
bypasss on page 118	0x18	32	RW	0xFF	Bypass Set Register
bypassr on page 119	0x1C	32	RW	0xFF	Bypass Reset Register
cntr2clk on page 121	0x28	32	RW	0x0	Peripheral PLL Control Register for Counter 2 Clock
cntr3clk on page 122	0x2C	32	RW	0x0	Peripheral PLL Control Register for Counter 3 Clock
cntr4clk on page 123	0x30	32	RW	0x0	Peripheral PLL Control Register for Counter 4 Clock
cntr5clk on page 124	0x34	32	RW	0x0	Peripheral PLL Control Register for Counter 5 Clock
cntr6clk on page 125	0x38	32	RW	0x0	Peripheral PLL Control Register for Counter 6 Clock
cntr7clk on page 126	0x3C	32	RW	0x0	Peripheral PLL Control Register for Counter 7 Clock
cntr8clk on page 126	0x40	32	RW	0x0	Peripheral PLL Control Register for Counter 8 Clock
outrst on page 127	0x60	32	RW	0x0	Peripheral PLL Output Counter Reset Register
outrststat on page 128	0x64	32	RO	0x0	Peripheral PLL Output Counter Reset Ack Status Register
emacctl on page 129	0x68	32	RW	0x0	Main Divide Register
gpiodiv on page 130	0x6C	32	RW	0x1	GPIO Divide Register

3.4.3.1. clkmgr_perpllgrp Summary

Base Address: 0xFFD040C0

Register Address Offset	Bit Fields															
i_clk_mgr_perpllgrp																
vco0 0x0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved			bwadjen RW 0x0	bwadj RW 0x1											
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved						psrc RW 0x0		Reserved	saten RW 0x1	fasten RW 0x0	regextsel RW 0x0	outreseteall RW 0x0	en RW 0x0	pwrdn RW 0x1	bgpwrdn RW 0x1
vco1 0x4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved										denom RW 0x1					
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved			numer RW 0x1												
en 0x8	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				qspiclken RW 0x1	nandclken RW 0x1	spimclken RW 0x1	usbclken RW 0x1	Reserved	s2fuser1clken RW 0x1	sdmmcclken RW 0x1	gpiodben RW 0x1	emacptpen RW 0x1	emac2en RW 0x1	emac1en RW 0x1	emac0en RW 0x1
ens 0xC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				qspiclken RW 0x1	nandclken RW 0x1	spimclken RW 0x1	usbclken RW 0x1	Reserved	s2fuser1clken RW 0x1	sdmmcclken RW 0x1	gpiodben RW 0x1	emacptpen RW 0x1	emac2en RW 0x1	emac1en RW 0x1	emac0en RW 0x1
enr 0x10	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved				qspiclken RW 0x1	nandclken RW 0x1	spimclken RW 0x1	usbclken RW 0x1	Reserved	s2fuser1clken RW 0x1	sdmmcclken RW 0x1	gpiodben RW 0x1	emacptpen RW 0x1	emac2en RW 0x1	emac1en RW 0x1	emac0en RW 0x1
bypass 0x14	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															

continued...

Register Address Offset	Bit Fields															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								fben RW 0x1	rfen RW 0x1	s2fuser1 RW 0x1	sdmmc RW 0x1	gpiodb RW 0x1	emacptp RW 0x1	emacb RW 0x1	emaca RW 0x1
bypasss 0x18	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								fben RW 0x1	rfen RW 0x1	s2fuser1 RW 0x1	sdmmc RW 0x1	gpiodb RW 0x1	emacptp RW 0x1	emacb RW 0x1	emaca RW 0x1
bypassr 0x1C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved								fben RW 0x1	rfen RW 0x1	s2fuser1 RW 0x1	sdmmc RW 0x1	gpiodb RW 0x1	emacptp RW 0x1	emacb RW 0x1	emaca RW 0x1
cntr2clk 0x28	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved													src RW 0x0		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr3clk 0x2C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved													src RW 0x0		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr4clk 0x30	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved													src RW 0x0		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr5clk 0x34	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved													src RW 0x0		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr6clk	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

continued...

Register Address Offset	Bit Fields															
0x38	Reserved														src RW 0x0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr7clk 0x3C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
cntr8clk 0x40	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved														src RW 0x0	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					cnt RW 0x0										
outrst 0x60	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	outreset RW 0x0															
outrststat 0x64	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	outresetack RO 0x0															
emacctl 0x68	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved			emac 2sel RW 0x0	emac 1sel RW 0x0	emac 0sel RW 0x0	Reserved									
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved															
gpiodiv 0x6C	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	gpiodbclk RW 0x1															

3.4.3.2. vco0

Contains settings that control the Peripheral PLL VCO. VCO0 register contains signals required for PLL reset and power down.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040C0

Offset: 0x0

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			bwadjen RW 0x0	bwadj RW 0x1											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						psrc RW 0x0		Reserved	saten RW 0x1	fasten RW 0x0	regextsel RW 0x0	outresetall RW 0x0	en RW 0x0	pwrdn RW 0x1	bgpwrdn RW 0x1

vco0 Fields

Bit	Name	Description	Access	Reset
28	bwadjen	If set to 1, the Loop Bandwidth Adjust value comes from the Loop Bandwidth Adjust field. If set to 0, the Loop Bandwidth Adjust value equals the M field divided by 2 value of the VCO Control Register. The M divided by 2 is the upper 12 bits (12:1) of the M field in the VCO register.	RW	0x0
27:16	bwadj	Provides Loop Bandwidth Adjust value.	RW	0x1
9:8	psrc	Controls the VCO input clock source. Value 0: HPS_CLK1 (external oscillator input pin that becomes osc1_clk signal in device). Value 1: cb_intosc_ls_clk (internal low speed oscillator) Value 2: f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0

continued...

Bit	Name	Description	Access	Reset
		<table><tr><th>Value</th><th>Description</th></tr><tr><td>3</td><td>MAIN</td></tr></table>		
6	saten	Enables saturation behavior.	RW	0x1
5	fasten	Enables fast locking circuit.	RW	0x0
4	regextsel	If set to '1', the external regulator is selected for the PLL. If set to '0', the internal regulator is slected. It is strongly recommended to select the external regulator while the PLL is not enabled (in reset), and then disable the external regulater once the PLL becomes enabled. Software should simulateously update the 'Enable' bit and the 'External Regulator Input Select' in the same write access to the VCO register. When the 'Enable' bit is clear, the 'External Regulator Input Select' should be set, and vice versa. The reset value of this bit is applied on a cold reset; warm reset has no affect on this bit.	RW	0x0
3	outresetall	Before releasing Bypass, All Output Counter Reset must be set and cleared by software for correct clock operation. If '1', Reset phase multiplexer and all output counter state. So that after the assertion all the clocks output are start from rising edge align. If '0', phase multiplexer and output counter state not reset and no change to the phase of the clock outputs.	RW	0x0
2	en	If '1', VCO is enabled. If '0', VCO is in reset.	RW	0x0
1	pwrdn	If '1', power down analog circuitry. If '0', analog circuitry not powered down.	RW	0x1
0	bgpwrdn	If '1', powers down bandgap. If '0', bandgap is not power down.	RW	0x1

3.4.3.3. vco1

Contains settings that control the Peripheral PLL VCO. The VCO1 register contains the numerator and denominator counter settings.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040C4

Offset: 0x4

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										denom RW 0x1					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			numer RW 0x1												

vco1 Fields

Bit	Name	Description	Access	Reset
21:16	denom	Denominator in VCO output frequency equation. For incremental frequency change, if the new value lead to less than 20% of the frequency change, this value can be changed without resetting the PLL. The Numerator and Denominator can not be changed at the same time for incremental frequency changed.	RW	0x1
12:0	numer	Numerator in VCO output frequency equation. For incremental frequency change, if the new value lead to less than 20% of the frequency change, this value can be changed without resetting the PLL. The Numerator and Denominator can not be changed at the same time for incremental frequency changed.	RW	0x1

3.4.3.4. en

```
        Contains fields that control clock enables for clocks derived from the
Peripheral PLL.
1: The clock is enabled.
0: The clock is disabled.
```

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040C8

Offset: 0x8

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				qspiclken RW 0x1	nandclken RW 0x1	spimclken RW 0x1	usbclken RW 0x1	Reserved	s2fuser1clken RW 0x1	sdmmcclken RW 0x1	gpiodben RW 0x1	emacptpen RW 0x1	emac2en RW 0x1	emac1en RW 0x1	emac0en RW 0x1

en Fields

Bit	Name	Description	Access	Reset
11	qspiclken	Enables QSPI peripheral clock. This enable goes outside of the Clock Manger to the QSPI directly.	RW	0x1
10	nandclken	Enables NAND peripheral clock. This enable goes outside of the Clock Manger to the NAND directly.	RW	0x1
9	spimclken	Enables SPI Master peripheral clock. This enable goes outside of the Clock Manger to the SPIM directly.	RW	0x1
8	usbclken	Enables USB peripheral clock. This enable goes outside of the Clock Manger to the USB directly.	RW	0x1
6	s2fuser1clken	Enables clock s2f_user1_clk output	RW	0x1
5	sdmmcclken	Enables SDMMC peripheral clock. This enable goes outside of the Clock Manger to the SDMMC directly.	RW	0x1
4	gpiodben	Enables clock gpio_db_clk output	RW	0x1
3	emacptpen	Enables clock emac_ptp_clk output	RW	0x1
2	emac2en	Enables clock emac2_clk output	RW	0x1
1	emac1en	Enables clock emac1_clk output	RW	0x1
0	emac0en	Enables clock emac0_clk output	RW	0x1

3.4.3.5. ens

Write One to Set corresonding fields in Enable Register.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040CC

Offset: 0xC

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				qspiclken RW 0x1	nandclken RW 0x1	spimclken RW 0x1	usbclken RW 0x1	Reserved	s2fuser1clken RW 0x1	sdmmcclken RW 0x1	gpiodben RW 0x1	emacptpen RW 0x1	emac2en RW 0x1	emac1en RW 0x1	emac0en RW 0x1

ens Fields

Bit	Name	Description	Access	Reset
11	qspiclken	Enables QSPI peripheral clock. This enable goes outside of the Clock Manger to the QSPI directly.	RW	0x1
10	nandclken	Enables NAND peripheral clock. This enable goes outside of the Clock Manger to the NAND directly.	RW	0x1
9	spimclken	Enables SPI Master peripheral clock. This enable goes outside of the Clock Manger to the SPIM directly.	RW	0x1
8	usbclken	Enables USB peripheral clock. This enable goes outside of the Clock Manger to the USB directly.	RW	0x1
6	s2fuser1clken	Enables clock s2f_user1_clk output	RW	0x1
5	sdmmcclken	Enables SDMMC peripheral clock. This enable goes outside of the Clock Manger to the SDMMC directly.	RW	0x1
4	gpiodben	Enables clock gpio_db_clk output	RW	0x1
3	emacptpen	Enables clock emac_ptp_clk output	RW	0x1

continued...

Bit	Name	Description	Access	Reset
2	emac2en	Enables clock emac2_clk output	RW	0x1
1	emac1en	Enables clock emac1_clk output	RW	0x1
0	emac0en	Enables clock emac0_clk output	RW	0x1

3.4.3.6. enr

Write One to Clear corresponding fields in Enable Register.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040D0

Offset: 0x10

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				qspiclken RW 0x1	nandclken RW 0x1	spimclken RW 0x1	usbclken RW 0x1	Reserved	s2fuser1clken RW 0x1	sdmmcclken RW 0x1	gpiodben RW 0x1	emacptpen RW 0x1	emac2en RW 0x1	emac1en RW 0x1	emac0en RW 0x1

enr Fields

Bit	Name	Description	Access	Reset
11	qspiclken	Enables QSPI peripheral clock. This enable goes outside of the Clock Manger to the QSPI directly.	RW	0x1
10	nandclken	Enables NAND peripheral clock. This enable goes outside of the Clock Manger to the NAND directly.	RW	0x1
9	spimclken	Enables SPI Master peripheral clock. This enable goes outside of the Clock Manger to the SPIM directly.	RW	0x1

continued...

Bit	Name	Description	Access	Reset
8	usbclken	Enables USB peripheral clock. This enable goes outside of the Clock Manger to the USB directly.	RW	0x1
6	s2fuser1clken	Enables clock s2f_user1_clk output	RW	0x1
5	sdmmcclken	Enables SDMMC peripheral clock. This enable goes outside of the Clock Manger to the SDMMC directly.	RW	0x1
4	gpiodben	Enables clock gpio_db_clk output	RW	0x1
3	emacptpen	Enables clock emac_ptp_clk output	RW	0x1
2	emac2en	Enables clock emac2_clk output	RW	0x1
1	emac1en	Enables clock emac1_clk output	RW	0x1
0	emac0en	Enables clock emac0_clk output	RW	0x1

3.4.3.7. bypass

Contains fields that control bypass for clocks derived from the Peripheral PLL.
1: The clock is bypassed.
0: The clock is derived from the 5:1 active mux.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040D4

Offset: 0x14

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								fben RW 0x1	rfen RW 0x1	s2fuser1 RW 0x1	sdmmc RW 0x1	gpiodb RW 0x1	emacptp RW 0x1	emacb RW 0x1	emaca RW 0x1

bypass Fields

Bit	Name	Description	Access	Reset
7	fben	If set, the pll_main_fben_clk will be bypassed to the boot_clk. The pll_main_fben_clk is used to synchronously update the Numerator to the Main PLL.	RW	0x1
6	rfen	If set, the pll_peri_rfen_clk will be bypassed to the boot_clk. The pll_peri_rfen_clk is used to synchronously update the Denominator to the Peripheral PLL.	RW	0x1
5	s2fuser1	If set, the s2f_user1_clk will be bypassed to the boot_clk.	RW	0x1
4	sdmmc	If set, the sdmmc_clk will be bypassed to the boot_clk.	RW	0x1
3	gpiodb	If set, the gpio_db_clk will be bypassed to the boot_clk.	RW	0x1
2	emacptp	If set, the emac_ptp_clk will be bypassed to the boot_clk.	RW	0x1
1	emacb	If set, the emacb_free_clk will be bypassed to the boot_clk.	RW	0x1
0	emaca	If set, the emaca_free_clk will be bypassed to the boot_clk.	RW	0x1

3.4.3.8. bypasss

Write One to Set corresponding fields in Bypass Register.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040D8

Offset: 0x18

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								fben RW 0x1	rfen RW 0x1	s2fuser1 RW 0x1	sdmmc RW 0x1	gpiodb RW 0x1	emacptp RW 0x1	emacb RW 0x1	emaca RW 0x1

bypasss Fields

Bit	Name	Description	Access	Reset
7	fben	If set, the pll_main_fben_clk will be bypassed and the boot_clk will be used. The pll_main_fben_clk is used to synchronously update the Numerator to the Main PLL.	RW	0x1
6	rfen	If set, the pll_peri_rfen_clk will be bypassed and the boot_clk will be used. The pll_peri_rfen_clk is used to synchronously update the Denominator to the Peripheral PLL.	RW	0x1
5	s2fuser1	If set, the s2f_user1_clk will be bypassed and the boot_clk will be used.	RW	0x1
4	sdmmc	If set, the sdmmc_clk will be bypassed and the boot_clk will be used.	RW	0x1
3	gpiodb	If set, the gpio_db_clk will be bypassed and the boot_clk will be used.	RW	0x1
2	emacptp	If set, the emac_ptp_clk will be bypassed and the boot_clk will be used.	RW	0x1
1	emacb	If set, the emacb_free_clk will be bypassed and the boot_clk will be used.	RW	0x1
0	emaca	If set, the emaca_free_clk will be bypassed and the boot_clk will be used.	RW	0x1

3.4.3.9. bypassr

Write One to Clear corresponding fields in Bypass Register.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040DC

Offset: 0x1C

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								fben RW 0x1	rfen RW 0x1	s2fuser1 RW 0x1	sdmmc RW 0x1	gpiodb RW 0x1	emacptp RW 0x1	emacb RW 0x1	emaca RW 0x1

bypassr Fields

Bit	Name	Description	Access	Reset
7	fben	If set, the pll_main_fben_clk will be bypassed to the boot_clk. The pll_main_fben_clk is used to synchronously update the Numerator to the Main PLL.	RW	0x1
6	rfen	If set, the pll_peri_rfen_clk will be bypassed to the boot_clk. The pll_peri_rfen_clk is used to synchronously update the Denominator to the Peripheral PLL.	RW	0x1
5	s2fuser1	If set, the s2f_user1_clk will be bypassed to the input clock reference of the Peripheral PLL.	RW	0x1
4	sdmmc	If set, the sdmmc_clk will be bypassed to the input clock reference of the Peripheral PLL.	RW	0x1
3	gpiodb	If set, the gpio_db_clk will be bypassed to the input clock reference of the Peripheral PLL.	RW	0x1
2	emacptp	If set, the emac_ptp_clk will be bypassed to the input clock reference of the Peripheral PLL.	RW	0x1
1	emacb	If set, the emacb_free_clk will be bypassed to the input clock reference of the Peripheral PLL.	RW	0x1
0	emaca	If set, the emaca_free_clk will be bypassed to the input clock reference of the Peripheral PLL.	RW	0x1

3.4.3.10. cntr2clk

Contains settings that control Counter 2 clock generated from the Peripheral PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040E8

Offset: 0x28

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr2clk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value 0: pll0_emaca_base_clk Value 1: pll1_emaca_base_clk Value 2: osc1_clk (external oscillator input pin, HPS_CLK1) Value 3: cb_intosc_hs_div2_clk (internal high speed oscillator divided by 2) Value 4: f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.3.11. cntr3clk

```
        Contains settings that control Counter 3 clock generated from the
Peripheral PLL VCO clock.
```

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040EC

Offset: 0x2C

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr3clk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value / Description: 0 — pll0_emacb_base_clk 1 — pll1_emacb_base_clk 2 — osc1_clk (external oscillator input pin, HPS_CLK1) 3 — cb_intosc_hs_div2_clk (internal high speed oscillator divided by 2) 4 — f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.3.12. cntr4clk

Contains settings that control Counter 4 clock generated from the Peripheral PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040F0

Offset: 0x30

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr4clk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value: 0 — pll0_emac_ptp_clk Value: 1 — pll1_emac_ptp_clk Value: 2 — osc1_clk (external oscillator input pin, HPS_CLK1) Value: 3 — cb_intosc_hs_div2_clk (internal high speed oscillator divided by 2) Value: 4 — f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.3.13. cntr5clk

```
        Contains settings that control Counter 5 clock generated from the
Peripheral PLL VCO clock.
```

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040F4

Offset: 0x34

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr5clk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value / Description: 0 — pll0_gpio_clk 1 — pll1_gpio_clk 2 — osc1_clk (external oscillator input pin, HPS_CLK1) 3 — cb_intosc_hs_div2_clk (internal high speed oscillator divided by 2) 4 — f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.3.14. cntr6clk

Contains settings that control Counter 6 clock generated from the Peripheral PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040F8

Offset: 0x38

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr6clk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value / Description: 0 — pll0_sdmmc_clk 1 — pll1_sdmmc_clk 2 — osc1_clk (external oscillator input pin, HPS_CLK1) 3 — cb_intosc_hs_div2_clk (internal high speed oscillator divided by 2) 4 — f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.3.15. cntr7clk

Contains settings that control Counter 7 clock generated from the Peripheral PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD040FC

Offset: 0x3C

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr7clk Fields

Bit	Name	Description	Access	Reset
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.3.16. cntr8clk

Contains settings that control Counter 8 clock generated from the Peripheral PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD04100

Offset: 0x40

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													src RW 0x0		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					cnt RW 0x0										

cntr8clk Fields

Bit	Name	Description	Access	Reset
18:16	src	Selects the source for the active 5:1 clock selection when the PLL is not bypassed. Value / Description: 0 — pll0_h2f_user1_clk 1 — pll1_h2f_user1_clk 2 — osc1_clk (external oscillator input pin, HPS_CLK1) 3 — cb_intosc_hs_div2_clk (internal high speed oscillator divided by 2) 4 — f2h_free_clk (FPGA fabric PLL clock reference)	RW	0x0
10:0	cnt	Divides the VCO frequency by the value+1 in this field.	RW	0x0

3.4.3.17. outrst

Contains settings to assert individual Outreset for all Peripheral PLL Counters.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD04120

Offset: 0x60

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
outreset RW 0x0															

outrst Fields

Bit	Name	Description	Access	Reset
15:0	outreset	Resets the individual PLL output counter. For software to change the PLL output counter without producing glitches on the respective clock, SW must set the Output Counter Reset Register 'Output Counter Reset' bit. Software then polls the respective Output Counter Reset Acknowledge bit in the Output Counter Reset Ack Status Register. Software then writes the appropriate counter register, and then clears the respective Output Counter Reset bit. LSB 'outreset[0]' corresponds to PLL output clock C0, etc. If set to '1', reset output divider, no clock output from counter. If set to '0', counter is not reset. The reset value of this bit is applied on a cold reset; warm reset has no affect on this bit.	RW	0x0

3.4.3.18. outrststat

Contains Output Clock Counter Reset acknowledge status.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD04124

Offset: 0x64

Access: R0

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
outresetack RO 0x0															

outrststat Fields

Bit	Name	Description	Access	Reset
15:0	outresetack	These read only bits per PLL output indicate that the PLL has received the Output Reset Counter request and has gracefully stopped the respective PLL output clock. For software to change the PLL output counter without producing glitches on the respective clock, SW must set the Output Counter Reset Register 'Output Counter Reset' bit. Software then polls the respective Output Counter Reset Acknowledge bit in the Output Counter Reset Ack Status Register. Software then writes the appropriate counter register, and then clears the respective Output Counter Reset bit. The reset value of this bit is applied on a cold reset; warm reset has no affect on this bit. **Value / Description** 0 — IDLE 1 — ACK_RECEIVED	RO	0x0

3.4.3.19. emacctl

Contains fields that control clock dividers for main clocks derived from the Main PLL

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD04128

Offset: 0x68

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			emac2 sel RW 0x0	emac1 sel RW 0x0	emac0 sel RW 0x0	Reserved									
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															

emacctl Fields

Bit	Name	Description	Access	Reset
28	emac2sel	Selects the source for emac2_clk as either emaca_free_clk or emacb_free_clk. Value: 0 — EMACA Value: 1 — EMACB	RW	0x0
27	emac1sel	Selects the source for emac1_clk as either emaca_free_clk or emacb_free_clk. Value: 0 — EMACA Value: 1 — EMACB	RW	0x0
26	emac0sel	Selects the source for emac0_clk as either emaca_free_clk or emacb_free_clk. Value: 0 — EMACA Value: 1 — EMACB	RW	0x0

3.4.3.20. gpiodiv

Contains a field that controls the clock divider for the GPIO De-bounce clock.

Module Instance	Base Address	Register Address
i_clk_mgr_perpllgrp	0xFFD040C0	0xFFD0412C

Offset: 0x6C

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
gpiodbclk RW 0x1															

gpiodiv Fields

Bit	Name	Description	Access	Reset
15:0	gpiodbclk	The gpio_db_clk is divided down from the periph_base_clk by the value plus one specified in this field. The value 0 (divide by 1) is illegal. A value of 1 indicates divide by 2, 2 divide by 3, etc.	RW	0x1

3.4.4. clkmgr_alteragrp Address Map

Module Instance	Base Address	End Address
i_clk_mgr_alteragrp	0xFFD04140	0xFFD04FFF

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Register	Offset	Width	Access	Reset Value	Description
mpuclk on page 132	0x0	32	RW	0x10001	MPU Internal PLL Counters
nocclk on page 133	0x4	32	RW	0x30003	NOC Internal PLL Counters

3.4.4.1. clkmgr_alteragrp Summary

Base Address: 0xFFD04140

Register Address Offset	Bit Fields															
i_clk_mgr_alteragrp																
mpuclk 0x0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved					pericnt RW 0x1										
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					maincnt										

continued...

Register Address Offset	Bit Fields															
						RW 0x1										
nocclk 0x4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Reserved					pericnt RW 0x3										
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved					maincnt RW 0x3										

3.4.4.2. mpuclk

Contains internal counter setting for MPU Clock groups for both PLLs.

Module Instance	Base Address	Register Address
i_clk_mgr_alteragrp	0xFFD04140	0xFFD04140

Offset: 0x0

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					pericnt RW 0x1										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					maincnt RW 0x1										

mpuclk Fields

Bit	Name	Description	Access	Reset
26:16	pericnt	Divides the VCO frequency by the value+1 in this field. This field loads the internal counter in the Main PLL for the Main Clock Group. *Note:* Intel recommends that you do not alter the value of this register. Changing the value at run time can cause the system to become unstable.	RW	0x1
10:0	maincnt	Divides the VCO frequency by the value+1 in this field. This field loads the internal counter in the Main PLL for the Main Clock Group.	RW	0x1

continued...

Bit	Name	Description	Access	Reset
		Note: Intel recommends that you do not alter the value of this register. Changing the value at run time can cause the system to become unstable.		

3.4.4.3. nocclk

Contains settings that control clock main_clk generated from the Main PLL VCO clock.

Module Instance	Base Address	Register Address
i_clk_mgr_alteragrp	0xFFD04140	0xFFD04144

Offset: 0x4

Access: RW

Important: The value of a reserved bit must be maintained in software. When you modify registers containing reserved bit fields, you must use a read-modify-write operation to preserve state and prevent indeterminate system behavior.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					pericnt RW 0x3										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					maincnt RW 0x3										

nocclk Fields

Bit	Name	Description	Access	Reset
26:16	pericnt	Divides the VCO frequency by the value+1 in this field. This field loads the internal counter in the NOC PLL for the NOC Clock Group.	RW	0x3
10:0	maincnt	Divides the VCO frequency by the value+1 in this field. This field loads the internal counter in the NOC PLL for the NOC Clock Group.	RW	0x3

4. Reset Manager

The reset manager generates module reset signals based on reset requests from the various sources in the HPS and FPGA fabric, and software writing to the module-reset control registers. The reset manager ensures that a reset request from the FPGA fabric can occur only after the FPGA portion of the system-on-a-chip (SoC) device is configured.

The HPS contains multiple reset domains. Each reset domain can be reset independently. A reset may be initiated externally, internally or through software.

The Reset domains and sequences include security features. The security manager works with Power On Reset (POR) and brings the reset manager out of reset only when the secure fuses have been loaded and validated. Once this process is complete, the reset manager brings the rest of the HPS out of reset.

Table 17. HPS Reset Domains

Domain Name	Domain Logic	Module Master	Description
POR	Power on Reset. The entire HPS is reset.	Security Manager	After POR, the security manager comes out of reset and validates the security fuses. After validation, the reset manager is released from reset and proceeds to perform a cold reset on the HPS.
System Cold	Cold Reset. All of the HPS except security manager and POR fuse logic.	Reset Manager	Using the known security state, the HPS is placed in the default state, allowing software to boot. Cold reset is triggered by POR as well as other sources.
System Warm	System Warm (SW) is a warm reset. All of the HPS except security manager, POR, Fuse Logic, and test access port (TAP), and Debug domains are reset.	Reset Manager	Used to recover system from a non-responsive condition. Resets a subset of the HPS state reset by a cold reset. Only affects the system reset domain, which allows debugging (including trace) to operate through the warm reset. It is possible to mask a warm reset for some modules such that they are not affected.

continued...

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Domain Name	Domain Logic	Module Master	Description
Debug	All debug logic including most of the DAP, CoreSight™ components connected to the debug peripheral bus, trace, the microprocessor unit (MPU) subsystem, and the FPGA fabric.	Reset Manager	May be asserted by cold reset, the debugger or software. Used to recover debug logic from a non-responsive condition.
TAP	JTAG test access port (TAP) controller, which is used by the debug access port (DAP).	Reset Manager	Asserted by cold reset or by Software.
RAM Clear	The on-chip memories are cleared	Reset Manager	Memories may be cleared on cold or warm as indicated by the corresponding bits in the `ramstat` register.

The HPS supports the following reset types:

- System cold reset
  - Used to ensure the HPS is placed in a default state sufficient for software to boot
  - Triggered by a power-on reset and other sources
  - Resets all HPS logic that can be reset
  - Affects all reset domains
- System warm reset
  - Used to recover system from a non-responsive condition
  - Does not reset the Debug or TAP reset domain, allowing debug functions including trace to operate through out a warm reset
  - Masks allow software to exclude modules from warm reset. Exceptions for masks:
    - To maintain security, there is no mask for security manager
    - The MPU cannot be masked
  - RAM Clearing domain (if not masked) logic is reset
- Debug reset
  - Used to recover debug logic from a non-responsive condition
  - Only affects the debug reset domain
- TAP Reset
  - JTAG TAP controller is reset
  - Software may trigger TAP reset

Related Information

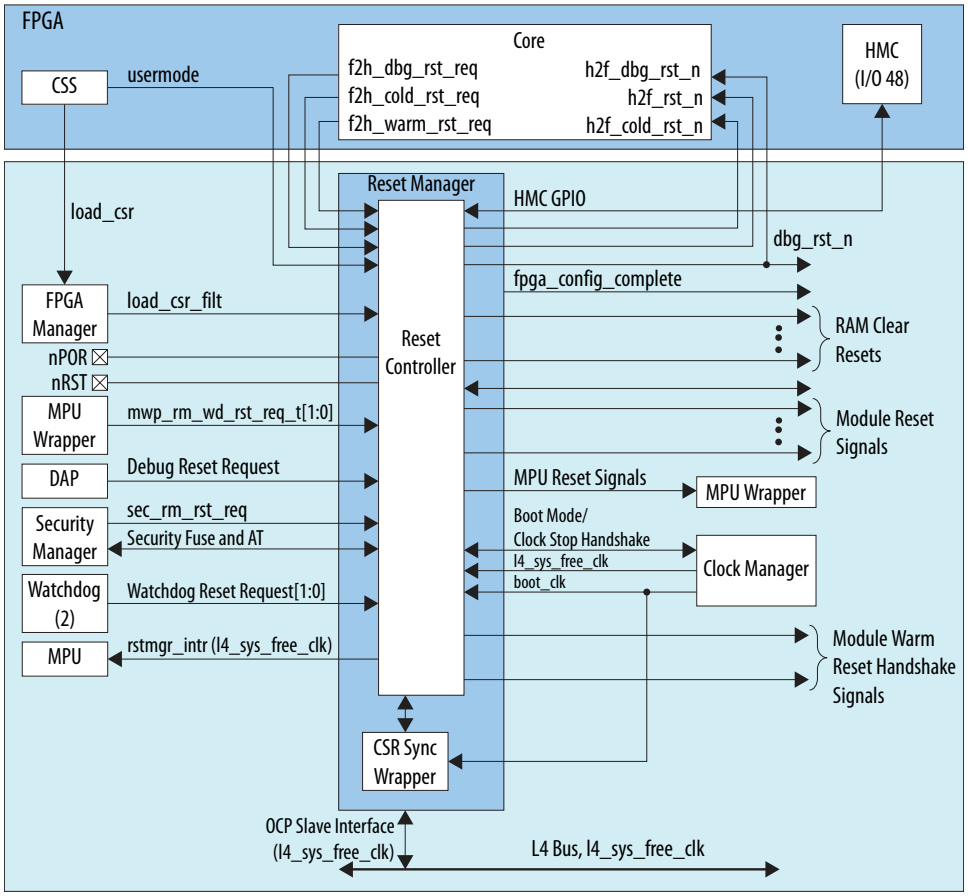
- SoC Security on page 168
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

4.1. Reset Manager Block Diagram and System Integration

The reset manager accepts reset requests from the security manager, FPGA control block, FPGA core, modules in the HPS, and reset pins and generates module reset signals. The reset manager receives security status from the security manager. The reset manager also has warm reset handshaking signals to support system reset behavior, clock manager interfaces, and the Anti-Tamper interface with security manager. The Boot clock (`boot_clk`) is used as the default clock for both cold or warm reset. When the device is configured to operate in secure mode, `boot_clk` is `cb_intosc_hs_div2_clk`. When the device is not in secure mode, `boot_clk` is sourced from the external oscillator input pin, `HPS_CLK1`.

The following figure shows a block diagram of the reset manager in the SoC device. For clarity, reset-related handshaking signals to other HPS modules and to the clock manager module are omitted.

Figure 9. Reset Manager Block Diagram



4.1.1. Reset Controller

In secure mode, all signals are synchronous to `boot_clk`. In non-secure mode, all signals are synchronous to the clock signal `osc1_clk` which is sourced from the external oscillator input pin, HPS_CLK1. The following table lists the reset sources external to the HPS.

Table 18. HPS External Reset Sources

Source	Description
`f2h_cold_rst_req_n`	Cold reset request from FPGA fabric (active low)
`f2h_warm_rst_req_n`	Warm reset request from FPGA fabric (active low)
`f2h_dbg_rst_req_n`	Debug reset request from FPGA fabric (active low)
`load_csr_filt`	Cold-only reset from FPGA control block (CB)
nPOR	Power-on reset pin (active low)
nRST	Warm reset pin (active low)

Table 19. HPS External Reset Outputs

Source	Description
`h2f_cold_rst_n`	Cold-only reset to FPGA fabric (active low)
`h2f_rst_n`	Cold or warm reset to FPGA fabric (active low)
`h2f_dbg_rst_n`	Debug reset (`dbg_rst_n`) to FPGA fabric (active low)

The reset controller performs the following functions:

- Accepts reset requests from the FPGA control block (CB), FPGA fabric, modules in the HPS, and reset pins
- Generates an individual reset signal for each module instance for all modules in the HPS
- Provides reset handshaking signals to support system reset behavior

The reset controller generates module reset signals from external reset requests and internal reset requests. External reset requests originate from sources external to the reset manager. Internal reset requests originate from control registers in the reset manager.

The reset controller supports the following cold reset requests:

- Security manager reset
- Cold reset request pin (nPOR)
- FPGA fabric
- FPGA CB
- Software cold reset request bit (`swcoldrstreq`) of the control register (`ctrl`)

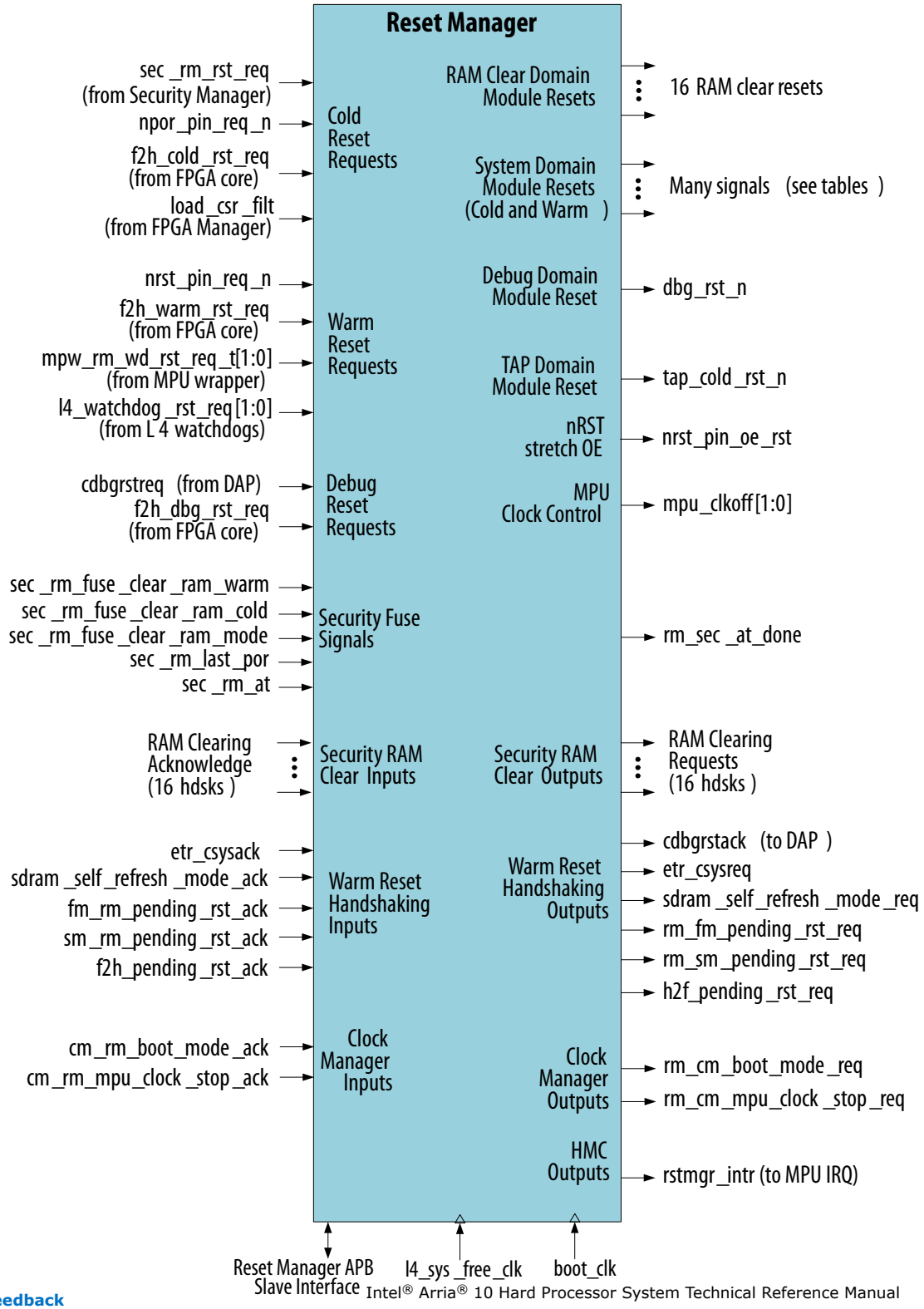
The reset controller supports the following warm reset requests:

- Warm reset request pin (nRST)
- FPGA fabric
- Software warm reset request bit (`swwarmrstreq`) of the `ctrl` register
- MPU watchdog reset requests for CPU0 and CPU1
- System watchdog timer 0 and 1 reset requests

The reset controller supports the following debug reset requests:

- CDBGRSTREQ from DAP
- FPGA fabric

Figure 10. Reset Controller Signals



4.1.2. Security Reset Functions

When secure mode is enabled on the device, reset manager performs some additional functions. To avoid unsecure snooping of existing RAM contents, the HPS can be configured to clear all RAM, both general-purpose RAM and dedicated RAM that is integrated into modules. The list of RAMs on the HPS is:

- Onchip RAM
- USB0
- USB1
- SDMMC
- EMAC0 - EMAC2 RX and TX buffers
- DMA
- NAND Read, Write and ECC RAMs
- QSPI
- MPU RAM

RAMs can be cleared on cold reset, warm reset or both as there are separate fuses that control behavior for cold and warm resets. When the corresponding fuse is blown, all RAMs are cleared for the indicated reset type. If an additional fuse is blown, the various RAM contents are cleared sequentially; otherwise, the contents of all of the RAMs are cleared simultaneously. Even if all security RAM clearing is disabled, the reset manager always invalidates the MPU L1 Cache.

The HPS also supports an Anti-Tamper interface from the FPGA fabric to security manager. If an Anti-Tamper event occurs, the HPS clears all RAMs and becomes unresponsive. The HPS only recovers from this state by a POR. Anti-Tamper logic, such as voltage detection, may reside in the FPGA.

4.1.3. Module Reset Signals

The following tables list the module reset signals. The module reset signals are organized in groups for the MPU, peripherals, bridges.

In the following tables, columns marked for Cold Reset, Warm Reset, and Debug Reset denote reset signals asserted by each type of reset. For example, writing a 1 to the `swwarmrstreq` bit in the `ctrl` register resets all the modules that have a checkmark in the Warm Reset column.

Note: For warm resets, software can set the warmmask registers to prevent the assertion of module reset signals to peripheral modules.

The column marked for Software Deassert denotes reset signals that are left asserted by the reset manager.

Table 20. MPU Group, Generated Module Resets

Module Reset Signal	Description	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
mpu_cpu_rst_n[0]	Resets each processor in the MPU	System	X	X		
mpu_cpu_rst_n[1]	Resets each processor in the MPU	System	X	X		X
mpu_wd_rst_n	Resets both per-processor watchdogs in the MPU	System	X	X		
mpu_scu_periph_rst_n	Resets Snoop Control Unit (SCU) and peripherals	System	X	X		

Table 21. PER1 Group, Generated Module Resets

Module Reset Signal	Description	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
watchdog_rst_n[1:0]	Resets corresponding system watchdog timer	System	X	X		X
l4sys_timer_rst_n[1:0]	Resets corresponding OSC1 timer	System	X	X		X
sp_timer_rst_n[1:0]	Resets corresponding SP timer	System	X	X		X
i2c_rst_n[4:0]	Resets corresponding I^2C controller	System	X	X		X
uart_rst_n[1:0]	Resets corresponding UART	System	X	X		X
gpio_rst_n[2:0]	Resets corresponding GPIO interface	System	X	X		X
dma_periph_if_rst_n[7:0]	DMA controller request interface from FPGA fabric to DMA controller	System	X	X		X
emac_ptp_rst_n [1:0]	Resets corresponding EMAC precision time protocol	System	X	X		
emac_ecc_rst_n [2:0]	Resets corresponding to EMAC ECC	System	X	X		
usb_ecc_rst_n [1:0]	Resets corresponding to USB ECC	System	X	X		
nand_flash_ecc_rst_n	Resets corresponding to NAND Flash ECC	System	X	X		
qspi_flash_ecc_rst_n	Resets corresponding to QSPI Flash ECC	System	X	X		
sdmmc_ecc_rst_n	Resets corresponding to SDMMC ECC	System	X	X		

Table 22. PER0 Group, Generated Module Resets

Module Reset Signal	Description	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
emac_rst_n[2:0]	Resets corresponding EMAC	System	X	X		X
usb_rst_n[1:0]	Resets corresponding USB	System	X	X		X
nand_flash_rst_n	Resets NAND flash controller	System	X	X		X
qspi_flash_rst_n	Resets quad SPI flash controller	System	X	X		X
spim_rst_n[1:0]	Resets SPI master controller	System	X	X		X
spis_rst_n[1:0]	Resets SPI slave controller	System	X	X		X
sdmmc_rst_n	Resets SD/MMC controller	System	X	X		X
dma_rst_n	Resets DMA controller	System	X	X		X
dma_ecc_rst_n	Resets DMA ECC	System	X	X		

Table 23. Bridge Group, Generated Module Resets

Module Reset Signal	Description	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
hps2fpga_bridge_rst_n	Resets HPS-to-FPGA AMBA Advanced eXtensible Interface (AXI) bridge	System	X	X		X
fpga2hps_bridge_rst_n	Resets FPGA-to-HPS AXI bridge	System	X	X		X
lwhps2fpga_bridge_rst_n	Resets lightweight HPS-to-FPGA AXI bridge	System	X	X		X
f2h_sdram_bridge0_rst_n	Resets SDRAM bridge 0	System	X	X		X
f2h_sdram_bridge1_rst_n	Resets SDRAM bridge 1	System	X	X		X
f2h_sdram_bridge2_rst_n	Resets SDRAM bridge 2	System	X	X		X
ddr_scheduler_rst_n	Resets DDR scheduler	System	X	X		X

Table 24. MISC Group, Generated Module Resets

Group	Module Reset Signal	Description	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
SYSMOD	boot_rom_rst_n	Resets boot ROM	System	X	X		
	onchip_ram_rst_nsy	Resets on-chip RAM	System	X	X		

continued...

Group	Module Reset Signal	Description	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
	sys_manager_rst_n	Resets system manager (resets logic associated with cold or warm reset)	System	X			
	fpga_manager_rst_n	Resets FPGA manager	System	X	X		
	sys_dbg_rst_n	Resets debug masters and slaves connected to L3 interconnect and level 4 (L4) buses	System	X	X		
	onchip_ram_ecc_rst_n	Onchip RAM ECC reset	System	X	X		
	h2f_rst_n		System	X	X		
	sec_rst_n	Security reset	System	X			
COLD	sys_manager_cold_rst_n	Resets system manager (resets logic associated with cold reset only)	System	X			
	rst_pin_oe_rst	Pulls nRST pin low	System	X	X		
	timestamp_cold_rst_n	Resets debug timestamp to 0x0	System	X			
	clk_manager_cold_rst_n	Resets clock manager (resets logic associated with cold reset only)	System	X			
	tap_cold_rst_n	Resets portion of TAP controller in the DAP that must be reset on a cold reset	TAP	X			
	sec_cold_rst_n	Security cold reset	System	X			
	hmc_cold_rst_n	HMC cold reset	System	X	X		
	io_manager_cold_rst_n	I/O manager cold reset	System	X	X		
DBUG	dbg_rst_n	Resets debug components including DAP, trace, MPU debug logic, and	Debug	X		X	

continued...

Group	Module Reset Signal	Description	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
		any user debug logic in the FPGA fabric					
L3	l3_rst_n	Resets L3 Interconnect and L4 buses	System	X	X		

Table 25. RAM Clear Group, Generated Module Resets

Module Reset Signal	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
onchip_sec_ram_rst_n	X	X	X		
otg0_sec_ram_rst_n		X	X		
otg1_sec_ram_rst_n		X	X		
sdmmc_sec_ram_rst_n		X	X		
emac0rx_sec_ram_rst_n		X	X		
emac0tx_sec_ram_rst_n		X	X		
emac1rx_sec_ram_rst_n		X	X		
emac1tx_sec_ram_rst_n		X	X		
emac2rx_sec_ram_rst_n		X	X		
emac2tx_sec_ram_rst_n		X	X		
dma_sec_ram_rst_n		X	X		
nandw_sec_ram_rst_n		X	X		
nandr_sec_ram_rst_n		X	X		
nande_sec_ram_rst_n		X	X		
qspi_sec_ram_rst_n		X	X		
mwp_sec_ram_rst_n		X	X		

Table 26. L3 Group, Generated Module Resets

Module Reset Signal	Description	Reset Domain	Cold Reset	Warm Reset	Debug Reset	Software Deassert
misc	Resets L3 interconnect and L4 buses	System	X	X		

4.1.3.1. Modules Requiring Software Deassert

The reset manager leaves the reset signal asserted on certain modules even after the rest of the HPS has come out of reset. These modules are likely to require software configuration before they can safely be taken out of reset.

When a module that has been held in reset is ready to start running, software can deassert the reset signal by writing to the appropriate register, shown in the following table.

Table 27. Module Reset Signals and Registers

HPS Peripheral	Reset Register	Module Reset Signal	Reset Group
MPU	mpumodrst	mpu_cpu_rst_n[1]	MPU
Watchdog	per1modrst	watchdog_rst_n[1:0]	PER1
Timer	per1modrst	l4sys_timer_rst_n[1:0]	PER1
Timer	per1modrst	sp_timer_rst_n[1:0]	PER1
I^2C	per1modrst	i2c_rst_n[4:0]	PER1
UART	per1modrst	uart_rst_n[1:0]	PER1
GPIO	per1modrst	gpio_rst_n[2:0]	PER1
DMA	per1modrst	dma_periph_if_rst_n[7:0]	PER1
Ethernet MAC	per0modrst	emac_rst_n[2:0]	PER0
USB 2.0 OTG	per0modrst	usb_rst_n[1:0]	PER0
NAND	per0modrst	nand_flash_rst_n	PER0
Quad SPI	per0modrst	qspi_flash_rst_n	PER0
SPI Master	per0modrst	spim_rst_n[1:0]	PER0
SPI Slave	per0modrst	spis_rst_n[1:0]	PER0
SD/MMC	per0modrst	sdmmc_rst_n	PER0
DMA	per0modrst	dma_rst_n	PER0
HPS-to-FPGA Bridge	brgmodrst	hps2fpga_bridge_rst_n	Bridge
FPGA-to-HPS Bridge	brgmodrst	fpga2hps_bridge_rst_n	Bridge
Lightweight HPS-to-FPGA Bridge	brgmodrst	lwhps2fpga_bridge_rst_n	Bridge
FPGA-to-SDRAM port 0	brgmodrst	f2s_sdram_bridge0_rst_n	Bridge
FPGA-to-SDRAM port 1	brgmodrst	f2s_sdram_bridge1_rst_n	Bridge
FPGA-to-SDRAM port 2	brgmodrst	f2s_sdram_bridge2_rst_n	Bridge
SDRAM Scheduler	brgmodrst	ddr_scheduler_rst_n	Bridge

4.1.4. Slave Interface and Status Register

The reset manager slave interface is used to control and monitor the reset states.

The status register (`stat`) in the reset manager contains the status of the reset requester. The register contains a bit for each monitored reset request. The `stat` register captures all reset requests that have occurred. Software is responsible for clearing the bits.

During the boot process, the Boot ROM copies the `stat` register value into memory before clearing it. After booting, you can read the value of the reset status register at memory address (`r0` + 0x0038). For more information, refer to the "HPS State on Entry to the Second-Stage Boot Loader" section of the *Booting and Configuration* appendix.

Related Information

HPS State on Entry to the Second-Stage Boot Loader on page 810

4.2. Functional Description of the Reset Manager

The reset manager generates reset signals to modules in the HPS and to the FPGA fabric. The following actions generate reset signals:

- Software writing a 1 to the `swcoldrstreq` or `swwarmrstreq` bits in the `ctrl` register. Writing either bit causes the reset controller to perform a reset sequence.
- Software writing to the `mpumodrst`, `per0modrst`, `per1modrst`, `brgmodrst`, `sysmodrst`, `coldmodrst`, `nrstmodrst` or `dbgmodrst` module reset control registers.
- Asserting reset request signals triggers the reset controller. All external reset requests cause the reset controller to perform a reset sequence.

Multiple reset requests can be driven to the reset manager at the same time. Cold reset requests take priority over warm and debug reset requests. Higher priority reset requests preempt lower priority reset requests. There is no priority difference among reset requests within the same domain.

If a cold reset request is issued while another cold reset is already underway, the reset manager extends the reset period for all the module reset outputs until all cold reset requests are removed. If a cold reset request is issued while the reset manager is removing other modules out of the reset state, the reset manager returns those modules back to the reset state.

If a warm reset request is issued while another warm reset is already underway, the first warm reset completes before the second warm reset begins. If the second warm reset request is removed before the first warm reset completes, the warm first reset is extended to meet the timing requirements of the second warm reset request.

The nPOR pin can be used to extend the cold reset beyond what the POR voltage monitor automatically provides. The use of the nPOR pin is optional and can be tied high when it is not required.

The nRST pin can be used to extend a warm reset. The nRST pin is tri-stated, and the Reset Manager stretches a warm reset by the count programed in the CSR (RSTMGR.COUNTS.NRSTCNT) [doc however you usually do registers and bits] After the count has elapsed, 256 additional clocks cycles elapse before the nRST input is sampled again.

The reset manager contains the stat register that indicates which reset source caused a reset as well as the `ramstat` register that indicates which RAM modules were cleared during the last reset. After a cold reset is complete, all bits are cleared except for the bit(s) that indicate the source of the cold reset. If multiple cold reset requests

overlap with each other, the bit corresponding to the source that de-asserts its request last is set. If more than one source is de-asserted in the same cycle, the value of the bit corresponding to each source is set.

After a warm reset is complete, the bit(s) that indicate the source of the warm reset are set to 1. A warm reset doesn't clear any bits in the stat register, so bits corresponding to any reset source that has caused a warm reset since the last cold reset or since the bits were last cleared are set. Any bit can be manually cleared by writing a 1 to it.

If RAM memory is cleared during a Cold or Warm reset, then the `ramstat` register indicates which RAMs during the reset. Bits are cleared manually by writing a 1.

The hard memory controller (HMC) in the FPGA is reset by the reset manager using a GPIO. The HMC is reset only through software; resetting the HMC is not part of the operation of any reset sources.

Related Information

Intel Intel Arria 10 Device Datasheet

4.2.1. Reset Sequencing

The reset controller sequences resets without software assistance. Module reset signals are asserted asynchronously and synchronously. The reset manager deasserts the module reset signals synchronous to the `boot_clk` clock. Module reset signals are deasserted in groups in a fixed sequence. All module reset signals in a group are deasserted at the same time.

The reset manager sends a request to the clock manager to put the clocks in boot mode, which creates a fixed and known relationship between the `boot_clk` clock and all other clocks generated by the clock manager.

After the reset manager releases the MPU subsystem from reset, CPU1 is left in reset and CPU0 begins executing code from the reset vector address. Software is responsible for deasserting CPU1 and other resets, as shown in MPU Group, Generated Module Resets Table. Software deasserts resets by writing the `mpumodrst`, `per0modrst`, `per1modrst`,`brgmodrst`, `sysmodrst`, `coldmodrst`, `nrstmodrst`, and `dbgmodrst` module-reset control registers.

Software can also bypass the reset controller and generate reset signals directly through the module-reset control registers. In this case, software is responsible for asserting module reset signals, driving them for the appropriate duration, and deasserting them in the correct order. The clock manager is not typically in boot mode during this time, so software is responsible for knowing the relationship between the clocks generated by the clock manager. Software must not assert a module reset signal that would prevent software from deasserting the module reset signal. For example, software should not assert the module reset to the processor executing the software.

Table 28. Minimum Pulse Width

Reset Type	Value
Warm Reset	200 ns
Cold Reset	6 `osc1_clk` cycles

Note: The `osc1_clk` clock signal is sourced from the external oscillator input, `HPS_CLK1`.

Figure 11. Cold Reset Timing Diagram

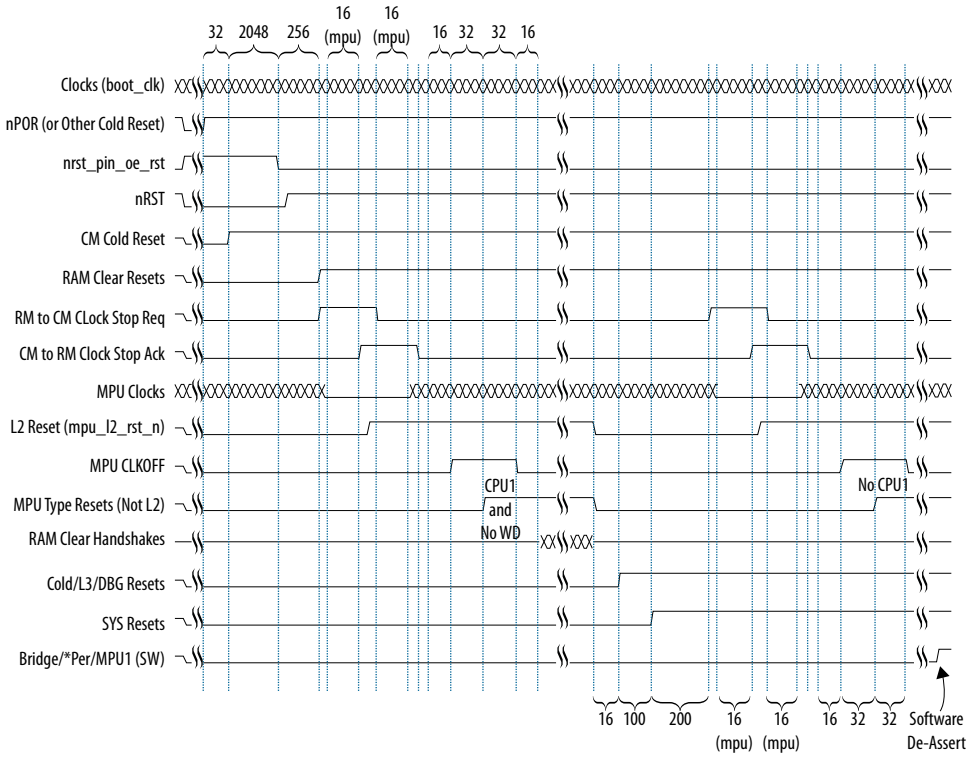
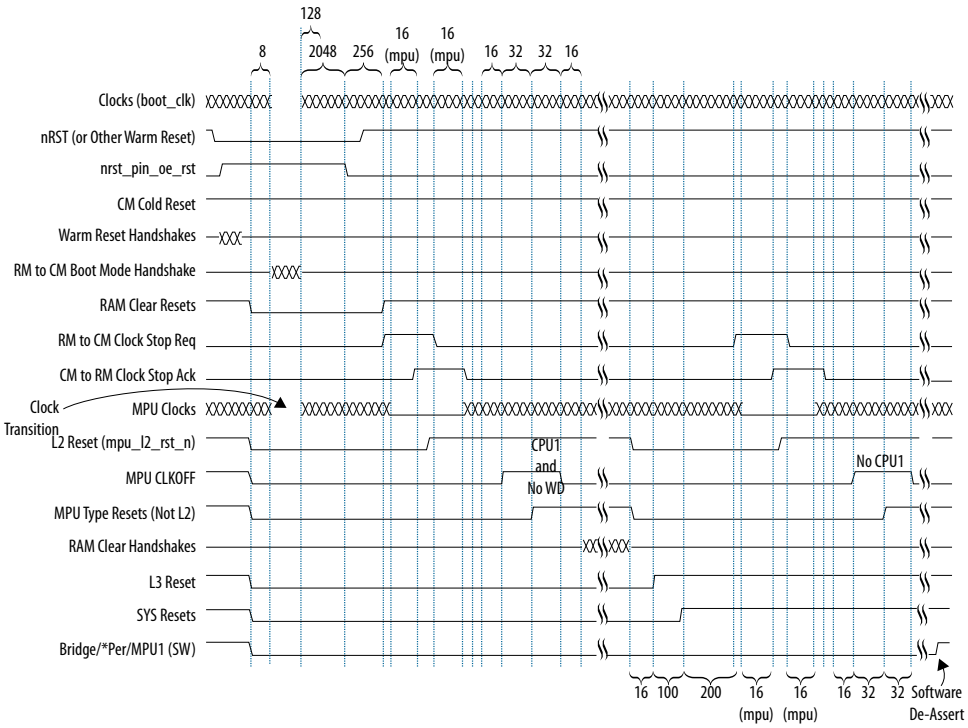


Figure 12. Warm Reset Timing Diagram



The cold and warm reset sequences consist of different reset assertion sequences and the same deassertion sequence. The following sections describe the sequences.

Note: Cold and warm reset affect only the `cpu0`, and by default cpu1 is held in reset until the software running in the `cpu0` releases it.

Related Information

- Module Reset Signals on page 140
- Clock Manager on page 52
  For more information about safe mode, refer to the *Clock Manager* chapter.

4.2.1.1. Cold Reset Assertion Sequence

The following list describes the assertion steps for cold reset shown in the Cold Reset timing diagram:

1. Assert all module resets.
2. Wait for level cold reset requests to de-assert
3. Wait for 32 cycles, de-asserts clock manager cold reset
4. Wait the nRST count (default is 2048). De-assert NRST Output Enable.
5. Wait 256 clocks to allow the nRST pin to stabilize. Start sampling nRST input pin.
6. Wait for level warm reset requests to all de-assert.
7. Go to de-assertion sequence.

Related Information

Cold and Warm Reset Deassertion Sequence on page 150

4.2.1.2. Warm Reset Assertion Sequence

The following list describes the assertion steps for warm reset shown in the Warm Reset Timing Diagram:

1. Reset manager performs optional handshake configured by SW with debug ETR. Wait for acknowledge.
2. Reset manager performs optional handshake configured by SW handshake with FPGA core. Wait for acknowledge.
3. Reset manager performs optional handshakes configured by SW with SDRAM and FPGA manager. Wait for acknowledges.
4. Reset manager asserts module resets except for MPU watchdogs if the watchdogs were the source of the warm reset.
5. Wait for 8 cycles, then reset manger sends `rm_cm_boot_mode_req` to clock manager. Wait until the Clock Manager acknowledges.
6. Start nRST count if non-zero and a fixed counter to 128 (COUNTS.WARMRSCYCLES).
7. If nRST count non-zero, start 256 stretch counter after nRST count is done to allow the nRST pin to stabilize. After 256 clocks, sample input pin nRST.
8. Wait for level warm reset requests to de-assert, the nRST count to be zero and the fixed 128 count to be zero.
9. Go to de-assertion sequence.

Related Information

Cold and Warm Reset Deassertion Sequence on page 150

4.2.1.3. Cold and Warm Reset Deassertion Sequence

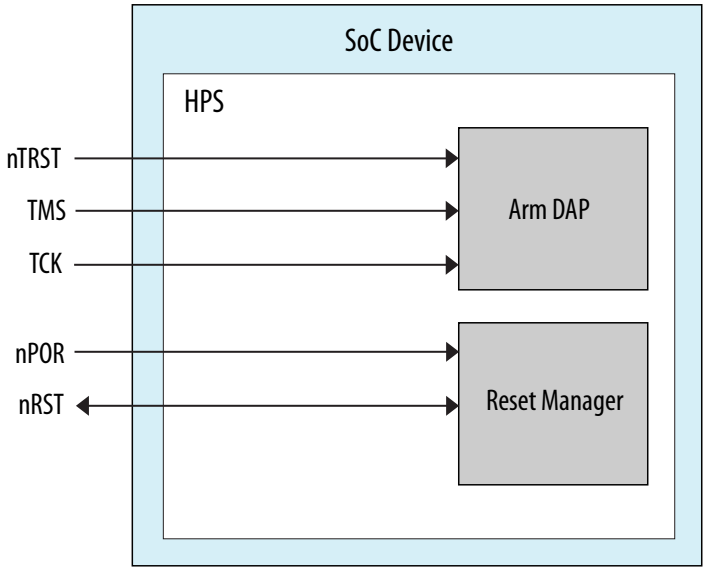
The following list describes the deassertion steps for both cold and warm reset shown in the Cold Reset Timing Diagram and Warm Reset Timing Diagram:

1. De-assert all RAM CLEAR resets. Reset Manager Initiates Clock Manager MPU clocks stop request.
2. Clock manager stops the MPU clocks and then waits 16 MPU clocks.
3. Clock manager asserts Clock stop acknowledge to Reset Manager.
4. Reset manager sees acknowledge and de-asserts L2 (mpu_l2_rst_n) reset only. Reset Manager de-asserts MPU stop request.
5. Clock manger sees request de-asserted. Wait 16 MPU clocks. Start MPU clocks, and Clock Manager de-asserts Clock stop acknowledge.
6. Reset manager sees de-assertion of acknowledge. Wait 16 cycles.
7. Reset manager asserts CPU0/1 CLKOFF signals and waits 32 cycles.
8. Reset manager de-asserts CPU0, CPU1, and SCU MPU resets (not WD). Wait 32 additional cycles.
9. De-assert CPU0/1 CLKOFF signals. Wait 16 cycles

10. Reset manager checks the Security fuse setting, and if the appropriate cold or warm reset fuse is set, do full RAM clearing sequence. Otherwise, clear only the MPU L1 RAM.
11. Wait for Security RAM Sequence to complete.
12. Assert again the CPU0, CPU1 and SCU MPU resets. Wait 16 cycles.
13. De-assert L3 reset, and if Cold Reset, de-assert DBG and COLD groups of resets. Wait for 100 cycles
14. De-assert SYS group module resets. Wait for 200 cycles.
15. Reset manager Initiates Clock Manager MPU clocks stop request.
16. Clock manager stops the MPU clocks and then waits 16 MPU clocks.
17. Clock manager asserts Clock stop acknowledge to Reset Manager.
18. Reset manager sees acknowledge and de-asserts L2 (mpu_l2_rst_n) reset only. Reset Manager de-asserts MPU stop request.
19. Clock manger sees request de-asserted. Wait 16 MPU clocks. Start MPU clocks, and Clock Manager de-asserts Clock stop acknowledge.
20. Reset manager sees de-assertion of acknowledge. Wait 16 cycles.
21. Reset manager asserts CPU0 CLKOFF signal(s) and waits 32 cycles.
22. Reset manager de-asserts remaining CPU0, SCU, and WD resets (no CPU1). Wait 32 additional cycles.
23. De-assert CPU0 CLKOFF signal(s). De-assertion sequence is finished.
24. Now SW should start running. SW to de-assert FPER/PER/SPER/BRIDGE/MPU2 resets.

4.2.2. Reset Pins

Figure 13. Reset Pins



The test reset (nTRST), test mode select (TMS), and test clock (TCK) pins are associated with the TAP reset domain and are used to reset the TAP controller in the DAP. These pins are not connected to the reset manager.

The nPOR and nRST pins are used to request cold and warm resets respectively. The nRST pin is an output as well. Any warm reset request causes the reset manager to drive the `rst_pin_oe_rst` output enable high, which drives the nRST pin low. The amount of time the reset manager pulls nRST low is controlled by the nRST pin count field (`nrstcnt`) of the reset cycles count register (`counts`); the default is 2048 clocks. This technique can be used to reset external devices (such as external memories) connected to the HPS.

4.2.3. Reset Effects

The following list describes how reset affects HPS logic:

- In the security manager domain, some status bits are cleared differently for cold and warm reset.
- The TAP reset domain ignores warm reset
- The Debug reset domain ignores warm reset
- The Clock Manager module is reset only by cold reset. After warm reset, the Clock Manager is put into Boot Mode.
- Each peripheral module and System Manager define reset behavior differently. See the appropriate control register for details.
- In the MPU module, the Watchdog Reset Status Registers are reset if an MPU Watchdog triggered the warm reset
- The Pin MUX is reset only by cold reset.

4.2.4. Reset Handshaking

The reset manager participates in several reset handshaking protocols to ensure other modules are safely reset.

Before issuing a warm reset, the reset manager performs a handshake with several modules to allow them to prepare for a warm reset. The handshake logic ensures the following conditions:

- Optionally the ETR master has no pending master transactions to the L3 interconnect
- Optionally preserve SDRAM contents during warm reset by issuing self-refresh mode request
- FPGA manager stops generating configuration clock
- Warns the FPGA fabric of the forthcoming warm reset

Similarly, the handshake logic associated with ETR also occurs during the debug reset to ensure that the ETR master has no pending master transactions to the L3 interconnect before the debug reset is issued. This action ensures that when ETR undergoes a debug reset, the reset has no adverse effects on the system domain portion of the ETR.

4.3. Reset Manager Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

5. FPGA Manager

The FPGA manager in the hard processor system (HPS) manages and monitors the FPGA portion of the system on a chip (SoC) device. The FPGA manager can configure the FPGA fabric from the HPS, monitor the state of the FPGA, and drive or sample signals to or from the FPGA fabric.

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

5.1. Features of the FPGA Manager

The FPGA manager provides the following functionality and features:

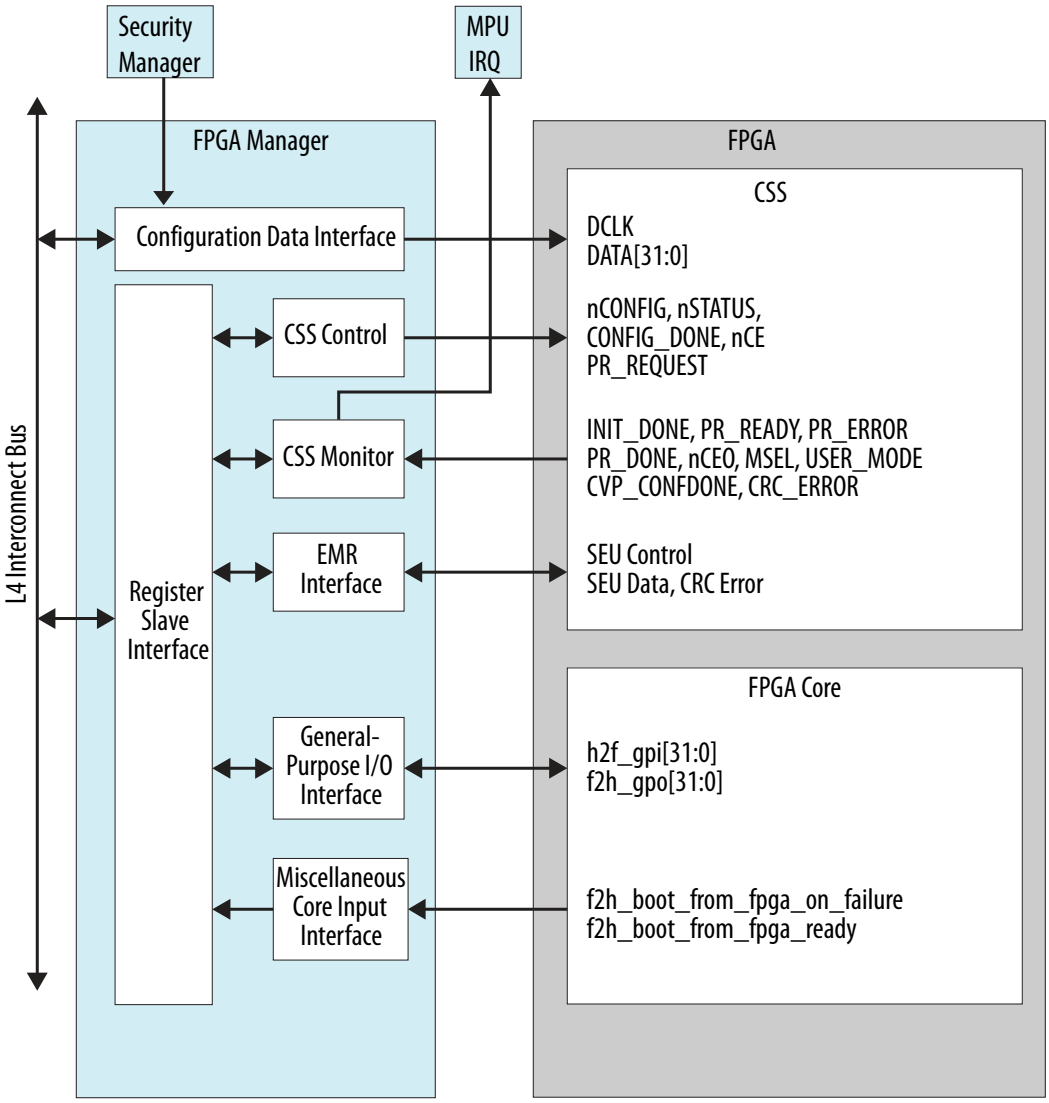
- Full configuration and partial reconfiguration
- CRC error message extraction
- General purpose I/O signals to the FPGA
- Boot from FPGA control
- Generation of interrupts based on FPGA status changes

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

5.2. FPGA Manager Block Diagram and System Integration

Figure 14. FPGA Manager Block Diagram



The FPGA Manager consists of the following blocks:

- Configuration Data Interface - Accepts and transfers the configuration and decryption data to the FPGA configuration sub system (CSS)
- Register Slave Interface - Accesses the control and status registers in the FPGA Manager
- CSS Control - Controls full and partial configuration of the FPGA
- CSS Monitor - Monitors the status of the FPGA during a full or partial configuration of the device

- Error Message Register (EMR) Interface - Error message extraction, in case of CRC errors in the FPGA
- General Purpose I/O Interface - Receives and drives 32 bits of general purpose I/O to and from the FPGA
- Miscellaneous Core Input Interface - Receives control input signals from the FPGA to support booting the HPS from the FPGA

5.3. Functional Description of the FPGA Manager

5.3.1. FPGA Configuration

You can configure the FPGA using an external device or through the HPS. This section highlights configuring the FPGA through the HPS.

The FPGA manager connects to the configuration logic in the FPGA portion of the device using a mode similar to how external logic (for example, MAX II or an intelligent host) configures the FPGA in fast passive parallel (FPP) mode. FPGA configuration through the HPS supports all the capabilities of FPP mode, including the following items:

- FPGA configuration
- Partial FPGA reconfiguration
- Decompression
- Advanced Encryption Standard (AES) encryption

Note: The FPGA manager supports a data width of 16 or 32 bits. When configuring the FPGA fabric from the HPS, Intel recommends that you always set the data width to 32 bits. For partial reconfiguration, the 16-bit and 32-bit data widths are options.

Table 29. MSEL Pin Settings for Each Configuration Scheme of Arria 10 Devices

Configuration Scheme	V_{CCPGM}(V)	Power-On Reset (POR) Delay	Valid MSEL[2:0]
FPP (x16 and x32)	1.8	Fast	000
		Standard	001

The FPGA Manager can be configured to accept configuration data directly from the MPU or the DMA engine. Either the processor or the DMA engine moves data from memory to the FPGA Manager data image register space `img_data_w`.

Configuration data is buffered in a 64 deep x 32 bits wide FIFO in the FPGA Manager. Status information such as FIFO empty/full and remaining depth can be accessed through a status register `imgcfg_stat`. FIFO empty/full conditions can also be enabled to generate an interrupt.

If using the DMA engine to move FPGA configuration data to the FPGA Manager, a FIFO threshold value can be set in the `dma_config` register. This value is used to control the assertion of a DMA transfer request from the FPGA Manager to the DMA engine.

Before sending FPGA configuration data to the FPGA Manager HPS, software sets the clock-to-data ratio field (`CDRATIO`) and configuration data width bit (`CFGWIDTH`) in the image control 2 register (`imgcfg_ctrl_02`).

Related Information

- Configuration, Design Security, and Remote System Upgrades in Arria 10 Devices
  For more information about configuring the FPGA through the HPS, refer to the "Configuration, Design Security, and Remote System Upgrade in Arria 10 Devices" appendix in the *Arria 10 Core Fabric and General Purpose I/Os Handbook*.
- Booting and Configuration on page 771

5.3.2. FPGA Status

Configuration signals from the FPGA CSS such as `INIT_DONE`, `CRC_ERROR` and `PR_DONE` are monitored by the FPGA Manager. Software configures the monitor block through the register slave interface, and can either poll FPGA signals or be interrupted. Monitored signals can be read through the `imgcfg_stat` register as well as the `intr_masked_status` register. Each of the monitored signals can generate an interrupt to the MPU global interrupt controller. Each interrupt source can be enabled and the polarity of the signals generating the interrupt can also be selected through the `intr_mask` and `intr_polarity` registers in the FPGA Manager.

5.3.3. Error Message Extraction

Cyclic redundancy check (CRC) errors from the FPGA fabric are monitored by the FPGA Manager. Upon assertion of a CRC error signal from the FPGA, the FPGA Manager extracts information about the error including:

- Error syndrome
- Error location
- Error type

A CRC error interrupt from the FPGA manager can be enabled through software. Software can then extract the CRC error information from the error message register (EMR) data interface. The number of valid error information bits in the EMR data registers depends on the specific FPGA device.

5.3.4. Data AES Decryption

The configuration data interface can also be used to transmit data from the HPS to the FPGA CSS Advanced Encryption Standard (AES) decryption engine. Software should sample the status registers in FPGA manager before switching modes between FPGA configuration and AES decryption. The FPGA manager does not do anything specific to handle a situation where software starts an AES decryption while in the middle of an FPGA configuration initiated by HPS, and the end result in this case is undefined.

5.3.5. Boot Handshake

There are two input signals from the FPGA to control HPS boot from the FPGA. Both are synchronized within the FPGA Manager. Boot software reads these signals before accessing a boot image in the FPGA. The following table describes the functionality of these signals.

Signal	Description
f2h_boot_from_fpga_on_failure	Indicates whether a fallback second-stage boot loader image is available in the FPGA on-chip RAM at memory location 0x0. The fallback second-stage boot loader image is used only if the HPS boot ROM does not find a valid second-stage boot loader image in the selected flash memory device.
f2h_boot_from_fpga_ready	The f2h_boot_from_fpga_ready signal is used by the Boot ROM when accessing the public key stored in the FPGA. The Boot ROM only uses the signal if asserted to boot with the public key. Indicates a second-stage boot loader image is available in an FPGA on-chip RAM at memory location 0x0 and it is ready to be accessed.

5.3.6. General Purpose I/O

Thirty-two general purpose inputs and thirty-two general purpose outputs are provided to the FPGA and are controlled through registers in the FPGA Manager.

No interrupts are generated through the input pins. All inputs are synchronized within the FPGA Manager. Output signals should be synchronized in the FPGA.

5.3.7. Clock

The FPGA manager has two clock input signals which are asynchronous to each other. The clock manager generates these two clocks:

- cfg_clk—the configuration slave interface clock input and also the DCLK output reference for FPGA configuration. Enable this clock in the clock manager only when configuration is active or when the configuration slave interface needs to respond to master requests.
- l4_mp_clk—the register slave interface clock.

Related Information

Clock Manager on page 52

5.3.8. Reset

The FPGA manager has the fpga_manager_rst_n reset signal. The reset manager drives this signal to the FPGA manager on a cold or warm reset. All distributed reset signals in the FPGA manager are asserted asynchronously at the same time and deasserted synchronously to their associated clocks.

Related Information

Reset Manager on page 134

5.4. FPGA Manager Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

6. System Manager

The system manager in the hard processor system (HPS) contains memory-mapped control and status registers (CSRs) and logic to control system level functions as well as other modules in the HPS.

The system manager connects to the following modules in the HPS:

- Direct memory access (DMA) controller
- Ethernet media access controllers (EMAC0, EMAC1, and EMAC2)
- Error Checking and Correction Controller (ECC) for RAMs
- Microprocessor unit (MPU) subsystem
- NAND flash controller
- Secure Digital/MultiMediaCard (SD/MMC) controller
- Quad serial peripheral interface (SPI) flash controller
- USB 2.0 On-The-Go (OTG) controllers (USB0 and USB1)
- Watchdog timers

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

6.1. Features of the System Manager

Software accesses the CSRs in the system manager to control and monitor various functions in other HPS modules that require external control signals. The system manager connects to these modules to perform the following functions:

- Sends pause signals to pause the watchdog timers when the processors in the MPU subsystem are in debug mode
- Selects the EMAC system interconnect master access options and other EMAC clock and interface options.
- Selects the SD/MMC controller clock options and system interconnect master access options.
- Selects the NAND flash controller bootstrap options and system interconnect master access option.
- Selects USB controller system interconnect master access option.
- Provides control over the DMA security settings when the HPS exits from reset.
- Provides boot source information that can be read during the boot process.
- Provides the capability to enable or disable an interface to the FPGA.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

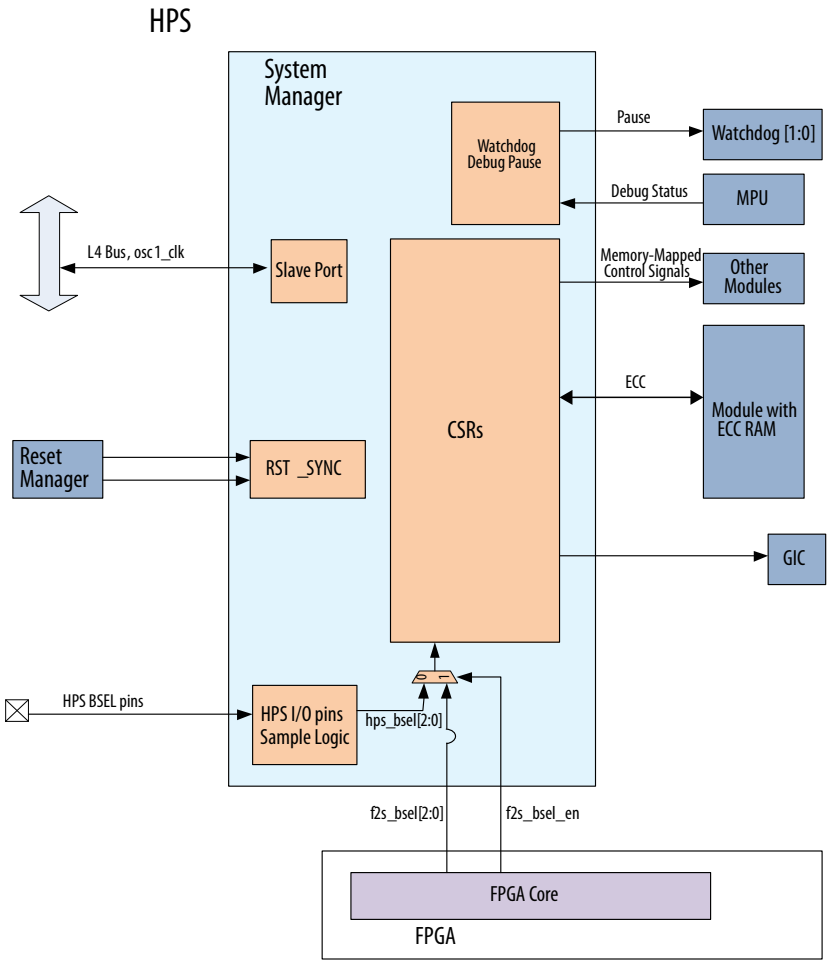
ISO 9001:2015 Registered

- Provides combined ECC status and interrupts from other HPS modules with ECC-protected RAM.
- Routes parity failure interrupts from the L1 caches to the Global Interrupt Controller.
- Provides the capability to inject errors during testing in the MPU L2 ECC-protected RAM.

6.2. System Manager Block Diagram and System Integration

The system manager connects to the level 4 (L4) bus through a slave interface. The CSRs connect to signals in the FPGA and other HPS modules.

Figure 15. System Manager Block Diagram



The system manager consists of the following:

- CSRs—Provide memory-mapped access to control signals and status for the following HPS modules:
  - EMACs
  - Debug core
  - SD/MMC controller
  - NAND controller
  - USB controllers
  - DMA controller
  - System interconnect
  - ECC memory interfaces for the following peripherals:
    - USB controllers
    - SD/MMC controller
    - Ethernet MACs
    - DMA controller
    - NAND flash controller
    - On-chip RAM
- Slave port interface—provides access to system manager CSRs for connected masters.
- Watchdog debug pause—accepts the debug mode status from the MPU subsystem and pauses the L4 watchdog timers.
- Reset Manager— system manager receives the reset signals from reset manager.

6.3. Functional Description of the System Manager

The system manager serves the following purposes:

- Provides software access to boot configuration and system information
- Provides software access to control and status signals in other HPS modules
- Provides combined ECC status and interrupt from other HPS modules with ECC-protected RAM
- Enables and disables HPS peripheral interfaces to the FPGA
- Provides eight registers that software can use to pass information between boot stages

6.3.1. Boot Configuration and System Information

The system manager provides boot configuration information through the `bootinfo` register. Sampled value of the HPS boot select (BSEL) pins are available to the Boot ROM software.

The boot source is determined by a combination of the BSEL pins and a fuse bit that can bypass the BSELpins.

Related Information

Booting and Configuration on page 771

6.3.2. Additional Module Control

Each module in the HPS has its own CSRs, providing access to the internal state of the module. The system manager provides registers for additional module control and monitoring. To fully control each module, you must program both the peripheral's CSR and its corresponding CSR in the System Manager. This section describes how to configure the system manager CS for each module.

6.3.2.1. DMA Controller

The security state of the DMA controller is controlled by the manager thread security (`mgr_ns`) and interrupt security (`irq_ns`) bits of the DMA register.

The `periph_ns` register bits determine if a peripheral request interface is secure or non-secure.

Note: The `periph_ns` register bits must be configured before the DMA is released from global reset.

Related Information

- DMA Controller on page 512
- System Manager Address Map and Register Definitions on page 167

6.3.2.2. NAND Flash Controller

The bootstrap control register (`nand_bootstrap`) modifies the default behavior of the NAND flash controller after reset. The NAND flash controller samples the bootstrap control register bits when it comes out of reset.

The following `nand_bootstrap` register bits control configuration of the NAND flash controller:

- Bootstrap inhibit initialization bit (`noinit`)—inhibits the NAND flash controller from initializing when coming out of reset, and allows software to program all registers pertaining to device parameters such as page size and width.
- Bootstrap 512-byte device bit (`page512`)—informs the NAND flash controller that a NAND flash device with a 512-byte page size is connected to the system.
- Bootstrap inhibit load block 0 page 0 bit (`noloadb0p0`)—inhibits the NAND flash controller from loading page 0 of block 0 of the NAND flash device during the initialization procedure.
- Bootstrap two row address cycles bit (`tworowaddr`)—informs the NAND flash controller that only two row address cycles are required instead of the default three row address cycles.

You can use the system manager's `nanad_l3master` register to control the following signals:

- ARPROT
- AWPROT
- ARDOMAIN
- AWDOMAIN
- ARCACHE
- AWCACHE

These bits define the cache attributes for the master transactions of the DMA engine in the NAND controller.

Note: Register bits must be accessed only when the master interface is guaranteed to be in an inactive state.

Related Information

- NAND Flash Controller on page 370
- System Manager Address Map and Register Definitions on page 167

6.3.2.3. EMAC

You can program the `emac_global` register to select either `emac_ptp_clk` from the Clock Manager or `f2s_ptp_ref_clk` from the FPGA fabric as the source of the IEEE 1588 reference clock for each EMAC.

You can use the system manager's `l3master` register to control the EMAC's ARCACHE and AWCACHE signals, by setting or clearing the (`arcache, awcache`) bits. These bits define the cache attributes for the master transactions of the DMA engine in the EMAC controllers.

Note: Register bits must be accessed only when the master interface is guaranteed to be in an inactive state.

The `phy_intf_sel` bit is programmed to select between a GMII (MII), RGMII or RMII PHY interface when the peripheral is released from reset. The `ptp_ref_sel` bit selects if the timestamp reference is internally or externally generated. The `ptp_ref_sel` bit must be set to the correct value before the EMAC core is pulled out of reset.

Related Information

- Clock Manager on page 52
- Ethernet Media Access Controller on page 520

6.3.2.4. USB 2.0 OTG Controller

The `usb*_l3master` registers in the system manager control the HPROT and HAUSER fields of the USB master port of the USB 2.0 OTG Controller.

Note: Register bits should be accessed only when the master interface is guaranteed to be in an inactive state.

Related Information

USB 2.0 OTG Controller on page 591

6.3.2.5. SD/MMC Controller

The `sdmmc_l3master` register in the system manager controls the HPROT and HAUSER fields of the SD/MMC master port.

Note: Register bits should be accessed only when the master interface is guaranteed to be in an inactive state.

You can program software to select the clock's phase shift for `cclk_in_drv` and `cclk_in_sample` by setting the drive clock phase shift select (`drvsel`) and sample clock phase shift select (`smplsel`) bits of the `sdmmc` register in the system manager.

Related Information

SD/MMC Controller on page 405

6.3.2.6. Watchdog Timer

The system manager controls the watchdog timer behavior when the CPUs are in debug mode. The system manager sends a pause signal to the watchdog timers depending on the setting of the debug mode bits of the L4 watchdog debug register (`wddbg`). Each watchdog timer built into the MPU subsystem is paused when its associated CPU enters debug mode.

Related Information

Watchdog Timer on page 693

6.3.3. Boot ROM Code

Registers in the system manager control whether the boot ROM code configures the pin multiplexing for boot pins after a warm reset. Set the warm-reset-configure-pin-multiplex for boot pins bit (`warmrstcfgpinmux`) of the boot ROM code register to enable or disable this control.

Note: The boot ROM code always configures the pin multiplexing for boot pins after a cold reset.

Registers in the system manager also control whether the boot ROM code configures the I/O pins used during the boot process after a warm reset. Set the warm reset configure I/Os for boot pins bit (`warmrstcfgio`) of the `ctrl` register to enable or disable this control.

Note: By default, the boot ROM code always configures the I/O pins used by boot after a cold reset.

There can be up to four preloader images stored in flash memory. The (`initswlastld`) register contains the index of the preloader's last image that is loaded in the on-chip RAM.

The boot ROM software state register (`bootromswstate`) is a 32-bit general-purpose register reserved for the boot ROM.

The following warmram related registers are used to configure the warm reset from on-chip RAM feature and must be written by software prior to the warm reset occurring.

Table 30. The warmram Registers

Register	Name	Purpose
`enable`	Enable	Controls whether the boot ROM attempts to boot from the contents of the on-chip RAM on a warm reset.
`datastart`	Data start	Contains the byte offset of the warm boot CRC validation region in the on-chip RAM. The offset must be word-aligned to an integer multiple of four.
`length`	Length	Contains the length in bytes of the region in the on-chip RAM available for warm boot CRC validation.
`execution`	Execution address	Contains the global address the boot code jumps to in the on-chip RAM if the CRC validation succeeds.
`crc`	Expected CRC	Contains the expected CRC of the region in the on-chip RAM.

The number of wait states applied to the boot ROM's read operation is determined by the wait state bit (`waitstate`) of the `ctrl` register. After the boot process, software might require reading the code in the boot ROM. If software has changed the clock frequency of the `l3_main_clk` after reset, an additional wait state is necessary to access the boot ROM. Set the `waitstate` bit to add an additional wait state to the read access of the boot ROM. The enable safe mode warm reset update bit controls whether the wait state bit is updated during a warm reset.

6.3.3.1. Controlling the Boot Memory Map Through the System Interconnect

The System Manager contains a register, `noc_addr_remap_value`, that controls how the system interconnect maps memory addresses starting at 0x00000000. This register remaps the bottom of the memory map to either boot ROM or on-chip RAM.

Refer to "Address Remapping" in the System Interconnect chapter for details about using the `noc_addr_remap_value` register.

Related Information

Address Remapping on page 0

6.3.4. FPGA Interface Enables

The system manager can enable or disable interfaces between the FPGA and HPS.

The global interface bit (`intf`) of the global disable register (`gbl`) disables all interfaces between the FPGA and HPS.

Note: Ensure that the FPGA in configured before enabling the interfaces and that all interfaces between the FPGA and HPS are inactive before disabling them.

You can program the individual disable register (`indiv`) to disable the following interfaces between the FPGA and HPS:

You can program the FPGA interface enable registers (`fpgaintf_en_*`) to disable the following interfaces between the FPGA and HPS:

- Reset request interface
- JTAG enable interface
- I/O configuration interface
- Boundary scan interface
- Debug interface
- Trace interface
- System Trace Macrocell (STM) interface
- Cross-trigger interface (CTI)
- QSPI interface
- SD/MMC interface
- SPI Master interface
- SPI Slave interface
- EMAC interface
- UART interface

6.3.5. ECC and Parity Control

The system manager can mask the ECC interrupts from each of the following HPS modules with ECC-protected RAM:

- MPU L2 cache data RAM
- On-chip RAM
- USB 2.0 OTG controller (USB0 and USB1) RAM
- EMAC (EMAC0, EMAC1, and EMAC2) RAM
- DMA controller RAM
- NAND flash controller RAM
- Quad SPI flash controller RAM
- SD/MMC controller RAM
- DDR interfaces

The system manager can inject single-bit or double-bit errors into the MPU L2 ECC memories for testing purposes. Set the bits in the appropriate memory enable register to inject errors. For example, to inject a single bit ECC error, set the `injs` bit of the `mpu_ctrl_l2_ecc`register.

Note: The injection request is edge-sensitive, meaning that the request is latched on 0 to 1 transitions on the injection bit. The next time a write operation occurs, the data is corrupted, containing either a single or double bit error as selected. When the data is read back, the ECC logic detects the single or double bit error appropriately. The injection request cannot be cancelled, and the number of injections is limited to once every five MPU cycles.

The system manager can also inject parity failures into the parity-protected RAM in the MPU L1 to test the parity failure interrupt handler. Set the bits of the parity fail injection register (`parityinj`) to inject parity failures.

Note: Injecting parity failures into the parity-protected RAM in the MPU L1 causes the interrupt to be raised immediately. There is no actual error injected and the data is not corrupted. Furthermore, there is no need for a memory operation to actually be performed for the interrupt to be raised.

6.3.6. Preloader Handoff Information

The system manager provides eight 32-bit registers to store handoff information between the preloader and the operating system. The preloader can store any information in these registers. These register contents have no impact on the state of the HPS hardware. When the operating system kernel boots, it retrieves the information by reading the preloader to OS handoff information register array. These registers are reset only by a cold reset.

6.3.7. Clocks

The system manager is driven by a clock generated by the clock manager.

Related Information

Clock Manager on page 52

6.3.8. Resets

The system manager receives two reset signals from the reset manager. The `sys_manager_rst_n` signal is driven on a cold or warm reset and the `sys_manager_cold_rst_n` signal is driven only on a cold reset. This function allows the system manager to reset some CSR fields on either a cold or warm reset and others only on a cold reset.

Related Information

Reset Manager on page 134

6.4. System Manager Address Map and Register Definitions

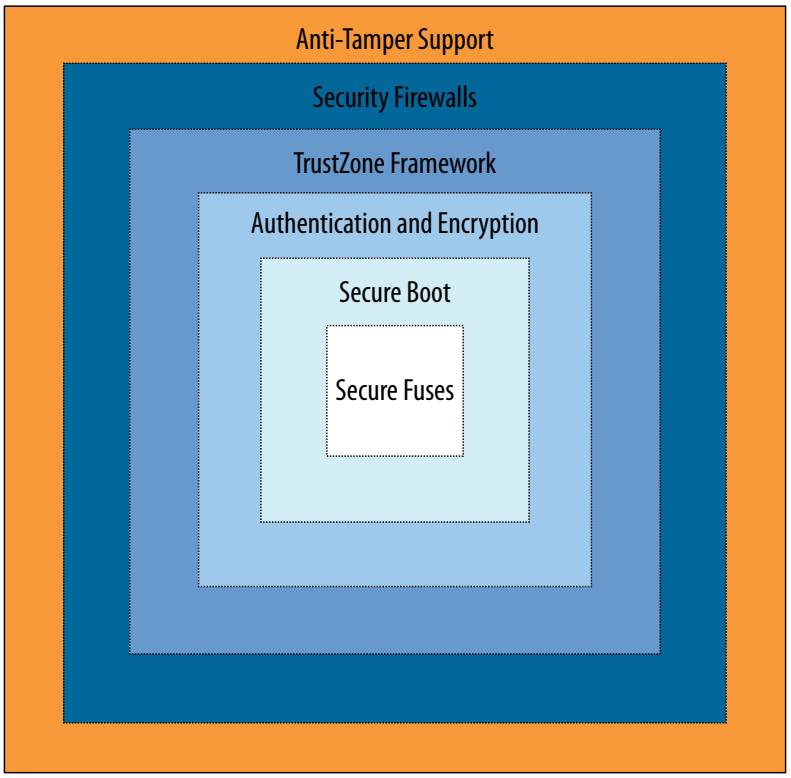
For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

7. SoC Security

Note: **For a complete NDA version of the entire Security SoC chapter, address map and register definitions, contact your local field sales office.**

The Intel Arria 10 SoC provides the framework for implementing secure systems by offering a layered hardware and software solution. A dedicated Security Manager module in the Hard Processor System (HPS) supervises secure initialization of the SoC and manages system response during tamper events. User fuses offer secure boot options and registers within the Security Manager provide further enhancement of secure states. Using Elliptical Curve Digital Signal Algorithm (ECDSA256) with Secure Hash Algorithm (SHA256) and Advanced Encryption Standard (AES) algorithms in the FPGA's Configuration Subsystem (CSS), boot images can be authenticated and decrypted. The integration of the Arm TrustZone* technology and security firewalls within the Intel Arria 10 system interconnect provides defined partitioning in the device for secure and non-secure accesses. Protection mechanisms are integrated into debug components of the SoC to allow varying levels of visibility for debug.

Figure 16. Intel Arria 10 Layered Security Solution



Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

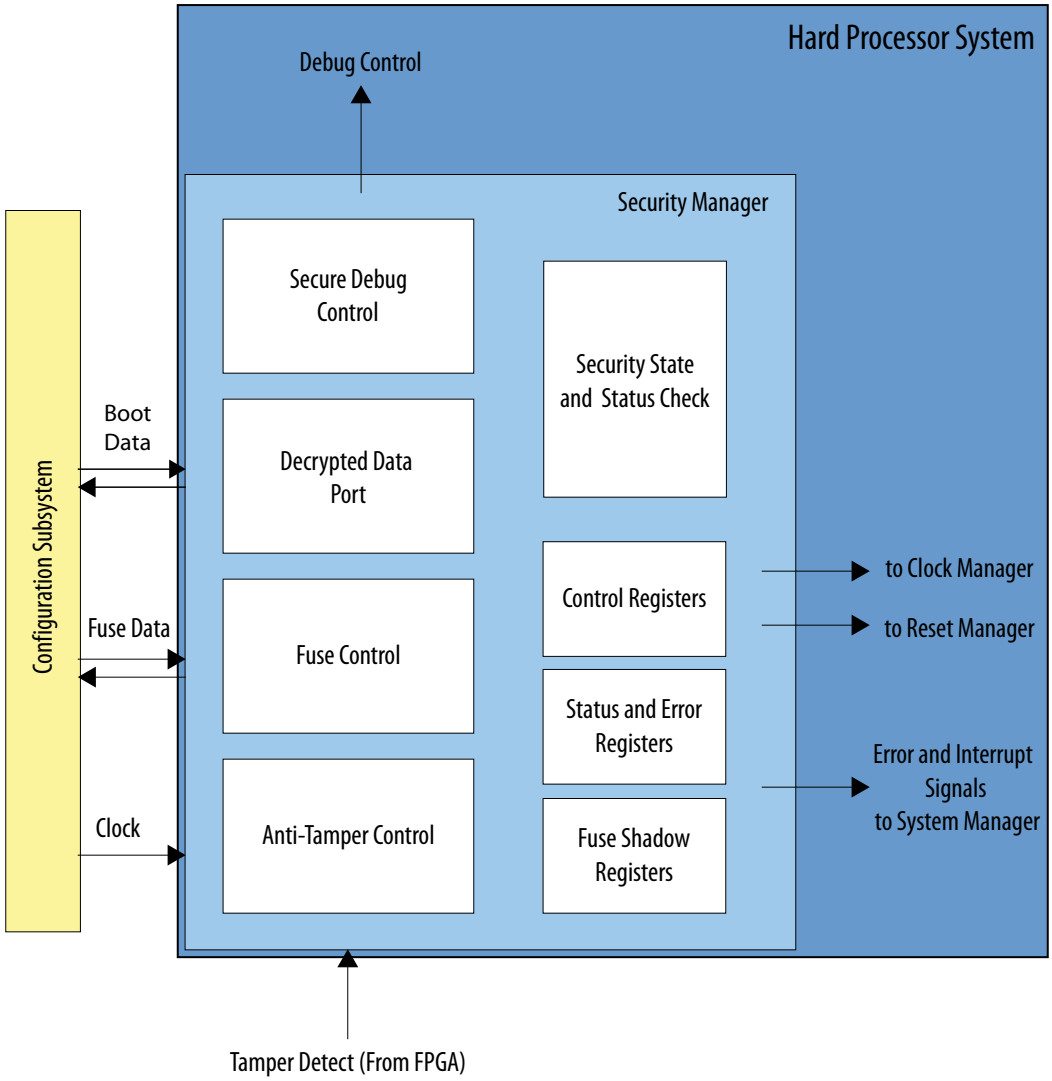
7.1. Security Manager

The Security Manager module is integrated in the Hard Processor System (HPS) and provides the overall management of security within the SoC, including:

- Recognition of secure fuse configuration
- Secure state control and status check of security features
- Secure boot options
- Varying levels of debug visibility
- Anti-Tamper support

7.1.1. Security Manager Block Diagram

Figure 17. Security Manager Block Diagram



7.1.2. Functional Overview

The Security Manager integrates several functions that support the TrustZone technology, manage security states in the device, and hold secure fuse information.

The main components of the Security Manager architecture include:

- Fuse Control (TEST):

  When the HPS is powered, the Security Manager ensures reliable and verified receipt of the fuse information from the Configuration Subsystem (CSS) in the FPGA, stores it in fuse shadow registers and can request further fuse information.
- Security State and Status Check:

  This sub-module holds the security state of the system, which is controlled by the fuse bits, hardware and software programming. This sub-module also has the ability to check and raise the level of security.
- Encryption Data Port:

  This interface receives authenticated and decrypted boot images from the CSS.

  — Support for ECDSA256 (SHA256) authenticated boot.

  — Support for AES-based encrypted boot.
- Registers:

  — Control Registers configure security state and debug options for the device.

  — Status and Error Registers flag transmission errors and interrupts.

  — Fuse Shadow Registers hold a copy of the user fuse information.
- Anti-Tamper Control:

  On a tamper event, this module sends a signal to the Reset Manager to initiate the scrambling and clearing of all memories, including on-chip RAM, peripheral memories, L1 cache and L2 cache. Upon completion, the Security Manager sends a signal to the FPGA to indicate that the anti-tamper event has been handled.

7.1.3. Functional Description

7.1.3.1. Secure Initialization Overview

Secure initialization allows the device to boot and configure in a secured state.

Secure initialization begins with the CSS module. On FPGA power-up, the Configuration Subsystem (CSS) powers, initializes and loads the fuse bits. The CSS sends the FPGA its fuse configuration information. If the HPS is powered, the CSS sends the HPS fuse information to the Security Manager. This information is held in the HPS_`fusesec` shadow register in the Security Manager.

When the Security Manager is released from reset, it requests configuration information from the CSS and performs security checks. At this point, the rest of the HPS is still in reset. The security checks validate whether the state of each security option is valid. The Security Manager decodes the fuse bits and brings the rest of the HPS out of reset.

If there are errors in the initial transmission of the secure fuse information, the HPS stays in reset and no automatic retry occurs. Only an FPGA re-configuration causes a retry.

When the HPS is released from reset, the Security Manager sends signals to initialize the system blocks, such as the Clock Manager, FPGA Manager, System Manager. The clock control fuse information is automatically sent to the Clock Manager, the memory control fuse information is automatically sent to the Reset Manager and all other fuse functions (authentication, encryption, and public key source and length) are stored in a memory-mapped location for the boot ROM code to read. After these tasks are successfully completed, CPU0 comes out of reset in a secure state.

After CPU0 is released from reset, the boot ROM begins executing. At this time, the HPS is in a trusted state and the boot ROM code is guaranteed to execute as expected. For both secure and non-secure boot, all slave peripherals are brought out of reset in a secure state.

After initial security controls have been set, the boot ROM determines the second-stage boot loader source from the `bootinfo` register in the System Manager. The boot source can be from external memory through the HPS or through the FPGA. If the second-stage boot loader code comes from HPS external memory, then the boot ROM configures the clock and the interface to external memory.

When the boot ROM attempts to read from the flash, it looks for one of four images in external memory. Initially, it looks for image 0. If it is found, the image is authenticated, decrypted, or both depending on the requirements of the current security state. If these steps are successful, then the image is executed. However, if any of these steps fail, then the boot ROM moves onto the next image until it runs out of images.

For example, during a secure boot, the second-stage boot loader header includes an authentication key and the incoming image is authenticated. If indicated from the current security state, the image may be decrypted as well. The boot ROM contains functions to establish transitive trust to the second-stage boot code loaded from external flash or from the FPGA into on-chip RAM. If trust cannot be established in a secure boot, the HPS does not come out of reset and can attempt to load a different second-stage boot image.

For comprehensive information on the booting and boot image partitioning refer to the *Booting and Configuration* appendix of the *Arria 10 Hard Processor Technical Reference Manual*.

If all the images fail, it attempts to locate an image from the FPGA fabric and boot from there. If the FPGA boot fails, then it halts.

If the HPS receives its second-stage boot loader code from the FPGA, it waits for the FPGA to finish configuration before it obtains the image from FPGA RAM.

If required, a portion of the second-stage boot loader header can be used to raise security states of security features in the device that are currently in non-secure mode.

Related Information

- Secure Boot on page 182
  For more information about the secure boot process, refer to the "Secure Boot" section.
- Booting and Configuration on page 771
  For more information about the boot process refer to the *Booting and Configuration* appendix.

7.1.3.1.1. Secure Fuses

During initialization of the device, the Configuration Subsystem (CSS) sends the value of the HPS secure fuses to the HPS and holds a copy of the FPGA secure fuses. The HPS and FPGA each hold their own fuse values. However, a copy of these fuse values is held in the HPS_fusesec and fpga_fusesec registers within the Security Manager.

The following table details the HPS security fuse bits sent by the CSS to the Security Manager and contained within the HPS_fusesec register. A "blown" fuse state is represented by a 1 in the HPS_fusesec register and a "not blown" fuse state is represented by a 0.

Table 31. HPS_fusesec Register Description

Bits	Name	Description
31:27	Reserved	Bit values in this field are undefined.
26:23	csel_f	This field indicates the value of the clock select fuses that are available for configuring the clock for the boot interface and for the PLLs. Refer to the *Clock Configuration* section for more information on CSEL encodings.
22	dbg_access_f	This fuse determines the initial state of the debug access domains.
21	dbg_lock_JTAG	This field indicates if the HPS JTAG access level can be changed through software when the HPS is released from reset. • 0x0= HPS JTAG access level can be changed through the sec_jtagdbg register. • 0x1= HPS JTAG access level cannot be changed (locked).
20	dbg_lock_DAP	This field indicates if the DAP access level can be changed through software when the HPS is released from reset. • 0x0= The DAP access level can be changed through the sec_dapdbg register. • 0x1= The DAP access level cannot be changed (locked).
19	dbg_lock_CPU0	This field indicates if the CPU0 debug access level can be changed through software when the HPS is released from reset. • 0x0= CPU0 debug access level can be changed through the sec_cpu0dbg register. • 0x1= CPU0 debug access level cannot be changed (locked).
18	dbg_lock_CPU1	This field indicates if the CPU1 debug access level can be changed through software when the HPS is released from reset. • 0x0= The CPU1 debug access level can be changed through the sec_cpu1dbg register. • 0x1= The CPU1 debug access level cannot be changed (locked).
17	dbg_lock_CS	This field indicates if the CoreSight debug access level can be changed through software when the HPS is released from reset. • 0x0= The CoreSight debug access level can be changed through the sec_csdbg register. • 0x1= The CoreSight debug access level cannot be changed (locked).

continued...

Bits	Name	Description
16	dbg_lock_FPGA	This field indicates if the FPGA debug access level can be changed through software when the HPS is released from reset. • 0x0= The FPGA debug access level can be changed through the sec_fpgadbg register. • 0x1= The FPGA debug access level cannot be changed (locked).
15:12	Reserved	Bit values in this field are undefined.
11	clr_ram_order_f	This fuse value determines how RAMs are cleared during a tamper event. • 0x0= All RAMs are cleared in parallel. • 0x1= All RAMs are cleared in series.
10	clr_ram_cold_f	This fuse value indicates what happens to the RAM on a cold reset. • 0x0= All RAMs are not cleared on a cold reset. • 0x1= All RAMs are cleared on a cold reset.
9	clr_ram_warm_f	This fuse value indicates what happens to the RAMs on a warm reset. • 0x0= All RAMs are not cleared on a warm reset. • 0x1= All RAMs are cleared on a warm reset.
8	oc_boot_f	This fuse value determines if the second-stage boot code is allowed to boot from on-chip RAM. • 0x0= Second-stage boot can be from on-chip RAM if enabled by the System Manager. • 0x1= Second-stage boot is not from on-chip RAM.
7	hps_clk_f	This fuse value selects the clock used for the boot process and in the case of a tamper event, memory scrambling. • 0x0= The external oscillator, HPS_CLK1, is used for boot. • 0x1= The internal oscillator, cb_intosc_ls_clk, is used for boot.
6	fpga_boot_f	If blown, this fuse value allows the FPGA to configure independently and allows the HPS to boot from an encrypted next-stage boot source that was decrypted into the FPGA. • 0x0= Booting is dependent on the BSEL pins. • 0x1= HPS only boots from the FPGA; BSEL options are ignored and CSEL fuse options are ignored.
5	aes_en_f	This fuse value indicates if a decryption of the flash image is always performed. • 0x0= An AES decryption of the flash image is determined from the second stage boot loader header. • 0x1= An AES decryption of the flash image is always performed.

continued...

Bits	Name	Description
4:2	kak_src_f	This bit field indicates the source of the Key Authorization Key (KAK) which can be in: • Proprietary ROM • FPGA logic elements • User fuses
1	kak_len_f	This fuse value indicates the Key Authorization Key (KAK) length: • 0x0= 256 bits • 0x1= 384 bits
0	authen_en_f	This fuse value determines whether authentication of flash images is required prior to execution. • 0x0= No authentication of the flash image is required prior to execution. • 0x1= HPS authentication of all flash images is required prior to execution.

At initialization, the FPGA also receives fuse information that is pertinent to its configuration. The HPS can read this information through a secure serial interface, which shifts the FPGA fuse values into the fpga_fusesec register in the Security Manager. The CSS shifts in a 32-bit value although some of the bits are considered reserved.

Related Information

- Clock Configuration on page 178
  For more information about the CSEL settings, refer to the "Clock Configuration" section.
- Configuration, Design Security, and Remote System Upgrades in Arria 10 Devices

7.1.3.2. Security State

The security state of the entire device is defined by the combination of the security state of the HPS and the FPGA.The initial security state of the device is determined by the fuse settings in the SoC. The values of these fuses are written to two shadow registers within the Security Manager: the hps_fusesec and the fpga_fusesec registers.

Each feature's security level can be increased from its initial fuse level through software, and for debug, through hardware as well. Changes to security level can only increase security and can never lower security. Both the HPS and FPGA have a mechanism to load security option bits as part of the second-stage boot loader and POF file, respectively, which can perform a check of the current security state and raise the security level.

7.1.3.2.1. Security Check

The Security Manager has the capability to perform security checks to identify discrepancies between the actual and expected security level so that measured actions can be taken through software and hardware if required.

There are three types of security checks that can be performed by the Security Manager:

- A fuse security level check
- A hardware access security level check
- A software access security level check

7.1.3.2.2. Secure Serial Interface

The HPS and FPGA communicate through a secure serial interface. The interface allows security of the HPS to be separate from the security of the FPGA. Software can initiate a request of the serial fuse interface via the Security Manager register.

7.1.3.3. Secure Debug

Debug logic in the Security Manager controls whether to enable or disable the JTAG, CPUs, Debug Access Port (DAP) and CoreSight domains. Debug access is determined by a combination of debug fuses, option bits, and registers that change the default state and set the functional state of debug peripherals.

During a power-on reset, access to all debug domains are prevented. During a warm reset, access to all debug domains except JTAG are prevented. Once a cold or warm reset is exited, the default debug access is defined by the value of the dbg_disable_access fuse and the dbg_lock* fuses. When the dbg_disable_access fuse is blown, HPS debug accesses to all domains are disabled by default out of reset. If the fuse is not blown, then the initial HPS debug state out of reset is enabled. To ensure that the device is brought up in a secure state, the dbg_disable_access fuse must be blown.

Related Information

- ARM Debug Permissions
  Refer to this location for information on debug signals and permissions.
- ARM Infocenter
  For more information about debug, refer to the ARM Cortex-A9 Technical Reference Manual.

7.1.3.3.1. MPU

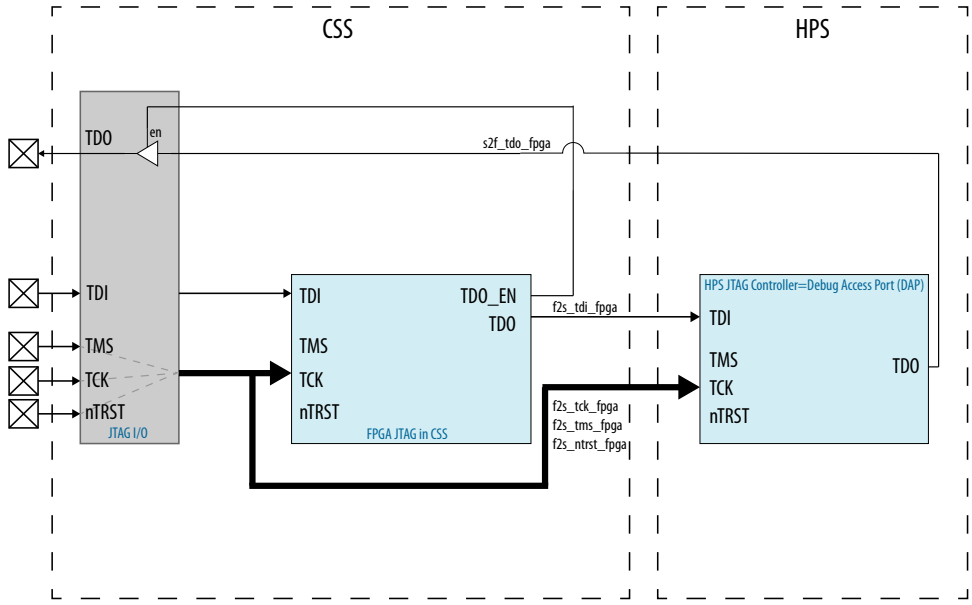
The MPU supports invasive and non-invasive debug. Invasive debug allows the user to control and observe the processor. Non-invasive debug allows you to observe the processor but not control it and is always available as part of invasive debug. The NIDEN and SPNIDEN signals are used for non-invasive debug in the MPU.

7.1.3.3.2. JTAG

The HPS and FPGA share a common set of JTAG pins and each have their own TAP controller which are chained together inside the device.

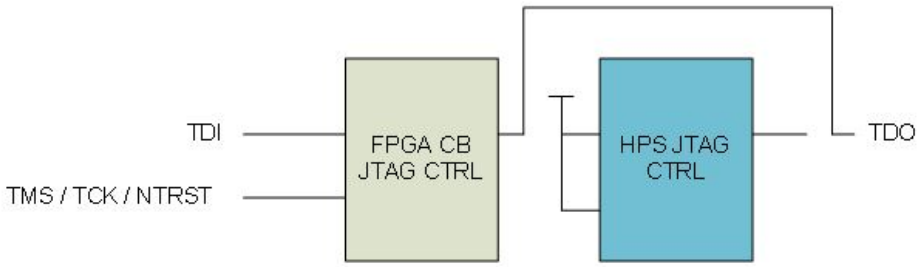
In the default JTAG configuration, the FPGA and HPS JTAG controllers are daisy-chained and controlled by an external JTAG port. The HPS JTAG controller is also referred to as the Debug Access Port.

Figure 18. JTAG Chaining



During power-on-reset, the JTAG and all debug domains are disabled. After the device is released from reset, the debug fuses are read by the Configuration Subsystem to determine if the external JTAG port or the JTAG controller to the FPGA or HPS is bypassed. Bypass can additionally be enabled through option bits in the Security Option registers discussed in the *Security State* section.

Figure 19. HPS JTAG Bypass



7.1.3.4. Reset Manager

To avoid unsecure snooping of RAM contents, the Security Manager can send signals to the Reset Manager to clear all RAM on a cold or warm reset.

These signals are the result of fuse programming combined with the configuration bits found in the HPS_Swoptset and HPS_secopt1 register in the Security Manager.

In addition, the fuse information that determines whether the RAMs are cleared in series or parallel is also sent to the Reset Manager. Clearing the memory in series prevents a power spike that could occur with parallel clearing of RAM. The policies for how or when security states take affect can be programmed in the HPS_secopt1 register. Refer to the "Security State" section for more information.

The RAM instances that are cleared during a warm or cold reset are:

- On-chip RAM
- USB0
- USB1
- SD/MMC
- EMAC0 RX and TX buffer
- EMAC1 RX and TX buffer
- EMAC2 RX and TX buffer
- DMA
- NAND read data, write data, and ECC RAMs
- QSPI
- MPU RAM

If a tamper event occurs, the Security Manager notifies the Reset Manager and the following sequence occurs:

1. All domain resets are asserted.
2. All RAMs are cleared.
3. The Reset Manager enters a dead state waiting for a POR release from Security Manager. All cold reset sources except Security Manager are ignored.

When FPGA POR and HPS POR are asserted, the Security Manager goes into reset.

Related Information

- Security State on page 175
  For more information regarding Security States, refer to the "Security State" section.
- Reset Manager on page 134
  For more information about reset, refer to the *Reset Manager* chapter.

7.1.3.5. Clock Configuration

Security Manager is driven by an internal oscillator, `cb_intosc_hs_clk`, in the Configuration Subsystem. This clock has wide variation across process and temperature. Once the Security Manager reads the fuse information, it drives the boot clock configuration to the Clock Manager. The Clock Manager samples the clock configuration on a cold and warm reset. If the `hps_clk_f` fuse is blown, the internal oscillator divided by 2, `cb_intosc_hs_div2_clk`, is used as the boot clock. If the fuse is not blown, an external oscillator is used. During boot ROM execution, the boot code configures the device clock based on the CSEL fuse settings or software code. The `cb_intosc_hs_div2_clk` can be considered the secure reference clock option.

The CSEL fuse settings have different meanings depending on how the hps_clk_f fuse is configured. When the hps_clk_f fuse is not blown, the external oscillator is the boot clock input.

Table 32. CSEL Encodings for hps_clk_f = 0

CSEL[3:0] Fuse Value	Description	MPU Clock Value
0x0	When no fuses are blown, the boot ROM returns clocks back to their default (bypass) boot mode and the CPU is driven by the external oscillator (HPS_CLK1), which must be in the range of 10 to 50 MHz. No PLL is enabled.	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x1	This encoding is typically used during a warm reset if you have already configured your clocks or PLL in the clock manager and would like to continue to use this configuration. If this is encoding is selected, no changes are made to these settings and the clock configuration is essentially user selected. If this is encoding is used during a power-on reset, then whatever the default reset values of the clocks are used.	User-selected clock source. Refer to the *Clock Manager* chapter for clock register settings.
0x2	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x3	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x4	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x5	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x6	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x7	External oscillator input clock, HPS_CLK1, is in the range of 10 to 15 MHz	533 to 800Mhz
0x8	External oscillator input clock, HPS_CLK1, is in the range 15 to 20 MHz.	600 to 800Mhz
0x9	External oscillator input clock, HPS_CLK1, is in the range 20 to 25 MHz.	640 to 800Mhz
0xA	External oscillator input clock, HPS_CLK1, is in the range 25 to 30 MHz.	666 to 800Mhz
0xB	External oscillator input clock, HPS_CLK1, is in the range 30 to 35 MHz.	685 to 800Mhz
0xC	External oscillator input clock, HPS_CLK1, is in the range 35 to 40 MHz.	700 to 800Mhz

continued...

CSEL[3:0] Fuse Value	Description	MPU Clock Value
0xD	External oscillator input clock, HPS_CLK1, is in the range 40 to 45 MHz.	711 to 800Mhz
0xE	External oscillator input clock, HPS_CLK1, is in the range 45 to 50 MHz.	720 to 800Mhz
0xF	The boot ROM returns clocks back to their default (bypass) boot mode and the CPU is driven by the external oscillator (HPS_CLK1). No PLL is enabled.	External Oscillator, HPS_CLK1 (10 to 50 MHz)

Table 33. CSEL Encodings for hps_clk_f = 1

CSEL[3:0] Fuse Value	Description	MPU Clock Value
0x0	Bypass mode; in this mode all clocks are reset and boot mode is ensured active; `cb_intosc_hs_div2_clk` (`cb_intosc_hs` clock divided by 2) is used.	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x1	This encoding is typically used during a warm reset if you have already configured your clocks or PLL in the Clock Manager and would like to continue to use this configuration. If this is encoding is selected, no changes are made to these settings and the clock configuration is essentially user selected. If this is encoding is used during a power-on reset, then whatever the default reset values of the clocks are used.	User-selected clock source. Refer to the *Clock Manager* chapter for clock register settings.
0x2	PLL is used with the internal oscillator (30 to 100 MHz)	120 to 800 MHz
0x3	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x4	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x5	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x6	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x7	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x8	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x9	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0xA	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0xB	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)

continued...

CSEL[3:0] Fuse Value	Description	MPU Clock Value
0xC	Reserved	cb_intosc_hs_div2_clk (60 to 200 MHz)
0xD	Reserved	cb_intosc_hs_div2_clk (60 to 200 MHz)
0xE	Reserved	cb_intosc_hs_div2_clk (60 to 200 MHz)
0xF	Bypass mode; reset all clocks and ensure boot mode is active; cb_intosc_hs_div2_clk is used.	cb_intosc_hs_div2_clk (60 to 200 MHz)

For more information regarding the CSEL encodings and descriptions, please refer to the *Booting and Configuration* Appendix.

Related Information

- Clock Manager on page 52
  For more information about the Clock Manager.
- Booting and Configuration on page 771
  For more information about the boot process refer to the *Booting and Configuration* appendix.
- Intel Intel Arria 10 Device Datasheet
  For more information regarding clock characteristics, refer to the *Intel Intel Arria 10 Device Datasheet*.

7.1.3.6. FPGA Security Features

The FPGA has several security fuses which can control the following functions:

- Limiting acceptance of only encrypted and authenticated POFs
- Disabling partial reconfiguration and scrubbing
- Disabling test access
- Disabling external configuration
- Limiting JTAG access
- Bypassing HPS or FPGA JTAG
- Locking or disabling the battery-backed volatile key
- Disabling the OTP key

Note that there is a fuse sent to the FPGA CSS, called hps_jtag_bypass, which performs the same function of removing the HPS from the JTAG chain. However, this fuse is a permanent disable, whereas using software to program the sec_jtagdbg register has the same affect, but may be changed later by software.

7.1.3.7. Secure Boot

No ordering of boot or FPGA configuration is required because the FPGA and HPS may be brought up in any order, and they can both raise the security level of the entire device at any time before user code is loaded to execute.

The SoC may securely boot in one of three ways:

1. The HPS boot and FPGA configuration occur separately.
2. The HPS boots from the FPGA after the FPGA is configured
3. The HPS boots first and configures the FPGA.

Note: The device has the capability to execute standard non-secure boot in any of these three ways, as well. In this section, only secure booting mechanisms are reviewed.

Note: The FPGA must be powered on for the HPS to come out of reset properly.

Related Information

Booting and Configuration on page 771
For more information about the boot process refer to the *Booting and Configuration* appendix.

7.1.3.7.1. Boot Configuration

Booting configuration is provided from a combination of the BSEL pins and the boot mode fuses. The BSEL pins indicate if the boot source is the FPGA or one of the HPS flash interfaces. If the `fpga_boot_f` fuse is blown, it overrides these pin values. The `fpga_boot_f` fuse value can be read from the `hps_fusesec` register.

Table 34. BOOTSEL Field Values and Flash Device Selection

BOOTSEL Field Value	Flash Device
0x0	Reserved
0x1	FPGA (HPS-to-FPGA bridge)
0x2	1.8 V NAND flash memory
0x3	3.0 V NAND flash memory
0x4	1.8 V SD/MMC flash memory with external transceiver
0x5	3.0 V SD/MMC flash memory with internal transceiver
0x6	1.8 V quad SPI flash memory
0x7	3.0 V quad SPI flash memory

Within the device, there are fuses that provide some of the boot mode configuration information. The fuse values can be read through the HPS_fusesec register:

Table 35. HPS_fusesec Register Description

Bits	Name	Description
31:27	Reserved	Bit values in this field are undefined.
26:23	csel_f	This field indicates the value of the clock select fuses that are available for configuring the clock for the boot interface and for the PLLs. Refer to the *Clock Configuration* section for more information on CSEL encodings.
22	dbg_access_f	This fuse determines the initial state of the debug access domains.
21	dbg_lock_JTAG	This field indicates if the HPS JTAG access level can be changed through software when the HPS is released from reset. • 0x0= HPS JTAG access level can be changed through the sec_jtagdbg register. • 0x1= HPS JTAG access level cannot be changed (locked).
20	dbg_lock_DAP	This field indicates if the DAP access level can be changed through software when the HPS is released from reset. • 0x0= The DAP access level can be changed through the sec_dapdbg register. • 0x1= The DAP access level cannot be changed (locked).
19	dbg_lock_CPU0	This field indicates if the CPU0 debug access level can be changed through software when the HPS is released from reset. • 0x0= CPU0 debug access level can be changed through the sec_cpu0dbg register. • 0x1= CPU0 debug access level cannot be changed (locked).
18	dbg_lock_CPU1	This field indicates if the CPU1 debug access level can be changed through software when the HPS is released from reset. • 0x0= The CPU1 debug access level can be changed through the sec_cpu1dbg register. • 0x1= The CPU1 debug access level cannot be changed (locked).
17	dbg_lock_CS	This field indicates if the CoreSight debug access level can be changed through software when the HPS is released from reset. • 0x0= The CoreSight debug access level can be changed through the sec_csdbg register. • 0x1= The CoreSight debug access level cannot be changed (locked).
16	dbg_lock_FPGA	This field indicates if the FPGA debug access level can be changed through software when the HPS is released from reset. • 0x0= The FPGA debug access level can be changed through the sec_fpgadbg register. • 0x1= The FPGA debug access level cannot be changed (locked).
15:12	Reserved	Bit values in this field are undefined.

continued...

Bits	Name	Description
11	clr_ram_order_f	This fuse value determines how RAMs are cleared during a tamper event. • 0x0= All RAMs are cleared in parallel. • 0x1= All RAMs are cleared in series.
10	clr_ram_cold_f	This fuse value indicates what happens to the RAM on a cold reset. • 0x0= All RAMs are not cleared on a cold reset. • 0x1= All RAMs are cleared on a cold reset.
9	clr_ram_warm_f	This fuse value indicates what happens to the RAMs on a warm reset. • 0x0= All RAMs are not cleared on a warm reset. • 0x1= All RAMs are cleared on a warm reset.
8	oc_boot_f	This fuse value determines if the second-stage boot code is allowed to boot from on-chip RAM. • 0x0= Second-stage boot can be from on-chip RAM if enabled by the System Manager. • 0x1= Second-stage boot is not from on-chip RAM.
7	hps_clk_f	This fuse value selects the clock used for the boot process and in the case of a tamper event, memory scrambling. • 0x0= The external oscillator, HPS_CLK1, is used for boot. • 0x1= The internal oscillator, cb_intosc_ls_clk, is used for boot.
6	fpga_boot_f	If blown, this fuse value allows the FPGA to configure independently and allows the HPS to boot from an encrypted next-stage boot source that was decrypted into the FPGA. • 0x0= Booting is dependent on the BSEL pins. • 0x1= HPS only boots from the FPGA; BSEL options are ignored and CSEL fuse options are ignored.
5	aes_en_f	This fuse value indicates if a decryption of the flash image is always performed. • 0x0= An AES decryption of the flash image is determined from the second stage boot loader header. • 0x1= An AES decryption of the flash image is always performed.
4:2	kak_src_f	This bit field indicates the source of the Key Authorization Key (KAK) which can be in: • Proprietary ROM • FPGA logic elements • User fuses
1	kak_len_f	This fuse value indicates the Key Authorization Key (KAK) length: • 0x0= 256 bits • 0x1= 384 bits
0	authen_en_f	This fuse value determines whether authentication of flash images is required prior to execution. • 0x0= No authentication of the flash image is required prior to execution. • 0x1= HPS authentication of all flash images is required prior to execution.

7.1.3.7.2. Secure Boot Process

Secure boot allows the HPS to release from reset into a known state and then validate the authenticity and integrity of the second-stage boot loader code prior to executing it. Secure boot ensures that only authorized code is executed on the system. In addition, the system has the option to configure the FPGA from the HPS, which provides a secure boot mechanism for the FPGA portion of the SoC. In this mode, the HPS boot code can authenticate the FPGA POF prior to loading it.

The HPS and FPGA can also be booted securely but independently, with the HPS executing authenticated and decrypted user code out of external RAM and the FPGA receiving an encrypted FOP configuration file by enabling an FPGA configuration source through the MSEL pins.

Boot ROM code runs on CPU0 and CPU1 is held in reset. Once control passes to the second-stage boot loader, CPU1 can be released from reset.

7.1.3.7.3. Authentication and Decryption

Authentication is provided for the second-stage boot loader code and both the HPS and FPGA can utilize the AES algorithms in the Configuration Subsystem (CSS) to decrypt boot images and POF files, respectively.

Three levels of boot are available to the device:

- Authentication only: The second-stage boot loader code is not encrypted, but there are public key signatures attached to the image and the code only executes if all of the signatures pass. ECDSA256 (SHA 256) is used for authenticated boot.
- Decryption only: The user boot code is encrypted and must be decrypted before execution. AES-based algorithms in the FPGA are used for decryption.
- Authentication and Decryption: The user boot code is encrypted and signed.

If authentication and decryption are enabled, the data is first authenticated and then decrypted using the AES algorithms. Authentication is performed using the public key authorization key (KAK) held in the user fuses. The KAK can be 256 bits. The KAK public key authentication fuses are lockable by the user in groups of 64 bits or less.

The Security Manager's `hps_fusesec` register can be read to identify the source of the public (root) key used for authentication of the second-stage boot loader. The source of the public (root) key can be one of the following:

- External software
- ROM
- FPGA logic elements
- User Access Fuses (UAF)

AES decryption algorithms are available through the FPGA CSS and are enabled prior to decryption through a combination of user fuses and software programming. Elliptic curve cryptographic algorithms are used in an authenticated boot. Decrypted data is sent to the Security Manager to be read by the CPU or DMA.

The FPGA has higher priority in the CSS, and the FPGA AES algorithms abort decryption and authentication of data if the FPGA starts configuration or reconfiguration.

7.1.3.8. Anti-Tamper

The Security Manager receives a tamper detection signal when a tamper event is detected in the device.

Anti-tamper logic can be implemented in the FPGA soft logic to detect a tamper event. No tamper detection is available until the FPGA is configured. The user must define the anti-tamper design. Some examples of tamper detection that could be implemented are:

- Using the FPGA voltage sensor to detect any IR drops across the device.
- Using an A/D controller to detect temperature changes in the device due to tamper.
- Assigning loaner I/O as dedicated tamper pins to detect package break-away.

The tamper detection signal sent to the Security Manager, `anti_tamper_in`, is an active low signal. When there is no tamper event, it is driven to 1. The Security Manager also provides a tamper acknowledge signal to the FPGA interface. The tamper event input to the HPS is gated with a signal indicating the completion of FPGA configuration. This prevents an inadvertent tamper event assertion or tamper response from the HPS to the rest of the system.

When the Security Manager receives a tamper detection assertion, it prepares the system to go into reset and notifies the Reset Manager of the event. When the Reset Manager receives the tamper event assertion from the Security Manager, it drives all the peripherals and MPU into reset. It asserts a request and enables all peripherals memories to initiate a memory scramble and a memory clear, either in series or parallel, depending on the information sent from the Security Manager. The memory clearing includes the on-chip RAM, L1 caches, and L2 cache and RAM for the peripherals listed below:

- Ethernet MAC 0/1/2
- USB OTG 0/1
- QSPI flash controller
- NAND flash controller
- SD/MMC controller
- DMA controller

Once the memory scramble and clear completes, each module sends an acknowledgment back to the Reset Manager.

Upon completion of memory scrambling of all IPs, the Security Manager sends a response to the FPGA indicating that a tamper event has been handled in the HPS. This response is followed by HPS entering into a cold reset without exiting until another power cycle occurs. Both Security Manager and Reset Manager enter reset. Other resets, such as warm reset or cold reset, do not affect the state of the Security Manager or the Reset Manager.

If the FPGA POR is de-asserted while the HPS is in a tamper event reset, then the Security Manager goes back to the reset state.

7.2. Security System Extensions

Beyond the Security Manager module, other secure features can be implemented at a system level within the SoC.

7.2.1. TrustZone

The Cortex-A9 MPU subsystem has integrated TrustZone technology, which provides a system solution to protect application platforms from malicious attack. The TrustZone hardware and supporting software are designed to provide a strong security solution regardless of the operating environment. TrustZone creates a separation between the secure and non-secure areas of the SoC and allows the designer to choose which assets in a design are designated as secure and non-secure.

TrustZone security is implemented in the Cortex-A9 MPU subsystem in three ways:

- Hardware partitioning through the implementation of firewalls: Resources can be assigned and identified as secure or non-secure.
- Virtual processor execution: Each core can context switch between secure and non-secure operation.
- Secure debug control: Both secure and non-secure hardware debug exist and the type of debug allowed can be configured by the user.

7.2.1.1. Secure Partitioning

System interconnect and cache accesses as well as processor execution can be securely partitioned using the TrustZone infrastructure.

7.2.1.1.1. Secure Bus Accesses

The Cortex-A9 MPCore and other masters that utilize the system interconnect can be configured for secure or non-secure accesses.

Master accesses drive a protection signal, AxPROT[1], to communicate the security level of the attempted transaction. The Cortex-A9 drives this signal low for secure accesses and high for non-secure accesses. The secure firewalls determine whether the access is allowed or not. A slave access type defaults to secure if no other configuration is programmed. Users can define whether a bus error is generated or if a bus failure occurs when a secure access is violated.

Note: Secure processor configuration is programmed through the CP15 register.

7.2.1.1.2. Secure Cache Accesses

Secure and non-secure partitioning also extends to the L1 and L2 caches.

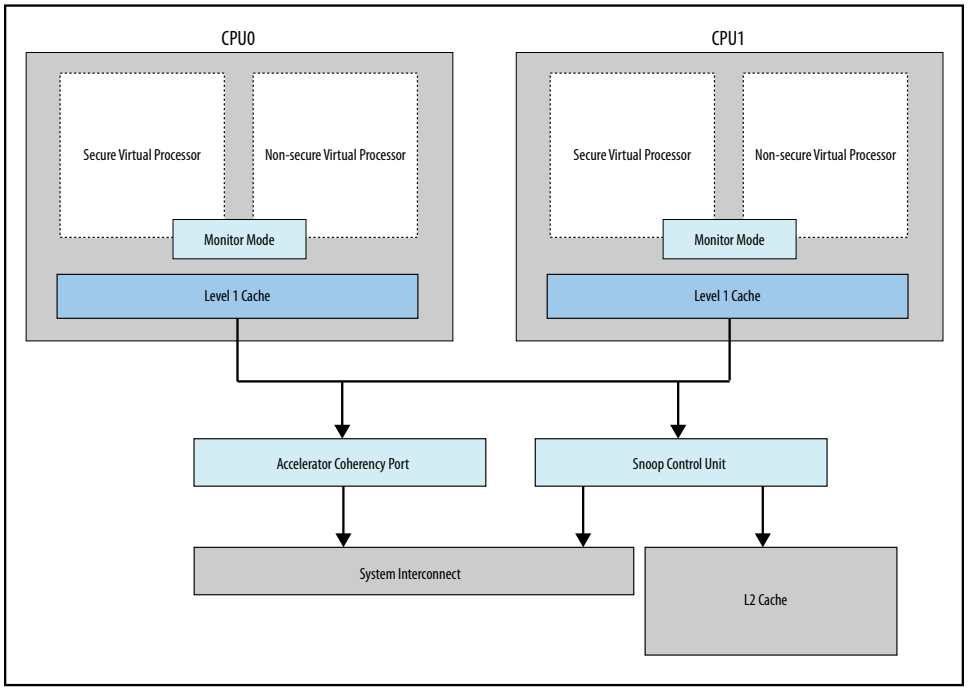
There is a 32-bit physical address space for secure and non-secure transactions and a 33rd bit is provided to indicate security. The cache is able to store both secure and non-secure lines.

The Accelerator Coherency Port (ACP) in the ArmCortex-A9 MPCore can be used with TrustZone technology. The security state of the masters using the ACP must be the same as the Cortex-A9 processor.

7.2.1.2. Virtual Processor Operation

Two virtual processors (secure and non-secure) with context switch capability (monitor mode) exist for each Cortex-A9 processor core. The secure virtual processor only accesses secure resources and the non-secure virtual processor only accesses non-secure resources.

Figure 20. Virtual Processor Environment with Monitor Mode



A context switch to secure operation is achieved through a secure monitor call (SMC) instruction or the following hardware exceptions (if configured to do so):

- IRQ interrupt
- Faster Interrupt Request (FIQ) interrupt
- External data abort
- External prefetch abort

When a context switch occurs, the state of the current mode is saved and restored on the next context switch or on a return from exception.

Exception Vector Tables

Three exception vector tables are provided for the MPU with TrustZone:

1. Non-secure
2. Secure
3. Monitor mode

Typically IRQs are used for non-secure interrupts and FIQs are used for secure interrupts. The location of each of the three vector tables can be moved at runtime.

The Generic Interrupt Controller (GIC) can handle secure and non-secure interrupts and prevent non-secure accesses from reading or modifying the configuration of a secure interrupt. An interrupt can be made secure by programming the appropriate bits in the Interrupt Security Register. Secure interrupts are always given a higher priority than non-secure interrupts.

7.2.2. Arria 10 HPS Secure Firewalls

You can use the system interconnect firewalls to enforce security policies for slave and memory regions in the system interconnect.

The firewalls support the following features:

- Secure or non-secure access configurable per peripheral
- Privileged or user access configurable for some peripherals
- Per-transaction security

Each firewall contains the security configuration registers (SCRs) that set security policies and define which transactions are allowed to go through the firewall. If you want to generate an error any time a transaction is blocked in the firewall, then you must set the `error_response` bit in the `global` register of the `noc_fw_ddr_l3_ddr_scr` module at base address 0xFFD13400. If this bit is clear, transactions blocked by the firewall return random data.

Note: Future devices may not support the return of random data and may only support an error response for blocked firewall transactions. For designs that may be ported to future devices, Intel recommends you to set the `error_response` bit in the `global` register.

There are five main firewalls in the HPS:

- Peripheral
- System
- HPS-to-FPGA
- On-Chip RAM
- SDRAM (which includes DDR and DDR L3 firewalls)

7.2.2.1. Firewall Diagrams

The system interconnect firewalls filter access to components on various buses.

Table 36. System Interconnect Firewalls

Name	Function
Peripherals	The peripherals firewall filters access to slave peripherals in the following buses: • L4 main bus • L4 master peripheral bus • L4 AHB bus • L4 slave peripherals bus
System	The system firewall filters access to system peripherals in the following components: • L4 system bus • L4 ECC bus • DAP
HPS-to-FPGA	The HPS-to-FPGA firewall filters access to FPGA through the following bridges: • HPS-to-FPGA bridge • Lightweight HPS-to-FPGA bridge
On-Chip RAM	The on-chip RAM firewall filters secure access to on-chip RAM
SDRAM	The SDRAM firewall filters access to DDR SDRAM

Figure 21. Peripheral Firewall

Security Configuration Registers (SCRs) within the `noc_fw_l4_per` register group can be programmed to mark the security status of all available masters.

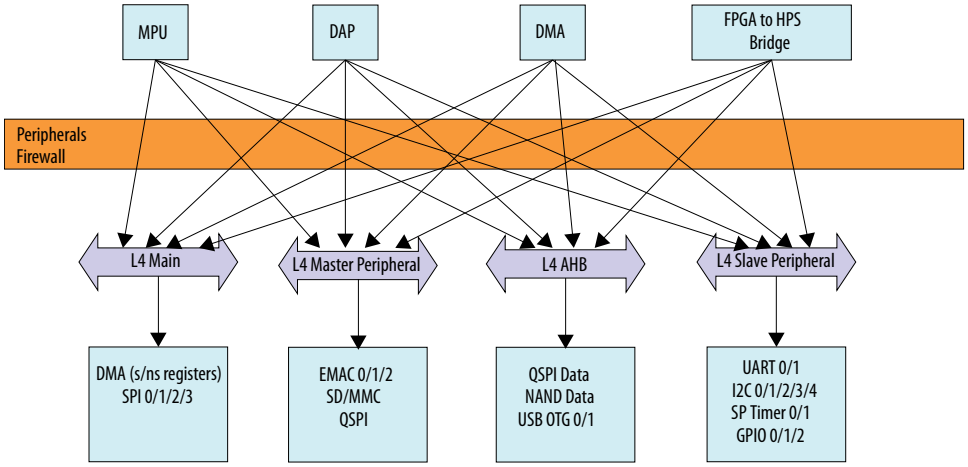


Table 37. Peripheral Firewall

Bus Behind Firewall	Bus Slaves Behind Firewall	Masters That Can Access Bus
L4 Main	DMA (secure/non-secure registers) SPI 0/1/2/3	MPU DAP DMA FPGA-to-HPS
L4 Master Peripheral Bus	EMAC 0/1/2 SD/MMC QSPI	MPU DAP DMA

continued...

Bus Behind Firewall	Bus Slaves Behind Firewall	Masters That Can Access Bus
		FPGA-to-HPS
L4 AHB Bus	QSPI Data NAND Data USB OTG 0/1	MPU DAP DMA FPGA-to-HPS
L4 Slave Peripheral Bus	UART 0/1 I2C 0/1/2/3/4 SP Timer 0/1 GPIO 0/1/2	MPU DAP DMA FPGA-to-HPS

Figure 22. System Firewall

The system firewall filters accesses to all system peripherals. The system firewall policies can be programmed through the SCRs of the `noc_fw_soc2fpga` register group.

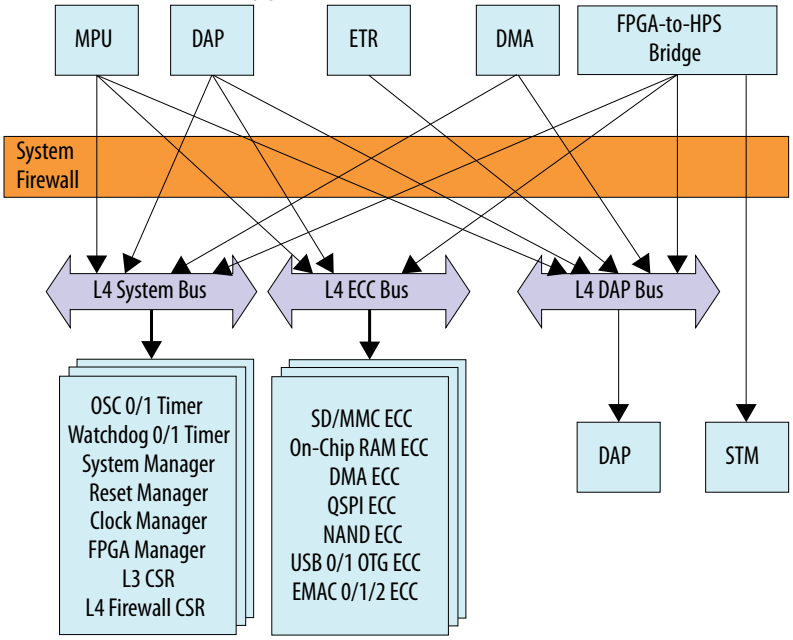


Table 38. System Firewall

Bus Behind Firewall	Bus Slaves Behind Firewall	Masters That Can Access Bus
L4 ECC	SD/MMC ECC DMA ECC QSPI ECC NAND ECC USB 0/1 ECC On-Chip RAM ECC EMAC 0/1/2 Rx ECC EMAC 0/1/2 Tx ECC NAND Read/Write ECC	MPU FPGA-to-HPS DAP
L4 System	FPGA Manager Data and Registers OSC Timer 0/1 Watchdog 0/1	MPU FPGA-to-HPS DAP

continued...

Bus Behind Firewall	Bus Slaves Behind Firewall	Masters That Can Access Bus
	System Manager L3 Interconnect Control and Status Registers (CSR) L4 Interconnect firewall Security Control Registers (SCR)	DMA
L4 DAP Bus	STM DAP	MPU FPGA-to-HPS DAP

Figure 23. HPS-to-FPGA Firewall

SCRs within the noc_fw_soc2fpga register group can be programmed to configure the security policy for the master-to-bridge pairs.

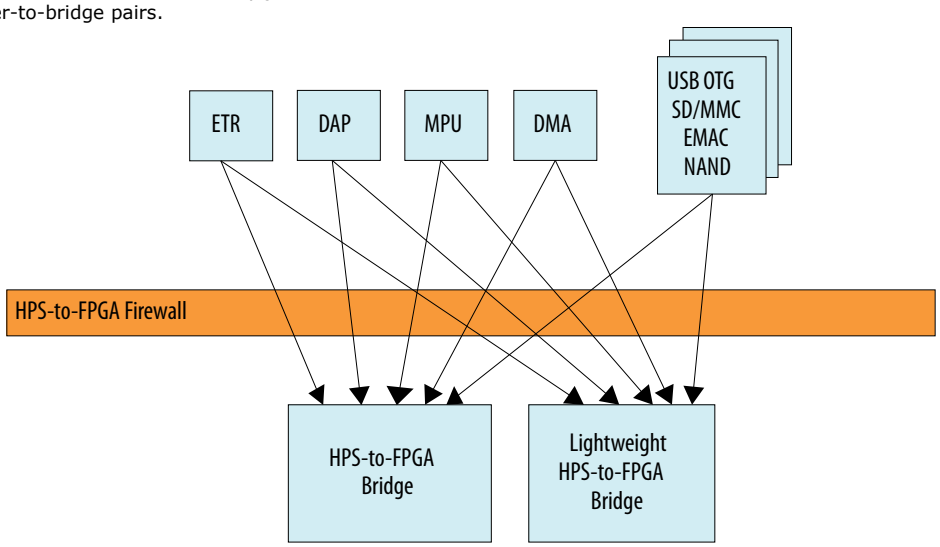


Table 39. System Firewall

Bus Behind Firewall	Bus Slaves	Masters that can Access Bus
L3	SD/MMC ECC DMA ECC QSPI ECC NAND ECC USB 0/1 ECC On-Chip RAM ECC EMAC 0/1/2 Rx ECC EMAC 0/1/2 Tx ECC NAND Read/Write ECC	MPU FPGA-to-HPS DAP
L4 System	FPGA Manager Data and Registers OSC Timer 0/1 Watchdog 0/1 System Manager L3 Interconnect CSR L4 Interconnect firewall SCR	MPU FPGA-to-HPS DAP DMA
L4 DAP Bus	STM DAP	MPU FPGA-to-HPS DAP

Figure 24. On-Chip RAM Firewall

Up to six regions of the on-chip RAM can be partitioned for non-secure accesses.

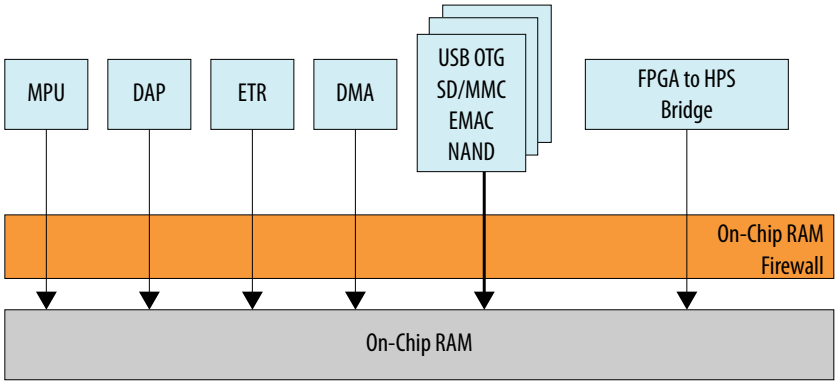
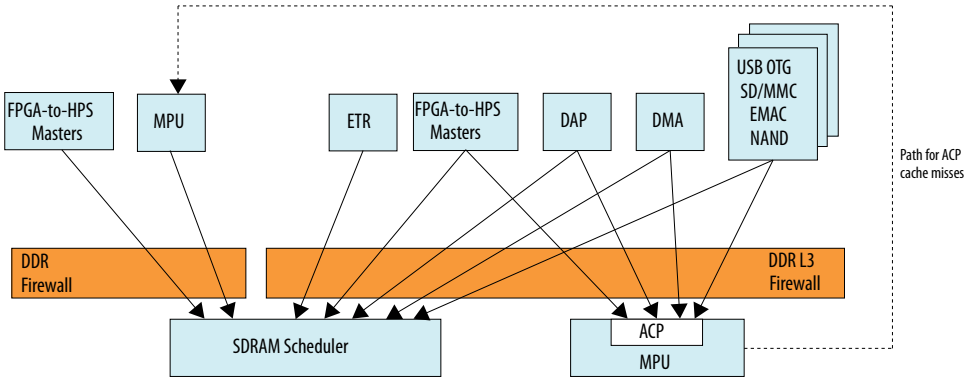


Figure 25. SDRAM firewall

Transactions from the DMA, ETR, DAP, FPGA-to-HPS masters or HPS masters (USB, SD/MMC, EMAC, NAND) are routed to either the ACP or the SDRAM scheduler depending on the cacheable bit. Only cacheable transactions are routed to the ACP, with non-cacheable transactions being routed directly to the SDRAM scheduler. If an access to the ACP results in a cache miss in the L1 and L2 cache systems, then the MPU master M1 issues a transaction to the SDRAM scheduler. Only cacheable transactions go through to the ACP. If a cache miss occurs on the ACP cycle, then the MPU masters a new transaction to the SDRAM scheduler. Accesses from the MPU or FPGA-to-SOC masters that are tightly coupled to the MPU go through the DDR firewall only.

Note: The dotted line in the diagram represents the path taken when an ACP access occurs and misses in the L1 and L2 cache.



7.2.2.2. Secure Transaction Protection

The interconnect provides two levels of transaction protection.

The two levels of transaction protection are:

- Security Firewalls:

  Security firewalls enforce the Secure World read and write transactions. The term "security firewall" and "firewall" may be used interchangeably within the context of this section.

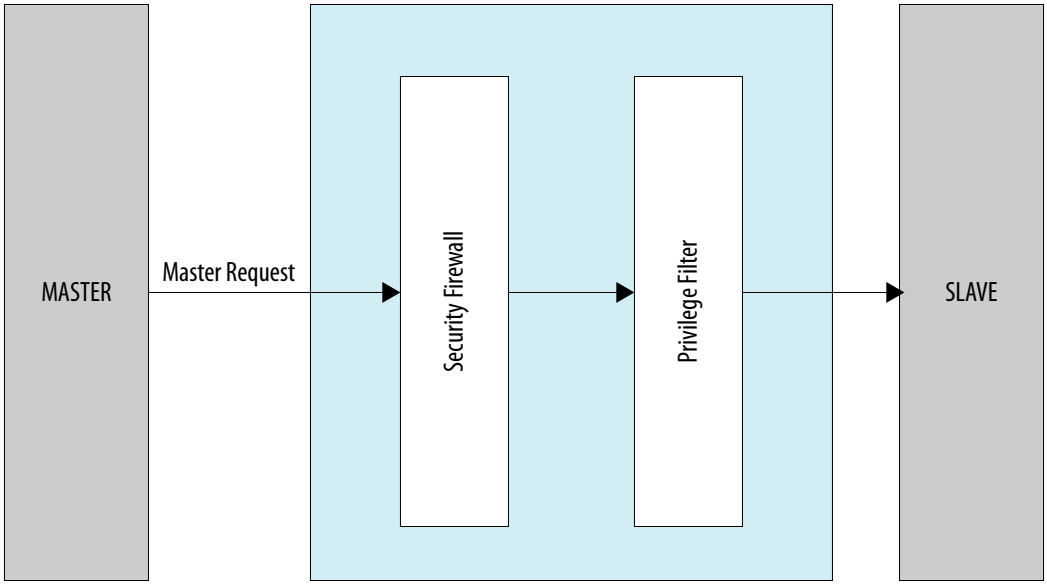
- Privilege filter

  The privilege filter leverages the firewall mechanism and provides additional filtering by allowing control of the privilege level of L4 slave transactions. The privilege filter applies to writes only.

All slaves on the SoC are placed behind a security firewall. A subset of slaves are also placed behind a privilege filter. Transactions to these slaves pass through two firewall structures as shown in the figure below. The transaction packet must pass both a security firewall and the privilege filter to reach the slave. A transaction error causes random data or a slave error depending on where the transaction failed and how the `error_response` bit in the `global` register of the `noc_fw_ddr_l3_ddr_scr` module is programmed.

Note: Future devices may not support the return of random data and may only support an error response for blocked firewall transactions. For designs that may be ported to future devices, Intel recommends you to set the `error_response` bit in the `global` register.

Figure 26. Successful Secure Transaction Path



Each firewall has an associated firewall register which can be programmed by the DAP, FPGA fabric or MPU in secure mode only. A subset of the L4 slaves can additionally be programmed to configure their privilege filter policy.

7.2.2.2.1. Peripheral and System Firewalls

The peripheral and system firewalls each have a group of Security Configuration Registers (SCRs) that can be programmed by the DAP, FPGA fabric or MPU only in secure mode.

The peripheral and system firewalls can be configured by programming the security bits in the SCR blocks. Each slave has an SCR containing a security bit for each available master, which enables a different security access level to be programmed for each master-slave pair. At reset, all accesses are configured to be secure. When a master-slave security bit is 0, only secure packets are allowed to pass the firewall and reach the intended target. When the master-slave security bit is 1, the firewall passes the transaction.

Table 40. Security Bit Value

Secure Flag Value	Access
0	Only secure packets are allowed to pass the firewall.
1	Transaction packets are allowed to pass the firewall.

When a transaction packet is sent from a master, the master also drives a secure flag signal on the bus. This flag indicates whether the attempted transaction from the master is secure or non-secure. For AXI master accesses, this flag is the ~AxPROT[1] signal.

For L4 ECC, L4 system or L4 DDR master accesses, this flag is the MSecure[0] signal.

When the flag signal is driven high, it indicates a secure access. When the flag signal is low, it indicates a non-secure access.

Secure Transaction Example

A successful secure access attempt by a master, depends on the security bit configuration in the Security Control Register and the value of the master secure flag sent by the master.

All SoC transactions are set to secure out of reset with a return of random data when transactions are blocked. If you want an error response returned for blocked transactions, you must set the `error_response` bit in the `global` register of the `noc_fw_ddr_l3_ddr_scr` module at base address 0xFFD13400. If this bit is clear, transactions blocked by the firewall return random data.

Note: Future devices may not support the return of random data and may only support an error response for blocked firewall transactions. For designs that may be ported to future devices, Intel recommends you to set the `error_response` bit in the `global` register.

To initiate a secure transaction:

1. The master drives the address of the slave as well as the master secure flag signal, indicating that the access is intended to be secure.
2. The firewall compares the programmed security bit of the master-slave peripheral pair to the master secure flag signal to determine if the access is valid.
3. If the access is valid, the transaction passes to the next level. If the access is not valid, random data is fed back to the master.

Table 41. Peripheral and System Transaction Results

Security Bit in Firewall Register	Master Secure Flag	Transaction Pass/Fail	Transaction Type
0	0	No	None (Blocked transaction)
0	1	Yes	Secure
1	0	Yes	Non-secure
1	1	Yes	Secure

Non-secure Transaction Configuration

When the SoC is released from reset, only secure transactions are allowed through the firewalls. The user must configure any master-slave pairs that are required to be non-secure.

The SCR registers must be accessed in secure mode to configure the firewall access to non-secure.

1. After the SoC is released from reset, the MPU, FPGA2SOC masters or the DAP accesses the required Security Configuration Register (SCR) by executing a secure write. The peripheral firewall SCR registers are within the `soc_noc_fw_l4_per` register group and the system firewall SCR registers are found within the `soc_noc_fw_l4_sys` register group.
2. To configure the slave to allow a non-secure access by a master, set the corresponding security bit to 1.

The configured master is able to successfully execute a non-secure read or write to the configured slave.

7.2.2.2.2. HPS-to-FPGA Firewall

The HPS-to-FPGA firewall is used to filter access to all the FPGA, including the HPS-to-FPGA bridge and the lightweight HPS-to-FPGA bridge.

The secure firewall for the HPS-to-FPGA can be disabled by programming the `lwsoc2fpga` register and `soc2fpga` register within the `noc_fw_soc2fpga` group.

7.2.2.2.3. On-Chip RAM Firewall

At reset all memories are secure. Out of reset, regions of memories can be configured for non-secure accesses.

The on-chip RAM is divided into six regions, where the granularity of the address boundary is 4 KB. Within each region, a non-secure or shared memory region can be assigned by programming a `base` and `limit` value in the corresponding `regionNaddr` register, with `N` denoting the region number. Each of these regions can be enabled by writing to the `enable` register or setting the corresponding bit in the `enable_set` register.

When an incoming transaction falls within any enabled non-secure regions, the firewall allows both secure and non-secure packets. When the transaction is outside of any enabled regions, the firewall only allows secure packets.

When a transaction packet is sent from a master, the master also drives a secure master flag signal on the bus. This flag indicates whether the attempted transaction from the master is for a secure or non-secure memory region. When the flag signal is driven high, it indicates a secure access. When the flag signal is low it indicates a non-secure access.

On-Chip RAM Transaction Configuration

Because all memories are secure when they exit reset, the on-chip RAM Security Control Register block must be configured by a secure master before non-secure or shared memory accesses are allowed.

1. Configure the `base` and `limit` fields of the `regionNaddr` register, where `N` denotes the region number.
2. Enable the non-secure memory region by setting the corresponding `regionNenable` bit in the `enable_set` register, where `N` denotes the region number.

7.2.2.2.4. SDRAM Firewall

Two SDRAM firewalls are implemented to allow a different security policy depending on the master.

One firewall filters accesses from the MPU and the FPGA-to-SDRAM interface. The other firewall filters accesses from all other masters on the HPS and applies the same security policy to both coherent and non-coherent accesses. Both firewalls are logically and physically split within the interconnect.

The SDRAM Firewall supports a minimum region size of 64 KB. Each master is allowed a different number of firewall regions:

- The MPU can have up to four firewall regions.
- The FPGA-to-SDRAM interface can have up to twelve firewall regions.
- Any HPS bus master can have up to eight firewall regions.

7.2.2.3. Privilege Filter

If a transaction packet has passed the security firewall, it may pass through a privilege filter. The privilege filter only applies to writes. All reads are passed without exception.

The privilege filter decodes the incoming user or privilege accesses and determines whether to pass or fail the transaction. It is separate from the security firewall and transactions generally carry both privilege and secure bits. The privilege filters are configured in the interconnect by executing a read-modify-write to the `l4_priv` register or setting individual bits in the `l4_set` register. If a privilege bit is set, both privilege and user mode transactions are allowed to the slave. If a privilege bit is cleared using the `l4_clear` register, then only privileged transactions are allowed to the slave.

The following slaves can be programmed using the privilege registers in the interconnect module:

- HPS-to-FPGA bridge
- Lightweight HPS-to-FPGA bridge
- UARTs
- SP Timers
- I2C Modules
- GPIO
- SD/MMC

- QSPI
- EMAC Modules
- SPI
- Secure and non-secure DMA
- USB
- NAND Controller

The following table shows the result of privileged and user master accesses based on the value of the programmed privilege bit for a particular slave.

Note: All read accesses pass regardless of the privilege bit value.

Table 42. Privilege Filter Transaction

Read/Write	Privilege Signal	Privilege bit	Transaction Pass/Fail
Read	0	0	Pass
	0	1	Pass
	1	0	Pass
	1	1	Pass
Write	0	0	Fail
	0	1	Pass
	1	0	Pass
	1	1	Pass

7.2.2.4. Master Security Policy

Each master has an inherent security transaction capability.

Masters accessing slaves can be configured to one of three different security policies:

- Per transaction: The master is capable of generating secure and non-secure transactions.
- Secure: The master only supplies secure transactions.
- Non-secure: The master only generates non-secure transactions.

At reset, all accesses default to secure transactions. The table below details the transaction capability of each master within the SoC. Some masters are only capable of non-secure transactions.

Table 43. Master Transaction Capability

Master	Transaction Capability
DMA	Secure/Non-secure
DAP	Secure/Non-secure
USB OTG 0/1	Non-secure
SD/MMC	Non-secure
EMAC0/1/2	Secure/Non-secure

continued...

Master	Transaction Capability
NAND	Non-secure
FPGA-to HPS Bridge	Secure/Non-secure
ETR	Secure/Non-secure
MPU	Secure/Non-secure
FPGA-to-SDRAM	Secure/Non-secure

Security policies are based on secure and privilege attributes. For instance, if CPU0 is configured to access NAND registers in both secure and non-secure mode and CPU0 attempts an access when the core is in secure or non-secure mode, no error occurs. However, if CPU0 is allowed to access NAND registers only in secure mode and CPU0 is operating in non-secure mode, then CPU0 receives an error response when accessing the NAND registers. If both the security firewall and privilege firewall are implemented, security firewall filters all of the accesses. If an access fails, random data or an error response is sent to the master, depending on how the `error_response` bit in the `global` register of the `noc_fw_ddr_l3_ddr_scr` module is programmed. If access is granted by the security firewall, then the transaction enters the privilege firewall. If access is granted, the request enters the peripheral IP.

Note: Future devices may not support the return of random data and may only support an error response for blocked firewall transactions. For designs that may be ported to future devices, Intel recommends you to set the `error_response` bit in the `global` register.

7.2.2.5. Secure Interconnect Reset

When the SoC is released from reset, every slave on the interconnect is in the secure state (boot secure).

The only masters capable of secure accesses out of reset are:

- MPU
- DAP
- HPS-to-FPGA fabric

8. System Interconnect

The components of the hard processor system (HPS) communicate with one another, and with other portions of the SoC device, through the system interconnect. The system interconnect consists of the following blocks:

- The main level 3 (L3) interconnect
- The SDRAM L3 interconnect
- The level 4 (L4) buses

The system interconnect is the main communication bus for the MPU and all IPs in the SoC device.

The system interconnect is implemented by the Arteris FlexNoC network-on-chip (NoC) interconnect module.

The system interconnect supports the following features:

- An L3 interconnect that provides high-bandwidth routing between masters and slaves in the HPS
- SDRAM L3 interconnect, providing access to a hard memory controller in the FPGA fabric through a multiport front end (MPFE) scheduler for sharing the external SDRAM between multiple masters in the SoC device
- Independent L4 buses running in several clock domains. These buses handle data traffic for low- to mid-level bandwidth slave peripherals and accesses to peripheral control and status registers throughout the address map.
- On-chip debugging and tracing capabilities
- Security firewalls with the following capabilities:
  - Secure or nonsecure access configured per peripheral
  - Privileged or user access configured per peripheral (for some peripherals)
  - Security optionally configured per transaction at the master
- Quality of service (QoS) with three programmable levels of service on a per-master basis

Related Information

- www.arteris.com
  For information about the FlexNoC Network-on-Chip Interconnect, refer to the Arteris website.
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

8.1. About the System Interconnect

8.1.1. Features of the System Interconnect

The system interconnect supports high-throughput peripheral devices. The system interconnect has the following characteristics:

- Byte oriented address handling
- Data bus width up to 128 bits
- Arm TrustZone-compliant firewall and security support, with additional security features configurable per master
- Programmable quality-of-service (QoS) optimization
- Dedicated SDRAM L3 interconnect, providing access and scheduling for the hard memory controller in the FPGA portion of the SoC device
- On-chip debug and tracing capabilities
- Multiple independent L4 buses with independent clock domains and protocols

  The L4 buses are each connected to a master in the main L3 interconnect for control and status register access. Each L4 bus is 32 bits wide and is connected to multiple slaves. The L4 buses also provide a low- to mid-level tier of the system interconnect for lower-bandwidth slave peripherals. Each L4 bus is 32 bits wide and operates in a separate clock domain.

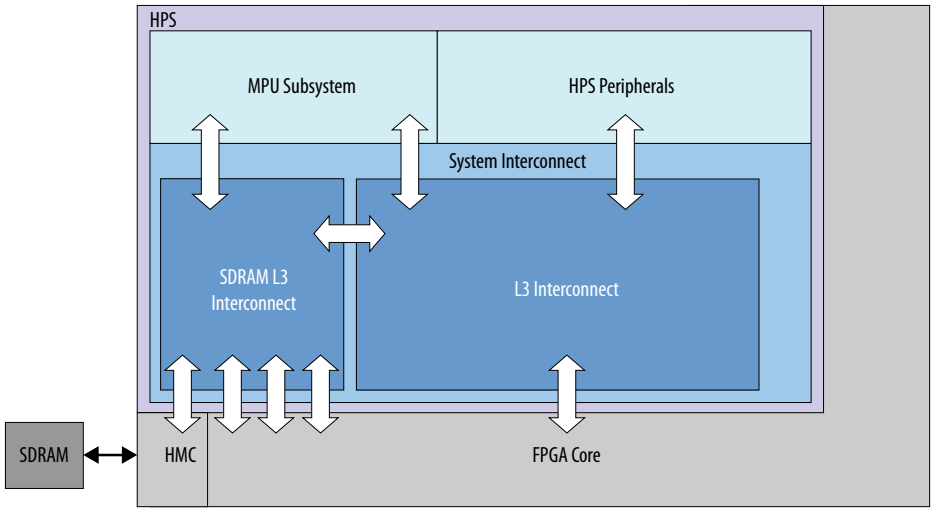
8.1.2. System Interconnect Block Diagram and System Integration

8.1.2.1. System Interconnect Block Diagram

The following figures show the system interconnect, including the main L3 interconnect, SDRAM L3 interconnect, and L4 buses.

Figure 27. High-Level System Interconnect Block Diagram

The following figure shows the relationships among the system interconnect and other major SoC components.



Related Information

Arria 10 HPS Master-to-Slave Connectivity Matrix on page 202

8.1.2.2. Connectivity

8.1.2.2.1. Arria 10 HPS Master-to-Slave Connectivity Matrix

The system interconnect is a highly efficient packet-switch network.

Each system interconnect packet carries a transaction between a master and a slave. The following figure shows the connectivity of all the master and slave interfaces in the system interconnect.

Figure 28. Master-to-Slave Connectivity

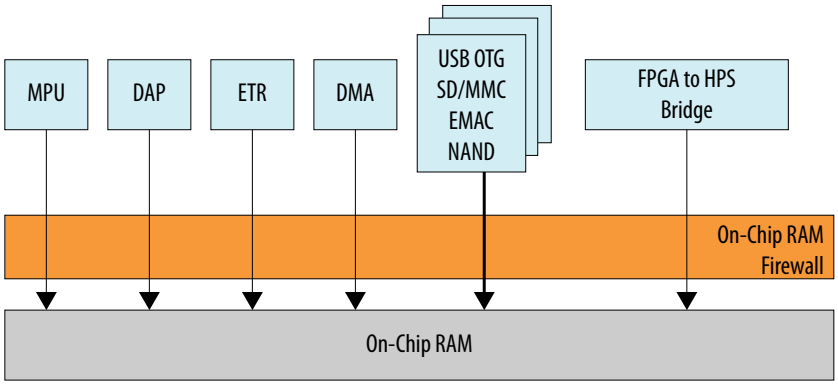
Masters	Slaves														
	L4 Main Bus Slaves	L4 MP Bus Slaves	L4 AHB Bus Slaves	L4 SP Bus Slaves	L4 SYS Bus Slaves	L4 ECC Bus Slaves	DAP	STM	On-Chip RAM	Boot ROM	ACP	DDR	Lightweight HPS-to-FPGA Bridge	HPS-to-FPGA Bridge	Service Network
MPU L2 Cache Master 0	✓	✓	✓	✓	✓		✓	✓	✓	✓			✓	✓	✓
MPU L2 Cache Master 1												✓			
FPGA-to-HPS Bridge	✓	✓	✓	✓	✓(1)	✓	✓	✓	✓		✓	✓			✓
FPGA-to-SDRAM Interface												✓			
DMA	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓	
EMAC 0/1/2									✓		✓	✓	✓	✓	
USB 0/1									✓		✓	✓	✓	✓	
NAND									✓		✓	✓	✓	✓	
SD/MMC									✓		✓	✓	✓	✓	
ETR							✓		✓			✓	✓	✓	
DAP	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓

Note:
1. The FPGA-to-HPS Bridge is connected to all L4 system slaves except the FPGA manager.

8.1.2.2.2. On-Chip RAM

Figure 29. On-Chip RAM Connections

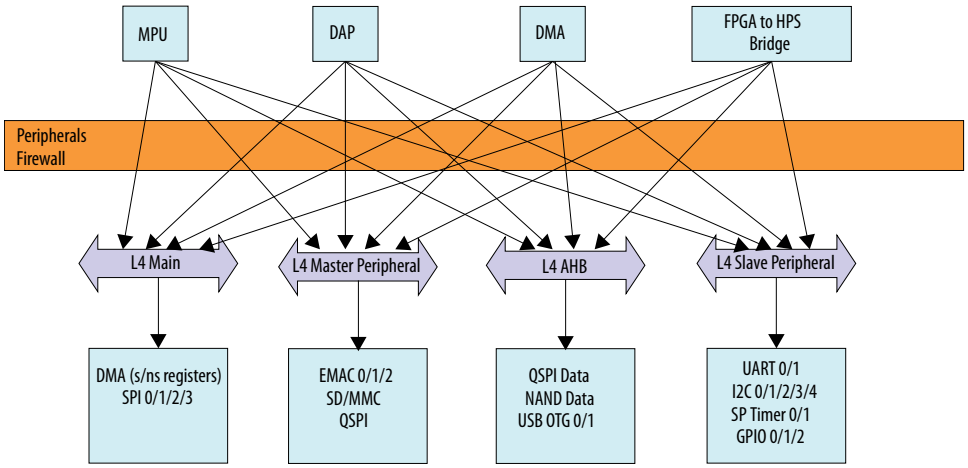
HPS masters connecting to the on-chip RAM



8.1.2.2.3. Peripherals Connections

Figure 30. Peripherals Connections

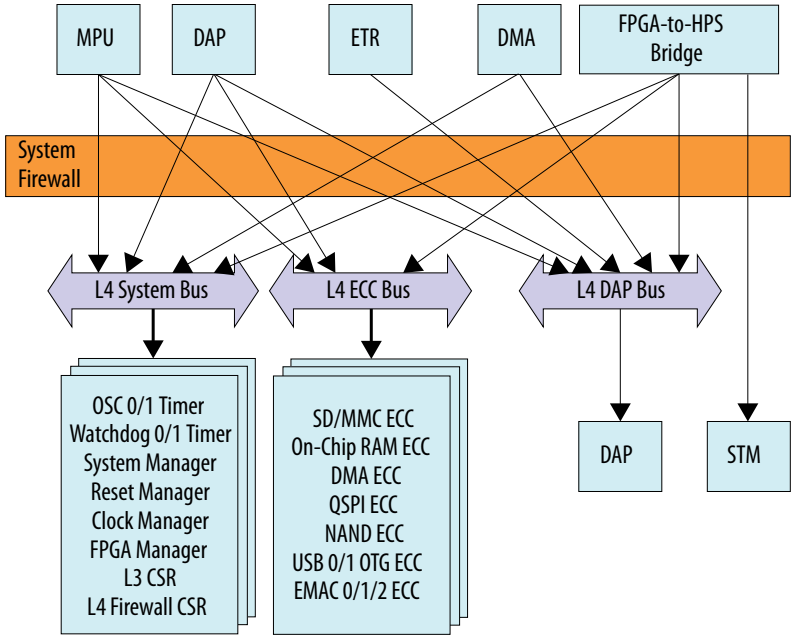
HPS masters connecting to HPS peripherals



8.1.2.2.4. System Connections

Figure 31. System Connections

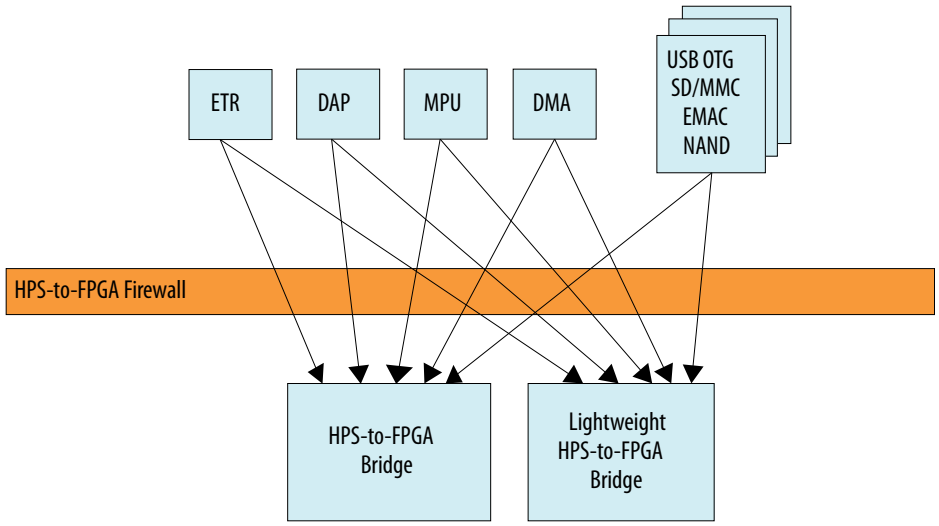
HPS masters connecting to HPS system peripherals



8.1.2.2.5. HPS-to-FPGA Bridge

Figure 32. HPS-to-FPGA Bridge Connections

HPS masters connecting to the HPS-to-FPGA bridges



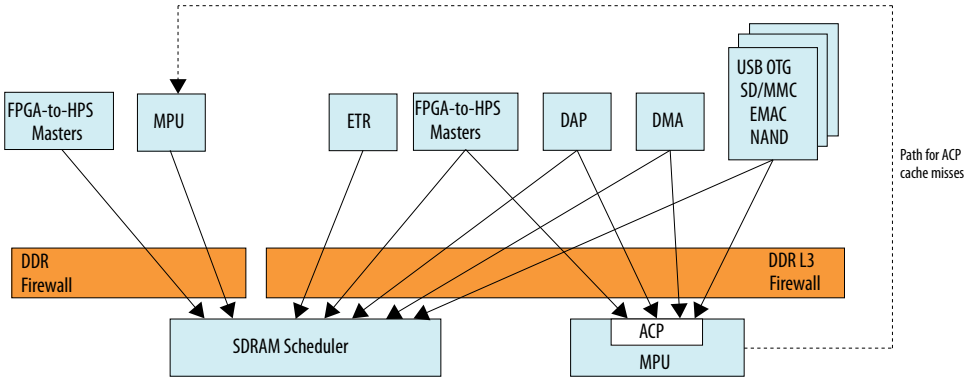
Related Information

HPS-FPGA Bridges on page 248
For more information, refer to the *HPS-FPGA Bridges* chapter.

8.1.2.2.6. SDRAM Connections

Figure 33. SDRAM Connections

HPS masters connecting to the SDRAM scheduler. Note that cache misses to the MPU Accelerator Coherency Port (ACP) pass through a secondary firewall.



Related Information

SDRAM L3 Interconnect Block Diagram and System Integration on page 0
More detailed information about the SDRAM L3 interconnect

8.1.2.3. System Interconnect Architecture

The system interconnect has a transaction-based architecture that functions as a partially-connected fabric. Not all masters can access all slaves.

The system interconnect is a packet-oriented network on chip (NoC), in which each packet carries a transaction between a master and a slave. The interconnect provides interface widths up to 128 bits, connecting to the L4 slave buses and to HPS and FPGA masters and slaves.

The system interconnect provides low-latency connectivity to the following bridges:

- HPS-to-FPGA bridge
- Lightweight HPS-to-FPGA bridge
- FPGA-to-HPS bridge
- Three FPGA-to-SDRAM ports

8.1.2.3.1. SDRAM L3 Interconnect Architecture

The SDRAM L3 interconnect provides access to the hard memory controller in the FPGA portion of the SoC device.

The SDRAM L3 interconnect is part of the system interconnect, and includes these components:

- SDRAM scheduler
- SDRAM adapter

The SDRAM L3 interconnect operates in a clock domain that is 1/2 the speed of the hard memory controller clock.

8.1.3. Arria 10 HPS Secure Firewalls

You can use the system interconnect firewalls to enforce security policies for slave and memory regions in the system interconnect.

The firewalls support the following features:

- Secure or non-secure access configurable per peripheral
- Privileged or user access configurable for some peripherals
- Per-transaction security

Each firewall contains the security configuration registers (SCRs) that set security policies and define which transactions are allowed to go through the firewall. If you want to generate an error any time a transaction is blocked in the firewall, then you must set the `error_response` bit in the `global` register of the `noc_fw_ddr_l3_ddr_scr` module at base address 0xFFD13400. If this bit is clear, transactions blocked by the firewall return random data.

Note: Future devices may not support the return of random data and may only support an error response for blocked firewall transactions. For designs that may be ported to future devices, Intel recommends you to set the `error_response` bit in the `global` register.

There are five main firewalls in the HPS:

- Peripheral
- System
- HPS-to-FPGA
- On-Chip RAM
- SDRAM (which includes DDR and DDR L3 firewalls)

8.1.4. About the Rate Adapters

The L3 interconnect contains a rate adapter wherever a low-bandwidth channel transfers data to a high-bandwidth channel.

Related Information

Functional Description of the Rate Adapters on page 225

8.1.5. About the SDRAM L3 Interconnect

The hard processor system (HPS) provides a specialized SDRAM L3 Interconnect dedicated to SDRAM accesses.

The SDRAM L3 Interconnect contains an SDRAM scheduler and an SDRAM adapter. The SDRAM scheduler functions as a multi-port front end (MPFE) for multiple HPS masters. The SDRAM adapter is responsible for connecting the HPS to the SDRAM hard memory controller in the FPGA portion of the device. The SDRAM L3 interconnect is part of the system interconnect.

8.1.5.1. Features of the SDRAM L3 Interconnect

The SDRAM L3 interconnect supports the following features:

- Connectivity to the SDRAM hard memory controller supporting:
  - DDR4-SDRAM
  - DDR3-SDRAM
- Integrated SDRAM scheduler, functioning as a multi-port front end (MPFE)
- Configurable SDRAM device data widths
  - 16-bit, with or without 8-bit error-correcting code (ECC)
  - 32-bit, with or without 8-bit ECC
  - 64-bit, with or without 8-bit ECC
- High-performance ports
  - MPU subsystem port
  - Main L3 interconnect port
  - Three FPGA ports
    - Two 32-, 64-, or 128-bit ports
    - One 32- or 64-bit port
  - Firewall and security support port

Note: At system startup, the SDRAM I/O pins can be configured separately from the FPGA fabric, allowing the SoC HPS to boot before any soft logic is configured in the FPGA.

Related Information

Boot Source I/O Pins on page 778
Details of the SoC hardware and software boot flow

8.1.5.2. SDRAM L3 Interconnect Block Diagram and System Integration

The SDRAM L3 interconnect is composed of two main blocks: SDRAM adapter and SDRAM scheduler.

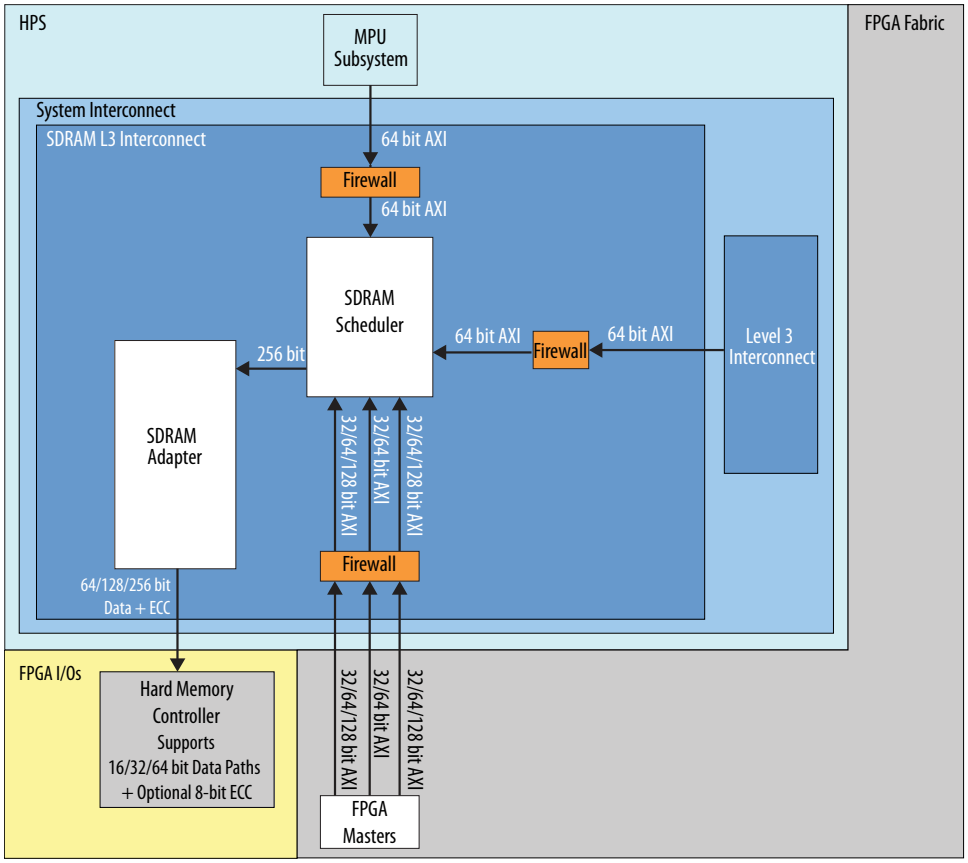
The SDRAM adapter is responsible for bridging the hard memory controller in the FPGA fabric to the SDRAM scheduler. The adapter is also responsible for ECC generation and checking.

The ECC register interface provides control to perform memory and ECC logic diagnostics.

The SDRAM scheduler is an MPFE, responsible for arbitrating collisions and optimizing performance in traffic to the SDRAM controller in the FPGA portion of the device.

Three Arm Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) ports are exposed to the FPGA fabric, allowing soft logic masters to access the SDRAM controller through the same scheduler unit as the MPU subsystem and other masters within the HPS. The MPU has access to the SDRAM adapter's control interface to the hard memory controller.

Figure 34. SDRAM L3 Interconnect Block Diagram



The hard memory controller in the FPGA portion of the device has a dedicated connection to the SDRAM L3 interconnect. This connection allows the hard memory controller to become operational before the rest of the FPGA has been configured.

8.1.5.3. About the SDRAM Scheduler

The SDRAM scheduler functions as a multi-port front end (MPFE), scheduling transactions from multiple masters to the SDRAM.

The SDRAM scheduler supports the following masters:

- The MPU
- The L3 system interconnect
- The FPGA-to-SDRAM bridges

The SDRAM scheduler arbitrates among transactions initiated by the masters, and determines the order of operations. The scheduler arbitrates among the masters, ensuring optimal interconnect performance based on configurable quality-of-service settings.

The SDRAM scheduler can be configured through the registers.

Related Information

- Functional Description of the QoS Generators on page 232
  Detailed description of the system interconnect's quality-of-service features
- Arbitration and Quality of Service in the SDRAM Scheduler on page 228

8.1.6. About Arbitration and Quality of Service

When multiple transactions need the same interconnect resource at the same time, arbitration logic resolves the contention. The quality-of-service (QoS) logic gives you control over how the contention is resolved.

Arbitration and QoS logic work together to enable optimal performance in your system. For example, by setting QoS parameters, you can prevent one master from using up the interconnect's bandwidth at the expense of other masters.

The system interconnect supports QoS optimization through programmable QoS generators. The QoS generators are located on interconnect initiators, which correspond to master interfaces. The initiators insert packets into the interconnect, with each packet carrying a transaction between a master and a slave. Each QoS generator creates control signals that prioritize the handling of individual transactions to meet performance requirements.

Arbitration and QoS in the HPS system interconnect are based on the following concepts:

- Priority—Each packet has a priority value. The arbitration logic generally gives resources to packets with higher priorities.
- Urgency—Each master has an urgency value. When it initiates a packet, it assigns a priority equal to its urgency.
- Pressure—Each data path has a pressure value. If the pressure is raised, packets on that path are treated as if their priority was also raised.
- Hurry—Each master has a hurry value. If the hurry is raised, all packets from that master are treated as if their priority was also raised.

Proper QoS settings depend on your performance requirements for each component and peripheral, and for system performance as a whole. Intel recommends that you become familiar with QoS optimization techniques before you try to change the QoS settings in the HPS system interconnect.

Related Information

- Arria 10 HPS Master-to-Slave Connectivity Matrix on page 202
- Functional Description of the QoS Generators on page 232
- Configuring the Quality of Service Logic on page 241

8.1.7. About the Service Network

The service network is physically separate from the NoC datapath.

Through the service network, you can perform these tasks:

- Access internal interconnect registers
- Update master and slave peripheral security features

8.1.8. About the Observation Network

The observation network connects probes to the CoreSight trace funnel through the debug channel. It is physically separate from the NoC datapath.

The observation network connects probes to the observer, which is a port in the CoreSight trace funnel. Through the observation network, you can perform these tasks:

- Enable error logging
- Selectively trace transactions in the system interconnect
- Collect HPS transaction statistics and profiling data

The observation network consists of probes in key locations in the interconnect, plus connections to observers. The observation network works across multiple clock domains, and implements its own clock crossing and pipelining where needed.

The observation network sends probe data to the CoreSight subsystem through the ATB interface. Software can enable probes and retrieve probe data through the interconnect observation registers.

Related Information

Functional Description of the Observation Network on page 236

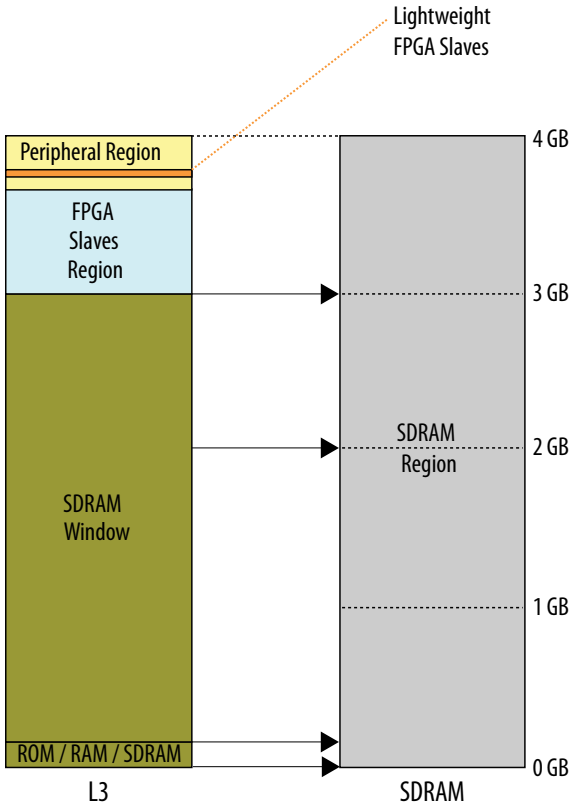
8.2. Functional Description of the System Interconnect

The system interconnect provides access to a 4 GB address space.

Address spaces are divided into one or more nonoverlapping contiguous regions.

Figure 35. HPS Address Space Relationships

Positions of HPS address spaces. For readability, some memory regions are shown out-of-scale to their addresses.



The window regions provide access to other address spaces. The thin black arrows indicate which address space is accessed by a window region (arrows point to accessed address space).

The following table shows the base address and size of each region that is common to the L3 and MPU address spaces.

Table 44. Common Address Space Regions

Region Name	Description	Base Address	Size
FPGASLAVES	FPGA slaves	0xC0000000	960 MB
PERIPH	Peripheral	0xFC000000	64 MB
LWFPGASLAVES (8)	Lightweight FPGA slaves	0xFF200000	2 MB

8.2.1. System Interconnect Address Spaces

The system interconnect supports multiple address spaces.

(8) The LWFPGASLAVES region is part of the "PERIPH" region.

Each address space uses some or all of the 4 GB address range. The address spaces overlap. Depending on the configuration, different address spaces are visible in different regions for each master.

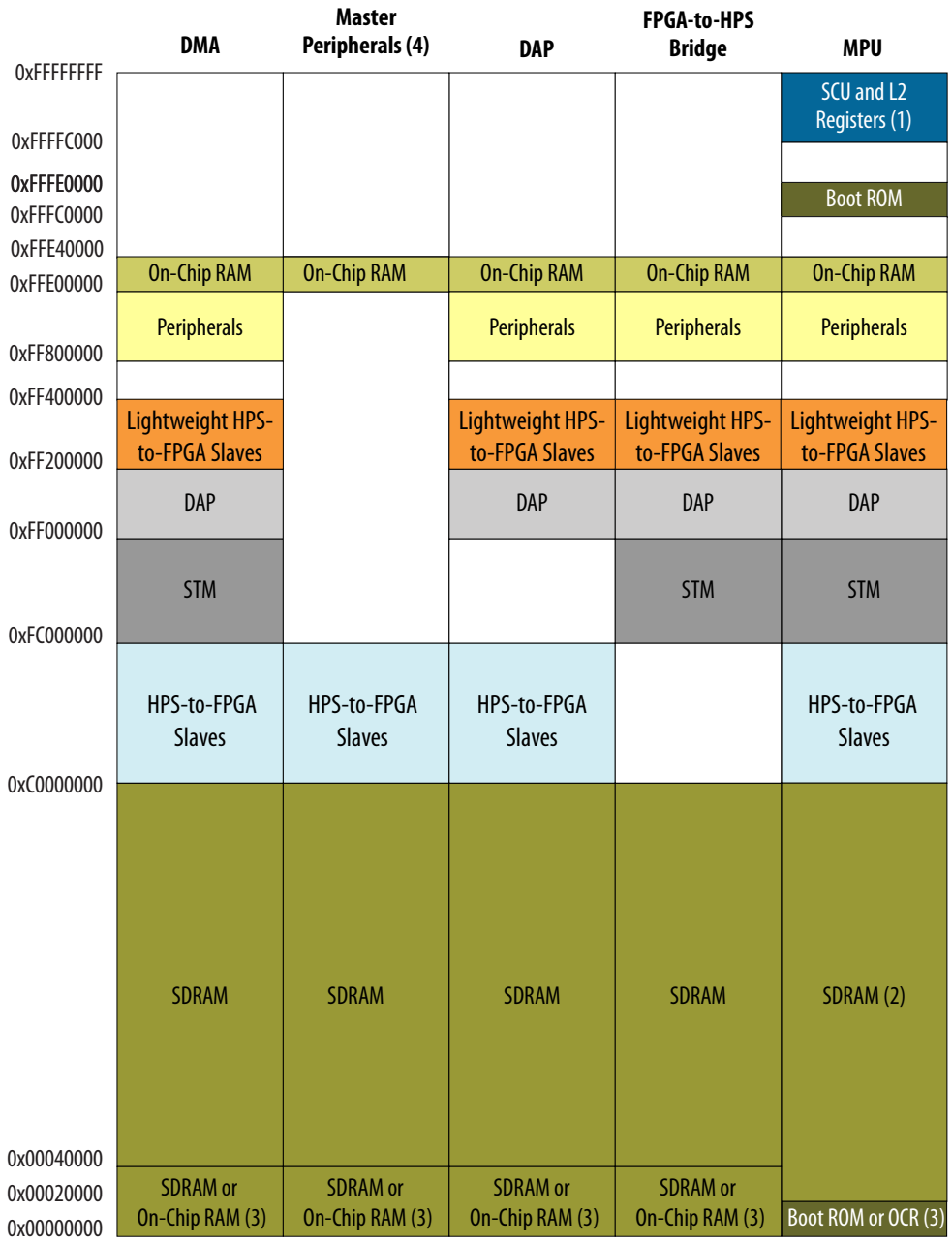
The following address spaces are available:

- The L3 address space
- The MPU address space
- The SDRAM address space

8.2.1.1. Arria 10 HPS Available Address Maps

The following figure shows the default system interconnect address maps for all masters. The figure is not to scale.

Figure 36. Address Maps for System Interconnect Masters



Notes on Address Maps

(1) Transactions on these addresses are directly decoded by the SCU and L2 cache.

(2) The MPU accesses SDRAM through a dedicated port to the SDRAM scheduler. The SDRAM window size also depends on L2 cache filtering.

(3) This region can be configured to access on-chip RAM, by using the `noc_addr_remap_value`, `noc_addr_remap_set`, and `noc_addr_remap_clear` registers, in the system manager.

(4) The following peripherals can master the interconnect:

- Ethernet MACs
- USB-2.0 OTG controllers
- NAND controller
- ETR
- SD/MMC controller

[4]To access registers that are connected to the HPS-to-FPGA AXI master bridge, you need to set the base address of your slave interface starting from 0xC0000000. The HPS-to-FPGA AXI master bridge can be connected to any type of slave interface such as APB and Avalon®.

For the MPU L3 master, either the boot ROM or on-chip RAM can map to address 0x0 and obscure the lowest 128 KB or 256 KB of SDRAM.

At boot time, the MPU does not have access to the SDRAM address space from the top of ROM or on-chip RAM to 0x00100000. This is because the MPU's SDRAM access is controlled by the MPU L2 filter registers, which only have a granularity of 1 MB. After booting completes, the MPU can change address filtering to use the lowest 1 MB of SDRAM.

For non-MPU masters, either the on-chip RAM or the SDRAM maps to address 0x0. When mapped to address 0x0, the on-chip RAM obscures the lowest 256 KB of SDRAM for non-MPU masters.

Related Information

- Memory Map Remap Bits on page 220
  Details of how to map ROM and RAM to 0x00000000
- Cortex-A9 MPU Subsystem Register Implementation on page 0
  Information about HPU register implementation, including the SCU and L2 cache registers
- Address Remapping on page 0
  Information about controlling system interconnect memory maps
- SDRAM Address Space on page 47
  An overview of the SDRAM address space is in the *Introduction to the Hard Processor System* chapter.
- The SDRAM Region on page 0
  More information about L2 cache filtering
- System Manager on page 159
  For more information about setting registers in the system manager, refer to the *System Manager* chapter.

8.2.1.2. Arria 10 HPS L3 Address Space

The Arria 10 HPS L3 address space is 4 GB and applies to all L3 masters except the MPU subsystem.

The system interconnect `noc_addr_remap_value`, `noc_addr_remap_set`, and `noc_addr_remap_clear` registers, in the System Manager, determine if the address space starting at address 0x0 is mapped to the on-chip RAM (256 KB) or the SDRAM. SDRAM is mapped to address 0x0 on reset.

The L3 address space configurations contain the regions shown in the following table:

Table 45. L3 Address Space Regions

Description	Condition	Base Address	End Address	Size
SDRAM window (without on-chip RAM)	`remap0` set, `remap1` clear [9]	0x00000000	0xBFFFFFFF	3 GB
On-chip RAM (low mapping)	`remap0` set, `remap1` set [9]	0x00000000	0x0003FFFF	256 KB
SDRAM window (with on-chip RAM)	`remap0` set, `remap1` set [9]	0x00040000	0xBFFFFFFF	3145471 KB = 3 GB - 256 KB
HPS-to-FPGA	Not visible to FPGA-to-HPS bridge.	0xC0000000	0xFBFFFFFF	960 MB
System trace macrocell	Always visible to DMA and FPGA-to-HPS	0xFC000000	0xFEFFFFFF	48 KB
Debug access port	Not visible to master peripherals. Always visible to other masters.	0xFF000000	0xFF1FFFFF	2 MB
Lightweight HPS-to-FPGA	Not visible to master peripherals. Always visible to other masters.	0xFF200000	0xFF3FFFFF	2 MB
Peripherals	Not visible to master peripherals. Always visible to other masters.	0xFF800000	0xFFFDFFFF	8064 KB
On-chip RAM (high mapping)	Always visible	0xFFE00000	0xFFFE3FFF	256 KB

The boot ROM and internal MPU registers (SCU and L2) are not accessible to L3 masters.

Cache coherent memory accesses have the same view of memory as the MPU.

SDRAM Window Region

The SDRAM window region is 3 GB and provides access to the bottom 3 GB of the SDRAM address space.

On-Chip RAM Region, Low Mapping

The system interconnect `noc_addr_remap_value` register determines if the 256 KB starting at address 0x0 is mapped to the on-chip RAM or the SDRAM. The SDRAM is mapped to address 0x0 on reset.

HPS-to-FPGA Slaves Region

The HPS-to-FPGA slaves region provides access to 960 MB of slaves in the FPGA fabric through the HPS-to-FPGA bridge.

Lightweight HPS-to-FPGA Slaves Region

The lightweight HPS-to-FPGA slaves provide access to slaves in the FPGA fabric through the lightweight HPS-to-FPGA bridge.

[9] For details about the `noc_addr_remap_value` register, refer to "Memory Map Remap Bits"

Peripherals Region

The peripherals region includes slaves connected to the L3 interconnect and L4 buses.

On-Chip RAM Region, High Mapping

The on-chip RAM is always mapped (independent of the boot region contents).

Related Information

- Memory Map Remap Bits on page 220
  Details of how to map ROM and RAM to 0x00000000
- Arria 10 HPS Available Address Maps on page 213
  Details of the L3 and MPU address maps
- HPS Address Spaces on page 46
  More information about L3 address space mapping is in the *Introduction to the Hard Processor System* chapter.
- Cortex-A9 Microprocessor Unit Subsystem on page 282
  For general information about the MPU subsystem, refer to the *Cortex-A9 Microprocessor Unit Subsystem* chapter.

8.2.1.3. MPU Address Space

The MPU address space is 4 GB and applies to MPU masters.

Addresses generated by the MPU are decoded in three ways:

- By default, MPU accesses to locations between 0x100000 (1 MB) to 0xC0000000 (3 GB) are made to the SDRAM controller.
- Addresses in the SCU and L2 register region (0xFFFFC000 to 0xFFFFFFFF) are the SCU and L2 bus.
- Accesses to all other locations are made to the L3 interconnect.

The MPU L2 cache controller contains a master connected to the main L3 interconnect and a master connected to the SDRAM L3 interconnect.

The system interconnect `noc_addr_remap_value`, `noc_addr_remap_set`, and `noc_addr_remap_clear` registers, in the System Manager, determine if the address space starting at address 0x0 is mapped to the on-chip RAM (256 KB) or the ROM (128 KB). The ROM is mapped to address 0x0 on reset.

The MPU address space contains the following regions:

Table 46. MPU Default Address Space Regions

Description	Condition	Base Address	End Address	Size
Boot ROM	`remap0` clear, `remap1` clear (10)	0x00000000	0x0001FFFF	128 KB
SDRAM window	Always visible	0x00100000	0xBFFFFFFF	3071 MB (3 GB – 1 MB)
HPS-to-FPGA	Always visible	0xC0000000	0xFBFFFFFF	960 MB

continued...

(10) For details about the `noc_addr_remap_value` register, refer to "Memory Map Remap Bits".

Description	Condition	Base Address	End Address	Size
				(1 GB – 64 MB)
System trace macrocell	Always visible	0xFC000000	0xFFEFFFFF	48 KB
Debug access port	Always visible	0xFF000000	0xFF1FFFFF	2 MB
Lightweight HPS-to-FPGA	Always visible	0xFF200000	0xFF3FFFFF	2 MB
Peripherals	Always visible	0xFF800000	0xFFDFFFFF	6 MB
On-chip RAM	Always visible	0xFFE00000	0xFFFE3FFF	256 KB
Boot ROM	Always visible	0xFFFC0000	0xFFFDFFFF	128 KB
SCU and L2 registers	Always visible	0xFFFFC000	0xFFFFFFFF	16 KB

Boot Region

The boot region is 1 MB, based at address 0x0. The boot region is visible to the MPU only when the L2 address filter start register is set to 0x100000. The L3 `noc_addr_remap_value` control register determines if the boot region is mapped to the on-chip RAM or the boot ROM.

The boot region is mapped to the boot ROM on reset. Only the lowest 128 KB of the boot region are legal addresses because the boot ROM is only 128 KB.

When the L2 address filter start register is set to 0, SDRAM obscures access to the boot region. This technique can be used to gain access to the lowest SDRAM addresses after booting completes.

SDRAM Window Region

The SDRAM window region provides access to a large, configurable portion of the 4 GB SDRAM address space. The address filtering start and end registers in the L2 cache controller define the SDRAM window boundaries.

The boundaries are megabyte-aligned. Addresses within the boundaries route to the SDRAM master, while addresses outside the boundaries route to the system interconnect master.

Addresses in the SDRAM window match addresses in the SDRAM address space. Thus, the lowest 1 MB of the SDRAM is not visible to the MPU unless the L2 address filter start register is set to 0.

HPS-to-FPGA Slaves Region

The HPS-to-FPGA slaves region provides access to slaves in the FPGA fabric through the HPS-to-FPGA bridge. Software can move the top of the SDRAM window by writing to the L2 address filter end register. If higher addresses are made available in the SDRAM window, part of the FPGA slaves region might be inaccessible to the MPU.

Lightweight HPS-to-FPGA Slaves Region

The lightweight FPGA slaves provide access to slaves in the FPGA fabric through the lightweight HPS-to-FPGA bridge.

Peripherals Region

The peripherals region is near the top of the address space. The peripherals region includes slaves connected to the L3 interconnect and L4 buses.

On-Chip RAM Region

The on-chip RAM is always mapped near the top of the address space, independent of the boot region contents.

Boot ROM Region

The boot ROM is always mapped near the top of the address space, independent of the boot region contents.

SCU and L2 Registers Region

The SCU and L2 registers region provides access to internally-decoded MPU registers (SCU and L2).

8.2.1.4. SDRAM Address Space

The SDRAM address space is up to 4 GB. The entire address space can be accessed through the FPGA-to-SDRAM interface from the FPGA fabric. The total amount of SDRAM addressable from the other address spaces varies.

There are cacheable and non-cacheable views into the SDRAM space. When a master of the system interconnect performs a cacheable access to the SDRAM, the transaction is performed through the ACP port of the MPU subsystem. When a master of the system interconnect performs a non-cacheable access to the SDRAM, the transaction is performed through the 32-bit main L3 interconnect master of the SDRAM L3 interconnect.

8.2.1.5. Address Remapping

The system interconnect supports address remapping through the `noc_addr_remap_value`, `noc_addr_remap_set`, and `noc_addr_remap_clear` registers in the system manager. Remapping allows software to control which memory device (on-chip RAM or boot ROM) is accessible at address 0x0. The remap registers can be modified by the MPU and the FPGA-to-HPS bridge.

The following master interfaces are affected by the remap bits:

- MPU master interface
  - L2 cache master 0 interface
- Non-MPU master interfaces
  - DMA master interface
  - Master peripheral interfaces
  - Debug Access Port (DAP) master interface
  - FPGA-to-HPS bridge master interface

8.2.1.5.1. Memory Map Remap Bits

Table 47. Remap Bit Usage

remap0	remap1	0x00000000 (MPU Master Interface)	0x00000000 (Non-MPU Master Interfaces)	0xFFE00000 (All Maps)	0xFFFC0000 (MPU & DAP)
0	0	Boot ROM	SDRAM	On-Chip RAM	Boot ROM
0	1	Invalid setting			
1	0	SDRAM	SDRAM	On-Chip RAM	Boot ROM
1	1	On-Chip RAM	On-Chip RAM	On-Chip RAM	Boot ROM

L2 filter registers in the MPU subsystem, not the interconnect, allow the SDRAM to be remapped to address `0x0` for the MPU.

8.2.2. Secure Transaction Protection

The system interconnect provides two levels of secure transaction protection:

- Security firewalls—Enforce secure read and write transactions.
- Privilege filter—Leverages the firewall mechanism and provides additional security by filtering the privilege level of L4 slave transactions. The privilege filter applies to writes only.

All slaves on the SoC are placed behind a security firewall. A subset of slaves are also placed behind a privilege filter. Transactions to these slaves must pass both a security firewall and the privilege filter to reach the slave.

Related Information

Functional Description of the Firewalls on page 225

8.2.3. System Interconnect Master Properties

The system interconnect connects to slave interfaces through the main L3 interconnect and SDRAM L3 interconnect.

Table 48. System Interconnect Master Interfaces

Master	Interface Width	Clock	Security	SCR[11] Access	Privilege	Issuance (Read/ Write/ Total)	Type
MPU Subsystem L2 cache M0/1	64	mpu_l2ram_clk	Per Transaction	Yes	Transaction based	7/12/23	AXI
FPGA-to-HPS Bridge [12]	32/64/128	fpga2hps_clk	Per Transaction	Yes	Transaction based	8/8/8	AXI

continued...

(11) Security control register (SCR)

(12) Ensure that Avalon-MM burst transactions into the HPS do not cross the 4 KB address boundary restriction specified by the AXI protocol.

Master	Interface Width	Clock	Security	SCR[11] Access	Privilege	Issuance (Read/ Write/ Total)	Type
FPGA-to-SDRAM Ports [12]	32/64/128	f2h_sdram_clk[2:0]	Per Transaction	No	Transaction based	8/8/8	AXI
DMA	64	l4_main_clk	Per Transaction	No	User mode	8/8/8	AXI
EMAC 0/1/2	32	l4_mp_clk	Secure/Non-Secure	No	User mode	16/16/32	AXI
USB OTG 0/1	32	l4_mp_clk	Nonsecure	No	User mode	2/2/4	AHB
NAND	32	l4_mp_clk	Nonsecure	No	User mode	1/1/2	AXI
SD/MMC	32	l4_mp_clk	Nonsecure	No	User mode	2/2/4	AHB
ETR	32	cs_at_clk	Per Transaction	No	Transaction based	32/1/32	AXI
AHB-AP	32	l4_mp_clk	Per Transaction	Yes	Transaction based	1/1/1	AHB

Related Information

SoC Security on page 168
For more information about master/slave security, refer to the *SoC Security* chapter.

8.2.3.1. Master Caching and Buffering Overrides

Some of the peripheral masters connected to the system interconnect do not have the ability to drive the caching and buffering signals of their interfaces. The system manager provides registers so that you can enable cacheable and bufferable transactions for these masters. The system manager drives the caching and buffering signals of the following masters:

Master Peripheral	System Manager Registers
EMAC0, EMAC1, and EMAC2	`emac0`, `emac1`, and `emac2`
USB OTG 0 and USB OTG 1	`usb0_l3master` and `usb1_l3master`
NAND flash	`nand_l3master`
SD/MMC	`sdmmc_l3master`

At reset time, the system manager drives the cache and buffering signals for these masters low. In other words, the masters listed do not support cacheable or bufferable accesses until you enable them after reset. There is no synchronization between the system manager and the system interconnect, so avoid changing these settings when any of the masters are active.

Related Information

System Manager on page 159
For more information about enabling or disabling this feature, refer to the *System Manager* chapter.

(11) Security control register (SCR)

8.2.4. System Interconnect Slave Properties

The system interconnect connects to various slave interfaces through the main L3 interconnect, the SDRAM L3 interconnect, and the L4 peripheral buses. After reset, all slave interfaces are set to the secure state.

Table 49. System Interconnect Slave Interfaces

Slave	Interface Width	Clock	Acceptance (Read/ Write/Total) (13)	Security	Privilege	Interface Type
SP Timer 0/1/2/3	32	l4_sp_clk	1/1/1	Boot Secure (14)	User Mode	APB(15)
I2C 0/1/2/3/4	32	l4_sp_clk	1/1/1	Boot Secure	User Mode	APB (15)
UART 0/1	32	l4_sp_clk	1/1/1	Boot Secure	User Mode	APB (15)
GPIO 0/1/2	32	l4_sp_clk	1/1/1	Boot Secure	User Mode	APB(15)
SD/MMC CSR	32	l4_mp_clk	1/1/1	Boot Secure	User Mode	APB(15)
EMAC 0/1/2	32	l4_mp_clk	1/1/1	Boot Secure	User Mode	APB(15)
OSC Timer 0/1/2/3	32	l4_sys_free_clk	1/1/1	Secure (16)	Privileged	OCP
Watchdog 0/1/2/3	32	l4_sys_free_clk	1/1/1	Secure	Privileged	APB(15)
Clock Manager	32	l4_sys_free_clk	1/1/1	Secure	Privileged	OCP
Reset Manager	32	l4_sys_free_clk	1/1/1	Secure	Privileged	OCP
System Manager	32	l4_sys_free_clk	1/1/1	Secure	Privileged	OCP
FPGA Manager Data	32	l4_sys_free_clk	1/1/1	Secure	Privileged	OCP
FPGA Manager CSR	32	l4_sys_free_clk	1/1/1	Secure	Privileged	OCP
DAP	32	l4_sys_free_clk	1/1/1	Secure	Privileged	APB(15)
DMA Secure CSR	32	l4_main_clk	1/1/1	Boot Secure	User Mode	APB(15)
DMA Non-Secure CSR	32	l4_main_clk	1/1/1	Boot Secure	User Mode	APB(15)
SPI Slave 0/1	32	l4_main_clk	1/1/1	Boot Secure	User Mode	APB(15)
SPI Master 0/1	32	l4_main_clk	1/1/1	Boot Secure	User Mode	APB(15)
USB OTG CSR 0/1	32	l4_mp_clk	1/1/1	Boot Secure	User Mode	AHB
NAND CSR	32	l4_mp_clk	1/1/1	Boot Secure	User Mode	AHB
NAND Command and Data	32	l4_mp_clk	1/1/1	Boot Secure	User Mode	AHB
SD/MMC ECC	32	l4_mp_clk	1/1/2	Secure	Privileged	OCP

continued...

(13) Acceptance is the maximum number of transactions accepted.

(14) "Boot Secure" means the slave is in the secure state after reset.

(15) The Advanced Peripheral Bus (APB) slave does not support byte enables. All writes must be 32 bits wide.

(16) "Secure" means that the slave is permanently in the secure state.

Slave	Interface Width	Clock	Acceptance (Read/ Write/Total) (13)	Security	Privilege	Interface Type
On Chip RAM ECC	32	l4_mp_clk	1/1/2	Secure	Privileged	OCP
DMA ECC	32	l4_mp_clk	1/1/2	Secure	Privileged	OCP
NAND ECC	32	l4_mp_clk	1/1/2	Secure	Privileged	OCP
USB OTG ECC	32	l4_mp_clk	1/1/2	Secure	Privileged	OCP
EMACS ECC	32	l4_mp_clk	1/1/2	Secure	Privileged	OCP
ACP	64	mpu_l2ram_clk	13/5/18	Secure	User Mode	AXI
APB-DP	32	cs_pl4	1/1/1	Secure	Privileged	APB(15)
DDR	256	f2h_sdram_clk[2:0]	16/16/16	Secure/Non-Secure	User Mode	Avalon
Lightweight HPS-to-FPGA Bridge	32	lwh2fpga_clk	8/8/8	Boot Secure	User Mode	AXI
HPS-to-FPGA Bridge	128/64/32	hps2fpga_clk	8/8/8	Boot Secure	User Mode	AXI
On-Chip RAM	64	l3_main_free_clk	2/2/2	Secure/Non-Secure Per 4KB region	User Mode	AXI
STM	32	cs_at_clk	1/2/2	Secure/Non-Secure	User Mode	AXI

Note: APB-DP has no direct connection to the system interconnect.

8.2.5. System Interconnect Clocks

The system interconnect clocks are driven by the clock manager. The system interconnect's clocks are part of the NoC clock group, which is hardware-sequenced.

Related Information

Hardware Sequenced Clock Groups on page 61
For more information about the NoC clock group

8.2.5.1. List of Main L3 Interconnect Clocks

Table 50. Main L3 Interconnect Clocks

Clock Name	Description	Synchronous to l3_main_free_clk
`l3_main_free_clk`	Clocks the main L3 interconnect	—
`l4_sys_free_clk`	Clocks the following interconnect components: • The L4 system bus • The interface to the DAP	Y
`l4_main_clk`	Clocks fast L4 peripherals on the L4 main bus	Y

continued...

(13) Acceptance is the maximum number of transactions accepted.

Clock Name	Description	Synchronous to l3_main_free_clk
	Refer to "System Interconnect Master Properties" and "System Interconnect Slave Properties" for detailed peripheral-to-clock mappings.	
l4_mp_clk	Clocks the following interconnect components: • Mid-speed L4 peripherals on the L4 MP bus • The L4 AHB bus • The L4 ECC bus Refer to "System Interconnect Master Properties" and "System Interconnect Slave Properties" for detailed peripheral-to-clock mappings.	Y
l4_sp_clk	Clocks slow L4 peripherals on the L4 SP bus. Refer to "System Interconnect Master Properties" and "System Interconnect Slave Properties" for detailed peripheral-to-clock mappings.	Y
cs_at_clk	CoreSight trace clock. Clocks the CoreSight Embedded Trace Router (ETR) master interface.	Y
mpu_l2ram_clk	Clocks the MPU subsystem master interfaces.	N
fpga2hps_clk	Clocks the FPGA-to-HPS bridge.	N
hps2fpga_clk	Clocks the HPS-to-FPGA bridge.	N
lwh2fpga_clk	Clocks the lightweight HPS-to-FPGA bridge.	N

Related Information

- System Interconnect Block Diagram on page 201
- System Interconnect Master Properties on page 220
  Detailed peripheral-to-clock mappings
- System Interconnect Slave Properties on page 222
  Detailed peripheral-to-clock mappings

8.2.5.2. List of SDRAM L3 Interconnect Clocks

Table 51. SDRAM L3 Interconnect Clocks

Clock Name	Description	Synchronous to l3_main_free_clk
hmc_free_clk	Clock from the hard memory controller. Clocks the SDRAM L3 interconnect. The hmc_free_clk frequency is ½ the interface clock frequency.	N
f2s_sdram_clk [2:0]	Clocks the FPGA-to-SDRAM interfaces.	N

8.2.6. System Interconnect Resets

The system interconnect has one reset signal, l3_rst_n. The reset manager drives this signal to the system interconnect on a cold or warm reset.

Related Information

Reset Manager on page 134

8.2.7. Functional Description of the Rate Adapters

Rate adapters are used at points in the system interconnect where there are bandwidth discontinuities, to ensure efficient use of interconnect data pathways. They are placed where a low-bandwidth source feeds a high-bandwidth destination. For this reason, they are sometimes positioned at the response side (where the Master is faster) and sometimes at the request side (where the Slave is faster).

The following table shows the rate adapters.

Table 52. Rate Adapter Modules

Rate Adapter Module Name	Data Path
noc_mpu_m0_MPU_M1toDDRResp_main_RateAdapter	Data packets from the SDRAM L3 interconnect in response to the MPU
noc_mpu_m0_MPU_M0_rate_adResp_main_RateAdapter	Data packets from the L3 interconnect in response to the MPU
noc_mpu_m0_L4_MP_rate_ad_main_RateAdapter	Data packets carrying requests from L4 master peripherals to the L3 interconnect
noc_mpu_m0_fpga2soc_rate_ad_main_RateAdapter	Data packets carrying requests from the FPGA-to-HPS bridge master to the L3 interconnect
noc_mpu_m0_L3Tosoc2fpgaResp_main_RateAdapter	Data packets from the HPS-to-FPGA and lightweight HPS-to-FPGA bridges in response to the L3 interconnect
noc_mpu_m0_acp_rate_ad_main_RateAdapter	Data packets carrying requests from the L3 interconnect to the ACP

Related Information

Configuring the Rate Adapters on page 238

8.2.8. Functional Description of the Firewalls

8.2.8.1. Security

8.2.8.1.1. Slave Security

The system interconnect enforces security through the slave settings. The slave settings are controlled by the NoC Security Control Register (SCR) in the service network. Each L3 and L4 slave has its own security check and programmable security settings. After reset, every slave of the system interconnect is set to a secure state (referred to as boot secure). Only secure masters are allowed to access secure slaves.

The NoC implements five firewalls to check the security state of each slave, as listed in the following table. At reset time, all firewalls default to the secure state.

Table 53. NoC Firewalls

Name	Function
On-Chip RAM Firewall	Filter access to on-chip RAM
Peripherals Firewall	Filter access to slave peripherals (SPs) in the following buses:

continued...

Name	Function
	• L4 main bus • L4 master peripherals bus • L4 AHB bus • L4 slave peripherals bus
System Firewall	Filter access to system peripherals in the following components: • L4 system bus • L4 ECC bus • DAP
HPS-to-FPGA Firewall	Filter access to FPGA through the following bridges: • HPS-to-FPGA bridge • Lightweight HPS-to-FPGA bridge
DDR and DDR L3 Firewalls	Filter access to DDR SDRAM

At reset, the privilege filters are configured to allow certain L4 slaves to receive only secure transactions. Software must either configure bridges secure at startup, or reconfigure the privilege filters to accept nonsecure transactions. You can reconfigure the privilege filters through the `l4_priv` register in the `noc_l4_priv_l4_priv_filter` module.

To change the security state, you must perform a secure write to the appropriate SCR register of a secure slave. A nonsecure access to the SCR register of a secure slave triggers a response with random data.

Note: Future devices might not support the return of random data and might only support an error response for blocked firewall transactions. For designs that may be ported to future devices, Intel recommends you to set the `error_response` bit in the `global` register of the `noc_fw_ddr_l3_ddr_scr` module.

Related Information

l4_priv on page 0
Privilege bits, used to enable or disable nonsecure access to individual L4 slaves

8.2.8.1.2. Master Security

Masters of the system interconnect are either secure, nonsecure, or the security is set on a per transaction basis. L2 cache masters 0 and 1, the FPGA-to-HPS bridge, DMA, the Ethernet MACs, and the DAP perform secure and nonsecure accesses on a per-transaction basis. All other system interconnect masters perform nonsecure accesses.

Accesses to secure slaves by unsecure masters result in a response with random data.

Note: Future devices might not support the return of random data and might only support an error response for blocked firewall transactions. For designs that may be ported to future devices, Intel recommends you to set the `error_response` bit in the `global` register of the `noc_fw_ddr_l3_ddr_scr` module.

Related Information

System Interconnect Master Properties on page 220

8.2.9. Functional Description of the SDRAM L3 Interconnect

The SDRAM L3 interconnect consists of two main blocks, serving the following two main functions:

- The SDRAM scheduler provides multi-port scheduling between the SDRAM L3 interconnect masters and the hard memory controller in the FPGA portion of the SoC. The SDRAM L3 interconnect is mastered by the MPU, the main L3 interconnect, and FPGA-to-SDRAM ports.
- The SDRAM adapter provides connectivity between the SDRAM L3 interconnect masters and the hard memory controller.

The SDRAM L3 interconnect also includes firewalls that can protect regions of SDRAM from unauthorized access.

The hard memory controller is physically located in the FPGA portion of the device, and therefore it is in a separate power domain from the HPS. The HPS cannot use the SDRAM L3 interconnect until the FPGA portion is powered up.

8.2.9.1. Functional Description of the SDRAM Scheduler

The SDRAM scheduler functions as a multi-port front end (MPFE), scheduling transactions from multiple masters to the SDRAM.

The SDRAM scheduler manages transactions to the memory access regions in the SDRAM. These memory regions are defined by the SDRAM L3 firewalls. The second-stage bootloader is expected to program the scheduler with the correct timings to implement optimal access patterns to the hard memory controller.

The SDRAM scheduler has the following features:

- Five input connections
  - One 64-bit connection from the MPU
  - One 64-bit connection from the main L3 interconnect
  - Two 128/64/32-bit connections from the FPGA
  - One 64/32-bit connection from the FPGA
- Single 256-bit connection to the SDRAM L3 adapter
  - Capable of issuing transactions at the memory device line rate
  - Traffic is comprised of aggregate inputs

Related Information

SDRAM L3 Firewalls on page 230

8.2.9.1.1. Monitors for Mutual Exclusion

The SDRAM scheduler implements support for mutually-exclusive (mutex) accesses on all ports to the SDRAM L3 interconnect.

The process for a mutually-exclusive access is as follows:

1. A master attempts to lock a memory location by performing an exclusive read from that address.
2. The master attempts to complete the exclusive operation by performing an exclusive write to the same address location.
3. The exclusive write access is signaled as:
   - Failed if another master has written to that location between the read and write accesses. In this case the address location is not updated.
   - Successful otherwise.

To support mutually-exclusive accesses from the MPU, the memory must be configured in the MMU page tables as normal memory, shareable, or non-cacheable.

Related Information

- On-Chip Memory on page 364
- Embedded Peripheral IP User Guide
  Contains detailed information about the mutex core

Exclusive Access Support

To ensure mutually exclusive access to shared data, use the exclusive access support built into the SDRAM scheduler. The AXI buses that interface to the scheduler provide `ARLOCK[0]` and `AWLOCK[0]`, signals which are used by the scheduler to arbitrate for exclusive access to a memory location. The SDRAM scheduler contains six monitors. Each monitor can be used by any of the following exclusive-capable masters:

- CPU 0
- CPU 1
- FPGA-to-HPS bridge
- FPGA-to-SDRAM0 port
- FPGA-to-SDRAM1 port
- FPGA-to-SDRAM2 port

Each master can lock only one memory location at a time.

8.2.9.1.2. Arbitration and Quality of Service in the SDRAM Scheduler

The SDRAM scheduler controls arbitration priorities internally for optimal end-to-end quality of service. You can apply QoS settings to each port of the SDRAM scheduler to control the bandwidth available to each master.

Each master on the SDRAM scheduler has software-programmable QoS signals. These signals are propagated to the scheduler and used as arbitration criteria for access to SDRAM.

For information about programming quality of service for the FPGA-to-SDRAM masters, refer to "Functional Description of the QoS Generators" and "Configuring the Quality of Service Logic".

Related Information

- Functional Description of the QoS Generators on page 232

- Configuring the Quality of Service Logic on page 241

8.2.9.2. Functional Description of the SDRAM Adapter

The SDRAM adapter connects the SDRAM scheduler with the Hard Memory Controller.

The SDRAM adapter provides the following functionality:

- ECC generation, detection, and correction
- Operates at memory half rate
  - Matches interface frequency of the single port memory controller in the FPGA
  - Connectivity to the MPU, main L3 interconnect, and FPGA undergo clock crossing

8.2.9.2.1. ECC

The SDRAM Adapter ECC can detect and correct single-bit errors and detect double-bit errors.

The addresses are merged with data and are used for checking. This configuration detects if correct write data is written to an incorrect location in memory; and if correct data is read from an incorrect location in memory.

Note: Data and address errors are detected independently.

ECC Write Behavior

When data is written to SDRAM, the SDRAM controller generates an ECC based on the write data and the write address.

If the write transaction is a partial write (less than 64 bits wide), the SDRAM adapter implements it as a read-modify-write (RMW) sequence, as follows:

- Reads existing data from the specified address
- Combine the write data with the existing data
- Generates the ECC based on the combined data and the write address
- Writes the combined data back to the write address

ECC Read Behavior

When data is read from SDRAM, the SDRAM controller checks the ECC to determine if the data or address is incorrect. It handles the following cases:

- If the SDRAM controller finds a single-bit data error, it corrects in the data returned to the master. You can also enable the SDRAM adapter to write the corrected data back to memory.
- If the SDRAM controller finds a double-bit data error, the SDRAM L3 interconnect issues a bus error. The SDRAM L3 interconnect can also issue an interrupt, if enabled. Double-bit errors cannot be corrected.
- If the SDRAM controller finds an error in the address, indicating an address bit upset between the adapter and the HMC, the SDRAM L3 interconnect hardware issues a bus error.

8.2.9.2.2. SDRAM Adapter Interrupt Support

The SDRAM adapter supports the following three interrupts:

- The status interrupt occurs when:
  - Calibration is complete.
  - The ECC is unable to schedule an auto-correction write-back to memory. This occurs only when the auto-write-back FIFO is full.
- The ECC read-back interrupt occurs when an ECC single-bit error is detected in the read data. When this happens, the return data is corrected and returned to the NoC.
- The double-bit or fatal error interrupt occurs when any of the following three errors happens:
  - A double-bit error in the read data has been detected, which cannot be corrected.
  - A single-bit error has been detected in the address field. This means that the data that the adapter is returning has no bit errors, but is not the requested data. When this happens, the adapter returns a data error along with the data.
  - Any of the DDR4 devices have triggered their ALERT pins.
    - Address or command parity check has failed
    - Write data CRC check has failed
    - Cannot gracefully recover because SDRAMs are not providing feedback on failure case

8.2.9.2.3. SDRAM Adapter Clocks

All the logic in the SDRAM adapter is synchronous and effectively running off a single clock, `hmc_free_clk`, which is provided by the hard memory controller.

8.2.9.3. SDRAM L3 Firewalls

All data that is routed to the SDRAM scheduler must pass through the firewalls.

The SDRAM L3 firewalls define memory access regions in the SDRAM. Each SDRAM L3 interconnect master has its own memory access regions, independent of the other masters. The firewalls define whether each memory access region is protected or unprotected relative to its master. The number of available memory access regions for each master is shown in the following table.

Table 54. Memory Access Regions for SDRAM Masters

SDRAM L3 Interconnect Master	Number of Memory Access Regions
MPU	4
Main L3 interconnect (including the FPGA-to-HPS bridge)	8
FPGA-to-SDRAM port 0	4
FPGA-to-SDRAM port 1	4
FPGA-to-SDRAM port 2	4

The SDRAM L3 interconnect regulates access to the hard memory controller with the firewalls, which support secure regions in the SDRAM address space. Accesses to the SDRAM pass through the firewalls and then through the scheduler.

Related Information

SoC Security on page 168

8.2.9.4. SDRAM L3 Interconnect Resets

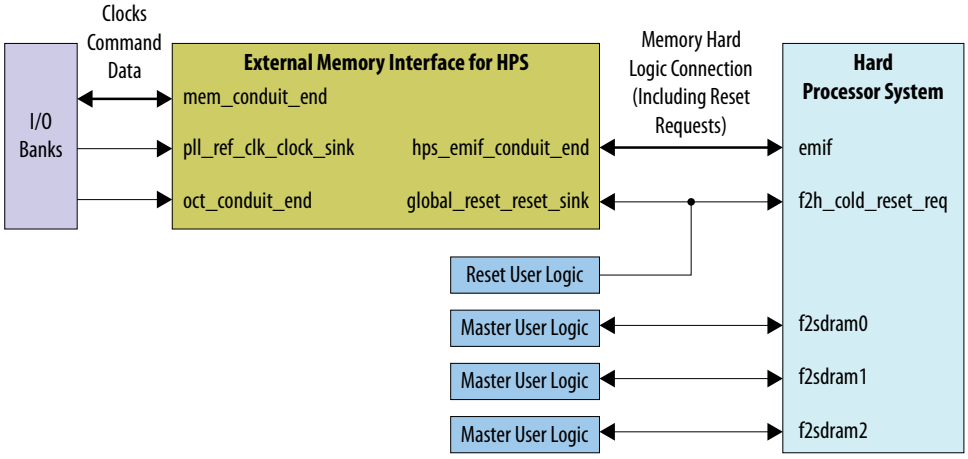
The reset signal `l3_rst_n` resets the system interconnect and the SDRAM L3 interconnect, but not the hard memory controller.

The reset signal in the hard memory controller is automatically connected to the SDRAM L3 interconnect when you instantiate the HPS component. For information about resetting the hard memory controller, refer to the *External Memory Interfaces in Arria 10 Devices* chapter of the *Arria 10 Core Fabric and General Purpose I/O Handbook*.

Soft logic in the FPGA must support the `global_reset_n` signal correctly. Refer to the *Instantiating the HPS Component* chapter for information about `global_reset_n`.

To optionally preserve the contents of the SDRAM on reset, refer to "Reset Handshaking" in the *Reset Manager* chapter.

Figure 37. Recommended SDRAM Reset Connections



Note: It is important to connect the reset user logic directly to both the HPS and the hard memory controller. If the hard memory controller is reset while the HPS is still running, the HPS is unable to access any external SDRAM memory.

Related Information

- System Interconnect Resets on page 224
- Instantiating the HPS Component on page 716
  Details about hard memory controller reset connections and correct support for global_reset_n
- Reset Handshaking on page 152

- Arria 10 Core Fabric and General Purpose I/O Handbook
  Refer to the *External Memory Interfaces in Arria 10 Devices* chapter for information about resetting the hard memory controller.

8.2.10. Functional Description of the Arbitration Logic

The interconnect arbitration logic handles situations in which multiple simultaneous packets need the same interconnect resource.

The system interconnect contains arbitration nodes at each point where multiple packets might demand the same resource. Each arriving packet has a priority. When multiple packets of different priorities arrive simultaneously at a node, the arbitration logic grants the resource to the highest-priority packet.

If there are simultaneous packets with the same priority, the system interconnect uses a simple round-robin (least recently used) algorithm to select the packet.

Each packet's priority is determined by urgency, pressure, and hurry, as described in "QoS Mechanisms".

Related Information

QoS Mechanisms on page 233

8.2.11. Functional Description of the QoS Generators

Quality of service is managed by information generated by a QoS generator at each initiator interface (master). The QoS generators work with the arbitration logic so you can tune the interconnect's performance to your system needs.

At the entry points to the system interconnect, the QoS generators create the following information to control how the arbitration logic allocates interconnect bandwidth among all transactions:

- Urgency—Each master has an urgency value. When it initiates a packet, it assigns a priority equal to its urgency.
- Pressure—Each data path has a pressure value. If the pressure is raised, packets on that path are treated as if their priority was also raised.
- Hurry—Each master has a hurry value. If the hurry is raised, all packets from that master are treated as if their priority was also raised.

The QoS information is used at each interconnect arbitration node to assign a packet priority. When there is contention for a resource, the arbitration logic generally gives the resource to the packets with higher priority.

For detailed descriptions of urgency, pressure, and hurry, refer to "QoS Mechanisms".

The values of urgency, pressure, and hurry are determined by each QoS generator's operating mode. QoS generators support the following modes:

- Limiter
- Regulator
- Fixed

For detailed descriptions of the QoS generator modes, refer to "QoS Generator Modes".

Related Information

- About Arbitration and Quality of Service on page 210
- Arbitration and Quality of Service in the SDRAM Scheduler on page 228
- QoS Generator Modes on page 235
- QoS Mechanisms on page 233

8.2.11.1. QoS Generators in the Interconnect Architecture

The principal masters into the system interconnect are equipped with QoS generators.

The following table shows the interconnect initiators (masters) that support QoS generation.

Table 55. QoS Generators on Interconnect Masters

Master Name(s)	Default QoS Generator Mode	Clock
MPU Subsystem L2 cache M0/1	Limiter	mpu_l2ram_clk
FPGA-to-HPS Bridge	Limiter	fpga2hps_clk
FPGA-to-SDRAM Ports	Regulator	f2h_sdram_clk
DMA	Limiter	l4_main_clk
EMAC 0/1/2	Regulator	l4_mp_clk
USB OTG 0/1	Regulator	l4_mp_clk
NAND	Regulator	l4_mp_clk
SD/MMC	Regulator	l4_mp_clk

Related Information

QoS Generator Modes on page 235

8.2.11.2. QoS Mechanisms

The QoS generators influence arbitration priorities by manipulating urgency, pressure, and hurry.

When two or more packets compete for an interconnect resource at an arbitration node, the arbitration logic selects the packet with the highest priority. QoS generators can control the behavior of the arbitration logic by modifying the priority of packets through urgency, pressure, and hurry.

The QoS controls for each master are separated into read and write QoS controls.

Related Information

Functional Description of the Arbitration Logic on page 232

8.2.11.2.1. Packet Urgency

An urgency value is attached to each packet as long as it is moving through the interconnect. Urgency is the primary QoS mechanism.

A QoS generator assigns an urgency level to each packet when the packet data arrives at an interconnect master port.

8.2.11.2.2. Pressure Propagation

Each active datapath in the interconnect has a pressure level, which is the highest urgency of all packets currently in the datapath. Pressure can change depending on the traffic through the datapath.

The pressure is based on the urgency of packets waiting for that datapath. If a higher-priority packet requests a datapath that is already active, the datapath's urgency increases.

When a packet arrives at an arbitration node, normally its priority is the urgency assigned by its master. However, if the datapath's pressure is greater than the urgency, that pressure tells the node that there is a higher-urgency packet somewhere behind the current one. This prompts the node to expedite the lower-urgency packet to free up the datapath.

For example, consider three active packets in the interconnect. Packets A and B are moving through the same datapath, with A ahead of B. Packet C is on a different datapath.

Packet A's urgency value is 1, and packet B's urgency is 3. Packet A arrives at a particular node simultaneously with Packet C—and Packet C's urgency is 2. Without the pressure mechanism, Packet C would pass through the node first, ahead of Packet B, even though Packet B has the highest urgency of the three packets.

However, because Packet B's urgency is 3, its datapath has a pressure of 3. That pressure value tells the node to accept the packet on that datapath—Packet A—even though it has a lower urgency. After Packet A is out of the way, the node can accept Packet B ahead of Packet C.

8.2.11.2.3. Hurry

Each QoS generator has a hurry level, which can change depending on the bandwidth usage and the QoS generator mode.

Hurry is a mechanism to propagate the master's urgency level to all nodes that receive packets from that master. Hurry allows the master to influence the priority not only of the current transaction, but also of all its transactions that are still active.

When the hurry level is increased at a master, it has the effect of setting the urgency of that master's transactions to at least the same level as the hurry. As a result, the targets respond to the transactions more quickly.

For example, consider a master whose QoS generator is configured in regulator mode. This QoS generator dynamically raises and lowers its urgency value depending on the traffic it experiences. Packets sent from the master use the QoS generator's current urgency value. If the QoS generator raises its urgency value, it uses the Hurry mechanism to increase the pressure value at every datapath where it has a pending transaction.

The QoS generator implements hurry by broadcasting an urgency packet on every datapath where it has pending transactions. The urgency packet contains the current urgency value, which raises pressure on each datapath.

Related Information

Regulator Mode on page 235

8.2.11.3. QoS Generator Modes

Each QoS generator can operate in any of the following modes:

- Limiter—Each initiator has a maximum bandwidth cap to prevent it from sending too many requests. When the cap is reached, the initiator stops accepting new requests.
- Regulator—Each initiator has an ideal minimum bandwidth threshold. When the initiator's bandwidth usage drops below threshold, it raises the urgency for new transactions. Any pending transactions from that initiator are also expedited by the hurry mechanism.
- Fixed—Each initiator assigns static urgency values to packets. The urgency value for read packets can be different from the value for write packets.

QoS generators can also be set to bypass mode. In this case, QoS signals are provided by the component connected to the master interface.

For the default QoS generator mode for each initiator, refer to "QoS Generators in the Interconnect Architecture".

Related Information

- QoS Generators in the Interconnect Architecture on page 233
  Default QoS modes for each initiator
- Hurry on page 234

8.2.11.3.1. Limiter Mode

In limiter mode, transactions from the initiator are limited to a specific bandwidth.

The QoS generator controls the initiator's access to the interconnect, by monitoring the amount of data transmitted and received in a sliding time window.

The purpose of limiter mode is to place a hard bound on the initiator's throughput. This avoids flooding the interconnect with transactions from that initiator. When a programmable bandwidth threshold is exceeded, the initiator is stalled, and stops accepting new requests. As the time window moves forward, the total bandwidth usage in the window falls, and the initiator resumes accepting requests.

In limiter mode, the QoS generator combines read and write requests to determine the bandwidth usage.

Related Information

Programming QoS Limiter Mode on page 241

8.2.11.3.2. Regulator Mode

In regulator mode, the QoS generator maintains bandwidth usage to transaction targets at a programmed level, by adjusting urgency, pressure, and hurry.

Use this mode to guarantee a specific bandwidth is available to the master, while preventing it from claiming more bandwidth than you requested.

When the initiator's bandwidth usage exceeds the specified threshold, the QoS downgrades the urgency, pressure, and hurry levels. When bandwidth usage is too low, the QoS increases urgency, pressure and hurry. Software can program the desired bandwidth threshold, the saturation, and maximum and minimum levels for urgency, pressure, and hurry.

Regulator mode controls bandwidth usage more smoothly than limiter mode. This mode is useful in achieving real-time performance.

Related Information

Programming QoS Regulator Mode on page 242

8.2.11.3.3. Fixed Mode

In fixed mode, the QoS generator assigns a fixed urgency to each packet. Software can configure separate urgencies for read and write transactions.

This mode is useful for coarse-grain control in a system where one master should have the first access to interconnect resources. However, this mode can have significant system level performance degradation due to other masters being starved.

Related Information

Programming QoS Fixed Mode on page 243

8.2.12. Functional Description of the Observation Network

8.2.12.1. Probes

The system interconnect includes the probes shown in the following table.

Table 56. Observation Probes

Probe Name	Probe ID	Probe Type	Probe Target(s)	Packet Tracing and Statistics	Transaction Profiling	Error Logging
MPU	6	Packet	MPU master 0	•		
SOC2FPGA	5	Packet	• HPS-to-FPGA bridge • Lightweight HPS-to-FPGA bridge	•		
DDR	10	Packet	SDRAM adapter	•		
EMAC	4	Packet	EMAC0		•	
EMAC	4	Packet	• EMAC1 • EMAC2	•		
ERROR	n/a	Error	• HPS-to-FPGA bridge • Lightweight HPS-to-FPGA bridge`			•

The probe ID is available on the ATB trace bus.

8.2.12.2. Packet Tracing, Profiling, and Statistics

Packet probes perform packet tracing, profiling and statistics. The following table shows how each packet probe is configured. All statistics counters are 16 bits wide, and all probes can generate level alarms from the statistics counter. All probes use the CoreSight cross-trigger protocol.

Table 57. Packet Probe Characteristics

Probe Name	Purpose	Number of Filters	Statistics Counters Can Count Enabled Bytes	Packet Payloads Included in Trace Data	Number of Statistics Counters
MPU	MPU	1	No	No	4
SOC2FPGA	HPS-to-FPGA bridge	2	No	No	4
EMAC	Ethernet MAC	2	No	No	4
DDR	SDRAM	4	Yes	Yes	2

8.2.12.2.1. Packet Filtering

You can set up filters to control how the observation network handles traced packets.

Filters can perform the following tasks:

- Select which packets the observation network routes to CoreSight
- Trigger a trace alarm when a packet meets specified criteria

8.2.12.2.2. Statistics Collection

To collect packet statistics, you specify packet criteria and set up counters for packets that meet those criteria. You can set up the observation network to trigger an alarm when a counter reaches a specified level.

8.2.12.2.3. Transaction Profiling

A transaction probe is available on Ethernet MAC 0. You can use the transaction probe to measure either the transaction latency or the number of pending packets on EMAC0. Data are collected as a histogram.

The EMAC0 transaction probe is configured as shown in the following table.

Table 58. EMAC0 Transaction Probe Configuration

Number of simultaneously tracked transactions	4
Width of counters	10 bits
Available delay thresholds	64, 128, 256, 512
Available pending transaction count thresholds	2, 4, 8
Number of comparators	3
Number of histogram bins	4

Profiling Transaction Latency

In latency mode (also called delay mode), one of the four delay threshold values can be chosen for each comparator. The threshold values represent the number of clock cycles that a transaction takes from the time the request is issued to the time the response is returned.

Profiling Pending Transactions

In pending transaction mode, one of the three available transaction count threshold values can be chosen for each comparator. The threshold values represent the number of requests pending on EMAC0.

8.2.12.3. Packet Alarms

Packet alarms can be used to trigger software interrupts.

The following types of alarms are available:

- Trace alarms—You can examine trace alarms through the system interconnect registers.
- Statistics alarms

8.2.12.4. Error Logging

The error probe logs errors on initiators, targets, and firewalls.

8.3. Configuring the System Interconnect

Related Information

SoC Security on page 168

8.3.1. Configuring the Rate Adapters

The default settings for the rate adapters are as shown in the following table.

Table 59. Rate Register Default Settings

Rate Register Name	Default
MPU_M1toDDRResp_main_RateAdapter_Rate	0x0
MPU_M0_rate_adResp_main_RateAdapter_Rate	0x0
L4_MP_rate_ad_main_RateAdapter_Rate	0x100
fpga2soc_rate_ad_main_RateAdapter_Rate	0x0
L3Tosoc2fpgaResp_main_RateAdapter_Rate	0x0
acp_rate_ad_main_RateAdapter_Rate	0x0

8.3.2. Configuring the SDRAM Scheduler

8.3.2.1. FPGA Port Configuration

The FPGA has three outputs that pass through the firewall before connecting to the SDRAM scheduler.

You can configure the FPGA-to-SDRAM (F2SDRAM) ports to data widths of 32, 64, or 128 bits:

- F2SDRAM 0 - 32-, 64-, or 128-bit data widths
- F2SDRAM 1 - 32- or 64-bit data widths
- F2SDRAM 2 - 32-, 64-, or 128-bit data widths

There are four port configurations that are a combination of the SDRAM ports 0 - 2 that you are able to select. Once a port configuration is selected, you can choose to disable ports that you do not need.

Note: The total data width of all interfaces is limited to a maximum of 256 bits in the read direction and 256 bits in the write direction.

Port Configuration	F2SDRAM 0	F2SDRAM 1	F2SDRAM 2
1	32-bit	32-bit	32-bit
2	64-bit	64-bit	64-bit
3	128-bit	unused	128-bit
4	128-bit	32-bit	64-bit

8.3.2.2. Memory Timing Configuration

The SDRAM L3 interconnect provides fully-programmable timing parameter support for all JEDEC-specified timing parameters.

The following lists the handoff information used to control the SDRAM scheduler:

- The scheduler is aware of the SDRAM timing so that it can guide traffic into the hard memory controller.
- This information is not used to control the subsystem that sets up memory timings in the hard memory controller hardware.

8.3.2.3. Configuring SDRAM Burst Sizes

You can optimize memory throughput by adjusting the SDRAM burst lengths for read and write transactions. The best burst length values are application-dependent. Intel recommends that you follow guidelines in *Arria 10 SoC Device Design Guidelines* to determine the optimal burst lengths.

Related Information

- Arria 10 SoC Device Design Guidelines
- HPS-to-FPGA Bridges Design Example
  Download **Arria 10 FPGA-to-HPS Bridges design example (.zip file)** for use with the *Arria 10 SoC Device Design Guidelines*.

8.3.3. Configuring the Hard Memory Controller

8.3.3.1. SDRAM Adapter Memory Mapped Registers

The SDRAM adapter memory mapped registers (MMRs) are used for configuring and reading ECC status; and configuring data width adaptation for 16-, 32-, and 64-bit data widths.

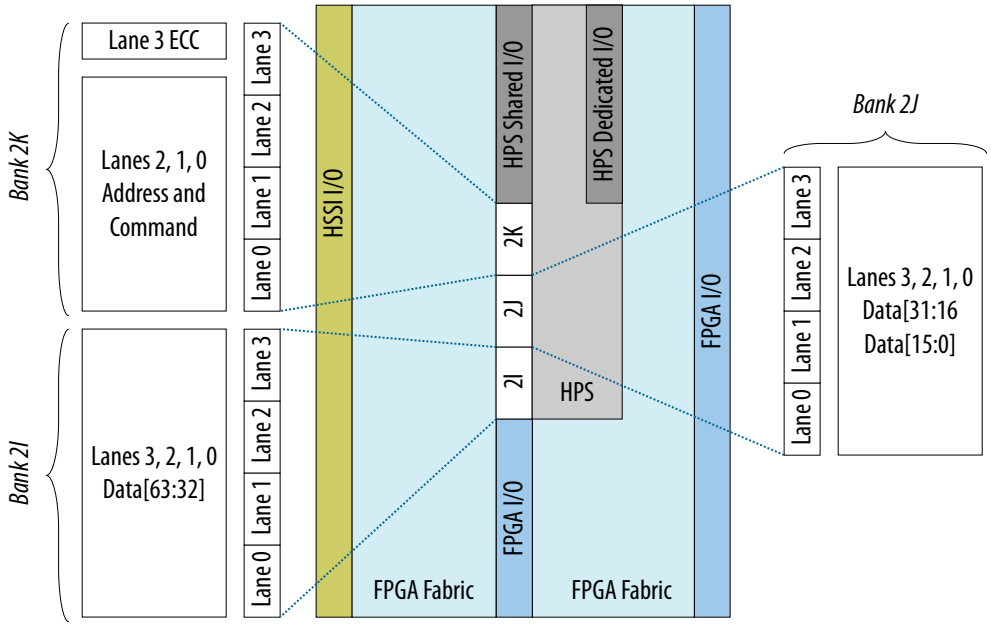
8.3.3.2. Sharing I/Os between the EMIF and the FPGA

Arria 10 SoC devices have three modular I/O banks to connect the HPS to an SDRAM (2K, 2J, and 2I) through a dedicated HPS EMIF.

Each bank has four I/O lanes that correspond to:

- Lane 3: IO[47:36]
- Lane 2: IO[35:24]
- Lane 1: IO[23:12]
- Lane 0: IO[11:0]

Figure 38. HPS Hard Memory Controller



To use SDRAM, you instantiate the HPS EMIF in Platform Designer, by selecting the **Arria 10 External Memory Interfaces for HPS** component. Quartus Prime software assigns the correct banks and lanes for the SDRAM I/O.

I/Os in these three banks can be shared between the HPS EMIF and FPGA. Quartus Prime software enforces the following guidelines on shared I/Os:

- Modular I/O banks 2K, 2J and 2I can be used entirely as FPGA GPIO when there is no HPS EMIF in the system.
- Bank 2K:
  - If there is an HPS EMIF in the system, lane 3 is used for ECC for the SDRAM. Unused pins in lane 3 may be used as FPGA inputs only.
  - Lanes 2, 1, and 0 are used for address and command for the SDRAM. Unused pins in these lanes may be used as FPGA inputs or outputs.
- Bank 2J:
  - If there is an HPS EMIF in the system, bank 2J is used for data bits [31:0].
    - With 16-bit data widths, unused pins in the two lanes of bank 2J used for data can be used as inputs only. The pins in the remaining two lanes can be used as FPGA inputs or outputs.
    - With 32-bit data widths, unused pins bank 2J can be used as FPGA inputs only.
- Bank 2I:
  - If there is an HPS EMIF in the system, bank 2I is used for data bits [63:32]
    - With 64-bit data widths, unused pins in bank 2I can be used as FPGA inputs only.
    - With 32- or 16-bit data widths, bank 2I can be used as FPGA inputs or outputs.

8.3.4. Configuring the Quality of Service Logic

You can programmatically configure the QoS generator for each master through the QoS registers.

Note: Before accessing the registers for any QoS generator, you must ensure that the corresponding peripheral is not in reset. Otherwise, the register access results in a bus error, causing a CPU fault.

8.3.4.1. Programming QoS Limiter Mode

QoS bandwidth limiter mode is mode 1. To put the QoS generator in limiter mode, set the registers as shown in the following table.

Table 60. QoS Generator Register Values for Limiter Mode

Register	Field	Value
I_main_QosGenerator_Mode	MODE	1
I_main_QosGenerator_Priority	P0	Urgency for write transactions

continued...

Register	Field	Value
I_main_QosGenerator_Priority	P1	Urgency for read transactions
I_main_QosGenerator_Bandwidth	BANDWIDTH	Maximum bandwidth, in units of (bytes/cycle)/256
I_main_QosGenerator_Saturation	SATURATION	Measurement window for bandwidth, in units of bytes/16. This register specifies the maximum number of bytes that the initiator can transmit or receive at full speed before the limiter triggers.

Higher priority (urgency) values mean that a packet receives preferential treatment at each arbitration node.

For detailed information about setting bandwidth and saturation, refer to "Bandwidth and Saturation".

When you switch QoS modes, the bandwidth counter is reset.

Related Information

- QoS Generators in the Interconnect Architecture on page 233
  List of clock domains for each initiator
- QoS Generator Registers on page 244
  List of QoS registers
- Bandwidth and Saturation on page 243

8.3.4.2. Programming QoS Regulator Mode

QoS bandwidth regulator mode is mode 3. To put the QoS generator in regulator mode, set the registers as shown in the following table.

Table 61. QoS Generator Register Values for Regulator Mode

Register	Field	Value
I_main_QosGenerator_Mode	MODE	3
I_main_QosGenerator_Priority	P0	Packet urgency when the actual throughput exceeds the threshold set in I_main_QosGenerator_Bandwidth.
I_main_QosGenerator_Priority	P1	Packet urgency when the actual throughput is less than the threshold set in I_main_QosGenerator_Bandwidth. P0 must be less than or equal to P1.
I_main_QosGenerator_Bandwidth	BANDWIDTH	Desired throughput, in units of (bytes/cycle)/256
I_main_QosGenerator_Saturation	SATURATION	Measurement window for bandwidth, in units of bytes/16.

Higher priority (urgency) values mean that a packet receives preferential treatment at each arbitration node.

For detailed information about setting bandwidth and saturation, refer to "Bandwidth and Saturation".

When you switch QoS modes, the bandwidth counter is reset.

Related Information

- QoS Generators in the Interconnect Architecture on page 233
  List of clock domains for each initiator
- QoS Generator Registers on page 244
  List of QoS registers
- Bandwidth and Saturation on page 243

8.3.4.3. Programming QoS Fixed Mode

Fixed mode is mode 0. To put the QoS generator in fixed mode, set the registers as shown in the following table.

Table 62. QoS Generator Register Values for Fixed Mode

Register	Field	Value
I_main_QosGenerator_Mode	MODE	0
I_main_QosGenerator_Priority	P0	Urgency for write transactions (default=0)
I_main_QosGenerator_Priority	P1	Urgency for read transactions (default=1)

Higher priority (urgency) values mean that a packet receives preferential treatment at each arbitration node.

Related Information

QoS Generator Registers on page 244
List of QoS registers

8.3.4.4. Bandwidth and Saturation

Encoding Bandwidth Values

The QoS bandwidth setting is frequency-dependent. Low-frequency masters require a larger value to achieve the same bandwidth as higher frequency masters.

You can calculate the register value for a bandwidth as follows:

`*I_main_QosGenerator_Bandwidth` = (bandwidth / frequency) * 256

where bandwidth is in MBps and frequency is in MHz.

For example, to configure the FPGA-to-SDRAM 0 QoS generator for a bandwidth of 1000 MBps at 200 MHz, calculate the register value as follows:

`fpga2sdram0_axi128_I_main_QosGenerator_Bandwidth` = (1000 / 200) * 256 = 1280.

Encoding Saturation Values

The QoS saturation represents the period of time that the master bandwidth is evaluated. The saturation value represents units of 16 bytes of data transferred. For example a 64-bit master that should have the bandwidth re-evaluated every 100 clock cycles would use a saturation setting of 100 cycles * 8 bytes / 16 bytes =50

The saturation register controls the number of bytes that the initiator can transmit at full speed before the limiter or regulator takes effect. The register encodes the byte count as a multiple of 16 bytes. Therefore you can calculate the register value as follows:

`*I_main_QosGenerator_Saturation` = nbytes / 16

For example, to let the MPU initiator send 64 bytes at full speed, calculate the register value as follows:

`mpu_m1_I_main_QosGenerator_Saturation` = 64 / 16 = 4

Related Information

QoS Generator Registers on page 244
List of QoS registers

8.3.4.5. QoS Generator Registers

There is a register set for each QoS generator.

Each QoS generator's register set includes the register types shown in the following table.

Table 63. QoS Generator Register Types

Name	Purpose
`I_main_QosGenerator_Priority`	Specifies master urgencies. Register usage depends on the QoS generator mode.
`I_main_QosGenerator_Mode`	Specifies the QoS generator mode (limiter, regulator, fixed, or bypass).
`I_main_QosGenerator_Bandwidth`	Specifies the bandwidth threshold for limiter and regulator modes.
`I_main_QosGenerator_Saturation`	Specifies the bandwidth measurement time window for limiter and regulator modes. (17)

The actual register names consist of the register type plus a prefix specific to the QoS generator. For example, the registers for MPU subsystem L2 cache master 0 are:

- `mpu_m0_I_main_QosGenerator_Priority`
- `mpu_m0_I_main_QosGenerator_Mode`
- `mpu_m0_I_main_QosGenerator_Bandwidth`
- `mpu_m0_I_main_QosGenerator_Saturation`
- `mpu_m0_I_main_QosGenerator_ExtControl`

The following table lists the prefixes for all QoS generators.

(17) For detailed information about setting bandwidth and saturation, refer to "Bandwidth and Saturation".

Table 64. QoS Register Name Prefixes

Initiator	Register Name Prefix
MPU Subsystem L2 Cache Master 0	mpu_m0_
MPU Subsystem L2 Cache Master 1	mpu_m1_
FPGA-to-HPS Bridge	fpga2soc_axi32_
FPGA-to-HPS Bridge	fpga2soc_axi64_
FPGA-to-HPS Bridge	fpga2soc_axi128_
DMA Controller	dma_m0_
Ethernet MAC 0	emac0_m_
Ethernet MAC 1	emac1_m_
Ethernet MAC 2	emac2_m_
USB-2 OTG 0	usb0_m_
USB-2 OTG 1	usb1_m_
NAND Flash Controller	nand_m_
SD/MMC Controller	sdmmc_m_
FPGA-to-SDRAM Port 0	fpga2sdram0_axi32_
FPGA-to-SDRAM Port 0	fpga2sdram0_axi64_
FPGA-to-SDRAM Port 0	fpga2sdram0_axi128_
FPGA-to-SDRAM Port 1	fpga2sdram1_axi32_
FPGA-to-SDRAM Port 1	fpga2sdram1_axi64_
FPGA-to-SDRAM Port 2	fpga2sdram2_axi32_
FPGA-to-SDRAM Port 2	fpga2sdram2_axi64_
FPGA-to-SDRAM Port 2	fpga2sdram2_axi128_

Related Information

Bandwidth and Saturation on page 243

8.3.4.6. QoS Programming Examples

8.3.4.6.1. Example: One Master Always Takes Precedence

In this example, one particular master's packets always take precedence.

Consider a system that includes three masters, A, B, and C. In this system, we require that master A always takes precedence over master B at each arbitration node, and master B always takes precedence over master C. To implement this arbitration scheme, we configure all QoS generators in fixed mode, and assign appropriate values for read and write urgency, as shown in the following table:

Table 65. Fixed Example Settings

Master	QoS Mode	P1 (Read Urgency)	P0 (Write Urgency)
A	Fixed (mode 0)	0x3	0x3
B	Fixed (mode 0)	0x2	0x2
C	Fixed (mode 0)	0x1	0x1

In fixed mode, masters by default have a read urgency of 1 and write urgency of 0. So master C has equal urgency for reads and higher urgency for writes compared to all the other masters in the system.

8.3.4.6.2. Example: Tuning for Specific Throughput Requirements

In this example, we require each master to meet specific throughput requirements.

Consider a system in which masters A, B, and C have the following throughput requirements:

- Master A: 1000 MBps operating at 100 MHz
- Master B: 800 MBps operating at 800 MHz
- Master C: 500 MBps operating at 400 Mhz

We achieve this by using the QoS generators in regulator mode. There are other masters in the system, and these three masters may need to "steal" bandwidth from them to achieve the required throughputs. Therefore, all the other QoS generators are configured in limiter mode to cap their bandwidth usage.

To set up the QoS generators for masters A, B, and C, we can use the settings in the following table.

Table 66. Regulator Example Settings

Master	QoS Mode	QoS Bandwidth	QoS Saturation (register value) [18]
A	Regulator	2560	64 bytes (saturation = 4)
B	Regulator	256	256 bytes (saturation = 16)
C	Regulator	320	128 bytes (saturation = 8)

Not shown are the master urgencies. When a QoS regulator achieves the required bandwidth the urgency is downgraded to allow other masters to achieve their required bandwidth.

[18] The QoS bandwidth register value is computed as (desired bandwidth (MBps) * 256) / master frequency (MHz)

We also set the urgency threshold values, P1 and P0. P1 is the urgency when a master falls below the bandwidth threshold. P0 is the urgency when a master is above the bandwidth threshold. regulator mode ensures a minimum bandwidth to the master, by raising the urgency when the bandwidth falls below the threshold, and downgrading it when the bandwidth is back above the threshold.

For any master the value of P1 must be greater or equal than P0.

8.4. System Interconnect Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

9. HPS-FPGA Bridges

This chapter describes the bridges in the hard processor system (HPS) used to communicate data between the FPGA fabric and HPS logic.

These bridges make use of Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) protocol, based on Arteris FlexNoC network-on-chip (NOC) interconnect IP.

The HPS contains the following HPS-FPGA bridges:

- FPGA-to-HPS Bridge
- HPS-to-FPGA Bridge
- Lightweight HPS-to-FPGA Bridge

Related Information

- www.arteris.com
  For detailed information about the FlexNoC Network-on-Chip Interconnect, refer to the Arteris website.
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter
- SoC Security on page 168
  Information about security features in the HPS-FPGA bridges
- ARM Infocenter
  Additional information is available in the AMBA AXI Protocol Specification v1.0, which you can download from the ARM Infocenter website.

9.1. Features of the HPS-FPGA Bridges

The HPS-FPGA bridges allow masters in the FPGA fabric to communicate with slaves in the HPS logic and vice versa. For example, if you implement memories or peripherals in the FPGA fabric, HPS components such as the MPU can access them. Components implemented in the FPGA fabric, such as the Nios® II processor, can also access memories and peripherals in the HPS logic.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Table 67. HPS-FPGA Bridge Features

Feature	FPGA-to-HPS Bridge	HPS-to-FPGA Bridge	Lightweight HPS-to-FPGA Bridge
Supports the AMBA AXI3 interface and FlexNoC low latency interface protocols	Y	Y	Y
Implements clock crossing and manages the transfer of data across the clock domains in the HPS logic and the FPGA fabric	Y	Y	Y
Performs data width conversion between the HPS logic and the FPGA fabric	Y	Y	Y
Allows configuration of FPGA interface widths at instantiation time	Y	Y	N

Each bridge consists of a master-slave pair with one interface exposed to the FPGA fabric and the other exposed to the HPS logic. The FPGA-to-HPS bridge exposes an AXI slave interface that you can connect to AXI or Avalon-MM master interfaces in the FPGA fabric. The HPS-to-FPGA and lightweight HPS-to-FPGA bridges expose an AXI master interface that you can connect to AXI or Avalon-MM slave interfaces in the FPGA fabric.

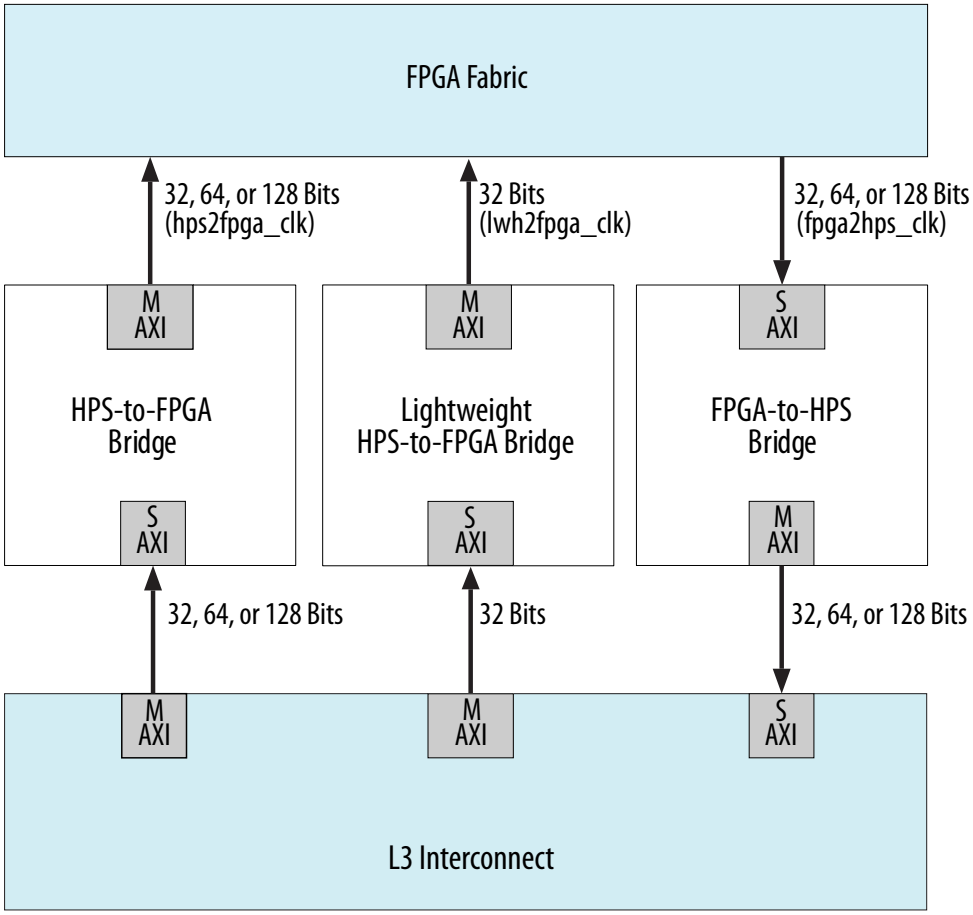
Related Information

AXI Bridges on page 722
Information about configuring the AXI bridges

9.2. Arria 10 HPS-FPGA Bridges Block Diagram and System Integration

Figure 39. HPS-FPGA Bridge Connectivity

The following figure shows the HPS-FPGA bridges in the context of the FPGA fabric and the L3 interconnect to the HPS. Each master (M) and slave (S) interface is shown with its data width(s). The clock domain for each interconnect is shown in parentheses.



The HPS-to-FPGA and lightweight HPS-to-FPGA bridges are both mastered by the level 3 (L3) interconnect. The FPGA-to-HPS bridge masters the L3 interconnect. This arrangement allows any master implemented in the FPGA fabric to access most slaves in the HPS. For example, the FPGA-to-HPS bridge can access the accelerator coherency port (ACP) of the MPU subsystem to perform cache-coherent accesses to the SDRAM subsystem.

9.3. Functional Description of the HPS-FPGA Bridges

9.3.1. Functional Description of the FPGA-to-HPS Bridge

The FPGA-to-HPS bridge provides access to the peripherals in the HPS from the FPGA. This access is available to any master implemented in the FPGA fabric. You can configure the bridge slave, which is exposed to the FPGA fabric, to support the AXI protocol, with a data width of 32, 64 or 128 bits. The FPGA-to-HPS bridge multiplexes the configured data width from the L3 interconnect to the FPGA interface.

Table 68. FPGA-to-HPS Bridge Properties

The following table lists the properties of the FPGA-to-HPS bridge, including the configurable slave interface exposed to the FPGA fabric.

Bridge Property	Value
Data width[19]	32, 64, or 128 bits
Clock domain	`fpga2hps_clk`
Byte address width	32 bits
ID width	4 bits
Read acceptance	8 transactions
Write acceptance	8 transactions
Total acceptance	16 transactions

The FPGA-to-HPS bridge is configurable in the HPS component parameter editor, available in Platform Designer and the IP Catalog. The HPS component parameter editor allows you to set the data path width and the bridge protocol, according to the FPGA bitstream.

Warning: To avoid memory aliasing issues, ensure that Avalon-MM burst transactions into the HPS do not cross the 4 KB address boundary restriction specified by the AXI protocol.

9.3.1.1. FPGA-to-HPS Access to ACP

When the error correction code (ECC) option is enabled in the level 2 (L2) cache controller, all accesses from the FPGA-to-HPS bridge to the ACP must be 64 bits wide and aligned on 8-byte boundaries after up- or downsizing takes place. Ensure that the address alignment is a multiple of 8 bytes and that (burst size) × (burst length) is a multiple of 8 bytes.

9.3.1.2. FPGA-to-HPS Bridge Slave Signals

The FPGA-to-HPS bridge slave address channels support user-sideband signals, routed to the ACP in the MPU subsystem. All the signals have a fixed width except the data and write strobes for the read and write data channels. The variable width signals depend on the data width setting of the bridge.

The FPGA-to-HPS bridge incorporates Arm's TrustZone technology by providing the `ARPROT[1]` and `AWPROT[1]` signals, which specify whether a transaction is secure or nonsecure. The firewall logic uses the `AxPROT` signals to determine if a bus transaction matches the security level allowed. You can program security values in the Secure Configuration Registers of the system interconnect.

(19) The bridge slave data width is user-configurable at the time you instantiate the HPS component in your system.

All peripheral slaves and memories in the SoC are secure when they are released from reset.

The following tables list all the signals exposed by the FPGA-to-HPS slave interface to the FPGA fabric.

Table 69. FPGA-to-HPS Bridge Slave Write Address Channel Signals

Signal	Width	Direction	Description
AWID	8 bits	Input	Write address ID
AWADDR	32 bits	Input	Write address
AWLEN	4 bits	Input	Burst length
AWSIZE	3 bits	Input	Burst size
AWBURST	2 bits	Input	Burst type
AWLOCK	2 bits	Input	Lock type—Valid values are 00 (normal access) and 01 (exclusive access)
AWCACHE	4 bits	Input	Cache policy type
AWPROT	3 bits	Input	Protection type
AWVALID	1 bit	Input	Write address channel valid
AWREADY	1 bit	Output	Write address channel ready
AWUSER	5 bits	Input	User sideband signals

Table 70. FPGA-to-HPS Bridge Slave Write Data Channel Signals

Signal	Width	Direction	Description
WID	8 bits	Input	Write ID
WDATA	32, 64, or 128 bits	Input	Write data
WSTRB	4, 8, or 16 bits	Input	Write data strobes
WLAST	1 bit	Input	Write last data identifier
WVALID	1 bit	Input	Write data channel valid
WREADY	1 bit	Output	Write data channel ready

Table 71. FPGA-to-HPS Bridge Slave Write Response Channel Signals

Signal	Width	Direction	Description
BID	8 bits	Output	Write response ID
BRESP	2 bits	Output	Write response
BVALID	1 bit	Output	Write response channel valid
BREADY	1 bit	Input	Write response channel ready

Table 72. FPGA-to-HPS Bridge Slave Read Address Channel Signals

Signal	Width	Direction	Description
ARID	8 bits	Input	Read address ID
ARADDR	32 bits	Input	Read address
ARLEN	4 bits	Input	Burst length
ARSIZE	3 bits	Input	Burst size
ARBURST	2 bits	Input	Burst type
ARLOCK	2 bits	Input	Lock type—Valid values are 00 (normal access) and 01 (exclusive access)
ARCACHE	4 bits	Input	Cache policy type
ARPROT	3 bits	Input	Protection type
ARVALID	1 bit	Input	Read address channel valid
ARREADY	1 bit	Output	Read address channel ready
ARUSER	5 bits	Input	Read user sideband signals

Table 73. FPGA-to-HPS Bridge Slave Read Data Channel Signals

Signal	Width	Direction	Description
RID	8 bits	Output	Read ID
RDATA	32, 64, or 128 bits	Output	Read data
RRESP	2 bits	Output	Read response
RLAST	1 bit	Output	Read last data identifier
RVALID	1 bit	Output	Read data channel valid
RREADY	1 bit	Input	Read data channel ready

Related Information

SoC Security on page 168
More information about Secure Firewall Bus Transactions

9.3.1.3. FPGA2HPS AXI Bridge Module Address Map

Registers in the FPGA2HPS AXI Bridge Module.

Base Address: 0xFF600000

ID Register Group

Register	Offset	Width	Access	Reset Value	Description
periph_id_4 on page 254	0x1FD0	32	RO	0x4	Peripheral ID4 Register
periph_id_0 on page 255	0x1FE0	32	RO	0x1	Peripheral ID0 Register
periph_id_1 on page 255	0x1FE4	32	RO	0xB3	Peripheral ID1 Register
periph_id_2 on page 256	0x1FE8	32	RO	0x6B	Peripheral ID2 Register

continued...

Register	Offset	Width	Access	Reset Value	Description
periph_id_3 on page 256	0x1FEC	32	RO	0x0	Peripheral ID3 Register
comp_id_0 on page 257	0x1FF0	32	RO	0xD	Component ID0 Register
comp_id_1 on page 258	0x1FF4	32	RO	0xF0	Component ID1 Register
comp_id_2 on page 258	0x1FF8	32	RO	0x5	Component ID2 Register
comp_id_3 on page 259	0x1FFC	32	RO	0xB1	Component ID3 Register

32-bit Slave

Register	Offset	Width	Access	Reset Value	Description
fn_mod2 on page 259	0x42024	32	RW	0x0	Functionality Modification 2 Register
fn_mod on page 260	0x42108	32	RW	0x0	Issuing Functionality Modification Register

128-bit Slave

Register	Offset	Width	Access	Reset Value	Description
fn_mod2 on page 261	0x44024	32	RW	0x0	Functionality Modification 2 Register
fn_mod on page 262	0x44108	32	RW	0x0	Issuing Functionality Modification Register

9.3.1.3.1. ID Register Group Register Descriptions

Contains registers that identify the Arm NIC-301 IP Core.

Offset: 0x1000

periph_id_4 on page 254
periph_id_0 on page 255
periph_id_1 on page 255
periph_id_2 on page 256
periph_id_3 on page 256
comp_id_0 on page 257
comp_id_1 on page 258
comp_id_2 on page 258
comp_id_3 on page 259

periph_id_4

JEP106 continuation code

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FD0

Offset: 0x1FD0

Access: R0

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								periph_id_4 R0 0x4							

periph_id_4 Fields

Bit	Name	Description	Access	Reset
7:0	periph_id_4	JEP106 continuation code	R0	0x4

periph_id_0

Peripheral ID0

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FE0

Offset: 0x1FE0

Access: R0

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								pn7to0 R0 0x1							

periph_id_0 Fields

Bit	Name	Description	Access	Reset
7:0	pn7to0	Part Number [7:0]	R0	0x1

periph_id_1

Peripheral ID1

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FE4

Offset: 0x1FE4

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								jep3to0_pn11to8 RO 0xB3							

periph_id_1 Fields

Bit	Name	Description	Access	Reset
7:0	jep3to0_pn11to8	JEP106[3:0], Part Number [11:8]	RO	0xB3

periph_id_2

Peripheral ID2

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FE8

Offset: 0x1FE8

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								rev_jepcode_jep6to4 RO 0x6B							

periph_id_2 Fields

Bit	Name	Description	Access	Reset
7:0	rev_jepcode_jep6to4	Revision, JEP106 code flag, JEP106[6:4]	RO	0x6B

periph_id_3

Peripheral ID3

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FEC

Offset: 0x1FEC

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								rev_and RO 0x0				cust_mod_num RO 0x0			

periph_id_3 Fields

Bit	Name	Description	Access	Reset
7:4	rev_and	Revision	RO	0x0
3:0	cust_mod_num	Customer Model Number	RO	0x0

comp_id_0

Component ID0

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FF0

Offset: 0x1FF0

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								preamble RO 0xD							

comp_id_0 Fields

Bit	Name	Description	Access	Reset
7:0	preamble	Preamble	RO	0xD

comp_id_1

Component ID1

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FF4

Offset: 0x1FF4

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								genipcompcls_preamble RO 0xF0							

comp_id_1 Fields

Bit	Name	Description	Access	Reset
7:0	genipcompcls_preamble	Generic IP component class, Preamble	RO	0xF0

comp_id_2

Component ID2

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FF8

Offset: 0x1FF8

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								preamble RO 0x5							

comp_id_2 Fields

Bit	Name	Description	Access	Reset
7:0	preamble	Preamble	RO	0x5

comp_id_3

Component ID3

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF601FFC

Offset: 0x1FFC

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								preamble RO 0xB1							

comp_id_3 Fields

Bit	Name	Description	Access	Reset
7:0	preamble	Preamble	RO	0xB1

9.3.1.3.2. Slave Register Group Register Descriptions

Registers associated with slave interfaces.

Offset: 0x42000

32-bit Slave Register Descriptions

Registers associated with the 32-bit AXI slave interface. These registers are only active when the FPGA2HPS AXI Bridge is configured with a 32-bit FPGA AXI slave interface.

Offset: 0x0

fn_mod2 on page 259
fn_mod on page 260

fn_mod2

Controls bypass merge of upsizing/downsizing.

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF642024

Offset: 0x42024

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															bypass_merge RW 0x0

fn_mod2 Fields

Bit	Name	Description	Access	Reset
0	bypass_merge	Controls bypass merge of upsizing/downsizing. Value: 0x0 — The network can alter transactions. Value: 0x1 — The network does not alter any transactions that could pass through the upsizer legally without alteration.	RW	0x0

fn_mod

Sets the block issuing capability to multiple or single outstanding transactions.

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF642108

Offset: 0x42108

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														wr RW 0x0	rd RW 0x0

fn_mod Fields

Bit	Name	Description	Access	Reset
1	wr	Value: 0x0 — Multiple outstanding write transactions Value: 0x1 — Only a single outstanding write transaction	RW	0x0
0	rd	Value: 0x0 — Multiple outstanding read transactions Value: 0x1 — Only a single outstanding read transaction	RW	0x0

128-bit Slave Register Descriptions

Registers associated with the 128-bit AXI slave interface. These registers are only active when the FPGA2HPS AXI Bridge is configured with a 128-bit FPGA AXI slave interface.

Offset: 0x2000

fn_mod2 on page 261
fn_mod on page 262

fn_mod2

Controls bypass merge of upsizing/downsizing.

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF644024

Offset: 0x44024

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															bypass_merge RW 0x0

fn_mod2 Fields

Bit	Name	Description	Access	Reset
0	bypass_merge	Controls bypass merge of upsizing/downsizing. Value: 0x0 — The network can alter transactions. Value: 0x1 — The network does not alter any transactions that could pass through the upsizer legally without alteration.	RW	0x0

fn_mod

Sets the block issuing capability to multiple or single outstanding transactions.

Module Instance	Base Address	Register Address
fpga2hpsregs	0xFF600000	0xFF644108

Offset: 0x44108

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														wr RW 0x0	rd RW 0x0

fn_mod Fields

Bit	Name	Description	Access	Reset
1	wr		RW	0x0

continued...

Bit	Name	Description	Access	Reset
		Value / Description: 0x0 — Multiple outstanding write transactions; 0x1 — Only a single outstanding write transaction		
0	rd	Value / Description: 0x0 — Multiple outstanding read transactions; 0x1 — Only a single outstanding read transaction	RW	0x0

9.3.2. Functional Description of the HPS-to-FPGA Bridge

The HPS-to-FPGA bridge provides a configurable-width, high-performance master interface to the FPGA fabric. The bridge provides most masters in the HPS with access to logic and peripherals implemented in the FPGA. The effective size of the address space is 0x3FFF0000, or 1 gigabyte (GB) minus the 64 megabytes (MB) occupied by peripherals, lightweight HPS-to-FPGA bridge, and on-chip RAM in the HPS. You can configure the bridge master exposed to the FPGA fabric for 32-, 64-, or 128-bit data. The amount of address space exposed to the MPU subsystem can also be reduced through the L2 cache address filtering mechanism.

The HPS-to-FPGA bridge multiplexes the configured data width from the L3 interconnect to the FPGA interface. The bridge provides width adaptation and clock crossing logic that allows the logic in the FPGA to operate in any clock domain, asynchronous from the HPS.

Warning: The HPS-to-FPGA bridge is accessed if the MPU boots from the FPGA. Before the MPU boots from the FPGA, the FPGA portion of the SoC device must be configured, and the HPS-to-FPGA bridge must be remapped into addressable space. Otherwise, access to the HPS-to-FPGA bridge during the boot process results in a bus error.

Table 74. HPS-to-FPGA Bridge Properties

The following table lists the properties of the HPS-to-FPGA bridge, including the configurable master interface exposed to the FPGA fabric.

Bridge Property	Value
Data width [20]	32, 64, or 128 bits
Clock domain	hps2fpga_clk
Byte address width	30 bits
ID width	4 bits
Read acceptance	16 transactions
Write acceptance	16 transactions
Total acceptance	16 transactions

(20) The bridge master data width is user-configurable at the time you instantiate the HPS component in your system.

The HPS-to-FPGA bridge is configurable in the HPS component parameter editor, available in Platform Designer and the IP Catalog. The HPS component parameter editor allows you to set the data path width and the bridge protocol, according to the FPGA bitstream.

Related Information

- Functional Description of the System Interconnect on page 0
  Detailed information about connectivity, such as which masters have access to each bridge
- Cortex-A9 Microprocessor Unit Subsystem on page 282
  Details about L2 cache address filtering
- AXI Bridges on page 722
  Information about configuring the AXI bridges

9.3.2.1. HPS-to-FPGA Bridge Master Signals

All the HPS-to-FPGA bridge master signals have a fixed width except the data and write strobes for the read and write data channels. The variable-width signals depend on the data width setting of the bridge interface exposed to the FPGA logic.

The HPS-to-FPGA bridge incorporates Arm's TrustZone technology by providing the ARPROT[1] and AWPROT[1] signals, which specify whether a transaction is secure or nonsecure. The firewall logic uses the AxPROT signals to determine if a bus transaction matches the security level allowed. You can program security values in the Secure Configuration Registers of the system interconnect.

All peripheral slaves and memories in the SoC are secure when they are released from reset.

The following tables list all the signals exposed by the HPS-to-FPGA master interface to the FPGA fabric.

Table 75. HPS-to-FPGA Bridge Master Write Address Channel Signals

Signal	Width	Direction	Description
AWID	12 bits	Output	Write address ID
AWADDR	30 bits	Output	Write address
AWLEN	4 bits	Output	Burst length
AWSIZE	3 bits	Output	Burst size
AWBURST	2 bits	Output	Burst type
AWLOCK	2 bits	Output	Lock type—Valid values are 00 (normal access) and 01 (exclusive access)
AWCACHE	4 bits	Output	Cache policy type
AWPROT	3 bits	Output	Protection type
AWVALID	1 bit	Output	Write address channel valid
AWREADY	1 bit	Input	Write address channel ready

Table 76. HPS-to-FPGA Bridge Master Write Data Channel Signals

Signal	Width	Direction	Description
WID	12 bits	Output	Write ID
WDATA	32, 64, or 128 bits	Output	Write data
WSTRB	4, 8, or 16 bits	Output	Write data strobes
WLAST	1 bit	Output	Write last data identifier
WVALID	1 bit	Output	Write data channel valid
WREADY	1 bit	Input	Write data channel ready

Table 77. HPS-to-FPGA Bridge Master Write Response Channel Signals

Signal	Width	Direction	Description
BID	12 bits	Input	Write response ID
BRESP	2 bits	Input	Write response
BVALID	1 bit	Input	Write response channel valid
BREADY	1 bit	Output	Write response channel ready

Table 78. HPS-to-FPGA Bridge Master Read Address Channel Signals

Signal	Width	Direction	Description
ARID	12 bits	Output	Read address ID
ARADDR	30 bits	Output	Read address
ARLEN	4 bits	Output	Burst length
ARSIZE	3 bits	Output	Burst size
ARBURST	2 bits	Output	Burst type
ARLOCK	2 bits	Output	Lock type—Valid values are 00 (normal access) and 01 (exclusive access)
ARCACHE	4 bits	Output	Cache policy type
ARPROT	3 bits	Output	Protection type
ARVALID	1 bit	Output	Read address channel valid
ARREADY	1 bit	Input	Read address channel ready

Table 79. HPS-to-FPGA Bridge Master Read Data Channel Signals

Signal	Width	Direction	Description
RID	12 bits	Input	Read ID
RDATA	32, 64, or 128 bits	Input	Read data
RRESP	2 bits	Input	Read response
RLAST	1 bit	Input	Read last data identifier
RVALID	1 bit	Input	Read data channel valid
RREADY	1 bit	Output	Read data channel ready

Related Information

SoC Security on page 168
More information about Secure Firewall Bus Transactions

9.3.2.2. HPS2FPGA AXI Bridge Module Address Map

Registers in the HPS2FPGA AXI Bridge Module.

Base Address: 0xFF500000

ID Register Group

Register	Offset	Width	Access	Reset Value	Description
periph_id_4 on page 267	0x1FD0	32	RO	0x4	Peripheral ID4 Register
periph_id_0 on page 267	0x1FE0	32	RO	0x1	Peripheral ID0 Register
periph_id_1 on page 268	0x1FE4	32	RO	0xB3	Peripheral ID1 Register
periph_id_2 on page 268	0x1FE8	32	RO	0x6B	Peripheral ID2 Register
periph_id_3 on page 269	0x1FEC	32	RO	0x0	Peripheral ID3 Register
comp_id_0 on page 269	0x1FF0	32	RO	0xD	Component ID0 Register
comp_id_1 on page 270	0x1FF4	32	RO	0xF0	Component ID1 Register
comp_id_2 on page 270	0x1FF8	32	RO	0x5	Component ID2 Register
comp_id_3 on page 271	0x1FFC	32	RO	0xB1	Component ID3 Register

32-bit Master

Register	Offset	Width	Access	Reset Value	Description
fn_mod2 on page 272	0x2024	32	RW	0x0	Functionality Modification 2 Register
fn_mod on page 272	0x2108	32	RW	0x0	Issuing Functionality Modification Register

128-bit Master

Register	Offset	Width	Access	Reset Value	Description
fn_mod2 on page 273	0x4024	32	RW	0x0	Functionality Modification 2 Register
fn_mod on page 274	0x4108	32	RW	0x0	Issuing Functionality Modification Register

9.3.2.2.1. ID Register Group Register Descriptions

Contains registers that identify the Arm NIC-301 IP Core.

Offset: 0x1000

periph_id_4 on page 267

periph_id_0 on page 267

periph_id_1 on page 268

periph_id_2 on page 268
periph_id_3 on page 269
comp_id_0 on page 269
comp_id_1 on page 270
comp_id_2 on page 270
comp_id_3 on page 271

periph_id_4

JEP106 continuation code

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FD0

Offset: 0x1FD0

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								periph_id_4 RO 0x4							

periph_id_4 Fields

Bit	Name	Description	Access	Reset
7:0	periph_id_4	JEP106 continuation code	RO	0x4

periph_id_0

Peripheral ID0

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FE0

Offset: 0x1FE0

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								pn7to0 RO 0x1							

periph_id_0 Fields

Bit	Name	Description	Access	Reset
7:0	pn7to0	Part Number [7:0]	RO	0x1

periph_id_1

Peripheral ID1

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FE4

Offset: 0x1FE4

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								jep3to0_pn11to8 RO 0xB3							

periph_id_1 Fields

Bit	Name	Description	Access	Reset
7:0	jep3to0_pn11to8	JEP106[3:0], Part Number [11:8]	RO	0xB3

periph_id_2

Peripheral ID2

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FE8

Offset: 0x1FE8

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								rev_jepcode_jep6to4 RO 0x6B							

periph_id_2 Fields

Bit	Name	Description	Access	Reset
7:0	rev_jepcode_jep6to4	Revision, JEP106 code flag, JEP106[6:4]	RO	0x6B

periph_id_3

Peripheral ID3

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FEC

Offset: 0x1FEC

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								rev_and RO 0x0				cust_mod_num RO 0x0			

periph_id_3 Fields

Bit	Name	Description	Access	Reset
7:4	rev_and	Revision	RO	0x0
3:0	cust_mod_num	Customer Model Number	RO	0x0

comp_id_0

Component ID0

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FF0

Offset: 0x1FF0

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								preamble RO 0xD							

comp_id_0 Fields

Bit	Name	Description	Access	Reset
7:0	preamble	Preamble	RO	0xD

comp_id_1

Component ID1

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FF4

Offset: 0x1FF4

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								genipcompcls_preamble RO 0xF0							

comp_id_1 Fields

Bit	Name	Description	Access	Reset
7:0	genipcompcls_preamble	Generic IP component class, Preamble	RO	0xF0

comp_id_2

Component ID2

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FF8

Offset: 0x1FF8

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								preamble RO 0x5							

comp_id_2 Fields

Bit	Name	Description	Access	Reset
7:0	preamble	Preamble	RO	0x5

comp_id_3

Component ID3

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF501FFC

Offset: 0x1FFC

Access: RO

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								preamble RO 0xB1							

comp_id_3 Fields

Bit	Name	Description	Access	Reset
7:0	preamble	Preamble	RO	0xB1

9.3.2.2.2. Master Register Group Register Descriptions

Registers associated with master interfaces.

Offset: 0x2000

32-bit Master Register Descriptions

Registers associated with the 32-bit AXI master interface. These registers are only active when the HPS2FPGA AXI Bridge is configured with a 32-bit FPGA AXI master interface.

Offset: 0x0

fn_mod2 on page 272
fn_mod on page 272

fn_mod2

Controls bypass merge of upsizing/downsizing.

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF502024

Offset: 0x2024

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															bypass_merge RW 0x0

fn_mod2 Fields

Bit	Name	Description	Access	Reset
0	bypass_merge	Controls bypass merge of upsizing/downsizing. Value: 0x0 — The network can alter transactions. Value: 0x1 — The network does not alter any transactions that could pass through the upsizer legally without alteration.	RW	0x0

fn_mod

Sets the block issuing capability to multiple or single outstanding transactions.

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF502108

Offset: 0x2108

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														wr RW 0x0	rd RW 0x0

fn_mod Fields

Bit	Name	Description	Access	Reset
1	wr	Value: 0x0 — Multiple outstanding write transactions Value: 0x1 — Only a single outstanding write transaction	RW	0x0
0	rd	Value: 0x0 — Multiple outstanding read transactions Value: 0x1 — Only a single outstanding read transaction	RW	0x0

128-bit Master Register Descriptions

Registers associated with the 128-bit AXI master interface. These registers are only active when the HPS2FPGA AXI Bridge is configured with a 128-bit FPGA AXI master interface.

Offset: 0x2000

fn_mod2 on page 273

fn_mod on page 274

fn_mod2

Controls bypass merge of upsizing/downsizing.

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF504024

Offset: 0x4024

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															bypass_merge RW 0x0

fn_mod2 Fields

Bit	Name	Description	Access	Reset
0	bypass_merge	Controls bypass merge of upsizing/downsizing. Value: 0x0 — The network can alter transactions. Value: 0x1 — The network does not alter any transactions that could pass through the upsizer legally without alteration.	RW	0x0

fn_mod

Sets the block issuing capability to multiple or single outstanding transactions.

Module Instance	Base Address	Register Address
hps2fpgaregs	0xFF500000	0xFF504108

Offset: 0x4108

Access: RW

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

Bit Fields															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														wr RW 0x0	rd RW 0x0

fn_mod Fields

Bit	Name	Description	Access	Reset
1	wr	Value: 0x0 — Multiple outstanding write transactions Value: 0x1 — Only a single outstanding write transaction	RW	0x0
0	rd	Value: 0x0 — Multiple outstanding read transactions Value: 0x1 — Only a single outstanding read transaction	RW	0x0

9.3.3. Functional Description of the Lightweight HPS-to-FPGA Bridge

The lightweight HPS-to-FPGA bridge provides a lower-performance interface to the FPGA fabric. This interface is useful for accessing the control and status registers of soft peripherals. The bridge provides a 2 MB address space and access to logic, peripherals, and memory implemented in the FPGA fabric. The MPU subsystem, direct memory access (DMA) controller, and debug access port (DAP) can use the lightweight HPS-to-FPGA bridge to access the FPGA fabric or NoC registers.

The lightweight HPS-to-FPGA bridge has a fixed data width of 32 bits.

Use the lightweight HPS-to-FPGA bridge as a secondary, lower-performance master interface to the FPGA fabric. With a fixed width and a smaller address space, the lightweight bridge is useful for low-bandwidth traffic, such as memory-mapped register accesses to FPGA peripherals. This approach diverts traffic from the high-performance HPS-to-FPGA bridge, and can improve both register access latency and overall system performance.

Warning: Before the lightweight HPS-to-FPGA bridge can be accessed, the FPGA portion of the SoC device must be configured, and the lightweight HPS-to-FPGA bridge must be remapped into addressable space. Otherwise, accesses to the lightweight HPS-to-FPGA bridge results in a bus error.

Table 80. Lightweight HPS-to-FPGA Bridge Properties

This table lists the properties of the lightweight HPS-to-FPGA bridge, including the master interface exposed to the FPGA fabric.

Bridge Property	Value
Data width	32 bits
Clock domain	`lwh2fpga_clk`
Byte address width	21 bits
ID width	4 bits
Read acceptance	16 transactions
Write acceptance	16 transactions
Total acceptance	16 transactions

The lightweight HPS-to-FPGA bridge is configurable in the HPS component parameter editor, available in Platform Designer and the IP Catalog. The HPS component parameter editor allows you to set the bridge protocol, according to the FPGA bitstream.

Related Information

- Arria 10 HPS-FPGA Bridges Block Diagram and System Integration on page 250
  Figure showing the lightweight HPS-to-FPGA bridge's three master interfaces
- Functional Description of the System Interconnect on page 0
  Detailed information about connectivity, such as which masters have access to each bridge
- AXI Bridges on page 722
  Information about configuring the AXI bridges

9.3.3.1. Lightweight HPS-to-FPGA Bridge Master Signals

All the lightweight HPS-to-FPGA bridge master signals have a fixed width.

The lightweight HPS-to-FPGA bridge incorporates Arm's TrustZone technology by providing the `ARPROT[1]` and `AWPROT[1]` signals, which specify whether a transaction is secure or nonsecure. The firewall logic uses the `AxPROT` signals to determine if a bus transaction matches the security level allowed. You can program security values in the Secure Configuration Registers of the system interconnect.

All peripheral slaves and memories in the SoC are secure when they are released from reset.

The following tables list all the signals exposed by the lightweight HPS-to-FPGA master interface to the FPGA fabric.

Table 81. Lightweight HPS-to-FPGA Bridge Master Write Address Channel Signals

Signal	Width	Direction	Description
`AWID`	12 bits	Output	Write address ID
`AWADDR`	21 bits	Output	Write address
`AWLEN`	4 bits	Output	Burst length

continued...

Signal	Width	Direction	Description
AWSIZE	3 bits	Output	Burst size
AWBURST	2 bits	Output	Burst type
AWLOCK	2 bits	Output	Lock type—Valid values are 00 (normal access) and 01 (exclusive access)
AWCACHE	4 bits	Output	Cache policy type
AWPROT	3 bits	Output	Protection type
AWVALID	1 bit	Output	Write address channel valid
AWREADY	1 bit	Input	Write address channel ready

Table 82. Lightweight HPS-to-FPGA Bridge Master Write Data Channel Signals

Signal	Width	Direction	Description
WID	12 bits	Output	Write ID
WDATA	32 bits	Output	Write data
WSTRB	4 bits	Output	Write data strobes
WLAST	1 bit	Output	Write last data identifier
WVALID	1 bit	Output	Write data channel valid
WREADY	1 bit	Input	Write data channel ready

Table 83. Lightweight HPS-to-FPGA Bridge Master Write Response Channel Signals

Signal	Width	Direction	Description
BID	12 bits	Input	Write response ID
BRESP	2 bits	Input	Write response
BVALID	1 bit	Input	Write response channel valid
BREADY	1 bit	Output	Write response channel ready

Table 84. Lightweight HPS-to-FPGA Bridge Master Read Address Channel Signals

Signal	Width	Direction	Description
ARID	12 bits	Output	Read address ID
ARADDR	21 bits	Output	Read address
ARLEN	4 bits	Output	Burst length
ARSIZE	3 bits	Output	Burst size
ARBURST	2 bits	Output	Burst type
ARLOCK	2 bits	Output	Lock type—Valid values are 00 (normal access) and 01 (exclusive access)
ARCACHE	4 bits	Output	Cache policy type
ARPROT	3 bits	Output	Protection type
ARVALID	1 bit	Output	Read address channel valid
ARREADY	1 bit	Input	Read address channel ready

Table 85. Lightweight HPS-to-FPGA Bridge Master Read Data Channel Signals

Signal	Width	Direction	Description
RID	12 bits	Input	Read ID
RDATA	32 bits	Input	Read data
RRESP	2 bits	Input	Read response
RLAST	1 bit	Input	Read last data identifier
RVALID	1 bit	Input	Read data channel valid
RREADY	1 bit	Output	Read data channel ready

Related Information

SoC Security on page 168
More information about Secure Firewall Bus Transactions

9.3.3.2. FPGA Slaves Accessed Via Lightweight HPS2FPGA AXI Bridge (lwfpgaslaves) Address Map

This address space is allocated for FPGA-configured slaves driven by the lightweight HPS-to-FPGA bridge master. Address assignment within this space is user-defined. For more information about Lightweight HPS-to-FPGA bridges, refer to the *HPS-FPGA Bridges* chapter of the *Hard Processor System Technical Reference Manual*.

Table 86. lwfpgaslaves Address Range

Module Instance	Start Address	End Address
LWFPGASLAVES	0xFF200000	0xFF3FFFFF

9.3.4. Clocks and Resets

9.3.4.1. FPGA-to-HPS Bridge Clocks and Resets

The master interface of the bridge in the HPS logic operates in the `l3_main_clk` clock domain. The slave interface exposed to the FPGA fabric operates in the `fpga2hps_clk` clock domain provided by the user logic. The bridge provides clock crossing logic that allows the logic in the FPGA to operate in any clock domain, asynchronous from the HPS.

The FPGA-to-HPS bridge has one reset signal, `fpga2hps_bridge_rst_n`. The reset manager drives this signal to the FPGA-to-HPS bridge on a cold or warm reset.

Related Information

- Clock Manager on page 52
- HPS Component Interfaces on page 731
  Information about the HPS-FPGA bridge clock interfaces

9.3.4.2. HPS-to-FPGA Bridge Clocks and Resets

The master interface into the FPGA fabric operates in the `hps2fpga_clk` clock domain. The clock is provided by user logic. The slave interface of the bridge in the HPS logic operates in the `l3_main_clk` clock domain. The bridge provides clock crossing logic that allows the logic in the FPGA to operate in any clock domain, asynchronous from the HPS.

The HPS-to-FPGA bridge has one reset signal, `hps2fpga_bridge_rst_n`. The reset manager drives this signal to the HPS-to-FPGA bridge on a cold or warm reset.

Note: Make sure that all transactions on the bridge are completed before shutting down and resetting the bridge. If the system is unable to meet this recommendation, the Arria 10 SoC device has a built-in timeout that can be enabled to force all outstanding transactions on the bridge to complete. The NoC timeout is enabled by setting the `en` bit in the System Manager's `noc_timeout` register. Ensure that the bridge operates at 100 MHz or faster when the NoC timeout is enabled. If the bridge is operating below 100 MHz, the timeout does not occur.

Related Information

- Clock Manager on page 52
- HPS Component Interfaces on page 731
  Information about the HPS-FPGA bridge clock interfaces
- System Manager on page 159
  For information about the noc_timeout register, refer to the *System Manager* chapter.

9.3.4.3. Lightweight HPS-to-FPGA Bridge Clocks and Resets

The master interface into the FPGA fabric operates in the `lwh2fpga_clk` clock domain. The clock is provided by custom logic in the FPGA fabric. The slave interface of the bridge in the HPS logic operates in the `l3_main_clk` clock domain. The bridge provides clock crossing logic that allows the logic in the FPGA to operate in any clock domain, asynchronous from the HPS.

The lightweight HPS-to-FPGA bridge has one reset signal, `lwhps2fpga_bridge_rst_n`. The reset manager drives this signal to the lightweight HPS-to-FPGA bridge on a cold or warm reset.

Note: Make sure that all transactions on the bridge are completed before shutting down and resetting the bridge. If the system is unable to meet this recommendation, the Arria 10 SoC device has a built-in timeout that can be enabled to force all outstanding transactions on the bridge to complete. The NoC timeout is enabled by setting the `en` bit in the System Manager's `noc_timeout` register. Ensure that the bridge operates at 100 MHz or faster when the NoC timeout is enabled. If the bridge is operating below 100 MHz, the timeout does not occur.

Related Information

- Clock Manager on page 52
- HPS Component Interfaces on page 731
  Information about the HPS-FPGA bridge clock interfaces

- System Manager on page 159
  For information about the noc_timeout register, refer to the *System Manager* chapter.

9.3.4.4. Taking HPS-FPGA Bridges Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

Related Information

Modules Requiring Software Deassert on page 144
Reset register names

9.3.5. Data Width Sizing

The HPS-to-FPGA and FPGA-to-HPS bridges allow 32-, 64-, and 128-bit interfaces to be exposed to the FPGA fabric. For 32-bit and 128-bit interfaces, the bridge performs data width conversion to the fixed 64-bit interface within the HPS. This conversion is called *upsizing* in the case of data being converted from a 64-bit interface to a 128-bit interface. It is called *downsizing* in the case of data being converted from a 64-bit interface to a 32-bit interface. If an exclusive access is split into multiple transactions, the transactions lose their exclusive access information.

During the upsizing or downsizing process, transactions can also be resized using a data merging technique. For example, in the case of a 32-bit to 64-bit upsizing, if the size of each beat entering the bridge's 32-bit interface is only two bytes, the bridge can merge up to four beats to form a single 64-bit beat. Similarly, in the case of a 128-bit to 64-bit downsizing, if the size of each beat entering the bridge's 128-bit interface is only four bytes, the bridge can merge two beats to form a single 64-bit beat.

9.3.5.1. Ready Latency Support

The HPS-to-FPGA and FPGA-to-HPS bridges support an optional ready latency feature, which allows your design to run at a higher F_{MAX}. When enabled, this feature adds a pipeline stage between the HPS and the FPGA fabric to improve ready/valid handshake timing. Enabling this feature entails a minimal increase in bridge latency.

You can enable this feature when you instantiate the HPS component in Platform Designer.

Related Information

AXI Bridges on page 722
Information about configuring the AXI bridges

9.4. HPS-FPGA Bridges Address Map and Register Definitions for Arria 10

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

10. Cortex-A9 Microprocessor Unit Subsystem

The hard processor system (HPS) in the Intel SoC device includes a stand-alone, full-featured Arm Cortex-A9, dual-core 32-bit application processor. The Cortex-A9 microprocessor unit (MPU) subsystem is composed of a Cortex-A9 MPCore, a level 2 (L2) cache, and debugging modules.

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

10.1. Features of the Cortex-A9 MPU Subsystem

The Intel Cortex-A9 MPU subsystem provides the following features:

- Two Arm Cortex-A9 processors, each with the following support modules:
  - Arm NEON single instruction, multiple data (SIMD) coprocesoor
  - Memory Management Unit (MMU)
  - 32 KB instruction cache
  - 32 KB data cache
  - Private interval timer
  - Private watchdog timer
- Interrupt controller
- Global timer
- Snoop control unit (SCU)
- Accelerator Coherency Port (ACP)
- Arm L2 cache controller
- TrustZone system security extensions
- Symmetric multiprocessing (SMP) and asymmetric multiprocessing (AMP) modes
- Debugging modules

The Cortex-A9 MPU incorporates the following core and L2 cache versions:

Table 87. Cortex-A9 MPU Module Versions

Feature	Version
Cortex-A9 Core	r4p1
L2-310 Level 2 Cache Controller	r3p3

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

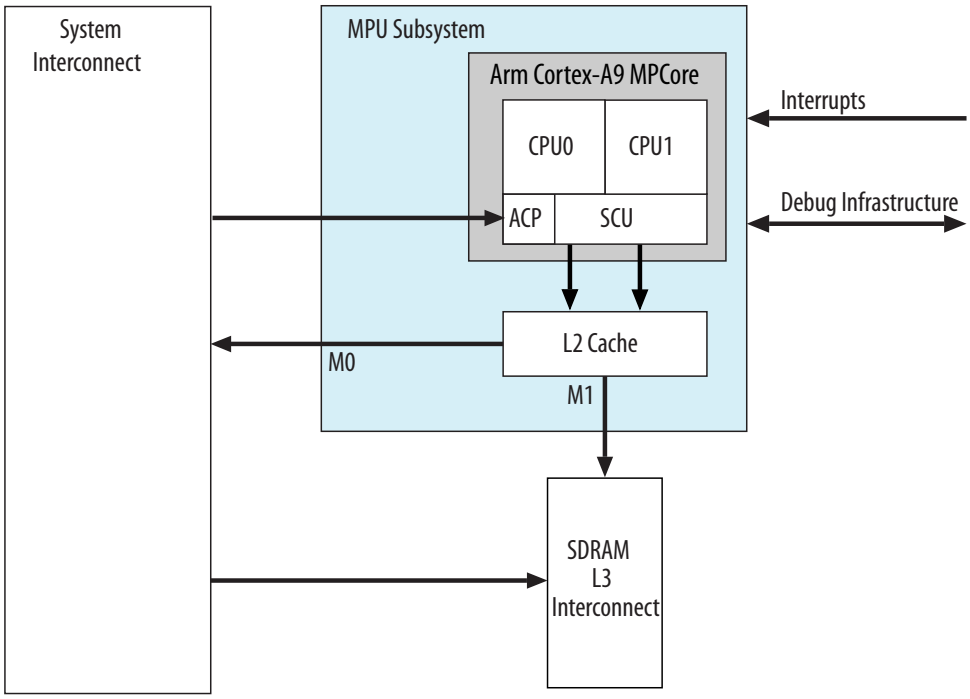
ISO 9001:2015 Registered

10.2. Cortex-A9 MPU Subsystem Block Diagram and System Integration

10.2.1. Cortex-A9 MPU Subsystem with System Interconnect

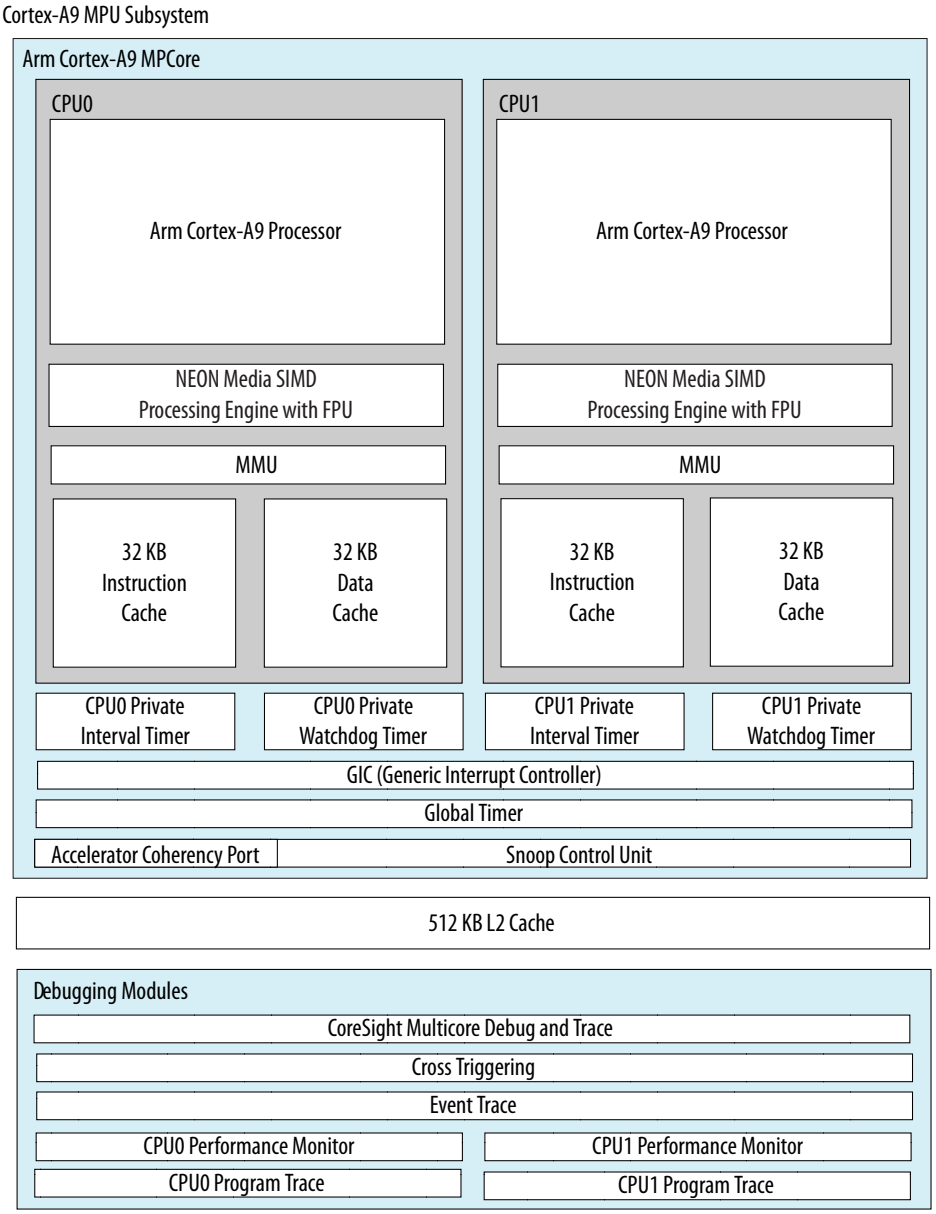
This block diagram shows a dual-core MPU subsystem in the context of the HPS, with the L2 cache. The L2 cache can access either the system interconnect or the SDRAM L3 Interconnect.

Figure 40. Cortex-A9 MPU Subsystem with Interconnect Block Diagram



10.2.2. Cortex-A9 MPU Subsystem Internals

Figure 41. Cortex-A9 MPU Subsystem Block Diagram



10.3. Cortex-A9 MPCore

The MPU subsystem includes a stand-alone, full-featured Arm Cortex-A9 MPCore dual-core 32-bit application processor. The processor, like other HPS masters, can access soft IP in the FPGA fabric through the HPS-to-FPGA bridges.

10.3.1. Functional Description

The Arm Cortex-A9 MPCore contains the following sub-modules:

- Two Cortex-A9 Revision r4p1 processors operating in SMP or AMP mode
- Snoop control unit (SCU)
- Private interval timer for each processor core
- Private watchdog timer for each processor core
- Global timer
- Interrupt controller

Each transaction originating from the Cortex-A9 MPU subsystem can be flagged as secure or non-secure.

Related Information

Cortex-A9 MPU Subsystem Register Implementation on page 0
For more information regarding the Cortex MPU Subsystem Address Map, refer to the *Cortex-A9 MPU Subsystem Register Implementation* section.

10.3.2. Implementation Details

Table 88. Cortex-A9 MPCore Processor Configuration

This table shows the parameter settings for the Cortex-A9 MPCore.

Feature	Options
Cortex-A9 processors	2
Instruction cache size per Cortex-A9 processor	32 KB
Data cache size per Cortex-A9 processor	32 KB
TLB size per Cortex-A9 processor	512 entries
Media Processing Engine with NEON technology per Cortex-A9 processor(21)	Included
Preload Engine per Cortex-A9 processor	Included
Number of entries in the Preload Engine FIFO per Cortex-A9 processor	16
Jazelle DBX extension per Cortex-A9 processor	Full
Program Trace Macrocell (PTM) interface per Cortex-A9 processor	Included
Support for parity error detection(22)	Included
Master ports	Two
Accelerator Coherency Port	Included

Related Information

Arm Infocenter
For more information regarding the Cortex-A9 MPCore features and functions.

10.3.3. Cortex-A9 Processor

Each Cortex-A9 processor includes the following hardware blocks:

- Arm NEON single instruction, multiple data (SIMD) coprocessor with vector floating-point (VFP) v3 double-precision floating point unit for media and signal processing acceleration
  - Single- and double-precision IEEE-754 floating point math support
  - Integer and polynomial math support
- Level 1 (L1) cache with parity checking
  - 32 KB four-way set-associative instruction cache
  - 32 KB four-way set-associative data cache
- CoreSight Program Trace Macrocell (PTM) supporting instruction trace

(21) Includes support for floating-point operations.

(22) For a description of the parity error scheme and parity error signals, refer to the Cortex-A9 Technical Reference Manual, available on the Arm website (infocenter.arm.com).

Each Cortex-A9 processor supports the following features:

- Dual-issue superscalar pipeline with advanced branch prediction
- Out-of-order (OoO) dispatch and speculative instruction execution
- 2.5 million instructions per second (MIPS) per MHz, based on the Dhrystone 2.1 benchmark
- 2048-entry Branch Target Address Cache (BTAC)
- 512-entry translation lookaside buffer (TLB)
- 64-entry instruction micro-TLB
- TrustZone security extensions
- Configurable data endianness
- Jazelle* DBX Extensions for byte-code dynamic compiler support
- The Cortex-A9 processor architecture supports the following instruction sets:
  - The Armv7-A performance-optimized instruction set
  - The memory-optimized Thumb*-2 mixed instruction set
    - Improves energy efficiency
    - 31% smaller memory footprint
    - 38% faster than the original Thumb instruction set
  - The Thumb instruction set—supported for legacy applications
- Each processor core in the Intel HPS includes an MMU to support the memory management requirements of common modern operating systems.

The Cortex-A9 processors are designated CPU0 and CPU1.

Related Information

Arm Infocenter
Detailed documentation of Arm Cortex-A9 series processors is available on the Arm Infocenter website.

10.3.3.1. Reset

When a cold or warm reset is issued in the MPU, CPU0 is released from reset automatically. If CPU1 is present, then its reset signals are left asserted when a cold or warm reset is issued. After CPU0 comes out of reset, it can deassert CPU1's reset signals by clearing the CPU1 bit in the MPU Module Reset (`mpumodrst`) register in the Reset Manager.

Related Information

Reset Manager on page 134

10.3.4. Interactive Debugging Features

Each Cortex-A9 processor has built-in debugging capabilities, including six hardware breakpoints (two with Context ID comparison capability) and four watchpoints. The interactive debugging features can be controlled by external JTAG tools or by processor-based monitor code.

Related Information

Arm Infocenter
For more information about the interactive debugging system, refer to the *Debug* chapter of the Cortex-A9 Technical Reference Manual, available on the Arm Infocenter website.

10.3.5. L1 Caches

Cache memory that is closely coupled with an associated processor is called level 1, or L1 cache. Each Cortex-A9 processor has two independent 32 KB L1 caches—one for instructions and one for data—allowing simultaneous instruction fetches and data access. Each L1 cache is four-way set associative, with 32 bytes per line, and supports parity checking.

Note: A parity error cannot be recovered and is indicated by one of the parity error interrupt signals. On a parity error interrupt, you can reset the system or perform further actions depending on the indication of the interrupt signals.

10.3.5.1. Cache Latency

Latency for cache hits and misses varies.

The latency for an L1 cache hit is 1 clock. The latency for an L1 cache miss and L2 cache hit is 6 clocks best case. Latency in the L2 cache can vary depending on other operations in the L2. Parity and ECC settings have no effect on latency. A single-bit ECC error is corrected during the L2 read, but is not re-written to the L2 RAM.

10.3.6. Preload Engine

The preload engine (PLE) is a hardware block that enables the L2 cache to preload selected regions of memory.

The PLE signals the L2 cache when a cache line is needed in the L2 cache, by making the processor data master port start fetching the data. The processor data master does not complete the fetch or return the data to the processor. However, the L2 cache can then proceed to load the cache line. The data is only loaded to the L2 cache, not to the L1 cache or processor registers.

The preload functionality is under software control. The following PLE control parameters must be programmed:

- Programmed parameters, including the following:
  - Base address
  - Length of stride
  - Number of blocks
- A valid bit
- TrustZone memory protection for the cache memory, with an NS (non-secure) state bit
- A translation table base (TTB) address
- An Address Space Identifier (ASID) value

Related Information

Arm Infocenter

For more information about the PLE, refer to the *Preload Engine* chapter of the *Cortex-A9 Technical Reference Manual*, available on the Arm Infocenter website.

10.3.7. Floating Point Unit

Each Arm Cortex-A9 processor includes full support for IEEE-754 floating point operations.

The floating-point unit (FPU) can execute half-, single-, and double-precision variants of the following operations:

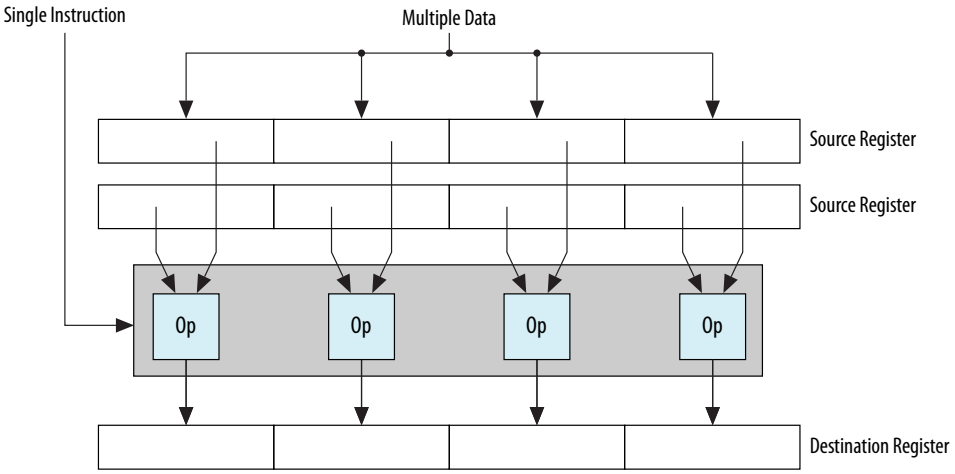
- Add
- Subtract
- Multiply
- Divide
- Multiply and accumulate (MAC)
- Square root

The FPU also converts between floating-point data formats and integers, including special operations to round towards zero required by high-level languages.

10.3.8. NEON Multimedia Processing Engine

The NEON multimedia processing engine (MPE) provides hardware acceleration for media and signal processing applications. Each CPU includes an Arm NEON MPE that supports SIMD processing.

10.3.8.1. Single Instruction, Multiple Data (SIMD) Processing



10.3.8.2. Features of the NEON MPE

The NEON processing engine accelerates multimedia and signal processing algorithms such as video encoding and decoding, 2-D and 3-D graphics, audio and speech processing, image processing, telephony, and sound synthesis.

The Cortex-A9 NEON MPE performs the following types of operations:

- SIMD and scalar single-precision floating-point computations
- Scalar double-precision floating-point computation
- SIMD and scalar half-precision floating-point conversion
- 8-, 16-, 32-, and 64-bit signed and unsigned integer SIMD computation
- 8-bit or 16-bit polynomial computation for single-bit coefficients

The following operations are available:

- Addition and subtraction
- Multiplication with optional accumulation (MAC)
- Maximum- or minimum-value driven lane selection operations
- Inverse square root approximation
- Comprehensive data-structure load instructions, including register-bank-resident table lookup

Related Information

Arm Infocenter

For more information about the Cortex-A9 NEON MPE, refer to the Cortex-A9 NEON Media Processing Engine Technical Reference Manual, which you can download from the Arm Infocenter website.

10.3.9. Memory Management Unit

The MMU is used in conjunction with the L1 and L2 caches to translate virtual addresses used by software to physical addresses used by hardware. Each processor has a private MMU.

10.3.9.1. TLBs Supported By the MMU

TLB Type	Memory Type	Number of Entries	Associativity
Micro TLB	Instruction	64	Fully associative
Micro TLB	Data	64	Fully associative
Main TLB	Instruction and Data	512	Two-way associative

10.3.9.2. TLB Features

The main TLB has the following features:

- Lockable entries using the lock-by-entry model
- Supports hardware page table walks to perform look-ups in the L1 data cache

The MPU address map is divided into the following regions:

- The boot region
- The SDRAM region
- The FPGA slaves region
- The HPS peripherals region

Note: The SMP bit in the ACTLR register must be set before enabling the MMU.

Related Information

- Arm Infocenter
  For more information about the MMU, refer to the *Memory Management Unit* chapter of the *Cortex-A9 Technical Reference Manual*, available on the Arm Infocenter website.
- Cortex-A9 MPCore Technical Reference Manual
  For more further information about the ACTLR `register`, refer to the *Cortex-A9 MPCore Technical Reference Manual*, available on the Arm Infocenter website.

10.3.9.3. The Boot Region

The boot region is 1 MB in size, based at address 0. After power-on, or after reset of the system interconnect, the boot region is occupied by the boot ROM, allowing the Cortex-A9 MPCore to boot. Although the boot region size is 1 MB, accesses beyond 128 KB are illegal because the boot ROM is only 128 KB.

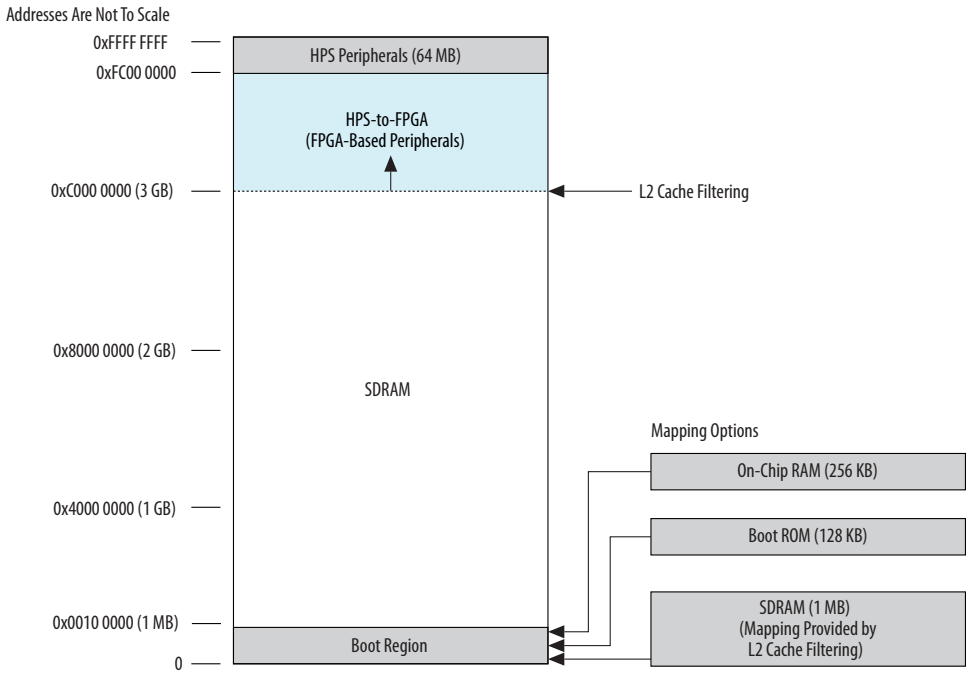
The 1 MB boot region can be subsequently remapped to the bottom 1 MB of SDRAM region by adjusting the L2 cache address filter.

Note: Alternatively, the boot region can be mapped to the 256 KB on-chip RAM.

Related Information

- Address Remapping on page 0
  For more information about address remapping
- System Interconnect on page 200

10.3.9.3.1. Boot ROM Mapping



10.3.9.4. The SDRAM Region

The SDRAM region starts at address `0x100000` (1 MB). The top of the region is determined by the L2 cache filter.

The L2 cache contains a filtering mechanism that routes accesses to the SDRAM L3 interconnect and system interconnect. The filter defines a filter range with start and end addresses. Any access within this filter range is routed to the SDRAM L3 interconnect. Accesses outside of this filter range are routed to the system interconnect.

The start and end addresses are specified in the following register fields:

- `reg12_addr_filtering_start.address_filtering_start`
- `reg12_address_filtering_end.address_filtering_end`

To remap the lower 1MB of SDRAM into the boot region, set the filter start address to `0x0` to ensure accesses between `0x0` and `0xFFFFF` are routed to the SDRAM. Independently, you can set the filter end address in 1 MB increments above `0xC0000000` to extend the upper bounds of the SDRAM region. However, you achieve this extended range at the expense of the FPGA peripheral address span. Depending on the address filter settings in the L2 cache, the top of the SDRAM region can range from `0xBFFFFFFF` to `0xFBFFFFFF`.

10.3.9.5. The FPGA Slaves Region

The Cortex-A9 MPU subsystem supports the variable-sized FPGA slaves region to communicate with FPGA-based peripherals. This region can start as low as `0xC0000000`, depending on the L2 cache filter settings. The top of the FPGA slaves region is located at `0xFBFFFFFF`. As a result, the size of the FPGA slaves region can range from 0 to `0x3C000000` bytes.

10.3.9.6. The HPS Peripherals Region

The HPS peripherals region is the top 64 MB in the address space, starting at `0xFC000000` and extending to `0xFFFFFFFF`. The HPS peripherals region is always allocated to the HPS dedicated peripherals for the Cortex-A9 MPU subsystem.

10.3.10. Performance Monitoring Unit

Each Cortex-A9 processor has a Performance Monitoring Unit (PMU). The PMU supports 58 events to gather statistics on the operation of the processor and memory system. Six counters in the PMU accumulate the events in real time. The PMU counters are accessible either from the processor itself, using the Coprocessor 14 (CP14) interface, or from an external debugger. The events are also supplied to the PTM and can be used for trigger or trace.

Related Information

Arm Infocenter
For more information about the PMU, refer to the *Performance Monitoring Unit* chapter of the *Cortex-A9 Technical Reference Manual*, available on the Arm Infocenter website.

10.3.11. Arm Cortex-A9 MPCore Timers

There is one interval timer and one watchdog timer for each processor.

10.3.11.1. Functional Description

Each timer is private, meaning that only its associated processor can access it. If the watchdog timer is not needed, it can be configured as a second interval timer.

Each private interval and watchdog timer has the following features:

- A 32-bit counter that optionally generates an interrupt when it reaches zero
- Configurable starting values for the counter
- An eight-bit prescaler value to qualify the clock period

10.3.11.2. Implementation Details

The timers are configurable to either single-shot or auto-reload mode. The timer blocks are clocked by `mpu_periph_clk` running at ¼ the rate of the `mpu_clk`.

Related Information

Arm Infocenter
For more information about private timers, refer to "About the private timer and watchdog blocks" in the *Global timer, Private timers, and Watchdog registers*

chapter of the *Cortex-A9 MPCore Technical Reference Manual*, available on the Arm Infocenter website.

10.3.12. Generic Interrupt Controller

10.3.12.1. Functional Description

The PL390 Generic Interrupt Controller (GIC) supports up to 128 interrupt sources, including dedicated peripherals and IP implemented in the FPGA fabric. In a dual-core system, the GIC is shared by both Cortex-A9 processors. Each processor also has 16 banked software-generated interrupts and 16 banked private peripheral interrupts, which occupy GIC interrupt numbers 0 to 31.

10.3.12.2. Implementation Details

The configuration and control for the GIC is memory-mapped and accessed through the SCU. The GIC are clocked by `mpu_periph_clk`, running at ¼ the rate of `mpu_clk`.

Related Information

- GIC Interrupt Map for the Arria 10 SoC HPS on page 294
- Arm Infocenter
  For more information about the PL390 GIC, refer to the *Interrupt Controller* chapter of the *Cortex-A9 MPCore Technical Reference Manual*, available on the Arm Infocenter website.

10.3.12.2.1. GIC Interrupt Map for the Arria 10 SoC HPS

Note: To ensure that you are using the correct GIC interrupt number, your code should refer to the symbolic interrupt name, as shown in the **Interrupt Name** column. Symbolic interrupt names are defined in a header file distributed with the source installation for your operating system.

Table 89. GIC Interrupt Map

GIC Interrupt Number	Source Block	Interrupt Name	Combined Interrupts	Triggering
32	System Manager	`DERR_Global`	This interrupt combines: `ddr_derr`, `dma_derr`, `emac0_tx_derr`, `emac0_rx_derr`, `emac1_tx_derr`, `emac1_rx_derr`, `emac2_tx_derr`, `emac2_rx_derr`, `usb0_derr`, `usb1_derr`, `sdmmc_porta_derr`, `sdmmc_portb_derr`, `nandr_derr`, `nandw_derr`, `nande_derr`,	Level

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Combined Interrupts	Triggering
			qspi_derr, ram_derr, and l2_derr.	
33	System Manager	cpux_parityfail	This interrupt combines: cpu0_parityfail_BTAC, cpu0_parityfail_GHB, cpu0_parityfail_I_Tag, cpu0_parityfail_I_Data, cpu0_parityfail_TLB, cpu0_parityfail_D_Outer, cpu0_parityfail_D_Tag, cpu0_parityfail_D_Data, cpu1_parityfail_BTAC, cpu1_parityfail_GHB, cpu1_parityfail_I_Tag, cpu1_parityfail_I_Data, cpu1_parityfail_TLB, cpu1_parityfail_D_Outer, cpu1_parityfail_D_Tag, cpu1_parityfail_D_Data, scu_parityfail0, and scu_parityfail1.	Level
34	System Manager	SERR_Global	This interrupt combines: ddr_serr, dma_serr, emac0_tx_serr, emac0_rx_serr, emac1_tx_serr, emac1_rx_serr, emac2_tx_serr, emac2_rx_serr, usb0_serr, usb1_serr, sdmmc_porta_serr, sdmmc_portb_serr, nandr_serr, nandw_serr, nande_serr, qspi_serr, ram_serr, and l2_serr.	Level
35	CortexA9_0	cpu0_deflags0	-	Level
36	CortexA9_0	cpu0_deflags1	-	Level
37	CortexA9_0	cpu0_deflags2	-	Level
38	CortexA9_0	cpu0_deflags3	-	Level
39	CortexA9_0	cpu0_deflags4	-	Level
40	CortexA9_0	cpu0_deflags5	-	Level

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Combined Interrupts	Triggering
41	CortexA9_0	cpu0_deflags6	-	Level
42	CortexA9_1	cpu1_deflags0	-	Level
43	CortexA9_1	cpu1_deflags1	-	Level
44	CortexA9_1	cpu1_deflags2	-	Level
45	CortexA9_1	cpu1_deflags3	-	Level
46	CortexA9_1	cpu1_deflags4	-	Level
47	CortexA9_1	cpu1_deflags5	-	Level
48	CortexA9_1	cpu1_deflags6	-	Level
49	SCU	scu_ev_abort	-	Edge-triggered
50	L2-Cache	l2_combined_IRQ	This interrupt combines: DECERRINTR, ECNTRINTR, ERRRDINTR, ERRRTINTR, ERRWDINTR, ERRWTINTR, PARRDINTR, PARRTINTR, and SLVERRINTR.	Level
51	FPGA	F2S_FPGA_IRQ0	-	Level or Edge
52	FPGA	F2S_FPGA_IRQ1	-	Level or Edge
53	FPGA	F2S_FPGA_IRQ2	-	Level or Edge
54	FPGA	F2S_FPGA_IRQ3	-	Level or Edge
55	FPGA	F2S_FPGA_IRQ4	-	Level or Edge
56	FPGA	F2S_FPGA_IRQ5	-	Level or Edge
57	FPGA	F2S_FPGA_IRQ6	-	Level or Edge
58	FPGA	F2S_FPGA_IRQ7	-	Level or Edge
59	FPGA	F2S_FPGA_IRQ8	-	Level or Edge
60	FPGA	F2S_FPGA_IRQ9	-	Level or Edge
61	FPGA	F2S_FPGA_IRQ10	-	Level or Edge
62	FPGA	F2S_FPGA_IRQ11	-	Level or Edge
63	FPGA	F2S_FPGA_IRQ12	-	Level or Edge
64	FPGA	F2S_FPGA_IRQ13	-	Level or Edge
65	FPGA	F2S_FPGA_IRQ14	-	Level or Edge
66	FPGA	F2S_FPGA_IRQ15	-	Level or Edge
67	FPGA	F2S_FPGA_IRQ16	-	Level or Edge
68	FPGA	F2S_FPGA_IRQ17	-	Level or Edge

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Combined Interrupts	Triggering
69	FPGA	F2S_FPGA_IRQ18	-	Level or Edge
70	FPGA	F2S_FPGA_IRQ19	-	Level or Edge
71	FPGA	F2S_FPGA_IRQ20	-	Level or Edge
72	FPGA	F2S_FPGA_IRQ21	-	Level or Edge
73	FPGA	F2S_FPGA_IRQ22	-	Level or Edge
74	FPGA	F2S_FPGA_IRQ23	-	Level or Edge
75	FPGA	F2S_FPGA_IRQ24	-	Level or Edge
76	FPGA	F2S_FPGA_IRQ25	-	Level or Edge
77	FPGA	F2S_FPGA_IRQ26	-	Level or Edge
78	FPGA	F2S_FPGA_IRQ27	-	Level or Edge
79	FPGA	F2S_FPGA_IRQ28	-	Level or Edge
80	FPGA	F2S_FPGA_IRQ29	-	Level or Edge
81	FPGA	F2S_FPGA_IRQ30	-	Level or Edge
82	FPGA	F2S_FPGA_IRQ31	-	Level or Edge
83	FPGA	F2S_FPGA_IRQ32	-	Level or Edge
84	FPGA	F2S_FPGA_IRQ33	-	Level or Edge
85	FPGA	F2S_FPGA_IRQ34	-	Level or Edge
86	FPGA	F2S_FPGA_IRQ35	-	Level or Edge
87	FPGA	F2S_FPGA_IRQ36	-	Level or Edge
88	FPGA	F2S_FPGA_IRQ37	-	Level or Edge
89	FPGA	F2S_FPGA_IRQ38	-	Level or Edge
90	FPGA	F2S_FPGA_IRQ39	-	Level or Edge
91	FPGA	F2S_FPGA_IRQ40	-	Level or Edge
92	FPGA	F2S_FPGA_IRQ41	-	Level or Edge
93	FPGA	F2S_FPGA_IRQ42	-	Level or Edge
94	FPGA	F2S_FPGA_IRQ43	-	Level or Edge
95	FPGA	F2S_FPGA_IRQ44	-	Level or Edge
96	FPGA	F2S_FPGA_IRQ45	-	Level or Edge
97	FPGA	F2S_FPGA_IRQ46	-	Level or Edge
98	FPGA	F2S_FPGA_IRQ47	-	Level or Edge
99	FPGA	F2S_FPGA_IRQ48	-	Level or Edge
100	FPGA	F2S_FPGA_IRQ49	-	Level or Edge
101	FPGA	F2S_FPGA_IRQ50	-	Level or Edge

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Combined Interrupts	Triggering
102	FPGA	F2S_FPGA_IRQ51	-	Level or Edge
103	FPGA	F2S_FPGA_IRQ52	-	Level or Edge
104	FPGA	F2S_FPGA_IRQ53	-	Level or Edge
105	FPGA	F2S_FPGA_IRQ54	-	Level or Edge
106	FPGA	F2S_FPGA_IRQ55	-	Level or Edge
107	FPGA	F2S_FPGA_IRQ56	-	Level or Edge
108	FPGA	F2S_FPGA_IRQ57	-	Level or Edge
109	FPGA	F2S_FPGA_IRQ58	-	Level or Edge
110	FPGA	F2S_FPGA_IRQ59	-	Level or Edge
111	FPGA	F2S_FPGA_IRQ60	-	Level or Edge
112	FPGA	F2S_FPGA_IRQ61	-	Level or Edge
113	FPGA	F2S_FPGA_IRQ62	-	Level or Edge
114	FPGA	F2S_FPGA_IRQ63	-	Level or Edge
115	DMA	dma_IRQ0	-	Level
116	DMA	dma_IRQ1	-	Level
117	DMA	dma_IRQ2	-	Level
118	DMA	dma_IRQ3	-	Level
119	DMA	dma_IRQ4	-	Level
120	DMA	dma_IRQ5	-	Level
121	DMA	dma_IRQ6	-	Level
122	DMA	dma_IRQ7	-	Level
123	DMA	dma_irq_abort	-	Level
124	EMAC0	emac0_IRQ	This interrupt combines: sbd_intr_o and lpi_intr_o.	Level
125	EMAC1	emac1_IRQ	This interrupt combines: sbd_intr_o and lpi_intr_o.	Level
126	EMAC2	emac2_IRQ	This interrupt combines: sbd_intr_o and lpi_intr_o.	Level
127	USB0	usb0_IRQ	-	Level
128	USB1	usb1_IRQ	-	Level
129	SDRAM scheduler	HMC_error	-	Level
130	SDMMC	sdmmc_IRQ	-	Level
131	NAND	nand_IRQ	-	Level
132	QSPI	qspi_IRQ	-	Level

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Combined Interrupts	Triggering
133	SPI0 master	spim0_IRQ	This interrupt combines: ssi_txe_intr, ssi_txo_intr, ssi_rxf_intr, ssi_rxo_intr, ssi_rxu_intr, and ssi_mst_intr.	Level
134	SPI1 master	spim1_IRQ	This interrupt combines: ssi_txe_intr, ssi_txo_intr, ssi_rxf_intr, ssi_rxo_intr, ssi_rxu_intr, and ssi_mst_intr.	Level
135	SPI0 slave	spis0_IRQ	This interrupt combines: ssi_txe_intr, ssi_txo_intr, ssi_rxf_intr, ssi_rxo_intr, ssi_rxu_intr, and ssi_mst_intr.	Level
136	SPI1 slave	spis1_IRQ	This interrupt combines: ssi_txe_intr, ssi_txo_intr, ssi_rxf_intr, ssi_rxo_intr, ssi_rxu_intr, and ssi_mst_intr.	Level
137	I2C0	i2c0_IRQ	This interrupt combines: ic_rx_under_intr, ic_rx_full_intr, ic_tx_over_intr, ic_tx_empty_intr, ic_rd_req_intr, ic_tx_abrt_intr, ic_rx_done_intr, ic_activity_intr, ic_stop_det_intr, ic_start_det_intr, and ic_gen_call_intr.	Level
138	I2C1	i2c1_IRQ	This interrupt combines: ic_rx_under_intr, ic_rx_full_intr, ic_tx_over_intr, ic_tx_empty_intr, ic_rd_req_intr, ic_tx_abrt_intr, ic_rx_done_intr, ic_activity_intr, ic_stop_det_intr, ic_start_det_intr, and ic_gen_call_intr.	Level

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Combined Interrupts	Triggering
139	I2C2 (can be used with EMAC0)	i2c2_IRQ	This interrupt combines: ic_rx_under_intr, ic_rx_full_intr, ic_tx_over_intr, ic_tx_empty_intr, ic_rd_req_intr, ic_tx_abrt_intr, ic_rx_done_intr, ic_activity_intr, ic_stop_det_intr, ic_start_det_intr, and ic_gen_call_intr.	Level
140	I2C3 can be used with EMAC1)	i2c3_IRQ	This interrupt combines: ic_rx_under_intr, ic_rx_full_intr, ic_tx_over_intr, ic_tx_empty_intr, ic_rd_req_intr, ic_tx_abrt_intr, ic_rx_done_intr, ic_activity_intr, ic_stop_det_intr, ic_start_det_intr, and ic_gen_call_intr.	Level
141	I2C4 (can be used with EMAC2)	i2c4_IRQ	This interrupt combines: ic_rx_under_intr, ic_rx_full_intr, ic_tx_over_intr, ic_tx_empty_intr, ic_rd_req_intr, ic_tx_abrt_intr, ic_rx_done_intr, ic_activity_intr, ic_stop_det_intr, ic_start_det_intr, and ic_gen_call_intr.	Level
142	UART0	uart0_IRQ	-	Level
143	UART1	uart1_IRQ	-	Level
144	GPIO0	gpio0_IRQ	-	Level
145	GPIO1	gpio1_IRQ	-	Level
146	GPIO2	gpio2_IRQ	-	Level
147	Timer0	timer_l4sp_0_IRQ	This interrupt combines: TIMINT1 and TIMINT2.	Level
148	Timer1	timer_l4sp_1_IRQ	This interrupt combines: TIMINT1 and TIMINT2.	Level
149	Timer2	timer_osc1_0_IRQ	This interrupt combines: TIMINT1 and TIMINT2.	Level

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Combined Interrupts	Triggering
150	Timer3	timer_osc1_1_IRQ	This interrupt combines: TIMINT1 and TIMINT2.	Level
151	Watchdog0	wdog0_IRQ	-	Level
152	Watchdog1	wdog1_IRQ	-	Level
153	Clock Manager	clkmgr_IRQ	-	Level
154	Reset Manager	restmgr_IRQ	-	Level
155	FPGA Manager	fpga_man_IRQ	-	Level
156	CoreSight	nCTIIRQ[0]	-	Level
157	CoreSight	nCTIIRQ[1]	-	Level
158	Security Manager	SEC_MGR_INTR	-	Level
159	L2-Cache	DATABWERR	-	Edge-triggered

Related Information

Implementation Details on page 294

10.3.13. Global Timer

The MPU features a global 64-bit, auto-incrementing timer, which is primarily used by the operating system.

10.3.13.1. Functional Description

The global timer is accessible by the processors using memory-mapped access through the SCU. The global timer has the following features:

- 64-bit incrementing counter with an auto-incrementing feature. It continues incrementing after sending interrupts.
- Memory-mapped in the private memory region.
- Accessed at reset in Secure State only. It can only be set once, but secure code can read it at any time.
- Accessible to both Cortex-A9 processors in the MPCore.
- Clocked by mpu_periph_clk, running at ¼ the rate of mpu_clk.

10.3.13.2. Implementation Details

Each Cortex-A9 processor has a private 64-bit comparator that generates a private interrupt when the counter reaches the specified value. Each Cortex-A9 processor uses the banked ID, ID27, for this interrupt. ID27 is sent to the GIC as a Private Peripheral Interrupt (PPI).

The global timer is clocked by mpu_periph_clk, running at ¼ the rate of mpu_clk.

Related Information

Arm Infocenter

For more information about the global timer, refer to "About the Global Timer" in the *Global timer, Private timers, and Watchdog registers* chapter of the Cortex-A9 MPCore Technical Reference Manual, available on the Arm Infocenter website.

10.3.14. Snoop Control Unit

The SCU manages data traffic for the Cortex-A9 processors and the memory system, including the L2 cache. In a multi-master system, the processors and other masters can operate on shared data. The SCU ensures that each processor operates on the most up-to-date copy of data, maintaining cache coherency.

10.3.14.1. Functional Description

The SCU is used to connect the Cortex-A9 processors and the ACP to the L2 cache controller. The SCU performs the following functions:

- When the processors are set to SMP mode, the SCU maintains data cache coherency between the processors.

  Note: The SCU does not maintain coherency of the instruction caches.
- Initiates L2 cache memory accesses
- Arbitrates between processors requesting L2 access
- Manages ACP access with cache coherency capabilities.

Related Information

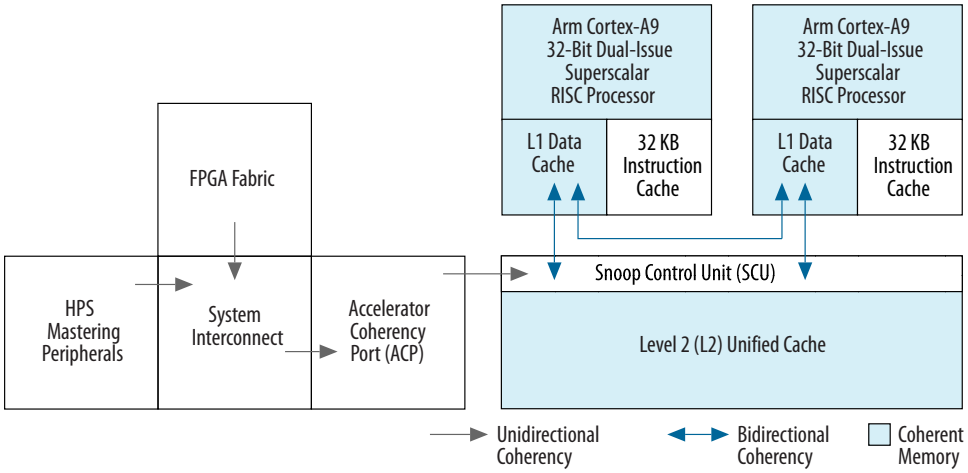
Arm Infocenter

For more information about the SCU, refer to the *Snoop Control Unit* chapter of the *Cortex-A9 MPCore Technical Reference Manual*, available on the Arm Infocenter website.

10.3.14.1.1. Coherent Memory, Snoop Control Unit, and Accelerator Coherency Port

Figure 42. Data Flow Between L1 Caches and SCU

This diagram illustrates the flow of data among the L1 data caches and the SCU.



Related Information

Accelerator Coherency Port on page 303

10.3.14.2. Implementation Details

When the processor writes to any coherent memory location, the SCU ensures that the relevant data is coherent (updated, tagged or invalidated). Similarly, the SCU monitors read operations from a coherent memory location. If the required data is already stored within the other processor's L1 cache, the data is returned directly to the requesting processor. If the data is not in L1 cache, the SCU issues a read to the L2 cache. If the data is not in the L2 cache memory, the read is finally forwarded to main memory. The primary goal is to maximize overall memory performance and minimize power consumption.

The SCU maintains bidirectional coherency between the L1 data caches belonging to the processors. When one processor performs a cacheable write, if the same location is cached in the other L1 cache, the SCU updates it.

Non-coherent data passes through as a standard read or write operation.

The SCU also arbitrates between the Cortex-A9 processors if both attempt simultaneous access to the L2 cache, and manages accesses from the ACP.

In rare circumstances, when the aggregate memory throughput requested by two cores exceeds the memory subsystem capacity and the Acceleration Coherency Port (ACP) is not being used, you may observe that the SCU master arbitration fairness is reduced. This reduction occurs because unused ACP arbitration shares are reassigned to CPU0, resulting in CPU0 gaining twice the memory bandwidth of CPU1. If your application requires a balanced throughput between CPU0 and CPU1, you must design the code that runs on CPU0 so that it prevents CPU0 from using more than 50% of the available memory bandwidth.

10.3.15. Accelerator Coherency Port

The ACP allows master peripherals—including FPGA-based master peripherals—to maintain data coherency with the Cortex-A9 MPCore processors and the SCU. Dedicated master peripherals in the HPS, and those built in FPGA logic, access the coherent memory through the ACP. Cacheable master accesses from the system interconnect are redirected to the ACP.

The ACP port allows one-way coherency. One-way coherency allows an external ACP master to see the coherent memory of the Cortex-A9 processors but does not allow the Cortex-A9 processors to see memory changes outside of the cache.

A master on the ACP port can read coherent memory directly from the L1 and L2 caches, but cannot write directly to the L1 cache. The possible ACP master read and write scenarios are as follows:

- ACP master read with coherent data in the L1 cache: The ACP gets data directly from the L1 cache and the read is interleaved with a processor access to the L1 cache.
- ACP master read with coherent data in L2 cache: The ACP request is queued in the SCU and the transaction is sent to the L2 cache.
- ACP master read with coherent data not in L1 or L2 cache: Depending on the previous state of the data, the L1 or L2 cache may request the data from the L3 system interconnect. The ACP is stalled until data is available, however ACP support of multiple outstanding transactions minimizes bottlenecks.
- ACP master write with coherent data in the L1 cache: The L1 cache data is marked as invalid in the Cortex-A9 MPU and the line is evicted from the L1 cache and sent to L2 memory. The ACP write data is scheduled in the SCU and is eventually written to the L2 cache. Later, when the Cortex-A9 processor accesses the same memory location, a cache miss in the L1 occurs.
- ACP master write, with coherent data in the L2 cache: The ACP write is scheduled in the SCU and then is written into the L2 cache.
- ACP master write, with coherent data not in L1 or L2 cache: ACP write is scheduled.

An ACP transaction is generated when a master on the system interconnect has its AxCACHE[1] attribute signal set to 1, which indicates a coherent request. All transactions that are set as cacheable are routed to the ACP instead of the normal mapping and are treated as coherent by the cache controllers of the MPU subsystem. The ACP ID generation follows an allocation mechanism that ensures that requests with the same master initiator flow and sequence identifiers always allocate the same local sequence ID and that requests with different identifiers always have different IDs. In addition, if an allocator runs out of sequence IDs, the ACP stalls requests until the reception of responses makes new sequence IDs available. Therefore, the guaranteed number of pending transactions that the interconnect can have is up to four pending transactions, with the allowed AXI ID[2:0] values of 0x4 through 0x7. AxID[2:0] values of 0x0 and 0x1 are assigned to CPU0 and CPU1, respectively.

Note: The entire 4 GB address space can be accessed coherently through the ACP.

Note: Avoid needing to coherently access back the HPS through ACP from the fabric in order to complete an access coming from HPS, as this may result in a deadlock situation.

Certain coherent access scenarios can create deadlock through the ACP and the CPU. However, you can avoid this type of deadlock with a simple access strategy.

In the following example, the CPU creates deadlock by initiating an access to the HPS through the FPGA fabric:

1. The CPU initiates a device memory access to the FPGA fabric. The CPU pipeline must stall until this type of access is complete.
2. Before the FPGA fabric state machine can respond to the device memory access, it must access the HPS coherently. It initiates a coherent access, which requires the ACP.
3. The ACP must perform a cache maintenance operation before it can complete the access. However, the CPU's pipeline stall prevents it from performing the cache maintenance operation. The system deadlocks.

You can implement the desired access without deadlock, by breaking it into smaller pieces. For example, you can initiate the operation with one access, then determine the operation status with a second access.

Note: Refer to the Arria 10 SoC device errata for details on Arm errata that directly affect the ACP.

Related Information

- Coherent Memory, Snoop Control Unit, and Accelerator Coherency Port on page 302
- Arria 10 SX Device Errata

10.3.15.1. AxUSER and AxCACHE Attributes

ARUSER[0] and AWUSER[0] Bits

The ACP distinguishes between shared and non-shared requests with the ARUSER[0] and AWUSER[0] signals. The other AxUSER signals, AxUSER[4:1] are not interpreted by the SCU and are simply forwarded to the L2 cache. They are typically set to AxUSER[4:1] = b'1111.The AxCACHE[1] attribute signals must also be set to allow L1 caches to be snooped. Sideband signals ARUSER[4:1] and AWUSER[4:1] convey the inner cache attributes.

The table below shows how the AxUSER[0] bit determines whether a request is shared or non-shared. ARUSER[0] applies to read transactions, and AWUSER[0] applies to write transactions.

Table 90. ARUSER[0] or AWUSER[0] Sideband Signal Information

ARUSER[0] or AWUSER[0]	Sideband Signal Information
1	Shared request
0	Non-shared request

Other AxUSER bits from the ACP are not interpreted by the SCU but are forwarded to the L2 cache.

ARCACHE[4:0] and AWCACHE[4:0] Bits

The Cortex-A9 MPU only interprets ARCACHE[1] and AWCACHE[1] from ACP requests. All other attributes are forwarded to the L2 cache. AxCACHE[1] is used in combination with AxUSER[0] to detect a coherent access. If AxCACHE[1]=0x1, (normal memory) and AxUSER[0]=0x1, then the access is considered coherent. All ACP requests with AxCACHE[1]=0x0 or AxUSER[0]=0x0 are seen as non-coherent requests.

Table 91. ARUSER[0] or AWUSER[0] Sideband Signal Information

ARCACHE[1] or AWCACHE[1]	ARUSER[0] or AWUSER[0]	Access Type
1	1	Coherent request
0	0	Non-coherent request

10.3.15.2. AxPROT Attributes

AxPROT specifies the secure state of the transaction, and must match the security state of the target memory to ensure a cache hit; otherwise, it is a cache miss. ARPROT applies to read transactions, and AWPROT applies to write transactions.

The Cortex-A9 MPU interprets ARPROT[1] and AWPROT[1] from ACP requests.

AxPROT[1] is used to protect against illegal transactions and to specify which address space is being accessed. This is important because in a Secure state, software can access both physical address spaces. The security of the access is not necessarily the same as the Security state of the processor that generated that access.

All ACP requests with AxPROT[0] identifies an access as unprivileged or privileged. When AxPROT[0] is set to 0, then access is 'Unprivileged'. When AxPROT[0] is set to 1, then access is 'Privileged'.

All ACP requests with AxPROT[2] indicates whether the transaction is a data access request or an instruction access. When AxPROT[2] is set to 0, then the request is a 'Data Access'. When AxPROT[2] is set to 1, then access is 'Instruction Access'.

The table below shows how the AxPROT[1] bit determines whether the access is secured or non-secured. When AxPROT[1] is set to 0, then the access is Secure. When AxPROT[1] is set to 1, then the access is Non-secure.

Table 92. Protection Encoding

AxPROT	Value	Function
[0]	0	Unprivileged access
	1	Privileged access
[1]	0	Secure access
	1	Non-secure access
[2]	0	Data access
	1	Instruction access

10.3.15.3. Cache Coherency for ACP Shared Requests

When a shared access is received on the ACP, caches are checked for coherency of the requested address. A shared access occurs when two masters access the same memory space.

This coherency check is performed by the SCU. If a cache hit occurs during a write access, the affected cache lines are cleaned and invalidated. This event may cause an eviction from the L1 cache to be sent to L2 memory if the L1 cache line is dirty. Once the invalidation and possible eviction is completed, the ACP write request is written to L2 memory. If an eviction was executed from L1 cache, then two consecutive writes to L2 memory occur over the AXI bus: the write from the eviction and the write from the ACP.

For a cache hit during a read access, the L1 cache line is provided by the CPU and is returned to the ACP.

For a cache miss during a write access, the invalidation is considered as complete and the ACP request is sent to L2 memory.

For a cache miss during a read access, the request is forwarded to L2 memory, which returns the data directly to the ACP.

Note: Shared write requests are always transferred to the L2 memory once the cache line is potentially clean and invalidated in the L1 cache memory. Reads are observed on the L2 only if they miss in the L1 cache.

10.3.15.4. AXI Master Configuration for ACP Access

To use the ACP for coherent accesses, the following configurations apply:

ACP master configurations must be as follows:

- For coherent ACP read accesses, the AXI bits must be programmed as follows to avoid compromising coherency:
  - AxCACHE[3:0] attributes must match the properties defined in the Cortex-A9 MPCore MMU page tables for the relevant memory region.
  - Shareable attribute AxUSER[0] must be set to 0x1

The Cortex-A9 MPCore configuration for ACP use should be as follows:

- The Snoop Control Unit must be enabled (by setting the SCU enable bit in the SCU Control Register at 0xFFFFC000).
- Coherent memory must be marked cacheable and shareable.
- The SMP bit of the ACTLR register must be set in the Cortex-A9 processor that shares data over the ACP.

Note: To achieve maximum performance on the ACP, avoid switching from shared to non-shared requests and vice-versa. When a shared request is latched in the ACP and there are non-shared requests still pending, the non-shared requests must be completed before the shared request can proceed.

The following sections detail the attribute configurations necessary to support coherency.

Related Information

Cortex-A9 MPCore Technical Reference Manual
For more information about the use of the L2 Cache Controller address space, refer to the *Cortex-A9 MPCore Technical Reference Manual*, available on the Arm Infocenter website.

10.3.15.4.1. Configuring AxCACHE[3:0] Sideband Signals for Coherent Accesses

The following list highlights how to correctly derive and apply the correct AxCACHE settings for coherent accesses.

- The correct AxCACHE[3:0] setting is dependent on the MMU page table settings. However, for coherent accesses, AxCACHE[1] must be set to 0x1.
- For HPS masters, AxCACHE[3:0] is static and applied to the relevant master port through the System Manager.
- For FPGA masters, AxCACHE[3:0] is applied in the FPGA fabric and can be set for each access.

Related Information

- System Manager on page 159
  For more information about programming AxCACHE[3:0] through the System Manager
- Cortex-A9 MPCore Technical Reference Manual
  For more information about the Accelerator Coherency Port.

10.3.15.4.2. Configuring AxUSER[4:0] Sideband Signals

The following list highlights how to correctly derive and apply the correct AxUSER settings for coherent accesses.

- AxUSER[0] (shared attribute) must be set to 0x1 for coherent accesses.
- Because the ACP has no inner cache policy, AxUSER[3:1] is not interpreted by the SCU. AxUSER[3:1] attributes pass to the L2 cache controller and are used when the cache is operating in exclusive mode.
- For FPGA masters, AxUSER[4:0] is applied in the FPGA fabric and can be set for each access.

Related Information

Cortex-A9 MPCore Technical Reference Manual
For more information about the Accelerator Coherency Port.

10.3.15.4.3. Configuring AxPROT[2:0] Sideband Signals for Coherent Accesses

The following list highlights how to correctly derive and apply the correct AxPROT settings for coherent accesses.

- AxPROT[1] must be set to 0x0 for coherent accesses.
- For FPGA masters, AxPROT[2:0] is applied in the FPGA fabric and can be set for each access.

10.3.15.5. Burst Sizes and Byte Strobes

The ACP improves system performance for hardware accelerators in the FPGA fabric. However, in order to achieve high levels of performance, you must use the one of the optimized burst types. Other burst configurations have significantly lower performance.

10.3.15.5.1. Recommended Burst Types

Table 93. Recommended Burst Types for Optimized Bursts

Burst Type	Beats	Width (Bits)	Address Type	Byte Strobes
Wrapping	4	64	64-bit aligned	Asserted
Incrementing	4	64	32-bit aligned	Asserted

Note: If the slave port of the FPGA-to-HPS bridge is not 64 bits wide, you must supply bursts to the FPGA-to-HPS bridge that are upsized or downsized to the burst types above. For example, if the slave data width of the FPGA-to-HPS bridge is 32 bits, then bursts of eight beats by 32 bits are required to access the ACP efficiently.

Note: If the address and burst size of the transaction to the ACP matches either of the conditions shown in the table "Recommended Burst Types for Optimized Bursts", the logic in the MPU assumes the transaction has all its byte strobes set. If the byte strobes are not all set, then the write does not actually overwrite all the bytes in the word. Instead, the cache assumes the whole cache line is valid. If this line is dirty (and therefore gets written out to SDRAM), data corruption might occur.

In addition to optimizing performance, using a 64-bit access width will allow you to use ECC. ECC is only supported for 64-bit accesses that are 64-bit aligned..

Related Information

Single Event Upset Protection on page 313

10.3.15.6. Exclusive and Locked Accesses

The ACP does not support exclusive accesses to coherent memory. To ensure mutually exclusive access to shared data, use the exclusive access support built into the SDRAM L3 interconnect scheduler. The AXI buses that interface to the scheduler provide `ARLOCK[0]` and `AWLOCK[0]` signals which are used by the scheduler to arbitrate for exclusive access to a memory location. The SDRAM L3 scheduler contains one monitor for each of the following exclusive-capable masters:

- CPU0
- CPU1
- FPGA-to-HPS
- FPGA-to-SDRAM0
- FPGA-to-SDRAM1
- FPGA-to-SDRAM2

Each monitor only tracks one exclusive address.

Related Information

System Interconnect on page 200
To learn more about Exclusive Monitors in the System Interconnect, refer to the System Interconnect Chapter.

10.3.15.7. Avoiding ACP Dependency Lockup

Certain coherent access scenarios can create deadlock through the ACP and the CPU. However, you can avoid this type of deadlock with a simple access strategy.

In the following example, the CPU creates deadlock by initiating an access to the HPS through the FPGA fabric:

1. The CPU initiates a device memory access to the FPGA fabric. The CPU pipeline must stall until this type of access is complete.
2. Before the FPGA fabric state machine can respond to the device memory access, it must access the HPS coherently. It initiates a coherent access, which requires the ACP.
3. The ACP must perform a cache maintenance operation before it can complete the access. However, the CPU's pipeline stall prevents it from performing the cache maintenance operation. The system deadlocks.

You can implement the desired access without deadlock, by breaking it into smaller pieces. For example, you can initiate the operation with one access, then determine the operation status with a second access.

10.4. L2 Cache

The MPU subsystem includes a secondary 512 KB L2 shared, unified cache memory.

10.4.1. Functional Description

The L2 cache is much larger than the L1 cache. The L2 cache has significantly lower latency than external memory. The L2 cache is up to eight-way associative, configurable down to one-way (direct mapped). Like the L1 cache, the L2 cache can be locked by cache line, locked by way, or locked by master (CPU and ACP masters).

The L2 cache implements error correction codes (ECCs) and ECC error reporting. The cache can report a number of events to the processor and operating system.

Related Information

Cortex-A9 MPU Subsystem Register Implementation on page 0
For more information regarding the L2 Cache Controller Address Map, refer to the *Cortex-A9 MPU Subsystem Register Implementation* section.

10.4.1.1. Cache Controller Configuration

The L2 cache consists of the Arm L2C-310 L2 cache controller configured as follows:

- 512 KB total memory
- Eight-way associativity
- Physically addressed, physically tagged
- Line length of 32 bytes

- Critical first word linefills
- Support for all AXI cache modes
  - Write-through
  - Write-back
  - Read allocate
  - Write allocate
  - Read and write allocate
- Single event upset (SEU) protection
  - Parity on Tag RAM
  - ECC on L2 Data RAM
- Two slave ports mastered by the SCU
- Two master ports connected to the following slave ports:
  - SDRAM L3 interconnect, 64-bit slave port width
  - L3 interconnect, 64-bit slave port width
- Cache lockdown capabilities as follows:
  - Line lockdown
  - Lockdown by way
  - Lockdown by master (both processors and ACP masters)
- TrustZone support
- Cache event monitoring

Related Information

- L2 Cache Event Monitoring on page 315
- System Manager on page 159
  For more information about SEU errors, refer to the *System Manager* chapter.
- Arm Infocenter
  For more information about AXI user sideband signals, refer to the CoreLink Level 2 Cache Controller L2C-310 Technical Reference Manual, which you can download from the Arm Infocenter website.

10.4.1.1.1. Implementation Details

The following table shows the parameter settings for the cache controller.

Table 94. Cache Controller Configuration

Feature	Meaning
Cache way size	64 KB
Number of cache ways	8 ways
Tag RAM write latency	1
Tag RAM read latency	1
Tag RAM setup latency	1

continued...

Feature	Meaning
Data RAM write latency	1
Data RAM read latency	2
Data RAM setup latency	1
Parity logic	Parity logic enabled
Lockdown by master	Lockdown by master enabled
Lockdown by line	Lockdown by line enabled
AXI ID width on slave ports	6 AXI ID bits on slave ports
Address filtering	Address filtering logic enabled
Speculative read	Logic for supporting speculative read enabled
Presence of ARUSERMx and AWUSERMx sideband signals	Sideband signals enabled

Related Information

Arm Infocenter

For further information about cache controller configurable options, refer to the CoreLink Level 2 Cache Controller L2C-310 Technical Reference Manual, available on the Arm Infocenter website.

L2 Cache Event Monitoring

The L2 cache supports the built-in cache event monitoring signals shown in the table below. The L2 cache can count two of the events at any one time.

Table 95. L2 Cache Events

Event	Description
CO	Eviction (cast out) of a line from the L2 cache.
DRHIT	Data read hit in the L2 cache.
DRREQ	Data read lookup to the L2 cache. Subsequently results in a hit or miss.
DWHIT	Data write hit in the L2 cache.
DWREQ	Data write lookup to the L2 cache. Subsequently results in a hit or miss.
DWTREQ	Data write lookup to the L2 cache with write-through attribute. Subsequently results in a hit or miss.
EPFALLOC	Prefetch hint allocated into the L2 cache.
EPFHIT	Prefetch hint hits in the L2 cache.
EPFRCVDS0	Prefetch hint received by slave port S0.
EPFRCVDS1	Prefetch hint received by slave port S1.
IPFALLOC	Allocation of a prefetch generated by L2 cache controller into the L2 cache.
IRHIT	Instruction read hit in the L2 cache.
IRREQ	Instruction read lookup to the L2 cache. Subsequently results in a hit or miss.
SPNIDEN	Secure privileged non-invasive debug enable.
SRCONFS0	Speculative read confirmed in slave port S0.

continued...

Event	Description
SRCONFS1	Speculative read confirmed in slave port S1.
SRRCVDS0	Speculative read received by slave port S0.
SRRCVDS1	Speculative read received by slave port S1.
WA	Allocation into the L2 cache caused by a write (with write-allocate attribute) miss.

In addition, the L2 cache events can be captured and timestamped using dedicated debugging circuitry.

10.4.1.2. Single Event Upset Protection

The L2 cache has the option of using ECCs to protect against Single Event Upset (SEU) errors in the cache RAM.

The L2 cache has two types of protection:

- The L2 cache tag RAMs are protected by parity.
- The L2 cache data RAMs are protected using single error correction, double error detection (SECDED) ECC Hamming code.

Enabling ECCs does not affect the performance of the L2 cache. The ECC bits are calculated only for writes to the data RAM that are 64 bits wide (8 bytes, or one-quarter of the cache line length). The ECC logic does not perform a read-modify-write when calculating the ECC bits.

The ECC protection bits are not valid in the following cases:

- Data is written that is not 64-bit aligned in memory
- Data is written that is less than 64 bits in width

In these cases the Byte Write Error interrupt is asserted. Cache data is still written when such an error occurs. However, the ECC error detection and correction continues to function. Therefore, the cache data is likely to be incorrect on subsequent reads.

To use ECCs, the software and system must meet the following requirements:

- L1 and L2 cache must be configured as write-back and write-allocate for any cacheable memory region
- Level 3 interconnect masters using the ACP must only perform the following types of data writes:
  - 64-bit aligned in memory
  - 64-bit wide accesses

  Note that system interconnect masters can include masters in the FPGA accessing the FPGA-to-HPS bridge.

If a correctable ECC error occurs, the ECC corrects the read data in parallel to asserting a correctable error signal on the AXI bus. If an uncorrectable error occurs in the L2 cache, the uncorrected data remains in the L2 cache and an AXI slave error (SLVERR) is sent to the L1 memory system. In addition, interrupts can be enabled for correctable and uncorrectable ECC errors through the GIC.

Related Information

System Manager on page 159
For more information about SEU errors, refer to the *System Manager* chapter.

10.4.1.3. L2 Cache Parity

Because the L2 data RAM is ECC-protected, parity checking on the data RAM is not required. However, because the tag RAM is not ECC protected, it requires parity checking. Because parity cannot be configured independently, parity checking for both the data RAM and tag RAM must be enabled.

Ideally, parity should be enabled before the L2 cache is enabled. If the cache is already enabled, you must clean the cache and disable it before parity is enabled. After enabling parity, the cache is invalidated and enabled. To enable parity:

1. Set the `Parity_on` bit in the L2 `Auxiliary Control register`.
2. Invalidate the L2 cache through the `Cache Maintenance Operations` registers.
3. Enable the cache through the `L2_Cache_enable bit` of the L2 `Control register`.

Note: If you would like the parity errors to be reported you must enable the parity interrupts in the `L2 Interrupt Mask` register along with the corresponding interrupts in the general interrupt controller (GIC).

Refer to the Arm Infocenter website for more information regarding L2 cache registers and programming.

Parity Error Handling

If a parity error occurs in the tag or data RAM during AXI read transactions, a SLVERR response is reported through the event bus and interrupt signals. If a parity error occurs on tag or data RAM during AXI write transactions, a SLVERR response is reported back through a SLVERRINTR interrupt signal. For cache maintenance operations, if a parity error occurs on tag or data RAM, the error is reported back through the interrupt lines only. In addition, the L2 Cache cannot refill a line from external memory due to a parity error.

Related Information

Arm Infocenter

10.4.1.4. L2 Cache Lockdown Capabilities

The L2 cache has three methods to lock data in the cache RAMs:

- Lockdown by line—Used to lock lines in the cache. This is commonly used for loading critical sections of software into the cache temporarily.
- Lockdown by way—Allows any or all of the eight cache ways to be locked. This is commonly used for loading critical data or code into the cache.
- Lockdown by master-- allows cache ways to be dedicated to a single master port. This allows a large cache to look like smaller caches to multiple master ports. The L2 cache can be mastered by any of the L3 masters (including the FPGA fabric). When using lockdown by master, AXI ID0 and 1 represent CPU0 and CPU1 respectively and AXI IDs 0x4 through 0x7 can be used for the L3 interconnect masters.

Related Information

Arm Infocenter

For more information about L2 cache lockdown capabilities, refer to "Cache operation" in the *Functional Overview* chapter of the *CoreLink Level 2 Cache Controller L2C-310 Technical Reference Manual*, available on the Arm Infocenter website.

10.4.1.5. L2 Cache Event Monitoring

The L2 cache supports the built-in cache event monitoring signals shown in the table below. The L2 cache can count two of the events at any one time.

Table 96. L2 Cache Events

Event	Description
CO	Eviction (cast out) of a line from the L2 cache.
DRHIT	Data read hit in the L2 cache.
DRREQ	Data read lookup to the L2 cache. Subsequently results in a hit or miss.
DWHIT	Data write hit in the L2 cache.
DWREQ	Data write lookup to the L2 cache. Subsequently results in a hit or miss.
DWTREQ	Data write lookup to the L2 cache with write-through attribute. Subsequently results in a hit or miss.
EPFALLOC	Prefetch hint allocated into the L2 cache.
EPFHIT	Prefetch hint hits in the L2 cache.
EPFRCVDS0	Prefetch hint received by slave port S0.
EPFRCVDS1	Prefetch hint received by slave port S1.
IPFALLOC	Allocation of a prefetch generated by L2 cache controller into the L2 cache.
IRHIT	Instruction read hit in the L2 cache.
IRREQ	Instruction read lookup to the L2 cache. Subsequently results in a hit or miss.
SPNIDEN	Secure privileged non-invasive debug enable.
SRCONFS0	Speculative read confirmed in slave port S0.

continued...

Event	Description
SRCONFS1	Speculative read confirmed in slave port S1.
SRRCVDS0	Speculative read received by slave port S0.
SRRCVDS1	Speculative read received by slave port S1.
WA	Allocation into the L2 cache caused by a write (with write-allocate attribute) miss.

In addition, the L2 cache events can be captured and timestamped using dedicated debugging circuitry.

Related Information

Arm Infocenter
For more information about L2 event capture, refer to the *Debug* chapter of the Cortex-A9 MPCore Technical Reference Manual, available on the Arm Infocenter website.

10.4.1.6. L2 Cache Address Filtering

The L2 cache can access either the system interconnect fabric or the SDRAM L3 interconnect. The L2 cache address filtering determines how much address space is allocated to the HPS-to-FPGA bridge and how much is allocated to SDRAM, depending on the configuration of the memory management unit.

Related Information

- Cortex-A9 MPU Subsystem with System Interconnect on page 283
- Memory Management Unit on page 290
  Information about how the address space is set based on L2 cache address filtering.

10.4.1.7. Cache Latency

Latency for cache hits and misses varies.

The latency for an L1 cache hit is 1 clock. The latency for an L1 cache miss and L2 cache hit is 6 clocks best case. Latency in the L2 cache can vary depending on other operations in the L2. Parity and ECC settings have no effect on latency. A single-bit ECC error is corrected during the L2 read, but is not re-written to the L2 RAM.

10.5. CPU Prefetch

The Cortex-A9 performs instruction and data prefetches regardless of the state of the MMU.

A prefetch occurs when any address that is 4 KB above the current instruction pointer is accessed. The system has been designed to ensure that the system bus does not lock on prefetches. Any prefetches to unmapped memory space produce a decode error on the system interconnect.

10.6. TrustZone

As platforms allow more outside application downloads and as sensitive data is shared on devices, risks of hardware or data asset attacks rise. The Cortex-A9 MPU subsystem has integrated TrustZone technology which provides a system solution to protect application platforms from malicious attack. The TrustZone hardware and supporting software are designed to provide a strong security solution regardless of the operating environment. TrustZone creates a separation between the secure and non-secure areas of the SoC and allows the designer to choose which assets in a design are designated as secure and non-secure.

TrustZone security is implemented in the Cortex-A9 MPU subsystem in three ways:

- Hardware partitioning: Resources can be assigned and identified as secure or non-secure.
- Virtual processor execution: Each core can context switch between secure and non-secure operation.
- Secure debug control: The MPU subsystem provides both secure and non-secure hardware debug features. The type of debug allowed can be configured by the user.

Related Information

TrustZone
For more information about the use of the TrustZone, refer to the *Cortex-A9 MPCore Technical Reference Manual*, available on the Arm Infocenter website.

10.6.1. Secure Partitioning

System interconnect and cache accesses as well as processor execution can be securely partitioned using the TrustZone infrastructure.

10.6.1.1. Secure Bus Accesses

The Cortex-A9 MPCore and other masters that utilize the system interconnect can be configured for secure or non-secure accesses.

Master accesses drive a protection signal, AxPROT[1], to communicate the security level of the attempted transaction. The Cortex-A9 drives this signal low for secure accesses and high for non-secure accesses. The secure firewalls determine whether the access is allowed or not. A slave access type defaults to secure if no other configuration is programmed. Users can define whether a bus error is generated or if a bus failure occurs when a secure access is violated.

Note: Secure processor configuration is programmed through the CP15 register.

Related Information

- Arria 10 HPS Secure Firewalls on page 0
  For more detailed information about secure bus accesses, refer to the "Secure Firewall" section.
- SoC Security on page 168
  For more information about SoC Security, refer to the *SoC Security* Chapter.

10.6.1.2. Secure Cache Accesses

Secure and non-secure partitioning also extends to the L1 and L2 caches.

There is a 32-bit physical address space for secure and non-secure transactions and a 33rd bit is provided to indicate security. The cache is able to store both secure and non-secure lines.

The Accelerator Coherency Port (ACP) in the ArmCortex-A9 MPCore can be used with TrustZone technology. The security state of the masters using the ACP must be the same as the Cortex-A9 processor.

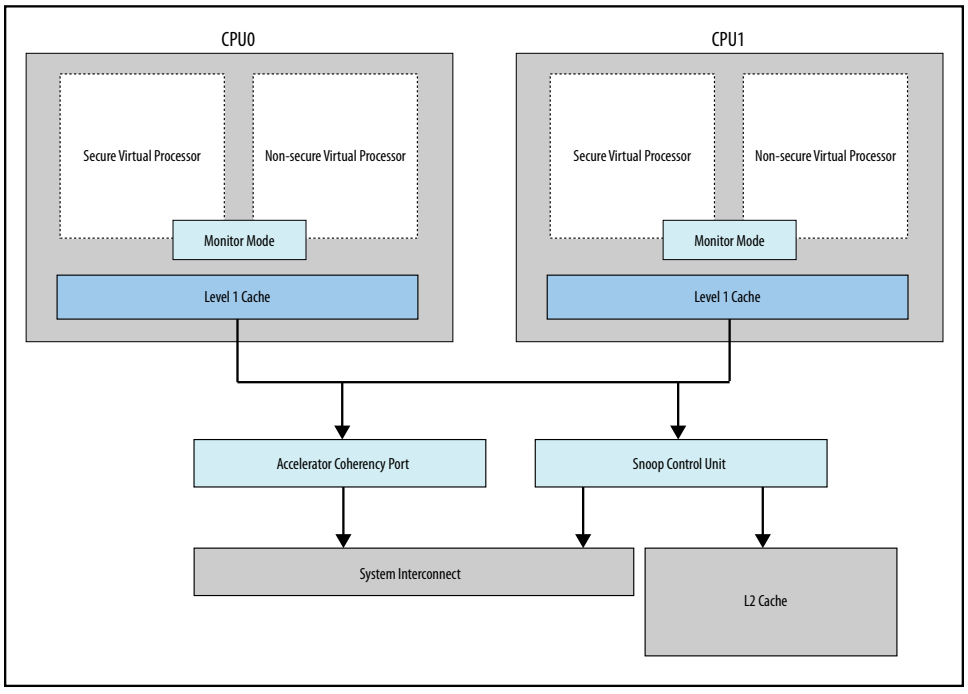
Related Information

- ARM TrustZone
  Refer to this location for information about virtual core operation.
- SoC Security on page 168
  For more information about SoC Security, refer to the *SoC Security* Chapter.

10.6.2. Virtual Processor Operation

Two virtual processors (secure and non-secure) with context switch capability (monitor mode) exist for each Cortex-A9 processor core. The secure virtual processor only accesses secure resources and the non-secure virtual processor only accesses non-secure resources.

Figure 43. Virtual Processor Environment with Monitor Mode



A context switch to secure operation is achieved through a secure monitor call (SMC) instruction or the following hardware exceptions (if configured to do so):

- IRQ interrupt
- Faster Interrupt Request (FIQ) interrupt
- External data abort
- External prefetch abort

When a context switch occurs, the state of the current mode is saved and restored on the next context switch or on a return from exception.

Exception Vector Tables

Three exception vector tables are provided for the MPU with TrustZone:

1. Non-secure
2. Secure
3. Monitor mode

Typically IRQs are used for non-secure interrupts and FIQs are used for secure interrupts. The location of each of the three vector tables can be moved at runtime.

The Generic Interrupt Controller (GIC) can handle secure and non-secure interrupts and prevent non-secure accesses from reading or modifying the configuration of a secure interrupt. An interrupt can be made secure by programming the appropriate bits in the Interrupt Security Register. Secure interrupts are always given a higher priority than non-secure interrupts.

10.6.3. Secure Debug

Debug security extensions exist for the MPU subsystem and the system SoC. Secure privileged and user processor debug access can be controlled by two input pins to the core logic. Depending on the configuration of these pins, secure privileged or user JTAG or trace can be used. In addition, normal user debug visibility can be allowed while preventing all secure world debug. To disable all visibility to the core, both secure and non-secure, a global debug pin, `NIDEN`, is provided to the core. Each Cortex-A9 processor in an MPU subsystem has its own dedicated debug visibility signals.

Related Information

CoreSight Debug and Trace on page 322

10.7. Debugging Modules

The MPU subsystem includes debugging resources through Arm CoreSight on-chip debugging and trace. The following functionality is included:

- Individual program trace for each processor
- Event trace for the Cortex-A9 MPCore
- Cross triggering between processors and other HPS debugging features

10.7.1. Program Trace

Each processor has an independent program trace monitor (PTM) that provides real-time instruction flow trace. The PTM is compatible with a number of third-party debugging tools.

The PTM provides trace data in a highly compressed format. The trace data includes tags for specific points in the program execution flow, called waypoints. Waypoints are specific events or changes in the program flow.

The PTM recognizes and tags the waypoints listed in Table 97 on page 320.

Table 97. Waypoints Supported by the PTM

Type	Additional Waypoint Information
Indirect branches	Target address and condition code
Direct branches	Condition code
Instruction barrier instructions	—
Exceptions	Location where the exception occurred
Changes in processor instruction set state	—
Changes in processor security state	—
Context ID changes	—
Entry to and return from debug state when Halting debug mode is enabled	—

The PTM optionally provides additional information for waypoints, including the following:

- Processor cycle count between waypoints
- Global timestamp values
- Target addresses for direct branches

Related Information

- CoreSight Debug and Trace on page 322
- Arm Infocenter
  For more information about the PTM, refer to the CoreSight PTM-A9 Technical Reference Manual, available on the Arm Infocenter website.

10.7.2. Event Trace

Events from each processor can be used as inputs to the PTM. The PTM can use these events as trace and trigger conditions.

Related Information

- Performance Monitoring Unit on page 293
- Arm Infocenter
  For more information about the trigger and trace capabilities, refer to the CoreSight PTM-A9 Technical Reference Manual, Revision r1p0, available on the Arm Infocenter website.

10.7.3. Cross-Triggering

The PTM can export trigger events and perform actions on trigger inputs. The cross-trigger signals interface with other HPS debugging components including the FPGA fabric. Also, a breakpoint in one processor can trigger a break in the other.

Related Information

CoreSight Debug and Trace on page 322
For detailed information about cross-triggering and about debugging hardware in the MPU, refer to the *CoreSight Debug and Trace* chapter.

10.8. Clocks

Three synchronous clocks and two debug clocks are provided to the MPU subsystem.

Table 98. MPU Subsystem Clocks

System Clock Name	Use
mpu_clk	Main clock for the MPU subsystem
mpu_periph_clk	Clock for peripherals inside the MPU subsystem
mpu_l2_ram_clk	Clock for L2 cache and AXI interface clocks to and from the system interconnect
dbg_at_clk	Trace bus clock
dbg_clk	Debug clock

Related Information

Clock Manager on page 52

10.9. Cortex-A9 MPU Subsystem Register Implementation

The following configurations are available through registers in the Cortex-A9 subsystem:

- All processor-related controls, including the MMU and L1 caches, are controlled using the Coprocessor 15 (CP15) registers of each individual processor.
- All SCU registers, including control for the timers and GIC, are memory mapped.
- All L2 cache registers are memory-mapped.

10.9.1. Cortex-A9 MPU Subsystem Address Map for Arria 10

The Cortex-A9 MPU subsystem registers reside within the address range from 0xFFFFC000 to 0xFFFFEFFF.

10.9.2. L2 Cache Controller Address Map for Arria 10

The register space for the L2 cache controller ranges from 0xFFFFF000 to 0xFFFFFFFF

11. CoreSight Debug and Trace

CoreSight systems provide all the infrastructure you require to debug, monitor, and optimize the performance of a complete HPS design. CoreSight technology addresses the requirement for a multicore debug and trace solution with high bandwidth for whole systems beyond the processor core.

CoreSight technology provides the following features:

- Cross-trigger support between SoC subsystems
- High data compression
- Multi-source trace in a single stream
- Standard programming models for standard tool support

The hard processor system (HPS) debug infrastructure provides visibility and control of the HPS modules, the Arm Cortex-A9 microprocessor unit (MPU) subsystem, and user logic implemented in the FPGA fabric. The debug system design incorporates Arm CoreSight components.

Details of the Arm CoreSight debug components can be viewed by clicking on the following related information links:

Related Information

- Debug Access Port on page 324
- System Trace Macrocell on page 326
- Trace Funnel on page 326
- Embedded Trace FIFO on page 327
- AMBA Trace Bus Replicator on page 328
- Embedded Trace Router on page 327
- Trace Port Interface Unit on page 328
- Embedded Cross Trigger System on page 328
- Program Trace Macrocell on page 332
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

11.1. Features of CoreSight Debug and Trace

The CoreSight debug and trace system offers the following features:

- Real-time program flow instruction trace through a separate Program Trace Macrocell (PTM) for each processor
- Host debugger JTAG interface
- Connections for cross-trigger and STM-to-FPGA interfaces, which enable soft IP generation of triggers and system trace messages
- External trace interface through Trace Port Interface Unit (TPIU) for trace analysis tools
- Custom message injection through the System Trace Macrocell (STM) into the trace stream for delivery to a host debugger
- STM, PTM, and level 3 (L3) trace sources multiplexed into a single stream through the Trace Funnel
- Capability to route trace data to any slave accessible to the Embedded Trace Router (ETR) AXI master connected to the L3 interconnect
- Capability for the following components to trigger each other through the embedded cross-trigger system:
  - Cortex-A9 PTM-0
  - Cortex-A9 PTM-1
  - STM
  - Embedded Trace FIFO (ETF)
  - ETR
  - TPIU
  - Cross Trigger Interface (CTI)
  - FPGA-CTI
  - Cross Trigger Matrix (CTM)

Related Information

Cross Trigger Interface on page 330

11.2. Arm CoreSight Documentation

The following Arm CoreSight specifications and documentation provide a more thorough description of the Arm CoreSight components in the HPS debug system:

- *CoreSight Technology System Design Guide* (Arm DGI 0012D)
- *CoreSight Architecture Specification*
- *Embedded Cross Trigger Technical Reference Manual* (Arm DDI 0291A)
- *CoreSight Components Technical Reference Manual* (Arm DDI 0314H)
- *CoreSight System Trace Macrocell Technical Reference Manual* (Arm DDI 0444A)
- *System Trace Macrocell Programmers' Model Architecture Specification* (Arm IHI 0054)
- *CoreSight Trace Memory Controller Technical Reference Manual* (Arm DDI 0461B)

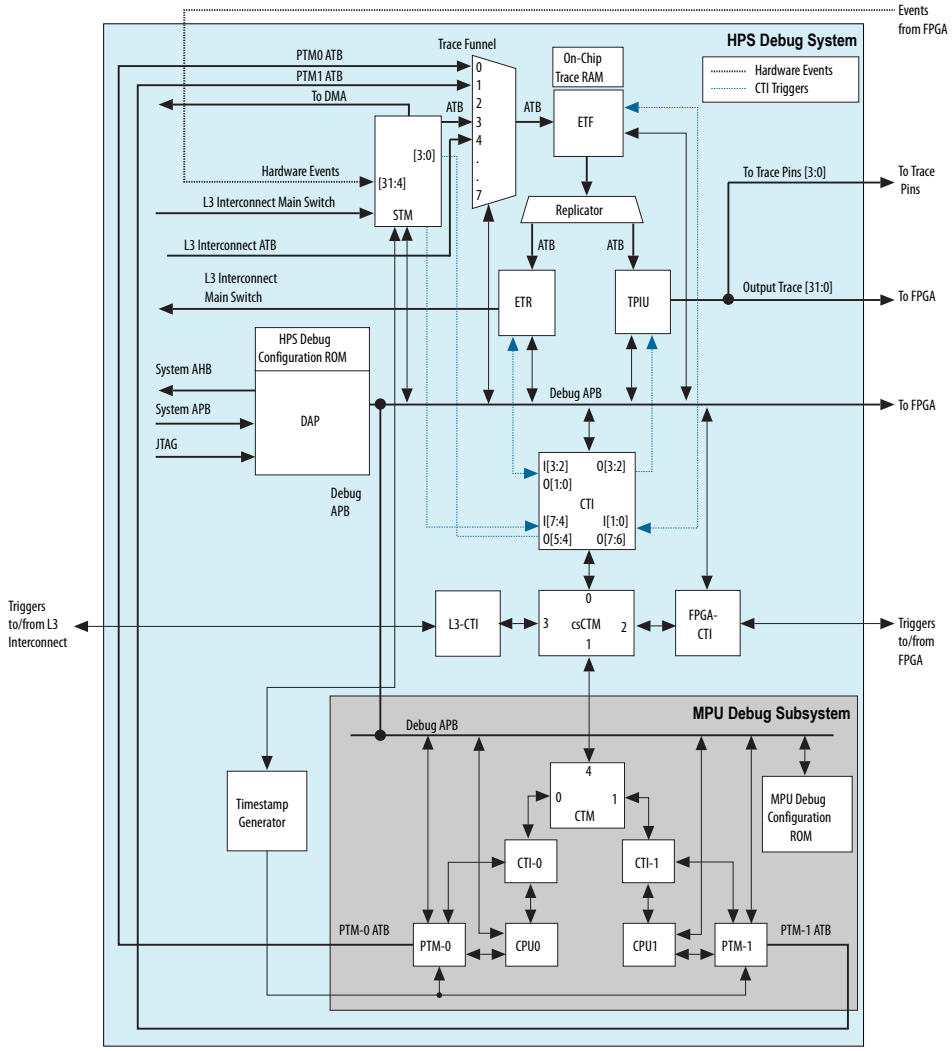
Related Information

Arm Infocenter

You can download these documents from the Arm Infocenter website.

11.3. CoreSight Debug and Trace Block Diagram and System Integration

Figure 44. HPS CoreSight Debug and Trace System Block Diagram



11.4. Functional Description of CoreSight Debug and Trace

11.4.1. Debug Access Port

The Debug Access Port (DAP) provides the necessary ports for a host debugger to connect to and communicate with the HPS through a JTAG interface, which is connected to the FPGA JTAG chain. The JTAG interface provided with the DAP allows a

host debugger to access various modules inside the HPS. Additionally, a debug monitor executing on either processor can access different HPS components by interfacing with the system Advanced Microcontroller Bus Architecture (AMBA) Advanced Peripheral Bus (APB) slave port of the DAP.

The system APB slave port occupies 2 MB of address space in the HPS. Both the JTAG port and system APB port have access to the debug APB master port of the DAP.

A host debugger can access any HPS memory-mapped resource in the system through the DAP system master port. Requests made over the DAP system master port are impacted by reads and writes to peripheral registers.

Note: The HPS JTAG interface does not support boundary scan tests (BST). In order to perform boundary scan testing on HPS I/Os, use the FPGA JTAG pins. This can be performed only when the HPS is powered on and the **HPS_nRST** is de-asserted.

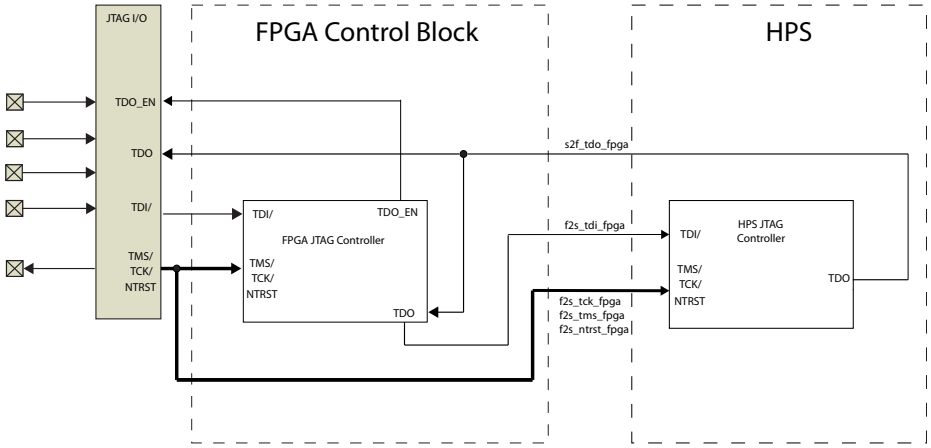
Related Information

- CoreSight Debug and Trace Block Diagram and System Integration on page 324
  Shows CoreSight components connected to the debug APB
- Arria 10 Core Fabric and General Purpose I/Os Handbook
  For more information about boundary scan tests, refer to the "JTAG Boundary-Scan Testing in Arria 10 Devices" chapter.
- Arm Infocenter
  Refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

11.4.1.1. JTAG Connection

The SoC device has two JTAG controllers: the FPGA JTAG (located in the configuration sub-system (CSS)) and the HPS JTAG (located in the HPS). The controllers have separate instruction sets and work independently. To share the same external JTAG port between the two controllers, they are internally daisy-chained, where the FPGA appears before the HPS on the chain.

The following figure shows how the two test access port (TAP) controllers are connected:



11.4.2. System Trace Macrocell

The STM allows messages to be injected into the trace stream for delivery to the host debugger receiving the trace data. These messages can be sent through stimulus ports or the hardware EVENT interface. The STM allows these messages to be time stamped.

The STM provides an AMBA Advanced eXtensible Interface (AXI) slave interface used to create trace events. The interface can be accessed by the MPU subsystem, direct memory access (DMA) controller, and masters implemented as soft logic in the FPGA fabric through the FPGA-to-HPS bridge. The AXI slave interface supports three address segments, where each address segment is 16 MB and each segment supports up to 65536 channels. Each channel occupies 256 bytes of address space.

The STM also provides 32 hardware EVENT pins. The higher-order 28 pins (31:4) are connected to the FPGA fabric, allowing logic inside FPGA to insert messages into the trace stream. When the STM detects a rising edge on an EVENT pin, a message identifying the EVENT is inserted into the stream. The lower four EVENT pins (3:0) are connected to CTI.

Related Information

- HPS-FPGA Bridges on page 248
- Arm Infocenter
  For more information, refer to the *CoreSight System Trace Macrocell Technical Reference Manual* on the Arm Infocenter website.
- CTI on page 339

11.4.3. Trace Funnel

The Trace Funnel is used to combine multiple trace streams into one trace stream. There are four trace streams that use the following funnel ports:

Table 99. Trace Stream Connections

Funnel Port	Description
0	The trace stream coming from the PTM connected to CPU0 uses this port.
1	The trace stream coming from the PTM connected to CPU1 uses this port.
2	Not connected.
3	The trace stream coming from the STM uses this port.
4	NoC trace data uses this port.
5 .. 7	Not connected.

Related Information

- Arm Infocenter
  Refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.
- Program Trace Macrocell on page 332

11.4.4. CoreSight Trace Memory Controller

The CoreSight Trace Memory Controller (TMC) has three possible configurations:

- Embedded Trace FIFO (ETF)
- Embedded Trace Router (ETR)
- Embedded Trace Buffer (ETB)

ETB is not used in this device.

Related Information

Arm Infocenter
For more information, refer to the *CoreSight System Trace Memory Controller Technical Reference Manual* on the Arm Infocenter website.

11.4.4.1. Embedded Trace FIFO

The Trace Funnel output is sent to the ETF. The ETF is used as an elastic buffer between trace generators (STM, PTM, L3 interconnect) and trace destinations. The ETF stores up to 32 KB of trace data in the on-chip trace RAM.

Related Information

Arm Infocenter
For more information, refer to the *CoreSight System Trace Memory Controller Technical Reference Manual* on the Arm Infocenter website.

11.4.4.2. Embedded Trace Router

The ETR can route trace data to the HPS on-chip RAM, the HPS SDRAM, and any memory in the FPGA fabric connected to the HPS-to-FPGA bridge. The ETR receives trace data from the AMBA Trace Bus Replicator. By default, the buffer to receive the trace data resides in SDRAM at offset 0x00100000 and is 32 KB. You can override this default configuration by programming registers in the ETR.

Related Information

- HPS-FPGA Bridges on page 248
- CoreSight Debug and Trace Programming Model on page 337
- Arm Infocenter
  For more information, refer to the *CoreSight System Trace Memory Controller Technical Reference Manual* on the Arm Infocenter website.

11.4.5. AMBA Trace Bus Replicator

The AMBA Trace Bus Replicator broadcasts trace data from the ETF to the ETR and TPIU.

Related Information

Arm Infocenter
Refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

11.4.6. Trace Port Interface Unit

The TPIU is a bridge between on-chip trace sources and an off-chip trace port. The TPIU receives trace data from the Trace Bus Replicator (Replicator) and drives the trace data to a trace port analyzer.

The trace output is routed to a 32-bit interface to the FPGA fabric. The trace data sent to the FPGA fabric can be transported off-chip using available serializer/deserializer (SERDES) resources in the FPGA.

Related Information

Arm Infocenter
Refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

11.4.7. Embedded Cross Trigger System

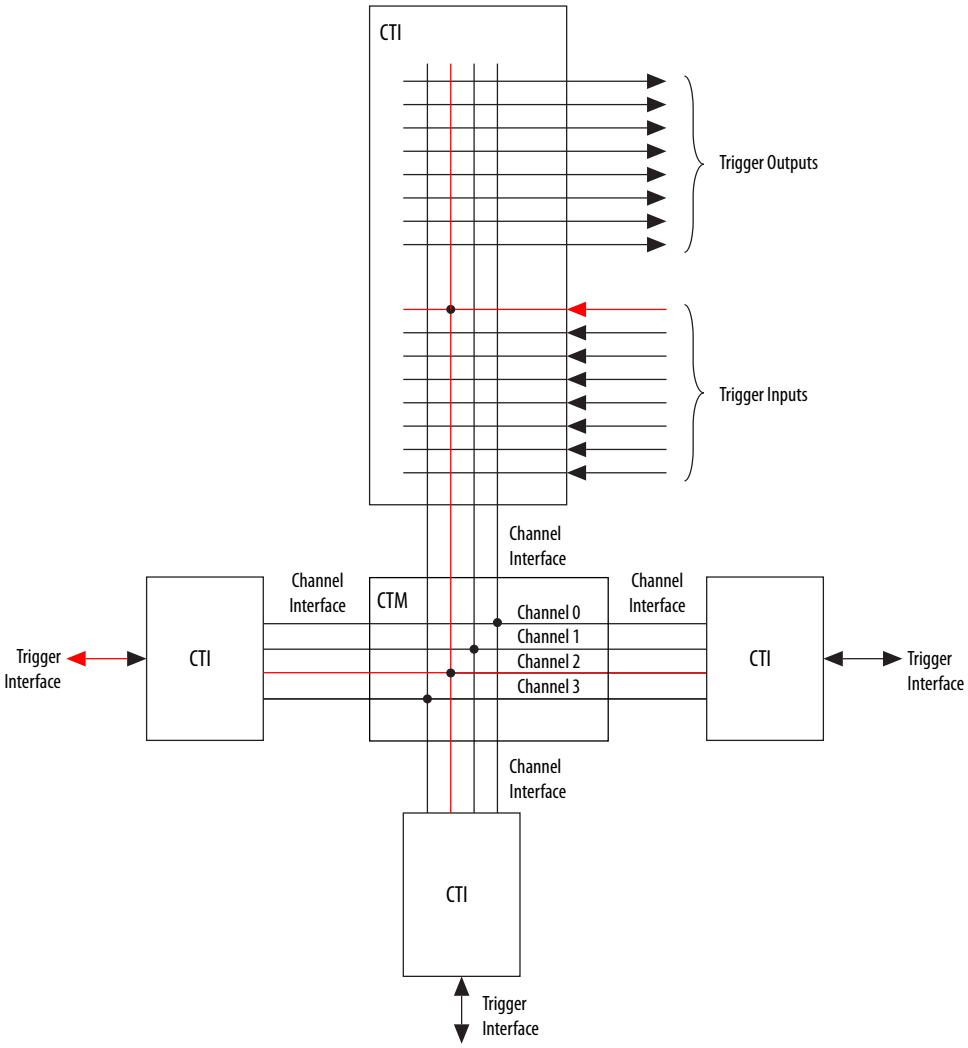
The ECT system provides a mechanism for the components listed in "Features of the CoreSight Debug and Trace" to trigger each other. The ECT consists of the following modules:

- Cross Trigger Interface (CTI)
- Cross Trigger Matrix (CTM)

Figure 45. Generic ECT System

The following figure shows an example of how CTIs and CTMs are used in a generic ECT setup. Though the signal travels through channel 2, it only enters and exits through trigger inputs and outputs you configure.

Note: The red line depicts an trigger input to one CTI generating a trigger output in another CTI.



Related Information

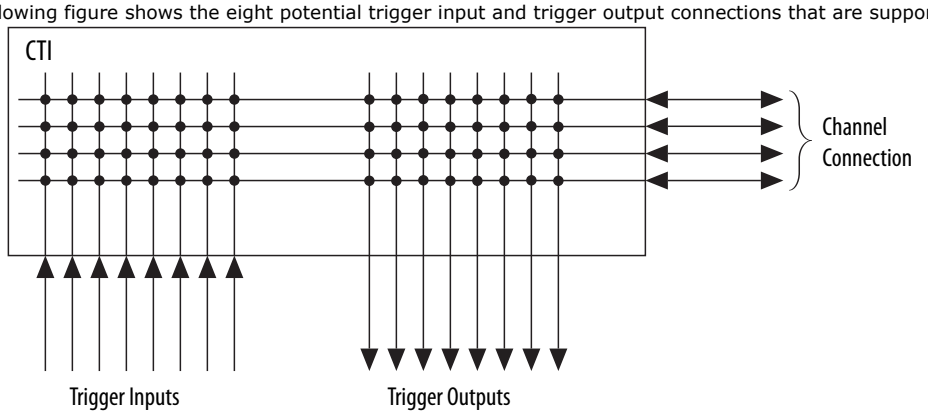
- Cross Trigger Interface on page 330
- Features of CoreSight Debug and Trace on page 323
- Cross Trigger Matrix on page 330

11.4.7.1. Cross Trigger Interface

CTIs allow trigger sources and sinks to interface with the ECT. Each CTI supports up to eight trigger inputs and eight trigger outputs, and is connected to a CTM. Figure 46 on page 330 shows the relationship of trigger inputs, trigger outputs, and CTM channels of a CTI.

Figure 46. CTI Connections

The following figure shows the eight potential trigger input and trigger output connections that are supported.



The HPS debug system contains the following CTIs:

- CTI—performs cross triggering between the STM, ETF, ETR, and TPIU.
- FPGA-CTI—exposes the cross-triggering system to the FPGA fabric.
- CTI-0 and CTI-1—reside in the MPU debug subsystem. Each CTI is associated with a processor and the processor's PTM.
- L3-CTI—allows cross triggering with L3 interconnect.

11.4.7.2. Cross Trigger Matrix

A CTM is a transport mechanism for triggers traveling from one CTI to one or more CTIs or CTMs. The HPS contains two CTMs. One CTM connects CTI and FPGA-CTI; the other connects CTI-0 and CTI-1. The two CTMs are connected together, allowing triggers to be transmitted between the MPU debug subsystem, the debug system, and the FPGA fabric.

Each CTM has four ports and each port has four channels. Each CTM port can be connected to a CTI or another CTM.

Figure 47. CTM Channel Structure

The following figure shows the structure of a CTM channel. Paths inside the CTM are purely combinatorial.

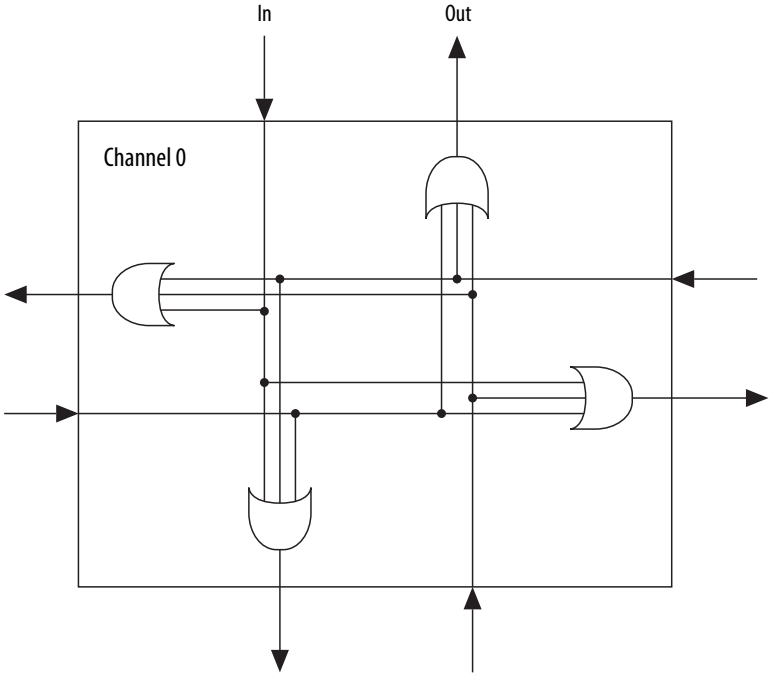
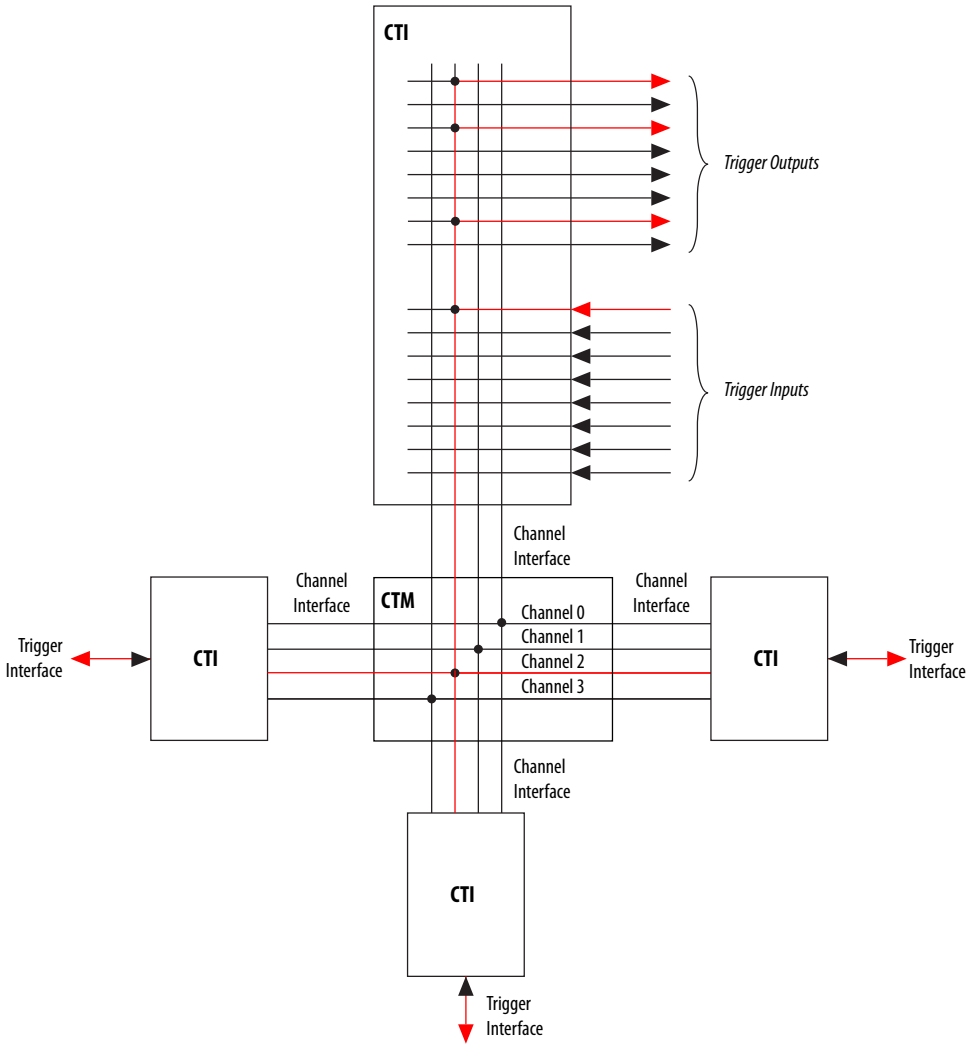


Figure 48. CTI Trigger Connections

Each CTI trigger input can be connected through a CTM to one or more trigger outputs under control by a debugger. The following figure shows an example of CTI trigger connections.

Note: The red lines depict the impact one trigger input can have on the entire system.



Related Information

Arm Infocenter

Refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

11.4.8. Program Trace Macrocell

The PTM performs real-time program flow instruction tracing and provides a variety of filters and triggers that can be used to trace specific portions of code.

The HPS contains two PTMs. Each PTM is paired with a processor and CTI. Trace data generated from the PTM can be transmitted off-chip using HPS pins, or to the FPGA fabric, where it can be pre-processed and transmitted off-chip using high-speed FPGA pins.

Related Information

Arm Infocenter
For more information, refer to the *CoreSight PTM-A9 Technical Reference Manual* .

11.4.9. HPS Debug APB Interface

The HPS can extend the CoreSight debug control bus into the FPGA fabric. The debug interface is an APB-compatible interface with built-in clock crossing.

Related Information

- FPGA Interface on page 333
- HPS Component Interfaces on page 731

11.4.10. FPGA Interface

The following components connect to the FPGA fabric. This section lists the signals from the debug system to the FPGA.

11.4.10.1. DAP

The DAP uses the system APB port to connect to the FPGA.

Table 100. DAP

The following table shows the signal description between DAP and FPGA.

Signal	Description
h2f_dbg_apb_PADDR	Address bus to system APB port, when PADDR
h2f_dbg_apb_PADDR31	Address bus to system APB port, when PADDR31
h2f_dbg_apb_PENABLE	Enable signal from system APB port
h2f_dbg_apb_PRDATA[32]	32-bit system APB port read data bus
h2f_dbg_apb_PREADY	Ready signal to system APB port
h2f_dbg_apb_PSEL	Select signal from system APB port
h2f_dbg_apb_PSLVERR	Error signal to system APB port
h2f_dbg_apb_PWDATA[32]	32-bit system APB port write data bus
h2f_dbg_apb_PWRITE	Select whether read or write to system APB port • 0 - System APB port read from DAP • 1 - System APB Port write to DAP

11.4.10.2. STM

The STM has 28 event pins, `f2h_stm_hw_events[28]`, for FPGA to trigger events to STM.

11.4.10.3. FPGA-CTI

The FPGA-CTI allows the FPGA to send and receive triggers from the debug system.

Table 101. FPGA-CTI Signal Description Table

The following table lists the signal descriptions between the FPGA-CTI and FPGA.

Signal	Description
h2f_cti_trig_in[8]	Trigger input from FPGA
h2f_cti_trig_in_ack[8]	ACK signal to FPGA
h2f_cti_trig_out[8]	Trigger output to FPGA
h2f_cti_trig_out_ack[8]	ACK signal from FPGA
h2f_cti_clk	Clock input from FPGA
h2f_cti_fpga_clk_en	Clock enable driven by FPGA

Related Information

Arm Infocenter
For more information about the cross-trigger interface

11.4.10.4. CTI-NoC

The CTI is used for receiving triggers from device that can generate triggers and for sending triggers to devices that can receive triggers. This instance of CTI is used exclusively by NoC and allows NoC probe units to send and receive triggers.

Table 102. Trigger Input Interface

Name	Input/Output	Description
CTITRIGIN	Input	Trigger In Connected to noc_CTITRIGIN
TISBYPASSIN	Input	8'b0000_0000 —Trigger In synchronization bypass (Static value) Enable synchronizers for all 8 inputs. CTI runs off cs_at_clk
CTITRIGINACK	Output	Trigger In acknowledge Connected to noc_CTITRIGINACK

Table 103. Trigger Output Interface

Name	Input/Output	Description
CTITRIGOUT	Output	Trigger Out Connected to noc_CTITRIGOUT
CTITRIGOUTACK	Input	Trigger Out acknowledge Connected to noc_CTITRIGOUTACK
TISBYPASSACK	Input	8'b0000_0000—Trigger Out acknowledge synchronization bypass (Static value) Enable synchronizers for all 8 inputs
TIHSBYPASS	Input	8'b0000_0000—Trigger interface handshake bypass (Static value) Hand shaking is enabled

11.4.10.5. TPIU

Figure 49. Trace Clock and Trace Data Width

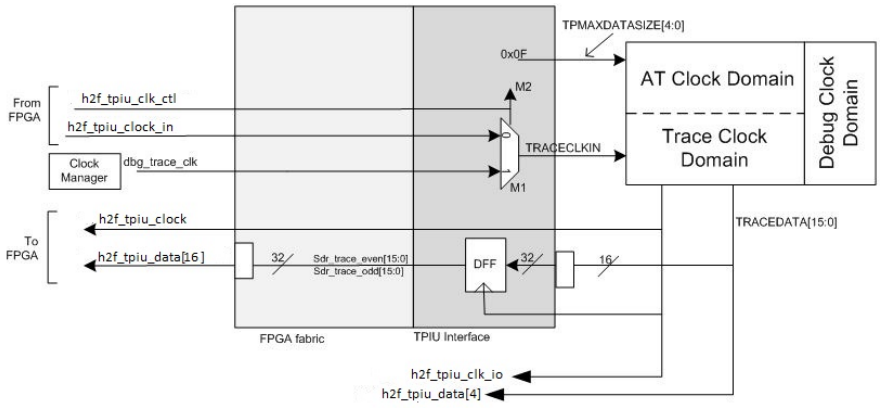


Table 104. TPIU Signals

The following table lists the signal descriptions between the TPIU and FPGA.

Signal	Description
`h2f_tpiu_clk_ctl`	Selects whether trace data is captured using the internal TPIU clock, which is the `dbg_trace_clk` signal from the clock manager; or an external clock provided as an input to the TPIU from the FPGA. 0 - use `h2f_tpiu_clock_in` 1 - use internal clock **Note**: When the FPGA is powered down or not configured the TPIU uses the internal clock.
`h2f_tpiu_data[16]`	16-bit trace data bus. Trace data changes on both edges of `h2f_tpiu_clock`.
`h2f_tpiu_clock_in`	Clock from the FPGA used to capture trace data.
`h2f_tpiu_clock`	Clock output from TPIU

11.4.11. Debug Clocks

Table 105. CoreSight Clocks

Port Name	Clock Source	Signal Name	Description
ATCLK	Clock manager	`cs_at_clk`	Trace bus clock.
CTICLK (for CTI)	Clock manager	`cs_at_clk`	Cross trigger interface clock for CTI. It can be synchronous or asynchronous to CTMCLK.
CTICLK (for FPGA-CTI)	FPGA fabric	`cs_at_clk`	Cross trigger interface clock for FPGA-CTI.
CTICLK (for CTI-0 and CTI-1)	Clock manager	`mpu_clk`	Cross trigger interface clock for CTI-0 and CTI-1. It can be synchronous or asynchronous to CTMCLK.

continued...

Port Name	Clock Source	Signal Name	Description
CTMCLK (for csCTM)	Clock manager	cs_pdbg_clk	Cross trigger matrix clock for csCTM. It can be synchronous or asynchronous to CTICLK.
CTMCLK (for CTM)	Clock manager	mpu_clk	Cross trigger matrix clock for CTM. It can be synchronous or asynchronous to CTICLK.
DAPCLK	Clock manager	cs_pdbg_clk	DAP internal clock. It must be equivalent to PCLKDBG.
PCLKDBG	Clock manager	cs_pdbg_clk	Debug APB (DAPB) clock.
HCLK	Clock manager	cs_pdbg_clk	Used by the AHB-Lite master inside the DAP. It is asynchronous to DAPCLK. In the HPS, the AHB-Lite port uses same clock as DAPCLK.
PCLKSYS	Clock manager	cs_pdbg_clk	Used by the APB slave port inside the DAP. It is asynchronous to DAPCLK.
SWCLKTCK	JTAG interface	dap_tck	The SWJ-DP clock driven by the external debugger through the JTAG interface. It is asynchronous to DAPCLK. When through the JTAG interface, this clock is the same as TCK of the JTAG interface.
TRACECLKIN	Clock manager	cs_trace_clk	TPIU trace clock input. It is asynchronous to ATCLK. In the HPS, this clock can come from the clock manager or the FPGA fabric.

Related Information

Arm Infocenter

For more information about the CoreSight port names, refer to the *CoreSight Technology System Design Guide*.

11.4.12. Debug Resets

The CoreSight system uses several resets.

Table 106. CoreSight Resets

Arm Reset Name	Clock Source	HPS Reset Signal Name	Description
ATRESETn	Reset manager	dbg_rst_n	Trace bus reset. It resets all registers in the ATCLK domain.
nCTIRESET	Reset manager	dbg_rst_n	CTI reset signal. It resets all registers in the CTICLK domain. In the HPS, there are four instances of CTI. All four use the same reset signal.
DAPRESETn	Reset manager	dbg_rst_n	DAP internal reset. It is connected to PRESETDBGn.

continued...

Arm Reset Name	Clock Source	HPS Reset Signal Name	Description
PRESETDBGn	Reset manager	dbg_rst_n	Debug APB reset. Resets all registers clocked by PCLKDBG.
HRESETn	Reset manager	sys_dbg_rst_n	SoC-provided reset signal that resets all of the AMBA on-chip interconnect. Use this signal to reset the DAP AHB-Lite master port.
PRESETSYSn	Reset manager	sys_dbg_rst_n	Resets system APB slave port of DAP.
nCTMRESET	Reset manager	dbg_rst_n	CTM reset signal. It resets all signals clocked by CTMCLK.
nPOTRST	Reset manager	tap_cold_rst_n	True power on reset signal to the DAP SWJ-DP. It must only reset at power-on.
nTRST	JTAG interface	nTRST pin	Resets the DAP TAP controller inside the SWJ-DP. This signal is driven by the host using the JTAG connector.
TRESETn	Reset manager	dbg_rst_n	Reset signal for TPIU. Resets all registers in the TRACECLKIN domain.

The ETR stall enable field (etrstallen) of the ctrl register in the reset manager controls whether the ETR is requested to stall its AXI master interface to the L3 interconnect before a warm or debug reset.

The level 4 (L4) watchdog timers can be paused during debugging to prevent reset while the processor is stopped at a breakpoint.

Related Information

- Reset Manager on page 134
- Watchdog Timer on page 693
- Arm Infocenter
  For more information about the CoreSight port names, refer to the *CoreSight Technology System Design Guide*.

11.5. CoreSight Debug and Trace Programming Model

This section describes programming model details specific to the device implementation of the Arm CoreSight technology.

The debug components can be configured to cause triggers when certain events occur. For example, soft logic in the FPGA fabric can signal an event which triggers an STM message injection into the trace stream.

Related Information

Arm Infocenter
Programming interface details of each CoreSight component.

11.5.1. Coresight Component Address

CoreSight components are configured through memory-mapped registers, located at offsets relative to the CoreSight component base address. CoreSight component base addresses are accessible through the component address table in the DAP ROM.

Table 107. Coresight Component Address Table

The following table is located in the ROM table portion of the DAP.

ROM Entry	Offset[30:12]	Description
0x0	0x00001	ETF Component Base Address
0x1	0x00002	CTI Component Base Address
0x2	0x00003	TPIU Component Base Address
0x3	0x00004	Trace Funnel Component Base Address
0x4	0x00005	STM Component Base Address
0x5	0x00006	ETR Component Base Address
0x6	0x00007	FPGA-CTI Component Base Address
0x7	0x00100	A9 ROM
0x8	0x00080	FPGA ROM
0x9	0x00008	L3-CTI

A host debugger can access this table at 0x80000000 through the DAP. HPS masters can access this ROM at 0xFF000000. Registers for a particular CoreSight component are accessed by adding the register offset to the CoreSight component base address, and adding that total to the base address of the ROM table.

The base address of the ROM table is different when accessed from the debugger (at 0x8000_0000) than when accessed from any HPS master (at 0xFF000000). For example, the CTI output enable (CTIOUTEN) register, `CTIOUTEN[2]` at offset 0xA8, can be accessed by the host debugger at 0x800020A8. To derive that value, add the host debugger access address to the ROM table of 0x80000000, to the CTI component base address of 0x00002000, to the `CTIOUTEN[2]` register offset of 0xA8.

11.5.2. STM Channels

The STM AXI slave is connected to the MPU, DMA, and FPGA-to-HPS bridge masters. Each master has up to 65536 channels where each channel occupies 256 bytes of address space, for a total of 16 MB per master. The HPS address map allocates 48 MB of consecutive address space to the STM AXI slave port, divided in three 16 MB segments.

Table 108. STM AXI Slave Port Address Allocation

Segment	Start Address	End Address
0	0xFC000000	0xFCFFFFFF
1	0xFD000000	0xFDFFFFFF
2	0xFE000000	0xFEFFFFFF

Each of the three masters can access any one of the three address segments. Your software design determines which master uses which segment, based on the value of bits 24 and 25 in the write address, `AWADDRS[25:24]`. Software must restrict each master to use only one of the three segments.

Table 109. STM AXI Address Fields

STM receives trace data over an AXI slave port. This table contains a list of signals associated with this interface.

AXI Signal Fields	Description
AWADDRS[7:0]	These bits index the 256 bytes of the stimulus port.
AWADDRS[23:8]	These bits identify the 65536 stimulus ports associated with a master.
AWADDRS[25:24]	These bits identify the three masters. Only 0, 1, and 2 are valid values.
AWADDRS[31:26]	Always 0x3F. Bits 24 to 31 combine to access 0xFC000000 through 0xFEFFFFFF.

Each STM message contains a master ID that tells the host debugger which master is associated with the message. The STM master ID is determined by combining a portion of the AWADDRS signal and the AWPROT protection bit. In the following table, MasterID[6] identifies the

Table 110. STM Master ID Calculation

Master ID Bits	AXI Signal Bits	Notes
Master ID[5:0]	AWADDRS[29:24]	The lowest two bits are sufficient to determine which master, but CoreSight uses a six-bit master ID.
Master ID[6]	AWPROT[1]	0 indicates a secure master; 1 indicates a nonsecure master.

In addition to access through STM channels, the higher-order 28 (31:4) of the 32 event signals are attached to the FPGA through the FPGA-CTI. These event signals allow the FPGA fabric to send additional messages using the STM.

Related Information

- HPS-FPGA Bridges on page 248
- Arm Infocenter
  For more information, refer to "System Trace Macrocell" in the *Programmers' Model Architecture Specification* .
- FPGA-CTI on page 334

11.5.3. CTI Trigger Connections to Outside the Debug System

The following CTIs in the HPS debug system connect to outside the debug system:

- CTI
- FPGA-CTI
- L3-CTI

11.5.3.1. CTI

This section lists the trigger input, output, and output acknowledge pin connections implemented for CTI in the debug system. The trigger input acknowledge signals are not connected to pins.

Table 111. CTI Trigger Input Signals

The following table lists the trigger input pin connections implemented for CTI .

Number	Signal	Source
7	ASYNCOUT	STM
6	TRIGOUTHETE	STM
5	TRIGOUTSW	STM
4	TRIGOUTSPTE	STM
3	ACQCOMP	ETR
2	FULL	ETR
1	ACQCOMP	ETF
0	FULL	ETF

Table 112. CTI Trigger Output Signals

The following table lists the trigger output pin connections implemented for CTI .

Number	Signal	Destination
7	TRIGIN	ETF
6	FLUSHIN	ETF
5	HWEVENTS[3:2]	STM
4	HWEVENTS[1:0]	STM
3	TRIGIN	TPIU
2	FLUSHIN	TPIU
1	TRIGIN	ETR
0	FLUSHIN	ETR

Table 113. CTI Trigger Output Acknowledge Signals

The following table lists the trigger output pin acknowledge connections implemented for CTI .

Number	Signal	Source
7	0	—
6	0	—
5	0	—
4	0	—
3	TRIGINACK	TPIU
2	FLUSHINACK	TPIU
1	0	—
0	0	—

11.5.3.2. FPGA-CTI

FPGA-CTI connects the debug system to the FPGA fabric. FPGA-CTI has all of its triggers available to the FPGA fabric.

Related Information

Configuring Embedded Cross-Trigger Connections on page 341
For more information about the triggers, refer to the "Configuring Embedded Cross-Trigger Connections" chapter.

11.5.3.3. L3-CTI

L3-CTI has all of its triggers available to the L3 interconnect.

11.5.4. Configuring Embedded Cross-Trigger Connections

CTI interfaces are programmable through a memory-mapped register interface.

The specific registers are described in the *CoreSight Components Technical Reference Manual*, which you can download from the Arm Infocenter.

To access registers in any CoreSight component through the debugger, the register offsets must be added to the CoreSight component's base address. That combined value must then be added to the address at which the ROM table is visible to the debugger (0x80000000).

Each CTI has two interfaces, the trigger interface and the channel interface. The trigger interface is the interface between the CTI and other components. It has eight trigger signals, which are hardwired to other components. The channel interface is the interface between a CTI and its CTM, with four bidirectional channels. The mapping of trigger interface to channel interface (and vice versa) in a CTI is dynamically configured. You can enable or disable each CTI trigger output and CTI trigger input connection individually.

For example, you can configure trigger input 0 in the FPGA-CTI to route to channel 3, and configure trigger output 3 in the FPGA-CTI and trigger output 7 in CTI-0 in the MPU debug subsystem to route from channel 3. This configuration causes a trigger at trigger input 0 in FPGA-CTI to propagate to trigger output 3 in the FPGA-CTI and trigger output 7 in CTI-0. Propagation can be single-to-single, single-to-multiple, multiple-to-single, and multiple-to-multiple.

A particular soft logic signal in the FPGA connected to a trigger input in the FPGA-CTI can be configured to trigger a flush of trace data to the TPIU. For example, you can configure channel 0 to trigger output 2 in the CTI. Then configure trigger input T3 to channel 0 in FPGA-CTI. Trace data is flushed to the TPIU when a trigger is received at trigger output 2 in the CTI.

Another soft logic signal in the FPGA connected to trigger input T2 in the FPGA-CTI can be configured to trigger a STM message. The CTI output triggers 4 and 5 are wired to the STM CoreSight component in the HPS. For example, configure channel 1 to trigger output 4 in the CTI; and then configure trigger input T2 to channel 1 in FPGA-CTI.

Another soft logic signal in the FPGA fabric connected to trigger input T1 in FPGA-CTI can be configured to trigger a breakpoint on CPU 1. Trigger output 1 in CTI-1 is wired to the debug request (EDBGRQ) signal of CPU-1. For example, configure channel 2 to trigger output 1 in CTI-1. Then configure trigger input T1 to channel 2 in FPGA-CTI.

Related Information

- Coresight Component Address on page 337

- Arm Infocenter
  For more information about the cross-trigger interface

11.5.4.1. Configuring Trigger Input 0

For example, you can configure trigger input 0 in the FPGA-CTI to route to channel 3, and configure trigger output 3 in the FPGA-CTI and trigger output 7 in CTI-0 in the MPU debug subsystem to route from channel 3. This configuration causes a trigger at trigger input 0 in FPGA-CTI to propagate to trigger output 3 in the FPGA-CTI and trigger output 7 in CTI-0. Propagation can be single-to-single, single-tomultiple, multiple-to-single, and multiple-to-multiple.

11.5.4.2. Triggering a Flush of Trace Data to the TPIU

A particular soft logic signal in the FPGA connected to a trigger input in the FPGA-CTI can be configured to trigger a flush of trace data to the TPIU. For example, you can configure channel 0 to trigger output 2 in CTI. Then configure trigger input T3 to channel 0 in FPGA-CTI. Trace data is flushed to the TPIU when a trigger is received at trigger output 2 in CTI.

11.5.4.3. Triggering an STM message

Another soft logic signal in the FPGA connected to trigger input T2 in FPGA-CTI can be configured to trigger an STM message. CTI output triggers 4 and 5 are wired to the STM CoreSight component in the HPS. For example, configure channel 1 to trigger output 4 in CTI. Then configure trigger input T2 to channel 1 in FPGA-CTI.

11.5.4.4. Triggering a Breakpoint on CPU 1

Another soft logic signal in the FPGA fabric connected to trigger input T1 in FPGA-CTI can be configured to trigger a breakpoint on CPU 1. Trigger output 1 in CTI-1 is wired to the external debug request (EDBGRQ) signal of CPU-1. For example, configure channel 2 to trigger output 1 in CTI-1. Then configure trigger input T1 to channel 2 in FPGA-CTI.

11.6. CoreSight Debug and Trace Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions .

11.6.1. stm Address Map

This address space is allocated for the CoreSight System Trace Macrocell (STM). For more information about the STM, refer to the ARM Infocenter.

Module Instance	Base Address	End Address
i_stm	0xFC000000	0xFEFFFFFF

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

11.6.2. dap Address Map

This address space is allocated for the Debug Access Port (DAP). For more information about the DAP, refer to the ARM Infocenter.

Module Instance	Base Address	End Address
i_dap	0xFF000000	0xFF1FFFFF

Register Group	Description	Start Address	End Address
DAP ROM	This address space is allocated for the Debug Access Port (DAP) ROM.	0xFF000000	0xFF000FFF
ETF	This address space is allocated for the Embedded Trace FIFO.	0xFF001000	0xFF001FFF
CTI	This address space is allocated for the Cross-Trigger Interface (CTI).	0xFF002000	0xFF002FFF
TPIU	This address space is allocated for the Trace Port Interface Unit.	0xFF003000	0xFF003FFF
Trace Funnel	This is the address space is allocated for the Trace Funnel.	0xFF004000	0xFF004FFF
STM	This address space is allocated for the System Trace Macrocell (STM).	0xFF005000	0xFF005FFF
ETR	This address space is allocated for the Embedded Trace Router.	0xFF006000	0xFF006FFF
CTI FPGA	This address space is allocated for the Cross-Trigger Interface of the FPGA.	0xFF007000	0xFF007FFF
FPGA ROM	This address space is allocated for the FPGA ROM.	0xFF080000	0xFF080FFF
FPGA CoreSight Components	This address space is allocated for the FPGA CoreSight Components.	0xFF081000	0xFF0FF000
Cortex-A9 ROM	This address space is allocated for the Cortex-A9 ROM.	0xFF100000	0xFF10FFFF
CPU0 Debug	This address space is allocated for CPU0 Debug.	0xFF110000	0xFF110FFF
CPU0 PMU	This address space is allocated for the CPU0 PMU.	0xFF111000	0xFF111FFF
CPU1 Debug	This address space is allocated for CPU1 Debug.	0xFF112000	0xFF112FFF
CPU1 PMU	This address space is allocated for the CPU1 PMU.	0xFF113000	0xFF117FFF

continued...

Register Group	Description	Start Address	End Address
CTI0	This address space is allocated for Cross-Trigger Interface 0 (CTI0).	0xFF118000	0xFF118FFF
CTI1	This address space is allocated for Cross-Trigger Interface 1 (CTI1)	0xFF119000	0xFF11BFFF
PTM0	This addres space is allocated for Program Trace Macrocell 0 (PTM0)	0xFF11C000	0xFF11CFFF
PTM1	This address space is allocated for Program Trace Macrocell 1 (PTM1).	0xFF11D000	0xFF11DFFF

Important: To prevent indeterminate system behavior, reserved areas of memory must not be accessed by software or hardware. Any area of the memory map that is not explicitly defined as a register space or accessible memory is considered reserved.

12. Error Checking and Correction Controller

Error Checking and Correction (ECC) controllers provide single- and double-bit error memory protection for integrated on-chip RAM and peripheral RAMs within the hard processor system (HPS).

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

12.1. ECC Controller Features

The features supported by each ECC controller are:

- Hamming code-based ECC calculations
- Single-bit error detection and correction
- Double-bit error detection
- Dedicated hardware block for memory data initialization
- Indirect memory access for:
  - Data correction on the corrupted memory address
  - Data and ECC syndrome bit error injection
- Watchdog timeout for indirect access to prevent bus stall
- Display of the current single or double-bit error memory address
- Single-bit error occurrence counter
- Look-up table (LUT) for logging single-bit error memory address
- Interrupt generated upon single and double-bit errors
- User-controllable interrupt assertion for test purposes

12.2. ECC Supported Memories

In addition to the 256 KB on-chip RAM, the peripherals that have integrated memories with ECC controllers are:

- USB OTG 0/1
- SD/MMC controller
- Ethernet MAC 0/1/2

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- DMA controller
- NAND flash controller
- QSPI flash controller

Note: The L2 cache and the SDRAM interface have their own dedicated ECC support.

Related Information

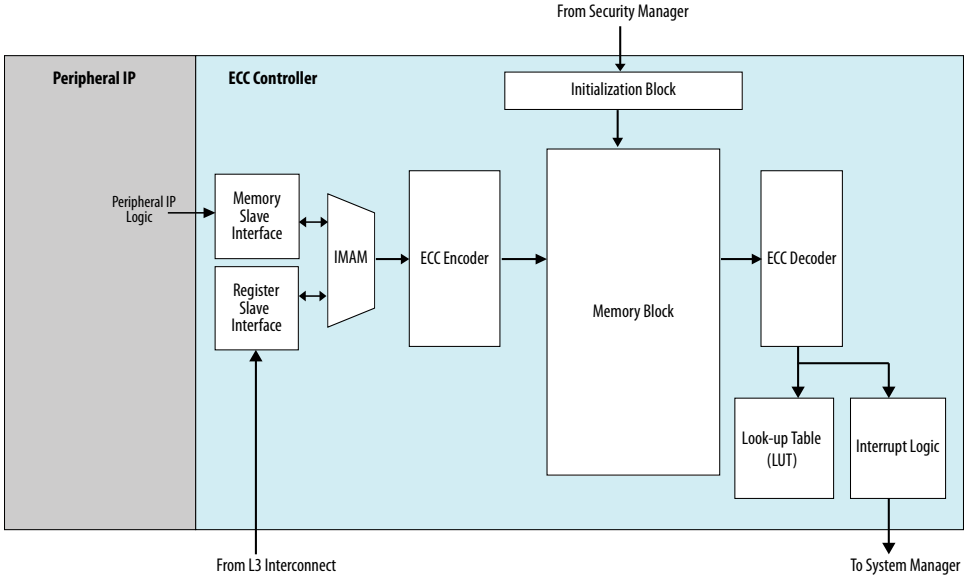
Cortex-A9 Microprocessor Unit Subsystem on page 282
For more information regarding L2 Cache ECC support.

12.3. ECC Controller Block Diagram and System Integration

The figure below shows the ECC controller components and the ECC controller communication with other HPS peripherals.

Figure 50. ECC Block Diagram and System Integration

This figure applies to any of the peripheral IP that have ECC-supported memories.



Each peripheral accesses its memory block through the memory slave interface. The register slave interface allows the Microprocessor Unit (MPU) subsystem to access registers in the ECC controller for software configuration of the ECC controller. The register slave interface also allows the MPU subsystem to indirectly access the memory block through the indirect memory access MUX (IMAM).

Before the peripheral writes data to its memory block, it is encoded in the ECC controller. Before memory sends read data to a peripheral, it is decoded by the ECC controller. The initialization block initializes the memory data content, as well as the ECC syndrome bits, to known values. This block is controlled by the register slave interface and also by the Security Manager when memory clearing is required during a tamper event.

When enabled, the look-up table (LUT) records the memory address of all single-bit errors, allowing you to analyze the error rate history.

The interrupt logic provides interrupt capability for single- and double-bit errors.

Related Information

Indirect Memory Access on page 350

12.4. ECC Controller Functional Description

12.4.1. Overview

An ECC controller can be enabled or disabled by programming the ECC Control (CTRL) register. The controller is disabled by default when the HPS is released from reset. When the ECC controller is disabled, data written to the memory block is not encoded, and data read from the memory block does not require ECC decoding. When the ECC controller is enabled, single-bit errors can be detected and corrected by the ECC controller. Double-bit errors are detected but not corrected.

12.4.2. ECC Structure

The ECC is calculated based on a Hamming code for the corresponding data word length.

Table 114. ECC Bits Required Based on Data Width

Data Bus Width	ECC Bits
8 to 15 bits	5
16 to 31 bits	6
32 to 63 bits	7
64 to 127 bits	8
128 to 255 bits	9
256 bits	10

Table 115. ECC Memory Characteristics

This table shows the memory data size and the Hamming code word length for each of the ECC-protected memories in the HPS, as well as the memory type. The Hamming code word length is calculated based on the full data width and whether the memory is byte- or word- addressable.

Notice that the on-chip RAM and DMA are byte-addressable. For each byte of data, five syndrome bits are used. For a data size of 64 bits (8 bytes), a total of `8 bytes*(8-bit data + 5-bit ECC)` bits are used for a Hamming code word.

Peripheral Memory	Data Size	Memory	ECC Bits	Data Width + ECC Bits	Hamming Code Word (length in bits)	Type [23]
On-chip RAM	64 x 32768	Byte-addressable	5 per byte lane	64+40[24]	104	Single port
USB RAM	35 x 8192	Word-addressable	7	35+7	42	Single port
SD/MMC FIFO	32 x 1024	Word-addressable	7	32+7	39	True dual port
EMAC Rx FIFO	35 x 4096	Word-addressable	7	35+7	42	Simple dual port
EMAC Tx FIFO	35 x 1024	Word-addressable	7	35+7	42	Simple dual port
DMA FIFO	64 x 512	Byte-addressable	5 per byte lane	64+40[24]	104	Simple dual port
NAND ECC Buffer	16 x 768	Word-addressable	6	16+6	22	Simple dual port
NAND Write FIFO	32 x 128	Word-addressable	7	32+7	39	Simple dual port
NAND Read FIFO	32 x 32	Word-addressable	7	32+7	39	Simple dual port
QSPI RAM	32 x 128	Word-addressable	7	32+7	39	Single port

12.4.2.1. RAM and ECC Memory Organization Example

The on-chip RAM and DMA have a memory organization that is byte-writeable, where every byte of data requires 5 bits of ECC.

The tables below shows the memory organization of the byte-writable memory with a 64-bit data size and 5 bits of ECC data.

Table 116. Organization of Byte-Writeable Memory with 64-bit Data Size

Address	RAM Bits							
	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]
0x0	data[7]	data[6]	data[5]	data[4]	data[3]	data[2]	data[1]	data[0]
0x8	data[15]	data[14]	data[13]	data[12]	data[11]	data[10]	data[9]	data[8]

(23) True dual-port memory has two writeable and two readable ports. Simple dual port memory has one write-only port and one read-only port.

(24) This is the same as 8 byte lanes with 5 ECC bits per lane.

Table 117. Memory Organization of 5-Bit ECC Data

Address	ECC Memory Bits							
	[31:29]	[28:24]	[23:21]	[20:16]	[15:13]	[12:8]	[7:5]	[4:0]
0x0	0x0	ecc_data[3]	0x0	ecc_data[2]	0x0	ecc_data[1]	0x0	ecc_data[0]
0x4	0x0	ecc_data[7]	0x0	ecc_data[6]	0x0	ecc_data[5]	0x0	ecc_data[4]
0x8	0x0	ecc_data[11]	0x0	ecc_data[10]	0x0	ecc_data[9]	0x0	ecc_data[8]
0xC	0x0	ecc_data[15]	0x0	ecc_data[14]	0x0	ecc_data[13]	0x0	ecc_data[12]

12.4.3. Memory Data Initialization

When an ECC controller is enabled, the memory data must be written first before any data read occurs. If the memory is not written, the ECC syndrome bits are random, potentially causing false single- or double-bit errors when the memory data is read.

Every byte of data in the RAM is protected with ECC. This protection can lead to spurious ECC errors under the following conditions:

- When the MPU prefetches any uninitialized locations.
- When the MPU (or any master) reads from an uninitialized byte.

To prevent spurious ECC errors, software must use the memory initialization block in the ECC controller to initialize the entire memory data and ECC bits. The initialization block clears the memory data. Enabling initialization in the ECC Control (CTRL) register is independent of enabling the ECC.

Note: Peripherals with true dual-port memories, such as SD/MMC, must initialize both memories explicitly.

Software controls initialization through the ECC Control (CTRL) register. This process cannot be interrupted or stopped after it starts, therefore software must wait for the initialization complete (INITCOMPLETE*) bit to be set in the Initialization Status (INITSTAT) register. Memory accesses are allowed after the initialization process is complete.

The initialization block is accessed through the register slave interface. Additionally, the Security Manager can access the initialization block. If there is an unauthorized access to the memory or a tamper attempt, the Security Manager signals the Reset Manager to begin scrambling the on-chip memories through the initialization block. After scrambling, the initialization block clears the memory data to zero. This security feature is useful in applications that require secure boot, configuration and authentication.

The Security Manager can also be configured to scramble and clear memory on a cold or warm reset, or both a cold and warm reset. This feature is configured through a combination of user fuses and configuration registers within the Security Manager.

In the Security Manager, you can also select whether the memory is cleared in series or parallel. If time savings is required, memories can be initialized in parallel. If power integrity and savings are required, memories can be initialized in series.

Related Information

SoC Security on page 168
For more information regarding Security Manager functions, refer to the *SoC Security* chapter.

12.4.4. Indirect Memory Access

The register slave interface on an ECC controller allows software to access the memory block indirectly.

Through this interface, software can alter the memory data content and the stored ECC syndrome bits. By directly altering the data and syndrome bits, software can manually correct corrupted data, and run tests and diagnostics on the ECC controller.

Accesses to syndrome bits are not supported by the conventional memory access through the memory slave interface and instead, the ECC encoder handles them automatically.

12.4.4.1. Watchdog Timer

To prevent stalled indirect memory accesses, each ECC controller has a watchdog timer.

For example, if the clock to the memory block is stopped, the watchdog timer can assert an interrupt indicating that memory failed to respond within the expected interval. The watchdog timer is in a separate clock domain from the memory, enabling it to continue running independently of any problem with the memory clock.

The watchdog timer can be enabled or disabled in the ECC Watchdog Control (`ECC_wdcrtl`) register. The watchdog timeout is 2048 clock cycles of the clock domain that is connected to the ECC control slave port. The watchdog timeout interval is not software-programmable.

12.4.4.2. Data Correction

The data in the memory block can be overwritten through an indirect memory access. This feature is particularly useful when a double-bit error is detected on the data. The ECC controller provides the memory address of the current double-bit error. The data can be corrected by writing to the given memory address using an indirect memory access.

12.4.4.3. Error Injection

The ECC controller allows you to explicitly inject errors into the memory block for testing purposes. You can alter the memory data or ECC syndrome bits to manually introduce errors. If you change one bit of the memory data to the opposite value, a single-bit error is detected and corrected by the ECC controller. You can also alter ECC syndrome bits to test if the ECC controller triggers an error detection signal as expected.

12.4.4.4. Memory Testing

Testing methods can include both direct and indirect methods of memory access through the peripheral slave interface and the register slave interface. You can use both paths to test the syndrome bits and the single-bit and double-bit error logic. However, you can only control ECC syndrome bits by injecting them through the register slave interface.

12.4.4.4.1. Peripheral Slave Interface Tests

Data and ECC overwrite bits in the ECC_accctrl register are provided to test the functionality of the peripheral interface to the ECC-protected RAM.

ECC-Enabled Test

The following sequence can be used to test if the ECC decoder works correctly.

1. Enable the ECC by setting the ECC_EN bit in the CTRL register.
2. Write data to any ECC-protected RAM memory location. This action generates an ECC value that can be read through the ECC_Rdataecc0bus and ECC_Rdataecc1bus registers.
3. Read back the memory data. The data read back is expected to match the data originally written (with or without a single-bit error) or generate a double-bit error. Refer to the "Error Logging" section for more details about identifying errors.

ECC-Disabled Test

This sequence can be used to test that the ECC decoder does not produce output when disabled.

1. Disable the ECC by clearing the ECC_EN bit in the CTRL register.
2. Write to any ECC-protected RAM memory location. No ECC value is expected to be generated and no interrupt or error logging occurs.
3. The ECC value can be read through the ECC_Rdataecc0bus and ECC_Rdataecc1bus registers to verify that the ECC values do not correspond to the read memory data.

ECC Disable/Enable Test

This sequence shows that memory data written when the ECC controller is disabled generates an error if the ECC controller is subsequently enabled and the same memory data location is read.

1. Disable the ECC by clearing the ECC_EN bit in the CTRL register.
2. Write to any ECC-protected RAM memory location. No ECC value is expected to be generated and no interrupt or error logging occurs.
3. Enable the ECC by setting the ECC_EN bit in the CTRL register.
4. Read the data from ECC-protected RAM memory location you wrote in step 2.
5. Expect an error to be generated because the ECC value corresponding to the memory data is not correct. Refer to the "Error Logging" section for more details about identifying errors.

Related Information

- ECC Controller Address Map and Register Descriptions on page 362

- Error Logging on page 355
  For more information about error logging.

12.4.4.4.2. Register Interface Tests

You can correct memory errors and test the memory register interface through registers in the ECC Controller.

The following registers can be used to test and correct memory:

- ECC_Addrbus: Holds the address of the memory and ECC data.
- ECC_RData3bus through ECC_RData0bus: Holds memory data from a read access.
- ECC_WData3bus through ECC_WData0bus: Holds the data to be written to memory.
- ECC_RDataecc1bus and ECC_RDataecc0bus: Holds the ECC data from a read access.
- ECC_WDataecc1bus and ECC_WDataecc0bus: Holds the ECC data to be written to memory.
- ECC_accctrl: Configures the access as a read or a write and enables memory and ECC data overwrites.
- ECC_startacc: Initiates the register interface access of memory data or ECC data.

Single-Bit Error Test

This sequence tests the single-bit error detection and correction in the ECC decoder.

1. Enable the ECC by setting the ECC_EN bit in the CTRL register.
2. Set the Data Override (DATAOVR) bit in the ECC_accctrl register.
3. Write data to an address location in memory using a normal memory write. Expect the correct ECC data to be generated.
4. Write a data value that has one bit altered in the ECC_WData3bus through ECC_WData0bus registers and write the address of the memory location in the ECC_Addrbus.
5. Configure the ECC_accctrl register to a write and set the ENBUSA bit of the ECC_startacc register to initiate the write. If the memory is dual-ported, an ENBUSB bit could optionally be enabled depending on the port access.
6. Read the same memory location using a normal memory read access. Expect a single-bit error to be logged but the data read to be correct. Refer to the "Error Logging" section for more details about identifying errors.

Double-Bit Error Test

This sequence tests the double-bit error detection in the ECC decoder.

1. Enable the ECC by setting the ECC_EN bit in the CTRL register.
2. Set the Data override (DATAOVR) bit in the ECC_accctrl register.
3. Write data to an address location in memory using a normal memory write. Expect the correct ECC data to be generated.

4. Write a data value that has two bits altered in the ECC_WData3bus through ECC_WData0bus registers and write the address of the memory location in the ECC_Addrbus.
5. Configure the ECC_accctrl register to a write and set the ENBUSA bit of the ECC_startacc register to initiate the write. If the memory is dual-ported, an ENBUSB bit could optionally be enabled depending on the port access.
6. Read the same memory location using a normal memory read access. Expect a double-bit error to be logged with no data correction. Refer to the "Error Logging" section for more details about identifying errors.

Related Information

- ECC Controller Address Map and Register Descriptions on page 362
- Error Logging on page 355
  For more information about error logging.

12.4.4.5. Error Checking and Correction Algorithm

The HPS error checking algorithm is based on an extended Hamming code, which is single-error correcting and double-error detecting (SECDED).

The computation can be understood by a given data (d) and a calculation of the check bits (c) through the equation:
$c = d \times H_d^T$

where H is the parity check matrix, $H = \{H_d, H_c\}$.

If the code word, designated by v and calculated by:
$v = \{d, c\}$

transmits to a noisy channel (for example in a RAM that is subjected to soft errors by cosmic rays) and becomes a contaminated code word, v', we can recover or discover the errors from its syndrome, s, by using the equation:
$s = v' \times H^T$

Errors are indicated when s does not equal 0. The syndrome shows the position of the error in the data.
The following examples show the parity check matrix for different data sizes.

Figure 51. 8-Bit Hamming Matrix

	d0	d1	d2	d3	d4	d5	d6	d7	c0	c1	c2	c3	c4
S0	0	0	0	1	1	1	1	1	1	-	-	-	-
S1	0	1	1	0	0	1	1	1	-	1	-	-	-
S2	1	0	1	1	1	0	0	1	-	-	1	-	-
S3	1	1	1	0	1	0	1	0	-	-	-	1	-
S4	1	1	0	1	0	1	0	0	-	-	-	-	1

Figure 52. 16-bit Hamming Matrix

	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	c0	c1	c2	c3	c4	c5
S0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	-	-	-	-	-
S1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	-	1	-	-	-	-
S2	0	1	1	1	0	0	0	1	0	1	1	1	0	0	0	1	-	-	1	-	-	-
S3	1	0	1	1	0	1	1	0	1	0	0	1	0	0	1	0	-	-	-	1	-	-
S4	1	1	0	1	1	0	1	0	1	0	1	0	0	1	0	0	-	-	-	-	1	-
S5	1	1	1	0	1	1	0	1	0	1	0	0	1	0	0	0	-	-	-	-	-	1

Figure 53. 32-bit Hamming Matrix

	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30	d31	c0	c1	c2	c3	c4	c5	c6
S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-	-	-	-	-	-
S1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	-	1	-	-	-	-	-
S2	0	0	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	-	-	1	-	-	-	-
S3	0	1	1	1	0	0	0	1	1	1	0	0	1	1	1	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	1	0	-	-	-	1	-	-	-
S4	1	0	1	1	0	1	1	0	0	1	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0	1	0	0	0	1	0	0	-	-	-	-	-	1	-	-
S5	1	1	0	1	1	0	1	0	1	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0	-	-	-	-	-	-	1	-
s6	1	1	1	0	1	1	0	1	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	0	0	-	-	-	-	-	-	-	1

Figure 54. 35-bit Hamming Matrix

	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20	d21	d22	d23	d24
S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
S1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
S2	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
S3	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1
S4	1	0	1	1	0	1	1	0	0	1	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0
S5	1	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	1
s6	1	1	1	0	1	1	0	1	0	0	1	1	0	1	0	0	1	0	0	0	1	1	0	1	0

	d25	d26	d27	d28	d29	d30	d31	d32	d33	d34	c0	c1	c2	c3	c4	c5	c6
S0	1	1	1	1	1	1	1	1	1	1	1	-	-	-	-	-	-
S1	0	0	0	0	0	1	1	1	1	1	-	1	-	-	-	-	-
S2	0	1	1	1	1	0	0	0	0	1	-	-	1	-	-	-	-
S3	1	0	0	0	1	0	0	0	1	0	-	-	-	1	-	-	-
S4	1	0	0	1	0	0	0	1	0	0	-	-	-	-	1	-	-
S5	0	0	1	0	0	0	1	0	0	0	-	-	-	-	-	1	-
s6	0	1	0	0	0	1	0	0	0	0	-	-	-	-	-	-	1

Figure 55. 136-bit Hamming Matrix

	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20	d21	d22	d23	d24
S0	1	1	1	0	1	1	0	1	0	0	1	1	0	1	0	0	1	0	0	0	1	1	0	1	0
S1	1	1	0	1	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	0	1	0	1	0	1
S2	1	0	1	1	0	1	1	0	0	1	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0
S3	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1
S4	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
S5	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
S6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
S7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	d25	d26	d27	d28	d29	d30	d31	d32	d33	d34	d35	d36	d37	d38	d39	d40	d41	d42	d43	d44	d45	d46	d47	d48	d49
S0	0	1	0	0	0	1	0	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	0
S1	0	0	1	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	0	0	0	1	0	0	0
S2	1	0	0	1	0	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0
S3	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	1	0	0	0	1	0	0	1	1	0
S4	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1
S5	0	0	0	0	0	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1
S6	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S7	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
S8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	d50	d51	d52	d53	d54	d55	d56	d57	d58	d59	d60	d61	d62	d63	d64	d65	d66	d67	d68	d69	d70	d71	d72	d73	d74
S0	1	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0
S1	0	1	0	0	0	0	1	0	1	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0
S2	0	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	1	0
S3	0	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	0	0	0	1
S4	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	0
S5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0
S6	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
S7	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S8	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	d75	d76	d77	d78	d79	d80	d81	d82	d83	d84	d85	d86	d87	d88	d89	d90	d91	d92	d93	d94	d95	d96	d97	d98	d99
S0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	1	0	1	1
S1	0	0	0	1	0	0	0	0	0	1	1	0	0	0	1	0	0	1	0	1	1	1	1	1	1
S2	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	1	0	1	1	0	1	1	0
S3	0	0	0	0	0	1	0	0	0	1	1	1	1	1	0	1	0	1	1	1	1	0	0	1	1
S4	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	1	0	0
S5	0	1	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0	0	1	0	0	1	0	0	1
S6	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	0	1	0	1
S7	0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	0	1	0	0	0
S8	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1	1	0	1	0	1	1	1	0

	d100	d101	d102	d103	d104	d105	d106	d107	d108	d109	d110	d111	d112	d113	d114	d115	d116	d117	d118	d119	d120	d121	d122	d123	d124
S0	1	0	1	1	1	0	0	0	0	1	1	0	0	1	0	0	1	1	0	1	1	0	0	0	1
S1	0	0	1	1	1	0	0	1	0	1	0	1	0	0	0	1	0	0	1	0	1	1	0	1	1
S2	0	1	1	0	1	1	0	0	1	1	1	1	1	0	1	0	1	0	0	0	1	0	1	0	0
S3	0	1	0	1	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0	1	1	0	0	1	0
S4	1	0	0	1	0	1	1	0	1	1	0	0	1	0	1	1	1	0	1	1	0	0	0	1	0
S5	0	1	0	0	0	1	1	1	1	0	1	1	1	1	0	1	1	0	1	1	0	1	1	0	1
S6	1	1	1	0	1	1	1	0	1	0	0	1	1	0	1	1	1	1	0	0	0	1	1	0	1
S7	1	0	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	1	1	1	1	1	1	0
S8	1	1	0	1	1	1	1	1	0	0	1	0	0	1	1	0	1	1	1	0	0	1	1	1	1

	d125	d126	d127	d128	d129	d130	d131	d132	d133	d134	d135	c0	c1	c2	c3	c4	c5	c6	c7	c8
S0	0	0	1	1	1	1	1	0	1	1	1	1	-	-	-	-	-	-	-	-
S1	1	0	1	1	1	0	1	1	0	1	1	-	1	-	-	-	-	-	-	-
S2	1	1	0	0	1	1	1	0	1	0	0	-	-	1	-	-	-	-	-	-
S3	0	1	0	1	0	1	0	0	0	1	1	-	-	-	1	-	-	-	-	-
S4	1	1	1	1	1	1	1	1	1	0	1	-	-	-	-	1	-	-	-	-
S5	0	0	1	1	0	1	0	1	1	1	0	-	-	-	-	-	1	-	-	-
S6	1	1	1	0	0	0	1	1	0	0	0	-	-	-	-	-	-	1	-	-
S7	1	0	0	0	1	0	0	1	1	0	1	-	-	-	-	-	-	-	1	-
S8	0	1	0	0	0	0	0	0	0	1	0	-	-	-	-	-	-	-	-	1

12.4.5. Error Logging

The ECC controller logs the errors that are detected in the memory block. This logging is useful for diagnosing the error and possibly recovering the correct memory data.

12.4.5.1. Recent Error Address Registers

Each ECC controller logs the most recent single-bit and double-bit error memory addresses.

These address values are stored in the ECC Single-Bit Error Address (SERRADDRx) and Double-Bit Error Address (DERRADDRx) registers and can be read by software. These registers store the most recent memory error address. The logged address is a word address, rather than a byte address value, so that software must shift the address left to obtain the true address corresponding to the address width. For example, a 16-bit address must be shifted left by one to represent the true address. For a 32-bit address the value must be shifted left by two; and for a 64-bit address, the value must be shifted left by three bits.

For a single-bit error, the SERRADDRx register logs the error address only if the single-bit error interrupt generation is enabled. Every double-bit error is logged if the ECC controller is enabled.

For true dual-port memory, two sets of recent error address registers are present. Each register shows the address of the error that has occurred on its corresponding memory port.

Related Information

ECC Structure on page 347
Refer to the "ECC Structure" section for more information regarding ECC Structure and Hamming Code word lengths.

12.4.5.2. Single-Bit Error Occurrence

The ECC controller has a 32-bit wide counter that increments on every occurrence of a single-bit error.

You can program the ECC controller to trigger an interrupt when the single-bit error counter has reached a specific value, which is configured in the Single-Bit Error Count (SERRCNTREG) register. You can reset the counter by clearing the CNT_RSTA bit in the ECC Control (CTRL) register.

For true dual-port memory, such as SD/MMC, two internal single-bit error counters are present in its ECC controller. Each counter counts the errors on its own memory port. However, both counters refer to the same user-configurable threshold for interrupt generation. In this case, program the counter threshold value in the SERRCNTREG register to represent the average number of errors of both memories.

12.4.5.3. Single-Bit Error Look-Up Table

The ECC controller of each memory port has a look-up table (LUT) that logs the memory addresses of all unique single-bit error occurrences. Repeated errors at the same memory address are not stored. The LUT keeps track of single-bit errors, but not double-bit errors.

The most significant bit (MSB) of each entry in the LUT is the valid bit. Whenever a single-bit error occurs and is logged by the LUT, the valid bit is set. The rest of the bits in a single LUT entry contain the memory address of the data error. After the memory address has been read from the LUT, software can clear the entry by writing a 1 to the valid bit in the entry. If all of the LUT entries are occupied and valid bits have not been

cleared, overflow occurs on the next single-bit error. An interrupt can be generated on a LUT overflow. For more information about interrupts, refer to the "ECC Controller Interrupts" section.

The table below lists the LUT depth for every ECC-protected memory in the HPS.

Table 118. LUT Depth for HPS ECC Memories

Peripheral	LUT Depth (entries)
On-chip RAM	16
USB	8
SD/MMC	4 x 2 [25]
Ethernet MAC (Rx FIFO)	4
Ethernet MAC (Tx FIFO)	4
DMA	4
NAND (ECC Buffer)	4
NAND (Write FIFO)	4
NAND (Read FIFO)	4
QSPI	4

The LUT entries are located in the ECC controller register map. Software can read the LUT error address and clear the valid bits. For more information, refer to the *ECC Controller Address Map and Register Description* section.

Related Information

- ECC Controller Interrupts on page 357
  For information about single- and double-bit error interrupts, refer to the "ECC Controller Interrupts" section.
- ECC Controller Address Map and Register Descriptions on page 362

12.4.6. ECC Controller Interrupts

The ECC controller has the ability to generate single- and double-bit error interrupts to the System Manager.

The ECC controller interrupt mechanism involves the System Manager, generic interrupt controller (GIC) and the Arm Cortex-A9 . The following steps outline the interrupt generation process when interrupts have been enabled through the Error Interrupt Enable (ERRINTEN) register.

[25] The ECC controller for SD/MMC peripheral has two LUTs, 4-entries deep, because the memory used for the SD/MMC controller is a true dual-port type, where both ports can perform read operations. Reading either of the ports can trigger a single-bit error and so, a LUT is required for each of the ports.

1. The ECC controller generates an interrupt when an error occurs and notifies the System Manager.
2. The System Manager updates its interrupt status register and sends the interrupt to the GIC.
3. The GIC sends the interrupt to the MPU.
4. The MPU services the interrupt and clears the interrupt in the ECC controller.
5. The System Manager clears the interrupt to the GIC and the corresponding interrupt status bit.

12.4.6.1. Single-Bit Error Interrupts

The Single-Bit Error Interrupt Enable (ERRINTEN) register must be configured for single-bit error interrupt generation.

For true dual port memory, a separate interrupt is generated for errors on each memory port.

The ECC controller can generate a single-bit error interrupt for:

- All single-bit errors
- LUT overflow
- Single-bit error counter match

The address of the most recent single-bit error is logged in the Single-Bit Error Address (SERRADDRx) register.

The interrupt status (INSTAT) register indicates if a single-bit error is pending in the ECC controller. All single-bit interrupts are cleared by clearing the single-bit error pending bit of the INTSTAT register. The single-bit interrupt generation can be disabled by setting the error interrupt reset bit of the Error Interrupt Reset (ERRINTENR) register.

Related Information

ECC Controller Address Map and Register Descriptions on page 362

12.4.6.1.1. All Single-Bit Error Interrupt

To generate an interrupt for every single-bit error that occurs, regardless of whether it is with a new or repeated memory address access, you must:

- Clear the INTMODE bit in the Interrupt Mode (INTMODE) register
- Enable the interrupt by setting the SERRINTEN bit of the Error Interrupt Enable (ERRINTEN) register

This mode generates the most frequent interrupts and therefore, consumes greater processor cycle resources to service all the interrupts.

Note: Overflow data is not logged in this interrupt configuration.

12.4.6.1.2. LUT Overflow Interrupt

The LUT table can be used to generate two types of interrupts.

On every single-bit error detection and correction, the address of the error is logged in the LUT. Each address logged is unique and is at the data word boundary of its RAM bank. Coherency of the address table is maintained by a valid bit.

An interrupt can be generated for each new LUT entry or only when the LUT overflows. The following table describes the interrupt result based on the INTMODE and INTONOVF values in the INTMODE register. In this table, it is assumed that interrupts have been enabled by setting the SERRINTEN bit of the Error Interrupt Enable (ERRINTEN) register.

Note: If the INTMODE bit is clear, then all errors generate an interrupt and no overflow data is logged. The INTMODE bit must be set to 1 for the LUT to log entries.

Table 119. LUT Overflow Interrupt Configuration Options

INTMODE value	INTONOVF value	Result
0	X = Don't care	All errors generate an interrupt. No overflow data is logged.
1	0	An interrupt is generated for each new LUT entry. Overflow detection is disabled. Example: For a four-entry LUT, an interrupt asserts for each unique address entered in the LUT.
1	1	An interrupt is generated only when the LUT overflows. Example: If the LUT depth is four, the occurrence of the fifth unique address causes an interrupt to assert.

12.4.6.1.3. Counter Match Interrupt

The counter match interrupt allows you to set a threshold for the number of single-bit errors captured before an interrupt flag is set.

The INTONCMP bit in the INTMODE register enables the internal counter to count and compare against the SERRCNT value in the Single-Bit Error Count (SERRCNTREG) register. The internal counter increments on every single-bit error, regardless of whether it is a new or repeated address. The INTONCMP bit has no influence on the INTMODE and INTONOVF bits of the INTMODE register. As long as the internal counter is less than the Single-Bit Error Count (SERRCNTREG) register value, no interrupt is generated. When the internal counter is greater than or equal to the SERRCNTREG value, a single-bit interrupt request is asserted, the CMPFLGx bit is set in the Mode Status (MODSTAT) register, and the SERRPENx bit is set in the Interrupt Status (INTSTAT) register. When the match occurs, additional errors do not increment the counter until the CMPFLGx bit is cleared in the MODSTAT register.

This resultant match can be handled in three ways:

- Reset the error counter without restarting it. The ECC controller does not count single-bit errors until you restart the counter. Set the CNT_RSTx bit in the CTRL register to 1, which clears the counter. The CMPFLGx bit remains set. The counter does not increment until the CMPFLGx bit is cleared.
- Reset and restart the counter and clear the compare flag. Set CNT_RSTx bit in the CTRL register to 1, which clears the counter. Write a 1 to the CMPFLGx bit, which clears it. The internal counter begins counting from zero.
- Set the count to a higher value and clear the compare flag. Write the SERRCNTREG value to a higher value than the initial compare match value. Write a 1 to the CMPFLGx bit. This clears the CMPFLGx bit, but the internal counter is not reset and the count continues from where it left off until it reaches the new SERRCNTREG value.

If you allow the counter to resume during your interrupt service routine (ISR), it is possible that the error counter could run out again before the ISR exits. If this happens, and you clear the interrupt and exit the ISR, you can never detect the new counter match condition. To avoid this problem, check the CMPFLGx bit in the MODSTAT register prior to exiting the ISR. If CMPFLGx indicates another counter match condition, ensure that you handle it.

To clear the single-bit error interrupt, set the SERRPENx bit in the INTSTAT register.

Related Information

ECC Controller Address Map and Register Descriptions on page 362

12.4.6.2. Double-Bit Error Interrupt

All double-bit errors generate interrupts and the error memory address is logged into the recent double-bit error (DERRADDRx) register.

The Interrupt Status (INTSTAT) register indicates if a double-bit error has occurred. The double-bit error interrupt generation cannot be disabled. The interrupt is de-asserted by writing to the double-bit error pending bit of the INTSTAT register.

Related Information

ECC Controller Address Map and Register Descriptions on page 362

12.4.6.3. Interrupt Testing

The ECC controller allows you to test the interrupt assertion and de-assertion to check if the interrupt logic is functioning properly. Set the single-or double-bit error test bits in the Interrupt Test (INTTEST) register to assert the corresponding interrupt. To clear the interrupt, set the single- or double-bit error pending bits in the Interrupt Status (INTSTAT) register.

12.4.7. ECC Controller Initialization and Configuration

The steps for initializing and configuring an ECC controller are as follows:

1. Turn off ECC interrupts by setting interrupt masks in the `ecc_intmask_set` register in the System Manager and disabling interrupts in the `ERRINTEN` register of the ECC Controller.
2. Ensure the ECC detection and correction logic is disabled by clearing the `ECC_EN` bit in the `CTRL` register.
3. Enable memory initialization through the ECC controller's memory initialization block by setting the `INITx` bit in the `CTRL` register. If the memory is dual-ported, initialization must be performed on both ports. Refer to the *ECC Structure* section to identify what type of memory you are initializing.
4. When the `INITCOMPLETEx` bit in the `INITSTAT` register is set, configure any single-bit, count, or compare match interrupts that are required. Enable ECC interrupts in the ECC controller and System Manager. Refer to the *ECC Controller Interrupts* section and the System Manager chapter for information on enabling interrupts.
5. (Optional) Test the ECC to see if it is functioning as expected. You can perform tests with or without the ECC controller enabled.

   Software can write to the control, data, and address registers provided in the ECC controller and then set the `ENBUSx` bit of the `ECC_startacc` register to initiate memory accesses. Alternatively, software can enable the ECC controller by setting the `ECC_EN` bit in the `CTRL` register and test the memory slave interface. Please refer to the *Memory Testing* section for more information on how to test the ECC controller.

   If optional testing is completed, clear the test memory.

After these steps are complete, normal accesses can occur.

When an ECC controller is enabled:

- The ECC controller writes the ECC bits whenever data is written to the RAM.
- Error interrupt requests can be enabled in the Interrupt Mode (`INTMODE`) register.
- Data errors are detected and correction is attempted.

The ECC calculation can only be performed when there is a valid RAM access.

Related Information

- Memory Testing on page 351
- ECC Structure on page 347
- ECC Controller Interrupts on page 357
  For details of single-bit, count, and compare-match interrupts.

12.4.8. ECC Controller Clocks

The ECC controller for each ECC-protected memory operates at the same clock frequency as its associated RAM port.

The ECC register interface, however, is in the `l4_mp_clk` domain. The clock source names for each ECC controller and its RAM are determined by the specific peripheral.

Table 120. Clock Source for Each ECC Controller and Memory

ECC Memory	Functional Clock
On-chip RAM	l3_main_free_clk
USB	asynchronous l4_mp_clk
SD/MMC	Port A read and write: cclk_in
	Port B read and write: l4_mp_clk
Ethernet MAC (Rx FIFO)	Read: ap_clk
	Write: clk_rx_int
Ethernet MAC (Tx FIFO)	Read: ap_clk
	Write: clk_tx_int
DMA	l4_main_clk
NAND (ECC Buffer)	nand_clk
NAND (Write FIFO)	Write: nand_x_clk
	Read: nand_clk
NAND (Read FIFO)	Read: nand_x_clk
	Write: nand_clk
QSPI	l4_mp_clk

Related Information

Clock Manager on page 52
For details of the clock signals for each ECC controller.

12.4.9. ECC Controller Reset

The Reset Manager drives the reset signal to the ECC controllers on a cold or warm reset. This resets the logic in the ECC controllers. The Security Manager can also request the Reset Manager to send a signal to scramble and clear all of the memories if a security event occurs.

Related Information

- Reset Manager on page 134
  For more information regarding reset, refer to the *Reset Manager* chapter.
- SoC Security on page 168
  For more information regarding Security Manager functions, refer to the *SoC Security* chapter.

12.5. ECC Controller Address Map and Register Descriptions

Each peripheral with ECC-supported memory has its own ECC Controller address space and registers. This section lists the individual peripheral's ECC Controller address maps and registers but this information can also be found within the peripheral chapters of this technical reference manual.

Related Information

- Ethernet Media Access Controller on page 520
  For more information regarding the EMAC controller.
- NAND Flash Controller on page 370
  For more information regarding the NAND Flash Controller.
- SD/MMC Controller on page 405
  For more information regarding the SD/MMC Controller.
- On-Chip Memory on page 364
  For more information regarding On-Chip RAM.
- DMA Controller on page 512
  For more information regarding the DMA Controller.
- Quad SPI Flash Controller on page 490
  For more information regarding the Quad SPI Flash Controller.
- USB 2.0 OTG Controller on page 591
  For more information regarding the USB 2.0 OTG Controller.

13. On-Chip Memory

The hard processor system (HPS) contains two types of on-chip memory. The on-chip memory types are:

- On-chip RAM—The on-chip RAM provides 256 KB of general-purpose single-port RAM
- Boot ROM—The boot ROM provides 128 KB and is available to store the code required to boot the HPS from cold or warm reset

Both on-chip memories connect to the level 3 (L3) interconnect.

Related Information

- On-Chip RAM on page 364
- Boot ROM on page 367
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

13.1. On-Chip RAM

13.1.1. Features of the On-Chip RAM

The on-chip RAM offers the following features:

- 64-bit slave interface
- 256 KB of single-ported RAM
- Error correction code (ECC) support
- Capable of data transfers for every clock cycle
- Security firewall
- Supports scrambling and reset of memory on detection of a tamper event

Related Information

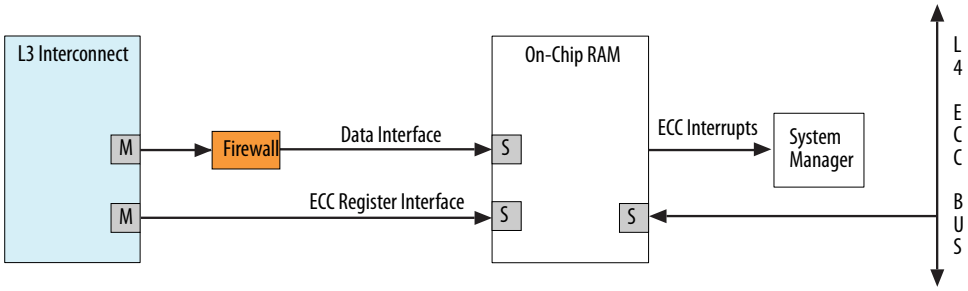
Clock Manager on page 52

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

13.1.2. On-Chip RAM Block Diagram and System Integration

Figure 56. On-Chip RAM Block Diagram



The on-chip RAM and L3 interconnect transfers data through a 64-bit interface that passes through a firewall, operating at the `l3_main_free_clk` interconnect clock frequency. For memory, read acceptance is two, write acceptance is two, and total acceptance is two with a round-robin arbitration.

Related Information

- Error Checking and Correction Controller on page 345
- SoC Security on page 168

13.1.3. Functional Description of the On-Chip RAM

The on-chip RAM uses a 64-bit slave interface. The slave interface supports transfers between memory and the L3 interconnect. All reads and writes are serviced in order.

The on-chip RAM has an exclusive monitor, as well. To ensure mutually exclusive access to shared data, use the exclusive access support built into the on-chip RAM. The on-chip RAM contains five monitors. Each monitor can be used by any of the following:

- CPU and CPU1
- FPGA2SOC
- FPGA2SDRAM0
- FPGA2SDRAM1
- FPGA2SDRAM2

Note: The on-chip RAM exclusive monitor for the MPU does not differentiate between individual CPUs.

Related Information

Clock Manager on page 52

13.1.3.1. On-Chip RAM Clocks

The on-chip RAM is driven by the `l3_main_clk` interconnect clock.

There are two clocks to the on-chip RAM that must be enabled. The first clock is for the memory bus and the second clock is for the ECC register interface. The ECC register interface clock is the `l4_mp_clk`.

13.1.3.2. On-Chip RAM Resets

The contents of the RAM remain unchanged on a cold or warm reset. Reset only clears the state associated with the slave interface.

However, when a reset is initiated because the Security Manager receives a tamper detection assertion, the memory is scrambled and cleared.

Related Information

- Error Checking and Correction Controller on page 345
- SoC Security on page 168

13.1.3.3. On-Chip RAM Initialization

You must initialize the on-chip RAM before you enable the ECC. Failure to do so triggers spurious interrupts.

The on-chip RAM can be initialized by setting the `INITA` bit in the ECC Control Register (CTRL). After the Initialization Status (`INITSTAT`) register indicates the initialization has completed, the ECC Controller can be enabled and ECC interrupts can be configured.

Related Information

Error Checking and Correction Controller on page 345
For more information about ECC, refer to the *Error Checking and Correction Controller* chapter of the Arria 10 Device Handbook.

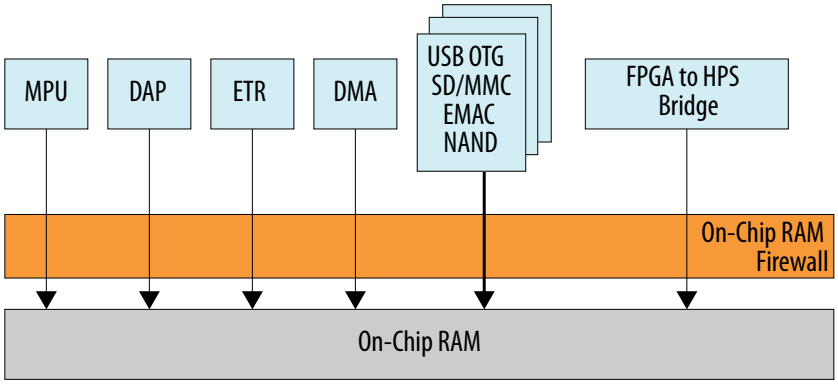
13.1.3.4. On-Chip RAM Firewall

At reset all memories are secure. Out of reset, regions of memories can be configured for non-secure accesses.

The on-chip RAM is divided into six regions of 4 KB. Within each region, a non-secure or shared memory region can be assigned by programming a `base` and `limit` value in the corresponding `regionNaddr` register, with `N` denoting the region number. Each of these regions can be enabled by writing to the `enable` register or setting the corresponding bit in the `enable_set` register.

When an incoming transaction falls within any enabled non-secure regions, the firewall allows both secure and non-secure packets. When the transaction is outside of any enabled regions, the firewall only allows secure packets.

Figure 57. On-Chip RAM Firewall



13.1.3.5. ECC Protection

The ECC controller operation and functionality is programmable through the ECC register slave interface, as shown in the Figure 56 on page 365 in the "On-Chip RAM Block Diagram and System Integration" section. The ECC controller's register interface provides host access to configure the ECC logic as well as inject bit errors into the memory for testing purposes. It also provides host access to memory initialization hardware used to clear the memory contents, including the ECC bits. The ECC controller generates interrupts upon occurrences of single- and double-bit errors, and the interrupt signals are connected to the system manager.

Related Information

- Error Checking and Correction Controller on page 345
- SoC Security on page 168

13.2. Boot ROM

13.2.1. Features of the Boot ROM

The boot ROM offers the following features:

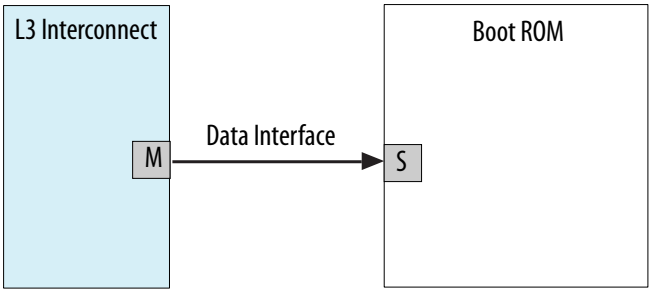
- 32-bit interface
- 128 KB size
- Capable of data transfers for every clock cycle
- Secure initialization

Related Information

Clock Manager on page 52

13.2.2. Boot ROM Block Diagram and System Integration

Figure 58. Block diagram of On-Chip ROM



Boot ROM and L3 interconnect transfers data through a 32-bit data interface that passes through a firewall, operating at the `l3_main_free_clk` interconnect clock frequency. The memory has a read acceptance of one, a write acceptance of one, and a total acceptance of two with round-robin arbitration.

Related Information

SoC Security on page 168

13.2.3. Functional Description of the Boot ROM

The boot ROM is used only for booting the system.

The boot ROM contains the execution code that identifies if there is a secure or non-secure boot, initializes the system to bring it out of reset, finds bootloader and can decrypt or authenticate it based on user fuses.

On a cold or warm reset of the microprocessor unit (MPU) subsystem, MPU0 executes the Preloader code stored in the boot ROM. The Preloader occurs after the initial boot stage, which is the boot ROM. It is loaded by the boot ROM and copied to the on-chip RAM. However, if you choose to boot the Preloader from the FPGA, or if a failure occurred at boot ROM, then as a fallback boot device, the Preloader can then run directly on the FPGA without the need to copy to on-chip RAM.

The boot ROM uses an 32-bit slave interface. The slave interface supports transfers between memory and the L3 interconnect. All writes return an error response.

Related Information

- Clock Manager on page 52
- SoC Security on page 168
- Booting and Configuration on page 771

13.2.3.1. Boot ROM Clocks

The boot ROM is driven by the `l3_main_clk` interconnect clock.

The boot ROM needs to set up the Main PLL and make sure it sources all of the system's clocks. The sequence applied for both Warm and Cold resets is similar. The PLL's are not set up on an FPGA boot, a successful RAM boot, or if the user selects a bypass mode.

Related Information

- Clock Manager on page 52
- SoC Security on page 168

13.2.3.2. Boot ROM Resets

Reset to on-chip ROM does not need to distinguish between warm and cold reset. Therefore, the content of the ROM remains unchanged on a cold or warm reset.

13.3. On-Chip Memory Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

Related Information

Error Checking and Correction Controller on page 345
For more information about how to program the ECC registers, refer to this chapter.

14. NAND Flash Controller

The hard processor system (HPS) provides a NAND flash controller to interface with external NAND flash memory in Intel system-on-a-chip (SoC) systems. You can use external flash memory to store software, or as extra storage capacity for large applications or user data. The HPS NAND flash controller is based on the CadenceDesign IP* NAND Flash Memory Controller.

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

14.1. NAND Flash Controller Features

The NAND flash controller provides the following functionality and features:

- Supports one x8 or x16 NAND flash device

  Note: The HPS supports booting only from a x8 NAND flash device.
- Supports Open NAND Flash Interface (ONFI) 1.0
- Supports NAND flash memories from Hynix, Samsung, Toshiba, Micron, and STMicroelectronics
- Supports error correction codes (ECCs) providing single-bit error correction and double-bit error detection, with:
  - Sector size programmable 512 byte (4-, 8-, or 16-bit correction) or 1024 byte (24-bit correction)
  - Three NAND FIFOs - ECC Buffer, write FIFO and read FIFO
- Supports pipeline read-ahead and write commands for enhanced read and write throughput
- Supports devices with 32, 64, 128, 256, 384, or 512 pages per block
- Supports multiplane devices
- Supports page sizes of 512 bytes, 2 kilobytes (KB), 4 KB, or 8 KB
- Supports single layer cell (SLC) and multiple layer cell (MLC) devices with programmable correction capabilities
- Provides internal direct memory access (DMA)
- Provides programmable access timing

Related Information

Supported Flash Devices for Arria 10 SoC
For more information, refer to the supported NAND flash devices section on this page.

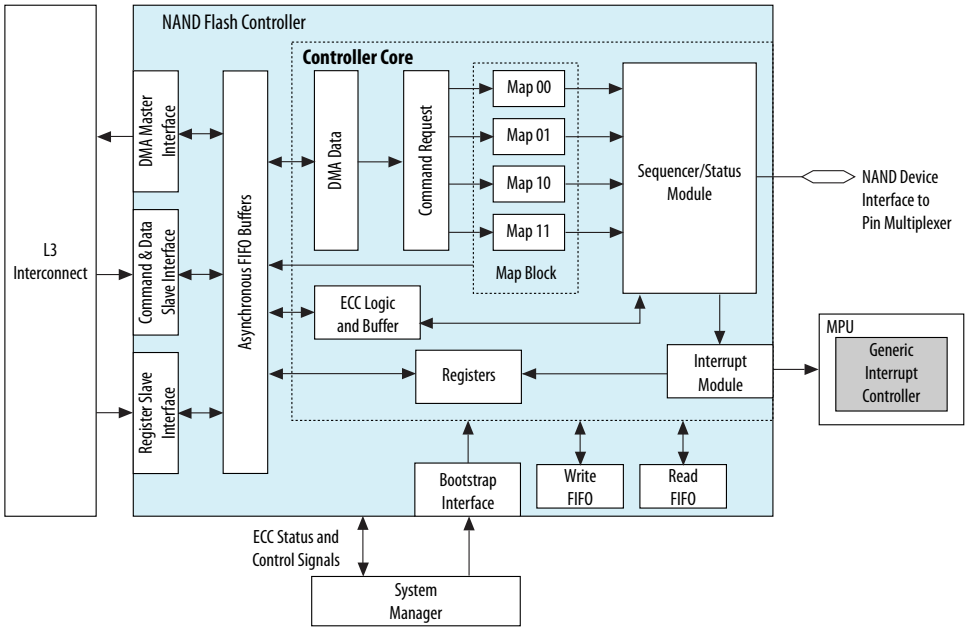
Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

14.2. NAND Flash Controller Block Diagram and System Integration

Figure 59. NAND Flash Controller Block Diagram

The following figure shows integration of the NAND flash controller in the HPS.



Features of the flash controller:

- Receives commands and data from the host through memory-mapped control and data registers connected to the command and data slave interface
- The host accesses the flash controller's control and status registers (CSRs) through the register slave interface.
- Handles all command sequencing and flash device interactions
- Generates interrupts to the HPS Cortex-A9 MPCore
- The DMA master interface provides accesses to and from the flash controller through the controller's built-in DMA.

14.3. NAND Flash Controller Signal Descriptions

All NAND pins have to be from one of the following categories:

- HPS Dedicated
- Shared
- FPGA

The following table lists all NAND Flash Interface signals available to both the HPS and FPGA.

Pins	Supported Data Width	Supported Number of CE and R/B
HPS Dedicated Pins	x8	1
Shared Pins	x8/x16	1
FPGA Pins	x8/x16	1 – 4

The type of pins you use determines the number of Chip Enable (CE) and Ready/Busy (RB) pairs available to you for use. For example, if you use HPS dedicated pins or shared pins, you can only use one CE and R/B pair. If you use FPGA pins, you can use multiple CE and R/B pairs.

Note: The options are mutually exclusive, which means you cannot use HPS dedicated pins, and route the CE and R/B signals to FPGA dedicated pins.

For more information on which signals route to the FPGA and HPS I/O, please refer to the *HPS Component Interface* chapter.

Table 121. NAND Flash Interface Signals

Platform Designer (Standard) Port Name	Connected to FPGA	Connected to HPS I/O	HPS Pin Name
nand_adq_i[15:0]	Yes	Yes	NAND_ADQ[15:0]
nand_adq_oe	Yes	Yes	
nand_adq_o[15:0]	Yes	Yes	
nand_ale_o	Yes	Yes	NAND_ALE
nand_ce_o[3:0]	Yes, 4 chip enables	Yes, 1 chip enable	NAND_CE_N
nand_cle_o	Yes	Yes	NAND_CLE
nand_re_o	Yes	Yes	NAND_RE_N
nand_rdy_busy_i[3:0]	Yes, 4 ready/busy signals	Yes, 1 ready/busy signal	NAND_RB
nand_we_o	Yes	Yes	NAND_WE_N
nand_wp_o	Yes	Yes	NAND_WP_N

Related Information

HPS Component Interfaces on page 737
For more information about NAND Flash Controller signal routing to the FPGA and HPS I/O, refer to this chapter.

14.4. Functional Description of the NAND Flash Controller

This section describes the functionality of the NAND flash controller.

14.4.1. Discovery and Initialization

The NAND flash controller performs a specific initialization sequence after the HPS receives power and the flash device is stable. During initialization, the flash controller queries the flash device and configures itself according to one of the following flash device types:

- ONFI 1.0-compliant devices
- Legacy (non-ONFI) NAND devices

The NAND flash controller identifies ONFI-compliant connected devices using ONFI discovery protocol, by sending the `Read ID` command. For devices that do not recognize this command (especially for 512-byte page size devices), software must write to the system manager to assert the `bootstrap_512B_device` signal to identify the device type before releasing the NAND controller from reset.

To support booting and initialization, the `rdy_busy_in` pin must be connected.

The NAND flash controller performs the following initialization steps:

1. If the system manager is asserting `bootstrap_inhibit_init`, the flash controller goes directly to step 7.
2. When the device is ready, the flash controller sends the "Read ID" command to read the ONFI signature from the memory device, to determine whether an ONFI or a legacy device is connected.
3. If the data returned by the memory device has an ONFI signature, the flash controller then reads the device parameter page. The flash controller stores the relevant device feature information in internal memory control registers, enabling it to correctly program other registers in the flash device, and goes to step 5.
4. If the data does not have a valid ONFI signature, the flash controller assumes that it is a legacy (non-ONFI) device. The flash controller then performs the following steps:
   a. Sends the `reset` command to the device
   b. Reads the device signature information
   c. Stores the relevant values into internal memory controller registers
5. The flash controller resets the memory device. At the same time, it verifies the width of the memory interface. The HPS supports one 8-bit or 16-bit NAND flash device. The flash controller detects the memory interface width.
6. The flash controller sends the `Page Load` command to block 0, page 0 of the device, configuring direct read access, so the processor can boot from that page. The processor can start reading from the first page of the flash memory, which is the expected location of the pre-loader software.

   Note: The system manager can bypass this step by asserting `bootstrap_inhibit_b0p0_load` before reset is de-asserted.
7. The flash controller sends the `reset` command to the flash.
8. The flash controller clears the `rst_comp` bit in the `intr_status0` register in the `status` group to indicate to software that the flash reset is complete.

14.4.2. Bootstrap Interface

The NAND flash controller provides a bootstrap interface that allows software to override the default behavior of the flash controller. The bootstrap interface contains four bits, which when set appropriately, allows the flash controller to skip the initialization phase and begin loading from flash memory immediately after reset. These bits are driven by software through the system manager. They are sampled by the NAND flash controller when the controller is released from reset.

Related Information

System Manager on page 159
For more information about the bootstrap interface control bits.

14.4.2.1. Bootstrap Setting Bits

The following table lists the relevant bootstrap setting bits, found in the system manager's NAND flash controller register group. As an example, this table also lists recommended bootstrap settings for a 512-byte page device.

Table 122. Bootstrap Setting Bits

Bit	Example Value for 512-Byte Page
noinit	1[26]
page512	1
noloadb0p0	1
tworowaddr	• 1—flash device supports two-cycle addressing • 0—flash device support three-cycle addressing

Related Information

Configuration by Host on page 374

14.4.3. Configuration by Host

If the system manager sets bootstrap_inhibit_init to 1, the NAND flash controller does not perform the process described in "Discovery and Initialization". In this case, the host processor must configure the flash controller.

When performance is not a concern in the design, the timing registers can be left unprogrammed.

Related Information

- Bootstrap Setting Bits on page 374
  For recommended configuration-by-host settings to enable the basic read, write, and erase operations for a single-plane, 512 bytes/page device.
- Discovery and Initialization on page 372

14.4.3.1. Recommended Bootstrap Settings for 512-Byte Page Device

Table 123. Recommended Bootstrap Settings for an 8-bit, 512-Byte Page Device

Register	Value
devices_connected	1
device_width	0 indicating an 8-bit NAND flash device
number_of_planes	1 indicating a single-plane device

continued...

(26) When this register is set, the NAND flash controller expects the host to program the related device parameter registers. For more information, refer to "Configuration by Host".

Register	Value
device_main_area_size	The value of this register must reflect the flash device's page main area size.
device_spare_area_size	The value of this register must reflect the flash device's page spare area size.
pages_per_block	The value of this register must reflect number of pages per block in the flash device.

14.4.3.2. NAND Page Main and Spare Areas

Each NAND page has a main area and a spare area. The main area is intended for data storage. The spare area is intended for ECC and maintenance data, such as wear leveling information. Each block consists of a group of pages.

The sizes of the main and spare areas, and the number of blocks in a page, depend on the specific NAND device connected to the NAND flash controller. Therefore, the device-dependent registers, `device_main_area_size`, `device_spare_area_size`, and `pages_per_block`, must be programmed to match the characteristics of the device.

If your software does not perform the discovery and initialization sequence, the software must include an alternative method to determine the correct value of the device-dependent registers. The HPS boot ROM code enables discovery and initialization by default (that is, `bootstrap_inhibit_init = 0`).

14.4.4. Local Memory Buffer

The NAND flash controller has three local SRAM memory buffers.

- The write FIFO buffer is a 128 × 32-bit memory (512 total bytes)
- The read FIFO buffer is a 32 × 32-bit memory (128 total bytes)
- The ECC buffer is a 96 × 16-bit memory (1536 total bytes)

Each of these memories is protected by ECC, and by interrupts for single and double-bit errors. The ECC block is integrated around a memory wrapper. It provides outputs to notify the system manager when single-bit correctable errors are detected (and corrected) and when double-bit uncorrectable errors are detected. The ECC logic also allows injection of single- and double-bit errors for test purposes. It must be initialized to enable the ECC function.

Related Information

Error Checking and Correction Controller on page 345
For more information about ECC, refer to the *Error Checking and Correction Controller* chapter of the Arria 10 Device Handbook.

14.4.5. Clocks

The NAND clock runs synchronously from the NOC clocks, which is always active. To minimize the number of clocks, Clock Manager outputs software managed enables to the USB, SPI Masters, QSPI and NAND peripherals. The software enable for NAND is `nand_clk_en` and is set to ENABLE by default. Also, in Boot Mode, `nand_clk_en` is active to ensure that all clocks are active if RAM is cleared for security.

Table 124. Clock Inputs to NAND Flash Controller

Clock Signal	Description
nand_x_clk	Clock for master and slave interfaces and the ECC sector buffer.
nand_clk	Clock for the NAND flash controller.

The frequency of nand_x_clk is four times the frequency of nand_clk.

14.4.5.1. Clock Generation

The clock manager sends the top level clock from the HPS.

The clock manager sends the 200 MHz clock, l4_mp_clk, to the NAND Flash Controller. This clock becomes the NAND reference clock called nand_mp_clk. The nand_mp_clk is divided by four and is used for input and output. Since the NAND places a 200 MHz limit on the clock, each of these four generated clocks are 50 MHz and called nand_clk.

14.4.5.2. Clock Enable

The nand_mp_clk and nand_clk clocks have enables.

14.4.5.3. Clock Switching

When you use clock switching, you must follow the following requirements:

- Ensure that there is no activity.
- Software must disable this module during the frequency switch and re-enable it after the frequency has changed.
- When clock switching is complete, the software must reconfigure the NAND initialization registers according to the new frequency before triggering any new transactions onto the flash interface.

14.4.6. Resets

When a tamper is detected, the reset manager sends a special signal to the NAND data RAM. This scrambles and clears the RAM. In addition, the RAM can be cleared during a cold or warm reset, depending on how fuse bits are programmed in the device.

Before the NAND flash controller comes out of the reset state, the pin multiplexers for the flash external interface must be configured.

Related Information

Reset Manager on page 134

14.4.6.1. Taking the NAND Flash Controller Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

Related Information

- Module Reset Signals on page 140
- Modules Requiring Software Deassert on page 144

14.4.7. Indexed Addressing

The NAND flash controller uses indexed addressing to reduce the address span consumed by the flash controller.

Indirect addressing is implemented by two registers, accessed through the `nanddata` region, as described in the "Register Map for Indexed Addressing" section.

14.4.7.1. Register Map for Indexed Addressing

Indexed addressing uses registers in the nanddata region of the HPS memory map. The nanddata region consists of a control register and a variable-size register that allows direct access to flash memory, as detailed in the following table.

Table 125. Register Map for Indexed Addressing

Register Name	Offset Address	Usage
`Control`	0x0	Identifies the page of flash memory to be read or written. Software writes the 32-bit control information consisting of **map** command type, block, and page address. The upper four bits must be set to 0. For specific usage of the `Control` register, refer to "Command Mapping".
`Data`	0x10	The `Data` register is a page-size window into the NAND flash. By reading from or writing to locations starting at this offset, the software reads directly from or writes directly to the page and block of NAND flash memory specified by the `Control` register. The Data register is always addressed on 32-bit word boundaries, although the physical flash device has an 8-bit-wide data path.

Related Information

- Command Mapping on page 378
- HPS Peripheral Region Address Map on page 48

14.4.7.2. Indexed Addressing Host Usage

The host uses indexed addressing as follows:

1. Program the 32-bit index-address field into the `Control` register in the `nanddata` region. This action provides the flash address parameters to the NAND flash controller.
2. Perform a 32-bit read or write in the `Data` register.
3. Perform additional 32-bit reads and writes if they are in the same page and block in flash memory.

It is unnecessary to write to the control register for every data transfer if a group of data transfers targets the same page and block address. For example, you can write the control register at the beginning of a page with the block and page address, and then read or write the entire page by directing consecutive transactions to the `Data` register.

14.4.8. Command Mapping

The NAND flash controller supports several flash controller-specific MAP commands, providing an abstraction level for programming a NAND flash device. By using the MAP commands, you can avoid directly programming device-specific commands. Using this abstraction layer provides enhanced performance. Commands take multiple cycles to send off-chip. The MAP commands let you initiate commands and let the flash controller sequence them off-chip to the NAND device.

The NAND flash controller supports the following flash controller-specific MAP commands:

- MAP00 commands—boot-read or buffer read/write during read-modify-write operations
- MAP01 commands—memory arrays read/write
- MAP10 commands—NAND flash controller commands
- MAP11 commands—low-level direct access

14.4.8.1. MAP00 Commands

MAP00 commands access a page buffer in the NAND flash device. Addressing always begins at 0x0 and extends to the page size specified by the `device_main_area_size` and `device_spare_area_size` registers in the `config` group. You can use this command to perform a boot read. Use MAP00 commands in read-modify-write (RMW) operations to read or write any word in the buffer. MAP00 commands allow a direct data path to the page buffer in the device.

The host can access the page buffer directly using the MAP00 commands only if there are no other MAP01 or MAP10 commands active on the NAND flash controller.

14.4.8.1.1. MAP00 Command Format

Table 126. MAP00 Command Format with Address Mapping

The following table shows the format of a MAP00 command. This command is written to the Command register in the nanddata region.

Address Bits	Name	Description
31:28	(reserved)	Set to 0
27:26	CMD_MAP	Set to 0
25:13	(reserved)	Set to 0
12:2	BUFF_ADDR	Data width-aligned buffer address on the memory device. Maximum page access is 8 KB.
1:0	(reserved)	Set to 0

14.4.8.1.2. MAP00 Usage Limitations

The usage of these commands under normal operations is limited to the following situations:

- They can be used to perform an Execute-in-Place (XIP) boot from the device; reading directly from the page buffer while booting directly from the device.
- MAP00 commands can be used to perform RMW operations where MAP00 writes are used to modify a read page in the device page buffer. Because the NAND flash controller does not perform ECC correction during such an operation, Intel does not recommend this method in an MLC device.
- In association with MAP11 commands, MAP00 commands provide a way for the host to directly access the device bypassing the hardware abstractions provided by NAND flash controller with MAP01 and MAP10 commands. This method is also used for debugging, or for issuing an operation that the flash controller might not support with MAP01 or MAP10 commands.

Restrictions:

- MAP00 commands cannot be used with MAP01 commands to read part of a page. Accesses using MAP01 commands must perform a complete page transfer.
- No ECC is performed during a MAP00 data access.
- DMA must be disabled (the flag bit of the dma_enable register in the dma group must be set to 0) while performing MAP00 operations.

14.4.8.2. MAP01 Commands

MAP01 commands transfer complete pages between the host memory and a specific page of the NAND flash device. Because the MAP01 commands support only page addresses, the entire page must be read or written at once. The actual number of commands required depends on the size of the data transfer. The Command register points to the first page and block in the transfer. You do not change the Command register when you initiate subsequent transactions in the transfer, but only when the entire page is transferred.

When the NAND flash controller receives a read command, it issues a load operation on the device, waits for the load to complete, and then returns read data. Read data must be read from the start of the page to the end of the page.

Write data must be written from the start of the page to the end of the page. When the NAND flash controller receives confirmation of the transfer, it issues commands to program the data into the device.

The flash controller ignores the byte enables for read and write commands and transfers the entire data width.

14.4.8.2.1. MAP01 Command Format

Table 127. MAP01 Command Format with Address Mapping

The following table shows the format of a MAP01 command. This command is written to the Command register in the nanddata region.

Address Bits	Name	Description
31:28	(reserved)	Set to 0
27:26	CMD_MAP	Set to 1
25:24	(reserved)	Set to 0
23:<*M*>	BLK_ADDR	Block address in the device
(<*M*>-1):0	PAGE_ADDR	Page address in the device

Note: <*M*> depends on the number of pages per block in the device. <*M*> = ceil(log2(<*device pages per block*>)). Therefore, use the following values:

- 32 pages per block: <*M*>=5
- 64 pages per block: <*M*>=6
- 128 pages per block: <*M*>=7
- 256 pages per block: <*M*>=8
- 384 pages per block: <*M*>=9
- 512 pages per block: <*M*>=9

14.4.8.2.2. MAP01 Usage Limitations

Use the MAP01 command as follows:

- A complete page must be read or written using a MAP01 command. During such transfers, every transaction from the host must have the same block and page address. The NAND flash controller internally keeps track of how much data it reads or writes.
- MAP00 commands cannot be used in between using MAP01 commands for reading or writing a page.
- DMA must be disabled (the flag bit of the dma_enable register in the dma group must be set to 0) while the host is performing MAP01 operations directly. If the host issues MAP01 commands to the NAND flash controller while DMA is enabled, the flash controller discards the request and generates an unsup_cmd interrupt.

14.4.8.3. MAP10 Commands

MAP10 commands provide an interface to the control plane of the NAND flash controller. MAP10 commands control special functions of the flash device, such as erase, lock, unlock, copy back, and page spare area access. Data passed in this command pathway targets the NAND flash controller rather than the flash device. Unlike other command types, the data (input or output) related to these transactions does not affect the contents of the flash device. Rather, this data specifies and performs the exact commands of the flash controller. Only the lower 16 bits of the Data register contain the relevant information.

14.4.8.3.1. MAP10 Command Format

Table 128. MAP10 Command Format with Address Mapping

The following table shows the format of a MAP10 command. This command is written to the Command register in the nanddata region.

Address Bits	Name	Description
31:28	(reserved)	Set to 0
27:26	CMD_MAP	Set to 2
25:24	(reserved)	Set to 0
23:<*M*>	BLK_ADDR	Block address in the device
(<*M*>-1):0	PAGE_ADDR	Page address in the device
Note: <*M*> depends on the number of pages per block in the device, as follows: • 32 pages per block: <*M*>=5 • 64 pages per block: <*M*>=6 • 128 pages per block: <*M*>=7 • 256 pages per block: <*M*>=8 • 384 pages per block: <*M*>=9 • 512 pages per block: <*M*>=9		

14.4.8.3.2. MAP10 Operations

Table 129. MAP10 Operations

Command	Function
0x01	Sets block address for erase and initiates operation
0x10	Sets unlock start address
0x11	Sets unlock end address and initiates unlock
0x21	Initiates a lock of all blocks
0x31	Initiates a lock-tight of all blocks
0x41	Sets up for spare area access
0x42	Sets up for default area access
0x43	Sets up for main+spare area access
0x60	Loads page to the buffer for a RMW operation
0x61	Sets the destination address for the page buffer in RMW operation
0x62	Writes the page buffer for a RMW operation
0x1000	Sets copy source address
0x11<*PP*>	Sets copy destination address and initiates a copy of <*PP*> pages
0x20<*PP*>	Sets up a pipeline read-ahead of <*PP*> pages
0x21<*PP*>	Sets up a pipeline write of <*PP*> pages

14.4.8.3.3. MAP10 Usage Limitations

Use the MAP10 commands as follows:

- MAP10 commands should be used to issue commands to the controller, such as erase, copy-back, lock, or unlock.
- MAP10 pipeline commands should also be used to read or write consecutive multiple pages from the flash device within a device block boundary. The host must first issue a MAP10 pipeline read or write command and then issue MAP01 commands to do the actual data transfers. The MAP10 pipeline read or write command instructs the NAND flash controller to use high-performance commands such as cache or multiplane because the flash controller has knowledge of multiple consecutive pages to be read. The pages must not cross a block boundary. If a block boundary is crossed, the flash controller generates an unsupported command (`unsup_cmd`) interrupt and drops the command.
- Up to four pipeline read or write commands, at the same time, can be issued to the NAND flash controller.
- While the NAND flash controller is performing MAP10 pipeline read or write commands, DMA must be disabled (the `flag` bit of the `dma_enable` register in the `dma` group must be set to 0). DMA must be disabled because the host is directly transferring data from and to the flash device through the flash controller.

14.4.8.4. MAP11 Commands

MAP11 commands provide direct access to the NAND flash controller's address and control cycles, allowing software to issue the commands directly to the flash device using the Command and Data registers. The MAP11 command is useful if the flash device supports a device-specific command not included with standard flash commands. It can also be useful for low-level debugging.

MAP11 commands provide a direct control path to the flash device. These commands execute command, address, and data read and write cycles directly on the NAND device interface. The host can issue only single-beat accesses to the `nanddata` region while using MAP11 commands. The following are the usage requirements:

- Command, address, and write data values are placed in the `Data` register.
- Command and address cycles to the device must be a write transaction on the host bus.
- For data cycles, the type of transaction on the host bus (read/write) determines the data cycle type on the device interface.
- On a read, the returned data also appears in the `Data` register.
- The Control register encodes the control operation type.

14.4.8.4.1. MAP11 Control Format

Table 130. MAP11 Control Format with Address Mapping

The following table shows the format of a MAP11 command. This command is written to the Command register in the nanddata region.

Address Bits	Name	Description
31:28	(reserved)	Set to 0
27:26	CMD_MAP	Set to 3
25:2	(reserved)	Set to 0
1:0	TYPE	Sets the control type as follows: • 0 = Command cycle • 1 = Address cycle • 2 = Data Read/Write Cycle

14.4.8.4.2. MAP11 Usage Limitations

Use the MAP11 commands as follows:

- Use MAP11 commands only in special cases, for debugging or sending device-specific commands that are not supported by the NAND flash controller.
- DMA must be disabled before you use MAP11 operations.
- The host can use only single beat access transfers when using MAP11 commands.

Note: MAP11 commands provide direct, unstructured access to the NAND flash device. Incorrect use can lead to unpredictable behavior.

14.4.9. Data DMA

The DMA transfers data with minimal host involvement. Software initiates data DMA with the MAP10 command.

The flag bit of the dma_enable register in the dma group enables data DMA functionality. Only enable or disable this functionality when there are no active transactions pending in the NAND flash controller. When the DMA is enabled, the flash controller initiates one DMA transfer per MAP10 command over the DMA master interface. When the DMA is disabled, all operations with the flash controller occur through the memory-mapped nanddata region.

The NAND flash controller supports up to four outstanding DMA commands, and ignores additional DMA commands. If software issues more than four outstanding DMA commands, the flash controller issues the unsup_cmd interrupt. On receipt of a DMA command, the flash controller performs command sequencing to transfer the number of pages requested in the DMA command. The DMA master reads or writes page data from the system memory in programmed burst-length chunks. After the DMA command completes, the flash controller issues an interrupt, and starts working on the next queued DMA command.

Pipelining allows the NAND flash controller to optimize its performance while executing back-to-back commands of the same type.

With certain restrictions, non-DMA MAP10 commands can be issued to the NAND flash controller while the flash controller is servicing DMA transactions. MAP00, MAP01, and MAP11 commands cannot be issued while DMA mode is enabled because the flash

controller is operating in an extremely tightly-coupled, high-performance data transfer mode. On receipt of erroneous commands (MAP00, MAP01 or MAP11), the flash controller issues an `unsup_cmd` interrupt to inform the host about the violating command.

Consider the following points when using the DMA:

- A data DMA command is a type of MAP10 command. This command is interpreted by the data DMA engine and not by the flash controller core.
- No MAP01, MAP00, or MAP11 commands are allowed when DMA is enabled.
- Before the flash controller can accept data DMA commands, DMA must be enabled by setting the `flag` bit of the `dma_enable` register in the `dma` group.
- When DMA is enabled and the DMA engine initiates data transfers, ECC can be enabled for as-needed data correction concurrent with the data transfer.
- MAP10 commands are used along with data movements similar to MAP01 commands.
- With the exception of data DMA commands and MAP10 pipeline read and write commands, all other MAP10 commands such as erase, lock, unlock, and copy-back are forwarded to the flash controller.
- At any time, up to four outstanding data DMA commands can be handled by flash controller. During multi-page operations, the DMA transfer must not cross a flash block boundary. If it does, the flash controller generates an unsupported command (`unsup_cmd`) interrupt and drops the command.
- Data DMA commands are typically multi-page read and write commands with an associated pointer in host memory. The multi-page data is transferred to or from the host memory starting from the host memory pointer.
- Data DMA uses the `flash_burst_length` register in the `dma` group to determine the burst length value to drive on the interconnect. The data DMA hardware does not account for the interconnect's boundary crossing restrictions. The host must initialize the starting host address so that the DMA master burst transaction does not cross a 4 KB boundary.

There are two methods for initiating a DMA transaction: the multitransaction DMA command, and the burst DMA command.

14.4.9.1. Multi-Transaction DMA Command

The NAND flash controller processes multitransaction DMA commands only if it receives all four command-data pairs in order. The flash controller responds to out-of-order commands with an `unsup_cmd` interrupt. The flash controller also responds with an `unsup_cmd` interrupt if sequenced commands are interleaved with other flash controller MAP commands.

To initiate DMA with a multitransaction DMA command, you send four command-data pairs to the NAND flash controller through the Control and Data registers in the `nanddata` region, as shown in "Command-Data Pair Formats".

Related Information

Command-Data Pair Formats on page 385

14.4.9.1.1. Command-Data Pair Formats

Table 131. Command-Data Pair 1

	31:28	27:26	25:24	23:<M>	(<M> – 1):0
Command	0x0	0x2	0x0	Block address	Page address
Note: <M> = ceil(log2(<*device pages per block*>)). Therefore, use the following values: • 32 pages per block: <M>=5 • 64 pages per block: <M>=6 • 128 pages per block: <M>=7 • 256 pages per block: <M>=8 • 384 pages per block: <M>=9 • 512 pages per block: <M>=9					

	31:16	15:12	11:8	7:0
Data	0x0	0x2	0x0 = Read 0x1 = Write	<*PP*>= Number of pages

Table 132. Command-Data Pair 2

	31:28	27:26	25:24	23:8	7:0
Command	0x0	0x2	0x0	Memory address high	0x0

	31:16	15:12	11:8	7:0
Data	0x0	0x2	0x2	0x0

Table 133. Command-Data Pair 3

	31:28	27:26	25:24	23:8	7:0
Command	0x0	0x2	0x0	Memory address low[28]	0x0

	31:16	15:12	11:8	7:0
Data	0x0	0x2	0x3	0x0

Table 134. Command-Data Pair 4

	31:28	27:26	25:24	23:17	16	15:8	7:0
Command	0x0	0x2	0x0	0x0	INT[29]	Burst length	0x0
Note: INT controls the value of the `dma_cmd_comp` bit of the `intr_status0` register in the `status` group at the end of the DMA transfer. INT can take on one of the following values:							
continued...							

(27) <M> depends on the number of pages per block in the device. For more information about <M>, see the Note at the bottom of this table.

(28) The buffer address in host memory, which must be aligned to 32 bits.

(29) INT specifies the host interrupt to be generated at the end of the complete DMA transfer. For more information about INT, see the Note at the bottom of this table.

	31:28	27:26	25:24	23:17	16	15:8	7:0
• 0—Do not interrupt host. The dma_cmd_comp bit is set to 0. • 1—Interrupt host. The dma_cmd_comp bit is set to 1.							

	31:16	15:12	11:8	7:0
Data	0x0	0x2	0x4	0x0

Related Information

- Indexed Addressing on page 377
- Burst DMA Command on page 386

14.4.9.1.2. Using Multi-Transaction DMA Commands

If you want the NAND flash controller DMA to perform cacheable accesses then you must configure the cache bits by writing the `l3master` register in the `nandgrp` group in the system manager. The NAND flash controller DMA must be idle before you use the system manager to change its cache capabilities.

You can issue non-DMA MAP10 commands while the NAND flash controller is in DMA mode. For example, you might trigger a host-initiated page move between DMA commands, to achieve wear leveling. However, do not interleave non-DMA MAP10 commands between the command-data pairs in a set of multitransaction DMA commands. You must issue all four command-data pairs shown in the above tables before sending a different command.

Note: Do not issue MAP00, MAP01 or MAP11 commands while DMA is enabled.

MAP10 commands in multitransaction format are written to the `Data` register at offset 0x10 in `nanddata`, the same as MAP10 commands in increment four (INCR4) format (described in "Burst DMA Command").

Related Information

- Indexed Addressing on page 377
- Burst DMA Command on page 386
- System Manager on page 159

14.4.9.2. Burst DMA Command

You can initiate a DMA transfer by sending a command to the NAND flash controller as a burst transaction of four 16-bit accesses. This form of DMA command might be useful for initiating DMA transfers from custom IP in the FPGA fabric. Most processor cores cannot use this form of DMA command, because they cannot control the width of the burst.

When DMA is enabled, the NAND flash controller recognizes the MAP10 pipeline DMA command as an INCR4 command, in the format shown in the following table. The address decoding for MAP10 pipeline DMA command remains the same, as shown in "MAP10 Command Format".

MAP10 commands in INCR4 format are written to the `Data` register at offset 0x10 in `nanddata`, the same as MAP10 commands in multitransaction format (described in the "Multi-Transaction DMA Command").

Table 135. MAP10 Burst DMA (INCR4) Command Structure

The following table lists the MAP10 burst DMA command structure. The burst DMA command carries the same information as the multi-transaction DMA command-data pairs, but in a very different format.

Data Beat	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Beat 0	0x2				0x0: read. 0x1: write.				<*PP*>=number of pages							
Beat 1(30)	Memory address high															
Beat 2(30)	Memory address low															
Beat 3	0x0							INT(31)	Burst length							

Note: INT controls the value of the `dma_cmd_comp` bit of the `intr_status0` register in the `status` group at the end of the DMA transfer. INT can take on one of the following values:

0—Do not interrupt host. The `dma_cmd_comp` bit is set to 0.

1—Interrupt host. The `dma_cmd_compp` bit is set to 1.

You can optionally send the 16-bit fields in the above table to the NAND flash controller as four separate bursts of length 1 in sequential order. Intel recommends this method.

If you want the NAND flash controller DMA to perform cacheable accesses, you must configure the cache bits by writing the `l3master` register in the `nandgrp` group in the system manager. The NAND flash controller DMA must be idle before you use the system manager to modify its cache capabilities.

Related Information

- Multi-Transaction DMA Command on page 384
- MAP10 Command Format on page 381
- System Manager on page 159

14.4.10. ECC

The NAND flash controller incorporates ECC logic to calculate and correct bit errors. The flash controller uses a Bose-Chaudhuri-Hocquenghem (BCH) algorithm for detection of multiple errors in a page.

The NAND flash controller supports 512- and 1024-byte ECC sectors. The flash controller inserts ECC check bits for every 512 or 1024 bytes of data, depending on the selected sector size. After 512 or 1024 bytes, the flash controller writes the ECC check bit information to the device page.

ECC information is striped in between 512 or 1024 bytes of data across the page. The NAND flash controller reads ECC information in the same pattern and performs a calculation to check for the presence of errors.

(30) The buffer address in host memory, which must be aligned to 32 bits.

(31) INT specifies the host interrupt to be generated at the end of the complete DMA transfer. For more information about INT, see the Note at the bottom of this table.

Related Information

Memory Data Initialization on page 349
For more information about clearing memory data before enabling ECC, refer to "Memory Data Initialization" in the Error Checking and Correction section.

14.4.10.1. Correction Capability, Sector Size, and Check Bit Size

Table 136. Correction Capability, Sector Size, and Check Bit Size

Correction	Sector Size in Bytes	Check Bit Size in Bytes
4	512	8
8	512	14
16	512	26
24	1024	42

14.4.10.2. ECC Programming Modes

The NAND flash controller provides the following ECC programming modes that software uses to format a page:

- Main Area Transfer Mode
- Spare Area Transfer Mode
- Main+Spare Area Transfer Mode

Related Information

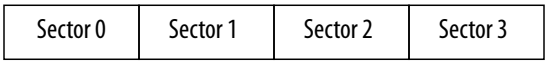
- Main Area Transfer Mode on page 388
- Spare Area Transfer Mode on page 388
- Main+Spare Area Transfer Mode on page 389

14.4.10.2.1. Main Area Transfer Mode

In main area transfer mode, when ECC is enabled, the NAND flash controller inserts ECC check bits in the data stream on writes and strips ECC check bits on reads. Software does not need to manage the ECC sectors when writing a page. ECC checking is performed by the flash controller, so software simply transfers the data.

If ECC is turned off, the NAND flash controller does not read or write ECC check bits.

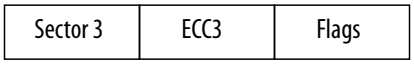
Figure 60. Main Area Transfer Mode for ECC



14.4.10.2.2. Spare Area Transfer Mode

The NAND flash controller does not introduce or interpret ECC check bits in spare area transfer mode, and acts as a pass-through for data transfer.

Figure 61. Spare Area Transfer Mode for ECC



14.4.10.2.3. Main+Spare Area Transfer Mode

In main+spare area transfer mode, the NAND flash controller expects software to format a page as shown in following figure. When ECC is enabled during a write operation, the flash controller-generated ECC check bits replace the ECC check bit data provided by software. During read operations, the flash controller forwards the ECC check bits from the device to the host. If ECC is disabled, page data received from the software is written to the device, and read data received from the device is forwarded to the host.

Figure 62. Main+Spare Area Transfer Mode for ECC



14.4.10.3. Preserving Bad Block Markers

When flash device manufacturers test their devices at the time of manufacture, they mark any bad device blocks that are found. Each bad block is marked at specific, known offsets, typically at the base of the spare area. A bad block marker is any byte value other than 0xFF (the normal state of erased flash).

Bad block markers can be overwritten by the last sector data in a page when ECC is enabled. This happens because the NAND flash controller also uses the main area of a page to store ECC information, which causes the last sector to spill over into the spare area. It is necessary for the system to preserve the bad block information prior to writing data, to ensure the correct identification of bad blocks in the flash device.

You can configure the NAND flash controller to skip over a specified number of bytes when it writes the last sector in a page to the spare area. This option allows the flash controller to preserve bad block markers. To use this option, write the desired offset to the `spare_area_skip_bytes` register in the `config` group. For example, if the device page size is 2 KB, and the device manufacturer stores the bad block markers in the first two bytes in the spare area, set the `spare_area_skip_bytes` register to 2. When the flash controller writes the last sector of the page that overlaps with the spare area, it starts at offset 2 in the spare area, skipping the bad block marker at offset 0. A value of 0 (default) specifies that no bytes are skipped. The value of `spare_area_skip_bytes` must be an even number. For example, if the bad block marker is a single byte, set `spare_area_skip_bytes` to 2.

In main area transfer mode, the NAND flash controller does not skip the bad block marker. Instead, it overrides the bad block marker with the value programmed in the `spare_area_marker` register in the `config` group. This 8-bit register is used in conjunction with the `spare_area_skip_bytes` register in the `config` group to determine which bytes in the spare area of a page should be written with a the new marker value. For example, to mark a block as good set the `spare_area_marker` register to 0xFF and set the `spare_area_skip_bytes` register to the number of bytes that the marker should be written to, starting from the base of the spare area.

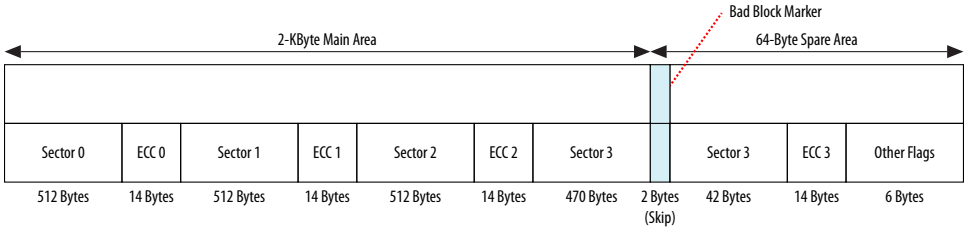
In the spare area transfer mode, the NAND flash controller ignores the `spare_area_skip_bytes` and `spare_area_marker` registers. The flash controller transfers the data exactly as received from the host or device.

In the main+spare area transfer mode, the NAND flash controller starts writing the last sector in a page into the spare area, starting at the offset specified in the `spare_area_skip_bytes` register. However, the area containing the bad block

identifier information is overwritten by the data the host writes into the page. The host writes both the data sectors and the bad block markers. The flash controller depends on the host software to set up the bad block markers properly before writing the data.

Figure 63. Bad Block Marker

The following figure shows an example of how the NAND flash controller can skip over a bad block marker. In this example, the flash device has a 2-KB page with a 64-byte spare area. A 14-byte sector ECC is shown, with 8 byte per sector correction.



Related Information

Transfer Mode Operations on page 397
For detailed information about configuring the NAND flash controller for default, spare, or main+spare area transfer mode.

14.4.10.4. Error Correction Status

The ECC error correction information (`ECCCorInfo_b01`) register, in the `ecc` group, contains error correction information for each read or write that the NAND flash controller performs. The `ECCCorInfo_b01` register contains ECC error correction information in the `max_errors_b0` and `uncor_err_b0` fields.

At the end of data correction for the transaction in progress, `ECCCorInfo_b01` holds the maximum number of corrections applied to any ECC sector in the transaction. In addition, this register indicates whether the transaction as a whole has correctable errors, uncorrectable errors, or no errors at all. A transaction has no errors when none of the ECC sectors in the transaction has any errors. The transaction is marked as uncorrectable if any one of the sectors is uncorrectable. The transaction is marked as correctable if any one sector has correctable errors and none is uncorrectable.

At the end of each transaction, the host must read this register. The value of this register provides error data to the host about the block. The host can take corrective action after the number of correctable errors encountered reaches a particular threshold value.

14.5. NAND Flash Controller Programming Model

This section describes how the NAND flash controller is to be programmed by software running on the microprocessor unit (MPU).

Note: If you write a configuration register and follow it up with a data operation that is dependent on the value of this configuration register, Intel recommends that you read the value of the register before performing the data operation. This read operation ensures that the posted write of the register is completed and takes effect before the data operation is issued to the NAND flash controller.

14.5.1. Basic Flash Programming

This section describes the steps that must be taken by the software to access and control the NAND flash controller.

14.5.1.1. NAND Flash Controller Optimization Sequence

The software must configure the flash device for interrupt or polling mode, using the `bank0` bit of the `rb_pin_enabled` register in the `config` group. If the device is in polling mode, the software must also program the additional registers, to select the times and frequencies of the polling. Program the following registers in the `config` group:

- Set the `rb_pin_enabled` register to the desired mode of operation for each flash device.
- For polling mode, set the `load_wait_cnt` register to the appropriate value depending on the speed of operation of the NAND flash controller, and the desired wait value.
- For polling mode, set the `program_wait_cnt` register to the appropriate value by software depending on the speed of operation of the NAND flash controller, and the desired wait value.
- For polling mode, set the `erase_wait_cnt` register to the appropriate value by software depending on the speed of operation of the NAND flash controller, and the desired wait value.
- For polling mode, set the `int_mon_cyccnt` register to the appropriate value by software depending on the speed of operation of the NAND flash controller, and the desired wait value.

At any time, the software can change any flash device from interrupt mode to polling mode or vice-versa, using the `bank0` bit of the `rb_pin_enabled` register.

The software must ensure that the particular flash device does not have any outstanding transactions before changing the mode of operation for that particular flash device.

14.5.1.2. Device Initialization Sequence

At initialization, the host software must program the following registers in the `config` group:

- Set the `devices_connected` register to 1.
- Set the `device_width` register to 8.
- Set the `device_main_area_size` register to the appropriate value.
- Set the `device_spare_area_size` register to the appropriate value.
- Set the `pages_per_block` register according to the parameters of the flash device.

- Set the `number_of_planes` register according to the parameters of the flash device.
- If the device allows two ROW address cycles, the `flag` bit of the `two_row_addr_cycles` register must be set to 1. The host program can ensure this condition either of the following ways:
  - Set the `flag` bit of the `bootstrap_two_row_addr_cycles` register to 1 prior to the NAND flash controller's reset initialization sequence, causing the flash controller to initialize the bit automatically.
  - Set the `flag` bit of the `two_row_addr_cycles` register directly to 1.
- Clear the `chip_enable_dont_care` register in the `config` group to 0.

The NAND flash controller can identify the flash device features, allowing you to initialize the flash controller registers to interface correctly with the device, as described in *Discovery and Initialization*.

However, a few NAND devices do not follow any universally accepted identification protocol. If connected to such a device, the NAND flash controller cannot identify it correctly. If you are using such a device, your software must use other means to ensure that the initialization registers are set up correctly.

Related Information

Discovery and Initialization on page 372

14.5.1.3. Device Operation Control

This section provides a list of registers that you need to program while choosing to use multi-plane or cache operations on the device. If the device does not support multi-plane operations or cache operations, then these registers can be left at their power-on reset values with no impact on the functionality of the NAND flash controller. Even if the device supports these sequences, the software does not need to use them. Software can leave these registers at their power-on reset values.

Program the following registers in the `config` group to achieve the best performance from a given device:

- Set `flag` bit in the `multiplane_operation` register in the `config` group to 1 if the device supports multi-plane operations to access the data on the flash device connected to the NAND flash controller. If the flash controller is set up for multi-plane operations, the number of pages to be accessed is always a multiple of the number of planes in the device.
- If the NAND flash controller is configured for multi-plane operation, and if the device has support for multi-plane read command sequence, set the `multiplane_read_enable` register in the `config` group.
- If the device implements multiplane address restrictions, set the `flag` bit in the `multiplane_addr_restrict` register to 1.
- Initialize the `die_mask` and `first_block_of_next_plane` registers as per device requirements.

- If the device supports cache command sequences, enable the `cache_write_enable` and `cache_read_enable` registers in the `config` group.
- Clear the `flag` bit of the `copyback_disable` register in the `config` group to 0 if the device does not support the copyback command sequences. The register defaults to enabled state.
- The `read_mode`, `write_mode` and `copyback_mode` registers, in the `config` group, currently need not be written by software, because the NAND flash controller is capable of using the correct sequences based on a combination of some multi-plane or cache-related settings of the NAND flash controller and the manufacturer ID. If at some future time these settings change, program the registers to accommodate the change.

14.5.1.4. ECC Enabling

Before you start any data operation on the flash device, you must decide whether you want the ECC enabled or disabled.

To prevent spurious ECC errors, software must use the memory initialization block in the ECC controller to clear the entire memory data and initialize the ECC bits. The initialization block clears the memory data. Initializing the memory with the initialization block is independent of enabling the ECC.

If the ECC needs enabling, set up the appropriate correction level depending on the page size and the spare area available on the device.

Set the `flag` bit in the `ecc_enable` register in the `config` group to 1 to enable ECC. If enabled, the following registers in the `config` group must be programmed accordingly, else they can be ignored:

- Initialize the `ecc_correction` register to the appropriate correction level.
- Program the `spare_area_skip_bytes` and `spare_area_marker` registers in the `config` group if the software needs to preserve the bad block marker.

Related Information

ECC on page 387

14.5.1.5. NAND Flash Controller Performance Registers

These registers specify the size of the bursts on the device interface, which maximizes the overall performance on the NAND flash controller.

Initialize the `flash_burst_length` register in the `dma` group to a value which maximizes the performance of the device interface by minimizing the number of bursts required to transfer a page.

14.5.1.6. Interrupt and DMA Enabling

Prior to initiating any data operation on the NAND flash controller, the software must set appropriate interrupt status register bits. If the software uses the DMA logic in the flash controller, then the appropriate DMA enable and interrupts bits in the register space must be set.

1. Set the `flag` bit in the `global_int_enable` register in the `config` group to 1, to enable global interrupt.
2. Set the relevant bits of the `intr_en0` register in the `status` group to 1 before initiating any operations if the flash controller is in interrupt mode. Intel recommends that the software reads back this register to ensure clearing an interrupt status. This recommendation applies also to an interrupt service routine.
3. Enable DMA if your application needs DMA mode. Enable DMA by setting the `flag` bit of the `dma_enable` register in the `dma` group. Intel recommends that the software reads back this register to ensure that the mode change is accepted before sending a DMA command to the flash controller.
4. If the DMA is enabled, then set up the appropriate bits of the `dma_intr_en` register in the `dma` group.

14.5.1.6.1. Order of Interrupt Status Bits Assertion

The following interrupt status bits, in the `intr_status0` register in the `status` group, are listed in the order of interrupt bit setting:

1. `time_out`—All other interrupt bits are set to 0 when the watchdog `time_out` bit is asserted.
2. `dma_cmd_comp`—This bit signifies the completion of data transfer sequence.(32)
3. `pipe_cpybck_cmd_comp`—This bit is asserted when a copyback command or the last page of a pipeline command completes.
4. `locked_blk`—This bit is asserted when a program (or erase) is performed on a locked block.
5. `INT_act`—No relationship with other interrupt status bits. Indicates a transition from 0 to 1 on the `ready_busy` pin value for that flash device.
6. `rst_comp`—No relationship with other interrupt status bits. Occurs after a reset command has completed.
7. For an erase command:
   a. `erase_fail` (if failure)
   b. `erase_comp`
8. For a program command:
   a. `locked_blk` (if performed on a locked block)
   b. `pipe_cmd_err` (if the pipeline sequence is broken by a MAP01 command)
   c. `page_xfer_inc` (at the end of each page data transfer)
   d. `program_fail` (if failure)
   e. `pipe_cpybck_cmd_comp`
   f. `program_comp`
   g. `dma_cmd_comp` (If DMA enabled)
9. For a read command:

(32) This interrupt status bit is the last to be asserted during a DMA operation to transfer data.

a. `pipe_cmd_err` (if the pipeline sequence is broken by a MAP01 command)
b. `page_xfer_inc` (at the end of each page data transfer)
c. `pipe_cpybck_cmd_comp`
d. `load_comp`
e. `ecc_uncor_error` (if failure)
f. `dma_cmd_comp` (If DMA enabled)

14.5.1.7. Timing Registers

You must optimize the following registers for your flash device's speed grade and clock frequency. The NAND flash controller operates correctly with the power-on reset values. However, functioning with power-on reset values is a non-optimal mode that provides loose timing (large margins to the signals).

Set the following registers in the `config` group to optimize the NAND flash controller for the speed grade of the connected device and frequency of operation of the flash controller:

- `twhr2_and_we_2_re`
- `tcwaw_and_addr_2_data`
- `re_2_we`
- `acc_clks`
- `rdwr_en_lo_cnt`
- `rdwr_en_hi_cnt`
- `max_rd_delay`
- `cs_setup_cnt`
- `re_2_re`

14.5.1.8. Registers to Ignore

You do not need to initialize the following registers in the `config` group:

- The `transfer_spare_reg` register—Data transfer mode can be initialized using MAP10 commands.
- The `write_protect` register—Does not need initializing unless you are testing the write protection feature.

14.5.2. Flash-Related Special Function Operations

This section describes all the special functions that can be performed on the flash memory.

The functions are defined by MAP10 commands as described in *Command Mapping*.

Related Information

Command Mapping on page 378

14.5.2.1. Erase Operations

Before data can be written to flash, an erase cycle must occur. The NAND flash memory controller supports single block and multi-plane erases.

The controller decodes the block address from the indirect addressing shown in "MAP10 Command Format".

Related Information

MAP10 Command Format on page 381

14.5.2.1.1. Single Block Erase

A single command is needed to complete a single-block erase, as follows:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the desired erase block.
2. Write 0x01 to the Data register.

For a single block erase, the register `multiplane_operation` in the `config` group must be reset.

After the device completes the erase operation, the controller generates an `erase_comp` interrupt. If the erase operation fails, the `erase_fail` interrupt is issued. The failing block's address is updated in the `err_block_addr0` register in the `status` group.

14.5.2.1.2. Multi-Plane Erase

For multi-plane erases, the `number_of_planes` register in the `config` group holds the number of planes in the flash device, and the block address specified must be aligned to the number of planes in the device. The NAND flash controller consecutively erases each block of the memory, up to the number of planes available. Issue this command as follows:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the desired erase block.
2. Write 0x01 to the Data register.

For multi-plane erase, the register `multiplane_operation` in the `config` group must be set.

After the device completes erase operation on all planes, the NAND flash controller generates an `erase_comp` interrupt. If the erase operation fails on any of the blocks in a multi-plane erase command, an `erase_fail` interrupt is issued. The failing block's address is updated in the `err_block_addr0` register in the `status` group.

14.5.2.2. Lock Operations

The NAND flash controller supports the following features:

- Flash locking—The NAND flash controller supports all flash locking operations.

  The flash device itself might have limited support for these functions. If the device does not support locking functions, the flash controller ignores these commands.
- Lock-tight—With the lock-tight feature, the NAND flash controller can prevent lock status from being changed. After the memory is locked tight, the flash controller must be reset before any flash area can be locked or unlocked.

14.5.2.2.1. Unlocking a Span of Memory Blocks

To unlock several blocks of memory, perform the following steps:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the starting address of the area to unlock.
2. Write 0x10 to the `Data` register.
3. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the ending address of the area to unlock.
4. Write 0x11 to the `Data` register.

When unlocking a range of blocks, the start block address must be less than the end block address. Otherwise, the NAND flash controller exhibits undetermined behavior.

14.5.2.2.2. Locking All Memory Blocks

To lock the entire memory:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to any memory address.
2. Write 0x21 to the `Data` register.

14.5.2.2.3. Setting Lock-Tight on All Memory Blocks

After the lock-tight is applied, unlocked areas cannot be locked, and locked areas cannot be unlocked. To lock-tight the entire memory:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to any memory address.
2. Write 0x31 to the `Data` register.

To disable the lock-tight, reset the memory controller.

14.5.2.3. Transfer Mode Operations

You can configure the NAND flash controller in one of the following modes of data transfer:

- Default area transfer mode
- Spare area transfer mode
- Main+spare area transfer mode

The NAND flash controller determines the default transfer mode from the setting of `transfer_spare_reg` register in the `config` group. Use MAP10 commands to dynamically change the transfer mode from the existing mode to the new mode. All subsequent commands are in the new mode of transfer. You must consider that transfer modes can be changed at logical data transfer boundaries. For example:

- At the beginning or end of a page in case of single page read or write.
- At the beginning or end of a complete multi-page pipeline read or write command.

14.5.2.3.1. transfer_spare_reg and MAP10 Transfer Mode Commands

The following table lists the functionality of the MAP10 transfer mode commands, and their mappings to the `transfer_spare_reg` register in the `config` group.

Table 137. transfer_spare_reg and MAP10 Transfer Mode Commands

transfer_spare_reg	MAP10 Transfer Mode Commands	Resulting NAND Flash Controller Mode
0	0x42	Main(33)
0	0x41	Spare
0	0x43	Main+spare
1	0x42	Main+spare(33)
1	0x41	Spare
1	0x43	Main+spare

Related Information

MAP10 Commands on page 380

14.5.2.3.2. Configure for Default Area Access

You only need to configure for default area access if the transfer mode was previously changed to spare area or main+spare area. To configure default area access:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to any block.
2. Write 0x42 to the `Data` register.

The NAND flash controller determines the default area transfer mode from the setting of the `transfer_spare_reg` register in the `config` group. If it is set to 1, then the transfer mode becomes main+spare area, otherwise it is main area.

14.5.2.3.3. Configure for Spare Area Access

To access only the spare area of the flash device, use the MAP10 command to set up the NAND flash controller to read or write only the spare area on the device. After the flash controller is set up, use MAP01 read and write commands to access the spare area of the appropriate block and page addresses. To configure the NAND flash controller to access the spare area only, perform the following steps:

(33) Default access mode (0x42) maps to either main (only) or main+spare mode, depending on the value of `transfer_spare_reg`.

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the target block.
2. Write 0x41 to the Data register.

14.5.2.3.4. Configure for Main+Spare Area Access

To configure the NAND flash controller to access the main+spare area:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the target block.
2. Write 0x43 to the Data register.

14.5.2.4. Read-Modify-Write Operations

To read a specific page or modify a few words, bytes, or bits in a page, use the RMW operations. A read command copies the desired data from flash memory to a page buffer. You can then modify the information in the buffer using MAP00 buffer read and write commands and issue another command to write that information back to the memory.

The read-modify-write command operates on an entire page. This command is also useful for a copy type operation, where most of a page is saved to a new location. In this type of operation, the NAND flash controller reads the data, modifies a specified number of words in the page, and then writes the modified page to a new location.

Note: Because the data is modified within the page buffer of the flash device, the NAND flash controller ECC hardware is not used in RMW operations. Software must update the ECC during RMW operations.

Note: For a read-modify-write command to work with hardware ECC, the entire page must be read into system memory, modified, then written back to flash without relying on the RMW feature.

14.5.2.4.1. Read-Modify-Write Operation Flow

1. Start the flow by reading a page from the memory:
   - Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the starting address of the desired block.
   - Write 0x60 to the Data register.

   This step makes the page available to you in the page buffer in the flash device.
2. Provide the destination page address:
   - Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the destination address of the desired block.
   - Write 0x61 to the Data register.

   This step initiates the page program and provides the destination address to the device.
3. Use the MAP00 page buffer read and write commands to modify the data in the page buffer.
4. Write the page buffer data back to memory:

- Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the same destination address.
- Write 0x62 to the Data register.

This step performs the write.

After the device completes the load operation, the NAND flash controller issues a load_comp interrupt. A program_comp interrupt is issued when the host issues the write command and the device completes the program operation.

If the page program operation (as a part of an RMW operation) results in a program failure in the device, program_fail interrupt is issued. The failing page's block and page address is updated in the err_block_addr0 and err_page_addr0 registers in the status group.

14.5.2.5. Copy-Back Operations

The NAND flash controller supports copy back operations. However, the flash device might have limited support for this function. If you attempt to perform a copy-back operation on a device that does not support copy-back, the NAND flash controller triggers an interrupt. An interrupt is also triggered if the source block is not specified before the destination block is specified, or if the destination block is not specified in the next command following a source block specification.

The NAND flash controller cannot do ECC validation in case of copy-back commands. The flash controller copies the ECC data, but does not check it during the copy operation.

Note: Intel recommends that you use copy-back only if the ECC implemented in the flash controller is strong enough so that the next access can correct accumulated errors.

The 8-bit value <*PP*> specifies the number of pages for copy-back. With this feature, the NAND flash controller can copy multiple consecutive pages with a single command. When you issue a copy-back command, the flash controller performs the operation in the background. The flash controller puts other commands on hold until the current copy-back completes.

For a multi-plane device, if the flag bit in the multiplane_operation register in the config group is set to 1, multi-plane copy-back is available as an option. In this case, the block address specified must be plane-aligned and the value <*PP*> must specify the total number of pages to copy as a multiple of the number of planes. The block address continues incrementing, keeping the page address fixed, for the total number of planes in the device before incrementing the page address.

A pipe_cpyback_cmd_comp interrupt is generated when the flash controller has completed copy-back operation of all <*PP*> pages. If any page program operation (as a part of copy back operation) results in a program failure in the device, the program_fail interrupt is issued. The failing page's block and page address is updated in the err_block_addr0 and err_page_addr0 registers in the status group.

14.5.2.5.1. Copying a Memory Area (Single Plane)

To copy <*PP*> pages from one memory location to another:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the starting address of the area to be copied.
2. Write 0x1000 to the Data register.
3. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the starting address of the new area to be written.
4. Write 0x11<*PP*> to the Data register, where <*PP*> is the number of pages to copy.

14.5.2.5.2. Copying a Memory Area (Multi-Plane)

To copy <*PP*> pages from one memory location to another:

1. Set the flag bit of the multiplane_operation register in the config group to 1.
2. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the starting address of the area to be copied. The address must be plane-aligned.
3. Write 0x1000 to the Data register.
4. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the starting address of the new area to be written. This address must also be plane-aligned.
5. Write 0x11<*PP*> to the Data register, where <*PP*> is the number of pages to copy.

The parameter <*PP*> must be a multiple of the number of planes in the device.

14.5.2.6. Pipeline Read-Ahead and Write-Ahead Operations

The NAND flash controller supports pipeline read-ahead and write-ahead operations. However, the flash device might have limited support for this function. If the device does not support pipeline read-ahead or write-ahead, the flash controller processes these commands as standard reads or writes.

The NAND flash controller can handle at the most four outstanding pipeline commands, queued up in the order in which the flash controller received the commands. The flash controller operates on the pipeline command at the head of the queue until all the pages corresponding to the pipeline command are executed. The flash controller then pops the pipeline command at the head of the queue and proceeds to work on the next pipeline command in the queue.

14.5.2.6.1. Pipeline Read-Ahead Function

The pipeline read-ahead function allows for a continuous reading of the flash memory. On receiving a pipeline read command, the flash controller immediately issues a load command to the device. While data is read out with MAP01 commands in a consecutive or multi-plane address pattern, the flash controller maintains additional cache or multi-plane read command sequencing for continuous streaming of data from the flash device.

Pipeline read-ahead commands can read data from the queue in this interleaved fashion. The parameter <PP> denotes the total number of pages in multiples of the number of planes available, and the block address must be plane-aligned, which keeps the page address constant while incrementing the block address for each page-size

chunk of data. After reading from every plane, the NAND flash controller increments the page address and resets the block address to the initial address. You can also use pipeline write-ahead commands in multi-plane mode. The write operation works similarly to the read operation, holding the page address constant while incrementing the block address until all planes are written.

Note: The same four-entry queue is used to queue the address and page count for pipeline read-ahead and write-ahead commands. This commonality requires that you use MAP01 commands to read out all pages for a pipeline read-ahead command before the next pipeline command can be processed. Similarly, you must write to all pages pertaining to pipeline write-ahead command before the next pipeline command can be processed.

Because the value of the `flag` bit of the `multiplane_operation` register in the `config` group determines pipeline read-ahead or write-ahead behavior, it can only be changed when the pipeline registers are empty.

When the host issues a pipeline read-ahead command, and the flash controller is idle, the load operation occurs immediately.

Note: The read-ahead command does not return the data to the host, and the write-ahead command does not write data to the flash address. The NAND flash controller loads the read data. The read data is returned to the host only when the host issues MAP01 commands to read the data. Similarly, the flash controller loads the write data, and writes it to the flash only when the host issues MAP01 commands to write the data.

Set Up a Single Area for Pipeline Read-Ahead

To set up an area for pipeline read-ahead, perform the following steps:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the starting address of the block to pre-read.
2. Write 0x20<*PP*> to the `Data` register, where the 0 sets this command as a read-ahead and <*PP*> is the number of pages to pre-read. The pages must not cross a block boundary. If a block boundary is crossed, the NAND flash controller generates an unsupported command (`unsup_cmd`) interrupt and drops the command.

The read-ahead command is a hint to the flash device to start loading the next page in the page buffer as soon as the previous page buffer operation has completed. After you set up the read-ahead, use a MAP01 command to actually read the data. In the MAP01 command, specify the same starting address as in the read-ahead.

If the read command received following a pipeline read-ahead request is not to a pre-read page, then an interrupt bit is set to 1 and the pipeline read-ahead or write-ahead registers are cleared. You must issue a new pipeline read-ahead request to re-load the same data. You must use MAP01 commands to read all of the data that is pre-read before the NAND flash controller returns to the idle state.

14.5.2.6.2. Pipeline Write-Ahead Function

The pipeline write-ahead function allows for a continuous writing of the flash memory. While data is written with MAP01 commands in a consecutive or multi-plane address pattern, the NAND flash controller maintains cache or multi-plane command sequences for continuous streaming of data into the flash device.

For pipeline write commands, if any page program results in a failure in the device, a `program_fail` interrupt is issued. The failing page's block and page addresses are updated in the `err_block_addr0` and `err_page_addr0` registers in the `status` group.

Set Up a Single Area for Pipeline Write-Ahead

To set up an area for pipeline write-ahead:

1. Write to the command register, setting the CMD_MAP field to 2 and the BLK_ADDR field to the starting address of the block to pre-write.
2. Write 0x21<PP> to the `Data` register, where the value 1 sets this command as a write-ahead and <PP> is the number of pages to pre-write. The pages must not cross a block boundary. If a block boundary is crossed, the NAND flash controller generates an unsupported command (`unsup_cmd`) interrupt and drops the command.

After you set up the write-ahead, use a MAP01 command to write the data. In the MAP01 command, specify the same starting address as in the write-ahead.

If the write command received following a pipeline write-ahead request is not to a pre-written page, then an interrupt bit is set to 1 and the pipeline read-ahead or write-ahead registers are cleared. You must issue a new pipeline write-ahead request to configure the write logic.

You must use MAP01 commands to write all of the data that is pre-written before the NAND flash controller returns to the idle state.

14.5.2.6.3. Other Supported Commands

MAP01 commands must read or write pages in the same sequence that the pipelined commands were issued to the NAND flash controller. If the host issues multiple pipeline commands, pages must be read or written in the order the pipeline commands were issued. It is not possible to read or write pages for a second pipeline command before completing the first pipeline command. If the pipeline sequence is broken by a MAP01 command, the `pipe_cmd_err` interrupt is issued, and the flash controller clears the pipeline command queue. The flash controller services the violating incoming MAP01 read or write request with a normal page read or write sequence.

For a multi-plane device that supports multi-plane programming, you must set the `flag` bit of the `multiplane_operation` register in the `config` group to 1. In this case, the data is interleaved into page-size chunks to consecutive blocks.

A `pipe_cpyback_cmd_comp` interrupt is generated when the NAND flash controller has finished processing a pipeline command and has discarded that command from its queue. At this point of time, the host can send another pipeline command. A pipeline command is popped from the queue, and an interrupt is issued when the flash controller has started processing the last page of pipeline command. Hence, the `pipe_cpyback_cmd_comp` interrupt is issued prior to the last page load in the case of a pipeline read command and start of data transfer of the last page to be programmed, in the case of a pipeline write command.

An additional `program_comp` interrupt is generated when the last page program operation completes in the case of a pipeline write command.

If the device command set requires the NAND flash controller to issue a load command for the last page in the pipeline read command, a `load_comp` interrupt is generated after the last page load operation completes.

The pipeline commands sequence advanced commands in the device, such as cache and multi-plane. When the NAND flash controller receives a multi-page read or write pipeline command, it sequences commands sent to the device depending on settings in the following registers in the `config` group:

- `cache_read_enable`
- `cache_write_enable`
- `multiplane_operation`

For a device that supports cache read sequences, the `flag` bit of the `cache_read_enable` register must be set to 1. The NAND flash controller sequences each multi-page pipeline read command as a cache read sequence. For a device that supports cache program command sequences, `cache_write_enable` must be set. The flash controller sequences each multi-page write pipeline command as a cache write sequence.

For a device that has multi-planes and supports multi-plane program commands, the NAND flash controller register `multiplane_operation`, in the `config` group, must be set. On receiving the multi-page pipeline write command, the flash controller sequences the device with multi-plane program commands and expects that the host transfers data to the flash controller in an even-odd block increment addressing mode.

14.6. NAND Flash Controller Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

Related Information

Error Checking and Correction Controller on page 345
For more information about how to program the ECC registers, refer to this chapter.

15. SD/MMC Controller

The hard processor system (HPS) provides a Secure Digital/Multimedia Card (SD/MMC) controller for interfacing to external SD and MMC flash cards, secure digital I/O (SDIO) devices, and Consumer Electronics Advanced Transport Architecture (CE-ATA) hard drives. The SD/MMC controller enables you to store boot images and boot the processor system from the removable flash card. You can also use the flash card to expand the on-board storage capacity for larger applications or user data. Other applications include interfacing to embedded SD (eSD) and embedded MMC (eMMC) non-removable flash devices.

The SD/MMC controller is based on the Synopsys DesignWare Mobile Storage Host (SD/MMC controller) controller.(34)

This document refers to SD/SDIO commands, which are documented in detail in the *Physical Layer Simplified Specification, Version 3.01* and the *SDIO Simplified Specification, Version 2.00* described on the SD Association website.

Related Information

- SD Association
  To learn more about how SD technology works, visit the SD Association website (www.sdcard.org).
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

15.1. Features of the SD/MMC Controller

The HPS SD/MMC controller offers the following features:

(34) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Supports HPS boot from mobile storage
- Supports the following standards or card types:[35]
  - SD, including eSD—version 3.0
  - SDIO, including embedded SDIO (eSDIO)—version 3.0
  - CE-ATA—version 1.1
- Supports various types of multimedia cards (MMC version 4.41 and eMMC version 4.5[36])[37]
  - MMC: 1-bit data bus
  - Reduced-size MMC (RSMMC): 1-bit and 4-bit data bus
  - MMCPlus: 1-bit, 4-bit, and 8-bit data bus
  - MMCMobile: 1-bit data bus
  - Embedded MMC (eMMC) version 4.5: 1-bit, 4-bit, and 8-bit data bus
- Supports the Micron MTFC16GJDDQ-4M IT (16 GB) eMMC flash memory that is verified to work with the HPS.
- Supports the Kingston* 4 GB SD Micro flash card
- Integrated descriptor-based direct memory access (DMA)
- Internal 4 KB receive and transmit FIFO buffer

Unsupported Features

- Card Detect is only supported on interfaces routed via the FPGA fabric. The Card Detect Interface signals are not included if the interface is pinned out to HPS I/O.
- The SD/MMC controller does not directly support voltage switching, card interrupts, or back-end power control of eSDIO card devices. However, you can connect these signals to general-purpose I/Os (GPIOs).
- The SD/MMC controller does not contain a reset output as part of the external card interface. To reset the flash card device, consider using a general purpose output pin.

Related Information

- MMC Support Matrix on page 407
  For more information on what is supported, refer to the MMC Support Matrix table.
- Supported Flash Devices for Arria 10 SoC
  For more information, refer to the supported SD/SDHC/SDXC/MMC/eMMC flash devices section on this page.

(35) SD and SDIO do not support SDR50, SDR104, and DDR50 modes.

(36) This is a new bus mode for Arria 10.

(37) The MMC and RSMMC does not support DDR. However, the MMCPlus, MMCMobile, and eMMC do support DDR, but not by the HPS.

15.1.1. SD Card Support Matrix

Table 138. SD Card Support Matrix

Device Card Type	Voltages Supported[38]		Bus Modes Supported			Bus Speed Modes Supported			
						Default Speed	High Speed	SDR12	SDR25[39]
	3.0 V	1.8 V[40]	1 bit	4 bit	8 bit	12.5 MBps 25 MHz	25 MBps 50 MHz	12.5 MBps 25 MHz	25 MBps 50 MHz
SDSC (SD)	√		√			√	√		
SDHC	√	√	√	√		√	√	√	√
SDXC	√	√	√	√		√	√	√	√
eSD	√	√	√	√		√	√	√	√
SDIO	√	√	√	√		√	√	√	√
eSDIO	√	√	√	√	√	√	√	√	√

Note: Card form factors (such as mini and micro) are not enumerated in the above table because they do not impact the card interface functionality.

15.1.2. MMC Support Matrix

Table 139. MMC Support Matrix

Card Device Type	Max Clock Speed (MHz)	Max Data Rate (MBps)	Voltages Supported		Bus Modes Supported			Bus Speed Modes Supported	
			3.3 V	1.8 V	1 bit	4 bit	8 bit	Default Speed	High Speed
MMC	20	2.5	√		√			√	
RSMMC	20	10	√		√	√		√	√
MMCPlus	50[41]	50	√		√	√	√	√	√
MMCMobile	50	6.5	√	√	√			√	√
eMMC	50	50	√	√	√	√	√	√	√

(38) Because SD cards initially operate at 3.0 V and can switch to 1.8V after power-up and the BSEL values are constant during the boot process, transceivers are required to support level-shifting and isolation.

(39) SDR25 speed mode requires 1.8-V signaling. Note that even if a card supports UHS-I modes (for example SDR50, SDR104, DDR50) it can still communicate at the lower speeds (for example SDR12, SDR25).

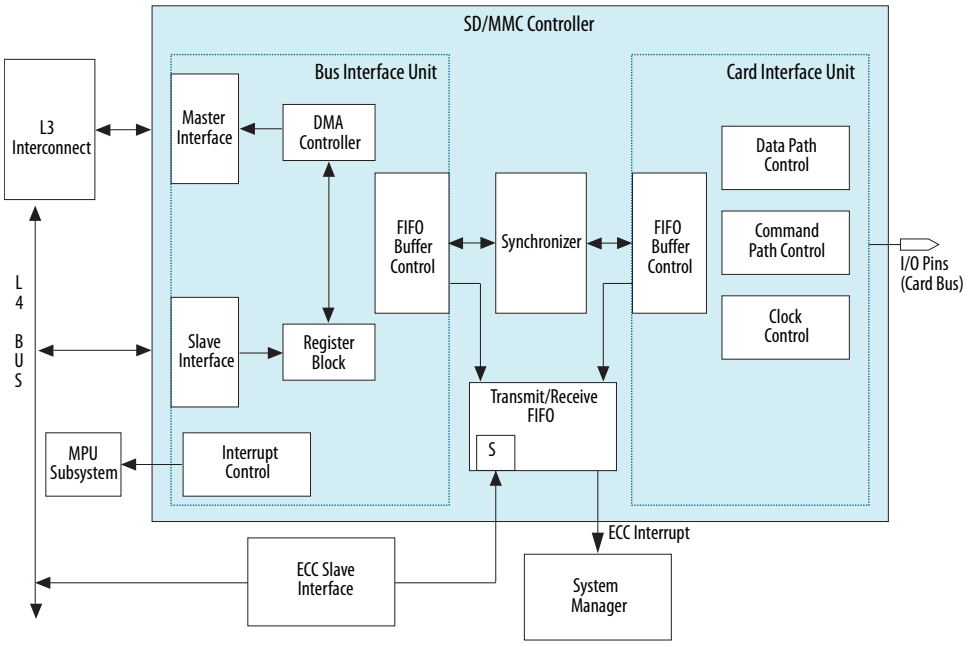
(40) Where supported, external transceivers are needed to switch the voltage.

(41) Supports a maximum clock rate of 50 MHz instead of 52 MHz (specified in MMC specification).

15.2. SD/MMC Controller Block Diagram and System Integration

The SD/MMC controller includes a bus interface unit (BIU) and a card interface unit (CIU). The BIU provides a slave interface for a host to access the control and status registers (CSRs). Additionally, this unit also provides independent FIFO buffer access through a DMA interface. The DMA controller is responsible for exchanging data between the system memory and FIFO buffer. The DMA registers are accessible by the host to control the DMA operation. The CIU supports the SD, MMC, and CE-ATA protocols on the controller, and provides clock management through the clock control block. The interrupt control block for generating an interrupt connects to the generic interrupt controller in the Arm Cortex-A9 microprocessor unit (MPU) subsystem.

Figure 64. SD/MMC Controller Connectivity



15.3. SD/MMC Controller Signal Description

The following table shows the SD/MMC controller signals that are routed to the FPGA and the HPS I/O.

Note: The last five signals in the table are not routed to HPS I/O, but only to the FPGA. For more information on how each of the signals are routed to the FPGA and HPS I/O pins, refer to the *HPS Component Interfaces* chapter.

Table 140. SD/MMC Controller Interface I/O Pins

Signal	Width	Direction	Description
sdmmc_cclk_out	1	Out	Clock from controller to the card
sdmmc_cmd_i	1	In	Card command
sdmmc_cmd_o	1	Out	

continued...

Signal	Width	Direction	Description
sdmmc_cmd_oe		Out	
sdmmc_pwr_ena_o	1	Out	External device power enable
sdmmc_data_i	8	In	Card data
sdmmc_data_o	8	Out	
sdmmc_data_oe	1	Out	
sdmmc_cdn_i	1	In	Card detect signal
sdmmc_wp_i	1	In	Card write protect signal
sdmmc_vs_o	1	Out	Voltage switching between 3.3V and 1.8V
sdmmc_rstn_o	1	Out	Card reset signal used in MMC mode
sdmmc_card_intn_i	1	In	Card interrupt signal

Related Information

HPS Component Interfaces on page 738
For more information which SD/MMC Controller signals are routed to the FPGA and the HPS I/O, refer to this chapter.

15.4. Functional Description of the SD/MMC Controller

This section describes the SD/MMC controller components and how the controller operates.

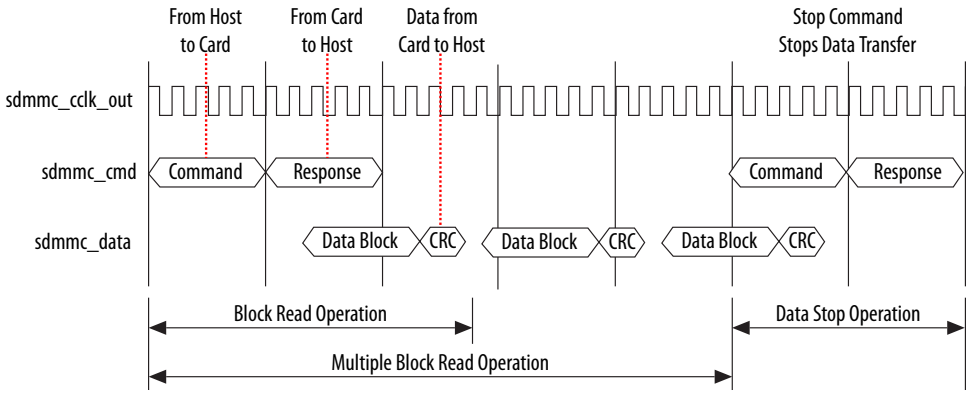
15.4.1. SD/MMC/CE-ATA Protocol

The SD/MMC/CE-ATA protocol is based on command and data bit streams that are initiated by a start bit and terminated by a stop bit. Additionally, the SD/MMC controller provides a reference clock and is the only master interface that can initiate a transaction.†

- Command—a token transmitted serially on the CMD pin that starts an operation.†
- Response—a token from the card transmitted serially on the CMD pin in response to certain commands.†
- Data—transferred serially using the data pins for data movement commands.†

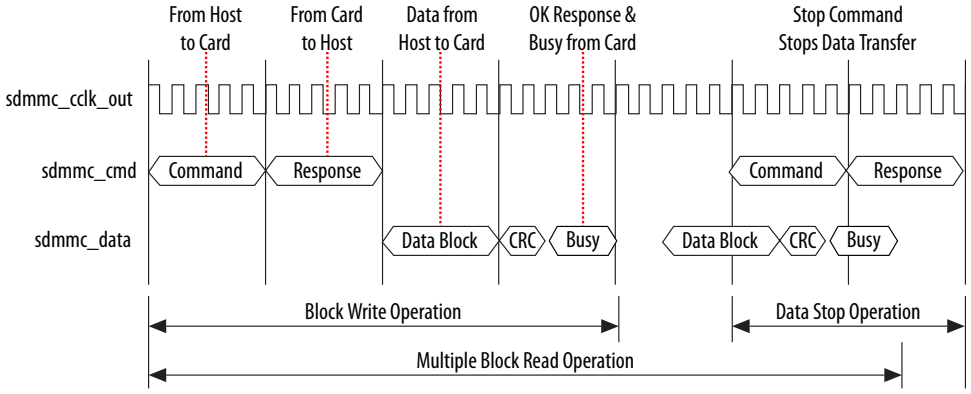
In the following figure, the clock is a representative only and does not show the exact number of clock cycles.

Figure 65. Multiple–Block Read Operation†



The following figure illustrates an example of a command token sent by the host in a multiple-block write operation.

Figure 66. Multiple-–Block Write Operation†



15.4.2. BIU

The Bus Interface Unit (BIU) interfaces with the Card Interface Unit (CIU), and is connected to the level 3 (L3) interconnect and level 4 (L4) peripheral buses. The BIU consists of the following primary functional blocks, which are defined in the following sections:

- Slave interface
- Register block
- FIFO buffer
- Interrupt control
- Internal DMA controller

15.4.2.1. Slave Interface

The host processor accesses the SD/MMC controller registers and data FIFO buffers through the slave interface.

15.4.2.2. Register Block

The register block is part of the BIU and provides read and write access to the CSRs.

All registers reside in the BIU clock domain, `l4_mp_clk`. When a command is sent to a card by setting the start command bit (`start_cmd`) of the command register (`cmd`) to 1, all relevant registers needed for the CIU operation are copied to the CIU block. During this time, software must not write to the registers that are transferred from the BIU to the CIU. The software must wait for the hardware to reset the `start_cmd` bit to 0 before writing to these registers again. The register unit has a hardware locking feature to prevent illegal writes to registers.

15.4.2.2.1. Registers Locked Out Pending Command Acceptance

After a command start is issued by setting the `start_cmd` bit of the `cmd` register, the following registers cannot be rewritten until the command is accepted by the CIU:†

- Command (`cmd`)†
- Command argument (`cmdarg`)†
- Byte count (`bytcnt`)†
- Block size (`blksiz`)†
- Clock divider (`clkdiv`)†
- Clock enable (`clkena`)†
- Clock source (`clksrc`)†
- Timeout (`tmout`)†
- Card type (`ctype`)†

The hardware resets the `start_cmd` bit after the CIU accepts the command. If a host write to any of these registers is attempted during this locked time, the write is ignored and the hardware lock write error bit (`hle`) is set to 1 in the raw interrupt status register (`rintsts`). Additionally, if the interrupt is enabled and not masked for a hardware lock error, an interrupt is sent to the host.†

Once a command is accepted, you can send another command to the CIU—which has a one-deep command queue—under the following conditions:†

- If the previous command is not a data transfer command, the new command is sent to the SD/MMC/CE-ATA card once the previous command completes.†
- If the previous command is a data transfer command and if the wait previous data complete bit (`wait_prvdata_complete`) of the `cmd` register is set to 1 for the new command, the new command is sent to the SD/MMC/CE-ATA card only when the data transfer completes.†
- If the `wait_prvdata_complete` bit is 0, the new command is sent to the SD/MMC/CE-ATA card as soon as the previous command is sent. Typically, use this feature to stop or abort a previous data transfer or query the card status in the middle of a data transfer.†

15.4.2.3. Interrupt Controller Unit

The interrupt controller unit generates an interrupt that depends on the `rintsts` register, the interrupt mask register (`intmask`), and the interrupt enable bit (`int_enable`) of the control register (`ctrl`). Once an interrupt condition is detected, the controller sets the corresponding interrupt bit in the `rintsts` register. The bit in the `rintsts` register remains set until the software clears the bit by writing a 1 to the interrupt bit; writing a 0 leaves the bit untouched.

The interrupt controller unit generates active high, level sensitive interrupts that are asserted only when at least one bit in the `rintsts` register is set to 1, the corresponding `intmask` register bit is 1, and the `int_enable` bit of the `ctrl` register is 1.

The `int_enable` bit of the `ctrl` register is cleared during a power-on reset, and the `intmask` register bits are set to 0x0000000, which masks all the interrupts.

Table 141. Interrupt Status Register Bits†

Bits	Interrupt	Description
16	SDIO Interrupts†	Interrupts from SDIO cards.†
15	End Bit Error (read)/Write no CRC (EBE)†	Error in end-bit during read operation, or no data CRC received during write operation.† *Note:* For MMC CMD19, there may be no CRC status returned by the card. Hence, EBE is set for CMD19. The application should not treat this as an error. †
14	Auto Command Done (ACD)†	Stop/abort commands automatically sent by card unit and not initiated by host; similar to Command Done (CD) interrupt. † **Recommendation:** Software typically need not enable this for non CE-ATA accesses; Data Transfer Over (DTO) interrupt that comes after this interrupt determines whether data transfer has correctly competed. For CE-ATA accesses, if the software sets `send_auto_stop_ccsd` bit in the control register, then software should enable this bit.†
13	Start Bit Error (SBE)	Error in data start bit when data is read from a card. In 4-bit mode, if all data bits do not have start bit, then this error is set.
12	Hardware Locked write Error (HLE)†	During hardware-lock period, write attempted to one of locked registers. †
11	FIFO Underrun/Overrun Error (FRUN)†	Host tried to push data when FIFO was full, or host tried to read data when FIFO was empty. Typically this should not happen, except due to error in software. † Card unit never pushes data into FIFO when FIFO is full, and pop data when FIFO is empty. † If IDMAC (Internal Direct Memory Access Controller) is enabled, FIFO underrun/overrun can occur due to a programming error on MSIZE and watermark values in FIFOTH register; for more information, refer to *Internal Direct Memory Access Controller (IDMAC)* section in the "Synopsys DesignWare Cores Mobile Storage Host Databook".†
10	Data Starvation by Host Timeout (HTO)†	To avoid data loss, card clock out (`cclk_out`) is stopped if FIFO is empty when writing to card, or FIFO is full when reading from card. Whenever card clock is

continued...

Bits	Interrupt	Description
		stopped to avoid data loss, data-starvation timeout counter is started with data-timeout value. This interrupt is set if host does not fill data into FIFO during write to card, or does not read from FIFO during read from card before timeout period. † Even after timeout, card clock stays in stopped state, with CIU state machines waiting. It is responsibility of host to push or pop data into FIFO upon interrupt, which automatically restarts `cclk_out` and card state machines. † Even if host wants to send stop/abort command, it still must ensure to push or pop FIFO so that clock starts in order for stop/abort command to send on cmd signal along with data that is sent or received on data line. †
9	Data Read Timeout (DRTO)/Boot Data Start (BDS)†	• In Normal functioning mode: Data read timeout (DRTO) Data timeout occurred. Data Transfer Over (DTO) also set if data timeout occurs. † • In Boot Mode: Boot Data Start (BDS) When set, indicates that SD/MMC controller has started to receive boot data from the card. A write to this register with a value of 1 clears this interrupt.†
8	Response Timeout (RTO)/ Boot Ack Received (BAR)†	• In Normal functioning mode: Response timeout (RTO) Response timeout occurred. Command Done (CD) also set if response timeout occurs. If command involves data transfer and when response times out, no data transfer is attempted by SD/MMC controller.† • In Boot Mode: Boot Ack Received (BAR) When expect_boot_ack is set, on reception of a boot acknowledge pattern—0-1-0—this interrupt is asserted. A write to this register with a value of 1 clears this interrupt.†
7	Data CRC Error (DCRC)†	Received Data CRC does not match with locally-generated CRC in CIU; expected when a negative CRC is received. †
6	Response CRC Error (RCRC)†	Response CRC does not match with locally-generated CRC in CIU.†
5	Receive FIFO Data Request (RXDR)†	Interrupt set during read operation from card when FIFO level is greater than Receive-Threshold level.† **Recommendation:** In DMA modes, this interrupt should not be enabled.† ISR, in non-DMA mode: `pop RX_WMark + 1 data from FIFO †`
4	Transmit FIFO Data Request (TXDR)†	Interrupt set during write operation to card when FIFO level reaches less than or equal to Transmit-Threshold level.† **Recommendation:** In DMA modes, this interrupt should not be enabled.† ISR in non-DMA mode: † `if (pending_bytes > \† (FIFO_DEPTH - TX_WMark))† push (FIFO_DEPTH - \† TX_WMark) data into FIFO† else† push pending_bytes data \† into FIFO†`

continued...

Bits	Interrupt	Description
3	Data Transfer (DTO)†	Data transfer completed, even if there is Start Bit Error or CRC error. This bit is also set when "read data-timeout" occurs or CCS is sampled from CE-ATA device.† **Recommendation:** In non-DMA mode, when data is read from card, on seeing interrupt, host should read any pending data from FIFO. In DMA mode, DMA controllers guarantee FIFO is flushed before interrupt.† *Note:* DTO bit is set at the end of the last data block, even if the device asserts MMC busy after the last data block.†
2	Command Done (CD)†	Command sent to card and received response from card, even if Response Error or CRC error occurs. Also set when response timeout occurs or CCSD sent to CE-ATA device.†
1	Response Error (RE)†	Error in received response set if one of following occurs:† • Transmission bit != 0† • Command index mismatch† • End-bit != 1†
0	Card-Detect (CDT)†	When one or more cards inserted or removed, this interrupt occurs. Software should read card-detect register (CDETECT, 0x50) to determine current card status.† **Recommendation:** After power-on and before enabling interrupts, software should read card detect register and store it in memory. When interrupt occurs, it should read card detect register and compare it with value stored in memory to determine which card(s) were removed/inserted. Before exiting ISR, software should update memory with new card-detect value.†

15.4.2.3.1. Interrupt Setting and Clearing

The SDIO Interrupts, Receive FIFO Data Request, and Transmit FIFO Data Request interrupts are set by level-sensitive interrupt sources. Therefore, the interrupt source must be first cleared before you can reset the interrupt's corresponding bit in the `rintsts` register to 0.†

For example, on receiving the Receive FIFO Data Request interrupt, the FIFO buffer must be emptied so that the FIFO buffer count is not greater than the RX watermark, which causes the interrupt to be triggered.†

The rest of the interrupts are triggered by single clock-pulse-width sources.†

15.4.2.4. FIFO Buffer

The SD/MMC controller has a 4 KB data FIFO buffer for storing transmit and receive data.

The FIFO has an ECC controller built-in to provide ECC protection. The ECC controller is able to detect single-bit and double-bit errors, and correct the single-bit errors. The ECC operation and functionality is programmable through the ECC register slave interface. The ECC register slave interface provides host access to configure the ECC logic as well as inject bit errors into the memory. It also provides the host access to memory initialization hardware used to clear out the memory contents including the ECC bits. The ECC controller generates interrupts upon occurrences of single- and double-bit errors, and the interrupt signals are connected to the system manager.

Note: Since SD/MMC has multiple memories, it must initialize both memories explicitly. Initialization is controlled by software through the ECC Control (CTRL) register. This process cannot be interrupted or stopped once it starts; hence software must wait for the initialization complete bit to be set in the Initialization Status (INITSTAT) register. Memory accesses are allowed after the initialization process is complete.

Related Information

- System Manager on page 159
- Error Checking and Correction Controller on page 345
  For more information about ECC, refer to the *Error Checking and Correction Controller* chapter of the Arria 10 Device Handbook.
- Memory Data Initialization on page 349
  For more information about clearing memory data before enabling ECC, refer to "Memory Data Initialization" in the Error Checking and Correction section.

15.4.2.5. Internal DMA Controller

Internal DMA controller (AHB Master) enables the core to act as a Master on the AHB to transfer data to and from the AHB.

- Supports 32-bit data
- Supports split, retry, and error AHB responses, but does not support wrap
- Configurable for little-endian or big-endian mode
- Allows the selection of AHB burst type through software

The internal DMA controller has a CSR and a single transmit or receive engine, which transfers data from system memory to the card and vice versa. The controller uses a descriptor mechanism to efficiently move data from source to destination with minimal host processor intervention. You can configure the controller to interrupt the host processor in situations such as transmit and receive data transfer completion from the card, as well as other normal or error conditions. The DMA controller and the host driver communicate through a single data structure.†

The internal DMA controller transfers the data received from the card to the data buffer in the system memory, and transfers transmit data from the data buffer in the memory to the controller's FIFO buffer. Descriptors that reside in the system memory act as pointers to these buffers.†

A data buffer resides in the physical memory space of the system memory and consists of complete or partial data. The buffer status is maintained in the descriptor. Data chaining refers to data that spans multiple data buffers. However, a single descriptor cannot span multiple data buffers.†

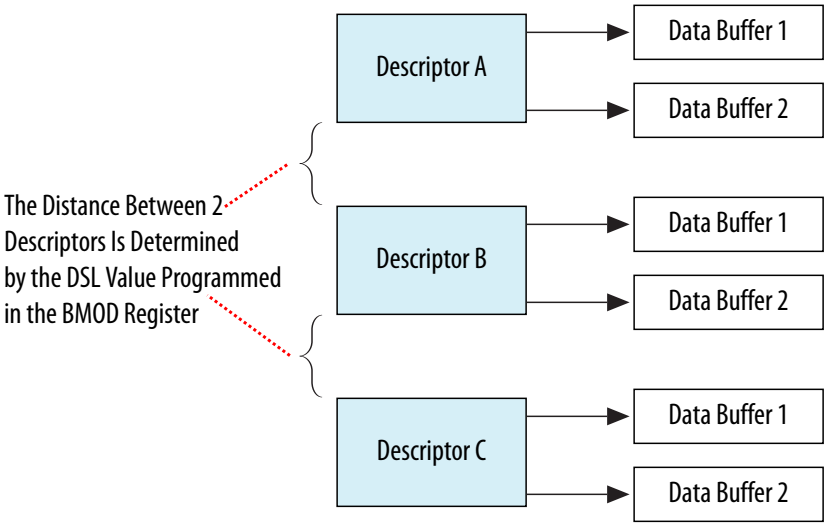
A single descriptor is used for both reception and transmission. The base address of the list is written into the descriptor list base address register (dbaddr). A descriptor list is forward linked. The last descriptor can point back to the first entry to create a ring structure. The descriptor list resides in the physical memory address space of the host. Each descriptor can point to a maximum of two data buffers.†

15.4.2.5.1. Internal DMA Controller Descriptors

The internal DMA controller uses these types of descriptor structures:†

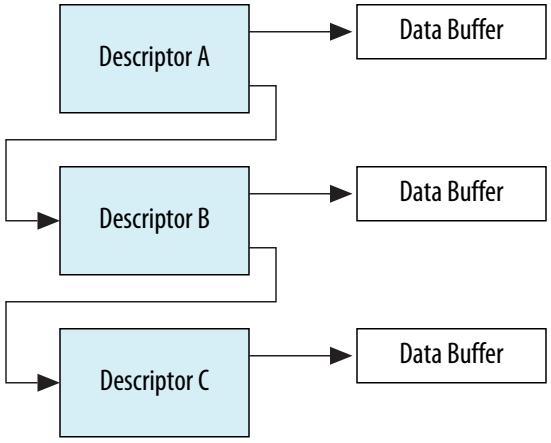
- Dual-buffer structure—The distance between two descriptors is determined by the skip length value written to the descriptor skip length field (`dsl`) of the bus mode register (`bmod`).†

Figure 67. Dual-Buffer Descriptor Structure†



- Chain structure—Each descriptor points to a unique buffer, and to the next descriptor in a linked list.†

Figure 68. Chain Descriptor Structure†



15.4.2.5.2. Internal DMA Controller Descriptor Address

The descriptor address must be aligned to the 32-bit bus. Each descriptor contains 16 bytes of control and status information.†

Table 142. Descriptor Format

Name	Off-set	31	30	29:27	26	25:14	13	12:7	6	5	4	3	2	1	0
DES0	0	OWN	CES	—						ER	CH	FS	LD	DIC	—
DES1	4	—				BS2		BS1							
DES2	8	BAP1													
DES3	12	BAP2 or Next Descriptor Address													

Related Information

Internal DMA Controller Descriptor Fields on page 417
Refer to this table for information about each of the bits of the descriptor.

15.4.2.5.3. Internal DMA Controller Descriptor Fields

The DES0 field in the internal DMA controller descriptor contains control and status information.

Table 143. Internal DMA Controller DES0 Descriptor Field†

Bits	Name	Description
31	OWN	When set to 1, this bit indicates that the descriptor is owned by the internal DMA controller. When this bit is set to 0, it indicates that the descriptor is owned by the host. The internal DMA controller resets this bit to 0 when it completes the data transfer.
30	Card Error Summary (CES)	The CES bit indicates whether a transaction error occurred. The CES bit is the logical OR of the following error bits in the `rintsts` register. • End-bit error (`ebe`) • Response timeout (`rto`) • Response CRC (`rcrc`) • Start-bit error (`sbe`) • Data read timeout (`drto`) • Data CRC for receive (`dcrc`) • Response error (`re`)
29:6	Reserved	—
5	End of Ring (ER)	When set to 1, this bit indicates that the descriptor list reached its final descriptor. The internal DMA controller returns to the base address of the list, creating a descriptor ring. ER is meaningful for only a dual-buffer descriptor structure.
4	Second Address Chained (CH)	When set to 1, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When this bit is set to 1, BS2 (DES1[25:13]) must be all zeros.
3	First Descriptor (FD)	When set to 1, this bit indicates that this descriptor contains the first buffer of the data. If the size of the first buffer is 0, next descriptor contains the beginning of the data.

continued...

Bits	Name	Description
2	Last Descriptor (LD)	When set to 1, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the data.
1	Disable Interrupt on Completion (DIC)	When set to 1, this bit prevents the setting of the TI/RI bit of the internal DMA controller status register (`idsts`) for the data that ends in the buffer pointed to by this descriptor.
0	Reserved	—

Table 144. Internal DMA Controller DES1 Descriptor Field†

The DES1 descriptor field contains the buffer size.

Bits	Name	Description
31:26	Reserved	—
25:13	Buffer 2 Size (BS2)	This field indicates the second data buffer byte size. The buffer size must be a multiple of four. When the buffer size is not a multiple of four, the resulting behavior is undefined. This field is not valid if DES0[4] is set to 1.
12:0	Buffer 1 Size (BS1)	Indicates the data buffer byte size, which must be a multiple of four bytes. When the buffer size is not a multiple of four, the resulting behavior is undefined. If this field is 0, the DMA ignores the buffer and proceeds to the next descriptor for a chain structure, or to the next buffer for a dual-buffer structure. If there is only one descriptor and only one buffer to be programmed, you need to use only buffer 1 and not buffer 2.

Table 145. Internal DMA Controller DES2 Descriptor Field†

The DES2 descriptor field contains the address pointer to the data buffer.

Bits	Name	Description
31:0	Buffer Address Pointer 1 (BAP1)	These bits indicate the physical address of the first data buffer. The internal DMA controller ignores DES2 [1:0], because it only performs 32-bit aligned accesses.

Table 146. Internal DMA Controller DES3 Descriptor Field†

The DES3 descriptor field contains the address pointer to the next descriptor if the present descriptor is not the last descriptor in a chained descriptor structure or the second buffer address for a dual-buffer structure.†

Bits	Name	Description
31:0	Buffer Address Pointer 2 (BAP2) or Next Descriptor Address	These bits indicate the physical address of the second buffer when the dual-buffer structure is used. If the Second Address Chained (DES0[4]) bit is set to 1, this address contains the pointer to the physical memory where the next descriptor is present. If this is not the last descriptor, the next descriptor address pointer must be aligned to 32 bits. Bits 1 and 0 are ignored.

15.4.2.5.4. Host Bus Burst Access

The internal DMA controller attempts to issue fixed-length burst transfers on the master interface if configured using the fixed burst bit (`fb`) of the `bmod` register. The maximum burst length is indicated and limited by the programmable burst length (`pbl`) field of the `bmod` register. When descriptors are being fetched, the master interface always presents a burst size of four to the interconnect.†

The internal DMA controller initiates a data transfer only when sufficient space to accommodate the configured burst is available in the FIFO buffer or the number of bytes to the end of transfer is less than the configured burst-length. When the DMA master interface is configured for fixed-length bursts, it transfers data using the most efficient combination of INCR4, INCR8 or INCR16 and SINGLE transactions. If the DMA master interface is not configured for fixed length bursts, it transfers data using INCR (undefined length) and SINGLE transactions.†

15.4.2.5.5. Host Data Buffer Alignment

The transmit and receive data buffers in system memory must be aligned to a 32-bit boundary.

15.4.2.5.6. Buffer Size Calculations

The driver knows the amount of data to transmit or receive. For transmitting to the card, the internal DMA controller transfers the exact number of bytes from the FIFO buffer, indicated by the buffer size field of the DES1 descriptor field.†

If a descriptor is not marked as last (with the LD bit of the DES0 field set to 0) then the corresponding buffer(s) of the descriptor are considered full, and the amount of valid data in a buffer is accurately indicated by its buffer size field. If a descriptor is marked as last, the buffer might or might not be full, as indicated by the buffer size in the DES1 field. The driver is aware of the number of locations that are valid.† The driver is expected to ignore the remaining, invalid bytes.

15.4.2.5.7. Internal DMA Controller Interrupts

Interrupts can be generated as a result of various events. The `idsts` register contains all the bits that might cause an interrupt. The internal DMA controller interrupt enable register (`idinten`) contains an enable bit for each of the events that can cause an interrupt to occur.†

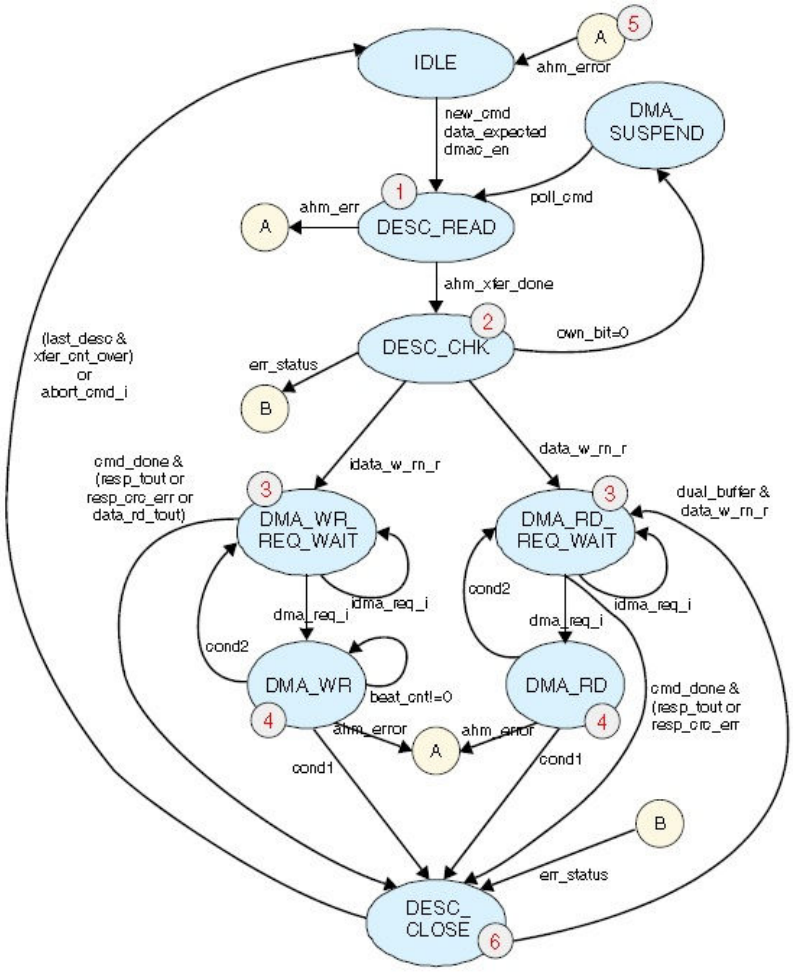
There are two summary interrupts—the normal interrupt summary bit (`nis`) and the abnormal interrupt summary bit (`ais`)—in the `idsts` register.† The `nis` bit results from a logical OR of the transmit interrupt (`ti`) and receive interrupt (`ri`) bits in the `idsts` register. The `ais` bit is a logical OR result of the fatal bus error interrupt (`fbe`), descriptor unavailable interrupt (`du`), and card error summary interrupt (`ces`) bits in the `idsts` register.

Interrupts are cleared by writing a 1 to the corresponding bit position.† If a 0 is written to an interrupt's bit position, the write is ignored, and does not clear the interrupt. When all the enabled interrupts within a group are cleared, the corresponding summary bit is set to 0. When both the summary bits are set to 0, the interrupt signal is de-asserted.†

Interrupts are not queued. If another interrupt event occurs before the driver has responded to the previous interrupt, no additional interrupts are generated. For example, the `ri` bit of the `idsts` register indicates that one or more data has been transferred to the host buffer.†

An interrupt is generated only once for simultaneous, multiple events. The driver must scan the `idsts` register for the interrupt cause.† The final interrupt signal from the controller is a logical OR of the interrupts from the BIU and internal DMA controller.

15.4.2.5.8. Internal DMA Controller Functional State Machine†



The following list explains each state of the functional state machine:†

1. The internal DMA controller performs four accesses to fetch a descriptor.†
2. The DMA controller stores the descriptor information internally. If it is the first descriptor, the controller issues a FIFO buffer reset and waits until the reset is complete.†
3. The internal DMA controller checks each bit of the descriptor for the correctness. If bit mismatches are found, the appropriate error bit is set to 1 and the descriptor is closed by setting the OWN bit in the DES0 field to 1.†

   The `rintsts` register indicates one of the following conditions:†

   - Response timeout†
   - Response CRC error†
   - Data receive timeout†
   - Response error†

4. The DMA waits for the RX watermark to be reached before writing data to system memory, or the TX watermark to be reached before reading data from system memory. The RX watermark represents the number of bytes to be locally stored in the FIFO buffer before the DMA writes to memory. The TX watermark represents the number of free bytes in the local FIFO buffer before the DMA reads data from memory.†
5. If the value of the programmable burst length (PBL) field is larger than the remaining amount of data in the buffer, single transfers are initiated. If dual buffers are being used, and the second buffer contains no data (buffer size = 0), the buffer is skipped and the descriptor is closed.†
6. The OWN bit in descriptor is set to 0 by the internal DMA controller after the data transfer for one descriptor is completed. If the transfer spans more than one descriptor, the DMA controller fetches the next descriptor. If the transfer ends with the current descriptor, the internal DMA controller goes to idle state after setting the `ri` bit or the `ti` bit of the `idsts` register. Depending on the descriptor structure (dual buffer or chained), the appropriate starting address of descriptor is loaded. If it is the second data buffer of dual buffer descriptor, the descriptor is not fetched again.†

15.4.2.6. Abort During Internal DMA Transfer

If the host issues an SD/SDIO STOP_TRANSMISSION command (CMD12) to the card while data transfer is in progress, the internal DMA controller closes the present descriptor after completing the data transfer until a Data Transfer Over (DTO) interrupt is asserted. Once a STOP_TRANSMISSION command is issued, the DMA controller performs single burst transfers.†

- For a card write operation, the internal DMA controller keeps writing data to the FIFO buffer after fetching it from the system memory until a DTO interrupt is asserted. This is done to keep the card clock running so that the STOP_TRANSMISSION command is reliably sent to the card.†
- For a card read operation, the internal DMA controller keeps reading data from the FIFO buffer and writes to the system memory until a DTO interrupt is generated. This is required because DTO interrupt is not generated until and unless all the FIFO buffer data is emptied.†

Note: For a card write abort, only the current descriptor during which a STOP_TRANSMISSION command is issued is closed by the internal DMA controller. The remaining unread descriptors are not closed by the internal DMA controller.†

Note: For a card read abort, the internal DMA controller reads the data out of the FIFO buffer and writes them to the corresponding descriptor data buffers. The remaining unread descriptors are not closed.†

15.4.2.7. Fatal Bus Error Scenarios

A fatal bus error occurs when the master interface issues an error response. This error is a system error, so the software driver must not perform any further setup on the controller. The only recovery mechanism from such scenarios is to perform one of the following tasks:†

- Issue a reset to the controller through the reset manager.†
- Issue a program controller reset by writing to the controller reset bit (`controller_reset`) of the `ctrl` register.†

15.4.2.7.1. FIFO Buffer Overflow and Underflow

During normal data transfer conditions, FIFO buffer overflow and underflow does not occur. However, if there is a programming error, a FIFO buffer overflow or underflow can result. For example, consider the following scenarios.†

For transmit:†

- PBL=4†
- TX watermark = 1†

For these programming values, if the FIFO buffer has only one location empty, the DMA attempts to read four words from memory even though there is only one word of storage available. This results in a FIFO Buffer Overflow interrupt.†

For receive:†

- PBL=4†
- RX watermark = 1†

For these programming values, if the FIFO buffer has only one location filled, the DMA attempts to write four words, even though only one word is available. This results in a FIFO Buffer Underflow interrupt.†

The driver must ensure that the number of bytes to be transferred, as indicated in the descriptor, is a multiple of four bytes. For example, if the `bytcnt` register = 13, the number of bytes indicated in the descriptor must be rounded up to 16 because the length field must always be a multiple of four bytes.†

15.4.2.7.2. PBL and Watermark Levels

This table shows legal PBL and FIFO buffer watermark values for internal DMA controller data transfer operations.†

Table 147. PBL and Watermark Levels†

PBL (Number of transfers)	TX/RX FIFO Buffer Watermark Value
1	greater than or equal to 1
4	greater than or equal to 4
8	greater than or equal to 8
16	greater than or equal to 16
32	greater than or equal to 32
64	greater than or equal to 64
128	greater than or equal to 128
256	greater than or equal to 256

15.4.3. CIU

The Card Interface Unit (CIU) interfaces with the BIU and SD/MMC cards or devices. The host processor writes command parameters to the SD/MMC controller's BIU control registers and these parameters are then passed to the CIU. Depending on control register values, the CIU generates SD/MMC command and data traffic on the

card bus according to the SD/MMC protocol. The control register values also decide whether the command and data traffic is directed to the CE-ATA card, and the SD/MMC controller controls the command and data path accordingly.†

The following list describes the CIU operation restrictions:†

- After a command is issued, the CIU accepts another command only to check read status or to stop the transfer.†
- Only one data transfer command can be issued at a time.†
- During an open-ended card write operation, if the card clock is stopped because the FIFO buffer is empty, the software must first fill the data into the FIFO buffer and start the card clock. It can then issue only an SD/SDIO STOP_TRANSMISSION (CMD12) command to the card.†
- During an SDIO/COMBO card transfer, if the card function is suspended and the software wants to resume the suspended transfer, it must first reset the FIFO buffer and start the resume command as if it were a new data transfer command.†
- When issuing SD/SDIO card reset commands (GO_IDLE_STATE, GO_INACTIVE_STATE or CMD52_reset) while a card data transfer is in progress, the software must set the stop abort command bit (`stop_abort_cmd`) in the `cmd` register to 1 so that the controller can stop the data transfer after issuing the card reset command.†
- If the card clock is stopped because the FIFO buffer is full during a card read, the software must read at least two FIFO buffer locations to start the card clock.†
- If CE-ATA card device interrupts are enabled (the `nIEN` bit is set to 0 in the ATA control register), a new RW_BLK command must not be sent to the same card device if there is a pending RW_BLK command in progress (the RW_BLK command used in this document is the RW_MULTIPLE_BLOCK MMC command defined by the CE-ATA specification). Only the Command Completion Signal Disable (CCSD) command can be sent while waiting for the Command Completion Signal (CCS).†
- For the same card device, a new command is allowed for reading status information, if interrupts are disabled in the CE-ATA card (the `nIEN` bit is set to 1 in the ATA control register).†
- Open-ended transfers are not supported for the CE-ATA card devices.†
- The `send_auto_stop` signal is not supported (software must not set the `send_auto_stop` bit in the `cmd` register) for CE-ATA transfers.†

The CIU consists of the following primary functional blocks:†

- Command path†
- Data path†
- Clock control†

15.4.3.1. Command Path

The command path performs the following functions:†

- Load card command parameters†
- Send commands to card bus†
- Receive responses from card bus†

- Send responses to BIU†
- Load clock parameters†
- Drives the P-bit on command pin†

A new command is issued to the controller by writing to the BIU registers and setting the `start_cmd` bit in the `cmd` register. The command path loads the new command (command, command argument, timeout) and sends an acknowledgement to the BIU.†

After the new command is loaded, the command path state machine sends a command to the card bus—including the internally generated seven-term CRC (CRC-7)—and receives a response, if any. The state machine then sends the received response and signals to the BIU that the command is done, and then waits for eight clock cycles before loading a new command. In CE-ATA data payload transfer (RW_MULTIPLE_BLOCK) commands, if the card device interrupts are enabled (the nIEN bit is set to 0 in the ATA control register), the state machine performs the following actions after receiving the response:†

- Does not drive the P-bit; it waits for CCS, decodes and goes back to idle state, and then drives the P-bit.†
- If the host wants to send the CCSD command and if eight clock cycles are expired after the response, it sends the CCSD pattern on the command pin.†

15.4.3.1.1. Load Command Parameters

Commands or responses are loaded in the command path in the following situations:†

- New command from BIU—When the BIU sends a new command to the CIU, the `start_cmd` bit is set to 1 in the `cmd` register.†
- Internally-generated `send_auto_stop`—When the data path ends, the SD/SDIO STOP command request is loaded.†
- Interrupt request (IRQ) response with relative card address (RCA) 0x000—When the command path is waiting for an IRQ response from the MMC and a "send irq response" request is signaled by the BIU, the send IRQ request bit (`send_irq_response`) is set to 1 in the `ctrl` register.†

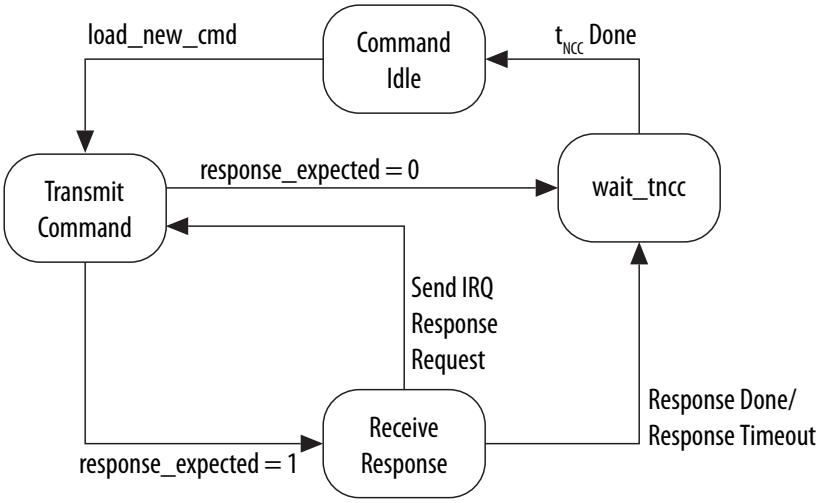
Loading a new command from the BIU in the command path depends on the following `cmd` register bit settings:†

- `update_clock_registers_only`—If this bit is set to 1 in the `cmd` register, the command path updates only the `clkena`, `clkdiv`, and `clksrc` registers. If this bit is set to 0, the command path loads the `cmd`, `cmdarg`, and `tmout` registers. It then processes the new command, which is sent to the card.†
- `wait_prvdata_complete`—If this bit is set to 1, the command path loads the new command under one of the following conditions:†
  - Immediately, if the data path is free (that is, there is no data transfer in progress), or if an open-ended data transfer is in progress (`bytcnt` = 0).†
  - After completion of the current data transfer, if a predefined data transfer is in progress.†

15.4.3.1.2. Send Command and Receive Response

After a new command is loaded in the command path (the `update_clock_registers_only` bit in the `cmd` register is set to 0), the command path state machine sends out a command on the card bus.†

Figure 69. Command Path State Machine†



The command path state machine performs the following functions, according to `cmd` register bit values:†

1. `send_initialization`—Initialization sequence of 80 clock cycles is sent before sending the command.†
2. `response_expected`—A response is expected for the command. After the command is sent out, the command path state machine receives a 48-bit or 136-bit response and sends it to the BIU. If the start bit of the card response is not received within the number of clock cycles (as set up in the `tmout` register), the `rto` bit and command done (`CD`) bit are set to 1 in the `rintsts` register, to signal to the BIU. If the response-expected bit is set to 0, the command path sends out a command and signals a response done to the BIU, which causes the `cmd` bit to be set to 1 in the `rintsts` register.†
3. `response_length`—If this bit is set to 1, a 136-bit long response is received; if it is set to 0, a 48-bit short response is received.†
4. `check_response_crc`—If this bit is set to 1, the command path compares CRC-7 received in the response with the internally-generated CRC-7. If the two do not match, the response CRC error is signaled to the BIU, that is, the `rcrc` bit is set to 1 in the `rintsts` register.†

15.4.3.1.3. Send Response to BIU

If the `response_expected` bit is set to 1 in the `cmd` register, the received response is sent to the BIU. Response register 0 (`resp0`) is updated for a short response, and the response register 3 (`resp3`), response register 2 (`resp2`), response register 1 (`resp1`), and `resp0` registers are updated on a long response, after which the `cmd` bit

is set to 1 in the `rintsts` register. If the response is for an AUTO_STOP command sent by the CIU, the response is written to the `resp1` register, after which the auto command done bit (`acd`) is set to 1 in the `rintsts` register.†

The command path verifies the contents of the card response.

Table 148. Card Response Fields†

Field	Contents
Response transmission bit	0
Command index	Command index of the sent command
End bit	1

The command index is not checked for a 136-bit response or if the `check_response_crc` bit in the `cmd` register is set to 0. For a 136-bit response and reserved CRC 48-bit responses, the command index is reserved, that is, 0b111111.†

Related Information

SD Association
For more information about response values, refer to Physical Layer Simplified Specification, Version 3.01 as described on the SD Association website.

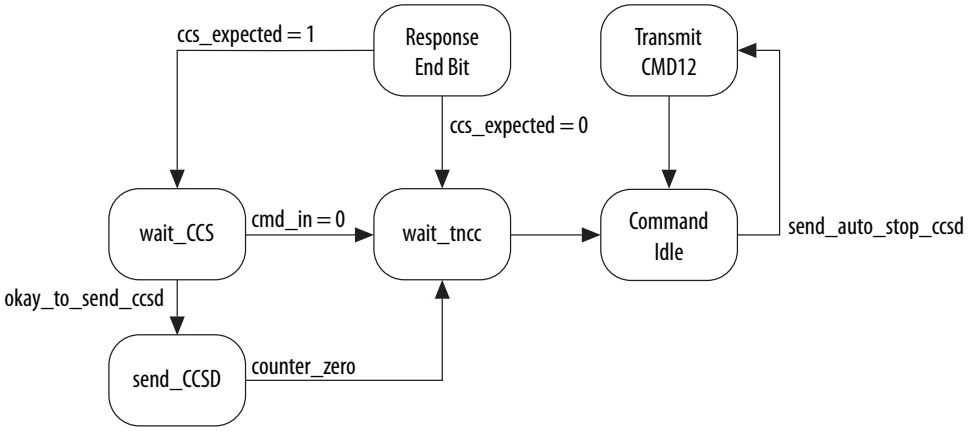
15.4.3.1.4. Driving P-bit to the CMD Pin

The command path drives a one-cycle pull-up bit (P-bit) to 1 on the CMD pin between two commands if a response is not expected. If a response is expected, the P-bit is driven after the response is received and before the start of the next command. While accessing a CE-ATA card device, for commands that expect a CCS, the P-bit is driven after the response only if the interrupts are disabled in the CE-ATA card (the `nIEN` bit is set to 1 in the ATA control register), that is, the CCS expected bit (`ccs_expected`) in the `cmd` register is set to 0. If the command expects the CCS, the P-bit is driven only after receiving the CCS.†

15.4.3.1.5. Polling the CCS

CE-ATA card devices generate the CCS to notify the host controller of the normal ATA command completion or ATA command termination. After receiving the response from the card, the command path state machine performs the functions illustrated in the following figure according to `cmd` register bit values.†

Figure 70. CE-ATA Command Path State Machine†



The above figure illustrates:

- Response end bit state—The state machine receives the end bit of the response from the card device. If the `ccs_expected` bit of the `cmd` register is set to 1, the state machine enters the wait CCS state.†
- Wait CCS—The state machine waits for the CCS from the CE-ATA card device. While waiting for the CCS, the following events can happen:†
  1. Software sets the send CCSD bit (`send_ccsd`) in the `ctrl` register, indicating not to wait for CCS and to send the CCSD pattern on the command line.†
  2. Receive the CCS on the CMD line.†
- Send CCSD command—Sends the CCSD pattern (0b00001) on the CMD line.†

15.4.3.1.6. CCS Detection and Interrupt to Host Processor

If the `ccs_expected` bit in the `cmd` register is set to 1, the CCS from the CE-ATA card device is indicated by setting the data transfer over bit (`dto`) in the `rintsts` register. The controller generates a DTO interrupt if this interrupt is not masked.†

For the RW_MULTIPLE_BLOCK commands, if the CE-ATA card device interrupts are disabled (the `nIEN` bit is set to 1 in the ATA control register)— that is, the `ccs_expected` bit is set to 0 in the `cmd` register—there are no CCSs from the card. When the data transfer is over—that is, when the requested number of bytes are transferred—the `dto` bit in the `rintsts` register is set to 1.†

15.4.3.1.7. CCS Timeout

If the command expects a CCS from the card device (the `ccs_expected` bit is set to 1 in the `cmd` register), the command state machine waits for the CCS and remains in the wait CCS state. If the CE-ATA card fails to send out the CCS, the host software must implement a timeout mechanism to free the command and data path. The controller does not implement a hardware timer; it is the responsibility of the host software to maintain a software timer.†

In the event of a CCS timeout, the host must issue a CCSD command by setting the `send_ccsd` bit in the `ctrl` register. The controller command path state machine sends the CCSD command to the CE-ATA card device and exits to an idle state. After sending the CCSD command, the host must also send an SD/SDIO STOP_TRANSMISSION command to the CE-ATA card to abort the outstanding ATA command.†

15.4.3.1.8. Send CCSD Command

If the `send_ccsd` bit in the `ctrl` register is set to 1, the controller sends a CCSD pattern on the CMD line. The host can send the CCSD command while waiting for the CCS or after a CCS timeout happens.†

After sending the CCSD pattern, the controller sets the `cmd` bit in the `rintsts` register and also generates an interrupt to the host if the Command Done interrupt is not masked.†

Note: Within the CIU block, if the `send_ccsd` bit in the `ctrl` register is set to 1 on the same clock cycle as CCS is sampled, the CIU block does not send a CCSD pattern on the CMD line. In this case, the `dto` and `cmd` bits in the `rintsts` register are set to 1.†

Note: Due to asynchronous boundaries, the CCS might have already happened and the `send_ccsd` bit is set to 1. In this case, the CCSD command does not go to the CE-ATA card device and the `send_ccsd` bit is not set to 0. The host must reset the `send_ccsd` bit to 0 before the next command is issued.†

If the send auto stop CCSD (`send_auto_stop_ccsd`) bit in the `ctrl` register is set to 1, the controller sends an internally generated STOP_TRANSMISSION command (CMD12) after sending the CCSD pattern. The controller sets the `acd` bit in the `rintsts` register.†

15.4.3.1.9. I/O transmission delay (N_{ACIO} Timeout)

The host software maintains the timeout mechanism for handling the I/O transmission delay (N_{ACIO} cycles) time-outs while reading from the CE-ATA card device. The controller neither maintains any timeout mechanism nor indicates that N_{ACIO} cycles are elapsed while waiting for the start bit of a data token. The I/O transmission delay is applicable for read transfers using the RW_REG and RW_BLK commands; the RW_REG and RW_BLK commands used in this document refer to the RW_MULTIPLE_REGISTER and RW_MULTIPLE_BLOCK MMC commands defined by the CE-ATA specification.†

Note: After the N_{ACIO} timeout, the application must abort the command by sending the CCSD and STOP commands, or the STOP command. The Data Read Timeout (DRTO) interrupt might be set to 1 while a STOP_TRANSMISSION command is transmitted out of the controller, in which case the data read timeout boot data start bit (`bds`) and the `dto` bit in the `rintsts` register are set to 1.†

15.4.3.2. Data Path

The data path block reads the data FIFO buffer and transmits data on the card bus during a write data transfer, or receives data and writes it to the FIFO buffer during a read data transfer. The data path loads new data parameters—data expected, read/write data transfer, stream/block transfer, block size, byte count, card type, timeout

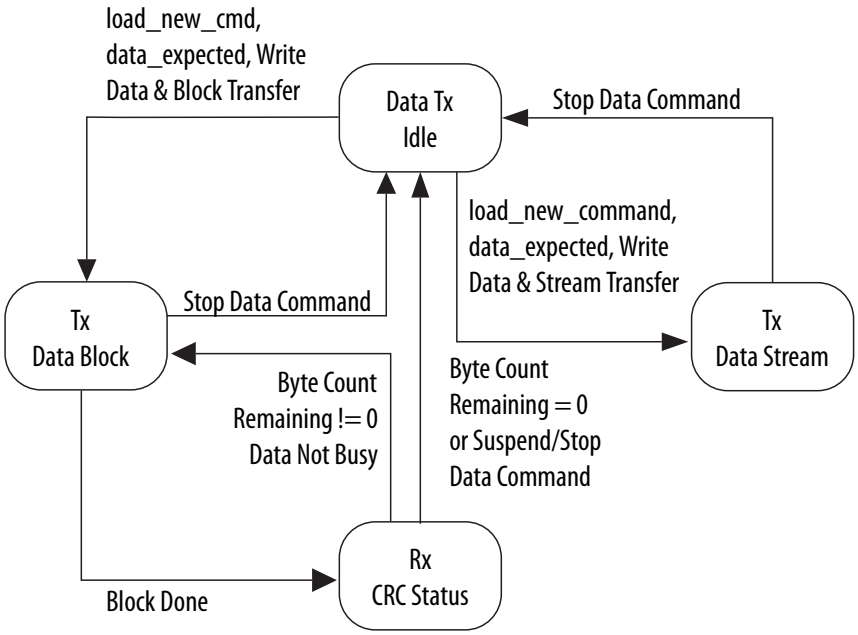
registers—whenever a data transfer command is not in progress. If the data transfer expected bit (`data_expected`) in the `cmd` register is set to 1, the new command is a data transfer command and the data path starts one of the following actions:†

- Transmits data if the read/write bit = 1†
- Receives data if read/write bit = 0†

15.4.3.2.1. Data Transmit

The data transmit state machine starts data transmission two clock cycles after a response for the data write command is received. This occurs even if the command path detects a response error or response CRC error. If a response is not received from the card because of a response timeout, data is not transmitted. Depending upon the value of the transfer mode bit (`transfer_mode`) in the `cmd` register, the data transmit state machine puts data on the card data bus in a stream or in blocks.†

Figure 71. Data Transmit State Machine†



Stream Data Transmit

If the `transfer_mode` bit in the `cmd` register is set to 1, the transfer is a stream-write data transfer. The data path reads data from the FIFO buffer from the BIU and transmits in a stream to the card data bus. If the FIFO buffer becomes empty, the card clock is stopped and restarted once data is available in the FIFO buffer.†

If the `bytcnt` register is reset to 0, the transfer is an open-ended stream-write data transfer. During this data transfer, the data path continuously transmits data in a stream until the host software issues an SD/SDIO STOP command. A stream data transfer is terminated when the end bit of the STOP command and end bit of the data match over two clock cycles.†

If the `bytcnt` register is written with a nonzero value and the `send_auto_stop` bit in the `cmd` register is set to 1, the STOP command is internally generated and loaded in the command path when the end bit of the STOP command occurs after the last

byte of the stream write transfer matches. This data transfer can also terminate if the host issues a STOP command before all the data bytes are transferred to the card bus.†

Single Block Data

If the `transfer_mode` bit in the `cmd` register is set to 0 and the `bytcnt` register value is equal to the value of the `block_size` register, a single-block write-data transfer occurs. The data transmit state machine sends data in a single block, where the number of bytes equals the block size, including the internally-generated 16-term CRC (CRC-16).†

If the `ctype` register is set for a 1-bit, 4-bit, or 8-bit data transfer, the data is transmitted on 1, 4, or 8 data lines, respectively, and CRC-16 is separately generated and transmitted for 1, 4, or 8 data lines, respectively.†

After a single data block is transmitted, the data transmit state machine receives the CRC status from the card and signals a data transfer to the BIU. This happens when the `dto` bit in the `rintsts` register is set to 1.†

If a negative CRC status is received from the card, the data path signals a data CRC error to the BIU by setting the `dcrc` bit in the `rintsts` register.†

Additionally, if the start bit of the CRC status is not received by two clock cycles after the end of the data block, a CRC status start-bit error (SBE) is signaled to the BIU by setting the `sbe` bit in the `rintsts` register.†

Multiple Block Data

A multiple-block write-data transfer occurs if the `transfer_mode` bit in the `cmd` register is set to 0 and the value in the `bytcnt` register is not equal to the value of the `block_size` register. The data transmit state machine sends data in blocks, where the number of bytes in a block equals the block size, including the internally-generated CRC-16 value.†

If the `ctype` register is set to 1-bit, 4-bit, or 8-bit data transfer, the data is transmitted on 1-, 4-, or 8-data lines, respectively, and CRC-16 is separately generated and transmitted on 1-, 4-, or 8-data lines, respectively.†

After one data block is transmitted, the data transmit state machine receives the CRC status from the card. If the remaining byte count becomes 0, the data path signals to the BIU that the data transfer is done. This happens when the `dto` bit in the `rintsts` register is set to 1.†

If the remaining data bytes are greater than zero, the data path state machine starts to transmit another data block.†

If a negative CRC status is received from the card, the data path signals a data CRC error to the BIU by setting the `dcrc` bit in the `rintsts` register, and continues further data transmission until all the bytes are transmitted.†

If the CRC status start bit is not received by two clock cycles after the end of a data block, a CRC status SBE is signaled to the BIU by setting the `ebe` bit in the `rintsts` register and further data transfer is terminated.†

If the send_auto_stop bit is set to 1 in the cmd register, the SD/SDIO STOP command is internally generated during the transfer of the last data block, where no extra bytes are transferred to the card. The end bit of the STOP command might not exactly match the end bit of the CRC status in the last data block.†

If the block size is less than 4, 16, or 32 for card data widths of 1 bit, 4 bits, or 8 bits, respectively, the data transmit state machine terminates the data transfer when all the data is transferred, at which time the internally-generated STOP command is loaded in the command path.†

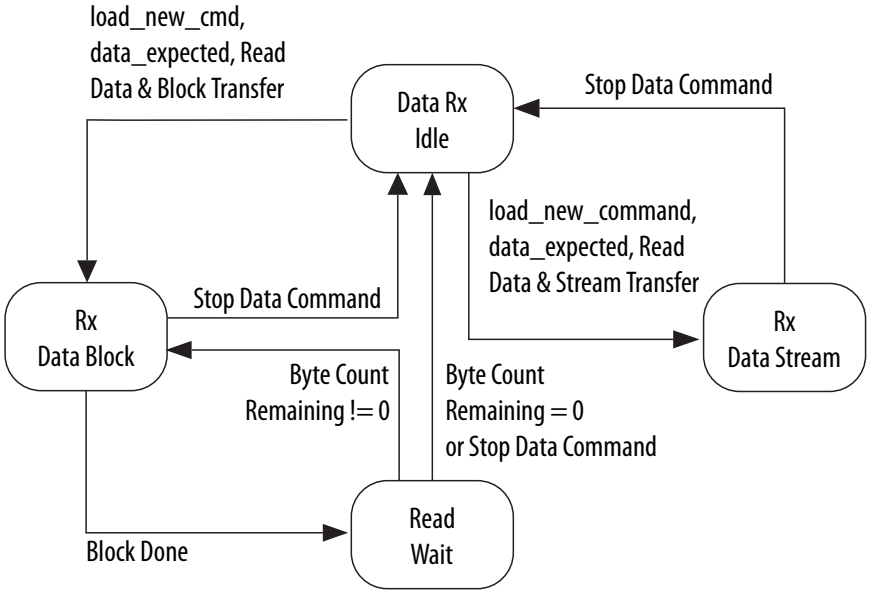
If the bytcnt is zero (the block size must be greater than zero) the transfer is an open-ended block transfer. The data transmit state machine for this type of data transfer continues the block-write data transfer until the host software issues an SD/SDIO STOP or STOP_TRANSMISSION (CMD12) command.†

15.4.3.2.2. Data Receive

The data-receive state machine receives data two clock cycles after the end bit of a data read command, even if the command path detects a response error or response CRC error. If a response is not received from the card because a response timeout occurs, the BIU does not receive a signal that the data transfer is complete. This happens if the command sent by the controller is an illegal operation for the card, which keeps the card from starting a read data transfer.†

If data is not received before the data timeout, the data path signals a data timeout to the BIU and an end to the data transfer done. Based on the value of the transfer_mode bit in the cmd register, the data-receive state machine gets data from the card data bus in a stream or block(s).†

Figure 72. Data Receive State Machine†



Stream Data Read

A stream-read data transfer occurs if the `transfer_mode` bit in the `cmd` register is set to 1, at which time the data path receives data from the card and writes it to the FIFO buffer. If the FIFO buffer becomes full, the card clock stops and restarts once the FIFO buffer is no longer full.†

An open-ended stream-read data transfer occurs if the `bytcnt` register is set to 0. During this type of data transfer, the data path continuously receives data in a stream until the host software issues an SD/SDIO STOP command. A stream data transfer terminates two clock cycles after the end bit of the STOP command.†

If the `bytcnt` register contains a nonzero value and the `send_auto_stop` bit in the `cmd` register is set to 1, a STOP command is internally generated and loaded into the command path, where the end bit of the STOP command occurs after the last byte of the stream data transfer is received. This data transfer can terminate if the host issues an SD/SDIO STOP or STOP_TRANSMISSION (CMD12) command before all the data bytes are received from the card.†

Single-block Data Read

If the `ctype` register is set to a 1-bit, 4-bit, or 8-bit data transfer, data is received from 1, 4, or 8 data lines, respectively, and CRC-16 is separately generated and checked for 1, 4, or 8 data lines, respectively. If there is a CRC-16 mismatch, the data path signals a data CRC error to the BIU. If the received end bit is not 1, the BIU receives an End-bit Error (EBE).†

Multiple-block Data Read

If the `transfer_mode` bit in the `cmd` register is clear and the value of the `bytcnt` register is not equal to the value of the `block_size` register, the transfer is a multiple-block read-data transfer. The data-receive state machine receives data in blocks, where the number of bytes in a block is equal to the block size, including the internally-generated CRC-16.†

If the `ctype` register is set to a 1-bit, 4-bit, or 8-bit data transfer, data is received from 1, 4, or 8 data lines, respectively, and CRC-16 is separately generated and checked for 1, 4, or 8 data lines, respectively. After a data block is received, if the remaining byte count becomes zero, the data path signals a data transfer to the BIU.†

If the remaining data bytes are greater than zero, the data path state machine causes another data block to be received. If CRC-16 of a received data block does not match the internally-generated CRC-16, a data CRC error to the BIU and data reception continue further data transmission until all bytes are transmitted. Additionally, if the end of a received data block is not 1, data on the data path signals terminate the bit error to the CIU and the data-receive state machine terminates data reception, waits for data timeout, and signals to the BIU that the data transfer is complete.†

If the `send_auto_stop` bit in the `cmd` register is set to 1, the SD/SDIO STOP command is internally generated when the last data block is transferred, where no extra bytes are transferred from the card. The end bit of the STOP command might not exactly match the end bit of the last data block.†

If the requested block size for data transfers to cards is less than 4, 16, or 32 bytes for 1-bit, 4-bit, or 8-bit data transfer modes, respectively, the data-transmit state machine terminates the data transfer when all data is transferred, at which point the internally-generated STOP command is loaded in the command path. Data received from the card after that are then ignored by the data path.†

If the `bytcnt` register is 0 (the block size must be greater than zero), the transfer is an open-ended block transfer. For this type of data transfer, the data-receive state machine continues the block-read data transfer until the host software issues an SD/SDIO STOP or STOP_TRANSMISSION (CMD12) command.†

15.4.3.2.3. Auto-Stop

The controller internally generates an SD/SDIO STOP command and is loaded in the command path when the `send_auto_stop` bit in the `cmd` register is set to 1. The AUTO_STOP command helps to send an exact number of data bytes using a stream read or write for the MMC, and a multiple-block read or write for SD memory transfer for SD cards. The software must set the `send_auto_stop` bit according to the following details: †

The following list describes conditions for the AUTO_STOP command:†

- Stream-read for MMC with byte count greater than zero—The controller generates an internal STOP command and loads it into the command path so that the end bit of the STOP command is sent when the last byte of data is read from the card and no extra data byte is received. If the byte count is less than six (48 bits), a few extra data bytes are received from the card before the end bit of the STOP command is sent.†
- Stream-write for MMC with byte count greater than zero—The controller generates an internal STOP command and loads it into the command path so that the end bit of the STOP command is sent when the last byte of data is transmitted on the card bus and no extra data byte is transmitted. If the byte count is less than six (48 bits), the data path transmits the data last to meet these condition.†
- Multiple-block read memory for SD card with byte count greater than zero—If the block size is less than four (single-bit data bus), 16 (4-bit data bus), or 32 (8-bit data bus), the AUTO_STOP command is loaded in the command path after all the bytes are read. Otherwise, the STOP command is loaded in the command path so that the end bit of the STOP command is sent after the last data block is received.†
- Multiple-block write memory for SD card with byte count greater than zero—If the block size is less than three (single-bit data bus), 12 (4-bit data bus), or 24 (8-bit data bus), the AUTO_STOP command is loaded in the command path after all data blocks are transmitted. Otherwise, the STOP command is loaded in the command path so that the end bit of the STOP command is sent after the end bit of the CRC status is received.†
- Precaution for host software during auto-stop—When an AUTO_STOP command is issued, the host software must not issue a new command to the controller until the AUTO_STOP command is sent by the controller and the data transfer is complete. If the host issues a new command during a data transfer with the AUTO_STOP command in progress, an AUTO_STOP command might be sent after the new command is sent and its response is received. This can delay sending the STOP command, which transfers extra data bytes. For a stream write, extra data bytes are erroneous data that can corrupt the card data. If the host wants to terminate the data transfer before the data transfer is complete, it can issue an SD/SDIO STOP or STOP_TRANSMISSION (CMD12) command, in which case the controller does not generate an AUTO_STOP command.†

Auto-Stop Generation for MMC Cards

Table 149. Auto-Stop Generation for MMC Cards†

Transfer Type	Byte Count	send_auto_stop bit set	Comments
Stream read	0	No	Open-ended stream
Stream read	>0	Yes	Auto-stop after all bytes transfer
Stream write	0	No	Open-ended stream
Stream write	>0	Yes	Auto-stop after all bytes transfer
Single-block read	>0	No	Byte count = 0 is illegal
Single-block write	>0	No	Byte count = 0 is illegal
Multiple-block read	0	No	Open-ended multiple block
Multiple-block read	>0	Yes(42)†	Pre-defined multiple block
Multiple-block write	0	No	Open-ended multiple block
Multiple-block write	>0	Yes (42)†	Pre-defined multiple block

Auto-Stop Generation for SD Cards

Table 150. Auto-Stop Generation for SD Cards†

Transfer Type	Byte Count	send_auto_stop bit set	Comments
Single-block read	>0	No	Byte count = 0 is illegal
Single-block write	>0	No	Byte count = 0 illegal
Multiple-block read	0	No	Open-ended multiple block
Multiple-block read	>0	Yes	Auto-stop after all bytes transfer
Multiple-block write	0	No	Open-ended multiple block
Multiple-block write	>0	Yes	Auto-stop after all bytes transfer

Auto-Stop Generation for SDIO Cards

Table 151. Auto-Stop Generation for SDIO Cards†

Transfer Type	Byte Count	send_auto_stop bit set	Comments
Single-block read	>0	No	Byte count = 0 is illegal
Single-block write	>0	No	Byte count = 0 illegal
Multiple-block read	0	No	Open-ended multiple block

continued...

(42) The condition under which the transfer mode is set to block transfer and byte_count is equal to block size is treated as a single-block data transfer command for both MMC and SD cards. If byte_count = n*block_size (n = 2, 3, …), the condition is treated as a predefined multiple-block data transfer command. In the case of an MMC card, the host software can perform a predefined data transfer in two ways: 1) Issue the CMD23 command before issuing CMD18/CMD25 commands to the card – in this case, issue CMD18/CMD25 commands without setting the send_auto_stop bit. 2) Issue CMD18/CMD25 commands without issuing CMD23 command to the card, with the send_auto_stop bit set. In this case, the multiple-block data transfer is terminated by an internally-generated auto-stop command after the programmed byte count.†

Transfer Type	Byte Count	send_auto_stop bit set	Comments
Multiple-block read	>0	No	Pre-defined multiple block
Multiple-block write	0	No	Open-ended multiple block
Multiple-block write	>0	No	Pre-defined multiple block

15.4.3.2.4. Non-Data Transfer Commands that Use Data Path

Some SD/SDIO non-data transfer commands (commands other than read and write commands) also use the data path.

Table 152. Non-Data Transfer Commands and Requirements†

	PROGRAM_CSD (CMD27)	SEND_WRITE_PROT (CMD30)	LOCK_UNLOCK (CMD42)	SD_STATUS (ACMD13)	SEND_NUM_WR_BLOCKS (ACMD22)	SEND_SCR (ACMD51)
Command register programming†						
Cmd_index	0x1B=27	0x1E=30	0x2A=42	0x0D=13	0x16=22	0x33=51
Response_expect†	1	1	1	1	1	1
Response_length†	0	0	0	0	0	0
Check_response_crc†	1	1	1	1	1	1
Data_expected†	1	1	1	1	1	1
Read/write†	1	0	1	0	0	0
Transfer_mode†	0	0	0	0	0	0
Send_auto_stop†	0	0	0	0	0	0
Wait_prevdata_complete†	0	0	0	0	0	0
Stop_abort_cmd†	0	0	0	0	0	0

Table 153. Non-Data Transfer Commands and Requirements (Cont.)†

	PROGRAM_CSD (CMD27)	SEND_WRITE_PROT (CMD30)	LOCK_UNLOCK (CMD42)	SD_STATUS (ACMD13)	SEND_NUM_WR_BLOCKS (ACMD22)	SEND_SCR (ACMD51)
Command Argument register programming†						
	Stuff bits	32-bit write protect data address	Stuff bits	Stuff bits	Stuff bits	Stuff bits

Table 154. Non-Data Transfer Commands and Requirements†

PROGRAM_CSD (CMD27)	SEND_WRITE_PROT (CMD30)	LOCK_UNLOCK (CMD42)	SD_STATUS (ACMD13)	SEND_NUM_WR_BLOCKS (ACMD22)	SEND_SCR (ACMD51)
Block Size register programing†					
16	4	Num_bytes(43)	64	4	8

Table 155. Non-Data Transfer Commands and Requirements†

PROGRAM_CSD (CMD27)	SEND_WRITE_PROT (CMD30)	LOCK_UNLOCK (CMD42)	SD_STATUS (ACMD13)	SEND_NUM_WR_BLOCKS (ACMD22)	SEND_SCR (ACMD51)
Byte Count register programming†					
16	4	Num_bytes(44)	64	4	8

Related Information

- SD Association
  For more information, the SD specification can be purchased from this organization.
- JEDEC Global Standards of the Microelectronics Industry
  For more information, the MMC specification can be purchased from this organization.

15.4.3.3. Clock Control Block

The clock control block provides different clock frequencies required for SD/MMC/CE-ATA cards. The clock control block has one clock divider, which is used to generate different card clock frequencies.†

The clock frequency of a card depends on the following clock `ctrl` register settings:†

- `clkdiv` register—Internal clock dividers are used to generate different clock frequencies required for the cards. The division factor for the clock divider can be set by writing to the `clkdiv` register. The clock divider is an 8-bit value that provides a clock division factor from 1 to 510; a value of 0 represents a clock-divider bypass, a value of 1 represents a divide by 2, a value of 2 represents a divide by 4, and so on.†
- `clksrc` register—Set this register to 0 as clock is divided by clock divider 0.†
- `clkena` register—The `cclk_out` card output clock can be enabled or disabled under the following conditions:†
  - `cclk_out` is enabled when the `cclk_enable` bit in the `clkena` register is set to 1 and disabled when set to 0.†
  - Low-power mode can be enabled by setting the `cclk_low_power` bit of the `clkena` register to 1. If low-power mode is enabled to save card power, the `cclk_out` signal is disabled when the card is idle for at least eight card clock cycles. Low-power mode is enabled when a new command is loaded and the command path goes to a non-idle state.†

(43) Num_bytes = Number of bytes specified as per the lock card data structure. Refer to the SD specification and the MMC specification.†

(44) Num_bytes = Number of bytes specified as per the lock card data structure. Refer to the SD specification and the MMC specification.†

Under the following conditions, the card clock is stopped or disabled:†

- Clock can be disabled by writing to the `clkena` register.†
- When low-power mode is selected and the card is idle for at least eight clock cycles.†
- FIFO buffer is full, data path cannot accept more data from the card, and data transfer is incomplete—to avoid FIFO buffer overflow.†
- FIFO buffer is empty, data path cannot transmit more data to the card, and data transfer is incomplete—to avoid FIFO buffer underflow.†

Note: The card clock must be disabled through the `clkena` register before the host software changes the values of the `clkdiv` and `clksrc` registers.†

15.4.3.4. Error Detection

Errors can occur during card operations within the CIU in the following situations.

15.4.3.4.1. Response†

- Response timeout—did not receive the response expected with response start bit within the specified number of clock cycles in the timeout register.†
- Response CRC error—response is expected and check response CRC requested; response CRC-7 does not match with the internally-generated CRC-7.†
- Response error—response transmission bit is not 0, command index does not match with the command index of the send command, or response end bit is not 1.†

15.4.3.4.2. Data Transmit†

- No CRC status—during a write data transfer, if the CRC status start bit is not received for two clock cycles after the end bit of the data block is sent out, the data path performs the following actions:†
  - Signals no CRC status error to the BIU†
  - Terminates further data transfer†
  - Signals data transfer done to the BIU†
- Negative CRC—if the CRC status received after the write data block is negative (that is, not 0b010), the data path signals a data CRC error to the BIU and continues with the data transfer.†
- Data starvation due to empty FIFO buffer—if the FIFO buffer becomes empty during a write data transmission, or if the card clock stopped and the FIFO buffer remains empty for a data-timeout number of clock cycles, the data path signals a data-starvation error to the BIU and the data path continues to wait for data in the FIFO buffer.†

15.4.3.4.3. Data Receive

- Data timeout—during a read-data transfer, if the data start bit is not received before the number of clock cycles specified in the timeout register, the data path does the following action: †
  - — Signals a data-timeout error to the BIU†
  - — Terminates further data transfer†
  - — Signals data transfer done to BIU†
- Data SBE—during a 4-bit or 8-bit read-data transfer, if the all-bit data line does not have a start bit, the data path signals a data SBE to the BIU and waits for a data timeout, after which it signals that the data transfer is done.†
- Data CRC error—during a read-data-block transfer, if the CRC-16 received does not match with the internally generated CRC-16, the data path signals a data CRC error to the BIU and continues with the data transfer.†
- Data EBE—during a read-data transfer, if the end bit of the received data is not 1, the data path signals an EBE to the BIU, terminates further data transfer, and signals to the BIU that the data transfer is done.†
- Data starvation due to FIFO buffer full—during a read data transmission and when the FIFO buffer becomes full, the card clock stops. If the FIFO buffer remains full for a data-timeout number of clock cycles, the data path signals a data starvation error to the BIU, by setting the data starvation host timeout bit (`hto`) in `rintsts` register to 1, and the data path continues to wait for the FIFO buffer to empty.†

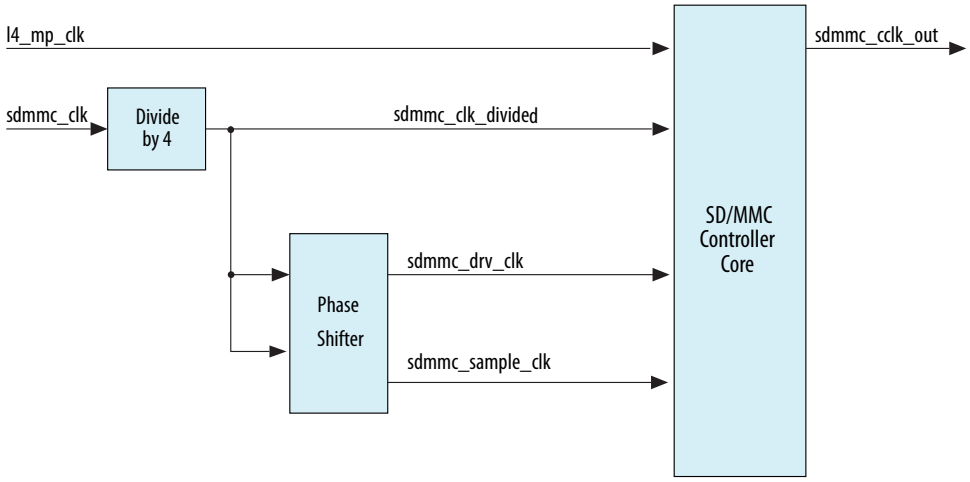
15.4.4. Clocks

Table 156. SD/MMC Controller Clocks

Clock Name	Direction	Description
`l4_mp_clk`	In	Clock for SD/MMC controller BIU
`sdmmc_clk`	In	Clock for SD/MMC controller
`sdmmc_cclk_out`	Out	Generated output clock for card
`sdmmc_clk_divided`	Internal	Divide-by-four clock of `sdmmc_clk`
`sdmmc_sample_clk`	Internal	Phase-shifted clock of `sdmmc_clk_divided` used to sample the command and data from the card
`sdmmc_drv_clk`	Internal	Phase-shifted clock of `sdmmc_clk_divided` for controller to drive command and data to the card to meet hold time requirements

Figure 73. SD/MMC Controller Clock Connections - HPS

This figure displays the controller clock connections from the HPS.



The `sdmmc_clk` clock from the clock manager is divided by four and becomes the `sdmmc_clk_divided` clock before passing to the phase shifters and the SD/MMC controller CIU. The phase shifters are used to generate the `sdmmc_drv_clk` and `sdmmc_sample_clk` clocks. These phase shifters provide up to eight phases shift which include 0, 45, 90, 135, 180, 225, 270, and 315 degrees. The `sdmmc_sample_clk` clock can be driven by the output from the phase shifter.

Note: The selections of phase shift degree and `sdmmc_sample_clk` source are done in the system manager. For information about setting the phase shift and selecting the source of the `sdmmc_sample_clk` clock, refer to the "Clock Setup" section within this document.

The controller generates the `sdmmc_cclk_out` clock, which is driven to the card. For more information about the generation of the `sdmmc_cclk_out` clock, refer to the "Clock Control Block" section within this document.

Related Information

- Clock Setup on page 449
  Refer to this section for information about setting the phase shift.
- Clock Control Block on page 436
  Refer to this section for information about the generation of the `sdmmc_cclk_out`clock.

15.4.5. Resets

The SD/MMC controller has one reset signal. The reset manager drives this signal to the SD/MMC controller on a cold or warm reset.

Related Information

Reset Manager on page 134

15.4.5.1. Taking the SD/MMC Controller Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

Related Information

Module Reset Signals on page 140

15.4.6. Voltage Switching

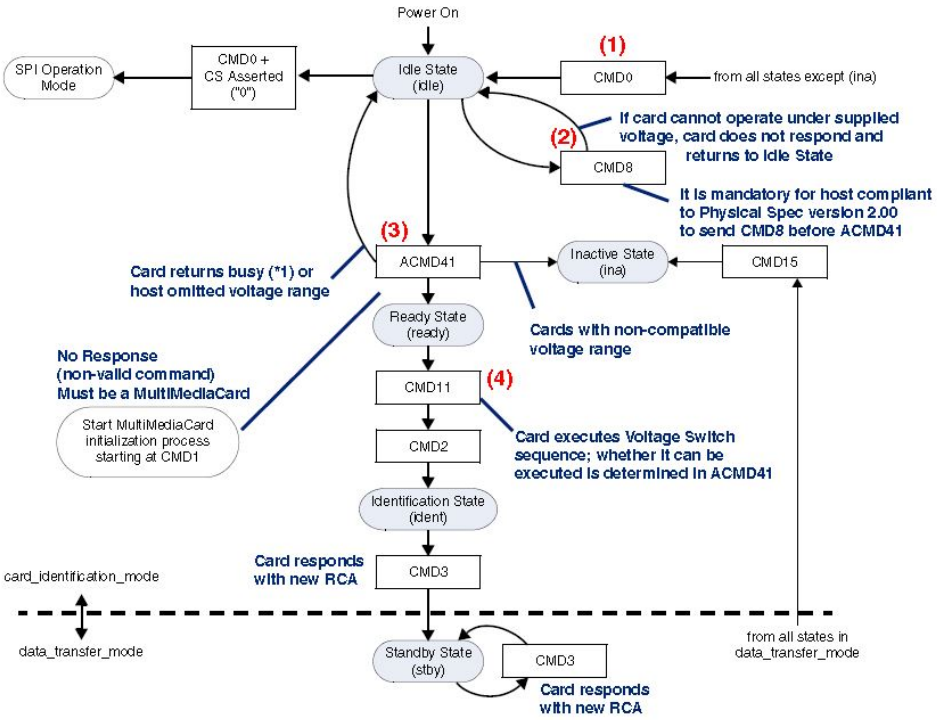
This section describes the general steps to switch voltage level.

The SD/MMC cards support various operating voltages, for example 1.8V and 3.0V. If you have a card which is at 1.8V and you eject it and replace it with another card, which is 3.0V, then voltage switching is required.

Many SD cards have an option to signal at 1.8 or 3.0 V, however the initial power-up voltage requirement is 3.0V. To support these different voltage requirements, external transceivers are needed.

The general steps to switch the voltage level requires you to use a SD/MMC voltage-translation transceiver in between the HPS and the SD/MMC card.

Figure 74. Voltage Switching Command Flow Diagram†



(*1) **Note:** Card returns busy when:†

- Card executes internal initialization process†
- Card is High or Extended capacity SD Memory Card and host does not support High†

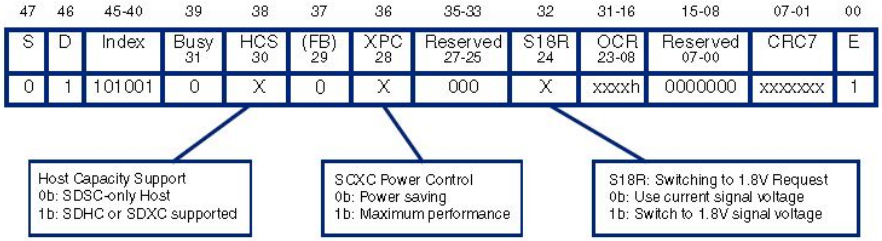
The following outlines the steps for the voltage switch programming sequence.†

1. Software Driver starts CMD0, which selects the bus mode as SD.†
2. After the bus is in SD card mode, CMD8 is started in order to verify if the card is compatible with the SD Memory Card Specification, Version 2.00.†

   CMD8 determines if the card is capable of working within the host supply voltage specified in the VHS (19:16) field of the CMD; the card supports the current host voltage if a response to CMD8 is received.†
3. ACMD 41 is started.†

   The response to this command informs the software if the card supports voltage switching; bits 38, 36, and 32 are checked by the card argument of ACMD41.†

Figure 75. ACMD41 Argument

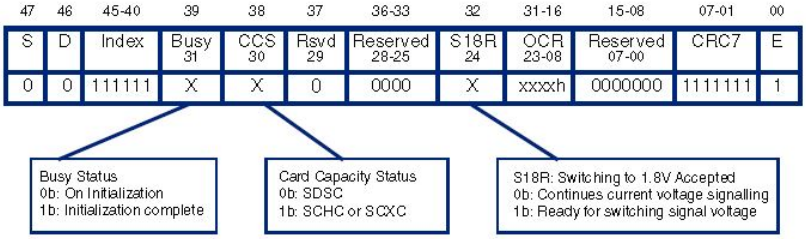


a. Bit 30 informs the card if host supports SDHC/SDXC or not; this bit should be set to 1'b1.†

b. Bit 28 can be either 1 or 0.†

c. Bit 24 should be set to 1'b1, indicating that the host is capable of voltage switching.†

Figure 76. ACMD41 Response (R3)†



d. Bit 30 – If set to 1'b1, card supports SDHC/SDXC; if set to 1'b0, card supports only SDSC.†

e. Bit 24 – If set to 1'b1, card supports voltage switching and is ready for the switch.†

f. Bit 31 – If set to 1'b1, initialization is over; if set to 1'b0, means initialization in process†

4. If the card supports voltage switching, then the software must perform the steps discussed for either the "Voltage Switch Normal Scenario" or the "Voltage Switch Error Scenario", located in the *Synopsys DesignWare Cores Mobile Storage Host Databook*.

Related Information

Synopsys DesignWare Cores Mobile Storage Host Databook
For more information about Voltage Switching

15.5. SD/MMC Controller Programming Model

15.5.1. Software and Hardware Restrictions†

Only one data transfer command should be issued at one time. For CE-ATA devices, if CE-ATA device interrupts are enabled (nIEN=0), only one RW_MULTIPLE_BLOCK command (RW_BLK) should be issued; no other commands (including a new RW_BLK) should be issued before the Data Transfer Over status is set for the outstanding RW_BLK.†

Before issuing a new data transfer command, the software should ensure that the card is not busy due to any previous data transfer command. Before changing the card clock frequency, the software must ensure that there are no data or command transfers in progress.†

If the card is enumerated in SDR12 or SDR25 mode, the application must program the `use_hold_reg` bit[29] in the CMD register to 1'b1.†

This programming should be done for all data transfer commands and non-data commands that are sent to the card. When the `use_hold_reg` bit is programmed to 1'b0, the SD/MMC controller bypasses the Hold Registers in the transmit path. The value of this bit should not be changed when a Command or Data Transfer is in progress.†

For more information on using the `use_hold_reg` and the implementation requirements for meeting the card input hold time, refer to the latest version of the Synopsys DesignWare Cores Mobile Storage Host Databook.

15.5.1.1. Avoiding Glitches in the Card Clock Outputs†

To avoid glitches in the card clock outputs (`sdmmc_cclk_out`), the software should use the following steps when changing the card clock frequency:†

1. Before disabling the clocks, ensure that the card is not busy due to any previous data command. To determine this, check for 0 in bit 9 of the STATUS register.†
2. Update the Clock Enable register to disable all clocks. To ensure completion of any previous command before this update, send a command to the CIU to update the clock registers by setting:†
   - `start_cmd` bit†
   - "update clock registers only" bits†
   - "wait_previous data complete"†

   Note: Wait for the CIU to take the command by polling for 0 on the `start_cmd` bit.†
3. Set the `start_cmd` bit to update the Clock Divider or Clock Source register, or both, Send a command to the CIU to update the clock registers. Wait for the CIU to take the command.
4. Set `start_cmd` to update the Clock Enable register in order to enable the required clocks and send a command to the CIU to update the clock registers; wait for the CIU to take the command.†

15.5.1.2. Reading from a Card in Non-DMA Mode†

When a card is read in non-DMA mode, the Data Transfer Over (RINTSTS[3]) interrupt occurs as soon as the data transfer from the card is over. There still could be some data left in the FIFO, and the RX_WMark interrupt may or may not occur, depending on the remaining bytes in the FIFO. Software should read any remaining bytes upon

seeing the Data Transfer Over (DTO) interrupt. While using the external DMA interface for reading from a card, the DTO interrupt occurs only after all the data is flushed to memory by the DMA Interface unit.†

15.5.1.3. Writing to a Card in External DMA Mode†

While writing to a card in external DMA mode, if an undefined-length transfer is selected by setting the Byte Count register to 0, the DMA logic may request more data than it sends to the card, since it has no way of knowing at which point the software stops the transfer. The DMA request stops as soon as the DTO is set by the CIU.†

15.5.1.4. Software Issues a `Controller_Reset` Command†

If the software issues a `controller_reset` command by setting control register bit[0] to 1, all the CIU state machines are reset; the FIFO is not cleared. The DMA sends all remaining bytes to the host. In addition to a card-reset, if a FIFO reset is also issued, then:†

- Any pending DMA transfer on the bus completes correctly†
- DMA data read is ignored†
- Write data is unknown (x)†

Additionally, if dma_reset is also issued, any pending DMA transfer is abruptly terminated. When the DW-DMA/Non-DW-DMA is used, the DMA controller channel should also be reset and reprogrammed.†

If any of the previous data commands do not properly terminate, then the software should issue the FIFO reset in order to remove any residual data, if any, in the FIFO. After asserting the FIFO reset, you should wait until this bit is cleared.†

15.5.1.5. Data-Transfer Requirement Between the FIFO and Host†

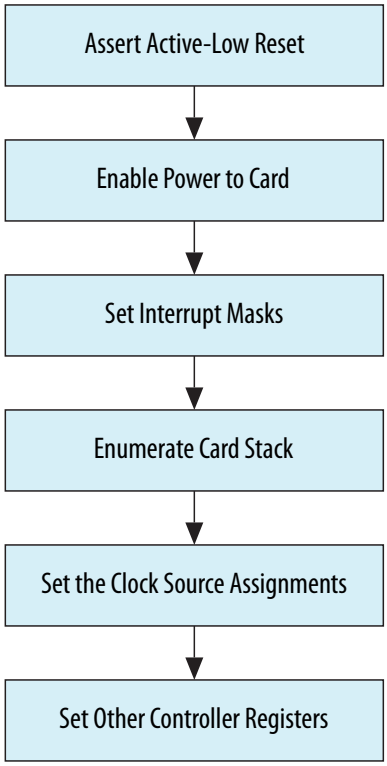
One data-transfer requirement between the FIFO and host is that the number of transfers should be a multiple of the FIFO data width (`F_DATA_WIDTH`). For example, if `F_DATA_WIDTH` = 32 and you want to write only 15 bytes to an `SD_MMC_CEATA` card (`BYTCNT`), the host should write 16 bytes to the FIFO or program the DMA to do 16-byte transfers, if external DMA mode is enabled. The software can still program the Byte Count register to only 15, at which point only 15 bytes are transferred to the card. Similarly, when 15 bytes are read from a card, the host should still read all 16 bytes from the FIFO.†

It is recommended that you not change the FIFO threshold register in the middle of data transfers when DW-DMA/Non-DW-DMA mode is chosen.†

15.5.2. Initialization†

After the power and clock to the controller are stable, the controller active-low reset is asserted. The reset sequence initializes the registers, FIFO buffer pointers, DMA interface controls, and state machines in the controller.†

Figure 77. SD/MMC Controller Initialization Sequence†



15.5.2.1. Power-On Reset Sequence

Software must perform the following steps after the power-on-reset:

1. Before enabling power to the card, confirm that the voltage setting to the voltage regulator is correct. †
2. Enable power to the card by setting the power enable bit (`power_enable`) in the power enable register (`pwren`) to 1. Wait for the power ramp-up time before proceeding to the next step.†
3. Set the interrupt masks by resetting the appropriate bits to 0 in the `intmask` register.†
4. Set the `int_enable` bit of the `ctrl` register to 1.†

   Note: Intel recommends that you write 0xFFFFFFFF to the `rintsts` register to clear any pending interrupts before setting the `int_enable` bit to 1.†
5. Discover the card stack according to the card type. For discovery, you must restrict the clock frequency to 400 kHz in accordance with SD/MMC/CE-ATA standards. For more information, refer to *Enumerated Card Stack*.†
6. Set the clock source assignments. Set the card frequency using the `clkdiv` and `clksrc` registers of the controller. For more information, refer to *Clock Setup*.†
7. The following common registers and fields can be set during initialization process:†

- The response timeout field (`response_timeout`) of the `tmout` register. A typical value is `0x40`.†
- The data timeout field (`data_timeout`) of the `tmout` register, highest of the following:†
  - 10 * N_{AC}†

    N_{AC} = card device total access time†

    = 10 * ((TAAC * F_{OP}) + (100 * NSAC))†

    where:†

    TAAC = Time-dependent factor of the data access time†

    F_{OP} = The card clock frequency used for the card operation†

    NSAC = Worst-case clock rate-dependent factor of the data access time†
  - Host FIFO buffer latency†

    On read: Time elapsed before host starts reading from a full FIFO buffer†

    On write: Time elapsed before host starts writing to an empty FIFO buffer†
- Debounce counter register (`debnce`). A typical debounce value is 25 ms.†
- TX watermark field (`tx_wmark`) of the FIFO threshold watermark register (`fifoth`). Typically, the threshold value is set to 512, which is half the FIFO buffer depth.†
- RX watermark field (`rx_wmark`) of the `fifoth` register. Typically, the threshold value is set to 511.†

These registers do not need to be changed with every SD/MMC/CE-ATA command. Set them to a typical value according to the SD/MMC/CE-ATA specifications.

Related Information

- Clock Setup on page 449
  Refer to this section for information on setting the clock source assignments.
- Enumerated Card Stack on page 446
  Refer to this section for information on discovering the card stack according to the card type.

15.5.2.2. Enumerated Card Stack

The card stack performs the following tasks:

- Discovers the connected card †
- Sets the relative Card Address Register (RCA) in the connected card†
- Reads the card specific information†
- Stores the card specific information locally†

The card connected to the controller can be an MMC, CE-ATA, SD or SDIO (including IO ONLY, MEM ONLY and COMBO) card.

15.5.2.2.1. Identifying the Connected Card Type

To identify the connected card type, the following discovery sequence is needed:

1. Reset the card width 1 or 4 bit (`card_width2`) and card width 8 bit (`card_width1`) fields in the `ctype` register to 0.
2. Identify the card type as SD, MMC, SDIO or SDIO-COMBO:
   a. Send an `SD/SDIO IO_SEND_OP_COND (CMD5)` command with argument 0 to the card.
   b. Read `resp0` on the controller. The response to the `IO_SEND_OP_COND` command gives the voltage that the card supports.
   c. Send the `IO_SEND_OP_COND` command, with the desired voltage window in the arguments. This command sets the voltage window and makes the card exit the initialization state.
   d. Check bit 27 in `resp0`:
      - If bit 27 is 0, the SDIO card is IO ONLY. In this case, proceed to step 5.
      - If bit 27 is 1, the card type is SDIO COMBO. Continue with the following steps.
3. Go to Card Type is Either SDIO COMBO or Still in Initialization on page 448.
4. Go to Determine if Card is a CE-ATA 1.1, CE-ATA 1.0, or MMC Device on page 449.
5. At this point, the software has determined the card type as SD/SDHC, SDIO or SDIO-COMBO. Now it must enumerate the card stack according to the type that has been discovered.
6. Set the card clock source frequency to the frequency of identification clock rate, 400 KHz. Use one of the following discovery command sequences:
   - For an SD card or an SDIO memory section, send the following SD/SDIO command sequence:
     — `GO_IDLE_STATE`
     — `SEND_IF_COND`
     — `SD_SEND_OP_COND (ACMD41)`
     — `ALL_SEND_CID (CMD2)`
     — `SEND_RELATIVE_ADDR (CMD3)`
   - For an SDIO card, send the following command sequence:
     — `IO_SEND_OP_COND`
     — If the function count is valid, send the `SEND_RELATIVE_ADDR` command.
   - For an MMC, send the following command sequence:
     — `GO_IDLE_STATE`
     — `SEND_OP_COND (CMD1)`
     — `ALL_SEND_CID`
     — `SEND_RELATIVE_ADDR`
7. You can change the card clock frequency after discovery by writing a value to the `clkdiv` register that divides down the `sdmmc_clk` clock.

The following list shows typical clock frequencies for various types of cards:

- SD memory card, 25 MHz†
- MMC card device, 12.5 MHz†
- Full speed SDIO, 25 MHz†
- Low speed SDIO, 400 kHz†

Related Information

SD Association
To learn more about how SD technology works, visit the SD Association website (www.sdcard.org).

Card Type is Either SDIO COMBO or Still in Initialization

Only continue with this step if the SDIO card type is COMBO or there is no response received from the previous IO_SEND_OP_COND command. Otherwise, skip to Step 5 in the *Identifying the Connected Card Type* section.

1. Send the SD/SDIO SEND_IF_COND (CMD8) command with the following arguments:
   - Bit[31:12] = 0x0 (reserved bits)†
   - Bit[11:8] = 0x1 (supply voltage value)†
   - Bit[7:0] = 0xAA (preferred check pattern by SD memory cards compliant with SDIO Simplified Specification Version 2.00 and later.)†

   Refer to *SDIO Simplified Specification Version 2.00* as described on the SD Association website.
   - If a response is received to the previous SEND_IF_COND command, the card supports SD High-Capacity, compliant with *SD Specifications, Part 1, Physical Layer Simplified Specification Version 2.00*.
   - If no response is received, proceed to the next decision statement.
2. Send the SD_SEND_OP_COND (ACMD41) command with the following arguments:
   - Bit[31] = 0x0 (reserved bits)†
   - Bit[30] = 0x1 (high capacity status)†
   - Bit[29:25] = 0x0 (reserved bits)†
   - Bit[24] = 0x1 (S18R --supports voltage switching for 1.8V)†
   - Bit[23:0] = supported voltage range†
   - If the previous SD_SEND_OP_COND command receives a response, then the card type is SDHC. Otherwise, the card is MMC or CE-ATA. In either case, skip the following steps and proceed to Step 5 in the *Identifying the Connected Card Type* section.
   - If the initial SEND_IF_COND command does not receive a response, then the card does not support High Capacity SD2.0. Now, proceed to step 3.
3. Next, issue the GO_IDLE_STATE command followed by the SD_SEND_OP_COND command with the following arguments:

- Bit[31] = 0x0 (reserved bits)†
- Bit[30] = 0x0 (high capacity status)†
- Bit[29:24] = 0x0 (reserved bits)†
- Bit[23:0] = supported voltage range†

If a response is received to the previous SD_SEND_OP_COND command, the card is SD type. Otherwise, the card is MMC or CE-ATA.

Note: You must issue the SEND_IF_COND command prior to the first SD_SEND_OP_COND command, to initialize the High Capacity SD memory card. The card returns busy as a response to the SD_SEND_OP_COND command when any of the following conditions are true:

- The card executes its internal initialization process.
- A SEND_IF_COND command is not issued before the SD_SEND_OP_COND command.
- The ACMD41 command is issued. In the command argument, the Host Capacity Support (HCS) bit is set to 0, for a high capacity SD card.

Determine if Card is a CE-ATA 1.1, CE-ATA 1.0, or MMC Device

Use the following sequence to determine whether the card is a CE-ATA 1.1, CE-ATA 1.0, or MMC device:

Determine whether the card is a CE-ATA v1.1 card device by attempting to select ATA mode.

1. Send the SD/SDIO SEND_IF_COND command, querying byte 504 (S_CMD_SET) of the EXT_CSD register block in the external card.

   If bit 4 is set to 1, the card device supports ATA mode.

2. Send the SWITCH_FUNC (CMD6) command, setting the ATA bit (bit 4) of the EXT_CSD register slice 191 (CMD_SET) to 1.

   This command selects ATA mode and activates the ATA command set.

3. You can verify the currently selected mode by reading it back from byte 191 of the EXT_CSD register.

4. Skip to Step 5 in the *Identifying the Connected Card Type* section.

   If the card device does not support ATA mode, it might be an MMC card or a CE-ATA v1.0 card. Proceed to the next section to determine whether the card is a CE-ATA 1.0 card device or an MMC card device.

Determine whether the card is a CE-ATA 1.0 card device or an MMC card device by sending the RW_REG command.

If a response is received and the response data contains the CE-ATA signature, the card is a CE-ATA 1.0 card device. Otherwise, the card is an MMC card device.

15.5.2.3. Clock Setup

The following registers of the SD/MMC controller allow software to select the desired clock frequency for the card:

- clksrc
- clkdiv
- clkena

The controller loads these registers when it receives an update clocks command.

15.5.2.3.1. Changing the Card Clock Frequency

To change the card clock frequency, perform the following steps:

1. Before disabling the clocks, ensure that the card is not busy with any previous data command. To do so, verify that the data_busy bit of the status register (status) is 0.
2. Reset the cclk_enable bit of the clkena register to 0, to disable the card clock generation.
3. Reset the clksrc register to 0.
4. Set the following bits in the cmd register to 1:
   - update_clk_regs_only—Specifies the update clocks command†
   - wait_prvdata_complete—Ensures that clock parameters do not change until any ongoing data transfer is complete†
   - start_cmd—Initiates the command†
5. Wait until the start_cmd bit changes to 0. There is no interrupt when the clock modification completes. The controller does not set the command_done bit in the rintsts register upon command completion. The controller might signal a hardware lock error if it already has another command in the queue. In this case, return to Step 4.

   For information about hardware lock errors, refer to the "Interrupt and Error Handling" chapter.
6. Reset the sdmmc_clk_enable bit to 0 in the enable register of the clock manager peripheral PLL group (perpllgrp).
7. In the control register (ctrl) of the SDMMC controller group (sdmmcgrp) in the system manager, set the drive clock phase shift select (drvsel) and sample clock phase shift select (smplsel) bits to specify the required phase shift value.
8. Set the sdmmc_clk_enable bit in the Enable register of the clock manager perpllgrp group to 1.
9. Set the clkdiv register of the controller to the correct divider value for the required clock frequency.
10. Set the cclk_enable bit of the clkena register to 1, to enable the card clock generation.

    You can also use the clkena register to enable low-power mode, which automatically stops the sdmmc_cclk_out clock when the card is idle for more than eight clock cycles.

Related Information

Interrupt and Error Handling on page 476
Refer to this section for information about hardware lock errors.

15.5.2.3.2. Timing Tuning

This section is pending further information.

15.5.3. Controller/DMA/FIFO Buffer Reset Usage

The following list shows the effect of reset on various parts in the SD/MMC controller:†

- Controller reset—resets the controller by setting the `controller_reset` bit in the `ctrl` register to 1. Controller reset resets the CIU and state machines, and also resets the BIU-to-CIU interface. Because this reset bit is self-clearing, after issuing the reset, wait until this bit changes to 0.†
- FIFO buffer reset—resets the FIFO buffer by setting the FIFO reset bit (`fifo_reset`) in the `ctrl` register to 1. FIFO buffer reset resets the FIFO buffer pointers and counters in the FIFO buffer. Because this reset bit is self-clearing, after issuing the reset, wait until this bit changes to 0.†
- DMA reset—resets the internal DMA controller logic by setting the DMA reset bit (`dma_reset`) in the `ctrl` register to 1, which immediately terminates any DMA transfer in progress. Because this reset bit is self-clearing, after issuing the reset, wait until this bit changes to 0.†

Note: Ensure that the DMA is idle before performing a DMA reset. Otherwise, the L3 interconnect might be left in an indeterminate state.†

Intel recommends setting the `controller_reset`, `fifo_reset`, and `dma_reset` bits in the `ctrl` register to 1 first, and then resetting the `rintsts` register to 0 using another write, to clear any resultant interrupt.

15.5.4. Non-Data Transfer Commands

To send any non-data transfer command, the software needs to write the `cmd` register and the `cmdarg` register with appropriate parameters. Using these two registers, the controller forms the command and sends it to the CMD pin. The controller reports errors in the command response through the error bits of the `rintsts` register.†

When a response is received—either erroneous or valid—the controller sets the `command_done` bit in the `rintsts` register to 1. A short response is copied to `resp0`, while a long response is copied to all four response registers (`resp0`, `resp1`, `resp2`, and `resp3`).† For long responses, bit 31 of `resp3` represents the MSB and bit 0 of `resp0` represents the LSB.†

For basic and non-data transfer commands, perform the following steps:

1. Write the `cmdarg` register with the appropriate command argument parameter.†
2. Write the `cmd` register with the settings in *Register Settings for Non-Data Transfer Command*.†
3. Wait for the controller to accept the command. The `start_cmd` bit changes to 0 when the command is accepted.†

The following actions occur when the command is loaded into the controller:†

- If no previous command is being processed, the controller accepts the command for execution and resets the `start_cmd` bit in the `cmd` register to 0. If a previous command is being processed, the controller loads the new command in the command buffer.†
- If the controller is unable to load the new command—that is, a command is already in progress, a second command is in the buffer, and a third command is attempted—the controller generates a hardware lock error.†

4. Check if there is a hardware lock error.†
5. Wait for command execution to complete. After receiving either a response from a card or response timeout, the controller sets the `command_done` bit in the `rintsts` register to 1. Software can either poll for this bit or respond to a generated interrupt (if enabled).†
6. Check if the response timeout boot acknowledge received (`bar`), `rcrc`, or `re` bit is set to 1. Software can either respond to an interrupt raised by these errors or poll the `re`, `rcrc`, and `bar` bits of the `rintsts` register. If no response error is received, the response is valid. If required, software can copy the response from the response registers.†

Note: Software cannot modify clock parameters while a command is being executed.†

Related Information

cmd Register Settings for Non-Data Transfer Command† on page 452
Refer to this table for information about Non-Data Transfer commands.

15.5.4.1. cmd Register Settings for Non-Data Transfer Command†

Table 157. Default

Parameter	Value	Comment
`start_cmd`	1	This bit resets itself to 0 after the command is committed.
`use_hold_reg`	1 or 0	Choose the value based on the speed mode used.
`update_clk_regs_only`	0	Indicates that the command is not a clock update command
`data_expected`	0	Indicates that the command is not a data command
`card_number`	1	For one card
`cmd_index`	Command Index	Set this parameter to the command number. For example, set to 8 for the SD/SDIO SEND_IF_COND (CMD8) command.
`send_initialization`	0 or 1	1 for card reset commands such as the SD/SDIO GO_IDLE_STATE command 0 otherwise
`stop_abort_cmd`	0 or 1	1 for a command to stop data transfer, such as the SD/SDIO STOP_TRANSMISSION command 0 otherwise
`response_length`	0 or 1	1 for R2 (long) response

continued...

Parameter	Value	Comment
		0 for short response
response_expect	0 or 1	0 for commands with no response, such as SD/SDIO GO_IDLE_STATE, SET_DSR (CMD4), or GO_INACTIVE_STATE (CMD15). 1 otherwise

Table 158. User Selectable

Parameter	Value	Comment
wait_prvdata_complete	1	Before sending a command on the command line, the host must wait for completion of any data command already in process. Intel recommends that you set this bit to 1, unless the current command is to query status or stop data transfer when transfer is in progress.
check_response_crc	1 or 0	1 if the response includes a valid CRC, and the software is required to crosscheck the response CRC bits. 0 otherwise

15.5.5. Data Transfer Commands

Data transfer commands transfer data between the memory card and the controller. To issue a data command, the controller requires a command argument, total data size, and block size. Data transferred to or from the memory card is buffered by the controller FIFO buffer.†

15.5.5.1. Confirming Transfer State

Before issuing a data transfer command, software must confirm that the card is not busy and is in a transfer state, by performing the following steps:†

1. Issue an SD/SDIO SEND_STATUS (CMD13) command. The controller sends the status of the card as the response to the command.†
2. Check the card's busy status.†
3. Wait until the card is not busy.†
4. Check the card's transfer status. If the card is in the stand-by state, issue an SD/SDIO SELECT/DESELECT_CARD (CMD7) command to place it in the transfer state.†

15.5.5.2. Busy Signal After CE-ATA RW_BLK Write Transfer

During CE-ATA RW_BLK write transfers, the MMC busy signal might be asserted after the last block. If the CE-ATA card device interrupt is disabled (the nIEN bit in the card device's ATA control register is set to 1), the `dto` bit in the `rintsts` register is set to 1 even though the card sends MMC BUSY. The host cannot issue the CMD60 command to check the ATA busy status after a CMD61 command. Instead, the host must perform one of the following actions:†

- Issue the SEND_STATUS command and check the MMC busy status before issuing a new CMD60 command†
- Issue the CMD39 command and check the ATA busy status before issuing a new CMD60 command†

For the data transfer commands, software must set the ctype register to the bus width that is programmed in the card.†

15.5.5.3. Data Transfer Interrupts

The controller generates an interrupt for different conditions during data transfer, which are reflected in the following rintsts register bits:†

1. dto—Data transfer is over or terminated. If there is a response timeout error, the controller does not attempt any data transfer and the Data Transfer Over bit is never set.†
2. Transmit FIFO data request bit (txdr)—The FIFO buffer threshold for transmitting data is reached; software is expected to write data, if available, into the FIFO buffer.†
3. Receive FIFO data request bit (rxdr)—The FIFO buffer threshold for receiving data is reached; software is expected to read data from the FIFO buffer.†
4. hto—The FIFO buffer is empty during transmission or is full during reception. Unless software corrects this condition by writing data for empty condition, or reading data for full condition, the controller cannot continue with data transfer. The clock to the card is stopped.†
5. bds—The card has not sent data within the timeout period.†
6. dcrc—A CRC error occurred during data reception.†
7. sbe—The start bit is not received during data reception.†
8. ebe—The end bit is not received during data reception, or for a write operation. A CRC error is indicated by the card.†

dcrc, sbe, and ebe indicate that the received data might have errors. If there is a response timeout, no data transfer occurs.†

15.5.5.4. Single-Block or Multiple-Block Read

To implement a single-block or multiple-block read, the software performs the following steps:†

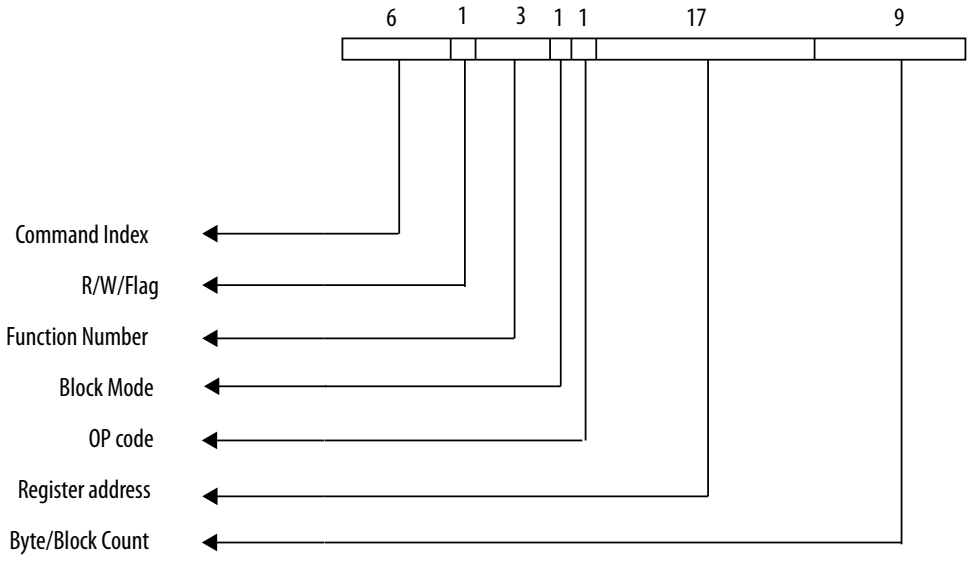
1. Write the data size in bytes to the bytcnt register. For a multi-block read, bytcnt must be a multiple of the block size.†
2. Write the block size in bytes to the blksiz register. The controller expects data to return from the card in blocks of size blksiz.†
3. If the read round trip delay, including the card delay, is greater than half of sdmmc_clk_divided, write to the card threshold control register (cardthrctl) to ensure that the card clock does not stop in the middle of a block of data being transferred from the card to the host. For more information, refer to *Card Read Threshold*.†

   Note: If the card read threshold enable bit (cardrdthren) is 0, the host system must ensure that the RX FIFO buffer does not become full during a read data transfer by ensuring that the RX FIFO buffer is read at a rate faster than that at which data is written into the FIFO buffer. Otherwise, an overflow might occur.†
4. Write the cmdarg register with the beginning data address for the data read.†

5. Write the cmd register with the parameters listed in *cmd Register Settings for Single-Block and Multiple-Block Reads*. For SD and MMC cards, use the SD/SDIO READ_SINGLE_BLOCK (CMD17) command for a single-block read and the READ_MULTIPLE_BLOCK (CMD18) command for a multiple-block read. For SDIO cards, use the IO_RW_EXTENDED (CMD53) command for both single-block and multiple-block transfers. The command argument for (CMD53) is shown in the figure, below. After writing to the cmd register, the controller starts executing the command. When the command is sent to the bus, the Command Done interrupt is generated.†
6. Software must check for data error interrupts, reported in the dcrc, bds, sbe, and ebe bits of the rintsts register. If required, software can terminate the data transfer by sending an SD/SDIO STOP command.†
7. Software must check for host timeout conditions in the rintsts register:†
   - Receive FIFO buffer data request†
   - Data starvation from host—the host is not reading from the FIFO buffer fast enough to keep up with data from the card. To correct this condition, software must perform the following steps:†
     — Read the fifo_count field of the status register†
     — Read the corresponding amount of data out of the FIFO buffer†

   In both cases, the software must read data from the FIFO buffer and make space in the FIFO buffer for receiving more data.†
8. When a DTO interrupt is received, the software must read the remaining data from the FIFO buffer.†

Figure 78. Command Argument for IO_RW_EXTENDED (CMD53)†



Related Information

- Card Read Threshold on page 473
  Refer to this section for information about the thresholds for a card read.
- cmd Register Settings for Single-Block and Multiple-Block Reads† on page 456
  Refer to this table for information about the settings for Single-Block and Multiple-Block Reads.

15.5.5.4.1. cmd Register Settings for Single-Block and Multiple-Block Reads†

Table 159. cmd Register Settings for Single-Block and Multiple-Block Reads (Default)

Parameter	Value	Comment
start_cmd	1	This bit resets itself to 0 after the command is committed.
use_hold_reg	1 or 0	Choose the value based on speed mode used.
update_clk_regs_only	0	Does not need to update clock parameters
data_expected	1	Data command
card_number	1	For one card
transfer_mode	0	Block transfer
send_initialization	0	1 for a card reset command such as the SD/SDIO GO_IDLE_STATE command 0 otherwise
stop_abort_cmd	0	1 for a command to stop data transfer such as the SD/SDIO STOP_TRANSMISSION command 0 otherwise
send_auto_stop	0 or 1	Refer to *Auto Stop* for information about how to set this parameter.
read_write	0	Read from card
response_length	0	1 for R2 (long) response 0 for short response
response_expect	1 or 0	0 for commands with no response, such as SD/SDIO GO_IDLE_STATE, SET_DSR, and GO_INACTIVE_STATE. 1 otherwise

Table 160. cmd Register Settings for Single-Block and Multiple-Block Reads (User Selectable)

Parameter	Value	Comment
wait_prvdata_complete	1 or 0	0 - sends command to CIU immediately 1 - sends command after previous data transfer ends
check_response_crc	1 or 0	0 - Controller must not check response CRC 1 - Controller must check response CRC
cmd_index	Command Index	Set this parameter to the command number. For example, set to 17 or 18 for SD/SDIO READ_SINGLE_BLOCK (CMS17) or READ_MULTIPLE_BLOCK (CMD18)

Related Information

Auto-Stop on page 433
Refer to this table for information about setting the send_auto_stop parameter.

15.5.5.5. Single-Block or Multiple-Block Write

The following steps comprise a single-block or multiple-block write:

1. Write the data size in bytes to the bytcnt register. For a multi-block write, bytcnt must be a multiple of the block size.†
2. Write the block size in bytes to the blksiz register. The controller sends data in blocks of size blksiz each.†
3. Write the cmdarg register with the data address to which data must be written.†
4. Write data into the FIFO buffer. For best performance, the host software should write data continuously until the FIFO buffer is full.†
5. Write the cmd register with the parameters listed in *cmd Register Settings for Single-Block and Multiple-Block Write*. For SD and MMC cards, use the SD/SDIO WRITE_BLOCK (CMD24) command for a single-block write and the WRITE_MULTIPLE_BLOCK (CMD25) command for a multiple-block writes. For SDIO cards, use the IO_RW_EXTENDED command for both single-block and multiple-block transfers.†

   After writing to the cmd register, the controller starts executing a command if there is no other command already being processed. When the command is sent to the bus, a Command Done interrupt is generated.†
6. Software must check for data error interrupts; that is, for dcrc, bds, and ebe bits of the rintsts register. If required, software can terminate the data transfer early by sending the SD/SDIO STOP command.†
7. Software must check for host timeout conditions in the rintsts register: †
   - Transmit FIFO buffer data request.†
   - Data starvation by the host—the controller wrote data to the card faster than the host could supply the data.†

   In both cases, the software must write data into the FIFO buffer.†

   There are two types of transfers: open-ended and fixed length.†
   - Open-ended transfers—For an open-ended block transfer, the byte count is 0. At the end of the data transfer, software must send the STOP_TRANSMISSION command (CMD12).†
   - Fixed-length transfers—The byte count is nonzero. You must already have written the number of bytes to the bytcnt register. The controller issues the STOP command for you if you set the send_auto_stop bit of the cmd register to 1. After completion of a transfer of a given number of bytes, the controller sends the STOP command. Completion of the AUTO_STOP command is reflected by the Auto Command Done interrupt. A response to the AUTO_STOP command is written to the resp1 register. If software does not set the send_auto_stop bit in the cmd register to 1, software must issue the STOP command just like in the open-ended case.†

When the dto bit of the rintsts register is set, the data command is complete.†

15.5.5.5.1. cmd Register Settings for Single-Block and Multiple-Block Write

Table 161. cmd Register Settings for Single-Block and Multiple-Block Write (Default)†

Parameter	Value	Comment
start_cmd	1	This bit resets itself to 0 after the command is committed (accepted by the BIU).
use_hold_reg	1 or 0	Choose the value based on speed mode used.
update_clk_regs_only	0	Does not need to update clock parameters
data_expected	1	Data command
card_number	1	For one card
transfer_mode	0	Block transfer
send_initialization	0	Can be 1, but only for card reset commands such as SD/SDIO GO_IDLE_STATE
stop_abort_cmd	0	Can be 1 for commands to stop data transfer such as SD/SDIO STOP_TRANSMISSION
send_auto_stop	0 or 1	Refer to *Auto Stop* for information about how to set this parameter.
read_write	1	Write to card
response_length	0	Can be 1 for R2 (long) responses
response_expect	1	Can be 0 for commands with no response. For example, SD/SDIO GO_IDLE_STATE, SET_DSR, GO_INACTIVE_STATE etc.

Table 162. cmd Register Settings for Single-Block and Multiple-Block Write (User Selectable)†

Parameter	Value	Comment
wait_prvdata_complete	1	0—Sends command to the CIU immediately 1—Sends command after previous data transfer ends
check_response_crc	1	0—Controller must not check response CRC 1—Controller must check response CRC
cmd_index	Command Index	Set this parameter to the command number. For example, set to 24 for SD/SDIO WRITE_BLOCK (CMD24) or 25 for WRITE_MULTIPLE_BLOCK (CMD25).

Related Information

Auto-Stop on page 433
Refer to this table for information about setting the send_auto_stop parameter.

15.5.5.6. Stream Read and Write

In a stream transfer, if the byte count is equal to 0, the software must also send the SD/SDIO STOP command. If the byte count is not 0, when a given number of bytes completes a transfer, the controller sends the STOP command automatically. Completion of this AUTO_STOP command is reflected by the Auto_command_done interrupt. A response to an AUTO_STOP command is written to the resp1 register. A stream transfer is allowed only for card interfaces with a 1-bit data bus.†

A stream read requires the same steps as the block read described in *Single-Block or Multiple-Block Read*, except for the following bits in the cmd register:†

- `transfer_mode` = 0x1 (for stream transfer)†
- `cmd_index` = 20 (SD/SDIO CMD20)†

A stream write requires the same steps as the block write mentioned in *Single-Block or Multiple-Block Write*, except for the following bits in the cmd register:†

- `transfer_mode` = 0x1 (for stream transfer)†
- `cmd_index` = 11 (SD/SDIO CMD11)†

Related Information

- Single-Block or Multiple-Block Read on page 454
  Refer to this section for more information about a stream read.
- Single-Block or Multiple-Block Write on page 457
  Refer to this section for more information about a stream write.

15.5.5.7. Packed Commands

To reduce overhead, read and write commands can be packed in groups of commands —either all read or all write—that transfer the data for all commands in the group in one transfer on the bus. Use the SD/SDIO SET_BLOCK_COUNT (CMD23) command to state ahead of time how many blocks are transferred. Then issue a single READ_MULTIPLE_BLOCK or WRITE_MULTIPLE_BLOCK command to read or write multiple blocks.†

- SET_BLOCK_COUNT—set block count (number of blocks transferred using the READ_MULTIPLE_BLOCK or WRITE_MULTIPLE_BLOCK command) †
- READ_MULTIPLE_BLOCK—multiple-block read command †
- WRITE_MULTIPLE_BLOCK—multiple-block write command†

Packed commands are organized in packets by the application software and are transparent to the controller.†

Related Information

JEDEC Global Standards of the Microelectronics Industry
For more information about packed commands, refer to JEDEC Standard No. 84-A441, available on the JEDEC website.

15.5.6. Transfer Stop and Abort Commands

This section describes stop and abort commands. The SD/SDIO STOP_TRANSMISSION command can terminate a data transfer between a memory card and the controller. The ABORT command can terminate an I/O data transfer for only an SDIO card. †

15.5.6.1. STOP_TRANSMISSION (CMD12)

The host can send the STOP_TRANSMISSION (CMD12) command on the CMD pin at any time while a data transfer is in progress. Perform the following steps to send the STOP_TRANSMISSION command to the SD/SDIO card device:†

1. Set the `wait_prvdata_complete` bit of the `cmd` register to 0.†
2. Set the `stop_abort_cmd` in the `cmd` register to 1, which ensures that the CIU stops.†

The STOP_TRANSMISSION command is a non-data transfer command.†

Related Information

Non-Data Transfer Commands on page 451
Refer to this section for information on the STOP_TRANSMISSION command.

15.5.6.2. ABORT

The ABORT command can only be used with SDIO cards. To abort the function that is transferring data, program the ABORT function number in the ASx[2:0] bits at address 0x06 of the card common control register (CCCR) in the card device, using the IO_RW_DIRECT (CMD52) command. The CCCR is located at the base of the card space 0x00 – 0xFF.†

Note: The ABORT command is a non-data transfer command.†

Related Information

Non-Data Transfer Commands on page 451
Refer to this section for information on the ABORT command.

15.5.6.2.1. Sending the ABORT Command

Perform the following steps to send the ABORT command to the SDIO card device:†

1. Set the `cmdarg` register to include the appropriate command argument parameters listed in *cmdarg Register Settings for SD/SDIO ABORT Command*.†
2. Send the IO_RW_DIRECT command by setting the following fields of the `cmd` register:†
   - Set the command index to 0x52 (IO_RW_DIRECT).†
   - Set the `stop_abort_cmd` bit of the `cmd` register to 1 to inform the controller that the host aborted the data transfer.†
   - Set the `wait_prvdata_complete` bit of the `cmd` register to 0.†
3. Wait for the `cmd` bit in the `rintsts` register to change to 1.†
4. Read the response to the IO_RW_DIRECT command (R5) in the response registers for any errors.†

For more information about response values, refer to the *Physical Layer Simplified Specification*, Version 3.01, available on the SD Association website.

Related Information

SD Association
To learn more about how SD technology works, visit the SD Association website (www.sdcard.org).

15.5.6.2.2. cmdarg Register Settings for SD/SDIO ABORT Command†

Table 163. cmdarg Register Settings for SD/SDIO ABORT Command

Bits	Contents	Value
31	R/W flag	1
30:28	Function number	0, for access to the CCCR in the card device
27	RAW flag	1, if needed to read after write
26	Don't care	-
25:9	Register address	0x06
8	Don't care	-
7:0	Write data	Function number to abort

15.5.7. Internal DMA Controller Operations

For better performance, you can use the internal DMA controller to transfer data between the host and the controller. This section describes the internal DMA controller's initialization process, and transmission sequence, and reception sequence.

15.5.7.1. Internal DMA Controller Initialization

To initialize the internal DMA controller, perform the following steps:†

1. Set the required `bmod` register bits: †
   - If the internal DMA controller enable bit (`de`) of the `bmod` register is set to 0 during the middle of a DMA transfer, the change has no effect. Disabling only takes effect for a new data transfer command.†
   - Issuing a software reset immediately terminates the transfer. Prior to issuing a software reset, Intel recommends the host reset the DMA interface by setting the `dma_reset` bit of the `ctrl` register to 1.†
   - The `pbl` field of the `bmod` register is read-only and a direct reflection of the contents of the DMA multiple transaction size field (`dw_dma_multiple_transaction_size`) in the `fifoth` register.†
   - The `fb` bit of the `bmod` register has to be set appropriately for system performance.†
2. Write to the `idinten` register to mask unnecessary interrupt causes according to the following guidelines:†
   - When a Descriptor Unavailable interrupt is asserted, the software needs to form the descriptor, appropriately set its own bit, and then write to the poll demand register (`pldmnd`) for the internal DMA controller to re-fetch the descriptor.†
   - It is always appropriate for the software to enable abnormal interrupts because any errors related to the transfer are reported to the software.†
3. Populate either a transmit or receive descriptor list in memory. Then write the base address of the first descriptor in the list to the internal DMA controller's descriptor list base address register (`dbaddr`). The DMA controller then proceeds

to load the descriptor list from memory. *Internal DMA Controller Transmission Sequences* and *Internal DMA Controller Reception Sequences* describe this step in detail. †

Related Information

- Internal DMA Controller Transmission Sequences on page 462
  Refer to this section for information about the Internal DMA Controller Transmission Sequences.
- Internal DMA Controller Reception Sequences on page 463
  Refer to this section for information about the Internal DMA Controller Reception Sequences.

15.5.7.2. Internal DMA Controller Transmission Sequences

To use the internal DMA controller to transmit data, perform the following steps:

1. The host sets up the Descriptor fields (DES0—DES3) for transmission and sets the OWN bit (DES0[31]) to 1. The host also loads the data buffer in system memory with the data to be written to the SD card.†
2. The host writes the appropriate write data command (SD/SDIO WRITE_BLOCK or WRITE_MULTIPLE_BLOCK) to the `cmd` register. The internal DMA controller determines that a write data transfer needs to be performed.†
3. The host sets the required transmit threshold level in the `tx_wmark` field in the `fifoth` register.†
4. The internal DMA controller engine fetches the descriptor and checks the OWN bit. If the OWN bit is set to 0, the host owns the descriptor. In this case, the internal DMA controller enters the suspend state and asserts the Descriptor Unable interrupt. The host then needs to set the descriptor OWN bit to 1 and release the DMA controller by writing any value to the `pldmnd` register.†
5. The host must write the descriptor base address to the `dbaddr` register.†
6. The internal DMA controller waits for the Command Done (CD) bit in the `rintsts` register to be set to 1, with no errors from the BIU. This condition indicates that a transfer can be done.†
7. The internal DMA controller engine waits for a DMA interface request from BIU. The BIU divides each transfer into smaller chunks. Each chunk is an internal request to the DMA. This request is generated based on the transmit threshold value.†
8. The internal DMA controller fetches the transmit data from the data buffer in the system memory and transfers the data to the FIFO buffer in preparation for transmission to the card.†
9. When data spans across multiple descriptors, the internal DMA controller fetches the next descriptor and continues with its operation with the next descriptor. The Last Descriptor bit in the descriptor DES0 field indicates whether the data spans multiple descriptors or not.†
10. When data transmission is complete, status information is updated in the `idsts` register by setting the `ti` bit to 1, if enabled. Also, the OWN bit is set to 0 by the DMA controller by updating the DES0 field of the descriptor.†

15.5.7.3. Internal DMA Controller Reception Sequences

To use the internal DMA controller to receive data, perform the following steps:

1. The host sets up the descriptor fields (DES0—DES3) for reception and sets the OWN (DES0 [31]) to 1.†
2. The host writes the read data command to the `cmd` register in BIU. The internal DMA controller determines that a read data transfer needs to be performed.†
3. The host sets the required receive threshold level in the `rx_wmark` field in the `fifoth` register.†
4. The internal DMA controller engine fetches the descriptor and checks the OWN bit. If the OWN bit is set to 0, the host owns the descriptor. In this case, the internal DMA controller enters suspend state and asserts the Descriptor Unable interrupt. The host then must set the descriptor OWN bit to 1 and release the DMA controller by writing any value to the `pldmnd` register.†
5. The host must write the descriptor base address to the `dbaddr` register.†
6. The internal DMA controller waits for the `CD` bit in the `rintsts` register to be set to 1, with no errors from the BIU. This condition indicates that a transfer can be done.†
7. The internal DMA controller engine waits for a DMA interface request from the BIU. The BIU divides each transfer into smaller chunks. Each chunk is an internal request to the DMA. This request is generated based on the receive threshold value.†
8. The internal DMA controller fetches the data from the FIFO buffer and transfers the data to system memory.†
9. When data spans across multiple descriptors, the internal DMA controller fetches the next descriptor and continues with its operation with the next descriptor. The Last Descriptor bit in the descriptor indicates whether the data spans multiple descriptors or not.†
10. When data reception is complete, status information is updated in the `idsts` register by setting the `ri` bit to 1, if enabled. Also, the OWN bit is set to 0 by the DMA controller by updating the DES0 field of the descriptor.†

15.5.8. Commands for SDIO Card Devices

This section describes the commands to temporarily halt the transfers between the controller and SDIO card device.

15.5.8.1. Suspend and Resume Sequence

For SDIO cards, a data transfer between an I/O function and the controller can be temporarily halted using the SUSPEND command. This capability might be required to perform a high-priority data transfer with another function. When desired, the suspended data transfer can be resumed using the RESUME command.†

The SUSPEND and RESUME operations are implemented by writing to the appropriate bits in the CCCR (Function 0) of the SDIO card. To read from or write to the CCCR, use the controller's IO_RW_DIRECT command.†

15.5.8.1.1. Suspend

To suspend data transfer, perform the following steps:†

1. Check if the SDIO card supports the SUSPEND/RESUME protocol by reading the SBS bit in the CCCR at offset 0x08 of the card.†
2. Check if the data transfer for the required function number is in process. The function number that is currently active is reflected in the function select bits (FSx) of the CCCR, bits 3:0 at offset 0x0D of the card.†

   Note: If the bus status bit (BS), bit 0 at address 0xC, is 1, only the function number given by the FSx bits is valid.†
3. To suspend the transfer, set the bus release bit (BR), bit 2 at address 0xC, to 1.†
4. Poll the BR and BS bits of the CCCR at offset 0x0C of the card until they are set to 0. The BS bit is 1 when the currently-selected function is using the data bus. The BR bit remains 1 until the bus release is complete. When the BR and BS bits are 0, the data transfer from the selected function is suspended.†
5. During a read-data transfer, the controller can be waiting for the data from the card. If the data transfer is a read from a card, the controller must be informed after the successful completion of the SUSPEND command. The controller then resets the data state machine and comes out of the wait state. To accomplish this, set the abort read data bit (`abort_read_data`) in the `ctrl` register to 1.†
6. Wait for data completion, by polling until the `dto` bit is set to 1 in the `rintsts` register. To determine the number of pending bytes to transfer, read the transferred CIU card byte count (`tcbcnt`) register of the controller. Subtract this value from the total transfer size. You use this number to resume the transfer properly.†

15.5.8.1.2. Resume

To resume the data transfer, perform the following steps:†

1. Check that the card is not in a transfer state, which confirms that the bus is free for data transfer.†
2. If the card is in a disconnect state, select it using the SD/SDIO SELECT/DESELECT_CARD command. The card status can be retrieved in response to an IO_RW_DIRECT or IO_RW_EXTENDED command.†
3. Check that a function to be resumed is ready for data transfer. Determine this state by reading the corresponding RF<*n*> flag in CCCR at offset 0x0F of the card. If RF<*n*> = 1, the function is ready for data transfer.†

   Note: For detailed information about the RF<*n*> flags, refer to SDIO Simplified Specification Version 2.00, available on the SD Association website.†
4. To resume transfer, use the IO_RW_DIRECT command to write the function number at the FSx bits in the CCCR, bits 3:0 at offset 0x0D of the card. Form the command argument for the IO_RW_DIRECT command and write it to the `cmdarg` register. Bit values are listed in the following table.†

Table 164. cmdarg Bit Values for RESUME Command†

Bits	Content	Value
31	R/W flag	1
30:28	Function number	0, for CCCR access
27	RAW flag	1, read after write
26	Don't care	-
25:9	Register address	0x0D
8	Don't care	-
7:0	Write data	Function number that is to be resumed

5. Write the block size value to the `blksiz` register. Data is transferred in units of this block size.†
6. Write the byte count value to the `bytcnt` register. Specify the total size of the data that is the remaining bytes to be transferred. It is the responsibility of the software to handle the data.†

   To determine the number of pending bytes to transfer, read the transferred CIU card byte count register (`tcbcnt`). Subtract this value from the total transfer size to calculate the number of remaining bytes to transfer.†
7. Write to the `cmd` register similar to a block transfer operation. When the `cmd` register is written, the command is sent and the function resumes data transfer. For more information, refer to *Single-Block or Multiple-Block Read* and *Single-Block or Multiple-Block Write*.†
8. Read the resume data flag (DF) of the SDIO card device. Interpret the DF flag as follows:†
   - DF=1—The function has data for the transfer and begins a data transfer as soon as the function or memory is resumed.†
   - DF=0—The function has no data for the transfer. If the data transfer is a read, the controller waits for data. After the data timeout period, it issues a data timeout error.†

Related Information

- SD Association
  To learn more about how SD technology works, visit the SD Association website (www.sdcard.org).
- Single-Block or Multiple-Block Read on page 454
  Refer to this section for more information about writing to the `cmd` register.
- Single-Block or Multiple-Block Write on page 457
  Refer to this section for more information about writing to the `cmd` register.

15.5.8.2. Read-Wait Sequence

Read_wait is used with SDIO cards only. It temporarily stalls the data transfer, either from functions or memory, and allows the host to send commands to any function within the SDIO card device. The host can stall this transfer for as long as required. The controller provides the facility to signal this stall transfer to the card.†

15.5.8.2.1. Signaling a Stall

To signal the stall, perform the following steps:†

1. Check if the card supports the read_wait facility by reading the SDIO card's SRW bit, bit 2 at offset 0x8 in the CCCR.†
2. If this bit is 1, all functions in the card support the read_wait facility. Use the SD/SDIO IO_RW_DIRECT command to read this bit.†
3. If the card supports the `read_wait` signal, assert it by setting the read wait bit (`read_wait`) in the `ctrl` register to 1.†
4. Reset the `read_wait` bit to 0 in the `ctrl` register.†

15.5.9. CE-ATA Data Transfer Commands

This section describes CE-ATA data transfer commands.

Related Information

Data Transfer Commands on page 453
Refer to this section for information about the basic settings and interrupts generated for different conditions.

15.5.9.1. ATA Task File Transfer Overview

ATA task file registers are mapped to addresses 0x00h through 0x10h in the MMC register space. The RW_REG command is used to issue the ATA command, and the ATA task file is transmitted in a single RW_REG MMC command sequence.†

The host software stack must write the task file image to the FIFO buffer before setting the `cmdarg` and `cmd` registers in the controller. The host processor then writes the address and byte count to the `cmdarg` register before setting the `cmd` register bits.†

For the RW_REG command, there is no CCS from the CE-ATA card device. †

15.5.9.2. ATA Task File Transfer Using the RW_MULTIPLE_REGISTER (RW_REG) Command

This command involves data transfer between the CE-ATA card device and the controller. To send a data command, the controller needs a command argument, total data size, and block size. Software receives or sends data through the FIFO buffer.†

15.5.9.2.1. Implementing ATA Task File Transfer

To implement an ATA task file transfer (read or write), perform the following steps:†

1. Write the data size in bytes to the `bytcnt` register. `bytcnt` must equal the block size, because the controller expects a single block transfer.†
2. Write the block size in bytes to the `blksiz` register.†
3. Write the `cmdarg` register with the beginning register address.†

You must set the `cmdarg`, `cmd`, `blksiz`, and `bytcnt` registers according to the tables in *Register Settings for ATA Task File Transfer*.†

Related Information

Register Settings for ATA Task File Transfer on page 467
Refer to this table for information on how to set these registers.

15.5.9.2.2. Register Settings for ATA Task File Transfer

Table 165. cmdarg Register Settings for ATA Task File Transfer†

Bit	Value	Comment
31	1 or 0	Set to 0 for read operation or set to 1 for write operation
30:24	0	Reserved (bits set to 0 by host processor)
23:18	0	Starting register address for read or write (DWORD aligned)
17:16	0	Register address (DWORD aligned)
15:8	0	Reserved (bits set to 0 by host processor)
7:2	16	Number of bytes to read or write (integral number of DWORD)
1:0	0	Byte count in integral number of DWORD

Table 166. cmd Register Settings for ATA Task File Transfer†

Bit	Value	Comment
start_cmd	1	
ccs_expected	0	CCS is not expected
read_ceata_device	0 or 1	Set to 1 if RW_BLK or RW_REG read
update_clk_regs_only	0	No clock parameters update command
card_num	0	
send_initialization	0	No initialization sequence
stop_abort_cmd	0	
send_auto_stop	0	
transfer_mode	0	Block transfer mode. Block size and byte count must match number of bytes to read or write
read_write	1 or 0	1 for write and 0 for read
data_expected	1	Data is expected
response_length	0	
response_expect	1	
cmd_index	Command index	Set this parameter to the command number. For example, set to 24 for SD/SDIO WRITE_BLOCK (CMD24) or 25 for WRITE_MULTIPLE_BLOCK (CMD25).
wait_prvdata_complete	1	• 0 for send command immediately • 1 for send command after previous DTO interrupt
check_response_crc	1	• 0 for not checking response CRC • 1 for checking response CRC

Table 167. blksiz Register Settings for ATA Task File Transfer†

Bit	Value	Comment
31:16	0	Reserved bits set to 0
15:0 (block_size)	16	For accessing entire task file (16, 8-bit registers). Block size of 16 bytes

Table 168. bytcnt Register Settings for ATA Task File Transfer

Bit	Value	Comment
31:0	16	For accessing entire task file (16, 8-bit registers). Byte count value of 16 is used with the block size set to 16.

15.5.9.2.3. Reset and Card Device Discovery Overview

Before starting any CE-ATA operations, the host must perform a MMC reset and initialization procedure. The host and card device must negotiate the MMC transfer (MMC TRAN) state before the card enters the MMC TRAN state.†

The host must follow the existing MMC discovery procedure to negotiate the MMC TRAN state. After completing normal MMC reset and initialization procedures, the host must query the initial ATA task file values using the RW_REG or CMD39 command.†

By default, the MMC block size is 512 bytes—indicated by bits 1:0 of the srcControl register inside the CE-ATA card device. The host can negotiate the use of a 1 KB or 4 KB MMC block sizes. The card indicates MMC block sizes that it can support through the srcCapabilities register in the MMC; the host reads this register to negotiate the MMC block size. Negotiation is complete when the host controller writes the MMC block size into the srcControl register bits 1:0 of the card.†

Related Information

JEDEC Global Standards of the Microelectronics Industry
For information about the (MMC TRAN) state, MMC reset and initialization, refer to JEDEC Standard No. 84-A441, available on the JEDEC website.

15.5.9.3. ATA Payload Transfer Using the RW_MULTIPLE_BLOCK (RW_BLK) Command

This command involves data transfer between the CE-ATA card device and the controller. To send a data command, the controller needs a command argument, total data size, and block size. Software receives or sends data through the FIFO buffer. †

15.5.9.3.1. Implementing ATA Payload Transfer

To implement an ATA payload transfer (read or write), perform the following steps:†

1. Write the data size in bytes to the `bytcnt` register.†
2. Write the block size in bytes to the `blksiz` register. The controller expects a single/multiple block transfer.†
3. Write to the `cmdarg` register to indicate the data unit count.†

15.5.9.3.2. Register Settings for ATA Payload Transfer

You must set the `cmdarg`, `cmd`, `blksiz`, and `bytcnt` registers according to the following tables.†

Table 169. cmdarg Register Settings for ATA Payload Transfer†

Bits	Value	Comment
31	1 or 0	Set to 0 for read operation or set to 1 for write operation
30:24	0	Reserved (bits set to 0 by host processor)
23:16	0	Reserved (bits set to 0 by host processor)
15:8	Data count	Data Count Unit [15:8]
7:0	Data count	Data Count Unit [7:0]

Table 170. cmd Register Settings for ATA Payload Transfer†

Bits	Value	Comment
start_cmd	1	-
ccs_expected	1	CCS is expected. Set to 1 for the RW_BLK command if interrupts are enabled in CE-ATA card device (the nIEN bit is set to 0 in the ATA control register)
read_ceata_device	0 or 1	Set to 1 for a RW_BLK or RW_REG read command
update_clk_regs_only	0	No clock parameters update command
card_num	0	-
send_initialization	0	No initialization sequence
stop_abort_cmd	0	-
send_auto_stop	0	-
transfer_mode	0	Block transfer mode. Byte count must be integer multiple of 4kB. Block size can be 512, 1k or 4k bytes
read_write	1 or 0	1 for write and 0 for read
data_expected	1	Data is expected
response_length	0	-
response_expect	1	-
cmd_index	Command index	Set this parameter to the command number. For example, set to 24 for SD/SDIO WRITE_BLOCK (CMD24) or 25 for WRITE_MULTIPLE_BLOCK (CMD25).
wait_prvdata_complete	1	• 0 for send command immediately • 1 for send command after previous DTO interrupt
check_response_crc	1	• 0 for not checking response CRC • 1 for checking response CRC

Table 171. blksiz Register Settings for ATA Payload Transfer†

Bits	Value	Comment
31:16	0	Reserved bits set to 0
15:0 (block_size)	512, 1024 or 4096	MMC block size can be 512, 1024 or 4096 bytes as negotiated by host

Table 172. bytcnt Register Settings for ATA Payload Transfer

Bits	Value	Comment
31:0	<n>*block_size	Byte count must be an integer multiple of the block size. For ATA media access commands, byte count must be a multiple of 4 KB. (<*n*>*block_size = <*x*>*4 KB, where <*n*> and <*x*> are integers)

15.5.9.4. CE-ATA CCS

This section describes disabling the CCS, recovery after CCS timeout, and recovery after I/O read transmission delay (N_{ACIO}) timeout. †

15.5.9.4.1. Disabling the CCS

While waiting for the CCS for an outstanding RW_BLK command, the host can disable the CCS by sending a CCSD command:†

- Send a CCSD command—the controller sends the CCSD command to the CE-ATA card device if the `send_ccsd` bit is set to 1 in the `ctrl` register of the controller. This bit can be set only after a response is received for the RW_BLK command.†
- Send an internal stop command—send an internally-generated SD/SDIO STOP_TRANSMISSION (CMD12) command after sending the CCSD pattern. If the `send_auto_stop_ccsd` bit of the `ctrl` register is also set to 1 when the controller is set to send the CCSD pattern, the controller sends the internally-generated STOP command to the `CMD` pin. After sending the STOP command, the controller sets the `acd` bit in the `rintsts` register to 1.†

15.5.9.4.2. Recovery after CCS Timeout

If a timeout occurs while waiting for the CCS, the host needs to send the CCSD command followed by a STOP command to abort the pending ATA command. The host can set up the controller to send an internally-generated STOP command after sending the CCSD pattern:†

- Send CCSD command—set the `send_ccsd` bit in the `ctrl` register to 1.†
- Send external STOP command—terminate the data transfer between the CE-ATA card device and the controller. For more information about sending the STOP command, refer to *Transfer Stop and Abort Commands*.†
- Send internal STOP command—set the `send_auto_stop_ccsd` bit in the `ctrl` register to 1, which tells the controller to send the internally-generated STOP command. After sending the STOP command, the controller sets the `acd` bit in the `rintsts` register to 1. The `send_auto_stop_ccsd` bit must be set to 1 along with setting the `send_ccsd` bit.†

Related Information

Transfer Stop and Abort Commands on page 459
Refer to this section for more information about sending the STOP command.

15.5.9.4.3. Recovery after I/O Read Transmission Delay (N_{ACIO}) Timeout

If the I/O read transmission delay (N_{ACIO}) timeout occurs for the CE-ATA card device, perform one of the following steps to recover from the timeout:†

- If the CCS is expected from the CE-ATA card device (that is, the `ccs_expected` bit is set to 1 in the `cmd` register), follow the steps in *Recovery after CCS Timeout*.†
- If the CCS is not expected from the CE-ATA card device, perform the following steps: †
  1. Send an external STOP command. †
  2. Terminate the data transfer between the controller and CE-ATA card device. †

Related Information

Recovery after CCS Timeout on page 470
For more information about what steps to take if the CCS is expected from the CE-ATA card device.

15.5.9.5. Reduced ATA Command Set

It is necessary for the CE-ATA card device to support the reduced ATA command subset. This section describes the reduced command set.†

15.5.9.5.1. The IDENTIFY DEVICE Command

The IDENTIFY DEVICE command returns a 512-byte data structure to the host that describes device-specific information and capabilities. The host issues the IDENTIFY DEVICE command only if the MMC block size is set to 512 bytes. Any other MMC block size has indeterminate results.†

The host issues a RW_REG command for the ATA command, and the data is retrieved with the RW_BLK command.†

The host controller uses the following settings while sending a RW_REG command for the IDENTIFY DEVICE ATA command. The following list shows the primary bit settings:†

- `cmd` register setting: `data_expected` bit set to 0†
- `cmdarg` register settings: †
  - Bit [31] set to 0†
  - Bits [7:2] set to 128 †
  - All other bits set to 0†
- Task file settings: †
  - Command field of the ATA task file set to 0xEC†
  - Reserved fields of the task file set to 0†
- `bytcnt` register and `block_size` field of the `blksiz` register: set to 16†

The host controller uses the following settings for data retrieval (RW_BLK command):†

- `cmd` register settings:†
  - `ccs_expected` set to 1†
  - `data_expected` set to 1†
- `cmdarg` register settings: †
  - Bit [31] set to 0 (read operation) †
  - Bits [15:0] set to 1 (data unit count = 1)†
  - All other bits set to 0†
- `bytcnt` register and `block_size` field of the `blksiz` register: set to 512†

15.5.9.5.2. The READ DMA EXT Command

The READ DMA EXT command reads a number of logical blocks of data from the card device using the Data-In data transfer protocol. The host uses a RW_REG command to issue the ATA command and the RW_BLK command for the data transfer.†

15.5.9.5.3. The WRITE DMA EXT Command

The WRITE DMA EXT command writes a number of logical blocks of data to the card device using the Data-Out data transfer protocol. The host uses a RW_REG command to issue the ATA command and the RW_BLK command for the data transfer.†

15.5.9.5.4. The STANDBY IMMEDIATE Command

This ATA command causes the card device to immediately enter the most aggressive power management mode that still retains internal device context. No data transfer (RW_BLK) is expected for this command.†

For card devices that do not provide a power savings mode, the STANDBY IMMEDIATE command returns a successful status indication. The host issues a RW_REG command for the ATA command, and the status is retrieved with the SD/SDIO CMD39 or RW_REG command. Only the status field of the ATA task file contains the success status; there is no error status.†

The host controller uses the following settings while sending the RW_REG command for the STANDBY IMMEDIATE ATA command: †

- `cmd` register setting: `data_expected` bit set to 0†
- `cmdarg` register settings: †
  - Bit [31] set to 1 †
  - Bits [7:2] set to 4 †
  - All other bits set to 0 †
- Task file settings: †
  - Command field of the ATA task file set to 0xE0†
  - Reserved fields of the task file set to 0†
- `bytcnt` register and `block_size` field of the `blksiz` register: set to 16 †

15.5.9.5.5. The FLUSH CACHE EXT Command

For card devices that buffer/cache written data, the FLUSH CACHE EXT command ensures that buffered data is written to the card media. For cards that do not buffer written data, the FLUSH CACHE EXT command returns a success status. No data transfer (RW_BLK) is expected for this ATA command. †

The host issues a RW_REG command for the ATA command, and the status is retrieved with the SD/SDIO CMD39 or RW_REG command. There can be error status for this ATA command, in which case fields other than the status field of the ATA task file are valid.†

The host controller uses the following settings while sending the RW_REG command for the STANDBY IMMEDIATE ATA command:†

- `cmd` register setting: `data_expected` bit set to 0 †
- `cmdarg` register settings: †
  - Bit [31] set to 1 †
  - Bits [7:2] set to 4†
  - All other bits set to 0†
- Task file settings: †
  - Command field of the ATA task file set to 0xEA †
  - Reserved fields of the task file set to 0†
- `bytcnt` register and `block_size` field of the `blksiz` register: set to 16 †

15.5.10. Card Read Threshold

When an application needs to perform a single or multiple block read command, the application must set the `cardthrctl` register with the appropriate card read threshold size in the card read threshold field (`cardrdthreshold`) and set the `cardrdthren` bit to 1. This additional information specified in the controller ensures that the controller sends a read command only if there is space equal to the card read threshold available in the RX FIFO buffer. This in turn ensures that the card clock is not stopped in the middle a block of data being transmitted from the card. Set the card read threshold to the block size of the transfer to guarantee there is a minimum of one block size of space in the RX FIFO buffer before the controller enables the card clock. †

The card read threshold is required when the round trip delay is greater than half of `sdmmc_clk_divided`.†

Table 173. Card Read Threshold Guidelines†

Bus Speed Modes	Round Trip Delay (Delay_R) (45)	Is Stopping of Card Clock Allowed?	Card Read Threshold Required?
SDR25	Delay_R > 0.5 * (sdmmc_clk/4) Delay_R < 0.5 * (sdmmc_clk/4)	No Yes	Yes No
SDR12	Delay_R > 0.5 * (sdmmc_clk/4) Delay_R < 0.5 * (sdmmc_clk/4)	No Yes	Yes No

Related Information

Intel Intel Arria 10 Device Datasheet

15.5.10.1. Recommended Usage Guidelines for Card Read Threshold

1. The `cardthrctl` register must be set before setting the `cmd` register for a data read command.†
2. The `cardthrctl` register must not be set while a data transfer command is in progress.†
3. The `cardrdthreshold` field of the `cardthrctl` register must be set to at the least the block size of a single or multiblock transfer. A `cardrdthreshold` field setting greater than or equal to the block size of the read transfer ensures that the card clock does not stop in the middle of a block of data.†
4. If the round trip delay is greater than half of the card clock period, card read threshold must be enabled and the card threshold must be set as per guideline 3 to guarantee that the card clock does not stop in the middle of a block of data.†
5. If the `cardrdthreshold` field is set to less than the block size of the transfer, the host must ensure that the receive FIFO buffer never overflows during the read transfer. Overflow can cause the card clock from the controller to stop. The controller is not able to guarantee that the card clock does not stop during a read transfer.†

Note: If the `cardrdthreshold` field of the `cardthrctl` register, and the `rx_wmark` and `dw_dma_multiple_transaction_size` fields of the `fifoth` register are set incorrectly, the card clock might stop indefinitely, with no interrupts generated by the controller.†

(45) Delay_R = Delay_O + tODLY + Delay_I †
Where: †
Delay_O = `sdmmc_clk` to `sdmmc_cclk_out` delay (including I/O pin delay) †
Delay_I = Input I/O pin delay + routing delay to the input register †
tODLY = `sdmmc_cclk_out` to card output delay (varies across card manufactures and speed modes) †

For the delay numbers needed for above calculation, refer to Arria 10 Datasheet. †

15.5.10.2. Card Read Threshold Programming Sequence

Most cards, such as SDHC or SDXC, support block sizes that are either specified in the card or are fixed to 512 bytes. For SDIO, MMC, and standard capacity SD cards that support partial block read (READ_BL_PARTIAL set to 1 in the CSD register of the card device), the block size is variable and can be chosen by the application.†

To use the card read threshold feature effectively and to guarantee that the card clock does not stop because of a FIFO Full condition in the middle of a block of data being read from the card, follow these steps:†

1. Choose a block size that is a multiple of four bytes.†
2. Enable card read threshold feature. The card read threshold can be enabled only if the block size for the given transfer is less than or equal to the total depth of the FIFO buffer:†

   (block size / 4) ≤ 1024†
3. Choose the card read threshold value: †
   - If (block size / 4) ≥ 512, choose `cardrdthreshold` such that:†
     - `cardrdthreshold` ≤ (block size / 4) in bytes†
   - If (block size / 4) < 512, choose `cardrdthreshold` such that:†
     - `cardrdthreshold` = (block size / 4) in bytes†
4. Set the `dw_dma_multiple_transaction_size` field in the `fifoth` register to the number of transfers that make up a DMA transaction. For example, size = 1 means 4 bytes are moved. The possible values for the size are 1, 4, 8, 16, 32, 64, 128, and 256 transfers. Select the size so that the value (block size / 4) is evenly divided by the size.†
5. Set the `rx_wmark` field in the `fifoth` register to the size – 1.†

For example, if your block size is 512 bytes, legal values of `dw_dma_multiple_transaction_size` and `rx_wmark` are listed in the following table.

Table 174. Legal Values of dw_dma_multiple_transaction_size and rx_wmark for Block Size = 512†

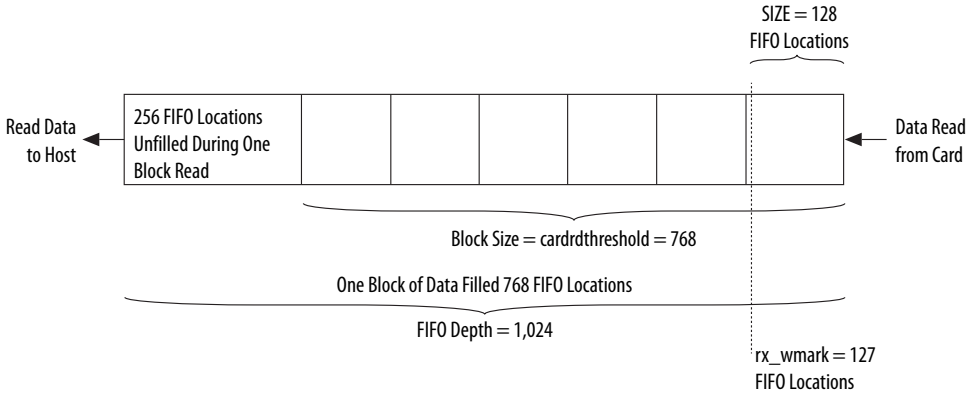
Block Size	dw_dma_multiple_transaction_size	rx_wmark
512	1	0
512	4	3
512	8	7
512	16	15
512	32	31
512	64	63
512	128	127

15.5.10.3. Card Read Threshold Programming Examples

This section shows examples of how to program the card read threshold.†

- Choose a block size that is a multiple of 4 (the number of bytes per FIFO location), and less than 4096 (1024 FIFO locations). For example, a block size of 3072 bytes is legal, because 3072 / 4 = 768 FIFO locations.†
- For DMA mode, choose the size so that block size is a multiple of the size. For example size = 128, where block size%size = 0.†
- Set the `rx_wmark` field = size – 1. For example, the `rx_wmark` field = 128 – 1 = 127.†
- Because block size > ½ FifoDepth, set the `cardrdthreshold` field to the block size. For example, the `cardrdthreshold` field = 3072 bytes.†

Figure 79. FIFO Buffer content when Card Read Threshold is set to 768†



15.5.11. Interrupt and Error Handling

This section describes how to use interrupts to handle errors. On power-on or reset, interrupts are disabled (the `int_enable` bit in the `ctrl` register is set to 0), and all the interrupts are masked (the `intmask` register default is 0). The controller error handling includes the following types of errors:

- Response and data timeout errors—For response time-outs, the host software can retry the command. For data time-outs, the controller has not received the data start bit from the card, so software can either retry the whole data transfer again or retry from a specified block onwards. By reading the contents of the `tcbcnt` register later, the software can decide how many bytes remain to be copied (read). †
- Response errors—Set to 1 when an error is received during response reception. If the response received is invalid, the software can retry the command. †
- Data errors—Set to 1 when a data receive error occurs. Examples of data receive errors: †
  - Data CRC†
  - Start bit not found †
  - End bit not found †

  These errors can be occur on any block. On receipt of an error, the software can issue an SD/SDIO STOP or SEND_IF_COND command, and retry the command for either the whole data or partial data.†

- Hardware locked error—Set to 1 when the controller cannot load a command issued by software. When software sets the `start_cmd` bit in the `cmd` register to 1, the controller tries to load the command. If the command buffer already contains a command, this error is raised, and the new command is discarded, requiring the software to reload the command.†
- FIFO buffer underrun/overrun error—If the FIFO buffer is full and software tries to write data to the FIFO buffer, an overrun error is set. Conversely, if the FIFO buffer is empty and the software tries to read data from the FIFO buffer, an underrun error is set. Before reading or writing data in the FIFO buffer, the software must read the FIFO buffer empty bit (`fifo_empty`) or FIFO buffer full bit (`fifo_full`) in the `status` register.†
- Data starvation by host timeout—This condition occurs when software does not service the FIFO buffer fast enough to keep up with the controller. Under this condition and when a read transfer is in process, the software must read data from the FIFO buffer, which creates space for further data reception. When a transmit operation is in process, the software must write data to fill the FIFO buffer so that the controller can write the data to the card.†
- CE-ATA errors†
- CRC error on command—If a CRC error is detected for a command, the CE-ATA card device does not send a response, and a response timeout is expected from the controller. The ATA layer is notified that an MMC transport layer error occurred.
- CRC error on command—If a CRC error is detected for a command, the CE-ATA card device does not send a response, and a response timeout is expected from the controller. The ATA layer is notified that an MMC transport layer error occurred.†
- Write operation—Any MMC transport layer error known to the card device causes an outstanding ATA command to be terminated. The ERR bits are set in the ATA status registers and the appropriate error code is sent to the Error Register (Error) on the ATA card device.†

  If the device interrupt bit of the CE-ATA card (the nIEN bit in the ATA control register) is set to 0, the CCS is sent to the host.†

  If the device interrupt bit is set to 1, the card device completes the entire data unit count if the host controller does not abort the ongoing transfer.†

  Note: During a multiple-block data transfer, if a negative CRC status is received from the card device, the data path signals a data CRC error to the BIU by setting the `dcrc` bit in the `rintsts` register to 1. It then continues further data transmission until all the bytes are transmitted.†
- Read operation—If MMC transport layer errors are detected by the host controller, the host completes the ATA command with an error status. The host controller can issue a CCSD command followed by a STOP_TRANSMISSION (CMD12) command to abort the read transfer. The host can also transfer the entire data unit count bytes without aborting the data transfer.†

15.5.12. Booting Operation for eMMC and MMC

This section describes how to set up the controller for eMMC and MMC boot operation.

Note: The BootROM and initial software do not use the boot partitions that are in the MMC card. This means that there is no boot partition support of the SD/MMC controller.

15.5.12.1. Boot Operation by Holding Down the CMD Line

The controller can boot from MMC4.3, MMC4.4, and MMC4.41 cards by holding down the CMD line.

For information about this boot method, refer to the following specifications, available on the JEDEC website:

- JEDEC Standard No. 84-A441
- JEDEC Standard No. 84-A44
- JEDEC Standard No. JESD84-A43

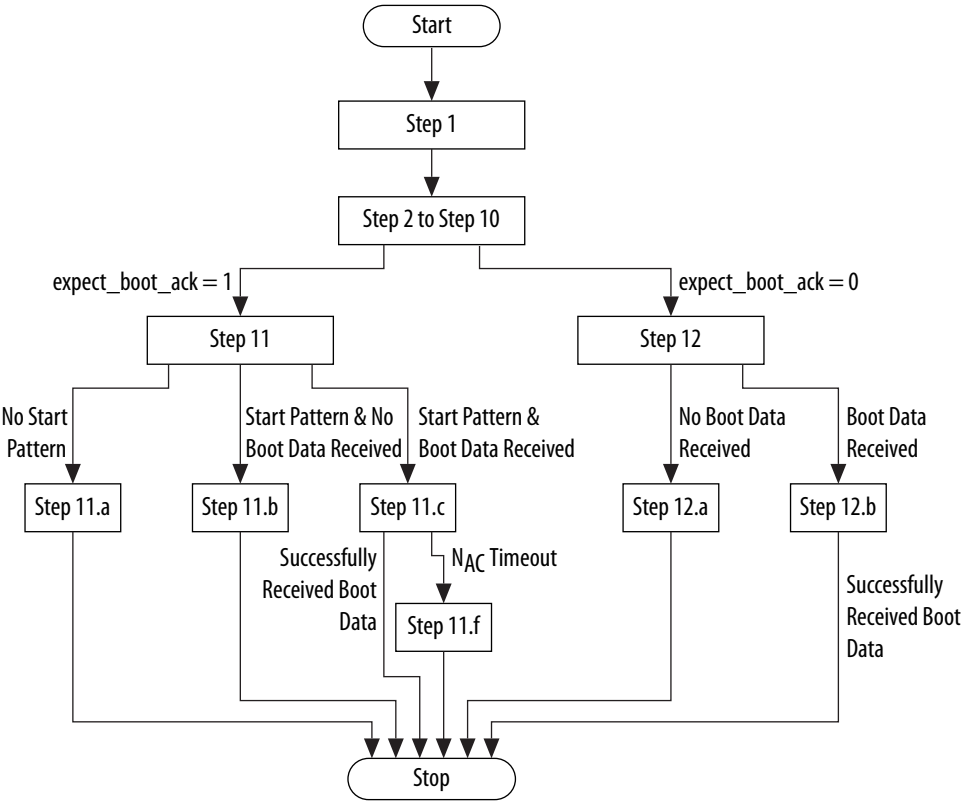
Related Information

JEDEC Global Standards of the Microelectronics Industry
For more information about this boot method, refer to the following JEDEC Standards available on the JEDEC website: No. 84-A441, No. 84-A44, and No. JESD84-A43.

15.5.12.2. Boot Operation for eMMC Card Device

The following figure illustrates the steps to perform the boot process for eMMC card devices. The detailed steps are described following the flow chart.

Figure 80. Flow for eMMC Boot Operation†



1. The software driver performs the following checks: †

- If the eMMC card device supports boot operation (the BOOT_PARTITION_ENABLE bit is set to 1 in the EXT_CSD register of the eMMC card).†
- The BOOT_SIZE_MULT and BOOT_BUS_WIDTH values in the EXT_CSD register, to be used during the boot process.†

2. The software sets the following bits: †
   - Sets masks for interrupts, by setting the appropriate bits to 0 in the `intmask` register.†
   - Sets the global `int_enable` bit of the `ctrl` register to 1. Other bits in the `ctrl` register must be set to 0. †

     Note: Intel recommends that you write 0xFFFFFFFF to the `rintsts` and `idsts` registers to clear any pending interrupts before setting the `int_enable` bit. For internal DMA controller mode, the software driver needs to unmask all the relevant fields in the `idinten` register.†
3. If the software driver needs to use the internal DMA controller to transfer the boot data received, it must perform the following steps: †
   - Set up the descriptors as described in *Internal DMA Controller Transmission Sequences* and *Internal DMA Controller Reception Sequences*". †
   - Set the `use_internal_dmac` bit of the `ctrl` register to 1.†
4. Set the card device frequency to 400 kHz using the `clkdiv` registers. For more information, refer to Clock Setup.†
5. Set the `data_timeout` field of the `tmout` register equal to the card device total access time, N_{AC}. †
6. Set the `blksiz` register to 0x200 (512 bytes). †
7. Set the `bytcnt` register to a multiple of 128 KB, as indicated by the BOOT_SIZE_MULT value in the card device.†
8. Set the `rx_wmark` field in the `fifoth` register. Typically, the threshold value can be set to 512, which is half the FIFO buffer depth.†
9. Set the following fields in the `cmd` register:†
   - Initiate the command by setting `start_cmd` = 1†
   - Enable boot (`enable_boot`) = 1†
   - Expect boot acknowledge (`expect_boot_ack`): †
     - If a start-acknowledge pattern is expected from the card device, set `expect_boot_ack` to 1.†
     - If a start-acknowledge pattern is not expected from the card device, set `expect_boot_ack` to 0.†
   - Card number (`card_number`) = 0†
   - `data_expected` = 1†
   - Reset the remainder of `cmd` register bits to 0†

10. If no start-acknowledge pattern is expected from the card device (`expect_boot_ack` set to 0) proceed to step 12.†
11. This step handles the case where a start-acknowledge pattern is expected (`expect_boot_ack` was set to 1 in step 9).†
    a. If the Boot ACK Received interrupt is not received from the controller within 50 ms of initiating the command (step 9), the software driver must set the following `cmd` register fields: †
       - `start_cmd` = 1†
       - Disable boot (`disable_boot`)= 1†
       - `card_number` = 0 †
       - All other fields = 0†

       The controller generates a Command Done interrupt after deasserting the `CMD` pin of the card interface.†

       If internal DMA controller mode is used for the boot process, the controller performs the following steps after the Boot ACK Received timeout:†
       - The DMA descriptor is closed.†
       - The `ces` bit in the `idsts` register is set, indicating the Boot ACK Received timeout.†
       - The `ri` bit of the `idsts` register is not set.†

    b. If the Boot ACK Received interrupt is received, the software driver must clear this interrupt by writing 1 to the `ces` bit in the `idsts` register.†

       Within 0.95 seconds of the Boot ACK Received interrupt, the Boot Data Start interrupt must be received from the controller. If this does not occur, the software driver must write the following `cmd` register fields:†
       - `start_cmd` = 1†
       - disable_boot = 1†
       - card_number = 0†
       - All other fields = 0†

       The controller generates a Command Done interrupt after deasserting the `CMD` pin of the card interface.†

       If internal DMA controller mode is used for the boot process, the controller performs the following steps after the Boot ACK Received timeout:†
       - The DMA descriptor is closed†
       - The `ces` bit in the `idsts` register is set, indicating Boot Data Start timeout†
       - The `ri` bit of the `idsts` register is not set†

    c. If the Boot Data Start interrupt is received, it indicates that the boot data is being received from the card device. When the DMA engine is not in internal DMA controller mode, the software driver can then initiate a data read from the controller based on the `rxdr` interrupt bit in the `rintsts` register.†

       In internal DMA controller mode, the DMA engine starts transferring the data from the FIFO buffer to the system memory as soon as the level set in the `rx_wmark` field of the `fifoth` register is reached.†

At the end of a successful boot data transfer from the card, the following interrupts are generated:†

- The `cmd` bit and `dto` bit in the `rintsts` register†
- The `ri` bit in the `idsts` register, in internal DMA controller mode only†

d. If an error occurs in the boot ACK pattern (0b010) or an EBE occurs: †

- The controller automatically aborts the boot process by pulling the CMD line high†
- The controller generates a Command Done interrupt†
- The controller does not generate a Boot ACK Received interrupt†
- The application aborts the boot transfer†

e. In internal DMA controller mode:†

- If the software driver creates more descriptors than required by the received boot data, the extra descriptors are not closed by the controller. Software cannot reuse the descriptors until they are closed.†
- If the software driver creates fewer descriptors than required by the received boot data, the controller generates a Descriptor Unavailable interrupt and does not transfer any further data to system memory.†

f. If N_{AC} is violated between data block transfers, the DRTO interrupt is asserted. In addition, if there is an error associated with the start or end bit, the SBE or EBE interrupt is also generated.†

The boot operation for eMMC card devices is complete. Do not execute the remaining (step 12).†

12. This step handles the case where no start-acknowledge pattern is expected (`expect_boot_ack` was set to 0 in step 9).†

a. If the Boot Data Start interrupt is not received from the controller within 1 second of initiating the command (step 9), the software driver must write the `cmd` register with the following fields:†

- `start_cmd` = 1†
- `disable_boot` = 1†
- `card_number` = 0†
- All other fields = 0†

The controller generates a Command Done interrupt after deasserting the CMD line of the card. In internal DMA controller mode, the descriptor is closed and the `ces` bit in the `idsts` register is set to 1, indicating a Boot Data Start timeout.†

b. If a Boot Data Start interrupt is received, it indicates that the boot data is being received from the card device. When the DMA engine is not in internal DMA controller mode, the software driver can then initiate a data read from the controller based on the `rxdr` interrupt bit in the `rintsts` register.†

In internal DMA controller mode, the DMA engine starts transferring the data from the FIFO buffer to the system memory as soon as the level specified in the `rx_wmark` field of the `fifoth` register is reached.†

At the end of a successful boot data transfer from the card, the following interrupts are generated:†

- The `cmd` bit and `dto` bit in the `rintsts` register†
- The `ri` bit in the `idsts` register, in internal DMA controller mode only†

c. In internal DMA controller mode:†

- If the software driver creates more descriptors than required by the received boot data, the extra descriptors are not closed by the controller.†
- If the software driver creates fewer descriptors than required by the received boot data, the controller generates a Descriptor Unavailable interrupt and does not transfer any further data to system memory.†

The boot operation for eMMC card devices is complete.†

Related Information

- Clock Setup on page 449
  Refer to this section for information on how to set the card device frequency.
- Internal DMA Controller Transmission Sequences on page 462
  Refer to this section for information about the Internal DMA Controller Transmission Sequences.
- Internal DMA Controller Reception Sequences on page 463
  Refer to this section for information about the Internal DMA Controller Reception Sequences.

15.5.12.3. Boot Operation for Removable MMC4.3, MMC4.4 and MMC4.41 Cards

15.5.12.3.1. Removable MMC4.3, MMC4.4, and MMC4.41 Differences

Removable MMC4.3, MMC4.4, and MMC4.41 cards differ with respect to eMMC in that the controller is not aware whether these cards support the boot mode of operation when plugged in. Thus, the controller must: †

1. Discover these cards as it would discover MMC4.0/4.1/4.2 cards for the first time†
2. Know the card characteristics †
3. Decide whether to perform a boot operation or not†

15.5.12.3.2. Booting Removable MMC4.3, MMC4.4 and MMC4.41 Cards

For removable MMC4.3, MMC4.4 and MMC4.41 cards, the software driver must perform the following steps:†

1. Discover the card as described in *Enumerated Card Stack*.†
2. Read the EXT_CSD register of the card and examine the following fields: †
   - BOOT_PARTITION_ENABLE †
   - BOOT_SIZE_MULT†
   - BOOT_INFO †
3. If necessary, the software can manipulate the boot information in the card. †

   Note: For more information, refer to "Access to Boot Partition" in the following specifications available on the JEDEC website:

- JEDEC Standard No. 84-A441
- JEDEC Standard No. 84-A44
- JEDEC Standard No. JESD84-A43

4. If the host processor needs to perform a boot operation at the next power-up cycle, it can manipulate the EXT_CSD register contents by using a SWITCH_FUNC command. †
5. After this step, the software driver must power down the card by writing to the pwren register. †
6. From here on, use the same steps as in *Alternative Boot Operation for eMMC Card Devices*.†

Related Information

- Enumerated Card Stack on page 446
  Refer to this section for more information on discovering removable MMC cards.
- JEDEC Global Standards of the Microelectronics Industry
  For more information, refer to "Access to Boot Partition" in the following specifications available on the JEDEC website: No. 84-A441, No. 84-A44, and No. JESD84-A43.
- Alternative Boot Operation for eMMC Card Devices on page 483
  Refer to this section for information about alternative boot operation steps.

15.5.12.4. Alternative Boot Operation

The alternative boot operation differs from the previous boot operation in that software uses the SD/SDIO GO_IDLE_STATE command to boot the card, rather than holding down the CMD line of the card. The alternative boot operation can be performed only if bit 0 in the BOOT_INFO register is set to 1. BOOT_INFO is located at offset 228 in the EXT_CSD registers. †

For detailed information about alternative boot operation, refer to the following specifications available on the JEDEC website:

- JEDEC Standard No. 84-A441
- JEDEC Standard No. 84-A44
- JEDEC Standard No. JESD84-A43

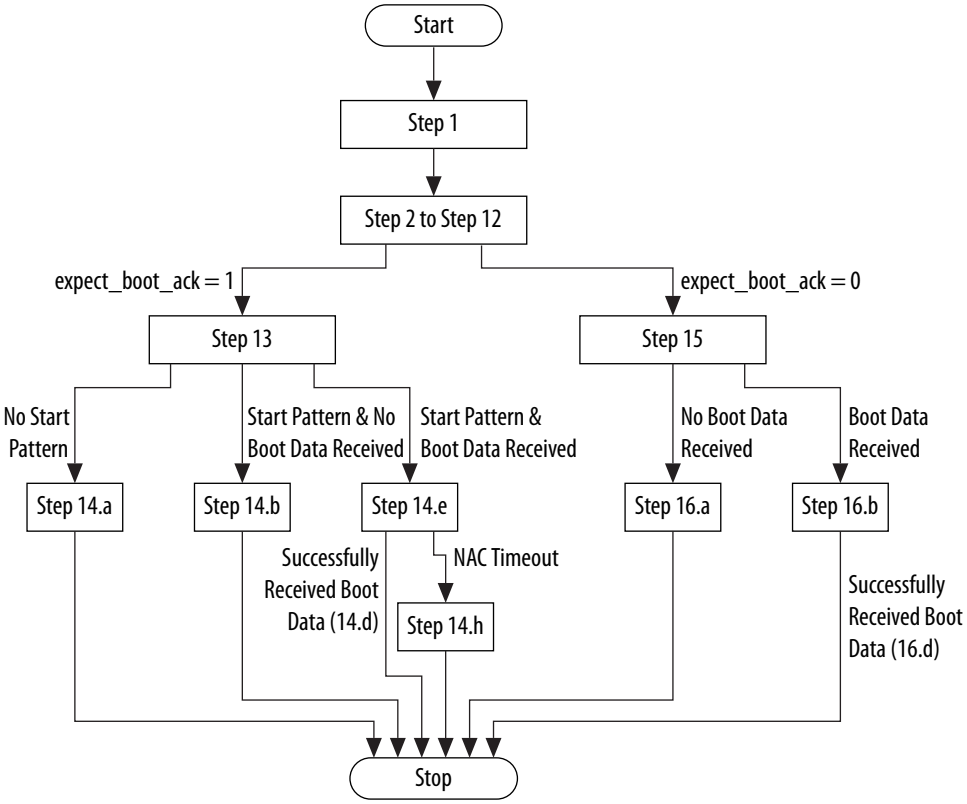
Related Information

JEDEC Global Standards of the Microelectronics Industry
For more information about alternative boot operation, refer to the following JEDEC Standards available on the JEDEC website: No. 84-A441, No. 84-A44, and No. JESD84-A43.

15.5.12.5. Alternative Boot Operation for eMMC Card Devices

The following figure illustrates the sequence of steps required to perform the alternative boot operation for eMMC card devices. The detailed steps are described following the flow chart.

Figure 81. Flow for eMMC Alternative Boot Operation†



1. The software driver checks:†
   - If the eMMC card device supports alternative boot operation (the BOOT_INFO bit is set to 1 in the eMMC card).†
   - The BOOT_SIZE_MULT and BOOT_BUS_WIDTH values in the card device to use during the boot process.†
2. The software sets the following bits: †
   - Sets masks for interrupts by resetting the appropriate bits to 0 in the `intmask` register.†
   - Sets the `int_enable` bit of the `ctrl` register to 1. Other bits in the `ctrl` register must be set to 0. †

     Note: Intel recommends writing 0xFFFFFFFF to the `rintsts` register and `idsts` register to clear any pending interrupts before setting the `int_enable` bit. For internal DMA controller mode, the software driver needs to unmask all the relevant fields in the `idinten` register.†
3. If the software driver needs to use the internal DMA controller to transfer the boot data received, it must perform the following actions: †
   - Set up the descriptors as described in *Internal DMA Controller Transmission Sequences* and *Internal DMA Controller Reception Sequences*. †
   - Set the use internal DMAC bit (`use_internal_dmac`) of the `ctrl` register to 1. †

4. Set the card device frequency to 400 kHz using the `clkdiv` registers. For more information, refer to *Clock Setup*. Ensure that the card clock is running.†
5. Wait for a time that ensures that at least 74 card clock cycles have occurred on the card interface.†
6. Set the `data_timeout` field of the `tmout` register equal to the card device total access time, N_{AC}. †
7. Set the `blksiz` register to 0x200 (512 bytes).†
8. Set the `bytcnt` register to multiples of 128K bytes, as indicated by the BOOT_SIZE_MULT value in the card device.†
9. Set the `rx_wmark` field in the `fifoth` register. Typically, the threshold value can be set to 512, which is half the FIFO buffer depth.†
10. Set the `cmdarg` register to 0xFFFFFFFA. †
11. Initiate the command, by setting the `cmd` register with the following fields: †
    - `start_cmd` = 1†
    - `enable_boot` = 1†
    - `expect_boot_ack`:†
      — If a start-acknowledge pattern is expected from the card device, set `expect_boot_ack` to 1.†
      — If a start-acknowledge pattern is not expected from the card device, set `expect_boot_ack` to 0. †
    - `card_number` = 0 †
    - `data_expected` = 1†
    - `cmd_index` = 0†
    - Set the remainder of `cmd` register bits to 0.†
12. If no start-acknowledge pattern is expected from the card device (`expect_boot_ack` set to 0) jump to step 15. †
13. Wait for the Command Done interrupt.†
14. This step handles the case where a start-acknowledge pattern is expected (`expect_boot_ack` was set to 1 in step 11).†
    a. If the Boot ACK Received interrupt is not received from the controller within 50 ms of initiating the command (step 11), the start pattern was not received. The software driver must discontinue the boot process and start with normal discovery. †

       If internal DMA controller mode is used for the boot process, the controller performs the following steps after the Boot ACK Received timeout:†
       - The DMA descriptor is closed.†
       - The `ces` bit in the `idsts` register is set to 1, indicating the Boot ACK Received timeout.†
       - The `ri` bit of the `idsts` register is not set.†

    b. If the Boot ACK Received interrupt is received, the software driver must clear this interrupt by writing 1 to it. †

Within 0.95 seconds of the Boot ACK Received interrupt, the Boot Data Start interrupt must be received from the controller. If this does not occur, the software driver must discontinue the boot process and start with normal discovery.†

If internal DMA controller mode is used for the boot process, the controller performs the following steps after the Boot ACK Received timeout: †

- The DMA descriptor is closed.†
- The `ces` bit in the `idsts` register is set to 1, indicating Boot Data Start timeout.†
- The `ri` bit of the `idsts` register is not set.†

c. If the Boot Data Start interrupt is received, it indicates that the boot data is being received from the card device. When the DMA engine is not in internal DMA controller mode, the software driver can then initiate a data read from the controller based on the `rxdr` interrupt bit in the `rintsts` register.†

   In internal DMA controller mode, the DMA engine starts transferring the data from the FIFO buffer to the system memory as soon as the level specified in the `rx_wmark` field of the `fifoth` register is reached. †

d. The software driver must terminate the boot process by instructing the controller to send the SD/SDIO GO_IDLE_STATE command:†

   - Reset the `cmdarg` register to 0.†
   - Set the `start_cmd` bit of the `cmd` register to 1, and all other bits to 0.†

e. At the end of a successful boot data transfer from the card, the following interrupts are generated: †

   - The `cmd` bit and `dto` bit in the `rintsts` register†
   - The `ri` bit in the `idsts` register, in internal DMA controller mode only†

f. If an error occurs in the boot ACK pattern (0b010) or an EBE occurs: †

   - The controller does not generate a Boot ACK Received interrupt. †
   - The controller detects Boot Data Start and generates a Boot Data Start interrupt. †
   - The controller continues to receive boot data. †
   - The application must abort the boot process after receiving a Boot Data Start interrupt.†

g. In internal DMA controller mode: †

   - If the software driver creates more descriptors than required by the received boot data, the extra descriptors are not closed by the controller. †
   - If the software driver creates fewer descriptors than required by the received boot data, the controller generates a Descriptor Unavailable interrupt and does not transfer any further data to system memory.†

h. If N_{AC} is violated between data block transfers, a DRTO interrupt is asserted. Apart from this, if there is an error associated with the start or end bit, the SBE or EBE interrupt is also generated.†

The alternative boot operation for eMMC card devices is complete. Do not execute the remaining steps (15 and 16). †

15. Wait for the Command Done interrupt.†
16. This step handles the case where a start-acknowledge pattern is not expected (`expect_boot_ack` was set to 0 in step 11). †
    a. If the Boot Data Start interrupt is not received from the controller within 1 second of initiating the command (step 11), the software driver must discontinue the boot process and start with normal discovery. † In internal DMA controller mode:†
       - The DMA descriptor is closed.†
       - The `ces` bit in the `idsts` register is set to 1, indicating Boot Data Start timeout.†
       - The `ri` bit of the `idsts` register is not set.†
    b. If a Boot Data Start interrupt is received, the boot data is being received from the card device. When the DMA engine is not in internal DMA controller mode, the software driver can then initiate a data read from the controller based on the `rxdr` interrupt bit in the `rintsts` register.†

       In internal DMA controller mode, the DMA engine starts transferring the data from the FIFO buffer to the system memory as soon as the level specified in the `rx_wmark` field of the `fifoth` register is reached.†
    c. The software driver must terminate the boot process by instructing the controller to send the SD/SDIO GO_IDLE_STATE (CMD0) command: †
       - Reset the `cmdarg` register to 0.†
       - Set the `start_cmd` bit in the `cmd` register to 1, and all other bits to 0.†
    d. At the end of a successful boot data transfer from the card, the following interrupts are generated: †
       - The `cmd` bit and `dto` bit in the `rintsts` register†
       - The `ri` bit in the `idsts` register, in internal DMA controller mode only †
    e. In internal DMA controller mode: †
       - If the software driver creates more descriptors than required by the received boot data, the extra descriptors are not closed by the controller. †
       - If the software driver creates fewer descriptors than required by the received boot data, the controller generates a Descriptor Unavailable interrupt and does not transfer any further data to system memory.†

The alternative boot operation for eMMC card devices is complete.†

Related Information

- Clock Setup on page 449
  Refer to this section for information on how to set the card device frequency.
- Internal DMA Controller Transmission Sequences on page 462
  Refer to this section for information about the Internal DMA Controller Transmission Sequences.
- Internal DMA Controller Reception Sequences on page 463
  Refer to this section for information about the Internal DMA Controller Reception Sequences.

15.5.12.6. Alternative Boot Operation for MMC4.3 Cards

15.5.12.6.1. Removable MMC4.3 Boot Mode Support

Removable MMC4.3 cards differ with respect to eMMC in that the controller is not aware whether these cards support the boot mode of operation. Thus, the controller must: †

1. Discover these cards as it would discover MMC4.0/4.1/4.2 cards for the first time †
2. Know the card characteristics †
3. Decide whether to perform a boot operation or not†

15.5.12.6.2. Discovering Removable MMC4.3 Boot Mode Support

For removable MMC4.3 cards, the software driver must perform the following steps: †

1. Discover the card as described in *Enumerated Card Stack*.†
2. Read the MMC card device's EXT_CSD registers and examine the following fields: †
   - BOOT_PARTITION_ENABLE †
   - BOOT_SIZE_MULT †
   - BOOT_INFO †

   Note: For more information, refer to "Access to Boot Partition" in JEDEC Standard No. JESD84-A43, available on the JEDEC website.†
3. If the host processor needs to perform a boot operation at the next power-up cycle, it can manipulate the contents of the EXT_CSD registers in the MMC card device, by using a SWITCH_FUNC command. †
4. After this step, the software driver must power down the card by writing to the `pwren` register. †
5. From here on, use the same steps as in *Alternative Boot Operation for eMMC Card Devices*. †

   Note: Ignore the EBE if it is generated during an abort scenario.

   If a boot acknowledge error occurs, the boot acknowledge received interrupt times out. †

   In internal DMA controller mode, the application needs to depend on the descriptor close interrupt instead of the data done interrupt. †

Related Information

- Enumerated Card Stack on page 446
  Refer to this section for more information on discovering removable MMC cards.
- JEDEC Global Standards of the Microelectronics Industry
  For more information, refer to "Access to Boot Partition" in JEDEC Standard No. JESD84-A43, available on the JEDEC website.
- Alternative Boot Operation for eMMC Card Devices on page 483
  Refer to this section for information about alternative boot operation steps.

15.6. SD/MMC Controller Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

Related Information

Error Checking and Correction Controller on page 345
For more information about how to program the ECC registers, refer to this chapter.

16. Quad SPI Flash Controller

The hard processor system (HPS) provides a quad serial peripheral interface (SPI) flash controller for access to serial NOR flash devices. The quad SPI flash controller supports standard SPI flash devices as well as high-performance dual and quad SPI flash devices. The quad SPI flash controller is based on Cadence Quad SPI Flash Controller (QSPI_FLASH_CTRL).

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

16.1. Features of the Quad SPI Flash Controller

The quad SPI flash controller supports the following features:

- SPIx1, SPIx2, or SPIx4 (quad SPI) serial NOR flash devices
- Any device clock frequencies up to 108 MHz(46)
- Direct access and indirect access modes
- Single I/O, dual I/O, or quad I/O instructions
- Up to four chip selects
- DMA transfers using the HPS DMA controller
- Configurable clock polarity and phase
- Programmable write-protected regions
- Programmable delays between transactions
- Programmable device sizes
- Read data capture tuning
- Local buffering with ECC logic for indirect transfers
- Support eXecute-In-Place(XIP) mode
- Supports the Micron N25Q00AA11GSF40F (1024 Mb, 108 MHz) Quad SPI flash memory that is verified working with the HPS

Related Information

Supported Flash Devices for Arria 10 SoC
For more information, refer to the supported QSPI flash devices section on this page.

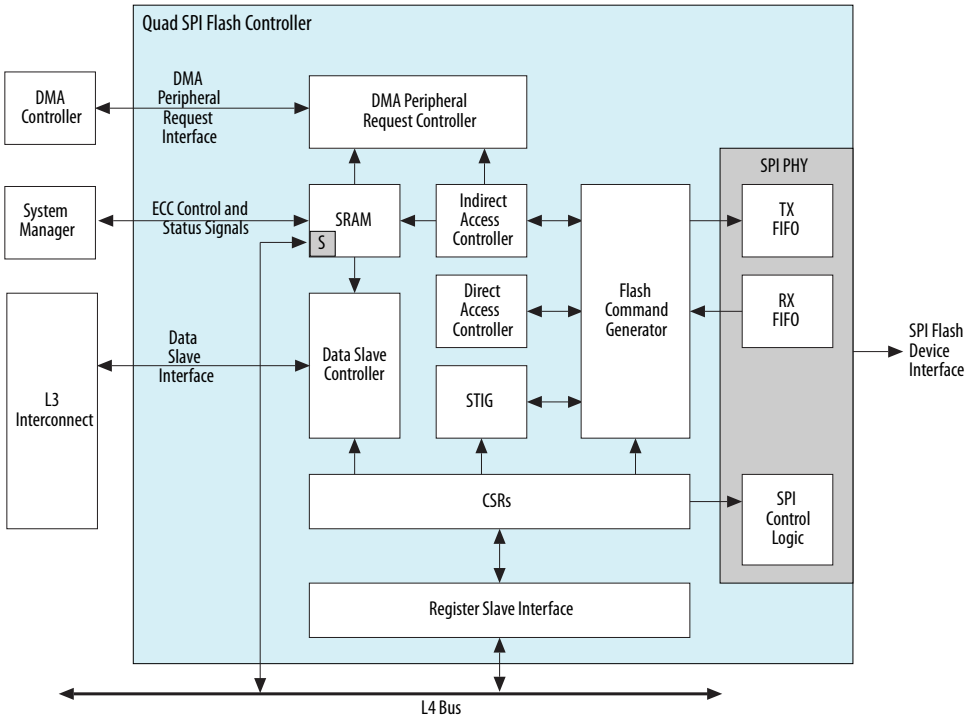
(46) The quad SPI controller supports any device frequencies, but this speed is limited by the supported flash devices.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

16.2. Quad SPI Flash Controller Block Diagram and System Integration

Figure 82. Quad SPI Flash Controller Block Diagram and System Integration



The quad SPI controller consists of the following blocks and interfaces:

- Register slave interface - Provides access to the control and status registers (CSRs)
- Data slave controller - Interface and controller that provides the following functionality:
  - — Performs data transfers to and from the level 3 (L3) interconnect
  - — Validates incoming accesses
  - — Performs byte or half-word reordering
  - — Performs write protection
  - — Forwards transfer requests to direct and indirect controller
- Direct access controller - provides memory-mapped slaves direct access to the flash memory
- Indirect access controller - provides higher-performance access to the flash memory through local buffering and software transfer requests
- Software triggered instruction generator (STIG) - generates flash commands through the flash command register (`flashcmd`) and provides low-level access to flash memory

- Flash command generator - generates flash command and address instructions based on instructions from the direct and indirect access controllers or the STIG
- DMA peripheral request controller - issues requests to the DMA peripheral request interface to communicate with the HPS DMA controller
- SPI PHY - serially transfers data and commands to the external SPI flash devices

The static RAM (SRAM) connected to the indirect access controller, DMA peripheral request controller, and data slave controller has an error correction code (ECC) controller built-in to provide ECC protection. The ECC controller is able to detect single-bit and double-bit errors, and correct the single-bit errors. The ECC operation and functionality is programmable via the ECC register slave interface, as shown in Figure 82 on page 491. The ECC register interface provides host access to configure the ECC logic as well as inject bit errors into the memory. It also provides host access to memory initialization hardware used to clear out the memory contents including the ECC bits. The ECC controller generates interrupts upon occurrences of single and double-bit errors, and the interrupt signals are connected to the system manager.

16.3. Quad SPI Flash Controller Signal Description

The quad SPI controller provides four chip select outputs to allow control of up to four external quad SPI flash devices. The outputs serve different purposes depending on whether the device is used in single, dual, or quad operation mode. The following table lists the I/O pin use of the quad SPI controller interface signals for each operation mode. For more information on which signals are available to the FPGA and HPS I/O, refer to the *HPS Component Interfaces* chapter.

Table 175. Interface Pins

Signal	Pin	Mode	Direction	Function
qspi_io0_i qspi_io0_o qspi_mo_oe[0]	IO0	Single	Output	Data output 0
		Dual or quad	Bidirectional	Data I/O 0
qspi_io1_i qspi_io1_o qspi_mo_oe[1]	IO1	Single	Input	Data input 0
		Dual or quad	Bidirectional	Data I/O 1
qspi_io2_i qspi_io2_wpn_o qspi_mo_oe[2]	IO2_WPN	Single or dual	Output	Active low write protect
		Quad	Bidirectional	Data I/O 2
qspi_io3_i qspi_io3_hold_o qspi_mo_oe[3]	IO3_HOLD	Single, dual, or quad	Bidirectional	Data I/O 3
qspi_ss_o[0]	SS0	Single, dual, or quad	Output	Active low slave select 0
qspi_ss_o[1]	SS1			Active low slave select 1
qspi_ss_o[2]	SS2			Active low slave select 2

continued...

Signal	Pin	Mode	Direction	Function
qspi_ss_o[3]	SS3			Active low slave select 3
qspi_sclk_out	CLK	Single	Output	QSPI serial clock output which connects to the FPGA core fabric
qspi_s2f_clk				QSPI serial clock output which connects to the FPGA clock network *Note:* Use the qspi_s2f_clk as the serial clock output to an external SPI device.

Related Information

HPS Component Interfaces on page 738
For more information on routing Quad SPI Flash Controller interface signals to the FPGA and HPS I/O, refer to this chapter.

16.4. Functional Description of the Quad SPI Flash Controller

16.4.1. Overview

The quad SPI flash controller uses the register slave interface to select the operation modes and configure the data slave interface for data transfers. The quad SPI flash controller uses the data slave interface for direct and indirect accesses, and the register slave interface for software triggered instruction generator (STIG) operation and SPI legacy mode accesses.

Accesses to the data slave are forwarded to the direct or indirect access controller. If the access address is within the configured indirect address range, the access is sent to the indirect access controller.

16.4.2. Data Slave Interface

The quad SPI flash controller uses the data slave interface for direct, indirect, and SPI legacy mode accesses.

The data slave interface is 32 bits wide and permits byte, half-word, and word accesses. For write accesses, incrementing burst lengths of 1, 4, 8 and 16 are supported. For read accesses, all burst types and sizes are supported.

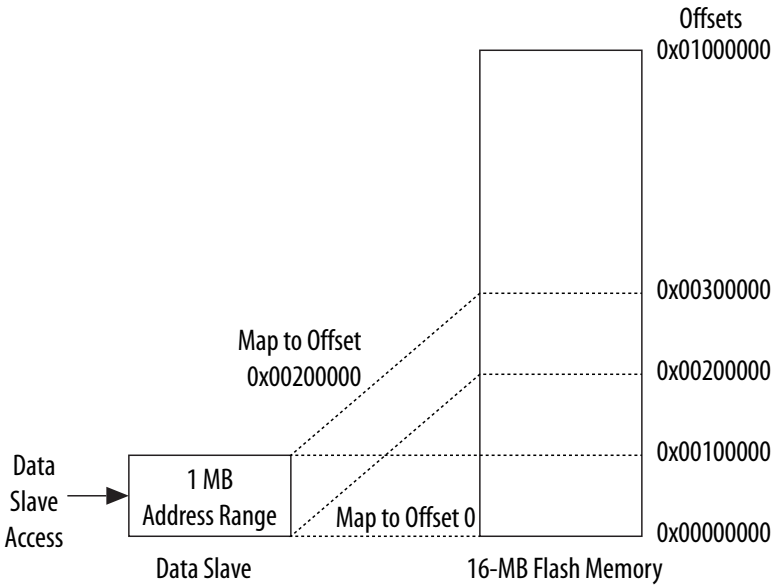
16.4.2.1. Direct Access Mode

In direct access mode, an access to the data slave triggers a read or write command to the flash memory. To use the direct access mode, enable the direct access controller with the enable direct access controller bit (endiracc) of the quad SPI configuration register (cfg).

An external master, for example a processor, triggers the direct access controller with a read or write operation to the data slave interface. The data slave exposes a 1 MB window into the flash device. You can remap this window to any 1 MB location within the flash device.

16.4.2.1.1. Data Slave Remapping Example

Figure 83. Data Slave Remapping Example



To remap the data slave to access other 1 MB regions of the flash device, enable address remapping in the enable Arm AMBA advanced high speed bus (AHB) address remapping field (`enahbremap`) of the `cfg` register. All incoming data slave accesses remap to the offset specified in the remap address register (`remapaddr`).

The 20 LSBs of incoming addresses are used for accessing the 1 MB region and the higher bits are ignored.

Note: The quad SPI controller does not issue any error status for accesses that lie outside the connected flash memory space.

16.4.2.1.2. AHB

The data slave interface is throttled as the read or write burst is carried out. The latency is designed to be as small as possible and is kept to a minimum when the use of XIP read instructions are enabled.

FLASH erase operations, which may be required before a page write, are triggered by software using the documented programming interface. They are not issued automatically.

Once a page program cycle has been started, the QSPI Flash Controller will automatically poll for the write cycle to complete before allowing any further data slave interface accesses to complete. This is achieved by holding any subsequent AHB direct accesses in wait state.

16.4.2.2. Indirect Access Mode

In indirect access mode, flash data is temporarily buffered in the quad SPI controller's static RAM (SRAM). Software controls and triggers indirect accesses through the register slave interface. The controller transfers data through the data slave interface.

16.4.2.2.1. Indirect Read Operation

An indirect read operation reads data from the flash memory, places the data into the SRAM, and transfers the data to an external master through the data slave interface. The indirect read operations are controlled by the following registers:

- Indirect read transfer register (`indrd`)
- Indirect read transfer watermark register (`indrdwater`)
- Indirect read transfer start address register (`indrdstaddr`)
- Indirect read transfer number bytes register (`indrdcnt`)
- Indirect address trigger register (`indaddrtrig`)

These registers need to be configured prior to issuing indirect read operations. The start address needs to be defined in the `indrdstaddr` register and the total number of bytes to be fetched is specified in the `indircnt` register. Writing 1 to the start indirect read bit (`start`) of the `indrd` register triggers the indirect read operation from the flash memory to populate the SRAM with the returned data.

To read data from the flash device into the SRAM, an external master issues 32-bit read transactions to the data slave interface. The address of the read access must be in the indirect address range. You can configure the indirect address through the `indaddrtrig` register. The external master can issue 32-bit reads until the last word of an indirect transfer. On the final read, the external master may issue a 32-bit, 16-bit or 8-bit read to complete the transfer. If there are less than four bytes of data to read on the last transfer, the external master can still issue a 32-bit read and the quad SPI controller pads the upper bits of the response data with zeros.

Assuming the requested data is present in the SRAM at the time the data slave read is received by the quad SPI controller, the data is fetched from SRAM and the response to the read burst is achieved with minimum latency. If the requested data is not immediately present in the SRAM, the data slave interface enters a wait state until the data has been read from flash memory into SRAM. Once the data has been read from SRAM by the external master, the quad SPI controller frees up the associated resource in the SRAM. If the SRAM is full, reads on the SPI interface are backpressured until space is available in the SRAM. The quad SPI controller completes any current read burst, waits for SRAM to free up, and issues a new read burst at the address where the previous burst was terminated.

The processor can also use the SRAM fill level in the SRAM fill register (`sramfill`) to control when data should be fetched from the SRAM.

Alternatively, you can configure the fill level watermark of the SRAM in the `indrdwater` register. When the SRAM fill level passes the watermark level, the indirect transfer watermark interrupt is generated. You can disable this watermark feature by writing a value of all zeroes to the `indrdwater` register.

For the final bytes of data read by the quad SPI controller and placed in the SRAM, if the watermark level is greater than zero, the indirect transfer watermark interrupt is generated even when the actual SRAM fill level has not risen above the watermark.

If the address of the read access is outside the range of the indirect trigger address, one of the following actions occurs:

- When direct access mode is enabled, the read uses direct access mode.
- When direct access mode is disabled, the slave returns an error back to the requesting master.

You can cancel an indirect operation by setting the cancel indirect read bit (`cancel`) of the `indrd` register to 1. For more information, refer to the "Indirect Read Operation with DMA Disabled" section.

Related Information

Indirect Read Operation with DMA Disabled on page 507

16.4.2.2.2. Indirect Write Operation

An indirect write operation programs data from the SRAM to the flash memory. The indirect write operations are controlled by the following registers:

- Indirect write transfer register (`indwr`)
- Indirect write transfer watermark register (`indwrwater`)
- Indirect write transfer start address register (`indwrstaddr`)
- Indirect write transfer number bytes register (`indwrcnt`)
- `indaddrtrig` register

These registers need to be configured prior to issuing indirect write operations. The start address needs to be defined in the `indwrstaddr` register and the total number of bytes to be written is specified in the `indwrcnt` register. The start indirect write bit (`start`) of the `indwr` register triggers the indirect write operation from the SRAM to the flash memory.

To write data from the SRAM to the flash device, an external master issues 32-bit write transactions to the data slave. The address of the write access must be in the indirect address range. You can configure the indirect address through the `indaddrtrig` register. The external master can issue 32-bit writes until the last word of an indirect transfer. On the final write, the external master may issue a 32-bit, 16-bit or 8-bit write to complete the transfer. If there are less than four bytes of data to write on the last transfer, the external master can still issue a 32-bit write and the quad SPI controller discards the extra bytes.

The SRAM size can limit the amount of data that the quad SPI controller can accept from the external master. If the SRAM is not full at the point of the write access, the data is pushed to the SRAM with minimum latency. If the external master attempts to push more data to the SRAM than the SRAM can accept, the quad SPI controller backpressures the external master with wait states. When the SRAM resource is freed up by pushing the data from SRAM to the flash memory, the SRAM is ready to receive more data from the external master. When the SRAM holds an equal or greater number of bytes than the size of a flash page, or when the SRAM holds all the remaining bytes of the current indirect transfer, the quad SPI controller initiates a write operation to the flash memory.

The processor can also use the SRAM fill level, in the `sramfill` register, to control when to write more data into the SRAM.

Alternatively, you can configure the fill level watermark of the SRAM in the `indwrwater` register. When the SRAM fill level falls below the watermark level, an indirect transfer watermark interrupt is generated to tell the software to write the next page of data to the SRAM. Because the quad SPI controller initiates non-end-of-data writes to the flash memory only when the SRAM contains a full flash page of data, you must set the watermark level to a value greater than one flash page to avoid the system stalling. You can disable this watermark feature by writing a value of all ones to the `indwrwater` register.

If the address of the write access is outside the range of the indirect trigger address, one of the following actions occurs:

- When direct access mode is enabled, the write uses direct access mode.
- When direct access mode is disabled, the slave returns an error back to the requesting master.

You can cancel an indirect operation by setting the cancel indirect write bit (`cancel`) of the `indwr` register to 1. For more information, refer to the "Indirect Write Operation with DMA Disabled" section.

Related Information

Indirect Write Operation with DMA Disabled on page 508

16.4.2.2.3. Consecutive Reads and Writes

It is possible to trigger two indirect operations at a time by triggering the `start` bit of the `indrd` or `indwr` register twice in short succession. The second operation can be triggered while the first operation is in progress. For example, software may trigger an indirect read or write operation while an indirect write operation is in progress. The corresponding start and count registers must be configured properly before software triggers each transfer operation.

This approach allows for a short turnaround time between the completion of one indirect operation and the start of a second operation. Any attempt to queue more than two operations causes the indirect read reject interrupt to be generated.

16.4.3. SPI Legacy Mode

SPI legacy mode allows software to access the internal TX FIFO and RX FIFO buffers directly, thus bypassing the direct, indirect and STIG controllers. Software accesses the TX FIFO and RX FIFO buffers by writing any value to any address through the data slave while legacy mode is enabled. You can enable legacy mode with the legacy IP mode enable bit (`enlegacyip`) of the `cfg` register.

Legacy mode allows the user to issue any flash instruction to the flash device, but imposes a heavy software overhead in order to manage the fill levels of the FIFO buffers effectively. The legacy SPI mode is bidirectional in nature, with data continuously being transferred both directions while the chip select is enabled. If the driver only needs to read data from the flash device, dummy data must be written to ensure the chip select stays active, and vice versa for write transactions.

For example, to perform a basic read of four bytes to a flash device that has three address bytes, software must write a total of eight bytes to the TX FIFO buffer. The first byte would be the instruction opcode, the next three bytes are the address, and the final four bytes would be dummy data to ensure the chip select stays active while

the read data is returned. Similarly, because eight bytes were written to the TX FIFO buffer, software should expect eight bytes to be returned in the RX FIFO buffer. The first four bytes of this would be discarded, leaving the final four bytes holding the data read from the device.

Because the TX FIFO and RX FIFO buffers are four bytes deep each, software must maintain the FIFO buffer levels to ensure the TX FIFO buffer does not underflow and the RX FIFO buffer does not overflow. Interrupts are provided to indicate when the fill levels pass the watermark levels, which are configurable through the TX threshold register (`txtresh`) and RX threshold register (`rxtresh`).

16.4.4. Register Slave Interface

The quad SPI flash controller uses the register slave interface to configure the quad SPI controller through the quad SPI configuration registers, and to access flash memory under software control, through the `flashcmd` register in the STIG.

16.4.4.1. STIG Operation

The Software Triggered Instruction Generator (STIG) is used to access the volatile and non-volatile configuration registers, the legacy SPI status register, and other status and protection registers. The STIG also is used to perform ERASE functions. The direct and indirect access controllers are used only to transfer data. The `flashcmd` register uses the following parameters to define the command to be issued to the flash device:

- Instruction opcode
- Number of address bytes
- Number of dummy bytes
- Number of write data bytes
- Write data
- Number of read data bytes

The address is specified through the flash command address register (`flashcmdaddr`). Once these settings have been specified, software can trigger the command with the execute command field (`execcmd`) of the `flashcmd` register and wait for its completion by polling the command execution status bit (`cmdexecstat`) of the `flashcmd` register. A maximum of eight data bytes may be read from the flash command read data lower (`flashcmdrddatalo`) and flash command read data upper (`flashcmdrddataup`) registers or written to the flash command write data lower (`flashcmdwrdatalo`) and flash command write data upper (`flashcmdwrdataup`) registers per command.

Commands issued through the STIG have a higher priority than all other read accesses and therefore interrupt any read commands being requested by the direct or indirect controllers. However, the STIG does not interrupt a write sequence that may have been issued through the direct or indirect access controller. In these cases, it might take a long time for the `cmdexecstat` bit of the `flashcmd` register indicates the operation is complete.

Note: Intel recommends using the STIG instead of the SPI legacy mode to access the flash device registers and perform erase operations.

16.4.5. Local Memory Buffer

The SRAM local memory buffer is a 128 by 32-bit (512 total bytes) memory and includes support for error correction code (ECC). The ECC controller is able to detect single-bit and double-bit errors, and correct the single-bit errors. The ECC operation and functionality is programmable through the ECC register slave interface, as shown in Figure 82 on page 491. The ECC register interface provides host access to configure the ECC logic as well as inject bit errors into the memory. It also provides host access to memory initialization hardware used to clear out the memory contents including the ECC bits.

To prevent spurious ECC errors, software must use the memory initialization block in the ECC controller to initialize the entire memory data and ECC bits. The initialization block clears the memory data. Enabling initialization in the ECC Control Register is independent of enabling the ECC. The ECC controller generates interrupts upon occurrences of single- and double-bit errors. The interrupt requests are sent to the system manager which routes them to the General Interrupt Controller (GIC).

The SRAM has two partitions, with the lower partition reserved for indirect read operations and the upper partition for indirect write operations. The size of the partitions is specified in the SRAM partition register (`srampart`), based on 32-bit word sizes. For example, to specify four bytes of storage, write the value 1. The value written to the indirect read partition size field (`addr`) defines the number of entries reserved for indirect read operations. For example, write the value 32 (0x20) to partition the 128-entry SRAM to 32 entries (25%) for read usage and 96 entries (75%) for write usage.

Related Information

- Memory Data Initialization on page 349
  For more information about clearing memory data before enabling ECC, refer to "Memory Data Initialization" in the Error Checking and Correction section.
- Error Checking and Correction Controller on page 345
  For more information about ECC, refer to the *Error Checking and Correction Controller* chapter of the Arria 10 Device Handbook.

16.4.6. DMA Peripheral Request Controller

The DMA peripheral request controller is only used for the indirect mode of operation where data is temporarily stored in the SRAM. The quad SPI flash controller uses the DMA peripheral request interface to trigger the external DMA into performing data transfers between memory and the quad SPI controller.

There are two DMA peripheral request interfaces, one for indirect reads and one for indirect writes. The DMA peripheral request controller can issue two types of DMA requests, single or burst, to the external DMA. The number of bytes for each single or burst request is specified in the number of single bytes (`numsglreqbytes`) and number of burst bytes (`numburstreqbytes`) fields of the DMA peripheral register (`dmaper`). The DMA peripheral request controller splits the total amount of data to be transferred into a number of DMA burst and single requests by dividing the total number of bytes by the number of bytes specified in the burst request, and then dividing the remainder by the number of bytes in a single request.

Note: When programming the DMA controller, the burst request size must match the burst request size set in the quad SPI controller to avoid quickly reaching an overflow or underflow condition.

For indirect reads, the DMA peripheral request controller only issues DMA requests after the data has been retrieved from flash memory and written to SRAM. The rate at which DMA requests are issued depends on the watermark level. The `indrdwater` register defines the minimum fill level in bytes at which the DMA peripheral request controller can issue the DMA request. The higher this number is, the more data that must be buffered in SRAM before the external DMA moves the data. When the SRAM fill level passes the watermark level, the transfer watermark reached interrupt is generated.

For example, consider the following conditions:

- The total amount of data to be read using indirect mode is 256 bytes
- The SRAM watermark level is set at 128 bytes
- Software configures the burst type transfer size to 64 bytes

Under these conditions, the DMA peripheral request controller issues the first DMA burst request when the SRAM fill level passes 128 bytes (the watermark level). The DMA peripheral request controller triggers consecutive DMA burst requests as long as there is sufficient data in the SRAM to perform burst type requests. In this example, DMA peripheral request controller can issue at least two consecutive DMA burst requests to transfer a total of 128 bytes. If there is sufficient data in the SRAM, the DMA peripheral request controller requests the third DMA burst immediately. Otherwise the DMA peripheral request controller waits for the SRAM fill level to pass the watermark level again to trigger the next burst request. When the watermark level is triggered, there is sufficient data in the SRAM to perform the third and fourth burst requests to complete the entire transaction.

For the indirect writes, the DMA peripheral request controller issues DMA requests immediately after the transfer is triggered and continues to do so until the entire indirect write transfer has been transferred. The rate at which DMA requests are issued depends on the watermark level. The `indwrwater` register defines the maximum fill level in bytes at which the controller can issue the first DMA burst or single request. When the SRAM fill level falls below the watermark level, the transfer watermark reached interrupt is generated. When there is one flash page of data in the SRAM, the quad SPI controller initiates the write operation from SRAM to the flash memory.

Software can disable the DMA peripheral request interface with the `endma` field of the `cfg` register. If a master other than the DMA performs the data transfer for indirect operations, the DMA peripheral request interface must be disabled. By default, the indirect watermark registers are set to zero, which means the DMA peripheral request controller can issue DMA request as soon as possible.

Related Information

DMA Controller on page 512

16.4.7. Arbitration between Direct/Indirect Access Controller and STIG

When multiple controllers are active simultaneously, a fixed-priority arbitration scheme is used to arbitrate between each interface and access the external FLASH. The fixed priority is defined as follows, highest priority first.

1. The Indirect Access Write
2. The Direct Access Write
3. The STIG
4. The Direct Access Read
5. The Indirect Access Read

Each controller is back pressured while waiting to be serviced.

16.4.8. Configuring the Flash Device

For read and write accesses, software must initialize the device read instruction register (`devrd`) and the device write instruction register (`devwr`). These registers include fields to initialize the instruction opcodes that should be used as well as the instruction type, and whether the instruction uses single, dual or quad pins for address and data transfer. To ensure the quad SPI controller can operate from a reset state, the opcode registers reset to opcodes compatible with single I/O flash devices.

The quad SPI flash controller uses the instruction transfer width field (`instwidth`) of the `devrd` register to set the instruction transfer width for both reads and writes. There is no `instwidth` field in the `devwr` register. If instruction type is set to dual or quad mode, the address transfer width (`addrwidth`) and data transfer width (`datawidth`) fields of both registers are redundant because the address and data type is based on the instruction type. Thus, software can support the less common flash instructions where the opcode, address, and data are sent on two or four lanes. For most instructions, the opcodes are sent serially to the flash device, even for dual and quad instructions. One of the flash devices that supports instructions that can send the opcode over two or four lanes is the Micron N25Q128. For reference, Table 176 on page 501 and Table 177 on page 502 show how software should configure the quad SPI controller for each specific read and write instruction, respectively, supported by the Micron N25Q128 device.

Table 176. Quad SPI Configuration for Micron N25Q128 Device (Read Instructions)

Instruction	Lanes Used By Opcode	Lanes Used to Send Address	Lanes Used to Send Data	instwidth Value	addrwidth Value	datawidth Value
Read	1	1	1	0	0	0
Fast read	1	1	1	0	0	0
Dual output fast read (DOFR)	1	1	2	0	0	1
Dual I/O fast read (DIOFR)	1	2	2	0	1	1
Quad output fast read (QOFR)	1	1	4	0	0	2

continued...

Instruction	Lanes Used By Opcode	Lanes Used to Send Address	Lanes Used to Send Data	instwidth Value	addrwidth Value	datawidth Value
Quad I/O fast read (QIOFR)	1	4	4	0	2	2
Dual command fast read (DCFR)	2	2	2	1	Don't care	Don't care
Quad command fast read (QCFR)	4	4	4	2	Don't care	Don't care

Table 177. Quad SPI Configuration for Micron N25Q128 Device (Write Instructions)

Instruction	Lanes Used By Opcode	Lanes Used to Send Address	Lanes Used to Send Data	instwidth Value	addrwidth Value	datawidth Value
Page program	1	1	1	0	0	0
Dual input fast program (DIFP)	1	1	2	0	0	1
Dual input extended fast program (DIEFP)	1	2	2	0	1	1
Quad input fast program (QIFP)	1	1	4	0	0	2
Quad input extended fast program (QIEFP)	1	4	4	0	2	2
Dual command fast program (DCFP)	2	2	2	1	Don't care	Don't care
Quad command fast program (QCFP)	4	4	4	2	Don't care	Don't care

16.4.8.1. Write Request

The `Write Enable Latch (WEL)` bit within the flash device itself must be high before a write sequence can be issued. The command generator automatically issues the `Write Enable (WREN)` instruction to set the `WEL` bit before triggering a write command through the direct or indirect access controllers. When write requests are no longer received and all outstanding requests have been sent, then the flash device starts the page program write cycle. Any incoming request at this time is held in wait states until the cycle has completed. The controller automatically polls the flash device `Read Status Register (RDSR)` to identify when the write cycle has completed. This continues until the `Write In Progress` bit and `WEL` bit have cleared to zero. The op-code for the WREN instruction is typically 0x06h and 0x05h for the RDSR instruction.

Note: These op-codes are common between devices.

16.4.9. XIP Mode

The quad SPI controller supports XIP mode, if the flash devices support XIP mode. Depending on the flash device, XIP mode puts the flash device in read-only mode, reducing command overhead.

The quad SPI controller must instruct the flash device to enter XIP mode by sending the mode bits. When the enter XIP mode on next read bit (`enterxipnextrd`) of the `cfg` register is set to 1, the quad SPI controller and the flash device are ready to enter XIP mode on the next read instruction. When the enter XIP mode immediately bit (`enterxipimm`) of the `cfg` register is set to 1, the quad SPI controller and flash device enter XIP mode immediately.

When the `enterxipnextrd` or `enterxipimm` bit of the `cfg` register is set to 0, the quad SPI controller and flash device exit XIP mode on the next read instruction. For more information, refer to the *"XIP Mode Operations"* section.

Related Information

XIP Mode Operations on page 509

16.4.10. Write Protection

You can program the controller to write protect a specific region of the flash device. The protected region is defined as a set of blocks, specified by a starting and ending block. Writing to an area of protected flash region memory generates an error and triggers an interrupt.

You define the block size by specifying the number of bytes per block through the number of bytes per block field (`bytespersubsector`) of the device size register (`devsz`). The lower write protection register (`lowwrprot`) specifies the first flash block in the protected region. The upper write protection register (`uppwrprot`) specifies the last flash block in the protected region.

The write protection enable bit (`en`) of the write protection register (`wrprot`) enables and disables write protection. The write protection inversion bit (`inv`) of the `wrprot` register flips the definition of protection so that the region specified by `lowwrprt` and `uppwrprt` is unprotected and all flash memory outside that region is protected.

16.4.11. Data Slave Sequential Access Detection

The quad SPI flash controller detects sequential accesses to the data slave interface by comparing the current access with the previous access. An access is sequential when it meets the following conditions:

- The address of the current access sequentially follows the address of the previous access.
- The direction of the current access (read or write) is the same as previous access.
- The size of the current access (byte, half-word, or word) is the same as previous access.

When the access is detected as nonsequential, the sequential access to the flash device is terminated and a new sequential access begins. Intel recommends accessing the data slave sequentially. Sequential access has less command overhead, and therefore, increases data throughput.

16.4.12. Clocks

There are two clock inputs to the quad SPI controller: `l4_mp_clk` and `l4_main_clk`; and one clock output: `qspi_clk`. The quad SPI flash controller uses the `l4_mp_clk` clock to clock the data slave transfers and register slave accesses. The `l4_main_clk` clock is the reference clock for the quad SPI controller and is used to serialize the data and drive the external SPI interface. The `qspi_clk` clock is the generated clock source for the connected flash devices.

The following is true for the following reference clocks:

- `l4_main_clk` should be greater than `l4_mp_clk`
- `l4_main_clk` must be greater than two times `qspi_clk`

The `qspi_clk` clock is derived by dividing down the reference clock, `l4_main_clk` clock by the baud rate divisor field (`bauddiv`) of the `cfg` register.

Related Information

Clock Manager on page 52

16.4.12.1. Clock Gating

You can clock gate `qspi_ref_clk` through software. By programming the `qspiclken` bit of the `en` register in the `perpllgrp` you can enable or disable the `qspi_ref_clk` to the QSPI Controller.

Related Information

Clock Manager Address Map and Register Definitions on page 67

16.4.13. Resets

A single reset signal (`qspi_flash_rst_n`) is provided as an input to the quad SPI controller. The reset manager drives the reset signal to the ECC controller on a cold or warm reset. This resets the logic inside the ECC controller.

Related Information

Reset Manager on page 134

16.4.13.1. Taking the Quad SPI Flash Controller Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

16.4.13.2. SRAM Initialization Procedure with ECC Enabled

1. Set 0x40 (default) to `srampart` register (SRAM_PARTITION_REG).
2. Enable ECC.
3. Fill up the read partition in indirect read transfer mode. Confirm `sramfill.Indrdpart` register is 0x41.
4. Fill up and initialize the write partition in indirect operation. Confirm `sramfill.Indwrpart` register is 0x40.
5. Clear ECC interrupt status register.
6. Enable ECC interrupt.

Note: SRAM Partition Configuration Register defines the size of the indirect read partition in the SRAM. By default, half of the SRAM is reserved for indirect read operation, and half for indirect write.

The number of locations allocated to indirect read = `SRAM_PARTITION_REG + 1`.

The number of locations allocated to indirect write = `128 - SRAM_PARTITION_REG`.

16.4.14. Interrupts

All interrupt sources are combined to create a single level-sensitive, active-high interrupt (`qspi_intr`). Software can determine the source of the interrupt by reading the interrupt status register (`irqstat`). By default, the interrupt source is cleared when software writes a one (1) to the interrupt status register. The interrupts are individually maskable through the interrupt mask register (`irqmask`). Table 178 on page 505 lists the interrupt sources in the `irqstat` register.

Table 178. Interrupt Sources in the irqstat Register

Interrupt Source	Description
Underflow detected	When 0, no underflow has been detected. When 1, the data slave write data is being supplied too slowly. This situation can occur when data slave write data is being supplied too slowly to keep up with the requested write operation This bit is reset only by a system reset and cleared only when a 1 is written to it.
Indirect operation complete	The controller has completed a triggered indirect operation.
Indirect read reject	An indirect operation was requested but could not be accepted because two indirect operations are already in the queue.
Protected area write attempt	A write to a protected area was attempted and rejected.
Illegal data slave access detected	An illegal data slave access has been detected. Data slave wrapping bursts and the use of split and retry accesses can cause this interrupt. It is usually an indicator that soft masters in the FPGA fabric are attempting to access the HPS in an unsupported way.
Transfer watermark reached	The indirect transfer watermark level has been reached.
Receive overflow	This condition occurs only in legacy SPI mode. When 0, no overflow has been detected. When 1, an over flow to the RX FIFO buffer has occurred. This bit is reset only by a system reset and cleared to zero only when this register is written to. If a new write to the RX FIFO buffer occurs at the same time as a register is read, this flag remains set to 1.

continued...

Interrupt Source	Description
TX FIFO not full	This condition occurs only in legacy SPI mode. When 0, the TX FIFO buffer is full. When 1, the TX FIFO buffer is not full.
TX FIFO full	This condition occurs only in legacy SPI mode. When 0, the TX FIFO buffer is not full. When 1, the TX FIFO buffer is full.
RX FIFO not empty	This condition occurs only in legacy SPI mode. When 0, the RX FIFO buffer is empty. When 1, the RX FIFO buffer is not empty.
RX FIFO full	This condition occurs only in legacy SPI mode. When 0, the RX FIFO buffer is not full. When 1, the RX FIFO buffer is full.
Indirect read partition overflow	Indirect Read Partition of SRAM is full and unable to immediately complete indirect operation

16.5. Quad SPI Flash Controller Programming Model

16.5.1. Setting Up the Quad SPI Flash Controller

The following steps describe how to set up the quad SPI controller:

1. Wait until any pending operation has completed.
2. Disable the quad SPI controller with the quad SPI enable field (`en`) of the `cfg` register.
3. Update the `instwidth` field of the `devrd` register with the instruction type you wish to use for indirect and direct writes and reads.
4. If mode bit enable bit (`enmodebits`) of the `devrd` register is enabled, update the mode bit register (`modebit`).
5. Update the `devsz` register as needed.
   Parts or all of this register might have been updated after initialization. The number of address bytes is a key configuration setting required for performing reads and writes. The number of bytes per page is required for performing any write. The number of bytes per device block is only required if the write protect feature is used.
6. Update the device delay register (`delay`).
   This register allows the user to adjust how the chip select is driven after each flash access. Each device may have different timing requirements. If the serial clock frequency is increased, these timing requirements become more critical. The numbers specified in this register are based on the period of the `qspi_ref_clk` clock. For example, some devices need 50 ns minimum time before the slave select can be reasserted after it has been deasserted. When the device is operating at 100 MHz, the clock period is 10 ns, so 40 ns extra is required. If the `qspi_ref_clk` clock is running at 400 MHz (2.5 ns period), specify a value of at least 16 to the clock delay for chip select deassert field (`nss`) of the `delay` register.
7. Update the `remapaddr` register as needed.
   This register only affects direct access mode.
8. Set up and enable the write protection registers (`wrprot`, `lowwrprot`, and `uppwrprot`) when write protection is required.
9. Enable required interrupts through the `irqmask` register.

10. Set up the `bauddiv` field of the `cfg` register to define the required clock frequency of the target device.
11. Update the read data capture register (`rddatacap`) if you need to change the auto-filled value.

    This register delays when the read data is captured and can help when the read data path from the device to the quad SPI controller is long and the device clock frequency is high.
12. Enable the quad SPI controller with the `en` field of the `cfg` register.

Related Information

Intel Arria 10 Device Datasheet

16.5.2. Indirect Read Operation with DMA Disabled

The following steps describe the general software flow to set up the quad SPI controller for indirect read operation with the DMA disabled:

1. Perform the steps described in the Setting Up the Quad SPI Flash Controller on page 506 section.
2. Set the flash memory start address in the `indrdstaddr` register.
3. Set the number of bytes to be transferred in the `indrdcnt` register.
4. Set the indirect transfer trigger address in the `indaddrtrig` register.
5. Set up the required interrupts through the `irqmask` register.
6. If the watermark level is used, set the SRAM watermark level through the `indrdwater` register.
7. Start the indirect read operation by setting the `start` field of the `indrd` register to 1.
8. Either use the watermark level interrupt or poll the SRAM fill level in the `sramfill` register to determine when there is sufficient data in the SRAM.
9. Issue a read transaction to the indirect address to access the SRAM. Repeat 8 if more read transactions are needed to complete the indirect read transfer.
10. Either use the indirect complete interrupt to determine when the indirect read operation has completed or poll the completion status of the indirect read operation through the indirect completion status bit (`ind_ops_done_status`) of the `indrd` register.

Related Information

Setting Up the Quad SPI Flash Controller on page 506

16.5.3. Indirect Read Operation with DMA Enabled

The following steps describe the general software flow to set up the quad SPI controller for indirect read operation with the DMA enabled:

1. Perform the steps described in the Setting Up the Quad SPI Flash Controller on page 506 section.
2. Set the flash memory start address in the `indrdstaddr` register.
3. Set the number of bytes to be transferred in the `indrdcnt` register.
4. Set the indirect transfer trigger address in the `indaddrtrig` register.
5. Set the number of bytes for single and burst type DMA transfers in the `dmaper` register.
6. Optionally set the SRAM watermark level in the `indrdwater` register to control the rate DMA requests are issued.
7. Start an indirect read access by setting the `start` field of the `indrd` register to 1.
8. Either use the indirect complete interrupt to determine when the indirect read operation has completed or poll the completion status of the indirect read operation through the `ind_ops_done_status` field of the `indrd` register.

Related Information

Setting Up the Quad SPI Flash Controller on page 506

16.5.4. Indirect Write Operation with DMA Disabled

The following steps describe the general software flow to set up the quad SPI controller for indirect write operation with the DMA disabled:

1. Perform the steps described in the Setting Up the Quad SPI Flash Controller on page 506 section.
2. Set the flash memory start address in the `indwrstaddr` register.
3. Set up the number of bytes to be transferred in the `indwrcnt` register.
4. Set the indirect transfer trigger address in the `indaddrtrig` register.
5. Set up the required interrupts through the interrupt mask register (`irqmask`).
6. Start the indirect write operation by setting the `start` field of the `indwr` register to 1.
7. Either use the watermark level interrupt or poll the SRAM fill level in the `sramfill` register to determine when there is sufficient space in the SRAM.
8. Issue a write transaction to the indirect address to write one flash page of data to the SRAM. Repeat 8 if more write transactions are needed to complete the indirect write transfer. The final write may be less than one page of data.

Related Information

- Indirect Write Operation on page 496
- Setting Up the Quad SPI Flash Controller on page 506

16.5.5. Indirect Write Operation with DMA Enabled

The following steps describe the general software flow to set up the quad SPI controller for indirect write operation with the DMA enabled:

1. Perform the steps described in the Setting Up the Quad SPI Flash Controller on page 506 section.
2. Set the flash memory start address in the `indwrstaddr` register.
3. Set the number of bytes to be transferred in the `indcnt` field of the `indwr` register.
4. Set the indirect transfer trigger address in the `indaddrtrig` register.
5. Set the number of bytes for single and burst type DMA transfers in the `dmaper` register.
6. Optionally set the SRAM watermark level in the `indwrwater` register to control the rate DMA requests are issued. The value set must be greater than one flash page. For more information, refer to the Indirect Write Operation on page 496 section.
7. Start the indirect write access by setting the `start` field of the `indirwr` register to 1.
8. Either use the indirect complete interrupt to determine when the indirect write operation has completed or poll the completion status of the indirect write operation through the `ind_ops_done_status` field of the `indwr` register.

Related Information

- Indirect Write Operation on page 496
- Setting Up the Quad SPI Flash Controller on page 506

16.5.6. XIP Mode Operations

XIP mode is supported in most SPI flash devices. However, flash device manufacturers do not use a consistent standard approach. Most use signature bits that are sent to the device immediately following the address bytes. Some devices use signature bits and also require a flash device configuration register write to enable XIP mode.

16.5.6.1. Entering XIP Mode

16.5.6.1.1. Micron Quad SPI Flash Devices with Support for Basic-XIP

To enter XIP mode in a Micron quad SPI flash device with support for Basic-XIP, perform the following steps:

1. Save the values in the mode bits, if you intend to restore them upon exit.
2. Disable the direct access controller and indirect access controller to ensure no new read or write accesses are sent to the flash device.
3. Set the XIP mode bits in the `modebit` register to 0x80.
4. Enable the quad SPI controller's XIP mode by setting the `enterxipnextrd` bit of the `cfg` register to 1.
5. Re-enable the direct access controller and, if required, the indirect access controller.

16.5.6.1.2. Micron Quad SPI Flash Devices without Support for Basic-XIP

To enter XIP mode in a Micron quad SPI flash device without support for Basic-XIP, perform the following steps:

1. Save the values in the mode bits, if you intend to restore them upon exit.
2. Disable the direct access controller and indirect access controller to ensure no new read or write accesses gets sent to the flash device.
3. Ensure XIP mode is enabled in the flash device by setting the volatile configuration register (VCR) bit 3 to 1. Use the `flashcmd` register to issue the VCR write command.
4. Set the XIP mode bits in the `modebit` register to 0x00.
5. Enable the quad SPI controller's XIP mode by setting the `enterxipnextrd` bit of the `cfg` register to 1.
6. Re-enable the direct access controller and, if required, the indirect access controller.

16.5.6.1.3. Winbond Quad SPI Flash Devices

To enter XIP mode in a Winbond quad SPI flash device, perform the following steps:

1. Save the values in the mode bits, if you intend to restore them upon exit.
2. Disable the direct access controller and indirect access controller to ensure no new read or write accesses are sent to the flash device.
3. Set the XIP mode bits in the `modebit` register to 0x20.
4. Enable the quad SPI controller's XIP mode by setting the `enterxipnextrd` bit of the `cfg` register to 1.
5. Re-enable the direct access controller and, if required, the indirect access controller.

16.5.6.1.4. Spansion Quad SPI Flash Devices

To enter XIP mode a Spansion quad SPI flash device, perform the following steps:

1. Save the values in the mode bits, if you intend to restore them upon exit.
2. Disable the direct access controller and indirect access controller to ensure no new read or write accesses are sent to the flash device.
3. Set the XIP mode bits in the `modebit` register to 0xA0.
4. Enable the quad SPI controller's XIP mode by setting the `enterxipnextrd` bit of the `cfg` register to 1.
5. Re-enable the direct access controller and, if required, the indirect access controller.

16.5.6.2. Exiting XIP Mode

To exit XIP mode, perform the following steps:

1. Disable the direct access controller and indirect access controller to ensure no new read or write accesses are sent to the flash device.
2. Restore the mode bits to the values before entering XIP mode, depending on the flash device and manufacturer.
3. Set the `enterxipnextrd` bit of the `cfg` register to 0.

The flash device must receive a read instruction before it can disable its internal XIP mode state. Thus, XIP mode internally stays active until the next read instruction is serviced. Ensure that XIP mode is disabled before the end of any read sequence.

16.5.6.3. XIP Mode at Power on Reset

Some flash devices can be XIP-enabled as a nonvolatile configuration setting, allowing the flash device to enter XIP mode at power-on reset (POR) without software intervention. Software cannot discover the XIP state at POR through flash status register reads because an XIP-enabled flash device can only be accessed through the XIP read operation. If you know the device may enter XIP mode at POR, have your initial boot software configure the `modebit` register and set the `enterxipimm` bit of the `cfg` register to 1.

If you do not know in advance whether or not the device may enter XIP mode at POR, have your initial boot software issue an XIP mode exit command through the `flashcmd` register, then follow the steps in the *"Entering XIP Mode"* section. Software must be aware of the mode bit requirements of the device, because XIP mode entry and exit varies by device.

Related Information

Entering XIP Mode on page 509

16.6. Quad SPI Flash Controller Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

Related Information

- Error Checking and Correction Controller on page 345
  For more information about how to program the ECC registers, refer to this chapter.
- Arria 10 Address Map and Register Definitions

17. DMA Controller

This chapter describes the direct memory access controller (DMAC) contained in the hard processor system (HPS). The DMAC transfers data between memory and peripherals and other memory locations in the system. The DMA controller is an instance of the Arm CoreLink* DMA Controller (DMA-330).

- Microcoded to support flexible transfer types
- Supports up to eight channels
- Provides 8-, 16-, 32-, and 64-bit transfer support
- Supports flow control with 32 peripheral request interfaces

Related Information

- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter
- Arm Information Center
  For more information about Arm's DMA-330 controller, refer to the *CoreLink DMA Controller DMA-330 Revision: r1p2* Technical Reference Manual on the Arm Infocenter website.

17.1. Features of the DMA Controller

The HPS provides one DMAC to handle the data transfer between memory-mapped peripherals and memories, off-loading this work from the MPU subsystem.

- The DMAC supports multiple transfer types:
  - Memory-to-memory
  - Memory-to-peripheral
  - Peripheral-to-memory
  - Scatter-gather
- Supports up to eight DMA channels
- Supports up to eight outstanding AXI read and eight outstanding AXI write transactions

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Supports scheduling up to 16 outstanding read and 16 outstanding write instructions
- Supports nine interrupt lines into the MPU subsystem:
  - One for DMA thread abort
  - Eight for events
- Supports 32 peripheral request interfaces:
  - Eight for FPGA
    - FPGA 5 is multiplexed with Security Manager TX
    - FPGA 6 is multiplexed with I^2C_EMAC2_TX
    - FPGA 7 is multiplexed with I^2C_EMAC2_RX
  - Five for I^2C
  - Three for I^2C (EMAC)
  - Eight for SPI
  - Two for quad SPI
  - One for System Trace Macrocell
  - Four for UART
  - One for FPGA manager

The DMA controller provides:

- An instruction processing block that enables it to process program code that controls a DMA transfer
- An Arm Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) master interface unit to fetch the program code from system memory into its instruction cache

  Note: The AXI master interface is used to perform DMA data transfer as well. The DMA instruction execution engine executes the program code from its instruction cache and schedules read or write AXI instructions through the respective instruction queues.
- A multi-FIFO (MFIFO) data buffer that it uses to store data that it reads, or writes, during a DMA transfer
- Nine interrupt outputs to enable efficient communication of events to the MPU interrupt controller

  Note: The peripheral request interfaces support the connection of DMA-capable peripherals to enable memory-to-peripheral and peripheral-to-memory transfers to occur, without intervention from the processor. Since the HPS supports some peripherals that do not comply with Arm DMA peripheral interface protocol, adapters are added to allow these peripherals to work with the DMAC.

The following peripheral interface protocols are supported:

- Synopsys protocol
  - FPGA manager
  - Serial peripheral interface (SPI)
  - Universal asynchronous receiver transmitter (UART)
  - Inter-integrated circuit (I^2C)
  - FPGA
- Arm protocol
  - Quad SPI flash controller
  - System trace macrocell (STM)

Dual slave interfaces enable the operation of the DMA controller to be partitioned into a secure and non-secure state. The network interconnect must be configured to ensure that only secure transactions can access the secure interface. The slave interfaces can access status registers and also be used to directly issue and execute instructions in the DMA controller.

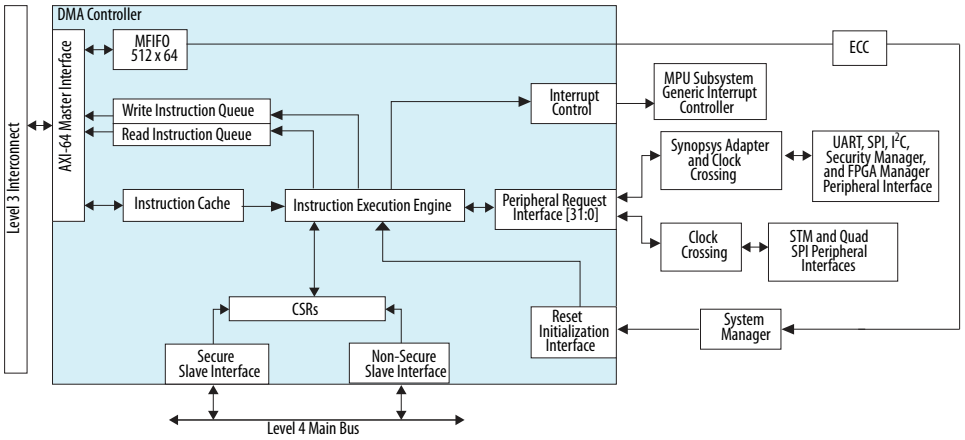
The DMAC has the following features:

- A small instruction set that provides a flexible method of specifying the DMA operations. This architecture provides greater flexibility than the fixed capabilities of a Linked-List Item (LLI) based DMA controller

17.2. DMA Controller Block Diagram and System Integration

The following figure shows a block diagram of the DMAC and how it integrates into the rest of the HPS system.

Figure 84. DMA Controller Connectivity



The `l4_main_clk` clock drives the DMA controller, controller logic, and all the interfaces. The DMA controller accesses the level 3 (L3) main switch with its 64-bit AXI master interface.

The DMA controller provides the following slave interfaces that connect to the L4 bus:

- Non-secure slave interface
- Secure slave interface
- ECC register slave interface

Both non-secure and secure slave interfaces may access registers that control the functionality of the DMA controller. The DMA controller implements TrustZone secure technology with one interface operating in the secure state and the other operating in the non-secure state.

The MFIFO has an ECC controller built-in to provide ECC protection. The ECC controller is able to detect single-bit and double-bit errors, and correct the single-bit errors. The ECC operation and functionality is programmable via the ECC register slave interface, as shown in Figure 84 on page 514. The ECC register interface provides host access to configure the ECC logic as well as inject bit errors into the memory. It also provides host access to memory initialization hardware used to clear out the memory contents including the ECC bits. The ECC controller generates interrupts upon occurrences of single and double-bit errors, and the interrupt signals are connected to the system manager.

Related Information

Error Checking and Correction Controller on page 345

17.3. Functional Description of the DMA Controller

This section describes the major interfaces and components of the DMAC and its operation.

The DMAC contains an instruction processing block that processes program code to control a DMA transfer. The program code is stored in a region of system memory that the DMAC accesses using its AXI master interface. The DMAC stores instructions temporarily in an internal cache.

The DMAC has eight DMA channels. Each channel supports a single concurrent thread of DMA operation. In addition, a single DMA manager thread exists, and you can use it to initialize the DMA channel threads.

The DMAC executes one instruction per clock cycle. To ensure that it regularly executes each active thread, the DMAC alternates by processing the DMA manager thread and then a DMA channel thread. It performs a round-robin process when selecting the next active DMA channel thread to execute.

The DMAC uses variable-length instructions that consist of one to six bytes. It provides a separate program counter (PC) register for each DMA channel.

The DMAC includes a 16-line instruction cache to improve the instruction fetch performance. Each instruction cache line contains eight, four-byte words for a total cache line size of 32 bytes. The DMAC instruction cache size is therefore 16 lines times 32 bytes per line which equals 512 bytes. When a thread requests an instruction from an address, the cache performs a lookup. If a cache hit occurs, then the cache immediately provides the instruction. Otherwise, the thread is stalled while the DMAC performs a cache line fill through the AXI master interface. If an instruction spans the end of a cache line, the DMAC performs multiple cache accesses to fetch the instruction.

Note: When a cache line fill is in progress, the DMAC enables other threads to access the cache. But if another cache miss occurs, the pipeline stalls until the first line fill is complete.

When a DMA channel thread executes a load or store instruction, the DMAC adds the instruction to the relevant read or write queue. The DMAC uses these queues as an instruction storage buffer prior to it issuing the instructions on the AXI. The DMAC also contains an MFIFO data buffer in which it stores data that it reads or writes during a DMA transfer.

The DMAC provides multiple interrupt outputs to enable efficient communication of events to the system CPUs. The peripheral request interfaces support the connection of DMA-capable peripherals to enable memory-to-peripheral and peripheral-to-memory DMA transfers to occur without intervention from the microprocessor.

Dual slave interfaces enable the operation of the DMAC to be partitioned into the secure state and non-secure states. You can access status registers and also directly execute instructions in the DMAC with the slave interfaces.

17.3.1. Peripheral Request Interface

You can enable each direct memory access (DMA) controller peripheral request interface, individually. Setting each of the eight peripheral request IDs enables or disables a peripheral request interface. When set to enable, it allows for FPGA soft logic to request a DMA transfer. For DMA transfers to or from the FPGA, this feature is only necessary if your design requires transfer flow control.

The DMA peripheral request interface communicates with peripherals by either the Arm protocol or the Synopsys protocol.

- Arm Protocol -

  For peripherals using the Arm protocol, clock-crossing logic is the only logic between the DMA and the peripheral.

- Synopsys Protocol Overview -

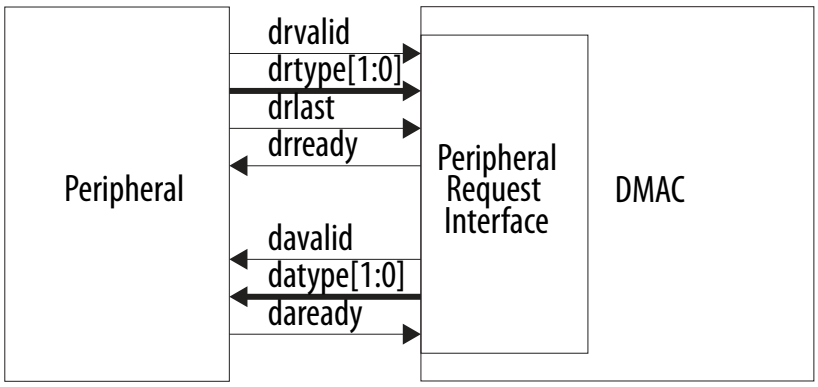
  The Synopsys protocol is used for the hardware flow control that is not the same as the protocol used by the DMA-330 core.

  For peripherals using the Synopsys protocol, clock-crossing logic and protocol adaptation logic exist between the DMA and the peripheral.

The following figure shows that the peripheral request interface consists of a peripheral request bus and a DMAC acknowledge bus that use the prefixes:

- `dr`—The peripheral request bus
- `da`—The DMAC acknowledge bus

Figure 85. Request and Acknowledge Buses on the Peripheral Request Interface



Both buses use the valid and ready handshake that the AXI protocol describes.

The peripheral uses `drtype[1:0]` to either:

- Request a single transfer
- Request a burst transfer
- Acknowledge a flush request

The peripheral uses `drlast` to notify the DMAC that the request on `drtype[1:0]` is the last request of the DMA transfer sequence. `drlast` is transferred at the same time as `drtype[1:0]`.

The DMAC can indicate the following using `datype[1:0]`:

- When it completes the requested single transfer
- When it completes the requested burst transfer
- When it issues a flush request

Note: If you configure the DMAC to provide more than one peripheral request interface, each interface is assigned a unique identifier, _<x> where <x> represents the number of the interface.

For the Synopsys protocol, the following signals are used in the handshaking protocol:

- `dma_tx_req_n`
- `dma_rx_req_n`
- `dma_tx_ack_n`
- `dma_rx_ack_n`
- `dma_tx_single_n`
- `dma_rx_single_n`

17.3.1.1. Peripheral Request Interface Mapping

You can assign a peripheral request interface to any of the DMA channels. When a DMA channel thread executes DMAWFP, the value programmed in the peripheral [4:0] field specifies the peripheral associated with that DMA channel.

The DMAC supports 32 peripheral request handshakes. Each request handshake can receive up to four outstanding requests, and is assigned a specific peripheral device ID. The following table lists the peripheral device ID assignments.

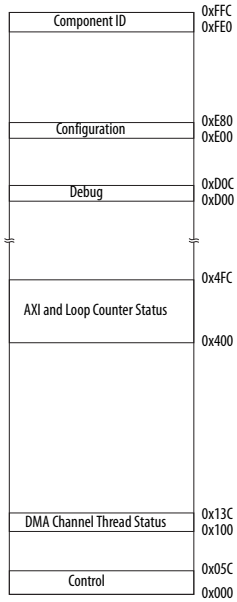
Table 179. Peripheral Request Interface Mapping

Peripheral	Request Interface ID	Protocol
FPGA 0	0	Synopsys
FPGA 1	1	Synopsys
FPGA 2	2	Synopsys
FPGA 3	3	Synopsys
FPGA 4	4	Synopsys
FPGA 5/Security Manager DMA Tx	5	Synopsys
FPGA 6/I^2C EMAC2 Tx	6	Synopsys
FPGA 7/I^2C EMAC2 Rx	7	Synopsys
I^2C0 Tx	8	Synopsys
I^2C0 Rx	9	Synopsys
I^2C1 Tx	10	Synopsys
I^2C1 Rx	11	Synopsys
I^2C EMAC0 Tx	12	Synopsys
I^2C EMAC0 Rx	13	Synopsys
I^2C EMAC1 Tx	14	Synopsys
I^2C EMAC1 Rx	15	Synopsys
SPI0 Master Tx	16	Synopsys
SPI0 Master Rx	17	Synopsys
SPI0 Slave Tx	18	Synopsys
SPI0 Slave Rx	19	Synopsys
SPI1 Master Tx	20	Synopsys
SPI1 Master Rx	21	Synopsys
SPI1 Slave Tx	22	Synopsys
SPI1 Slave Rx	23	Synopsys
Quad SPI Flash Tx	24	Arm
Quad SPI Flash Rx	25	Arm
STM	26	Arm
FPGA Manager DMA Rx	27	Synopsys
UART0 Tx	28	Synopsys
UART0 Rx	29	Synopsys
UART1 Tx	30	Synopsys
UART1 Rx	31	Synopsys

17.4. DMA Controller Address Map and Register Definitions

The following DMAC register map spans a 4 KB region, consists of the following sections.

Figure 86. DMAC Summary Register Map



- **Control registers—** allow you to control the DMAC.
- **DMA channel thread status registers—**provide the status of the DMA channel threads.
- **AXI and loop counter status registers—**provide the AXI transfer status and the loop counter status for each DMA channel thread.
- **Debug registers—**enable the following functionality:
  - — Allow you to send instructions to a thread when debugging the program code.
  - — Allow system firmware to send instructions to the DMA manager thread.
- **Configuration registers—**enable system firmware to identify the configuration of the DMAC and control the behavior of the watchdog.
- **Component ID registers—** enable system firmware to identify peripherals. Do not attempt to access reserved or unused address locations. Attempting to access these locations can result in an unpredictable behavior.

17.4.1. DMA Controller Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

Related Information

Error Checking and Correction Controller on page 345
For more information about how to program the ECC registers, refer to this chapter.

18. Ethernet Media Access Controller

The hard processor system (HPS) provides three Ethernet media access controller (EMAC) peripherals. Each EMAC can be used to transmit and receive data at 10/100/1000 Mbps over Ethernet connections in compliance with the IEEE 802.3 specification. The EMACs are instances of the Synopsys DesignWare Universal 10/100/1000 Ethernet MAC (version 3.72a).

The EMAC has an extensive memory-mapped control and status register (CSR) set, which can be accessed by the Arm Cortex-A9 MPCore.

For an understanding of this chapter, you should be familiar with the basics of IEEE 802.3 media access control (MAC). (47)

Related Information

- IEEE Standards Association
  For complete information about IEEE 802.3 MAC, refer to the *IEEE 802.3 2008 Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, available on the IEEE Standards Association website.
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

(47) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

18.1. Features of the Ethernet MAC

18.1.1. MAC

- IEEE 802.3-2008 compliant
- Data rates of 10/100/1000 Mbps
- Full duplex and half duplex modes
  - IEEE 802.3x flow control automatic transmission of zero-quanta pause frame on flow control input deassertion
  - Optional forwarding of received pause control frames to the user
  - Packet bursting and frame extension in 1000 Mbps half-duplex
  - IEEE 802.3x flow control in full-duplex
  - Back-pressure support for half-duplex
- 4 KB TX FIFO RAM and 16 KB RX FIFO RAM with ECC support
- IEEE 1588-2002 and IEEE 1588-2008 precision networked clock synchronization
- IEEE 802.3-az, version D2.0 for Energy Efficient Ethernet (EEE)
- IEEE 802.1Q Virtual Local Area Network (VLAN) tag detection for reception frames
- Preamble and start-of-frame data (SFD) insertion in transmit and deletion in receive paths
- Automatic cyclic redundancy check (CRC) and pad generation controllable on a per-frame basis
- Options for automatic pad/CRC stripping on receive frames
- Programmable frame length supporting standard and jumbo Ethernet frames (with sizes up to 3800 bytes)
- Programmable inter-frame gap (IFG), from 40- to 96-bit times in steps of eight bits
- Preamble lengths of one, three, five and seven bytes supported
- Supports internal loopback asynchronous FIFO on the GMII/MII for debugging
- Supports a variety of flexible address filtering modes
  - Up to 31 additional 48-bit perfect destination address (DA) filters with masks for each byte
  - Up to 31 48-bit source address (SA) comparison check with masks for each byte
  - 256-bit hash filter (optional) for multicast and unicast DAs
  - Option to pass all multicast addressed frames
  - Promiscuous mode support to pass all frames without any filtering for network monitoring
  - Passes all incoming packets (as per filter) with a status report

18.1.2. DMA

- 32-bit interface
- Programmable burst size for optimal bus utilization
- Single-channel mode transmit and receive engines
- Byte-aligned addressing mode for data buffer support
- Dual-buffer (ring) or linked-list (chained) descriptor chaining
- Descriptors can each transfer up to 8 KB of data
- Independent DMA arbitration for transmit and receive with fixed priority or round robin

18.1.3. Management Interface

- 32-bit host interface to CSR set
- Comprehensive status reporting for normal operation and transfers with errors
- Configurable interrupt options for different operational conditions
- Per-frame transmit/receive complete interrupt control
- Separate status returned for transmission and reception packets

18.1.4. Acceleration

- Transmit and receive checksum offload for transmission control protocol (TCP), user datagram protocol (UDP), or Internet control message protocol (ICMP) over Internet protocol (IP)

18.1.5. PHY Interface

Different external PHY interfaces are provided depending on whether the Ethernet Controller signals are routed through the HPS I/O pins or the FPGA I/O pins.

The PHY interfaces supported using the HPS I/O pins are:

- Reduced Media Independent Interface (RMII)
- Reduced Gigabit Media Independent Interface (RGMII)

The PHY interfaces supported using the FPGA I/O pins are:

- Media Independent Interface (MII)
- Gigabit Media Independent Interface (GMII)
- Reduced Media Independent Interface (RMII) with additional required adaptor logic

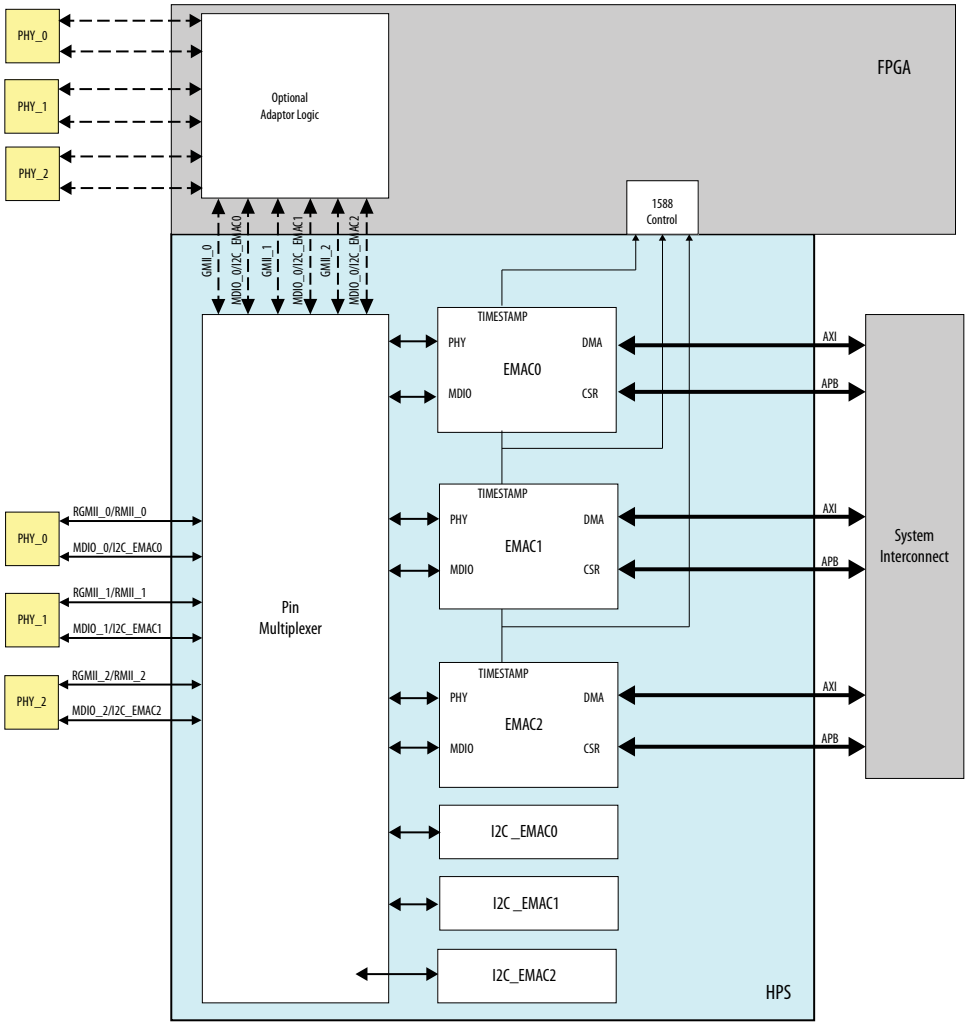
  Note: Additional adaptor logic for RMII not provided.
- Serial Gigabit Media Independent Interface (SGMII) supported through transceiver I/O or high-speed low-voltage differential signaling (LVDS) with soft clock data recover (CDR) I/O with additional required adaptor logic

The Ethernet Controller has two choices for the management control interface used for configuration and status monitoring of the PHY:

- Management Data Input/Output (MDIO)
- I^2C PHY management through a separate I^2C module within the HPS

18.2. EMAC Block Diagram and System Integration

Figure 87. EMAC System Integration



EMAC Overview

Each EMAC is an internal bus master that sends Ethernet packets to and from the System Interconnect. The EMAC uses a descriptor ring protocol, where the descriptor contains an address to a buffer to fetch or store the packet data.

Each EMAC has an MDIO Management port to send commands to the external PHY. Alternatively, you can use an I^2C module in the HPS for the management interface.

Each EMAC has an IEEE 1588 Timestamp interface with 10 ns resolution. The Arm Cortex-A9 microprocessor unit (MPU) subsystem can use it to maintain synchronization between the time counters that are internal to the three MACs. The

clock reference for the timestamp can be provided by the Clock Manager (`emac_ptp_clk`) or the FPGA fabric (`f2s_emac_ptp_ref_clk`). The clock reference is selected by the `ptp_clk_sel` bit in the `emac_global` register in the system manager.

Note: All three EMACs must use the same clock reference. In addition, EMAC0 can be configured to provide the timestamp for EMAC1, EMAC2, or both by setting the `ptp_ref_sel` bit in the `emac*` register in the System Manager.

18.3. EMAC Signal Description

The EMAC provides a variety of PHY interfaces and control options through the HPS and the FPGA I/Os.

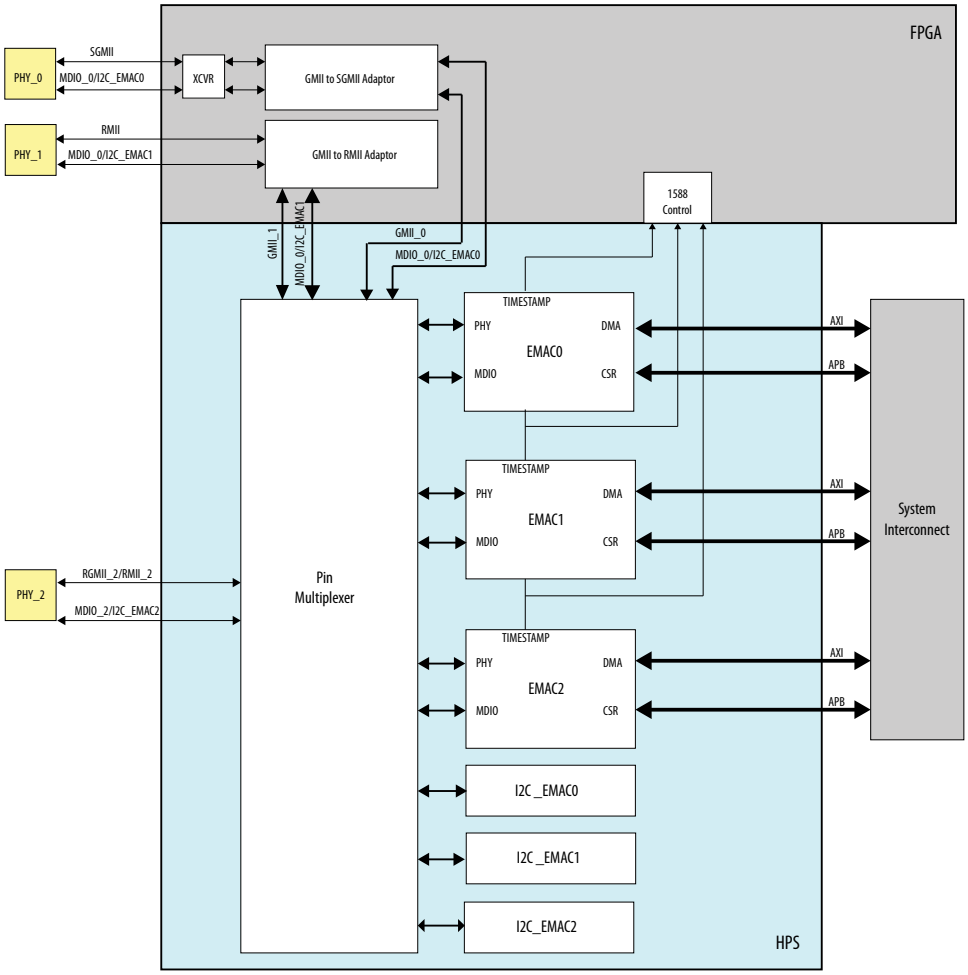
For designs in which the HPS is pin-limited, the EMAC signals can be routed through the FPGA as any one of the following interfaces by using soft adaptor logic in the FPGA:

- RMII
- MII/GMII
- SGMII

The figure below depicts a design which routes the EMAC0 and EMAC1 signals through the FPGA to provide an RMII and SGMII interface. EMAC2 is routed through the HPS.

Refer to the "EMAC FPGA Interface Initialization" section to find out more information about configuring EMAC interfaces through FPGA.

Figure 88. EMAC to FPGA Routing Example



Related Information

EMAC FPGA Interface Initialization on page 581
Information on how to initialize GMII/MII interface

18.3.1. HPS EMAC I/O Signals

There are three EMACs available in the HPS. The following table lists the EMAC signals that can be routed from the EMACs to the HPS I/O pins. These signals provide the RMII/RGMII interface. For more information on routing EMAC signals to the FPGA and the HPS I/O, refer to the *HPS Component Interfaces* chapter.

Note: The "n" in EMACn stands for the EMAC peripheral number.

Table 180. HPS EMAC I/O Signals

EMAC HPS I/O		In/Out	Width	Description
EMAC0_TX_CLKEMAC1_TX_CLKEMAC2_TX_CLK	Transmit Clock routed from one of three Platform Designer (Standard) port signals emac[2:0]_phy_txclk_o	Out	1	This signal provides the transmit clock for RGMII (125/25/2.5 MHz in 1G/100M/10Mbps). This signal is one option for the common transmit and receive clock in RMII mode (50 MHz in 100M or 10 Mbps mode). The other possible source for the common transmit and receive clock is an external clock source, in which case EMACn_TX_CLK is left unconnected. In RMII mode, if this signal is the clock source for the receiver, then connect EMACn_TX_CLK to EMACn_RX_CLK. All PHY transmit signals generated by the EMAC are synchronous to this clock.
EMAC0_TXD[3:0] EMAC1_TXD[3:0] EMAC2_TXD[3:0]	PHY Transmit Data, routed from one of three groups of Platform Designer (Standard) port signals emac[2:0]_phy_txd_o[3:0]	Out	4	This group of transmit data signals is driven by the MAC. Bits [3:0] provide the RGMII transmit data, and bits [1:0] provide the RMII transmit data. In RGMII mode, the data bus carries transmit data at double rate and are sampled on both the rising and falling edges of the transmit clock. The validity of the data is qualified with EMACn_TX_CTL.
EMAC0_TX_CTL EMAC1_TX_CTL EMAC2_TX_CTL	PHY Transmit Data Enable, routed from one of three Platform Designer (Standard) port signals emac[2:0]_phy_txen	Out	1	This signal is driven by the EMAC component. In RGMII mode, this signal acts as the control signal for the transmit data, and is driven on both edges of the transmit clock, EMACn_TX_CLK. In RMII mode, this signal is high to indicate valid data.
EMAC0_RX_CLK EMAC1_RX_CLK EMAC2_RX_CLK	Receive Clock, routed to one of three Platform Designer (Standard) port signals emac[2:0]_clk_rx_i	In	1	In RGMII mode, this clock frequency is 125/25/2.5 MHz in 1 G/100 M/10 Mbps modes. It is provided by the external PHY. All PHY signals received by the EMAC are synchronous to this clock. In RMII mode, this clock frequency is 50 MHz. The source of this clock can be: • An external source: In this case EMACn_TX_CLK should be left unconnected. • EMACn_TX_CLK: In this case, EMACn_TX_CLK must be connected to EMACn_RX_CLK.
EMAC0_RXD[3:0] EMAC1_RXD[3:0] EMAC2_RXD[3:0]	PHY Receive Data, routed to one of three groups of Platform Designer (Standard) port signals emac[2:0]_phy_rxd[3:0]	In	4	These data signals are received from the PHY. In RGMII mode, data is received at double data rate with bits[3:0] valid on the rising and falling edges of EMACn_RX_CLK. In RMII mode, data is received at single data rate with bits [1:0] valid on the rising edge of EMACn_RX_CLK. The validity of the data is qualified with EMACn_RX_CTL.
EMAC0_RX_CTL EMAC1_RX_CTL EMAC2_RX_CTL	PHY Receive Data Valid, routed to one of three groups of Platform	In	1	This signal is driven by the PHY and functions as the receive control signal used to qualify the data received on EMACn_RXD[3:0]. This signal is sampled on both edges of the clock in RGMII mode. See row above for clock to data relationships across the modes.

continued...

EMAC HPS I/O		In/Out	Width	Description
	Designer (Standard) port signals emac[2:0]_phy_rxdv.			

Related Information

EMAC HPS Interface Initialization on page 583
Information on how to initialize RGMII/RMII interface

18.3.2. FPGA EMAC I/O Signals

Three Ethernet Media Access Controllers are provided in the HPS. The table below describes the signals that are available from each Ethernet Media Access Controller to the FPGA I/O. For more information on routing EMAC signals to the FPGA and HPS I/O, refer to the *HPS Component Interfaces* chapter. See the HPS I/O table for general clock to data relationships across the modes.

Table 181. FPGA EMAC I/O Signals

Signal Name		In/Out	Width	Description
emac_clk_tx_i	Transmit Clock	In	1	This is the transmit clock (2.5 MHz/25 MHz) provided by the MII PHYs only. This clock comes from the FPGA Interface and is used for TX data capture. This clock is not used in GMII mode. *Note:* This clock must be able to perform glitch free switching between 2.5 and 25 MHz.
emac_phy_txclk_o	Transmit Clock Output	Out	1	In GMII mode, this signal is the transmit clock output to the PHY to sample data. For MII, this clock is unused.
emac_phy_txd_o[7:0]	PHY Transmit Data	Out	8	These are a group of eight transmit data signals driven by the EMAC. All eight bits provide the GMII transmit data byte. For the lower speed MII operation, only bits[3:0] are used. The validity of the data is qualified with phy_txen_o and phy_txer_o. Synchronous to phy_txclk_o.
emac_phy_txen_o	PHY Transmit Data Enable	Out	1	This signal is driven by the EMAC and is used in GMII mode. When driven high, this signal indicates that valid data is being transmitted on the clk_tx_o bus.
emac_phy_txer_o	PHY Transmit Error	Out	1	This signal is driven by the EMAC and when high, indicates a transmit error or carrier extension on the phy_txd bus. It is also used to signal low power states in Energy Efficient Ethernet operation.
emac_rst_clk_tx_n_o	Transmit Clock Reset output	Out	1	Transmit clock reset output to the FPGA fabric, which is the internal synchronized reset to clk_tx_int

continued...

Signal Name		In/Out	Width	Description
				output from the EMAC. May be used by logic implemented in the FPGA fabric as desired. The reset pulse width of the `rst_clk_tx_n_o` signal is three transmit clock cycles.
`emac_clk_rx_i`	Receive Clock	In	1	Receive clock from external PHY. For GMII, the clock frequency is 125 MHz. For MII, the receive clock is 25 MHz for 100 Mbps and 2.5 MHz for 10 Mbps.
`emac_phy_rxd_i[7:0]`	PHY Receive Data	In	8	This is an eight-bit receive data bus from the PHY. In GMII mode, all eight bits are sampled. The validity of the data is qualified with `phy_rxdv_i` and `phy_rxer_i`. For lower speed MII operation, only bits [3:0] are sampled. These signals are synchronous to `phy_clk_rx_i`.
`emac_phy_rxdv_i`	PHY Receive Data Valid	In	1	This signal is driven by PHY. In GMII mode, when driven high, it indicates that the data on the `phy_rxd` bus is valid. It remains asserted continuously from the first recovered byte of the frame through the final recovered byte.
`emac_phy_rxer_i`	PHY Receive Error	In	1	This signal indicates an error or carrier extension (GMII) in the received frame. This signal is synchronous to `phy_clk_rx_i`.
`emac_rst_clk_rx_n_o`	Receive clock reset output.	Out	1	Receive clock reset output. The reset pulse width of the `rst_clk_rx_n_o` signal is three transmit clock cycles.
`emac_phy_mac_speed_o[1:0]`	EMAC Speed	Out	2	This signal indicates the 10-Mbps, 100-Mbps or 1000-Mbps operating speed and reflects the `Port Select` and FES bit of the `MAC Configuration` register. The signal value encodings are: • 0x0: 1000 Mbps (GMII) • 0x1: 1000 Mbps (GMII) • 0x2: 10 Mbps (MII) • 0x3: 100 Mbps (MII)
`emac_phy_crs_i`	PHY Carrier Sense	In	1	This signal is asserted by the PHY when either the transmit or receive medium is not idle. The PHY de-asserts this signal when both transmit and receive interfaces are idle. This signal is not synchronous to any clock.
`emac_phy_col_i`	PHY Collision Detect	In	1	This signal, valid only when operating in half duplex, is asserted by the PHY when a collision is detected on the medium. This signal is not synchronous to any clock.

Related Information

EMAC FPGA Interface Initialization on page 581
Information on how to initialize GMII/MII interface

18.3.3. PHY Management Interface

The HPS can provide support for either MDIO or I^2C PHY management interfaces.

18.3.3.1. MDIO Interface

The MDIO interface signals are synchronous to `l4_mp_clk` in all supported modes.

Note: The MDIO interface signals can be routed to both the FPGA and HPS I/O.

Table 182. PHY MDIO Management Interface

Signal	HPS I/O Pin Name	In/Out	Width	Description
`emac_gmii_mdi_i`	`EMACn_MDIO`	In	1	Management Data In. The PHY generates this signal to transfer register data during a read operation. This signal is driven synchronously with the `gmii_mdc_o` clock.
`emac_gmii_mdo_o`		Out	1	Management Data Out. The EMAC uses this signal to transfer control and data information to the PHY.
`emac_gmii_mdo_o_e`		Out	1	Management Data Output Enable. This enable signal drives the `gmii_mdo_o` signal from an external three-state I/O buffer. This signal is asserted whenever valid data is driven on the `gmii_mdo_o` signal. The active state of this signal is high.
`emac_gmii_mdc_o`	`EMACn_MDC`	Out	1	Management Data Clock. The EMAC provides timing reference for the `gmii_mdi_i` and `gmii_mdo_o` signals on MII through this aperiodic clock. The maximum frequency of this clock is 2.5 MHz. This clock is generated from the application clock through a clock divider.

18.3.3.2. I^2C External PHY Management Interface

Some PHY devices use the I^2C instead of MDIO for their control interface. Small form factor pluggable (SFP) optical or pluggable modules are often among those with this interface.

The HPS or FPGA can use three of the five general purpose I^2C peripherals for controlling the PHY devices:

- i2c_emac_0 at base address 0xFFC02400
- i2c_emac_1 at base address 0xFFC02500
- i2c_emac_2 at base address 0xFFC02600

18.4. EMAC Internal Interfaces

18.4.1. DMA Master Interface

The DMA interface acts as a bus master on the system interconnect. Two types of data are transferred on the interface: data descriptors and actual data packets. The interface is very efficient in transferring full duplex Ethernet packet traffic. Read and write data transfers from different DMA channels can be performed simultaneously on this port, except for transmit descriptor reads and write-backs, which cannot happen simultaneously.

DMA transfers are split into a software configurable number of burst transactions on the interface. The `AXI_Bus_Mode` register in the `dmagrp` group is used to configure bursting behavior.

The interface assigns a unique ID for each DMA channel and also for each read DMA or write DMA request in a channel. Data transfers with distinct IDs can be reordered and interleaved.

The DMA interface can be configured to perform cacheable accesses. This configuration can be done in the System Manager when the DMA interface is inactive.

Write data transfers are generally performed as posted writes with OK responses returned as soon as the system interconnect has accepted the last beat of a data burst. Descriptors (status or timestamp), however, are always transferred as non-posted writes in order to prevent race conditions with the transfer complete interrupt logic.

The slave may issue an error response. When that happens, the EMAC disables the DMA channel that generated the original request and asserts an interrupt signal. The host must reset the EMAC with a hard or soft reset to restart the DMA to recover from this condition.

The EMAC supports up to 16 outstanding transactions on the interface. Buffering outstanding transactions smooths out back pressure behavior improving throughput when resource contention bottlenecks arise under high system load conditions.

Related Information

- DMA Controller on page 535
  Information regarding DMA Controller functionality
- System Manager on page 159

18.4.2. Timestamp Interface

The timestamp clock reference can come from either the Clock Manager or the FPGA fabric. If the FPGA has implemented the serial capturing of the timestamp interface, then the FPGA must provide the PTP clock reference.

Each EMAC provides its internal timestamp as an output. In some applications, it is advantageous to allow the FPGA fabric access to the Ethernet timestamp. In that case, the timestamp output from each EMAC is sampled in the `clk_ptp_ref_i` clock domain and serially shifted out to the FPGA fabric. The PTP timestamp clock must be selected to come from the FPGA fabric if the serial timestamp is used in the FPGA.

In addition to providing a timestamp clock reference, the FPGA can monitor the pulse-per-second output from each EMAC module and trigger a snapshot from each auxiliary time stamp timer.

The following table lists the EMAC to FPGA IEEE1588 Timestamp Interface signals to and from each EMAC module.

Table 183. EMAC to FPGA IEEE 1588 Timestamp Interface Signals

Signal Name		In/Out	Width	Description
f2s_emac_ptp_ref_clk	Timestamp PTP Clock reference from the FPGA	In	1	Used as PTP Clock reference for each EMAC when the FPGA has implemented Timestamp capture interface. The timestamp clock is common to all three EMACs. The frequency of this clock can be up to 100 MHz.
ptp_tstmp_en	Timestamp Serial Interface Enable	Out	1	When the local timestamp of each EMAC is sampled, the enable signal is pulsed with the first of 64 bits of serially shifted data. Synchronous to f2s_emac_ptp_ref_clk.
ptp_tstmp_data	Timestamp Serial Interface Data	Out	1	The 64-bit sampled timestamp is shifted serially to the FPGA fabric from the EMAC. The enable is asserted only on the first bit. The first bit transferred is the least significant bit of the sampled ptp_timestamp[63:0], or ptp_timestamp[0].
ptp_pps_o	Pulse Per Second Output	Out	1	This signal is asserted based on the PPS mode selected in the Register 459 (PPS Control Register). Otherwise, this pulse signal is asserted every time the seconds counter is incremented. This signal is synchronous to f2s_emac_ptp_ref_clk and may only be sampled if the FPGA clock is used as timestamp reference.
ptp_aux_ts_trig_i	Auxiliary Timestamp Trigger	In	1	This signal is asserted to take an auxiliary snapshot of the time. The rising edge of this internal signal is used to trigger the auxiliary snapshot. The signal is synchronized internally with clk_ptp_ref_i which results in an additional delay of 3 cycles. This input is asynchronous input and its assertion period must be greater than 2 PTP active clocks to be sampled.

Each EMAC supports either internal or external timestamp reference. In addition, EMAC0 has the option to be the master that provides the timestamp to EMAC1 and EMAC2. In this configuration, EMAC0 must be programmed to select internal timestamp generation in the System Manager and EMAC1 and EMAC2 must be programmed to select external timestamp generation.

Related Information

IEEE 1588-2002 Timestamps on page 560
More information regarding IEEE 1588-2002 timestamp functionality

18.4.3. System Manager Configuration Interface

The System Manager configures several static EMAC functions as shown in the following table. Software must configure these functions appropriately prior to using the EMAC module. Refer to the *Ethernet Programming Model* section for more details regarding pertinent System Manager registers.

Table 184. System Manager Control Settings

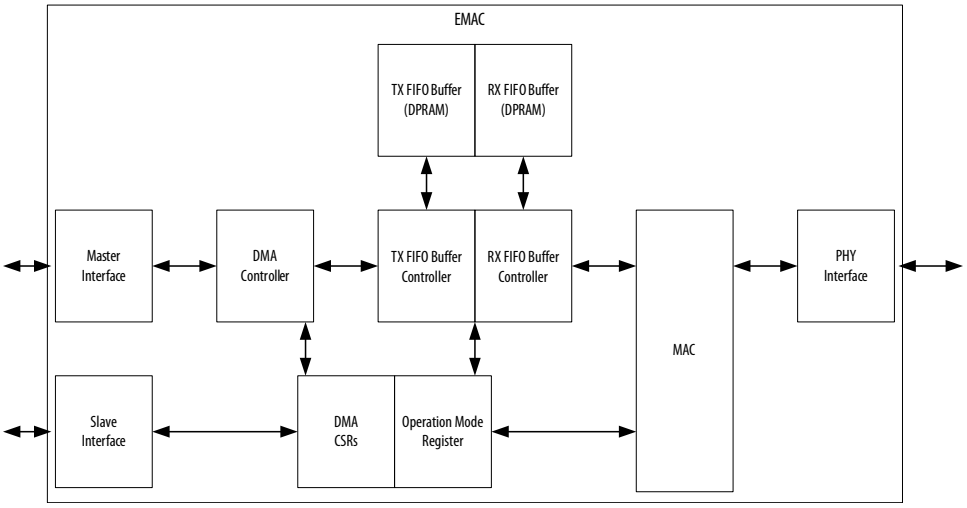
Function	Description
PHY Select	Select RESET, RGMII, RMII or GMII/MII as the PHY interface. The RESET mode is the default out of reset and configures the EMAC to use an internal clock rather than depending on a PHY to provide and active clock. The RESET mode cannot be used with any PHY, and another setting must be programmed before attempting to communicate with a PHY.
PTP Timestamp Reference Select	This field selects if the Timestamp reference is internally or externally generated. EMAC0 must be set to Internal Timestamp. EMAC0 may be the master to generate the timestamp for EMAC1 and EMAC2. EMAC1 and EMAC2 may be set to either Internal or External.
PTP Timestamp Clock Select	Selects the source of the PTP reference clock between `emac_ptp_clk` from the Clock Manager or `f2s_emac_ptp_ref_clk` from the FPGA Fabric. All three EMAC modules must use the same reference clock.
AXI Cache and Protection Settings	Static settings are provided to drive the ARCACHE, AWCACHE, ARPROT, and AWPROT signals for the AXI DMA bus.
FPGA Interface Enable	This field enables logic from the FPGA. This signal is only for safety to prevent spurious inputs from the FPGA before the FPGA is configured.

Related Information

- System Level EMAC Configuration Registers on page 580
  Information regarding system level registers involved in EMAC initialization
- System Manager on page 159

18.5. Functional Description of the EMAC

Figure 89. EMAC High-Level Block Diagram with Interfaces



There are two host interfaces to the Ethernet MAC. The management host interface, a 32-bit slave interface, provides access to the CSR set regardless of whether or not the EMACs are used directly through the FPGA fabric. The data interface is a 32-bit master interface, and it controls data transfer between the direct memory access (DMA) controller channels and the rest of the HPS system through the L3 interconnect.

The built-in DMA controller is optimized for data transfer between the MAC controller and system memory. The DMA controller has independent transmit and receive engines and a CSR set. The transmit engine transfers data from system memory to the device port, while the receive engine transfers data from the device port to the system memory. The controller uses descriptors to efficiently move data from source to destination with minimal host intervention.

The EMAC also contains FIFO buffer memory to buffer and regulate the Ethernet frames between the application system memory and the EMAC module. Each EMAC module has one 4 KB TX FIFO and one 16 KB RX FIFO. On transmit, the Ethernet frames write into the transmit FIFO buffer, and eventually trigger the EMAC to perform the transfer. Received Ethernet frames are stored in the receive FIFO buffer and the FIFO buffer fill level is communicated to the DMA controller. The DMA controller then initiates the configured burst transfers. Receive and transmit transfer statuses are read by the EMAC and transferred to the DMA.

18.5.1. Transmit and Receive Data FIFO Buffers

Each EMAC component has associated transmit and receive data FIFO buffers to regulate the frames between the application system memory and the EMAC. The RX FIFO buffer is a 16 KB dual-ported memory and the TX FIFO buffer is a 4 KB dual-ported memory. Both buffers are designed to support jumbo frames. A FIFO buffer word consists of:

- Data: 32 bits
- Sideband:
  - Byte enables: 2 bits
  - End of frame (EOF): 1 bit
  - Error correction code (ECC): 7 bits

The FIFO RAMs are each supported by an ECC controller that performs single-bit error detection and correction and double-bit error detection. The ECC Controllers have a dedicated hardware block for memory data initialization and can log error events and generate interrupts on single and double-error events. See the *Error Checking and Correction (ECC) Controller* for more information regarding the function of the ECC RAMs.

TX FIFO

The time at which data is sent from the TX FIFO to the EMAC is dependent on the transfer mode selected:

- Cut-through mode: Data is popped from the TX FIFO when the number of bytes in the TX FIFO crosses the configured threshold level (or when the end of the frame is written before the threshold is crossed). The threshold level is configured using the TTC bit of the Operation Mode Register (Register 6).

  Note: After more than 96 bytes (or 548 bytes in 1000 Mbps mode) are popped to the EMAC, the TX FIFO controller frees that space and makes it available to the DMA and a retry is not possible.

- Store-and-Forward mode: Data is popped from the TX FIFO when one or more of the following conditions are true:
  - A complete frame is stored in the FIFO
  - The TX FIFO becomes almost full

The application can flush the TX FIFO of all contents by setting bit 20 (FTF) of Register 6 (Operation Mode Register). This bit is self-clearing and initializes the FIFO pointers to the default state. If the FTF bit is set during a frame transfer to the EMAC, further transfers are stopped because the FIFO is considered empty. This cessation causes an underflow event and a runt frame to be transmitted and the corresponding status word is forwarded to the DMA.

If a collision occurs in half-duplex mode operation before an end of the frame, a retry attempt is sent before the end of the frame is transferred. When notified of the retransmission, the MAC pops the frame from the FIFO again.

Note: Only packets of 3800 bytes or less can be supported when the checksum offload feature is enabled by software.

RX FIFO

Frames received by the EMAC are pushed into the RX FIFO. The fill level of the RX FIFO is indicated to the DMA when it crosses the configured receive threshold which is programmed by the RTC field of Register 6 (Operation Mode Register). The time at which data is sent from the RX FIFO to the DMA is dependent on the configuration of the RX FIFO:

- Cut-through (default) mode: When 64 bytes or a full packet of data is received into the FIFO, data is popped out of the FIFO and sent to the DMA until a complete packet has been transferred. Upon completion of the end-of-frame transfer, the status word is popped and sent to the DMA.
- Store and forward mode: A frame is read out only after being written completely in the RX FIFO. This mode is configured by setting the RSF bit of Register 6 (Operation Mode Register).

If the RX FIFO is full before it receives the EOF data from the EMAC, an overflow is declared and the whole frame (including the status word) is dropped and the overflow counter in the DMA, (Register 8) Missed Frame and Buffer Overflow Counter Register, is incremented. This outcome is true even if the Forward Error Frame (FEF) bit of Register 6 (Operation Mode Register) is set. If the start address of such a frame has already been transferred, the rest of the frame is dropped and a dummy EOF is written to the FIFO along with the status word. The status indicates a partial frame because of overflow. In such frames, the Frame Length field is invalid. If the RX FIFO is configured to operate in the store-and-forward mode and if the length of the received frame is more than the FIFO size, overflow occurs and all such frames are dropped.

Note: In store-and-forward mode, only received frames with length 3800 bytes or less prevent overflow errors and frames from being dropped.

Related Information

Error Checking and Correction Controller on page 345

18.5.2. DMA Controller

The DMA has independent transmit and receive engines, and a CSR space. The transmit engine transfers data from system memory to the device port or MAC transaction layer (MTL), while the receive engine transfers data from the device port to the system memory. Descriptors are used to efficiently move data from source to destination with minimal Host CPU intervention. The DMA is designed for packet-oriented data transfers such as frames in Ethernet. The controller can be programmed to interrupt the Host CPU for situations such as frame transmit and receive transfer completion as well as error conditions.

The DMA and the Host driver communicate through two data structures:†

- Control and Status registers (CSR)†
- Descriptor lists and data buffers †

18.5.2.1. Descriptor Lists and Data Buffers†

The DMA transfers data frames received by the MAC to the receive Buffer in the Host memory, and transmit data frames from the transmit Buffer in the Host memory. Descriptors that reside in the Host memory act as pointers to these buffers.†

There are two descriptor lists: one for reception and one for transmission. The base address of each list is written into Register 3 (Receive Descriptor List Address Register) and Register 4 (Transmit Descriptor List Address Register), respectively. A descriptor list is forward linked (either implicitly or explicitly). The last descriptor may point back to the first entry to create a ring structure. Explicit chaining of descriptors is accomplished by setting the second address chained in both receive and transmit descriptors (RDES1[14] and TDES0[20]). The descriptor lists resides in the Host physical memory address space. Each descriptor can point to a maximum of two buffers. This enables two buffers to be used, physically addressed, rather than contiguous buffers in memory.†

A data buffer resides in the Host physical memory space, and consists of an entire frame or part of a frame, but cannot exceed a single frame. Buffers contain only data, buffer status is maintained in the descriptor. Data chaining refers to frames that span multiple data buffers. However, a single descriptor cannot span multiple frames. The DMA skips to the next frame buffer when end-of-frame is detected. Data chaining can be enabled or disabled.†

Figure 90. Descriptor Ring Structure

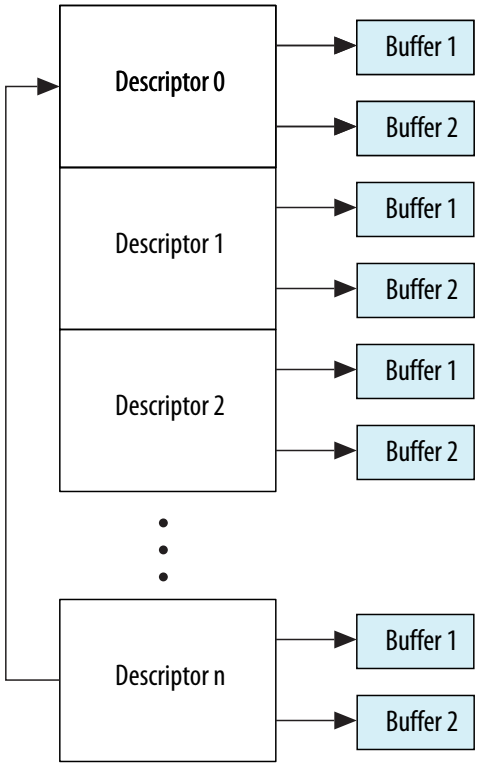
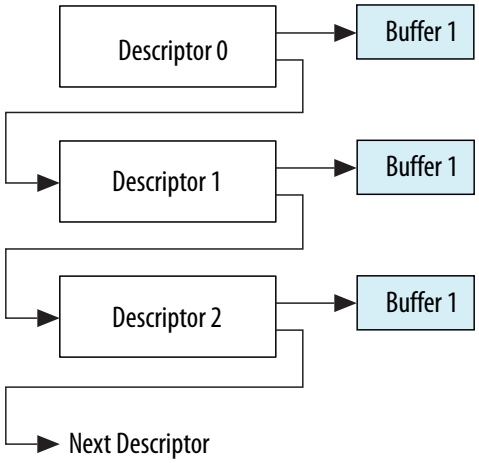


Figure 91. Descriptor Chain Structure



Note: You can select a descriptor structure during RTL configuration. The control bits in the descriptor structure are assigned so that the application can use an 8 KB buffer. All descriptions refer to the default descriptor structure.

Related Information

Ethernet MAC Address Map and Register Definitions on page 590
Information about control and status registers

18.5.2.2. Host Bus Burst Access

The DMA attempts to execute fixed-length burst transfers on the master interface if configured to do so through the FB bit of Register 0 (`Bus Mode Register`). The maximum burst length is indicated and limited by the PBL field (Bits [13:8]) Register 0 (Bus Mode Register). The receive and transmit descriptors are always accessed in the maximum possible (limited by packet burst length (PBL) or 16 * 8/bus width) burst size for the 16- bytes to be read.

The transmit DMA initiates a data transfer only when the MTL transmit FIFO has sufficient space to accommodate the configured burst or the number of bytes remaining in the frame (when it is less than the configured burst length). The DMA indicates the start address and the number of transfers required to the master interface. When the interface is configured for fixed-length burst, it transfers data using the best combination of INCR4, 8, or 16 and SINGLE transactions. When not configured for fixed-length burst, it transfers data using INCR (undefined length) and SINGLE transactions.

The receive DMA initiates a data transfer only when sufficient data to accommodate the configured burst is available in MTL receive FIFO buffer or when the end of frame (when it is less than the configured burst length) is detected in the receive FIFO buffer. The DMA indicates the start address and the number of transfers required to the master interface. When the interface is configured for fixed-length burst, it transfers data using the best combination of INCR4, 8, or 16 and SINGLE transactions. If the end-of-frame is reached before the fixed burst ends on the interface, then dummy transfers are performed in order to complete the fixed burst. If the FB bit of Register 0 (Bus Mode Register) is clear, it transfers data using INCR (undefined length) and SINGLE transactions.

When the interface is configured for address aligned words, both DMA engines ensure that the first burst transfer initiated is less than or equal to the size of the configured packet burst length. Thus, all subsequent beats start at an address that is aligned to the configured packet burst length. The DMA can only align the address for beats up to size 16 (for PBL > 16), because the interface does not support more than INCR16.

18.5.2.3. Host Data Buffer Alignment

The transmit and receive data buffers do not have any restrictions on start address alignment. For example, in systems with 32-bit memory, the start address for the buffers can be aligned to any of the four bytes. However, the DMA always initiates transfers with address aligned to the bus width with dummy data for the byte lanes not required. This typically happens during the transfer of the beginning or end of an Ethernet frame. The software driver should discard the dummy bytes based on the start address of the buffer and size of the frame.†

18.5.2.3.1. Example: Buffer Read

If the transmit buffer address is 0x00000FF2, and 15 bytes must be transferred, then the DMA reads five full words from address 0x00000FF0, but when transferring data to the MTL transmit FIFO buffer, the extra bytes (the first two bytes) are dropped or ignored. Similarly, the last three bytes of the last transfer are also ignored. The DMA always ensures it transfers data in 32-bit increments to the MTL transmit FIFO buffer, unless it is the end-of-frame.

18.5.2.3.2. Example: Buffer Write

If the receive buffer address is 0x00000FF2 and 16 bytes of a received frame must be transferred, then the DMA writes 3 full words from address 0x00000FF0. But the first two bytes of first transfer and the last two bytes of the fourth transfer have dummy data.

18.5.2.4. Buffer Size Calculations

The DMA does not update the size fields in the transmit and receive descriptors. The DMA updates only the status fields (RDES and TDES) of the descriptors. The driver must perform the size calculations.

The transmit DMA transfers the exact number of bytes (indicated by the buffer size field of TDES1) to the MAC. If a descriptor is marked as th first (FS bit of TDES1 is set), then the DMA marks the first transfer from the buffer as the start of frame. If a descriptor is marked as the last (LS bit of TDES1), then the DMA marks the last transfer from that data buffer as the end-of-frame to the MTL.

The receive DMA transfers data to a buffer until the buffer is full or the end-of-frame is received from the MTL. If a descriptor is not marked as the last (LS bit of RDES0), then the descriptor's corresponding buffer(s) are full and the amount of valid data in a buffer is accurately indicated by its buffer size field minus the data buffer pointer offset when the FS bit of that descriptor is set. The offset is zero when the data buffer pointer is aligned to the data bus width. If a descriptor is marked as the last, then the buffer may not be full (as indicated by the buffer size in RDES1). To compute the amount of valid data in this final buffer, the driver must read the frame length (FL bits of RDES0[29:16]) and subtract the sum of the buffer sizes of the preceding buffers in this frame. The receive DMA always transfers the start of next frame with a new descriptor.

Note: Even when the start address of a receive buffer is not aligned to the data width of system bus, the system should allocate a receive buffer of a size aligned to the system bus width. For example, if the system allocates a 1,024-byte (1 KB) receive buffer starting from address 0x1000, the software can program the buffer start address in the receive descriptor to have a 0x1002 offset. The receive DMA writes the frame to this buffer with dummy data in the first two locations (0x1000 and 0x1001). The actual frame is written from location 0x1002. Thus, the actual useful space in this buffer is 1,022 bytes, even though the buffer size is programmed as 1,024 bytes, because of the start address offset.

18.5.2.5. Transmission

The DMA can transmit with or without an optional second frame (OSF).

Related Information

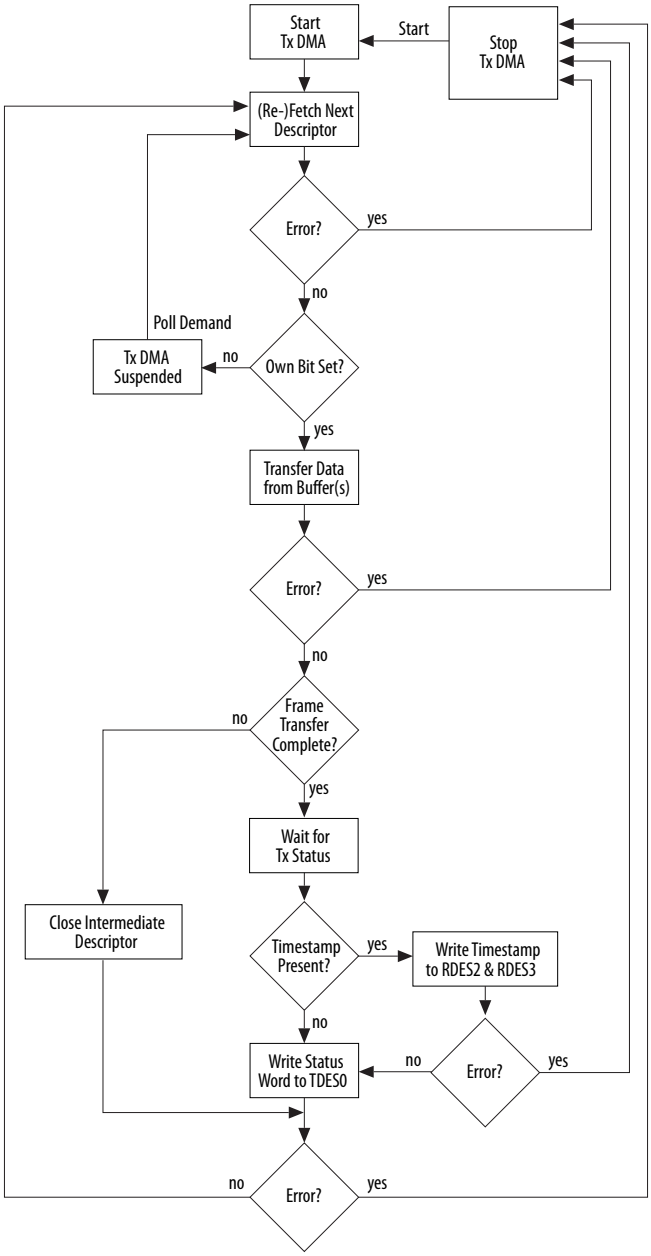
Transmit Descriptor on page 549

18.5.2.5.1. TX DMA Operation: Default (Non-OSF) Mode

The transmit DMA engine in default mode proceeds as follows: †

1. The Host sets up the transmit descriptor (TDES0-TDES3) and sets the Own bit (TDES0[31]) after setting up the corresponding data buffer(s) with Ethernet frame data. †
2. When Bit 13 (ST) of Register 6 (Operation Mode Register) is set, the DMA enters the Run state. †
3. While in the Run state, the DMA polls the transmit descriptor list for frames requiring transmission. After polling starts, it continues in either sequential descriptor ring order or chained order. If the DMA detects a descriptor flagged as owned by the Host (TDES0[31] = 0), or if an error condition occurs, transmission is suspended and both the Bit 2 (Transmit Buffer Unavailable) and Bit 16 (Normal Interrupt Summary) of the Register 5 (Status Register) are set. The transmit Engine proceeds to 9 on page 539.
4. If the acquired descriptor is flagged as owned by DMA (TDES0[31] = 1), the DMA decodes the transmit Data Buffer address from the acquired descriptor.
5. The DMA fetches the transmit data from the Host memory and transfers the data to the MTL for transmission. †
6. If an Ethernet frame is stored over data buffers in multiple descriptors, the DMA closes the intermediate descriptor and fetches the next descriptor. Repeat 3 on page 539, 4 on page 539, and 5 on page 539 until the end-of-Ethernet-frame data is transferred to the MTL. †
7. When frame transmission is complete, if IEEE 1588 timestamping was enabled for the frame (as indicated in the transmit status) the timestamp value obtained from MTL is written to the transmit descriptor (TDES2 and TDES3) that contains the end-of-frame buffer. The status information is then written to this transmit descriptor (TDES0). Because the Own bit is cleared during this step, the Host now owns this descriptor. If timestamping was not enabled for this frame, the DMA does not alter the contents of TDES2 and TDES3. †
8. Bit 0 (Transmit Interrupt) of Register 5 (Status Register) is set after completing transmission of a frame that has Interrupt on Completion (TDES1[31]) set in its Last descriptor. The DMA engine then returns to 3 on page 539. †
9. In the Suspend state, the DMA tries to re-acquire the descriptor (and thereby return to 3 on page 539) when it receives a Transmit Poll demand and the Underflow Interrupt Status bit is cleared. †

Figure 92. TX DMA Operation in Default Mode



18.5.2.5.2. TX DMA Operation: OSF Mode

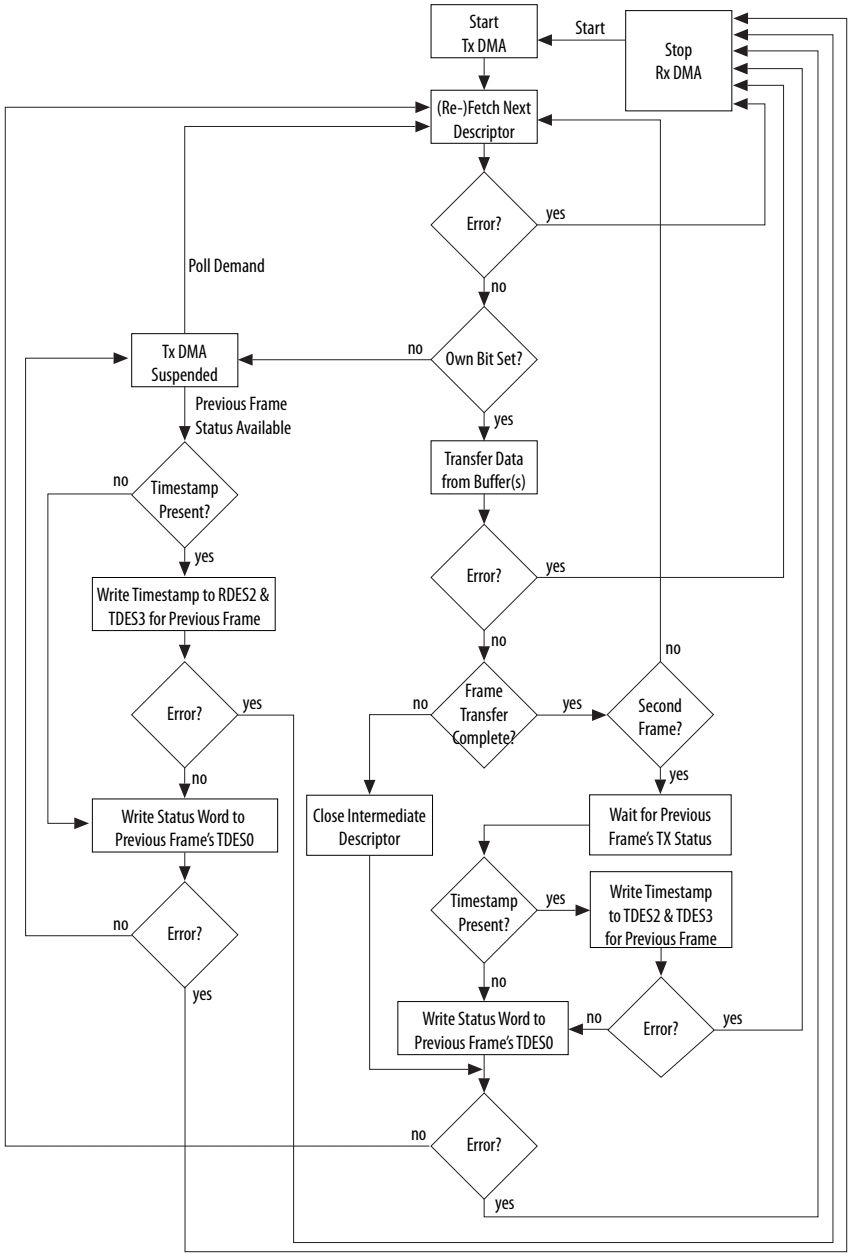
While in the Run state, the transmit process can simultaneously acquire two frames without closing the Status descriptor of the first [if Bit 2 (OSF) in Register 6 (Operation Mode Register) is set]. As the transmit process finishes transferring the first frame, it immediately polls the transmit descriptor list for the second frame. If the second frame is valid, the transmit process transfers this frame before writing the first frame's status information. †

In OSF mode, the Run state transmit DMA operates in the following sequence: †

1. The DMA operates as described in steps 1 - 6 of the TX DMA Operation: Default (Non-OSF) Mode on page 538 section.
2. Without closing the previous frame's last descriptor, the DMA fetches the next descriptor. †
3. If the DMA owns the acquired descriptor, the DMA decodes the transmit buffer address in this descriptor. If the DMA does not own the descriptor, the DMA goes into Suspend mode and skips to 7 on page 541.†
4. The DMA fetches the transmit frame from the Host memory and transfers the frame to the MTL until the End-of-frame data is transferred, closing the intermediate descriptors if this frame is split across multiple descriptors. †
5. The DMA waits for the previous frame's frame transmission status and timestamp. Once the status is available, the DMA writes the timestamp to TDES2 and TDES3, if such timestamp was captured (as indicated by a status bit). The DMA then writes the status, with a cleared Own bit, to the corresponding TDES0, thus closing the descriptor. If timestamping was not enabled for the previous frame, the DMA does not alter the contents of TDES2 and TDES3. †
6. If enabled, the transmit interrupt is set, the DMA fetches the next descriptor, then proceeds to 3 on page 541 (when Status is normal). If the previous transmission status shows an underflow error, the DMA goes into Suspend mode (7 on page 541). †
7. In Suspend mode, if a pending status and timestamp are received from the MTL, the DMA writes the timestamp (if enabled for the current frame) to TDES2 and TDES3, then writes the status to the corresponding TDES0. It then sets relevant interrupts and returns to Suspend mode. †
8. The DMA can exit Suspend mode and enter the Run state (go to 1 on page 541 or 2 on page 541 depending on pending status) only after receiving a Transmit Poll demand (Register 1 (Transmit Poll Demand Register). †

Note: As the DMA fetches the next descriptor in advance before closing the current descriptor, the descriptor chain should have more than two different descriptors for correct and proper operation. †

Figure 93. TX DMA Operation in OSF Mode



18.5.2.5.3. Transmit Frame Processing

The transmit DMA expects that the data buffers contain complete Ethernet frames, excluding preamble, pad bytes, and FCS fields and that the DA, SA, and Type/Len fields contain valid data. If the transmit descriptor indicates that the MAC must disable CRC or PAD insertion, the buffer must have complete Ethernet frames (excluding preamble), including the CRC bytes. †

Frames can be datachained and can span several buffers. Frames must be delimited by the First Descriptor (TDES1[29]) and the Last Descriptor (TDES1[30]), respectively. †

As transmission starts, the First Descriptor must have (TDES1[29]) set. When this occurs, frame data transfers from the Host buffer to the MTL transmit FIFO buffer. Concurrently, if the current frame has the Last Descriptor (TDES1[30]) clear, the transmit Process attempts to acquire the Next descriptor. The transmit Process expects this descriptor to have TDES1[29] clear. If TDES1[30] is clear, it indicates an intermediary buffer. If TDES1[30] is set, it indicates the last buffer of the frame. †

After the last buffer of the frame has been transmitted, the DMA writes back the final status information to the Transmit Descriptor 0 (TDES0) word of the descriptor that has the last segment set in Transmit Descriptor 1 (TDES1[30]). At this time, if Interrupt on Completion (TDES1[31]) is set, the Bit 0 (Transmit Interrupt) of Register 5 (Status Register) is set, the Next descriptor is fetched, and the process repeats. †

The actual frame transmission begins after the MTL transmit FIFO buffer has reached either a programmable transmit threshold (Bits [16:14] of Register 6 (Operation Mode Register)), or a full frame is contained in the FIFO buffer. There is also an option for Store and Forward Mode (Bit 21 of Register 6 (Operation Mode Register)). Descriptors are released (Own bit TDES0[31] clears) when the DMA finishes transferring the frame. †

Note: To ensure proper transmission of a frame and the next frame, you must specify a non-zero buffer size for the transmit descriptor that has the Last Descriptor (TDES1[30]) set. †

18.5.2.5.4. Transmit Polling Suspended

Transmit polling can be suspended by either of the following conditions: †

- The DMA detects a descriptor owned by the Host (TDES0[31]=0). To resume, the driver must give descriptor ownership to the DMA and then issue a Poll Demand command. †
- A frame transmission is aborted when a transmit error because of underflow is detected. The appropriate Transmit Descriptor 0 (TDES0) bit is set. †

If the DMA goes into SUSPEND state because of the first condition, then both Bit 16 (Normal Interrupt Summary) and Bit 2 (Transmit Buffer Unavailable) of Register 5 (Status Register) are set. If the second condition occur, both Bit 15 (Abnormal Interrupt Summary) and Bit 5 (Transmit Underflow) of Register 5 (Status Register) are set, and the information is written to Transmit Descriptor 0, causing the suspension. †

In both cases, the position in the transmit List is retained. The retained position is that of the descriptor following the Last descriptor closed by the DMA. †

The driver must explicitly issue a Transmit Poll Demand command after rectifying the suspension cause. †

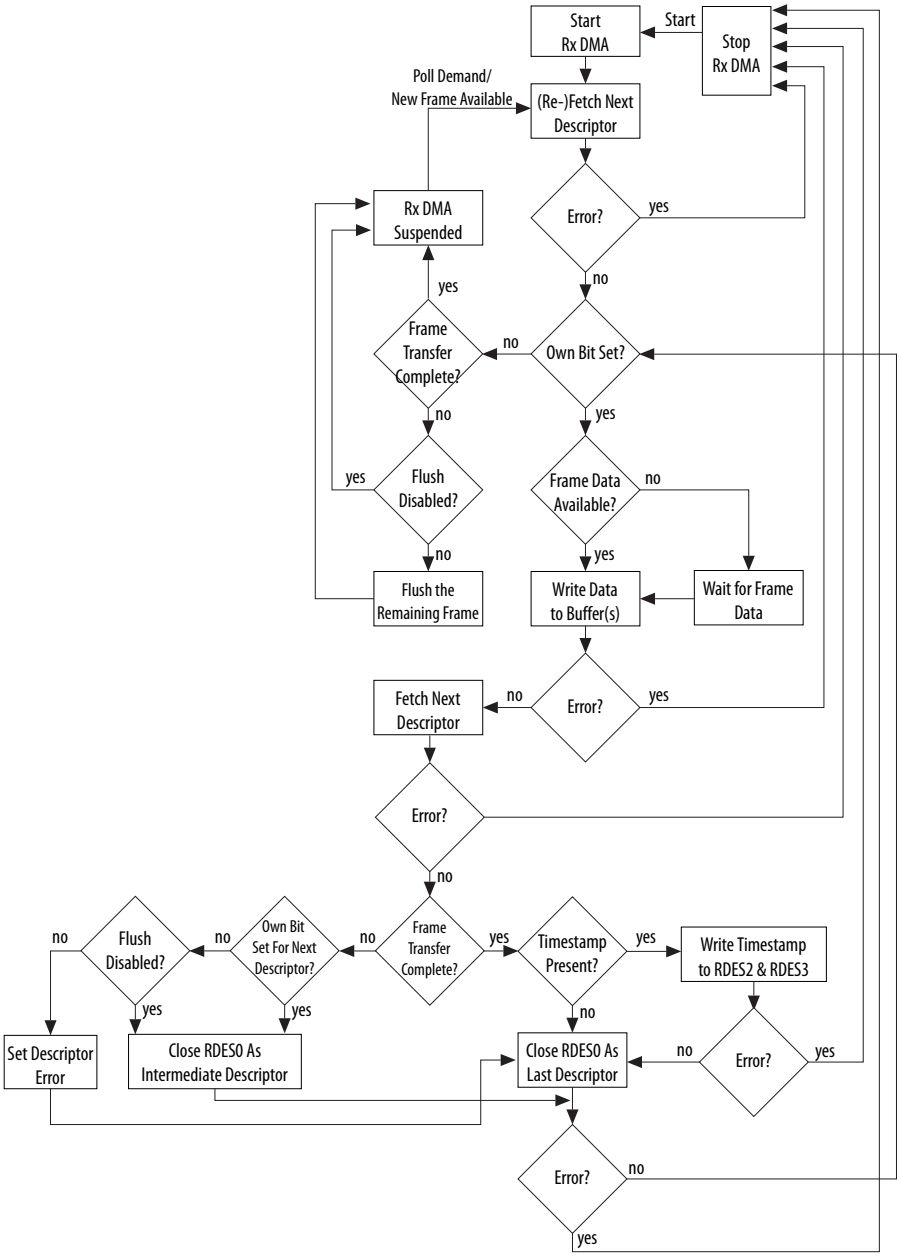
18.5.2.6. Reception

Receive functions use receive descriptors.

The receive DMA engine's reception sequence is proceeds as follows:

1. The host sets up receive descriptors (RDES0-RDES3) and sets the Own bit (RDES0[31]).†
2. When Bit 1 (SR) of Register 6 (Operation Mode Register) is set, the DMA enters the Run state. While in the Run state, the DMA polls the receive descriptor list, attempting to acquire free descriptors. If the fetched descriptor is not free (is owned by the host), the DMA enters the Suspend state and jumps to 9 on page 544.†
3. The DMA decodes the receive data buffer address from the acquired descriptors.†
4. Incoming frames are processed and placed in the acquired descriptor's data buffers.†
5. When the buffer is full or the frame transfer is complete, the receive engine fetches the next descriptor.†
6. If the current frame transfer is complete, the DMA proceeds to 7 on page 544. If the DMA does not own the next fetched descriptor and the frame transfer is not complete (EOF is not yet transferred), the DMA sets the Descriptor Error bit in the RDES0 (unless flushing is disabled in Bit 24 of Register 6 (Operation Mode Register)). The DMA closes the current descriptor (clears the Own bit) and marks it as intermediate by clearing the Last Segment (LS) bit in the RDES0 value (marks it as Last Descriptor if flushing is not disabled), then proceeds to 8 on page 544. If the DMA does own the next descriptor but the current frame transfer is not complete, the DMA closes the current descriptor as intermediate and reverts to 4 on page 544.†
7. If IEEE 1588 timestamping is enabled, the DMA writes the timestamp (if available) to the current descriptor's RDES2 and RDES3. It then takes the receive frame's status from the MTL and writes the status word to the current descriptor's RDES0, with the Own bit cleared and the Last Segment bit set.†
8. The receive engine checks the latest descriptor's Own bit. If the host owns the descriptor (Own bit is 0), the Bit 7 (Receive Buffer Unavailable) of Register 5 (Status Register) is set and the DMA receive engine enters the Suspended state (Step 9). If the DMA owns the descriptor, the engine returns to 4 on page 544 and awaits the next frame.
9. Before the receive engine enters the Suspend state, partial frames are flushed from the receive FIFO buffer. You can control flushing using Bit 24 of Register 6 (Operation Mode Register). †
10. The receive DMA exits the Suspend state when a Receive Poll demand is given or the start of next frame is available from the MTL's receive FIFO buffer. The engine proceeds to 2 on page 544 and refetches the next descriptor. †

Figure 94. Receive DMA Operation



When software has enabled timestamping through the `tsena` bit of register 448 (Timestamp Control Register) and a valid timestamp value is not available for the frame (for example, because the receive FIFO buffer was full before the timestamp could be written to it), the DMA writes all ones to RDES2 and RDES3 descriptors . Otherwise (that is, if timestamping is not enabled), the RDES2 and RDES3 descriptors remain unchanged.

18.5.2.6.1. Receive Descriptor Acquisition

The receive Engine always attempts to acquire an extra descriptor in anticipation of an incoming frame. Descriptor acquisition is attempted if any of the following conditions is satisfied: †

- Bit 1 (Start or Stop Receive) of Register 6 (Operation Mode Register) has been set immediately after being placed in the Run state. †
- The data buffer of the current descriptor is full before the frame ends for the current transfer. †
- The controller has completed frame reception, but the current receive descriptor is not yet closed. †
- The receive process has been suspended because of a host-owned buffer (RDES0[31] = 0) and a new frame is received. †
- A Receive poll demand has been issued. †

18.5.2.6.2. Receive Frame Processing

The MAC transfers the received frames to the Host memory only when the frame passes the address filter and frame size is greater than or equal to the configurable threshold bytes set for the receive FIFO buffer of MTL, or when the complete frame is written to the FIFO buffer in store-and-forward mode. †

If the frame fails the address filtering, it is dropped in the MAC block itself (unless Bit 31 (Receive All) of Register 1 (MAC Frame Filter) is set). Frames that are shorter than 64 bytes, because of collision or premature termination, can be removed from the MTL receive FIFO buffer. †

After 64 (configurable threshold) bytes have been received, the MTL block requests the DMA block to begin transferring the frame data to the receive buffer pointed by the current descriptor. The DMA sets the First Descriptor (RDES0[9]) after the DMA Host interface becomes ready to receive a data transfer (if the DMA is not fetching transmit data from the host), to delimit the frame. The descriptors are released when the Own (RDES[31]) bit is clear, either as the Data buffer fills up or as the last segment of the frame is transferred to the receive buffer. If the frame is contained in a single descriptor, both Last Descriptor (RDES0[8]) and First Descriptor (RDES0[9]) are set.

The DMA fetches the next descriptor, sets the Last Descriptor (RDES[8]) bit, and releases the RDES0 status bits in the previous frame descriptor. Then the DMA sets bit 6 (Receive Interrupt) of Register 5 (Status Register). The same process repeats unless the DMA encounters a descriptor flagged as being owned by the host. If this occurs,Bit 7 (Receive Buffer Unavailable) of Register 5 (Status Register) is set and the receive process enters the Suspend state. The position in the receive list is retained. †

18.5.2.6.3. Receive Process Suspended

If a new receive frame arrives while the receive process is in the suspend state, the DMA refetches the current descriptor in the Host memory. If the descriptor is now owned by the DMA, the receive process re-enters the run state and starts frame reception. If the descriptor is still owned by the host, by default, the DMA discards the current frame at the top of the MTL RX FIFO buffer and increments the missed frame counter. If more than one frame is stored in the MTL EX FIFO buffer, the process repeats. †

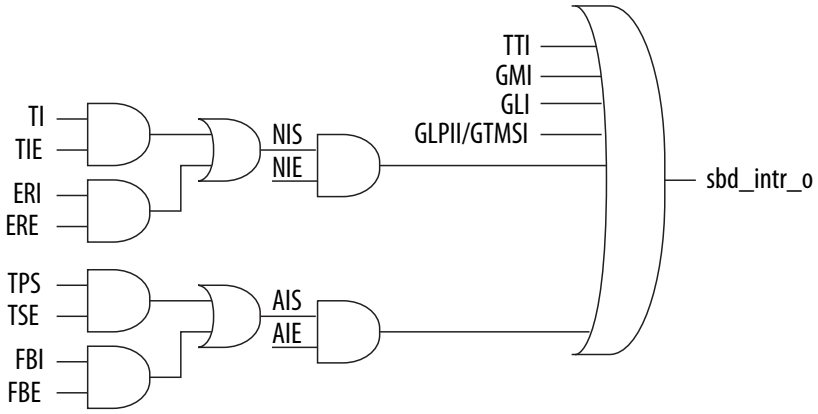
The discarding or flushing of the frame at the top of the MTL EX FIFO buffer can be avoided by disabling Flushing (Bit 24 of Register 6 (Operation Mode Register)). In such conditions, the receive process sets the Receive Buffer Unavailable status and returns to the Suspend state. †

18.5.2.7. Interrupts

Interrupts can be generated as a result of various events. The DMA Register 5 (Status Register) contains a status bit for each of the events that can cause an interrupt. Register 7 (Interrupt Enable Register) contains an enable bit for each of the possible interrupt sources.

There are two groups of interrupts, Normal and Abnormal, as described in Register 5 (Status Register). Interrupts are cleared by writing a 1 to the corresponding bit position. When all the enabled interrupts within a group are cleared, the corresponding summary bit is cleared. When both the summary bits are cleared, the `sbd_intr_o` interrupt signal is deasserted. If the MAC is the cause for assertion of the interrupt, then any of the `GLI`, `GMI`, `TTI`, or `GLPII` bits of Register 5 (Status Register) are set to 1.

Figure 95. Summary Interrupt (sbd_intr_o) Generation(48)



Note: Register 5 (Status Register) is the interrupt status register. The interrupt pin (`sbd_intr_o`) is asserted because of any event in this status register only if the corresponding interrupt enable bit is set in Register 7 (Interrupt Enable Register). †

Interrupts are not queued, and if the interrupt event occurs before the driver has responded to it, no additional interrupts are generated. For example, Bit 6 (Receive Interrupt) of Register 5 (Status Register) indicates that one or more frames were transferred to the Host buffer. The driver must scan all descriptors, from the last recorded position to the first one owned by the DMA.

An interrupt is generated only once for multiple, simultaneous events. The driver must scan Register 5 (Status Register) for the cause of the interrupt. After the driver has cleared the appropriate bit in Register 5 (Status Register), the interrupt is not generated again until a new interrupting event occurs. For example, the controller sets Bit 6 (Receive Interrupt) of Register 5 (Status Register) and the driver begins reading Register 5 (Status Register). Next, the interrupt indicated by Bit 7 (Receive Buffer

(48) Signals NIS and AIS are registered.

Unavailable) of Register 5 (Status Register) occurs. The driver clears the receive interrupt (bit 6). However, the `sbd_intr_o` signal is not deasserted, because of the active or pending Receive Buffer Unavailable interrupt.

Bits 7:0 (`riwt` field) of Register 9 (Receive Interrupt Watchdog Timer Register) provide for flexible control of the receive interrupt. When this Interrupt timer is programmed with a non-zero value, it gets activated as soon as the RX DMA completes a transfer of a received frame to system memory without asserting the receive Interrupt because it is not enabled in the corresponding Receive Descriptor (RDES1[31]). When this timer runs out as per the programmed value, the AIS bit is set and the interrupt is asserted if the corresponding AIE is enabled in Register 7 (Interrupt Enable Register). This timer is disabled before it runs out, when a frame is transferred to memory, and the receive interrupt is triggered if it is enabled.

Related Information

Receive Descriptor on page 554

18.5.2.8. Error Response to DMA

If the slave replies with an error response to any data transfer initiated by a DMA channel, that DMA stops all operations and updates the error bits and the Fatal Bus Error bit in the Register 5 (Status Register). The DMA controller can resume operation only after soft resetting or hard resetting the EMAC and reinitializing the DMA.

18.5.3. Descriptor Overview

The DMA in the Ethernet subsystem transfers data based on a single enhanced descriptor, as explained in the DMA Controller section. The enhanced descriptor is created in the system memory. The descriptor addresses must be word-aligned.

The enhanced or alternate descriptor format can have 8 DWORDS (32 bytes) instead of 4 DWORDS as in the case of the normal descriptor format.

The features of the enhanced or alternate descriptor structure are:

- The alternative descriptor structure is implemented to support buffers of up to 8 KB (useful for Jumbo frames).†
- There is a re-assignment of control and status bits in TDES0, TDES1, RDES0 (advanced timestamp or IPC full offload configuration), and RDES1. †
- The transmit descriptor stores the timestamp in TDES6 and TDES7 when you select the advanced timestamp. †
- The receive descriptor structure is also used for storing the extended status (RDES4) and timestamp (RDES6 and RDES7) when advanced timestamp, IPC Full Checksum Offload Engine, or Layer 3 and Layer 4 filter feature is selected. †
- You can select one of the following options for descriptor structure:
  - If timestamping is enabled in Register 448 (Timestamp Control Register) or Checksum Offload is enabled in Register 0 (MAC Configuration Register), the software must to allocate 32 bytes (8 DWORDS) of memory for every descriptor by setting Bit 7 (Descriptor Size) of Register 0 (Bus Mode Register).
  - If timestamping or Checksum Offload is not enabled, the extended descriptors (DES4 to DES7) are not required. Therefore, software can use descriptors with the default size of 16 bytes (4 DWORDS) by clearing Bit 7 (Descriptor Size) of Register 0 (Bus Mode Register) to 0.

Related Information

DMA Controller on page 535

18.5.3.1. Transmit Descriptor

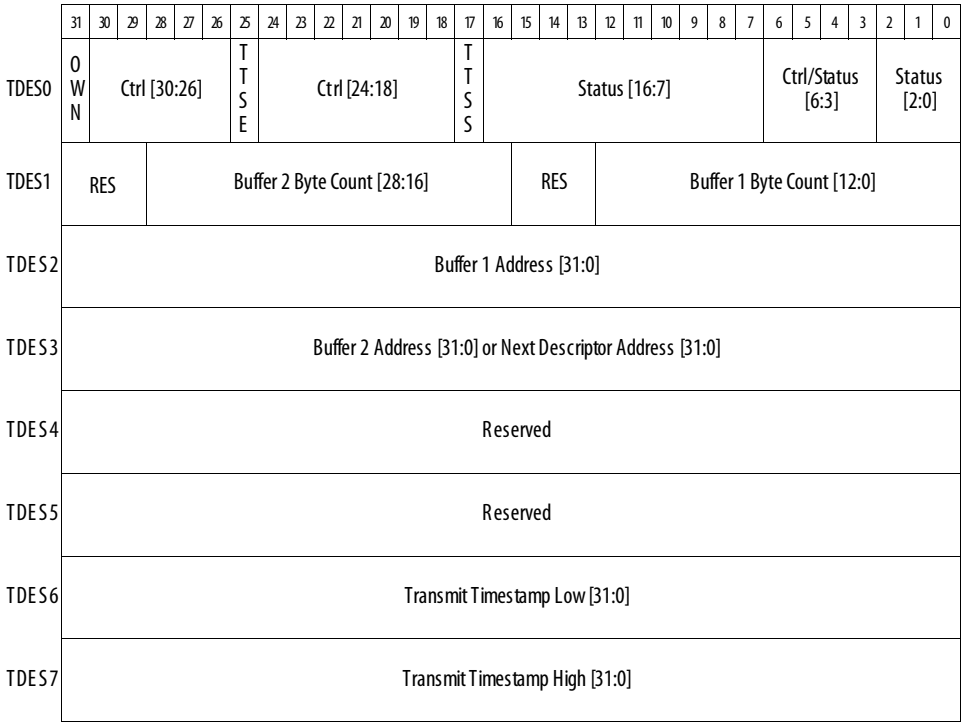
The application software must program the control bits TDES0[31:18] during the transmit descriptor initialization. When the DMA updates the descriptor, it writes back all the control bits except the OWN bit (which it clears) and updates the status bits[7:0].

With the advance timestamp support, the snapshot of the timestamp to be taken can be enabled for a given frame by setting Bit 25 (TTSE) of TDES0. When the descriptor is closed (that is, when the OWN bit is cleared), the timestamp is written into TDES6 and TDES7 as indicated by the status Bit 17 (TTSS) of TDES0.

Note: Only enhanced descriptor formats (4 or 8 DWORDS) are supported.

Note: When the advanced timestamp feature is enabled, software should set Bit 7 of Register 0 (Bus Mode Register), so that the DMA operates with extended descriptor size. When this control bit is clear, the TDES4-TDES7 descriptor space is not valid. †

Figure 96. Transmit Enhanced Descriptor Fields - Format



The DMA always reads or fetches four DWORDS of the descriptor from system memory to obtain the buffer and control information. †

Figure 97. Transmit Descriptor Fetch (Read) Format

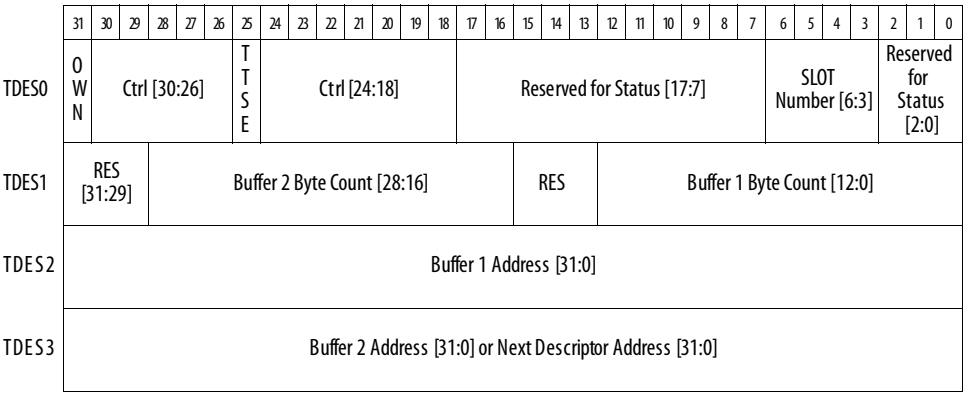


Table 185. Transmit Descriptor Word 0 (TDES0)

Bit	Description
31	OWN: Own Bit When set, this bit indicates that the descriptor is owned by the DMA. When this bit is cleared, it indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame transmission or when the buffers allocated in the descriptor are read completely. The ownership bit of the

continued...

Bit	Description
	frame's first descriptor must be set after all subsequent descriptors belonging to the same frame have been set to avoid a possible race condition between fetching a descriptor and the driver setting an ownership bit. †
30	IC: Interrupt on Completion When set, this bit enables the Transmit Interrupt (Register 5[0]) to be set after the present frame has been transmitted. †
29	LS: Last Segment When set, this bit indicates that the buffer contains the last segment of the frame. When this bit is set, the TBS1 or TBS2 field in TDES1 should have a non-zero value. †
28	FS: First Segment When set, this bit indicates that the buffer contains the first segment of a frame. †
27	DC: Disable CRC When this bit is set, the MAC does not append a CRC to the end of the transmitted frame. This bit is valid only when the first segment (TDES0[28]) is set. †
26	DP: Disable Pad When set, the MAC does not automatically add padding to a frame shorter than 64 bytes. When this bit is cleared, the DMA automatically adds padding and CRC to a frame shorter than 64 bytes, and the CRC field is added despite the state of the DC (TDES0[27]) bit. This bit is valid only when the first segment (TDES0[28]) is set. †
25	TTSE: Transmit Timestamp Enable When set, this bit enables IEEE1588 hardware timestamping for the transmit frame referenced by the descriptor. This field is valid only when the First Segment control bit (TDES0[28]) is set.
24	Reserved
23:22	CIC: Checksum Insertion Control. These bits control the checksum calculation and insertion. The following list describes the bit encoding: ☐ 0x0: Checksum insertion disabled. ☐ 0x1: Only IP header checksum calculation and insertion are enabled. ☐ 0x2: IP header checksum and payload checksum calculation and insertion are enabled, but pseudoheader checksum is not calculated in hardware. ☐ 0x3: IP Header checksum and payload checksum calculation and insertion are enabled, and pseudoheader checksum is calculated in hardware. This field is valid when the First Segment control bit (TDES0[28]) is set.
21	TER: Transmit End of Ring When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring. †
20	TCH: Second Address Chained When set, this bit indicates that the second address in the descriptor is the Next descriptor address rather than the second buffer address. When TDES0[20] is set, TBS2 (TDES1[28:16]) is a "don't care" value. TDES0[21] takes precedence over TDES0[20]. †
19:18	Reserved
17	TTSS: Transmit Timestamp Status This field is used as a status bit to indicate that a timestamp was captured for the described transmit frame. When this bit is set, TDES2 and TDES3 have a timestamp value captured for the transmit frame. This field is only valid when the descriptor's Last Segment control bit (TDES0[29]) is set. †
16	IHE: IP Header Error When set, this bit indicates that the MAC transmitter detected an error in the IP datagram header. The transmitter checks the header length in the IPv4 packet against the number of header bytes received from the application and indicates an error status if there is a mismatch. For IPv6 frames, a header error is reported if the main header length is not 40 bytes. Furthermore, the Ethernet Length/Type field value for an IPv4 or IPv6 frame must match the IPheader version received with the packet. For IPv4 frames, an error status is also indicated if the Header Length field has a value less than 0x5. †

continued...

Bit	Description
	This bit is valid only when the Tx Checksum Offload is enabled. If COE detects an IP header error, it still inserts an IPv4 header checksum if the Ethernet Type field indicates an IPv4 payload.†
15	ES: Error Summary Indicates the logical OR of the following bits: ☐ TDES0[14]: Jabber Timeout ☐ TDES0[13]: Frame Flush ☐ TDES0[11]: Loss of Carrier ☐ TDES0[10]: No Carrier ☐ TDES0[9]: Late Collision ☐ TDES0[8]: Excessive Collision ☐ TDES0[2]: Excessive Deferral ☐ TDES0[1]: Underflow Error ☐ TDES0[16]: IP Header Error ☐ TDES0[12]: IP Payload Error †
14	JT: Jabber Timeout When set, this bit indicates the MAC transmitter has experienced a jabber time-out. This bit is only set when Bit 22 (Jabber Disable) of Register 0 (MAC Configuration Register) is not set. †
13	FF: Frame Flushed When set, this bit indicates that the DMA or MTL flushed the frame because of a software Flush command given by the CPU. †
12	IPE: IP Payload Error When set, this bit indicates that MAC transmitter detected an error in the TCP, UDP, or ICMP IP datagram payload. The transmitter checks the payload length received in the IPv4 or IPv6 header against the actual number of TCP, UDP, or ICMP packet bytes received from the application and issues an error status in case of a mismatch. †
11	LC: Loss of Carrier When set, this bit indicates that a loss of carrier occurred during frame transmission (that is, the gmii_crs_i signal was inactive for one or more transmit clock periods during frame transmission). This bit is valid only for the frames transmitted without collision when the MAC operates in the half-duplex mode. †
10	NC: No Carrier When set, this bit indicates that the Carrier Sense signal form the PHY was not asserted during transmission. †
9	LC: Late Collision When set, this bit indicates that frame transmission is aborted because of a collision occurring after the collision window (64 byte-times, including preamble, in MII mode and 512 byte-times, including preamble and carrier extension, in GMII mode). This bit is not valid if the Underflow Error bit is set.
8	EC: Excessive Collision When set, this bit indicates that the transmission was aborted after 16 successive collisions while attempting to transmit the current frame. If Bit 9 (Disable Retry) in Register 0 (MAC Configuration Register) is set, this bit is set after the first collision, and the transmission of the frame is aborted. †
7	VF: VLAN Frame When set, this bit indicates that the transmitted frame is a VLAN-type frame. †
6:3	CC: Collision Count (Status field) These status bits indicate the number of collisions that occurred before the frame was transmitted. This count is not valid when the Excessive Collisions bit (TDES0[8]) is set. The EMAC updates this status field only in the half-duplex mode.
2	ED: Excessive Deferral When set, this bit indicates that the transmission has ended because of excessive deferral of over 24,288 bit times (155,680 bits times in 1,000-Mbps mode or if Jumbo frame is enabled) if Bit 4 (Deferral Check) bit in Register 0 (MAC Configuration Register) is set. †
1	UF: Underflow Error

continued...

Bit	Description
	When set, this bit indicates that the MAC aborted the frame because the data arrived late from the Host memory. Underflow Error indicates that the DMA encountered an empty transmit buffer while transmitting the frame. The transmission process enters the Suspended state and sets both Transmit Underflow (Register 5[5]) and Transmit Interrupt (Register 5[0]). †
0	DB: Deferred Bit When set, this bit indicates that the MAC defers before transmission because of the presence of carrier. This bit is valid only in the half-duplex mode. †

Table 186. Transmit Descriptor Word 1 (TDES1)

Bit	Description
31:29	Reserved
28:16	TBS2: Transmit Buffer 2 Size This field indicates the second data buffer size in bytes. This field is not valid if TDES0[20] is set. †
15:13	Reserved †
12:0	TBS1: Transmit Buffer 1 Size This field indicates the first data buffer byte size, in bytes. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor, depending on the value of TCH (TDES0[20]). †

Table 187. Transmit Descriptor 2 (TDES2)

Bit	Description
31:0	Buffer 1 Address Pointer These bits indicate the physical address of Buffer 1. There is no limitation on the buffer address alignment. †

Table 188. Transmit Descriptor 3 (TDES3)

Bit	Description
31:0	Buffer 2 Address Pointer (Next Descriptor Address) Indicates the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (TDES0[20]) bit is set, this address contains the pointer to the physical memory where the Next descriptor is present. The buffer address pointer must be aligned to the bus width only when TDES0[20] is set. (LSBs are ignored internally.) †

Table 189. Transmit Descriptor 6 (TDES6)

Bit	Description
31:0	TTSL: Transmit Frame Timestamp Low This field is updated by DMA with the least significant 32 bits of the timestamp captured for the corresponding transmit frame. This field has the timestamp only if the Last Segment bit (LS) in the descriptor is set and Timestamp status (TTSS) bit is set. †

Table 190. Transmit Descriptor 7 (TDES7)

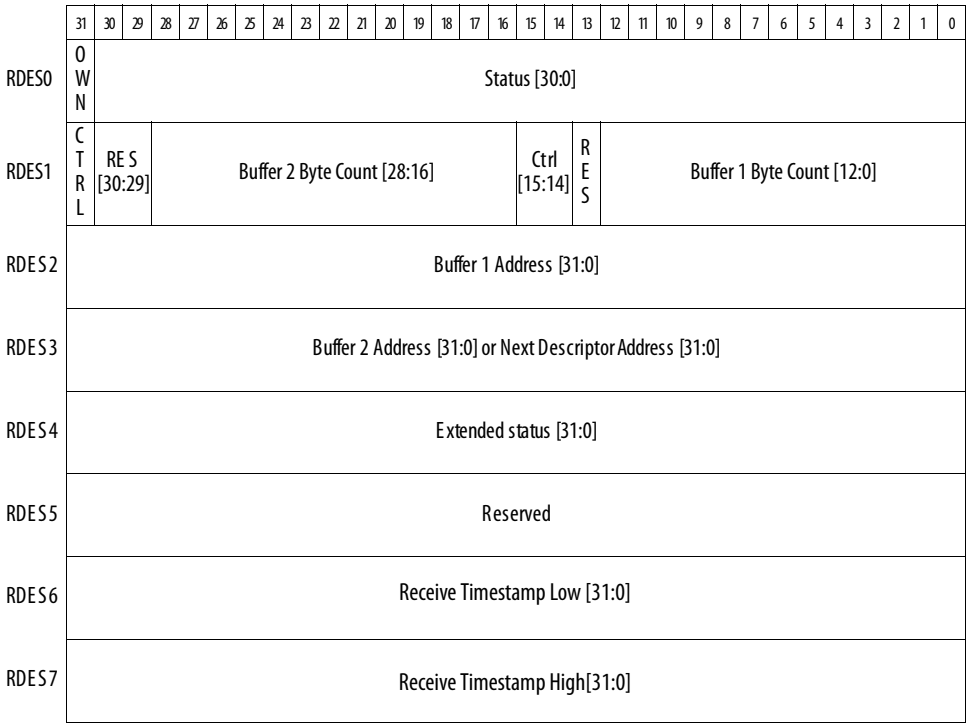
Bit	Description
31:0	TTSH: Transmit Frame Timestamp High This field is updated by DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field has the timestamp only if the Last Segment bit (LS) in the descriptor is set and Timestamp status (TTSS) bit is set. †

18.5.3.2. Receive Descriptor

The receive descriptor can have 32 bytes of descriptor data (8 DWORDs) when advanced timestamp or IPC Full Offload feature is selected. When either of these features is enabled, software should set bit 7 of Register 0 (Bus Mode Register) so that the DMA operates with extended descriptor size. When this control bit is clear, the RDES0[0] is always cleared and the RDES4-RDES7 descriptor space is not valid. †

Note: Only enhanced descriptor formats (4 or 8 DWORDS) are supported.

Figure 98. Receive Enhanced Descriptor Fields Format



18.5.3.2.1. Receive Descriptor Field 0 (RDES0)

Table 191. Receive Descriptor Field 0 (RDES0)

Bit	Description
31	OWN: Own Bit When set, this bit indicates that the descriptor is owned by the DMA of the EMAC. When this bit is cleared, this bit indicates that the descriptor is owned by the Host. The DMA clears this bit either when it completes the frame reception or when the buffers that are associated with this descriptor are full.
30	AFM: Destination Address Filter Fail When set, this bit indicates a frame that failed in the DA Filter in the MAC. †
29:16	FL: Frame Length

continued...

Bit	Description
	These bits indicate the byte length of the received frame that was transferred to host memory (including CRC). This field is valid when Last Descriptor (RDES0[8]) is set and either the Descriptor Error (RDES0[14]) or Overflow Error bits are cleared. The frame length also includes the two bytes appended to the Ethernet frame when IP checksum calculation (Type 1) is enabled and the received frame is not a MAC control frame. This field is valid when Last Descriptor (RDES0[8]) is set. When the Last Descriptor and Error Summary bits are not set, this field indicates the accumulated number of bytes that have been transferred for the current frame. †
15	ES: Error Summary Indicates the logical OR of the following bits: • RDES0[1]: CRC Error • RDES0[3]: Receive Error • RDES0[4]: Watchdog Timeout • RDES0[6]: Late Collision • RDES0[7]: Giant Frame • RDES4[4:3]: IP Header or Payload Error (Receive Descriptor Field 4 (RDES4)) • RDES0[11]: Overflow Error • RDES0[14]: Descriptor Error This field is valid only when the Last Descriptor (RDES0[8]) is set. †
14	DE: Descriptor Error When set, this bit indicates a frame truncation caused by a frame that does not fit within the current descriptor buffers, and that the DMA does not own the Next descriptor. The frame is truncated. This bit is valid only when the Last Descriptor (RDES0[8]) bit is set. †
13	SAF: Source Address Filter Fail When set, this bit indicates that the SA field of frame failed the SA Filter in the MAC. †
12	LE: Length Error When set, this bit indicates that the actual length of the frame received and that the Length/ Type field does not match. This bit is valid only when the Frame Type (RDES0[5]) bit is clear. †
11	OE: Overflow Error When set, this bit indicates that the received frame was damaged because of buffer overflow in MTL. Note: This bit is set only when the DMA transfers a partial frame to the application, which happens only when the RX FIFO buffer is operating in the threshold mode. In the store-and-forward mode, all partial frames are dropped completely in the RX FIFO buffer. †
10	VLAN: VLAN Tag When set, this bit indicates that the frame to which this descriptor is pointing is a VLAN frame tagged by the MAC. The VLAN tagging depends on checking the VLAN fields of the received frame based on the Register 7 (VLAN Tag Register) setting. †
9	FS: First Descriptor When set, this bit indicates that this descriptor contains the first buffer of the frame. If the size of the first buffer is 0, the second buffer contains the beginning of the frame. If the size of the second buffer is also 0, the next descriptor contains the beginning of the frame. †

continued...

Bit	Description
8	LD: Last Descriptor When set, this bit indicates that the buffers pointed to by this descriptor are the last buffers of the frame. †
7	Timestamp Available When set, bit[7] indicates that a snapshot of the Timestamp is written in descriptor words 6 (RDES6) and 7 (RDES7). This is valid only when the Last Descriptor bit (RDES0[8]) is set.
6	LC: Late Collision When set, this bit indicates that a late collision has occurred while receiving the frame in the half-duplex mode. †
5	FT: Frame Type When set, this bit indicates that the receive frame is an Ethernet-type frame (the LT field is greater than or equal to 0x0600). When this bit is cleared, it indicates that the received frame is an IEEE802.3 frame. This bit is not valid for Runt frames less than 14 bytes.
4	RWT: Receive Watchdog Timeout When set, this bit indicates that the receive Watchdog Timer has expired while receiving the current frame and the current frame is truncated after the Watchdog Timeout. †
3	RE: Receive Error When set, this bit indicates that the `gmii_rxer_i` signal is asserted while `gmii_rxdv_i` is asserted during frame reception.
2	DE: Dribble Bit Error When set, this bit indicates that the received frame has a non-integer multiple of bytes (odd nibbles). This bit is valid only in the MII Mode. †
1	CE: CRC Error When set, this bit indicates that a CRC error occurred on the received frame. This bit is valid only when the Last Descriptor (RDES0[8]) is set. †
0	Extended Status Available/RX MAC Address When either advanced timestamp or IP Checksum Offload (Type 2) is present, this bit, when set, indicates that the extended status is available in descriptor word 4 (RDES4). This bit is valid only when the Last Descriptor bit (RDES0[8]) is set. When the Advance Timestamp Feature or IPC Full Offload is not selected, this bit indicates RX MAC Address status. When set, this bit indicates that the RX MAC Address registers value (1 to 15) matched the frame's DA field. When clear, this bit indicates that the RX MAC Address Register 0 value matched the DA field. †

Related Information

- Receive Descriptor Field 4 (RDES4) on page 558
- Receive Descriptor Field 6 (RDES6) on page 560
- Receive Descriptor Field 7 (RDES7) on page 560

18.5.3.2.2. Receive Descriptor Field 1 (RDES1)

Table 192. Receive Descriptor Field 1 (RDES1)

Bit	Description
31	DIC: Disable Interrupt on Completion When set, this bit prevents setting the Status Register's RI bit (CSR5[6]) for the received frame ending in the buffer indicated by this descriptor. As a result, the RI interrupt for the frame is disabled and is not asserted to the Host.†
30:29	Reserved †
28:16	RBS2: Receive Buffer 2 Size These bits indicate the second data buffer size, in bytes. The buffer size must be a multiple of 4, even if the value of RDES3 (buffer2 address pointer) in the Receive Descriptor Field 3 (RDES3) is not aligned to the bus width. If the buffer size is not an appropriate multiple of 4, the resulting behavior is undefined. This field is not valid if RDES1[14] is set. For more information about calculating buffer sizes, refer to the Buffer Size Calculations section in this chapter.
15	RER: Receive End of Ring When set, this bit indicates that the descriptor list reached its final descriptor. The DMA returns to the base address of the list, creating a descriptor ring. †
14	RCH: Second Address Chained When set, this bit indicates that the second address in the descriptor is the next descriptor address rather than the second buffer address. When this bit is set, RBS2 (RDES1[28:16]) is a "don't care" value. RDES1[15] takes precedence over RDES1[14]. †
13	Reserved †
12:0	RBS1: Receive Buffer 1 Size Indicates the first data buffer size in bytes. The buffer size must be a multiple of 4, even if the value of RDES2 (buffer1 address pointer), in the Receive Descriptor Field 2 (RDES2), is not aligned. When the buffer size is not a multiple of 4, the resulting behavior is undefined. If this field is 0, the DMA ignores this buffer and uses Buffer 2 or the next descriptor depending on the value of RCH (Bit 14). For more information about calculating buffer sizes, refer to the Buffer Size Calculations section in this chapter.

Related Information

- Buffer Size Calculations on page 538
- Receive Descriptor Field 2 (RDES2) on page 557
- Receive Descriptor Field 3 (RDES3) on page 558

18.5.3.2.3. Receive Descriptor Fields (RDES2) and (RDES3)

Receive Descriptor Field 2 (RDES2)

Table 193. Receive Descriptor Field 2 (RDES2)

Bit	Description
31:0	Buffer 1 Address Pointer These bits indicate the physical address of Buffer 1. There are no limitations on the buffer address alignment except for the following condition: The DMA uses the value programmed in RDES2[1:0] for its address generation when the RDES2 value is used to store the start of the frame. The DMA performs a write operation with the RDES2[1:0] bits as 0 during the transfer of the start of the frame but the frame is shifted as per the actual buffer address pointer. The DMA ignores RDES2[1:0] if the address pointer is to a buffer where the middle or last part of the frame is stored. For more information about buffer address alignment, refer to the *Host Data Buffer Alignment* section.

Related Information

Host Data Buffer Alignment on page 537

Receive Descriptor Field 3 (RDES3)

Table 194. Receive Descriptor Field 3 (RDES3)

Bit	Description
31:0	Buffer 2 Address Pointer (Next Descriptor Address) These bits indicate the physical address of Buffer 2 when a descriptor ring structure is used. If the Second Address Chained (RDES1[14]) bit in Receive Descriptor Field 1 (RDES1) is set, this address contains the pointer to the physical memory where the Next descriptor is present. If RDES1[14], in the Receive Descriptor Field 1 (RDES1) is set, the buffer (Next descriptor) address pointer must be bus width-aligned (RDES3[1:0] = 0. LSBs are ignored internally.) However, when RDES1[14] in the Receive Descriptor Field 1 (RDES1) is cleared, there are no limitations on the RDES3 value, except for the following condition: the DMA uses the value programmed in RDES3 [1:0] for its buffer address generation when the RDES3 value is used to store the start of frame. The DMA ignores RDES3 [1:0] if the address pointer is to a buffer where the middle or last part of the frame is stored.

Related Information

- Host Data Buffer Alignment on page 537
- Receive Descriptor Field 1 (RDES1) on page 557

18.5.3.2.4. Receive Descriptor Field 4 (RDES4)

The extended status is written only when there is status related to IPC or timestamp available. The availability of extended status is indicated by Bit 0 in RDES0. This status is available only when the Advance Timestamp or IPC Full Offload feature is selected.

Table 195. Receive Descriptor Field 4 (RDES4)

Bit	Description
31:28	Reserved †
27:26	Layer 3 and Layer 4 Filter Number Matched These bits indicate the number of the Layer 3 and Layer 4 Filter that matched the received frame. • 00: Filter 0 • 01: Filter 1 • 10: Filter 2 • 11: Filter 3 This field is valid only when Bit 24 or Bit 25 is set. When more than one filter matches, these bits give only the lowest filter number. †
25	Layer 4 Filter Match When set, this bit indicates that the received frame matches one of the enabled Layer 4 Port Number fields. This status is given only when one of the following conditions is true: • Layer 3 fields are not enabled and all enabled Layer 4 fields match. • All enabled Layer 3 and Layer 4 filter fields match. When more than one filter matches, this bit gives the layer 4 filter status of filter indicated by Bits [27:26]. †
24	Layer 3 Filter Match When set, this bit indicates that the received frame matches one of the enabled Layer 3 IP Address fields. This status is given only when one of the following conditions is true: • All enabled Layer 3 fields match and all enabled Layer 4 fields are bypassed. • All enabled filter fields match. When more than one filter matches, this bit gives the layer 3 filter status of the filter indicated by Bits [27:26]. †
23:15	Reserved
14	Timestamp Dropped

continued...

Bit	Description
	When set, this bit indicates that the timestamp was captured for this frame but got dropped in the MTL RX FIFO buffer because of overflow.
13	PTP Version When set, this bit indicates that the received PTP message has the IEEE 1588 version 2 format. When clear, it has the version 1 format.
12	PTP Frame Type When set, this bit indicates that the PTP message is sent directly over Ethernet. When this bit is not set and the message type is non-zero, it indicates that the PTP message is sent over UDP-IPv4 or UDP-IPv6. The information about IPv4 or IPv6 can be obtained from Bits 6 and 7.
11:8	Message Type These bits are encoded to give the type of the message received. • 0000: No PTP message received • 0001: SYNC (all clock types) • 0010: Follow_Up (all clock types) • 0011: Delay_Req (all clock types) • 0100: Delay_Resp (all clock types) • 0101: Pdelay_Req (in peer-to-peer transparent clock) • 0110: Pdelay_Resp (in peer-to-peer transparent clock) • 0111: Pdelay_Resp_Follow_Up (in peer-to-peer transparent clock) • 1000: Announce • 1001: Management • 1010: Signaling • 1011-1110: Reserved • 1111: PTP packet with Reserved message type
7	IPv6 Packet Received When set, this bit indicates that the received packet is an IPv6 packet. This bit is updated only when Bit 10 (IPC) of Register 0 (MAC Configuration Register) is set.
6	IPv4 Packet Received When set, this bit indicates that the received packet is an IPv4 packet. This bit is updated only when Bit 10 (IPC) of Register 0 (MAC Configuration Register) is set.
5	IP Checksum Bypassed When set, this bit indicates that the checksum offload engine is bypassed.
4	IP Payload Error When set, this bit indicates that the 16-bit IP payload checksum (that is, the TCP, UDP, or ICMP checksum) that the EMAC calculated does not match the corresponding checksum field in the received segment. It is also set when the TCP, UDP, or ICMP segment length does not match the payload length value in the IP Header field. This bit is valid when either Bit 7 or Bit 6 is set.
3	IP Header Error When set, this bit indicates that either the 16-bit IPv4 header checksum calculated by the EMAC does not match the received checksum bytes, or the IP datagram version is not consistent with the Ethernet Type value. This bit is valid when either Bit 7 or Bit 6 is set.
2:0	IP Payload Type These bits indicate the type of payload encapsulated in the IP datagram processed by the receive Checksum Offload Engine (COE). The COE also sets these bits to 0 if it does not process the IP datagram's payload due to an IP header error or fragmented IP. • 0x0: Unknown or did not process IP payload • 0x1: UDP • 0x2: TCP • 0x3: ICMP • 0x4–0x7: Reserved This bit is valid when either Bit 7 or Bit 6 is set.

Related Information

Receive Descriptor Field 0 (RDES0) on page 554

18.5.3.2.5. Receive Descriptor Fields (RDES6) and (RDES7)

Receive Descriptor Fields 6 and 7 (RDES6 and RDES7) contain the snapshot of the timestamp. The availability of the snapshot of the timestamp in RDES6 and RDES7 is indicated by Bit 7 in the RDES0 descriptor.

Related Information

Receive Descriptor Field 0 (RDES0) on page 554

Receive Descriptor Field 6 (RDES6)

Table 196. Receive Descriptor Field 6 (RDES6)

Bit	Description
31:0	RTSL: Receive Frame Timestamp Low This field is updated by the DMA with the least significant 32 bits of the timestamp captured for the corresponding receive frame. This field is updated by the DMA only for the last descriptor of the receive frame, which is indicated by Last Descriptor status bit (RDES0[8]) in RDES0.

Related Information

Receive Descriptor Field 0 (RDES0) on page 554

Receive Descriptor Field 7 (RDES7)

Table 197. Receive Descriptor Field 7 (RDES7)

Bit	Description
31:0	RTSH: Receive Frame Timestamp High This field is updated by the DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field is updated by the DMA only for the last descriptor of the receive frame, which is indicated by Last Descriptor status bit (RDES0[8]) in RDES0.

Related Information

Receive Descriptor Field 0 (RDES0) on page 554

18.5.4. IEEE 1588-2002 Timestamps

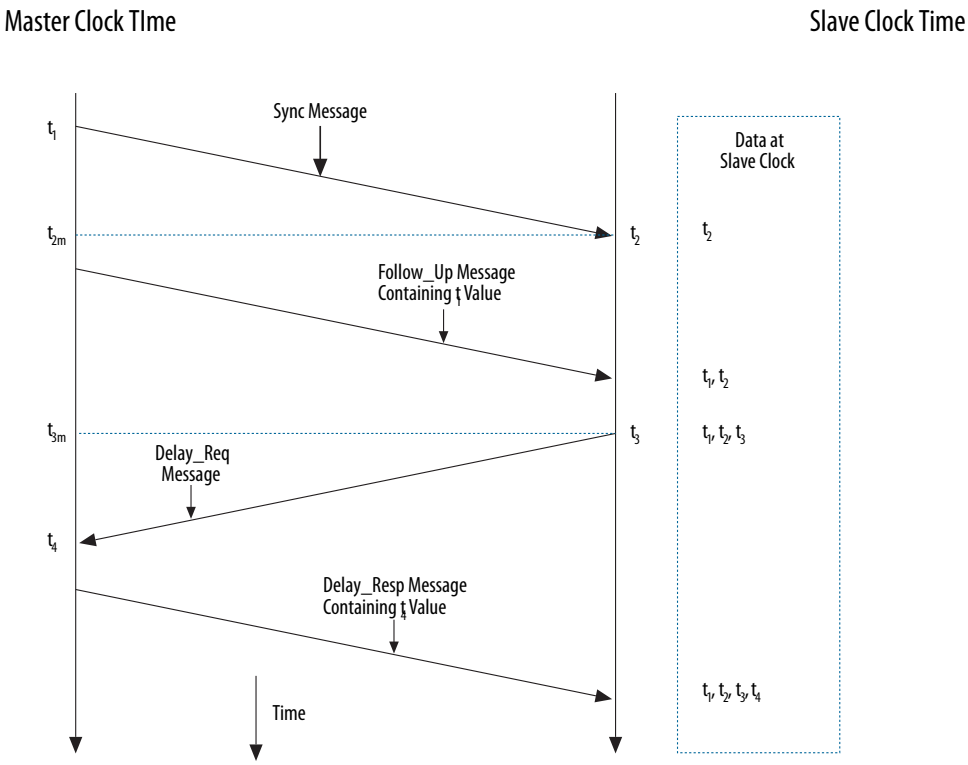
The IEEE 1588-2002 standard defines the Precision Time Protocol (PTP) that enables precise synchronization of clocks in a distributed network of devices. The PTP applies to systems communicating by local area networks supporting multicast messaging. This protocol enables heterogeneous systems that include clocks of varying inherent precision, resolution, and stability to synchronize. It is frequently used in automation systems where a collection of communicating machines such as robots must be synchronized and hence operate over a common time base.(†)

(†) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

The PTP is transported over UDP/IP. The system or network is classified into Master and Slave nodes for distributing the timing and clock information.†

The following figure shows the process that PTP uses for synchronizing a slave node to a master node by exchanging PTP messages.

Figure 99. Networked Time Synchronization



The PTP uses the following process for synchronizing a slave node to a master node by exchanging the PTP messages:

1. The master broadcasts the PTP Sync messages to all its nodes. The Sync message contains the master's reference time information. The time at which this message leaves the master's system is t1. This time must be captured, for Ethernet ports, at the PHY interface.†
2. The slave receives the sync message and also captures the exact time, t2, using its timing reference.†
3. The master sends a follow_up message to the slave, which contains t1 information for later use.†
4. The slave sends a delay_req message to the master, noting the exact time, t3, at which this frame leaves the PHY interface.†

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

5. The master receives the message, capturing the exact time, t4, at which it enters its system.†
6. The master sends the t4 information to the slave in the delay_resp message.†
7. The slave uses the four values of t1, t2, t3, and t4 to synchronize its local timing reference to the master's timing reference.†

Most of the PTP implementation is done in the software above the UDP layer. However, the hardware support is required to capture the exact time when specific PTP packets enter or leave the Ethernet port at the PHY interface. This timing information must be captured and returned to the software for the proper implementation of PTP with high accuracy.†

The EMAC is intended to support IEEE 1588 operation in all modes with a resolution of 10 ns. When the three EMACs are operating in an IEEE 1588 environment, the MPU subsystem is responsible for maintaining synchronization between the time counters internal to the three MACs.

The IEEE 1588 interface to the FPGA allows the FPGA to provide a source for the `emac_ptp_ref_clk` input as well to allow it to monitor the pulse per second output from each EMAC controller.

The EMAC component provides a hardware assisted implementation of the IEEE 1588 protocol. Hardware support is for timestamp maintenance. Timestamps are updated when receiving any frame on the PHY interface, and the receive descriptor is updated with this value. Timestamps are also updated when the SFD of a frame is transmitted and the transmit descriptor is updated accordingly.†

Note: You may use external time-stamp clock reference to accomplish timing. Use the `set_global_assignment -name ENABLE_HPS_INTERNAL_TIMING ON` to enable timing analysis.

Related Information

IEEE Standards Association
For details about the IEEE 1588-2002 standard, refer to IEEE Standard 1588-2002 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, available on the IEEE Standards Association website.†

18.5.4.1. Reference Timing Source

To get a snapshot of the time, the EMAC takes the reference clock input and uses it to generate the reference time (64-bit) internally and capture timestamps.†

18.5.4.2. System Time Register Module

The 64-bit time is maintained in this module and updated using the input reference clock, `clk_ptp_ref`, which can be the `emac_ptp_clk` from the HPS or the `f2s_ptp_ref_clk` from the FPGA. The `emac_ptp_clk` in the HPS is a derivative of the `osc1_clk` and is configured in the clock manager. This input reference clock is the source for taking snapshots (timestamps) of Ethernet frames being transmitted or received at the PHY interface.

Note: The `osc1_clk` signal is sourced from the external oscillator input, `HPS_CLK1`.

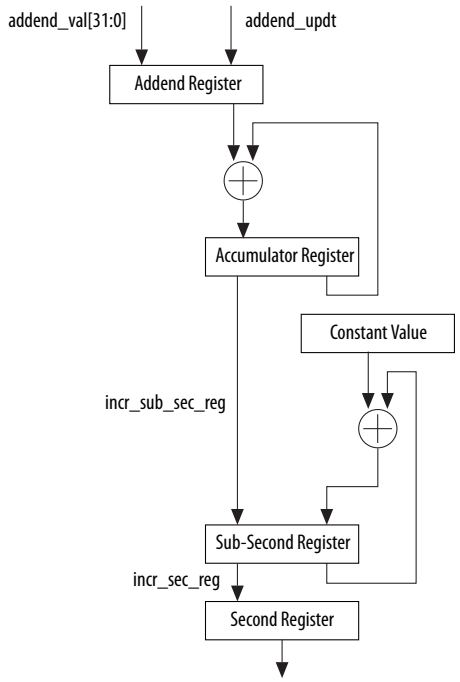
The system time counter can be initialized or corrected using the coarse correction method. In this method, the initial value or the offset value is written to the Timestamp Update register. For initialization, each EMAC's system time counter is written with the value in the Timestamp Update registers, while for system time correction, the offset value is added to or subtracted from the system time.

With the fine correction method, a slave clock's frequency drift with respect to the master clock is corrected over a period of time instead of in one clock, as in coarse correction. This protocol helps maintain linear time and does not introduce drastic changes (or a large jitter) in the reference time between PTP sync message intervals.†

With this method, an accumulator sums up the contents of the Timestamp_Addend register, as shown in the figure below. The arithmetic carry that the accumulator generates is used as a pulse to increment the system time counter. The accumulator and the addend are 32-bit registers. Here, the accumulator acts as a high-precision frequency multiplier or divider.

Note: You must connect a PTP clock with a frequency higher than the frequency required for the specified accuracy.†

Figure 100. Algorithm for System Time Update Using Fine Method



The System Time Update logic requires a 50-MHz clock frequency to achieve 20-ns accuracy. The frequency division ratio (FreqDivisionRatio) is the ratio of the reference clock frequency to the required clock frequency. Hence, if the reference clock

(`clk_ptp_ref_i`) is for example, 66 MHz, this ratio is calculated as 66 MHz / 50 MHz = 1.32. Hence, the default addend value to program in the register is 2^{32} / 1.32, 0xC1F07C1F.

If the reference clock drifts lower, to 65 MHz for example, the ratio is 65 / 50, or 1.3 and the value to set in the addend register is 2^{32} / 1.30, or 0xC4EC4EC4. If the clock drifts higher, to 67 MHz for example, the addend register must be set to 0xBF0B7672. When the clock drift is nil, the default addend value of 0xC1F07C1F (2^{32} / 1.32) must be programmed.†

In the above figure, the constant value used to accumulate the sub-second register is decimal 43, which achieves an accuracy of 20 ns in the system time (in other words, it is incremented in 20-ns steps).

The software must calculate the drift in frequency based on the Sync messages and update the Addend register accordingly.†

Initially, the slave clock is set with `FreqCompensationValue0` in the Addend register. This value is as follows:†

$$FreqCompensationValue_0 = 2^{32} / FreqDivisionRatio^{\dagger}$$

If MasterToSlaveDelay is initially assumed to be the same for consecutive sync messages, the algorithm described below must be applied. After a few sync cycles, frequency lock occurs. The slave clock can then determine a precise MasterToSlaveDelay value and re-synchronize with the master using the new value.†

The algorithm is as follows:†

- At time $MasterSyncTime_n$ the master sends the slave clock a sync message. The slave receives this message when its local clock is $SlaveClockTime_n$ and computes $MasterClockTime_n$ as:†

  $$MasterClockTime_n = MasterSyncTime_n + MasterToSlaveDelay_n^{\dagger}$$

- The master clock count for current sync cycle, $MasterClockCount_n$ is given by:†

  $$MasterClockCount_n = MasterClockTime_n - MasterClockTime_{n-1}$$

  (assuming that MasterToSlaveDelay is the same for sync cycles n and n – 1)†

- The slave clock count for current sync cycle, $SlaveClockCount_n$ is given by:†

  $$SlaveClockCount_n = SlaveClockTime_n - SlaveClockTime_{n-1}^{\dagger}$$

- The difference between master and slave clock counts for current sync cycle, $ClockDiffCount_n$ is given by:†

  $$ClockDiffCount_n = MasterClockCount_n - SlaveClockCount_n^{\dagger}$$

- The frequency-scaling factor for the slave clock, $FreqScaleFactor_n$ is given by:†

  $$FreqScaleFactor_n = (MasterClockCount_n + ClockDiffCount_n) / SlaveClockCount_n^{\dagger}$$

- The frequency compensation value for Addend register, $FreqCompensationValue_n$ is given by: †

  $$FreqCompensationValue_n = FreqScaleFactor_n \times FreqCompensationValue_{n-1} - 1^{\dagger}$$

In theory, this algorithm achieves lock in one Sync cycle; however, it may take several cycles, because of changing network propagation delays and operating conditions.†

This algorithm is self-correcting: if for any reason the slave clock is initially set to a value from the master that is incorrect, the algorithm corrects it at the cost of more Sync cycles.†

18.5.4.3. Transmit Path Functions

The MAC captures a timestamp when the start-of-frame data (SFD) is sent on the PHY interface. You can control the frames for which timestamps are captured on a per frame basis. In other words, each transmit frame can be marked to indicate whether a timestamp should be captured for that frame. †

You can use the control bits in the transmit descriptor to indicate whether a timestamp should be capture for a frame. The MAC returns the timestamp to the software inside the corresponding transmit descriptor, thus connecting the timestamp automatically to the specific PTP frame. The 64-bit timestamp information is written to the TDES2 and TDES3 fields. T†

18.5.4.4. Receive Path Functions

The MAC captures the timestamp of all frames received on the PHY interface. The DMA returns the timestamp to the software in the corresponding receive descriptor. The timestamp is written only to the last receive descriptor.†

18.5.4.5. Timestamp Error Margin

According to the IEEE1588 specifications, a timestamp must be captured at the SFD of the transmitted and received frames at the PHY interface. Because the PHY interface receive and transmit clocks are not synchronous to the reference timestamp clock (`clk_ptp_ref`) a small amount of drift is introduced when a timestamp is moved between asynchronous clock domains. In the transmit path, the captured and reported timestamp has a maximum error margin of two PTP clocks, meaning that the captured timestamp has a reference timing source value that is occurred within two clocks after the SFD is transmitted on the PHY interface.

Similarly, in the receive path, the error margin is three PHY interface clocks, plus up to two PTP clocks. You can ignore the error margin due to the PHY interface clock by assuming that this constant delay is present in the system (or link) before the SFD data reaches the PHY interface of the MAC.†

18.5.4.6. Frequency Range of Reference Timing Clock

The timestamp information is transferred across asynchronous clock domains, from the EMAC clock domain to the FPGA clock domain. Therefore, a minimum delay is required between two consecutive timestamp captures. This delay is four PHY interface clock cycles and three PTP clock cycles. If the delay between two timestamp captures is less than this amount, the MAC does not take a timestamp snapshot for the second frame.

The maximum PTP clock frequency is limited by the maximum resolution of the reference time (20 ns resulting in 50 MHz) and the timing constraints achievable for logic operating on the PTP clock. In addition, the resolution, or granularity, of the reference time source determines the accuracy of the synchronization. Therefore, a higher PTP clock frequency gives better system performance.†

The minimum PTP clock frequency depends on the time required between two consecutive SFD bytes. Because the PHY interface clock frequency is fixed by the IEEE 1588 specification, the minimum PTP clock frequency required for proper operation depends on the operating mode and operating speed of the MAC.†

Table 198. Minimum PTP Clock Frequency Example

Mode	Minimum Gap Between Two SFDs	Minimum PTP Frequency
100-Mbps full-duplex operation	168 MII clocks (128 clocks for a 64-byte frame + 24 clocks of min IFG + 16 clocks of preamble)	(3 * PTP) + (4 * MII) <= 168 * MII, that is, ~0.5 MHz (168 – 4) * 40 ns ÷ 3 = 2180 ns period)
1000-Mbps half duplex operation	24 GMII clocks (4 for a jam pattern sent just after SFD because of collision + 12 IFG + 8 preamble)	(3 * PTP) + 4 * GMII <= 24 * GMII, that is, 18.75 MHz

Related Information

IEEE Standards Association
For details about jam patterns, refer to the *IEEE Std 802.3 2008 Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, available on the IEEE Standards Association website.

18.5.5. IEEE 1588-2008 Advanced Timestamps

In addition to the basic timestamp features mentioned in IEEE 1588-2002 Timestamps, the EMAC supports the following advanced timestamp features defined in the IEEE 1588-2008 standard.†

- Supports the IEEE 1588-2008 (version 2) timestamp format. †
- Provides an option to take a timestamp of all frames or only PTP-type frames. †
- Provides an option to take a timestamp of event messages only. †
- Provides an option to take the timestamp based on the clock type: ordinary, boundary, end-to-end, or peer-to-peer. †
- Provides an option to configure the EMAC to be a master or slave for ordinary and boundary clock. †
- Identifies the PTP message type, version, and PTP payload in frames sent directly over Ethernet and sends the status. †
- Provides an option to measure sub-second time in digital or binary format. †

Related Information

IEEE Standards Association
For more information about advanced timestamp features, refer to the *IEEE Standard 1588 - 2008 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control System, available on the IEEE Standards Association website.*

18.5.5.1. Peer-to-Peer PTP Transparent Clock (P2P TC) Message Support

The IEEE 1588-2008 version supports Peer-to-Peer PTP (Pdelay) messages in addition to SYNC, Delay Request, Follow-up, and Delay Response messages.†

18.5.5.2. Clock Types

The EMAC supports the following clock types defined in the IEEE 1588-2008 standard:

- Ordinary clock†
- Boundary clock†
- End-to-End transparent clock†
- Peer-to-Peer transparent clock†

18.5.5.2.1. Ordinary Clock

The ordinary clock in a domain supports a single copy of the protocol. The ordinary clock has a single PTP state and a single physical port. In typical industrial automation applications, an ordinary clock is associated with an application device such as a sensor or an actuator. In telecom applications, the ordinary clock can be associated with a timing demarcation device. †

The ordinary clock can be a grandmaster or a slave clock. It supports the following features:†

- Sends and receives PTP messages. The timestamp snapshot can be controlled as described in the `Timestamp Control (gmacgrp_timestamp_control)` register.†
- Maintains the data sets such as timestamp values.†

The table below shows the messages for which you can take the timestamp snapshot on the receive side for Master and slave nodes. For an ordinary clock, you can take the snapshot of either of the following PTP message types: version 1 or version 2. You cannot take the snapshots for both PTP message types. You can take the snapshot by setting the control bit (`tsver2ena`) and selecting the snapshot mode in the `Timestamp Control (gmacgrp_timestamp_control)` register.†

Table 199. Ordinary Clock: PTP Messages for Snapshot†

Master	Slave
Delay_Req	SYNC

18.5.5.2.2. Boundary Clock

The boundary clock typically has several physical ports communicating with the network. The messages related to synchronization, master-slave hierarchy, and signaling terminate in the protocol engine of the boundary clock and are not forwarded. The PTP message type status given by the MAC helps you to identify the type of message and take appropriate action. The boundary clock is similar to the ordinary clock except for the following features:†

- The clock data sets are common to all ports of the boundary clock .†
- The local clock is common to all ports of the boundary clock. Therefore, the features of the ordinary clock are also applicable to the boundary clock.†

18.5.5.2.3. End-to-End Transparent Clock

The end-to-end transparent clock supports the end-to-end delay measurement mechanism between slave clocks and the master clock. The end-to-end transparent clock forwards all messages like normal bridge, router, or repeater. The residence time of a PTP packet is the time taken by the PTP packet from the ingress port to the egress port.†

The residence time of a SYNC packet inside the end-to-end transparent clock is updated in the correction field of the associated Follow_Up PTP packet before it is transmitted. Similarly, the residence time of a Delay_Req packet inside the end-to-end transparent clock is updated in the correction field of the associated Delay_Resp PTP packet before it is transmitted. Therefore, the snapshot needs to be taken at both ingress and egress ports only for PTP messages SYNC or Delay_req. You can take the snapshot by setting the snapshot select bits (SNAPTYPSEL) to b'10 in the `Timestamp Control (gmacgrp_timestamp_control)` register.†

The `snaptypsel` bits, along with bits 15 and 14 in the `Timestamp Control` register, decide the set of PTP packet types for which a snapshot needs to be taken. The encoding is shown in the table below:†

Table 200. Timestamp Snapshot Dependency on Register Bits†

X is defined as a "don't care" in the table.

snaptypsel (bits[17:16])	tsmstrena (bit 15)	tsevntena (bit 14)	PTP Messages
0x0	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp
0x0	0	1	SYNC
0x0	1	1	Delay_Req
0x1	X	0	SYNC, Follow_Up, Delay_Req, Delay_Resp, Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up
0x1	0	1	SYNC, Pdelay_Req, Pdelay_Resp
0x1	1	1	Delay_Req, Pdelay_Req, Pdelay_Resp
0x2	X	X	SYNC, Delay_Req
0x3	X	X	Pdelay_Req, Pdelay_Resp

18.5.5.2.4. Peer-to-Peer Transparent Clock

The peer-to-peer transparent clock differs from the end-to-end transparent clock in the way it corrects and handles the PTP timing messages. In all other aspects, it is identical to the end-to-end transparent clock.†

In the peer-to-peer transparent clock, the computation of the link delay is based on an exchange of Pdelay_Req, Pdelay_Resp, and Pdelay_Resp_Follow_Up messages with the link peer. The residence time of the Pdelay_Req and the associated Pdelay_resp packets is added and inserted into the correction field of the associated Pdely_Resp_Followup packet.†

Therefore, support for taking snapshot for the event messages related to Pdelay is added as shown in the table below.†

Table 201. Peer-to-Peer Transparent Clock: PTP Messages for Snapshot†

PTP Messages
SYNC
Pdelay_Req
Pdelay_Resp

You can take the snapshot by setting the snapshot select bits (`snaptypsel`) to b'11 in the `Timestamp Control` register.†

18.5.5.3. Reference Timing Source

The EMAC supports the following reference timing source features defined in the IEEE 1588-2008 standard:

- 48-bit seconds field†
- Fixed pulse-per-second output†
- Flexible pulse-per-second output†
- Auxiliary snapshots (timestamps) with external events

18.5.5.4. Transmit Path Functions

The advanced timestamp feature is supported through the descriptors format.

18.5.5.5. Receive Path Functions

The MAC processes the received frames to identify valid PTP frames. You can control the snapshot of the time to be sent to the application, by using the following options:†

- Enable timestamp for all frames.†
- Enable timestamp for IEEE 1588 version 2 or version 1 timestamp.†
- Enable timestamp for PTP frames transmitted directly over Ethernet or UDP/IP Ethernet.†
- Enable timestamp snapshot for the received frame for IPv4 or IPv6.†
- Enable timestamp snapshot for EVENT messages (SYNC, DELAY_REQ, PDELAY_REQ, or PDELAY_RESP) only.†
- Enable the node to be a master or slave and select the timestamp type to control the type of messages for which timestamps are taken.†

The DMA returns the timestamp to the software inside the corresponding transmit or receive descriptor.

18.5.5.6. Auxiliary Snapshot

The auxiliary snapshot feature allows you to store a snapshot (timestamp) of the system time based on an external event. The event is considered to be the rising edge of the sideband signal `ptp_aux_ts_trig_i` from the FPGA. One auxiliary snapshot input is available. The depth of the auxiliary snapshot FIFO buffer is 16.

The timestamps taken for any input are stored in a common FIFO buffer. The host can read Register 458 (Timestamp Status Register) to know which input's timestamp is available for reading at the top of this FIFO buffer.

Only 64-bits of the timestamp are stored in the FIFO. You can read the upper 16-bits of seconds from Register 457 (System Time - Higher Word Seconds Register) when it is present. When a snapshot is stored, the MAC indicates this to the host with an interrupt. The value of the snapshot is read through a FIFO register access. If the FIFO becomes full and an external trigger to take the snapshot is asserted, then a snapshot trigger-missed status (ATSSTM) is set in Register 458 (Timestamp Status Register). This indicates that the latest auxiliary snapshot of the timestamp was not stored in the FIFO. The latest snapshot is not written to the FIFO when it is full. When a host reads the 64-bit timestamp from the FIFO, the space becomes available to store the next snapshot. You can clear a FIFO by setting Bit 19 (ATSFC) in Register 448 (Timestamp Control Register). When multiple snapshots are present in the FIFO, the count is indicated in Bits [27:25], ATSNS, of Register 458 (Timestamp Status Register). †

18.5.6. IEEE 802.3az Energy Efficient Ethernet

Energy Efficient Ethernet (EEE) standardized by IEEE 802.3-az, version D2.0 is supported by the EMAC. It is supported by the MAC operating in 10/100/1000 Mbps rates. EEE is only supported when the EMAC is configured to operate with the RGMII PHY interface operating in full-duplex mode. It cannot be used in half-duplex mode.

EEE enables the MAC to operate in Low-Power Idle (LPI) mode. Either end point of an Ethernet link can disable functionality to save power during periods of low link utilization. The MAC controls whether the system should enter or exit LPI mode and communicates this information to the PHY.†

Related Information

IEEE 802.3 Ethernet Working Group
For details about the *IEEE 802.3az Energy Efficient Ethernet standard*, refer to the IEEE 802.3 Ethernet Working Group website.†

18.5.6.1. LPI Timers

Two timers internal to the EMAC are associated with LPI mode:

- LPI Link Status (LS) Timer†
- LPI Time Wait (TW) Timer†

The LPI LS timer counts, in ms, the time expired since the link status has come up. This timer is cleared every time the link goes down and is incremented when the link is up again and the terminal count as programmed by the software is reached. The PHY interface does not assert the LPI pattern unless the terminal count is reached. This protocol ensures a minimum time for which no LPI pattern is asserted after a link is established with the remote station. This period is defined as one second in the IEEE standard 802.3-az, version D2.0. The LPI LS timer is 10 bits wide, so the software can program up to 1023 ms.†

The LPI TW timer counts, in µs, the time expired since the deassertion of LPI. The terminal count of the timer is the value of resolved transmit TW that is the auto-negotiated time after which the MAC can resume the normal transmit operation. The LPI TW timer is 16 bits wide, so the software can program up to 65535 µs.†

The EMAC generates the LPI interrupt when the transmit or receive channel enters or exits the LPI state.†

18.5.7. Checksum Offload

Communication protocols such as TCP and UDP implement checksum fields, which help determine the integrity of data transmitted over a network. Because the most widespread use of Ethernet is to encapsulate TCP and UDP over IP datagrams, the EMAC has a Checksum Offload Engine (COE) to support checksum calculation and insertion in the transmit path, and error detection in the receive path. Supported offloading types:

- Transmit IP header checksum†
- Transmit TCP/UDP/ICMP checksum†
- Receive IP header checksum†
- Receive full checksum†

18.5.8. Frame Filtering

The EMAC implements the following types of filtering for receive frames.

18.5.8.1. Source Address or Destination Address Filtering

The Address Filtering Module checks the destination and source address field of each incoming packet.†

18.5.8.1.1. Unicast Destination Address Filter

Up to 128 MAC addresses for unicast perfect filtering are supported. The filter compares all 48 bits of the received unicast address with the programmed MAC address for any match. Default MacAddr0 is always enabled, other addresses MacAddr1–MacAddr127 are selected with an individual enable bit. For MacAddr1–MacAddr31 addresses, you can mask each byte during comparison with the corresponding received DA byte. This enables group address filtering for the DA. The MacAddr32-MacAddr127 addresses do not have mask control and all six bytes of the MAC address are compared with the received six bytes of DA.†

In hash filtering mode, the filter performs imperfect filtering for unicast addresses using a 256-bit hash table. It uses the upper ten bits of the CRC of the received destination address to index the content of the hash table. A value of 0 selects Bit 0 of the selected register, and a value of 111111 binary selects Bit 63 of the Hash Table register. If the corresponding bit is set to one, the unicast frame is said to have passed the hash filter; otherwise, the frame has failed the hash filter.†

18.5.8.1.2. Multicast Destination Address Filter

The MAC can be programmed to pass all multicast frames. In Perfect Filtering mode, the multicast address is compared with the programmed MAC Destination Address registers (1–31). Group address filtering is also supported. In hash filtering mode, the filter performs imperfect filtering using a 256-bit hash table. For hash filtering, it uses the upper ten bits of the CRC of the received multicast address to index the contents of the hash table. A value of 0 selects Bit 0 of the selected register and a value of

111111 binary selects Bit 63 of the Hash Table register. If the corresponding bit is set to one, then the multicast frame is said to have passed the hash filter; otherwise, the frame has failed the hash filter.†

18.5.8.1.3. Hash or Perfect Address Filter

The filter can be configured to pass a frame when its DA matches either the hash filter or the Perfect filter. This configuration applies to both unicast and multicast frames.†

18.5.8.1.4. Broadcast Address Filter

The filter does not filter any broadcast frames in the default mode. However, if the MAC is programmed to reject all broadcast frames, the filter drops any broadcast frame.†

18.5.8.1.5. Unicast Source Address Filter

The MAC can also perform a perfect filtering based on the source address field of the received frames. Group filtering with SA is also supported. You can filter a group of addresses by masking one or more bytes of the address.†

18.5.8.1.6. Inverse Filtering Operation (Invert the Filter Match Result at Final Output)

For both Destination and Source address filtering, there is an option to invert the filter-match result at the final output. The result of the unicast or multicast destination address filter is inverted in this mode.†

18.5.8.1.7. Destination and Source Address Filtering Summary

The tables below summarize the destination and source address filtering based on the type of frames received and the configuration of bits within the Mac_Frame_Filter register.†

Table 202. Destination Address Filtering†

Note: The "X" in the table represents a "don't care" term.

Frame Type	PR	HPF	HUC	DAIF	HMC	PM	DBF	Destination Address Filter Operation
Broadcast	1	X	X	X	X	X	X	Pass
	0	X	X	X	X	X	0	Pass
	0	X	X	X	X	X	1	Fail
Unicast	1	X	X	X	X	X	X	Pass all frames
	0	X	0	0	X	X	X	Pass on Perfect/Group filter match
	0	X	0	1	X	X	X	Fail on Perfect/Group filter match
	0	0	1	0	X	X	X	Pass on Hash filter match
	0	0	1	1	X	X	X	Fail on Hash filter match
	0	1	1	0	X	X	X	Pass on Hash or Perfect/Group filter match
	0	1	1	1	X	X	X	Fail on Hash or Perfect/Group filter match

continued...

Frame Type	PR	HPF	HUC	DAIF	HMC	PM	DBF	Destination Address Filter Operation
Multicast	1	X	X	X	X	X	X	Pass all frames
	X	X	X	X	X	1	X	Pass all frames
	0	X	X	0	0	0	X	Pass on Perfect/Group filter match and drop Pause frames if PCF= 0X
	0	0	X	0	1	0	X	Pass on Hash filter match and drop Pause frames if PCF = 0X
	0	1	X	0	1	0	X	Pass on Hash or Perfect/Group filter match and drop Pause frames if PCF = 0X
	0	X	X	1	0	0	X	Fail on Perfect/Group filter match and drop Pause frames if PCF=0X
	0	0	X	1	1	0	X	Fail on Hash filter match and drop Paus frames if PCF = 0X
	0	1	X	1	1	0	X	Fail Hash on Perfect/Group filter match and drop Pause frames if PCF = 0X

Table 203. Source Address Filtering†

Frame Type	PR	SAIF	SBF	Source Address Filter Operation
Unicast	1	X	X	Pass all frames
	0	0	0	Pass status on Perfect or Group filter match but do not drop frames that fail.
	0	1	0	Fail on Perfect or Group filter match but do not drop frame
	0	0	1	Pass on Perfect or Group filter match and drop frames that fail
	0	1	1	Fail on Perfect or Group filter match and drop frames that fail

18.5.8.2. VLAN Filtering

The EMAC supports the two kinds of VLAN filtering:

- VLAN tag-based filtering†
- VLAN hash filtering†

18.5.8.2.1. VLAN Tag-Based Filtering

In the VLAN tag-based frame filtering, the MAC compares the VLAN tag of the received frame and provides the VLAN frame status to the application. Based on the programmed mode, the MAC compares the lower 12 bits or all 16 bits of the received VLAN tag to determine the perfect match. If VLAN tag filtering is enabled, the MAC forwards the VLAN-tagged frames along with VLAN tag match status and drops the VLAN frames that do not match. You can also enable the inverse matching for VLAN frames. In addition, you can enable matching of SVLAN tagged frames along with the default Customer Virtual Local Area Network (C-VLAN) tagged frames.†

18.5.8.2.2. VLAN Hash Filtering with a 16-Bit Hash Table

The MAC provides VLAN hash filtering with a 16-bit hash table. The MAC also supports the inverse matching of the VLAN frames. In inverse matching mode, when the VLAN tag of a frame matches the perfect or hash filter, the packet should be dropped. If the VLAN perfect and VLAN hash match are enabled, a frame is considered as matched if either the VLAN hash or the VLAN perfect filter matches. When inverse match is set, a packet is forwarded only when both perfect and hash filters indicate mismatch.†

18.5.8.3. Layer 3 and Layer 4 Filters

Layer 3 filtering refers to source address and destination address filtering. Layer 4 filtering refers to source port and destination port filtering. The frames are filtered in the following ways:†

- Matched frames†
- Unmatched frames†
- Non-TCP or UDP IP frames†

18.5.8.3.1. Matched Frames

The MAC forwards the frames, which match all enabled fields, to the application along with the status. The MAC gives the matched field status only if one of the following conditions is true:†

- All enabled Layer 3 and Layer 4 fields match.†
- At least one of the enabled field matches and other fields are bypassed or disabled.†

Using the CSR set, you can define up to four filters, identified as filter 0 through filter 3. When multiple Layer 3 and Layer 4 filters are enabled, any filter match is considered as a match. If more than one filter matches, the MAC provides status of the lowest filter with filter 0 being the lowest and filter 3 being the highest. For example, if filter 0 and filter 1 match, the MAC gives the status corresponding to filter 0.†

18.5.8.3.2. Unmatched Frames

The MAC drops the frames that do not match any of the enabled fields. You can use the inverse match feature to block or drop a frame with specific TCP or UDP over IP fields and forward all other frames. You can configure the EMAC so that when a frame is dropped, it receives a partial frame with appropriate abort status or drops it completely.†

18.5.8.3.3. NonTCP or UDP IP Frames

By default, all non-TCP or UDP IP frames are bypassed from the Layer 3 and Layer 4 filters. You can optionally program the MAC to drop all non-TCP or UDP over IP frames.†

18.5.8.3.4. Layer 3 and Layer 4 Filters Register Set

The MAC implements a set of registers for Layer 3 and Layer 4 based frame filtering. In this register set, there is a control register for frame filtering and five address registers.†

You can configure the MAC to have up to four such independent set of registers.†

The registers available for programming are as follows:

- `gmacgrp_l3_l4_control0` through `gmacgrp_l3_l4_control3` registers: Layer and Layer 4 Control registers
- `gmacgrp_layer4_address0` through `gmacgrp_layer4_address3` registers: Layer 4 Address registers
- `gmacgrp_layer3_addr0_reg0` through `gmacgrp_layer3_addr0_reg3` registers: Layer 3 Address 0 registers
- `gmacgrp_layer3_addr1_reg0` through `gmacgrp_layer3_addr1_reg3` registers: Layer 3 Address 1 registers
- `gmacgrp_layer3_addr2_reg0` through `gmacgrp_layer3_addr2_reg3` registers: Layer 3 Address 2 registers
- `gmacgrp_layer3_addr3_reg0` through `gmacgrp_layer3_addr3_reg3` registers: Layer 3 Address 3 registers

Related Information

Ethernet MAC Address Map and Register Definitions on page 590

18.5.8.3.5. Layer 3 Filtering

The EMAC supports perfect matching or inverse matching for the IP Source Address and Destination Address. In addition, you can match the complete IP address or mask the lower bits. †

For IPv6 frames filtering, you can enable the last four data registers of a register set to contain the 128-bit IP Source Address or IP Destination Address. The IP Source or Destination Address should be programmed in the order defined in the IPv6 specification. The specification requires that you program the first byte of the received frame IP Source or Destination Address in the higher byte of the register. Subsequent registers should follow the same order.†

For IPv4 frames filtering, you can enable the second and third data registers of a register set to contain the 32-bit IP Source Address and IP Destination Address. The remaining two data registers are reserved. The IP Source and Destination Address should be programmed in the order defined in the IPv4 specification. The specification requires that you program the first byte of received frame IP Source and Destination Address in the higher byte of the respective register.†

18.5.8.3.6. Layer 4 Filtering

The EMAC supports perfect matching or inverse matching for TCP or UDP Source and Destination Port numbers. However, you can program only one type (TCP or UDP) at a time. The first data register contains the 16-bit Source and Destination Port numbers of TCP or UDP, that is, the lower 16 bits for Source Port number and higher 16 bits for Destination Port number. †

The TCP or UDP Source and Destination Port numbers should be programmed in the order defined in the TCP or UDP specification, that is, the first byte of TCP or UDP Source and Destination Port number in the received frame is in the higher byte of the register.†

18.5.9. Clocks and Resets

18.5.9.1. Clock Structure

The Ethernet Controller has four main clock domains.

- l4_mp_clk clock
- EMAC RX clock
- EMAC TX clock
- clk_ptp_ref

Figure 101. EMAC Clock Diagram

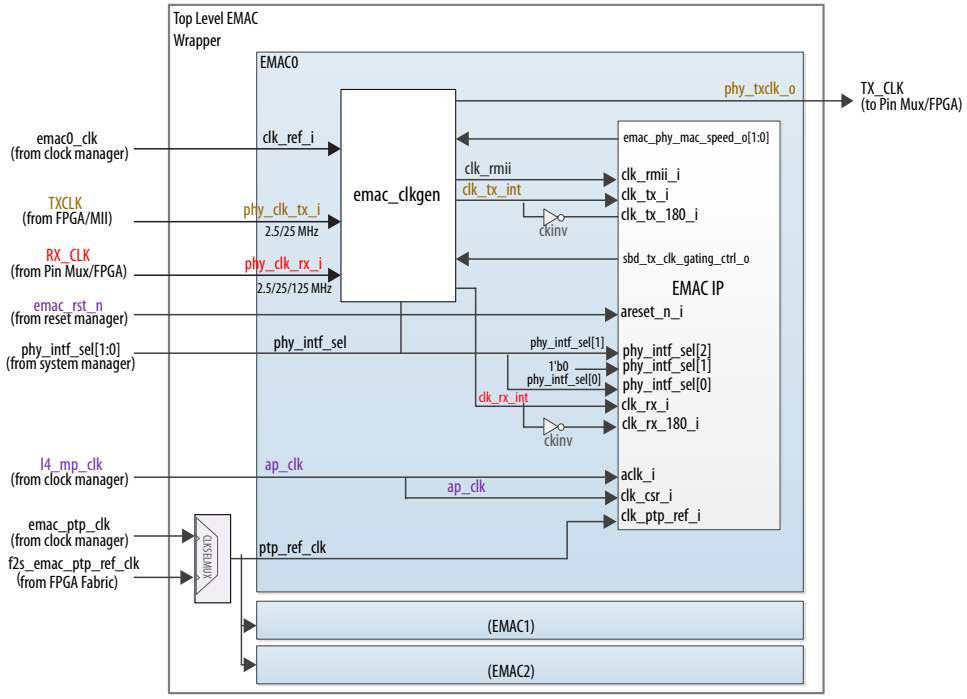
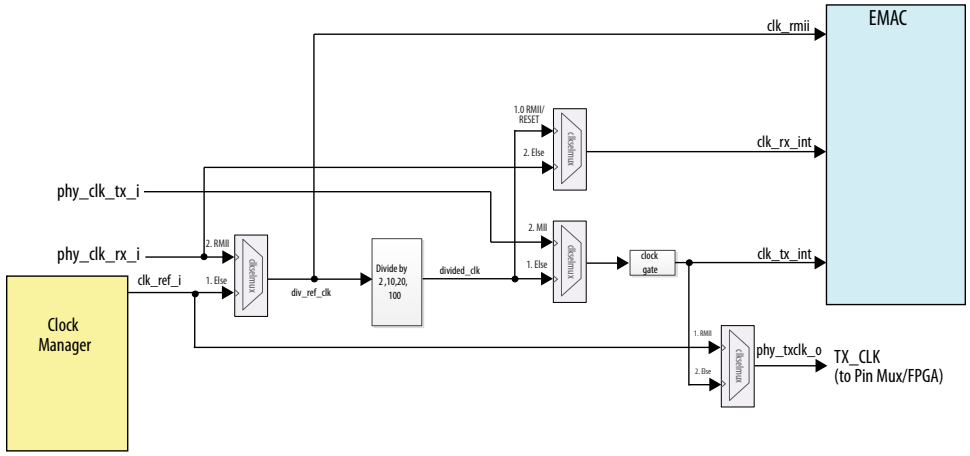


Figure 102. emac_clkgen Module



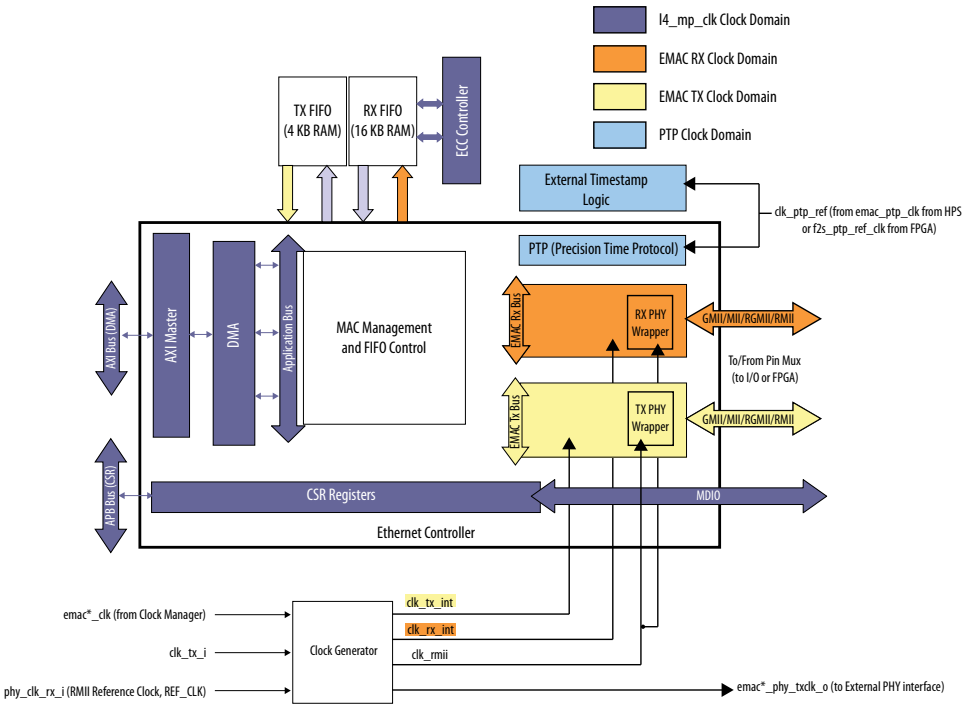
Depending on the interface, different clock domains are used:

- When the DMA master interface is used for EMAC packet transfers, the `l4_mp_clk` is used as a clock source for both the AXI bus and the CSR register interface. This clock domain is a fully synchronous.
- The RX and TX FIFO RAMs are driven by the `l4_mp_clk`.
- The MDIO interface's clock domain is a derivative of the CSR clock, which comes from l4_mp_clk. Typically MDC clock has a frequency between 1 to 2.5 MHz, however, faster MDC frequencies are supported in this design.
- The EMAC contains an RX datapath, TX datapath and timestamp interface that all run on separate clock domains.
  - The RX datapath is in the EMAC RX clock domain.
  - The TX datapath is in the EMAC TX clock domain.
  - The timestamp interface is in the clk_ptp_ref clock domain.

The timestamp clock domain provides the capability for EMAC0 to be a timestamp master with internal timestamp enabled and the other two EMACs to be timestamp slaves using the timestamp generated from EMAC0.

The diagram below summarizes the clock domains of the EMAC module:

Figure 103. EMAC Clock Domains



The following table summarizes the clock inputs and outputs to the EMAC.

Table 204. EMAC Module Clock Inputs and Outputs

Clock	Input/Output	Frequency	Source	Description
l4_mp_clk	Input	200 MHz	Clock Manager	Application clock for DMA bus interface, CSR interface and ECC FIFO RAMs.
clk_ptp_ref	Input	up to 100 MHz	Clock Manager or FPGA fabric	This signal is sourced by either the PTP reference clock from the Clock Manager or the FPGA fabric. The source can be selected through the ptp_clk_sel bit of the emac_global register in the System Manager module. When the bit is clear, the emac_ptp_clk is selected and when it is set, the f2s_ptp_ref_clk is selected.
emac*_clk	Input	Variable depending on divider value of programmed in Clock Manager.	Input from Clock Manager	This signal is configured in the Clock Manager module and can be enabled to drive the clk_tx_in and clk_rx int signals to the TX and RX clock domains.
clk_tx_i	Input	Used only in MII mode as a 25 or 2.5 MHz clock source at 100 Mbps and 10 Mbps, respectively.	Input from FPGA fabric I/O	This signal is used only in MII mode as a TX reference clock. *Note:* This clock must be able to perform glitch free switching between 2.5 and 25 MHz.
phy_clk_rx_i	Input	• GMII mode: 125 MHz • RGMII mode: 125, 25, or 2.5 MHz • MII mode: 25 or 2.5 MHz • RMII mode: 50 MHz	This clock input is driven to FPGA or by an HPS I/O input from an external PHY	For all modes except, RMII, this clock signal is the RX PHY input clock. For RMII mode, this input is a 50 MHz reference clock (REF_CLK) from the board or from phy_txclk_o that is divided down automatically to generate the datapath clocks, emac*_clk_rx_i and emac*_clk_tx_i signals. These datapath clocks are 2.5 MHz when operating in 10 Mbps mode and 25 MHz when operating in 100 Mbps mode.
phy_txclk_o	Output	125, 50, 25, or 2.5 MHz	From internal HPS clk_tx_int to HPS I/O or from FPGA fabric.	This signal is an TX output clock to the PHY. In RMII mode, this signal can provide the reference clock (50 MHz in 100M /10 Mbps).

18.5.9.2. Clock Gating for EEE

For the RGMII PHY interface, you can gate the transmit clock for Energy Efficient Ethernet (EEE) applications.

Related Information

Programming Guidelines for Energy Efficient Ethernet on page 586

18.5.9.3. Reset

The EMAC module accepts a single reset input, emac_rst_n, which is active low.

Note: In all modes, the EMAC core depends on the PHY clocks to be active for the internal EMAC clock sources to be valid.

18.5.9.3.1. Taking the Ethernet MAC Out of Reset

When a cold or warm reset is issued in the HPS, the Reset Manager resets the EMAC module and holds it in reset until software releases it.

After the MPU boots up, it can deassert the reset signal by clearing the appropriate bits in the Reset Manager's corresponding reset register. Before deasserting the reset signal, you must make sure the PHY interface type and all other corresponding EMAC settings in the System Manager have been configured. For details about reset registers, refer to the "Module Reset Signals" section in the *Reset Manager* chapter. For more information about EMAC configuration in the System Manager, refer to the "System Level EMAC Configuration Registers" section.

EMAC ECC RAM Reset

An EMAC ECC RAM reset asserts a reset to both the memory and the multiplexed EMAC bus interface clock, `ap_clk`. You should ensure that both the EMAC ECC RAM and the EMAC Module resets are deasserted before beginning transactions. Program the `emac*ocp` bits and the `emac*` bits in the `per0modrst` register of the Reset Manager to deassert reset in the EMAC's ECC RAM and the EMAC module, respectively.

Related Information

Module Reset Signals on page 140
For more information about reset registers refer to this section in the *Reset Manager* chapter.

18.5.10. Interrupts

Interrupts are generated as a result of specific events in the EMAC and external PHY device. The interrupt status register indicates all conditions which may trigger an interrupt and the interrupt enable register determines which interrupts can propagate.

18.6. Ethernet MAC Programming Model

The initialization and configuration of the EMAC and its interface is a multi-step process that includes system register programming in the System Manager and Clock Manager and configuration of clocks in multiple domains.

Note: When the EMAC interfaces to HPS I/O and register content is being transferred to a different clock domain after a write operation, no further writes should occur to the same location until the first write is updated. Otherwise, the second write operation does not get updated to the destination clock domain. Thus, the delay between two writes to the same register location should be at least 4 cycles of the destination clock (PHY receive clock, PHY transmit clock, or PTP clock).† If the CSR is accessed multiple times quickly, you must ensure that a minimum number of destination clock cycles have occurred between accesses.

Note: If the EMAC signals are routed through the FPGA fabric and it is assumed that the transmit clock supplied by the FPGA fabric switches within 6 transmit clock cycles, then the minimum time required between two write accesses to the same register is 10 transmit clock cycles.

18.6.1. System Level EMAC Configuration Registers

In addition to the registers in the Ethernet Controller, there are other system level registers in the Clock Manager, System Manager and Reset Manager that must be programmed in order to configure the EMAC and its interfaces.

The following table gives a summary of the important System Manager clock register bits that control operation of the EMAC. These register bits are static signals that must be set while the corresponding EMAC is in reset.

Table 205. System Manager Clock and Interface Settings

Register.Field	Description
`emac_global.ptp_clk_sel`	1588 PTP reference clock. This bit selects the source of the 1588 PTP reference clock. • 0x0= `emac_ptp_clk` (default from Clock Manager) • 0x1=`f2s_emac_ptp_ref_clk` (from FPGA fabric; in this case, the FPGA must be in usermode with an active reference clock)

`emac0.phy_intf_sel`
`emac1.phy_intf_sel`
`emac2.phy_intf_sel`
	PHY Interface Select. These two bits set the PHY mode.
	• 0x0= GMII or MII
	• 0x1= RGMII
	• 0x2= RMII
	• 0x3= RESET (default)

The following table summarizes the important System Manager configuration register bits. All of the fields, except the AXI cache settings, are assumed to be static and must be set before the EMAC is brought out of reset. If the FPGA interface is used, the FPGA must be in user mode and enabled with the appropriate clock signals active before the EMAC can be brought out of reset.

Table 206. System Manager Static Control Settings

Register.Field	Description

`fpgaintf_en_3.emac0`
`fpgaintf_en_3.emac1`
`fpgaintf_en_3.emac2`
	FPGA interface to EMAC disable. This field is used to disable signals from the FPGA to the EMAC modules that could potentially interfere with the EMAC's or FPGA's operation.
	• 0x0= Disable (default)
	• 0x1=Enable

`emac0.axi_disable`
`emac1.axi_disable`
`emac2.axi_disable`
	AXI Disable. Disables the AXI bus to EMAC.
	• 0x0= Enable (default)
	• 0x1= Disable

`emac0.awcache`
`emac1.awcache`
`emac2.awcache`
`emac0.arcache`
`emac1.arcache`
`emac2.arcache`
	EMAC AXI Master AxCACHE settings. It is recommended that these bits are set while the EMAC is idle or in reset.

`emac0.awprot`
`emac1.awprot`
`emac2.awprot`
`emac0.arprot`
`emac1.arprot`
	EMAC Master AxPROT settings. It is recommended that these bits are set while the EMAC is idle or in reset.

continued...

Register.Field	Description
emac2.arprot	
emac0.ptp_ref_sel emac1.ptp_ref_sel emac2.ptp_ref_sel	Internal/External Timestamp reference. This field selects if the timestamp reference is internally or externally generated. EMAC0 may be the master to generate the timestamp for EMAC1 and EMAC2. EMAC0 must be set to internal timestamp; EMAC1 and EMAC2 may be set either to internal or external. • 0x0= Internal (default) • 0x1= External

Various registers within the Clock Manager must also be configured in order for the EMAC controller to perform properly.

Table 207. Clock Manager Settings

Register.Field	Description
en.emacptpen	emac_ptp_clk output enable.
en.emac0en en.emac1en en.emac2en	Enables clock emac0_clk, emac1_clk and emac2_clk output. *Note:* There are corresponding ens and enr registers that allow the same fields to be set or cleared on a bit-by-bit basis.
bypass.emacptp	EMAC PTP clock bypass. This bit indicates if the emac_ptp_clk is bypassed to the input clock reference of the peripheral PLL. • 0x0= No bypass occurs • 0x1= emac_ptp_clk is bypassed to the input clock reference of the main PLL. *Note:* There are corresponding bypasss and bypassr registers that allow the same bits to be set or cleared on a bit-by-bit basis.
bypass.emaca bypass.emacb	Clock Bypass. This bit indicates whether emaca_free_clk or emacb_free_clk is bypassed to the input clock reference of the main PLL. • 0x0= No bypass occurs • 0x1= emac*_free_clk is bypassed to the input clock reference of the main PLL. *Note:* There are corresponding bypasss and bypassr registers that allow the same bits to be set or cleared on a bit-by-bit basis.
emacctl.emac0sel emacctl.emac1sel emacctl.emac2sel	EMAC clock source select. This bit selects the source for the emac*clk as either emaca_free_clk or emacb_free clk • 0x0= emaca_free_clk • 0x1=emacb_free_clk

18.6.2. EMAC FPGA Interface Initialization

To initialize the Ethernet controller to use the FPGA GMII/MII interface, specific software steps must be followed.

In general, the FPGA interface must be active in user mode with valid PHY clocks, the Ethernet Controller must be in a reset state during static configuration and the clock must be active and valid before the Ethernet Controller is brought out of reset.

1. After the HPS is released from cold or warm reset, reset the Ethernet Controller module by setting the appropriate emac* bit in the per0modrst register in the Reset Manager.

- — `emac reset bits [2:0]` – reset for EMAC0/1/2 port.
- — `emacptp reset bit [22]` – reset for EMAC PTP interface, only required when PTP timestamp interface is enabled.

2. Configure the EMAC Controller clock to 250 MHz by programming the appropriate registers in the Clock Manager.
3. Bring the Ethernet PHY out of reset to allow PHY to generate RX clocks and TX clocks.

   When using FPGA GMII/MII interface, you must have a stable RX clock (`emac_clk_rx_i`) and TX clock (`emac_clk_tx_i`) supply from PHY to EMAC before bringing EMAC out of reset.

   There are no registers to verify, but you can create the following custom logic block to cross check:

   - You can use Signal Tap to check, or create a simple counter block with the RX clock and TX clock as clock source to check if it runs.
4. If the PTP clock source is from the FPGA:
   - Release `per0modrest.emacptp` bit from reset.
   - Configure `emac_global` register to select `f2s_ptp_ref_clk` from the FPGA fabric.
   - Ensure that the FPGA `f2s_ptp_ref_clk` is active.
5. The soft GMII/MII adaptor must be loaded with active clocks propagating. The FPGA must be configured to user mode and a reset to the user soft FPGA IP may be required to propagate the PHY clocks to the HPS.
6. Once all clock sources are valid, apply the following clock settings:
   a. Program the `phy_intf_sel` field of the `emac*` register in the System Manager to 0x0 to select GMII/MII PHY interface.
   b. If the PTP clock source is from the FPGA, set the `ptp_clk_sel` bit to 0x1 in the `emac_global` register of the System Manager.
   c. Enable the Ethernet Controller FPGA interface by setting the `emac_*` bit in the `fpgaintf_en_3` register of the System Manager.
7. Configure all of the EMAC static settings if the user requires a different setting from the default value. These settings include the AxPROT[1:0] and AxCACHE signal values which are programmed in the `emac*` register of the System Manager.
8. Execute a register read back to confirm the clock and static configuration settings are valid.
9. After confirming the settings are valid, software can clear the `emac*` bit in the `per0modrst` register of the Reset Manager to bring the EMAC out of reset..

When these steps are completed, general Ethernet controller and DMA software initialization and configuration can continue.

Note: These same steps can be applied to convert the HPS GMII to an RMII or SGMII interface through the FPGA, except that in step 5 during FPGA configuration, you would load the appropriate soft adaptor for the interface and apply reset to it as well. The PHY interface select encoding would remain as 0x0. For the SGMII interface additional external transceiver logic would be required. Routing the Ethernet signals through the FPGA is useful for designs that are pin-limited in the HPS.

18.6.3. EMAC HPS Interface Initialization

To initialize the Ethernet controller to use the HPS interface, specific software steps must be followed including selecting the correct PHY interface through the System Manager.

In general, the Ethernet Controller must be in a reset state during static configuration and the clock must be active and valid before the Ethernet Controller is brought out of reset.

1. After the HPS is released from cold or warm reset, reset the Ethernet Controller module by setting the appropriate `emac*` bit in the `per0modrst` register in the Reset Manager.
2. Configure the EMAC Controller clock to 250 MHz by programming the appropriate registers in the Clock Manager.
3. Bring the Ethernet PHY out of reset to allow PHY to generate RX clocks.

   There are no registers to verify, but you can create the following custom logic block to cross check:

   - If the RX clock is routed through FPGA IO—you can use Signal Tap to check, or create a simple counter block with the RX clock as clock source to check if it runs.
   - If the RX clock is routed as HPS IO—you need to explore if the kernel application code is able to source through RX clock to check its status.
4. When all the clocks are valid, program the following clock settings:
   a. Program the `phy_intf_sel` field of the `emac*` register in the System Manager to 0x1 or 0x2 to select RGMII or RMII PHY interface.
   b. Disable the Ethernet Controller FPGA interface by clearing the `emac_*` bit in the `fpgaintf_en_3` register of the System Manager.
5. Configure all of the EMAC static settings if the user requires a different setting from the default value. These settings include the AxPROT[1:0] and AxCACHE signal values, which are programmed in the `emac*` register of the System Manager.
6. Execute a register read back to confirm the clock and static configuration settings are valid.
7. After confirming the settings are valid, software can clear the `emac*` bit in the `per0modrst` register of the Reset Manager to bring the EMAC out of reset..

When these steps are completed, general Ethernet controller and DMA software initialization and configuration can continue.

18.6.4. DMA Initialization

This section provides the instructions for initializing the DMA registers in the proper sequence. This initialization sequence can be done after the EMAC interface initialization has been completed. Perform the following steps to initialize the DMA:

1. Provide a software reset to reset all of the EMAC internal registers and logic. (DMA Register 0 (Bus Mode Register) – bit 0). †
2. Wait for the completion of the reset process (poll bit 0 of the DMA Register 0 (Bus Mode Register), which is only cleared after the reset operation is completed). †
3. Poll the bits of Register 11 (AXI Status) to confirm that all previously initiated (before software reset) or ongoing transactions are complete.

   Note: If the application cannot poll the register after soft reset (because of performance reasons), then it is recommended that you continue with the next steps and check this register again (as mentioned in step 12 on page 585) before triggering the DMA operations.†
4. Program the following fields to initialize the Bus Mode Register by setting values in DMA Register 0 (Bus Mode Register):†
   - Mixed Burst and AAL
   - Fixed burst or undefined burst†
   - Burst length values and burst mode values†
   - Descriptor Length (only valid if Ring Mode is used)†
5. Program the interface options in Register 10 (AXI Bus Mode Register). If fixed burst-length is enabled, then select the maximum burst-length possible on the bus (bits[7:1]).†
6. Create a proper descriptor chain for transmit and receive. In addition, ensure that the receive descriptors are owned by DMA (bit 31 of descriptor should be set). When OSF mode is used, at least two descriptors are required.
7. Make sure that your software creates three or more different transmit or receive descriptors in the chain before reusing any of the descriptors.†
8. Initialize receive and transmit descriptor list address with the base address of the transmit and receive descriptor (Register 3 (Receive Descriptor List Address Register) and Register 4 (Transmit Descriptor List Address Register) respectively).†
9. Program the following fields to initialize the mode of operation in Register 6 (Operation Mode Register):
   - Receive and Transmit Store And Forward†
   - Receive and Transmit Threshold Control (RTC and TTC)†
   - Hardware Flow Control enable†
   - Flow Control Activation and De-activation thresholds for MTL Receive and Transmit FIFO buffers (RFA and RFD)†
   - Error frame and undersized good frame forwarding enable†
   - OSF Mode†
10. Clear the interrupt requests, by writing to those bits of the status register (interrupt bits only) that are set. For example, by writing 1 into bit 16, the normal interrupt summary clears this bit (DMA Register 5 (Status Register)).†
11. Enable the interrupts by programming Register 7 (Interrupt Enable Register).†

Note: Perform step 12 on page 585 only if you did not perform step 3 on page 584.†

12. Read Register 11 (AHB or AXI Status) to confirm that all previous transactions are complete.†

    Note: If any previous transaction is still in progress when you read the Register 11 (AXI Status), then it is strongly recommended to check the slave components addressed by the master interface.†

13. Start the receive and transmit DMA by setting SR (bit 1) and ST (bit 13) of the control register (DMA Register 6 (Operation Mode Register). †

18.6.5. EMAC Initialization and Configuration

The following EMAC configuration operations can be performed after DMA initialization. If the EMAC initialization and configuration is done before the DMA is set up, then enable the MAC receiver (last step below) only after the DMA is active. Otherwise, the received frame could fill the RX FIFO buffer and overflow.

1. Program the `GMII Address Register` (offset 0x10) for controlling the management cycles for the external PHY. Bits[15:11] of the `GMII Address Register` are written with the Physical Layer Address of the PHY before reading or writing. Bit 0 indicates if the PHY is busy and is set before reading or writing to the PHY management interface. †
2. Read the 16-bit data of the `GMII Data Register` from the PHY for link up, speed of operation, and mode of operation, by specifying the appropriate address value in bits[15:11] of the `GMII Address Register`. †
3. Provide the MAC address registers (`MAC Address0 High Register` through `MAC Address15 High Register` and `MAC Address0 Low Register` through `MAC Address15 Low Register`).
4. Program the `Hash Table Registers` 0 through 7 (offset 0x500 to 0x51C).
5. Program the following fields to set the appropriate filters for the incoming frames in the `MAC Frame Filter Register`: †
   - Receive All †
   - Promiscuous mode †
   - Hash or Perfect Filter †
   - Unicast, multicast, broadcast, and control frames filter settings †
6. Program the following fields for proper flow control in the `Flow Control Register`: †
   - Pause time and other pause frame control bits †
   - Receive and Transmit Flow control bits †
   - Flow Control Busy/Backpressure Activate †
7. Program the `Interrupt Mask Register` bits, as required and if applicable for your configuration. †
8. Program the appropriate fields in `MAC Configuration Register` to configure receive and transmit operation modes. After basic configuration is written, set bit 3 (TE) and bit 2 (RE) in this register to enable the receive and transmit state machines. †

Note: Do not change the configuration (such as duplex mode, speed, port, or loopback) when the EMAC DMA is actively transmitting or receiving. Software should change these parameters only when the EMAC DMA transmitter and receiver are not active.

18.6.6. Performing Normal Receive and Transmit Operation

For normal operation, perform the following steps: †

1. For normal transmit and receive interrupts, read the interrupt status. Then, poll the descriptors, reading the status of the descriptor owned by the Host (either transmit or receive). †
2. Set appropriate values for the descriptors, ensuring that transmit and receive descriptors are owned by the DMA to resume the transmission and reception of data. †
3. If the descriptors are not owned by the DMA (or no descriptor is available), the DMA goes into SUSPEND state. The transmission or reception can be resumed by freeing the descriptors and issuing a poll demand by writing 0 into the TX/RX poll demand registers, (Register 1 (Transmit Poll Demand Register) and Register 2 (Receive Poll Demand Register)). †
4. The values of the current host transmitter or receiver descriptor address pointer can be read for the debug process (Register 18 (Current Host Transmit Descriptor Register) and Register 19 (Current Host Receive Descriptor Register)). †
5. The values of the current host transmit buffer address pointer and receive buffer address pointer can be read for the debug process (Register 20 (Current Host Transmit Buffer Address Register) and Register 21 (Current Host Receive Buffer Address Register)). †

18.6.7. Stopping and Starting Transmission

Perform the following steps to pause the transmission for some time: †

1. Disable the transmit DMA (if applicable), by clearing bit 13 (Start or Stop Transmission Command) of Register 6 (Operation Mode Register). †
2. Wait for any previous frame transmissions to complete. You can check this by reading the appropriate bits of Register 9 (Debug Register). †
3. Disable the EMAC transmitter and EMAC receiver by clearing Bit 3 (TE) and Bit 2 (RE) in Register 0 (MAC Configuration Register). †
4. Disable the receive DMA (if applicable), after making sure that the data in the RX FIFO buffer is transferred to the system memory (by reading Register 9 (Debug Register). †
5. Make sure that both the TX FIFO buffer and RX FIFO buffer are empty. †
6. To re-start the operation, first start the DMA and then enable the EMAC transmitter and receiver. †

18.6.8. Programming Guidelines for Energy Efficient Ethernet

18.6.8.1. Entering and Exiting the TX LPI Mode

The Energy Efficient Ethernet (EEE) feature is available in the EMAC. To use it, perform the following steps during EMAC initialization:

1. Read the PHY register through the MDIO interface, check if the remote end has the EEE capability, and then negotiate the timer values. †
2. Program the PHY registers through the MDIO interface (including the RX_CLK_stoppable bit that indicates to the PHY whether to stop the RX clock in LPI mode.) †
3. Program Bits[16:5], LST, and Bits[15:0], TWT, in Register 13 (LPI Timers Control Register). †
4. Read the link status of the PHY chip by using the MDIO interface and update Bit 17 (PLS) of Register 12 (LPI Control and Status Register) accordingly. This update should be done whenever the link status in the PHY changes. †
5. Set Bit 16 (LPIEN) of Register 12 (LPI Control and Status Register) to make the MAC enter the LPI state. The MAC enters the LPI mode after completing the transmission in progress and sets Bit 0 (TLPIEN). †

   Note: To make the MAC enter the LPI state only after it completes the transmission of all queued frames in the TX FIFO buffer, you should set Bit 19 (LPITXA) in Register 12 (LPI Control and Status Register). †

   Note: To switch off the transmit clock during the LPI state, use the `sbd_tx_clk_gating_ctrl_o` signal for gating the clock input. †

   Note: To switch off the CSR clock or power to the rest of the system during the LPI state, you should wait for the TLPIEN interrupt of Register 12 (LPI Control and Status Register) to be generated. Restore the clocks before performing step 6 on page 587 when you want to come out of the LPI state. †
6. Clear Bit 16 (LPIEN) of Register 12 (LPI Control and Status Register) to bring the MAC out of the LPI state. †

The MAC waits for the time programmed in Bits [15:0], TWT, before setting the TLPIEX interrupt status bit and resuming the transmission. †

18.6.8.2. Gating Off the CSR Clock in the LPI Mode

You can gate off the CSR clock to save the power when the MAC is in the Low-Power Idle (LPI) mode.†

18.6.8.2.1. Gating Off the CSR Clock in the RX LPI Mode

The following operations are performed when the MAC receives the LPI pattern from the PHY. †

1. The MAC RX enters the LPI mode and the RX LPI entry interrupt status [RLPIEN interrupt of Register 12 (`LPI_Control_Status`)] is set. †
2. The interrupt pin (`sbd_intr_o`) is asserted. The `sbd_intr_o` interrupt is cleared when the host reads the Register 12 (LPI_Control_Status). †

After the `sbd_intr_o` interrupt is asserted and the MAC TX is also in the LPI mode, you can gate off the CSR clock. If the MAC TX is not in the LPI mode when you gate off the CSR clock, the events on the MAC transmitter do not get reported or updated in the CSR. †

For restoring the CSR clock, wait for the LPI exit indication from the PHY after which the MAC asserts the LPI exit interrupt on `lpi_intr_o` (synchronous to `clk_rx_i`). The `lpi_intr_o` interrupt is cleared when Register 12 is read. †

18.6.8.2.2. Gating Off the CSR Clock in the TX LPI Mode

The following operations are performed when Bit 16 (LPIEN) of Register 12 (LPI Control and Status Register) is set: †

1. The Transmit LPI Entry interrupt (TLPIEN bit of Register 12) is set. †
2. The interrupt pin (`sbd_intr_o`) is asserted. The `sbd_intr_o` interrupt is cleared when the host reads the Register 12. †

After the `sbd_intr_o` interrupt is asserted and the MAC RX is also in the LPI mode, you can gate off the CSR clock. If the MAC RX is not in the LPI mode when you gate off the CSR clock, the events on the MAC receiver do not get reported or updated in the CSR. †

To restore the CSR clock, switch on the CSR clock when the MAC has to come out of the TX LPI mode. †

After the CSR clock is resumed, clear Bit 16 (LPIEN) of Register 12 (LPI Control and Status Register) to bring the MAC out of the LPI mode. †

18.6.9. Programming Guidelines for Flexible Pulse-Per-Second (PPS) Output

18.6.9.1. Generating a Single Pulse on PPS

To generate single Pulse on PPS: †

1. Program 11 or 10 (for interrupt) in Bits [6:5], TRGTMODSEL, of Register 459 (PPS Control Register) to instruct the MAC to use the Target Time registers (register 455 and 456) for the start time of PPS signal output. †
2. Program the start time value in the Target Time registers (register 455 and 456). †
3. Program the width of the PPS signal output in Register 473 (PPS0 Width Register). †
4. Program Bits [3:0], PPSCMD, of Register 459 (PPS Control Register) to 0001 to instruct the MAC to generate a single pulse on the PPS signal output at the time programmed in the Target Time registers (register 455 and 456). †

Once the PPSCMD is executed (PPSCMD bits = 0), you can cancel the pulse generation by giving the Cancel Start Command (PPSCMD=0011) before the programmed start time elapses. You can also program the behavior of the next pulse in advance. To program the next pulse: †

1. Program the start time for the next pulse in the Target Time registers (register 455 and 456). This time should be more than the time at which the falling edge occurs for the previous pulse. †
2. Program the width of the next PPS signal output in Register 473 (PPS0 Width Register). †
3. Program Bits [3:0], PPSCMD, of Register 459 (PPS Control Register) to generate a single pulse on the PPS signal output after the time at which the previous pulse is de-asserted. And at the time programmed in Target Time registers. If you give this command before the previous pulse becomes low, then the new command overwrites the previous command and the EMAC may generate only 1 extended pulse.

18.6.9.2. Generating a Pulse Train on PPS

To generate a pulse train on PPS: †

1. Program 11 or 10 (for an interrupt) in Bits [6:5], TRGTMODSEL, of Register 459 (PPS Control Register) to instruct the MAC to use the Target Time registers (register 455 and 456) for the start time of the PPS signal output. †
2. Program the start time value in the Target Time registers (register 455 and 456). †
3. Program the interval value between the train of pulses on the PPS signal output in Register 473 (PPS0 Width Register). †
4. Program the width of the PPS signal output in Register 473 (PPS0 Width Register). †
5. Program Bits[3:0], PPSCMD, of Register 459 (PPS Control Register) to 0010 to instruct the MAC to generate a train of pulses on the PPS signal output with the start time programmed in the Target Time registers (register 455 and 456). By default, the PPS pulse train is free-running unless stopped by 'STOP Pulse train at time' or 'STOP Pulse Train immediately' commands. †
6. Program the stop value in the Target Time registers (register 455 and 456). Ensure that Bit 31 (TSTRBUSY) of Register 456 (Target Time Nanoseconds Register) is clear before programming the Target Time registers (register 455 and 456) again. †
7. Program the PPSCMD field (bit 3:0) of Register 459 (PPS Control Register) to 0100 to stop the train of pulses on the PPS signal output after the programmed stop time specified in step 6 on page 589 elapses. †

You can stop the pulse train at any time by programming 0101 in the PPSCMD field. Similarly, you can cancel the Stop Pulse train command (given in step 7 on page 589) by programming 0110 in the PPSCMD field before the time (programmed in step 6 on page 589) elapses. You can cancel the pulse train generation by programming 0011 in the PPSCMD field before the programmed start time (in step 2 on page 589) elapses. †

18.6.9.3. Generating an Interrupt without Affecting the PPS

Bits [6:5], TRGTMODSEL, of Register 459 (PPS Control Register) enable you to program the Target Time registers (register 455 and 456) to do any one of the following: †

- Generate only interrupts. †
- Generate interrupts and the PPS start and stop time. †
- Generate only PPS start and stop time. †

To program the Target Time registers (register 455 and 456) to generate only interrupt events: †

1. Program 00 (for interrupt) in Bits [6:5], TRGTMODSEL, of Register 459 (PPS Control Register) to instruct the MAC to use the Target Time registers (register 455 and 456) for the target time interrupt. †
2. Program a target time value in the Target Time registers (register 455 and 456) to instruct the MAC to generate an interrupt when the target time elapses. If Bits [6:5], TRGTMODSEL, are changed (for example, to control the PPS), then the interrupt generation is overwritten with the new mode and new programmed Target Time register value.

18.7. Ethernet MAC Address Map and Register Definitions

The address map and register definitions pertain to the following modules:

- EMAC Module 0
- EMAC Module 1
- EMAC Module 2

In addition to the EMAC modules, the address map and register definitions for the ECC controllers of the 4-KB Tx FIFO RAM and the 16-KB Rx FIFO RAM are listed in this section.

Related Information

Error Checking and Correction Controller on page 345
For more information on the functionality and programming of the ECC Controller.

19. USB 2.0 OTG Controller

The hard processor system (HPS) provides two instances of a USB On-The-Go (OTG) controller that supports both device and host functions. The controller supports all high-speed, full-speed, and low-speed transfers in both device and host modes. The controller is fully compliant with the *On The Go and Embedded Host Supplement to the USB Revision 2.0 Specification*. The controller can be programmed for both device and host functions to support data movement over the USB protocol.

The controllers are operationally independent of each other. Each USB OTG controller supports a single USB port connected through a USB 2.0 Transceiver Macrocell Interface Plus (UTMI+) Low Pin Interface (ULPI) compliant PHY. The USB OTG controllers are instances of the Synopsys (†)DesignWare Cores USB 2.0 Hi-Speed On-The-Go (DWC_otg) controller.

The USB OTG controller is optimized for the following applications and systems: †

- Portable electronic devices †
- Point-to-point applications (no hub, direct connection to HS, FS, or LS device) †
- Multi-point applications (as an embedded USB host) to devices (hub and split support) †

Each of the two USB OTG ports supports both host and device modes, as described in the *On The Go and Embedded Host Supplement to the USB Revision 2.0 Specification*. The USB OTG ports support connections for all types of USB peripherals, including the following peripherals:

- Mouse
- Keyboard
- Digital cameras
- Network adapters
- Hard drives
- Generic hubs

(†) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non infringement, and any warranties arising out of a course of dealing or usage of trade.

† Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Related Information

- http://www.usb.org/home
  Additional information is available in the On The Go and Embedded Host Supplement to the USB Revision 2.0 Specification, which you can download from the USB Implementers Forum website
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

19.1. Features of the USB OTG Controller

The USB OTG controller has the following USB-specific features:

- Complies with both Revision 1.3 and Revision 2.0 of the *On The Go and Embedded Host Supplement to the USB Revision 2.0 Specification*
- Supports software-configurable modes of operation between OTG 1.3 and OTG 2.0
- Can operate in Host or Device mode
- Supports multi-point applications with hub and split support
- Supports all USB 2.0 speeds:
  - High speed (HS, 480-Mbps)
  - Full speed (FS, 12-Mbps)
  - Low speed (LS, 1.5-Mbps)

  Note: In host mode, all speeds are supported. However, in device mode, only high speed and full speed are supported.
- Integrated scatter-gather DMA supports moving data between memory and the controller
- Supports USB 2.0 in ULPI mode
- Supports all USB transaction types:
  - Control transfers
  - Bulk transfers
  - Isochronous transfers
  - Interrupts
- Supports automatic ping capability
- Supports Session Request Protocol (SRP) and Host Negotiation Protocol (HNP)
- Supports suspend, resume, and remote wake
- Supports up to 16 host channels

  Note: In host mode, when the number of device endpoints is greater than the number of host channels, software can reprogram the channels to support up to 127 devices, each having 32 endpoints (IN + OUT), for a maximum of 4,064 endpoints.

- Supports up to 16 bidirectional endpoints, including control endpoint 0

  Note: Only seven periodic device IN endpoints are supported.
- Supports a generic root hub
- Performs transaction scheduling in hardware

On the USB PHY layer, the USB OTG controller supports the following features:

- 8-bit ULPI PHY data width
- A single USB port connected to each OTG instance
- A ULPI connection to an off-chip USB transceiver
- Software-controlled access, supporting vendor-specific or optional PHY registers access to ease debug
- The OTG 2.0 support for Attach Detection Protocol (ADP) only through an external (off-chip) ADP controller

On the integration side, the USB OTG controller supports the following features:

- Different clocks for system and PHY interfaces
- Dedicated TX FIFO buffer for each device IN endpoint in direct memory access (DMA) mode
- Packet-based, dynamic FIFO memory allocation for endpoints for small FIFO buffers and flexible, efficient use of RAM that can be dynamically sized by software
- Ability to change an endpoint's FIFO memory size during transfers
- Clock gating support during USB suspend and session-off modes
  - PHY clock gating support
  - System clock gating support
- Data FIFO RAM clock gating support
- Local buffering with error correction code (ECC) support

Note: The USB OTG controller does not support the following protocols:

- Enhanced Host Controller Interface (EHCI)
- Open Host Controller Interface (OHCI)
- Universal Host Controller Interface (UHCI)

19.1.1. Supported PHYS

The USB OTG controller only supports USB 2.0 ULPI PHYs. Only the single data rate (SDR) mode is supported.

Refer to the *Device Datasheet* for specific timing information.

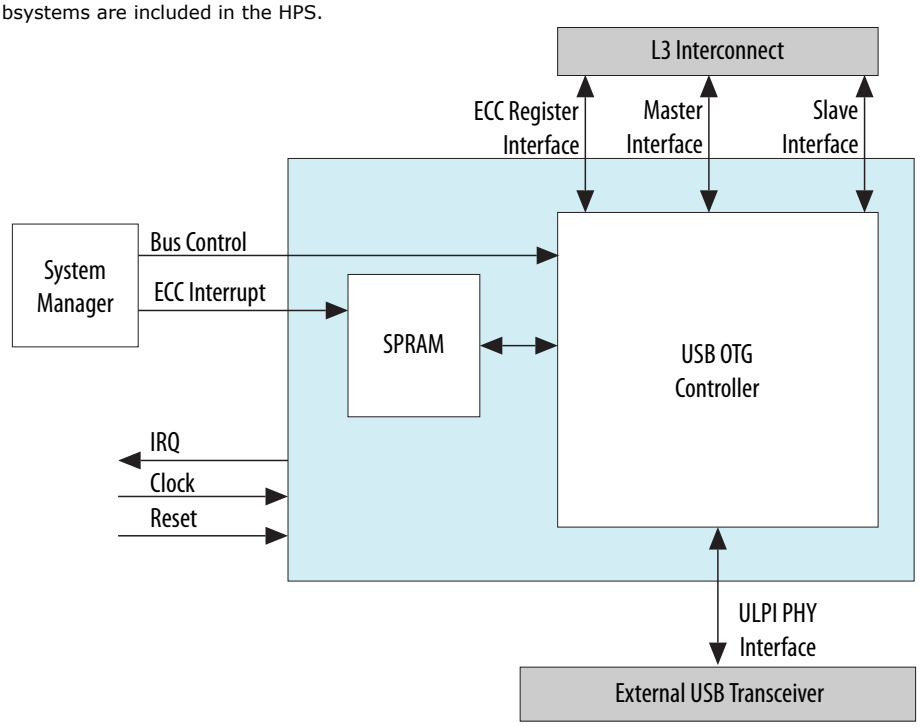
Related Information

Intel Intel Arria 10 Device Datasheet

19.2. USB OTG Controller Block Diagram and System Integration

Figure 104. USB OTG Controller System Integration

Two subsystems are included in the HPS.



The USB OTG controller connects to the level 3 (L3) interconnect through a slave interface, allowing other masters to access the control and status registers (CSRs) in the controller. The controller also connects to the L3 interconnect through a master interface, allowing the DMA engine in the controller to move data between external memory and the controller.

A single-port RAM (SPRAM) connected to the USB OTG controller is used to store USB data packets for both host and device modes. It is configured as FIFO buffers for receive and transmit data packets on the USB link.

Through the system manager, the USB OTG controller has control to use and test error correction codes (ECCs) in the SPRAM. Through the system manager, the USB OTG controller can also control the behavior of the master interface to the L3 interconnect.

The USB OTG controller connects to the external USB transceiver through a ULPI PHY interface. This interface also connects through pin multiplexers within the HPS. The pin multiplexers are controlled by the system manager.

Additional connections on the USB OTG controller include:

- Clock input from the clock manager to the USB OTG controller
- Reset input from the reset manager to the USB OTG controller
- Interrupt line from the USB OTG controller to the microprocessor unit (MPU) global interrupt controller (GIC).

The USB Controller signals are routed to the dedicated HPS pins.

Related Information

- System Manager on page 159
  Details available in the System Manager chapter.
- General-Purpose I/O Interface on page 684

19.3. USB 2.0 ULPI PHY Signal Description

Table 208. ULPI PHY Interfaces

The ULPI PHY interface is synchronous to the `ulpi_clk` signal coming from the PHY.

Note: For more information on how the USB signals are routed to the HPS I/O, refer to the *HPS Component Interfaces* chapter.

Port Name	Bit Width	Direction	Description
`ulpi_clk`	1	Input	ULPI Clock Receives the 60-MHz clock supplied by the high-speed ULPI PHY. All signals are synchronous to the positive edge of the clock.
`ulpi_dir`	1	Input	ULPI Data Bus Control 1—The PHY has data to transfer to the USB OTG controller. 0—The PHY does not have data to transfer.
`ulpi_nxt`	1	Input	ULPI Next Data Control Indicates that the PHY has accepted the current byte from the USB OTG controller. When the PHY is transmitting, this signal indicates that a new byte is available for the controller.
`ulpi_stp`	1	Output	ULPI Stop Data Control The controller drives this signal high to indicate the end of its data stream. The controller can also drive this signal high to request data from the PHY.
`ulpi_data[7:0]`	8	Bidirectional	Bidirectional data bus. Driven low by the controller during idle.

Related Information

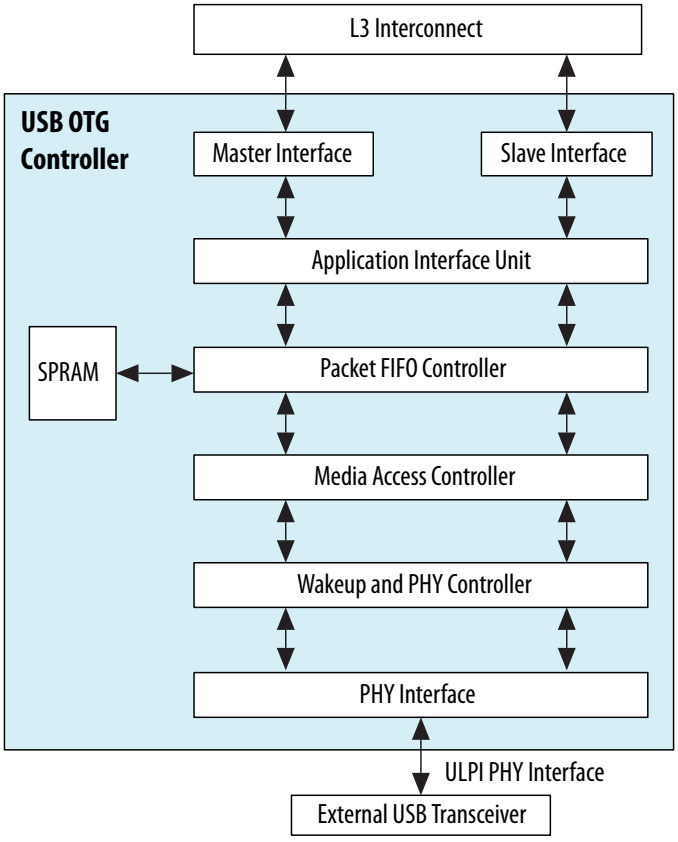
USB 2.0 OTG Controller Interface on page 741
For more information about how the interface signals are routed to the HPS I/O, refer to this chapter.

19.4. Functional Description of the USB OTG Controller

19.4.1. USB OTG Controller Block Description

Figure 105. USB OTG Controller Block Description

Details about each of the units that comprise the USB OTG controller are shown below.



19.4.1.1. Master Interface

The master interface includes a built-in DMA controller. The DMA controller moves data between external memory and the media access controller (MAC).

Properties of the master interface are controlled through the USB L3 Master HPROT Register (`l3master`) in the system manager. These bits provide access information to the L3 interconnect, including whether or not transactions are cacheable, bufferable, or privileged.

Note: Bits in the `l3master` register can be updated only when the master interface is guaranteed to be in an inactive state.

19.4.1.2. Slave Interface

The slave interface allows other masters in the system to access the USB OTG controller's CSRs. For testing purposes, other masters can also access the single port RAM (SPRAM).

19.4.1.2.1. Slave Interface CSR Unit

The slave interface can read from and write to all the CSRs in the USB OTG controllers. All register accesses are 32 bits.

The CSR is divided into the following groups of registers:

- Global
- Host
- Device
- Power and clock gating

Some registers are shared between host and device modes, because the controller can only be in one mode at a time. The controller generates a mode mismatch interrupt if a master attempts to access device registers when the controller is in host mode, or attempts to access host registers when the controller is in device mode. Writing to unimplemented registers is ignored. Reading from unimplemented registers returns indeterminate values.

19.4.1.3. Application Interface Unit

The application interface unit (AIU) generates DMA requests based on programmable FIFO buffer thresholds. The AIU generates interrupts to the GIC for both host and device modes. A DMA scheduler is included in the AIU to arbitrate and control the data transfer between packets in system memory and their respective USB endpoints.

19.4.1.4. Packet FIFO Controller

The Packet FIFO Controller (PFC) connects the AIU with the MAC through data FIFO buffers located in the SPRAM. In device mode, one FIFO buffer is implemented for each IN endpoint. In host mode, a single FIFO buffer stores data for all periodic (isochronous and interrupt) OUT endpoints, and a single FIFO buffer is used for nonperiodic (control and bulk) OUT endpoints. Host and device mode share a single receive data FIFO buffer.

19.4.1.5. SPRAM

An SPRAM implements the data FIFO buffers for host and device modes. The size of the FIFO buffers can be programmed dynamically.

The SPRAM supports ECCs.

19.4.1.6. MAC

The MAC module implements the following functionality:

- USB transaction support
- Host protocol support
- Device protocol support
- OTG protocol support

19.4.1.6.1. USB Transactions

In device mode, the MAC decodes and checks the integrity of all token packets. For valid OUT or SETUP tokens, the following DATA packet is also checked. If the data packet is valid, the MAC performs the following steps:

1. Writes the data to the receive FIFO buffer
2. Sends the appropriate handshake when required to the USB host

If a receive FIFO buffer is not available, the MAC sends a NAK response to the host. The MAC also supports ping protocol.

For IN tokens, if data is available in the transmit FIFO buffer, the MAC performs the following steps:

1. Reads the data from the FIFO buffer
2. Forms the data packet
3. Transmits the packet to the host
4. Receives the response from the host
5. Sends the updated status to the PFC

In host mode, the MAC receives a token request from the AIU. The MAC performs the following steps:

1. Builds the token packet
2. Sends the packet to the device

For OUT or SETUP transactions, the MAC also performs the following steps:

1. Reads the data from the transmit FIFO buffer
2. Assembles the data packet
3. Sends the packet to the device
4. Waits for a response

The response from the device causes the MAC to send a status update to the AIU.

For IN or PING transactions, the MAC waits for the data or handshake response from the device. For data responses, the MAC performs the following steps:

1. Validates the data
2. Writes the data to the receive FIFO buffer
3. Sends a status update to the AIU
4. Sends a handshake to the device, if appropriate

19.4.1.6.2. Host Protocol

In host mode, the MAC performs the following functions:

- Detects connect, disconnect, and remote wakeup events on the USB link
- Initiates reset
- Initiates speed enumeration processes
- Generates Start of Frame (SOF) packets.

19.4.1.6.3. Device Protocol

In device mode, the MAC performs the following functions:

- Handles USB reset sequence
- Handles speed enumeration
- Detects USB suspend and resume activity on the USB link
- Initiates remote wakeup
- Decodes SOF packets

19.4.1.6.4. OTG Protocol

The MAC handles HNP and SRP for OTG operation. HNP provides a mechanism for swapping host and device roles. SRP provides mechanisms for the host to turn off V_{BUS} to save power, and for a device to request a new USB session.

19.4.1.7. Wakeup and Power Control

To reduce power, the USB OTG controller supports a power-down mode. In power-down mode, the controller and the PHY can shut down their clocks. The controller supports wakeup on the detection of the following events:

- Resume
- Remote wakeup
- Session request protocol
- New session start

19.4.1.8. PHY Interface Unit

The USB OTG controller supports synchronous SDR data transmission to a ULPI PHY. The SDR mode implements an eight-bit data bus.

19.4.1.9. DMA

The DMA has two modes of operation. You program your software to select between scatter-gather DMA mode or buffer DMA mode depending on the controllers function.

If the controller is functioning as a generic root hub, you should program your software to select the buffer DMA mode that supports split transfers.

If generic root hub functionality is not required, or if the controller is configured in Device mode, you can program your software to select scatter-gather DMA mode.

If you wish to dynamically switch the mode of operation based on the queried device status or capability, your software driver must cleanly switch between the two modes of operation. For example, you may want the controller to default to scatter-gather DMA mode and only change mode when it detects a generic HUB with fast-speed and low-speed capability. Some basic requirements for switching include:

- A soft reset must be issued before changing modes.
- If buffer DMA mode is selected, then the Host mode periodic request queue depth must not be set to 16.
- Devices must be re-enumerated.

19.4.2. Local Memory Buffer

The USB controller has three local SRAM memory buffers.

- The write FIFO buffer is a 128 × 32-bit memory (512 total bytes)
- The read FIFO buffer is a 32 × 32-bit memory (128 total bytes)
- The ECC buffer is a 96 × 16-bit memory (1536 total bytes)

The SPRAM is a 8192 x 35-bit (32 data bits and 3 control bits) memory and includes support for ECC (Error Checking and Correction). The ECC block is integrated around a memory wrapper. It provides outputs to notify the system manager when single-bit correctable errors are detected (and corrected) and when double-bit uncorrectable errors are detected. The ECC logic also allows the injection of single- and double-bit errors for test purposes. The ECC feature is disabled by default. It must be initialized to enable the ECC function.

19.4.3. Clocks

Table 209. USB OTG Controller Clock Inputs

All clocks must be operational when reset is released. No special handling is required on the clocks.

Clock Signal	Frequency	Functional Usage
usb_mp_clk	60 – 200 MHz	Drives the master and slave interfaces, DMA controller, and internal FIFO buffers
usb0_ulpi_clk	60 MHz	ULPI reference clock for usb0 from external ULPI PHY I/O pin
usb1_ulpi_clk	60 MHz	ULPI reference clock for usb1 from external ULPI PHY I/O pin

19.4.3.1. Clock Gating

You can clock gate the ulpi_clk through software. By programming the usbclken bit of the en register in the perpllgrp you can enable or disable the ulpi_clk to the USB.

Related Information

Clock Manager Address Map and Register Definitions on page 67

19.4.4. Resets

The USB OTG controller can be reset either through the hardware reset input or through software.

19.4.4.1. Reset Requirements

There must be a minimum of 12 cycles on the ulpi_clk clock before the controller is taken out of reset. During reset, the USB OTG controller asserts the ulpi_stp signal. The PHY outputs a clock when it sees the ulpi_stp signal asserted. However, if the pin multiplexers are not programmed, the PHY does not see the ulpi_stp signal. As a result, the ulpi_clk clock signal does not arrive at the USB OTG controller.

Software must ensure that the reset is active for a minimum of two usb_mp_clk cycles. There is no maximum assertion time.

19.4.4.2. Hardware Reset

Each of the USB OTG controllers has one reset input from the reset manager. The reset signal is asserted during a cold or warm reset event. The reset manager holds the controllers in reset until software releases the resets. Software releases resets by clearing the appropriate USB bits in the Peripheral Module Reset Register (`permodrst`) in the HPS reset manager.

The reset input resets the following blocks:

- The master and slave interface logic
- The integrated DMA controller
- The internal FIFO buffers
- The CSR

The reset input is synchronized to the `usb_mp_clk` domain. The reset input is also synchronized to the ULPI clock within the USB OTG controller and is used to reset the ULPI PHY domain logic.

19.4.4.3. Software Reset

Software can reset the controller by setting the Core Soft Reset (`csftrst`) bit in the Reset Register (`grstctl`) in the Global Registers (`globgrp`) group of the USB OTG controller.

Software resets are useful in the following situations:

- A PHY selection bit is changed by software. Resetting the USB OTG controller is part of clean-up to ensure that the PHY can operate with the new configuration or clock.
- During software development and debugging.

19.4.4.4. Taking the USB 2.0 OTG Controller Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

19.4.5. Interrupts

Table 210. USB OTG Interrupt Conditions

Each USB OTG controller has a single interrupt output. Interrupts are asserted on the conditions shown in the following table.

Condition	Mode
Device-initiated remote wakeup is detected.	Host mode
Session request is detected from the device.	Host mode
Device disconnect is detected.	Host mode
Host periodic TX FIFO buffer is empty (can be further programmed to indicate half-empty).	Host mode
Host channels interrupt received.	Host mode
Incomplete periodic transfer is pending at the end of the microframe.	Host mode
Host port status interrupt received.	Host mode
External host initiated resume is detected.	Device mode
Reset is detected when in suspend or normal mode.	Device mode
USB suspend mode is detected.	Device mode
Data fetch is suspended due to TX FIFO buffer full or request queue full.	Device mode
At least one isochronous OUT endpoint is pending at the end of the microframe.	Device mode
At least one isochronous IN endpoint is pending at the end of the microframe.	Device mode
At least one IN or OUT endpoint interrupt is pending at the end of the microframe.	Device mode
The end of the periodic frame is reached.	Device mode
Failure to write an isochronous OUT packet to the RX FIFO buffer. The RX FIFO buffer does not have enough space to accommodate the maximum packet size for the isochronous OUT endpoint.	Device mode
Enumeration has completed.	Device mode
Connector ID change.	Common modes
Mode mismatch. Software accesses registers belonging to an incorrect mode.	Common modes
Nonperiodic TX FIFO buffer is empty.	Common modes
RX FIFO buffer is not empty.	Common modes
Start of microframe.	Common modes
Device connection debounce is complete in host mode.	OTG interrupts
A-Device timeout while waiting for B-Device connection.	OTG interrupts
Host negotiation is complete.	OTG interrupts
Session request is complete.	OTG interrupts
Session end is detected in device mode.	OTG interrupts

19.5. USB OTG Controller Programming Model

For detailed information about using the USB OTG controller, consult your operating system (OS) driver documentation. The OS vendor provides application programming interfaces (APIs) to control USB host, device and OTG operation. This section provides a brief overview of the following software operations:

- Enabling SPRAM ECCs
- Host operation
- Device operation

19.5.1. Enabling ECC

1. Turn on the ECC hardware, but disable the interrupts.
2. Initialize the SRAM in the peripheral.(49)
3. Clear the ECC event bits, because these bits may have become asserted after Step 1.
4. Enable the ECC interrupts now that the ECC bits have been set.

19.5.2. Enabling SPRAM ECCs

The L3 interconnect has access to the SPRAM and is accessible through the USB OTG L3 slave interface. Software accesses the SPRAM through the `directfifo` memory space, in the USB OTG controller address space.

To enable the ECC feature, refer to the ECC chapter in the *Arria 10 Hard Processor System Technical Reference Manual*.

Note: Software cannot access the SPRAM beyond the 32-KB range. Out-of-range read transactions return indeterminate data. Out-of-range write transactions are ignored.

Related Information

Error Checking and Correction Controller on page 345

19.5.3. Host Operation

19.5.3.1. Host Initialization

After power up, the USB port is in its default mode. No VBUS is applied to the USB cable. The following process sets up the USB OTG controller as a USB host.

1. To enable power to the USB port, the software driver sets the Port Power (`prtpwr`) bit to 1 in the Host Port Control and Status Register (`hprt`) of the Host Mode Registers (`hostgrp`) group. This action drives the V_{BUS} signal on the USB link.

(49) This is the case for the following peripherals: SD/MMC, NAND, On-Chip, DMA, USB, and EMAC.

The controller waits for a connection to be detected on the USB link.

2. When a USB device connects, an interrupt is generated. The Port Connect Detected (`PrtConnDet`) bit in `hprt` is set to 1.
3. Upon detecting a port connection, the software driver initiates a port reset by setting the Port Reset (`prtrst`) bit to 1 in `hprt`.
4. The software driver must wait a minimum of 10 ms so that speed enumeration can complete on the USB link.
5. After the 10 ms, the software driver sets `prtrst` back to 0 to release the port reset.
6. The USB OTG controller generates an interrupt. The Port Enable Disable Change (`prtenchng`) and Port Speed (`prtspd`) bits, in `hprt`, are set to reflect the enumerated speed of the device that attached.

   At this point the port is enabled for communication. Keep alive or SOF packets are sent on the port. If a USB 2.0-capable device fails to initialize correctly, it is reported as a USB 1.1 device.

   The Host Frame Interval Register (`hfir`) is updated with the corresponding PHY clock settings. The `hfir`, used for sending SOF packets, is in the Host Mode Registers (`hostgrp`) group.
7. The software driver must program the following registers in the Global Registers (`globgrp`) group, in the order listed:
   a. Receive FIFO Size Register (`grxfsiz`)—selects the size of the receive FIFO buffer
   b. Non-periodic Transmit FIFO Size Register (`gnptxfsiz`)—selects the size and the start address of the non-periodic transmit FIFO buffer for nonperiodic transactions
   c. Host Periodic Transmit FIFO Size Register (`hptxfsiz`)—selects the size and start address of the periodic transmit FIFO buffer for periodic transactions
8. System software initializes and enables at least one channel to communicate with the USB device.

19.5.3.2. Host Transaction

When configured as a host, the USB OTG controller pipes the USB transactions through one of two request queues (one for periodic transactions and one for nonperiodic transactions). Each entry in the request queue holds the SETUP, IN, or OUT channel number along with other information required to perform a transaction on the USB link. The sequence in which the requests are written to the queue determines the sequence of transactions on the USB link.

The host processes the requests in the following order at the beginning of each frame or microframe:

1. Periodic request queue, including isochronous and interrupt transactions
2. Nonperiodic request queue (bulk or control transfers)

The host schedules transactions for each enabled channel in round-robin fashion. When the host controller completes the transfer for a channel, the controller updates the DMA descriptor status in the system memory.

For OUT transactions, the host controller uses two transmit FIFO buffers to hold the packet payload to be transmitted. One transmit FIFO buffer is used for all nonperiodic OUT transactions and the other is used for all periodic OUT transactions.

For IN transactions, the USB host controller uses one receive FIFO buffer for all periodic and nonperiodic transactions. The controller holds the packet payload from the USB device in the receive FIFO buffer until the packet is transferred to the system memory. The receive FIFO buffer also holds the status of each packet received. The status entry holds the IN channel number along with other information, including received byte count and validity status.

For generic hub operations, the USB OTG controller uses SPLIT transfers to communicate with slower-speed devices downstream of the hub. For these transfers, the transaction accumulation or buffering is performed in the generic hub, and is scheduled accordingly. The USB OTG controller ensures that enough transmit and receive buffers are allocated when the downstream transactions are completed or when accumulated data is ready to be sent upstream.

19.5.4. Device Operation

19.5.4.1. Device Initialization

The following process sets up the USB OTG controller as a USB device:

1. After power up, the USB OTG controller must be set to the desired device speed by writing to the Device Speed (`devspd`) bits in the Device Configuration Register (`dcfg`) in the Device Mode Registers (`devgrp`) group. After the device speed is set, the controller waits for a USB host to detect the USB port as a device port.
2. When an external host detects the USB port, the host performs a port reset, which generates an interrupt to the USB device software. The USB Reset (`usbrst`) bit in the Interrupt (`port reset`) register in the Global Registers (`globgrp`) group is set. The device software then sets up the data FIFO buffer to receive a SETUP packet from the external host. Endpoint 0 is not enabled yet.
3. After completion of the port reset, the operation speed required by the external host is known. Software reads the device speed status and sets up all the remaining required transaction fields to enable control endpoint 0.

After completion of this process, the device is receiving SOF packets, and is ready for the USB host to set up the device's control endpoint.

19.5.4.2. Device Transaction

When configured as a device, the USB OTG controller uses a single FIFO buffer to receive the data for all the OUT endpoints. The receive FIFO buffer holds the status of the received data packet, including the byte count, the data packet ID (PID), and the validity of the received data. The DMA controller reads the data out of the FIFO buffer as the data are received. If a FIFO buffer overflow condition occurs, the controller responds to the OUT packet with a NAK, and internally rewinds the pointers.

For IN endpoints, the controller uses dedicated transmit buffers for each endpoint. The application does not need to predict the order in which the USB host accesses the nonperiodic endpoints. If a FIFO buffer underrun condition occurs during transmit, the controller inverts the cyclic redundancy code (CRC) to mark the packet as corrupt on the USB link.

The application handles one data packet at a time per endpoint in transaction-level operations. The software receives an interrupt on completion of every packet. Based on the handshake response received on the USB link, the application determines whether to retry the transaction or proceed with the next transaction, until all packets in the transfer are completed.

When an isochronous IN endpoint has more than one transaction (or packet) per micro-frame, it is referred to as a high-bandwidth isochronous endpoint. This endpoint should follow the Data PID sequencing specified in section 5.9.2 of the USB 2.0 Specification. This allows the device controller to support a maximum of three transactions per micro-frame and high-bandwidth isochronous IN endpoints in the Ignore Frame Number mode of the device descriptor DMA configuration.

19.5.4.2.1. IN Transactions

For an IN transaction, the application performs the following steps:

1. Enables the endpoint
2. Triggers the DMA engine to write the associated data packet to the corresponding transmit FIFO buffer
3. Waits for the packet completion interrupt from the controller

When an IN token is received on an endpoint when the associated transmit FIFO buffer does not contain sufficient data, the controller performs the following steps:

1. Generates an interrupt
2. Returns a NAK handshake to the USB host

If sufficient data is available, the controller transmits the data to the USB host.

19.5.4.2.2. OUT Transactions

For an OUT transaction, the application performs the following steps:

1. Enables the endpoint
2. Waits for the packet received interrupt from the USB OTG controller
3. Retrieves the packet from the receive FIFO buffer

When an OUT token or PING token is received on an endpoint where the receive FIFO buffer does not have sufficient space, the controller performs the following steps:

1. Generates an interrupt
2. Returns a NAK handshake to USB host

If sufficient space is available, the controller stores the data in the receive FIFO buffer and returns an ACK handshake to the USB link.

According to the USB 2.0 specifications, to transmit two packets back-to-back, the inter-packet delay must be a minimum of 88 bit times, measured at the first receptacle of the host. This guarantees an inter-packet delay of at least 32 bit times at all devices (when receiving these back-to-back packets). For the device controller to support the worst-case inter-packet delay of 32 bit times and isochronous OUT endpoints across multiple layers of hubs, the controller must operate in 8-bit UTMI mode and the AHB frequency must be at least 96 MHz.

Alternatively, choose the AHB clock frequency based on your targeted use case and supported number of hubs. However, in both cases the 8-bit PHY data interface should be used for operation using the SoC.

Table 211. Supported Number of Hubs Tiers for Different PHY and AHB Clock Frequencies

Number of Hub Levels	Worst Case IPG Possible at Hub/Host Downstream Port (in bit times)	Minimum HCLK Frequency Required (in MHz)	
		UTMI Data Width = 8	UTMI Data Width = 16
No Hub	88	30	110
1	78.4	32	213
2	68.8	37	-
3	59.2	44	-
4	49.6	54	-
5	40	70	-

19.5.4.2.3. Control Transfers

For control transfers, the application performs the following steps:

1. Waits for the packet received interrupt from the controller
2. Retrieves the packet from the receive buffer

Because the control transfer is governed by USB protocol, the controller always responds with an ACK handshake.

19.6. USB 2.0 OTG Controller Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

Related Information

ECC Controller Address Map and Register Descriptions on page 362

20. SPI Controller

The hard processor system (HPS) provides two serial peripheral interface (SPI) masters and two SPI slaves. The SPI masters and slaves are instances of the Synopsys DesignWare Synchronous Serial Interface (SSI) controller (DW_apb_ssi). † (50)

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

20.1. Features of the SPI Controller

The SPI controller has the following features: †

- Serial master and serial slave controllers – Enable serial communication with serial-master or serial-slave peripheral devices. †
- Each SPI master has a maximum bit rate of 60Mbps
- Each SPI slave has a maximum bit rate of 50Mbps
- Serial interface operation – Programmable choice of the following protocols:
  - Motorola SPI protocol
  - Texas Instruments Synchronous Serial Protocol
  - National Semiconductor Microwire
- DMA controller interface integrated with HPS DMA controller
- SPI master supports received serial data bit (RXD) sample delay
- Transmit and receive FIFO buffers are 256 words deep
- SPI master supports up to four slave selects
- Programmable master serial bit rate
- Programmable data item size of 4 to 16 bits
- Support for Multi-master mode

(50) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

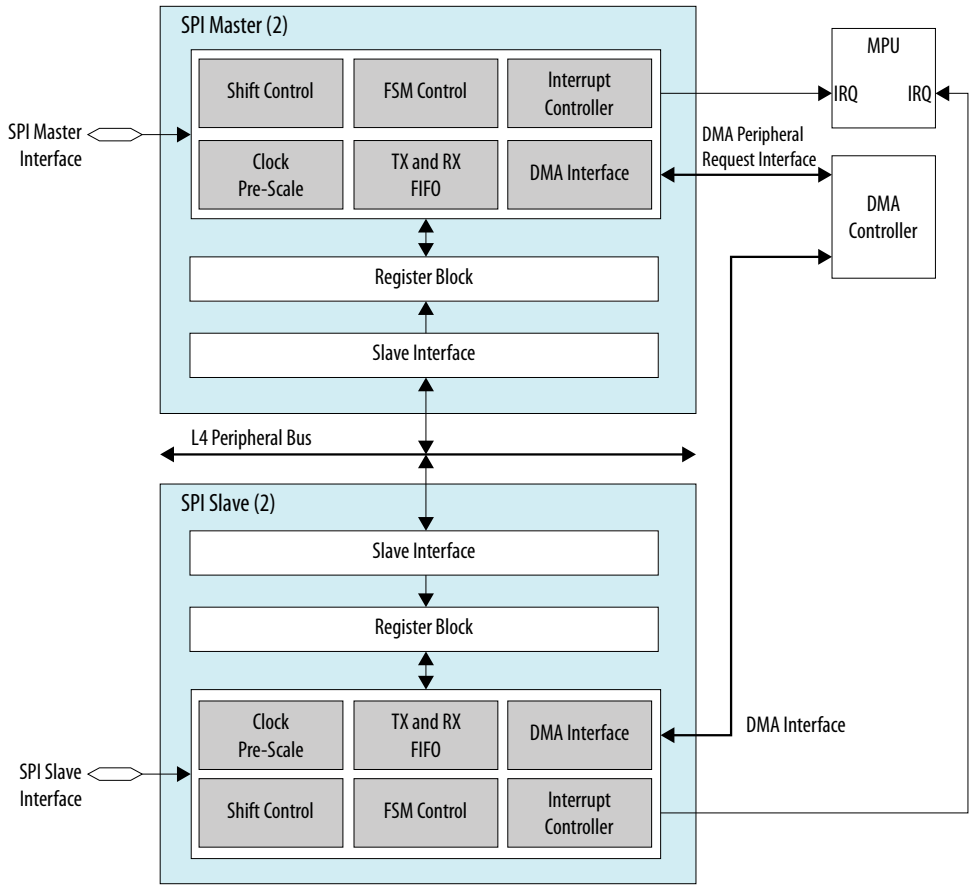
ISO 9001:2015 Registered

20.2. SPI Block Diagram and System Integration

The SPI supports data bus widths of 32 bits. †

20.2.1. SPI Block Diagram

Figure 106. SPI Block Diagram



The functional groupings of the main interfaces to the SPI block are as follows: †

- System bus interface
- DMA peripheral request interface
- Interrupt interface
- SPI interface

20.3. SPI Controller Signal Description

Signals from the two SPI masters and two SPI slaves can be routed to the FPGA or the HPS I/O pins. The following sections describe the signals available.

20.3.1. Interface to HPS I/O

Two sets of SPI Master and two sets of SPI Slave Pins are available to the HPS I/O. The pin names are shown below. For more information on routing SPI signals to HPS I/O, refer to the *HPS Component Interfaces* chapter.

Table 212. SPI Master Interface Pins

Signal Name	Signal Width	Direction	Description
CLK	1	Out	Serial clock output from the SPI master
MOSI	1	Out	Transmit data line for the SPI master
MISO	1	In	Receive data line for the SPI master
SS0_N	1	Out	Slave Select 0: Slave select signal from SPI master
SS1_N	1	Out	Slave Select 1: Slave select signal from SPI master

Table 213. SPI Slave Interface Pins

Signal Name	Signal Width	Direction	Description
CLK	1	In	Serial clock input to the SPI slave
MOSI	1	In	Receive data line for the SPI slave
MISO	1	Out	Transmit data line for the SPI slave
SS0_N	1	In	Slave select input to the SPI slave

Related Information

HPS Component Interfaces on page 741
For information about routing SPI Controller signals, refer to this chapter.

20.3.2. FPGA Routing

Two sets of SPI Master and two sets of SPI Slave Pins are available for routing to the FPGA. The signal names are shown below. For more information on routing SPI signals to the FPGA, refer to the *HPS Component Interfaces* chapter.

Table 214. SPI Master Signals for FPGA Routing

Signal Name	Signal Width	Direction	Description
spim_mosi_o	1	Out	Transmit data line for the SPI master
spim_miso_i	1	In	Receive data line for the SPI master
spim_ss_in_n	1	In	Master Contention Input
spim_mosi_oe	1	Out	Output enable for the SPI master
spim_ss0_n_o	1	Out	Slave Select 0 Slave select signal from SPI master
spim_ss1_n_o	1	Out	Slave Select 1

continued...

Signal Name	Signal Width	Direction	Description
			Allows second slave to be connected to this master
spim_ss2_n_o	1	Out	Slave Select 2 Allows third slave to be connected to this master
spim_ss3_n_o	1	Out	Slave Select 3 Allows fourth slave to be connected to this master
spim_sclk_out	1	Out	Serial clock output

Table 215. SPI Slave Signals for FPGA Routing

Signal Name	Signal Width	Direction	Description
spis_miso_o	1	Out	Transmit data line for the SPI Slave
spis_mosi_i	1	In	Receive data line for the SPI Slave
spis_ss_in_n	1	Out	Master Contention Input
spis_miso_oe	1	Out	Output enable for the SPI Slave
spis_sclk_in	1	In	Serial clock input

Related Information

HPS Component Interfaces on page 741
For information about routing SPI Controller signals, refer to this chapter.

20.4. Functional Description of the SPI Controller

20.4.1. Protocol Details and Standards Compliance

This section describes the functional operation of the SPI controller.

The host processor accesses data, control, and status information about the SPI controller through the system bus interface. The SPI also interfaces with the DMA Controller. †

The HPS includes two general-purpose SPI master controllers and two general-purpose SPI slave controllers.

The SPI controller can connect to any other SPI device using any of the following protocols:

- Motorola SPI Protocol †
- Texas Instruments Serial Protocol (SSP) †
- National Semiconductor Microwire Protocol †

20.4.2. SPI Controller Overview

In order for the SPI controller to connect to a serial-master or serial-slave peripheral device, the peripheral must have a least one of the following interfaces: †

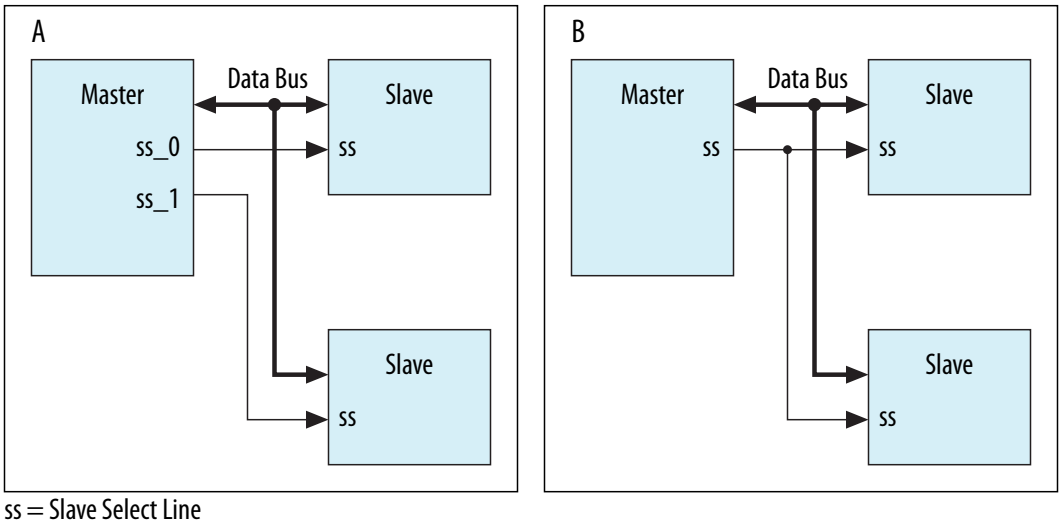
- Motorola SPI protocol – A four-wire, full-duplex serial protocol from Motorola. The slave select line is held high when the SPI controller is idle or disabled. For more information, refer to "Motorola SPI Protocol". †
- Texas Instruments Serial Protocol (SSP) – A four-wire, full-duplex serial protocol. The slave select line used for SPI and Microwire protocols doubles as the frame indicator for the SSP protocol. For more information, refer to "Texas Instruments Synchronous Serial Protocol (SSP)". †
- National Semiconductor Microwire – A half-duplex serial protocol, which uses a control word transmitted from the serial master to the target serial slave. For more information, refer to "National Semiconductor Microwire Protocol". You can program the FRF (frame format) bit field in the Control Register 0 (CTRLR0) to select which protocol is used. †

The serial protocols supported by the SPI controller allow for serial slaves to be selected or addressed using hardware. Serial slaves are selected under the control of dedicated hardware select lines. The number of select lines generated from the serial master is equal to the number of serial slaves present on the bus. The serial-master device asserts the select line of the target serial slave before data transfer begins. This architecture is illustrated in part A of Figure 107 on page 612. †

When implemented in software, the input select line for all serial slave devices should originate from a single slave select output on the serial master. In this mode it is assumed that the serial master has only a single slave select output. If there are multiple serial masters in the system, the slave select output from all masters can be logically ANDed to generate a single slave select input for all serial slave devices. †

The main program in the software domain controls selection of the target slave device; this architecture is illustrated in part B of the Figure 107 on page 612 figure below. Software would control which slave is to respond to the serial transfer request from the master device. †

Figure 107. Hardware/Software Slave Selection



Related Information

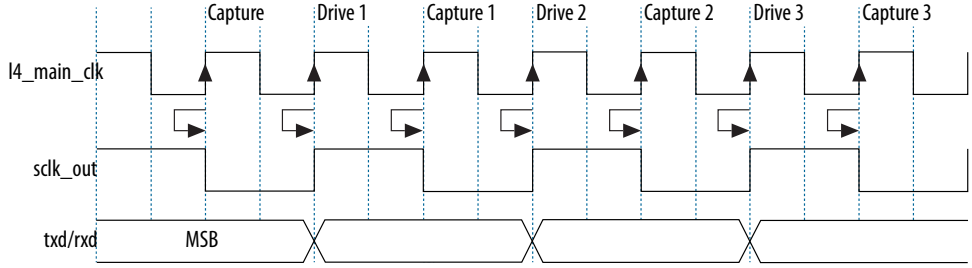
- Motorola SPI Protocol on page 624
- Texas Instruments Synchronous Serial Protocol (SSP) on page 626
- National Semiconductor Microwire Protocol on page 627

20.4.2.1. Serial Bit-Rate Clocks

20.4.2.1.1. SPI Master Bit-Rate Clock

The maximum frequency of the SPI master bit-rate clock (`sclk_out`) is one-half the frequency of SPI master clock (`l4_main_clk`). This allows the shift control logic to capture data on one clock edge of `sclk_out` and propagate data on the opposite edge. The `sclk_out` line toggles only when an active transfer is in progress. At all other times it is held in an inactive state, as defined by the serial protocol under which it operates. †

Figure 108. Maximum sclk_out/l4_main_clk Ratio



The frequency of `sclk_out` can be derived from the equation below, where <*SPI clock*> is `l4_main_clk` for both master and slave modules. †

Fsclk_out = F<*SPI clock*> / SCKDV

SCKDV is a bit field in the register BAUDR, holding any even value in the range 2 to 65,534. If SCKDV is 0, then `sclk_out` is disabled. †

The following equation describes the frequency ratio restrictions between the bit-rate clock `sclk_out` and the SPI master peripheral clock. The SPI master peripheral clock must be at least double the offchip master clock. †

Table 216. SPI Master Peripheral Clock

• SPI Master Peripheral Clock
Fl4_main_clk >= 2 x (maximum $_{Fsclk_out}$) †

20.4.2.1.2. SPI Slave Bit-Rate Clock

The minimum frequency of `l4_main_clk` depends on the operation of the slave peripheral. If the slave device is *receive only*, the minimum frequency of `l4_main_clk` is six times the maximum expected frequency of the bit-rate clock from the master device (`sclk_in`). The `sclk_in` signal is double synchronized to the `l4_main_clk` domain, and then it is edge detected; this synchronization requires three `l4_main_clk` periods. †

If the slave device is *transmit and receive*, the minimum frequency of `l4_main_clk` is eight times the maximum expected frequency of the bit-rate clock from the master device (`sclk_in`). This ensures that data on the master `rxd` line is stable before the master shift control logic captures the data. †

The frequency ratio restrictions between the bit-rate clock `sclk_in` and the SPI slave peripheral clock are as follows: †

- Slave (receive only): $F_{l4_main_clk}$ >= 6 multiply (maximum $_{Fsclk_in}$) †
- Slave: $F_{l4_main_clk}$ >= 8 multiply (maximum F_{sclk_in}) †

20.4.2.2. Transmit and Receive FIFO Buffers

There are two 16-bit FIFO buffers, a transmit FIFO buffer and a receive FIFO buffer, with a depth of 256. Data frames that are less than 16 bits in size must be right-justified when written into the transmit FIFO buffer. The shift control logic automatically right-justifies receive data in the receive FIFO buffer. †

Each data entry in the FIFO buffers contains a single data frame. It is impossible to store multiple data frames in a single FIFO buffer location; for example, you may not store two 8-bit data frames in a single FIFO buffer location. If an 8-bit data frame is required, the upper 8-bits of the FIFO buffer entry are ignored or unused when the serial shifter transmits the data. †

The transmit and receive FIFO buffers are cleared when the SPI controller is disabled (`SSIENR=0`) or reset.

The transmit FIFO buffer is loaded by write commands to the SPI data register (DR). Data are popped (removed) from the transmit FIFO buffer by the shift control logic into the transmit shift register. The transmit FIFO buffer generates a transmit FIFO empty interrupt request when the number of entries in the FIFO buffer is less than or equal to the FIFO buffer threshold value. The threshold value, set through the register TXFTLR, determines the level of FIFO buffer entries at which an interrupt is generated. The threshold value allows you to provide early indication to the processor that the transmit FIFO buffer is nearly empty. A Transmit FIFO Overflow Interrupt is generated if you attempt to write data into an already full transmit FIFO buffer. †

Data are popped from the receive FIFO buffer by read commands to the SPI data register (DR). The receive FIFO buffer is loaded from the receive shift register by the shift control logic. The receive FIFO buffer generates a receive FIFO full interrupt request when the number of entries in the FIFO buffer is greater than or equal to the FIFO buffer threshold value plus one. The threshold value, set through register RXFTLR, determines the level of FIFO buffer entries at which an interrupt is generated. †

The threshold value allows you to provide early indication to the processor that the receive FIFO buffer is nearly full. A Receive FIFO Overrun Interrupt is generated when the receive shift logic attempts to load data into a completely full receive FIFO buffer. However, the newly received data are lost. A Receive FIFO Underflow Interrupt is generated if you attempt to read from an empty receive FIFO buffer. This alerts the processor that the read data are invalid. †

Related Information

Reset Manager on page 134
For more information, refer to the *Reset Manager* chapter.

20.4.2.3. SPI Interrupts

The SPI controller supports combined interrupt requests, which can be masked. The combined interrupt request is the ORed result of all other SPI interrupts after masking. All SPI interrupts have active-high polarity level. The SPI interrupts are described as follows: †

- Transmit FIFO Empty Interrupt – Set when the transmit FIFO buffer is equal to or below its threshold value and requires service to prevent an underrun. The threshold value, set through a software-programmable register, determines the level of transmit FIFO buffer entries at which an interrupt is generated. This interrupt is cleared by hardware when data are written into the transmit FIFO buffer, bringing it over the threshold level. †
- Transmit FIFO Overflow Interrupt – Set when a master attempts to write data into the transmit FIFO buffer after it has been completely filled. When set, new data writes are discarded. This interrupt remains set until you read the transmit FIFO overflow interrupt clear register (TXOICR). †
- Receive FIFO Full Interrupt – Set when the receive FIFO buffer is equal to or above its threshold value plus 1 and requires service to prevent an overflow. The threshold value, set through a software-programmable register, determines the level of receive FIFO buffer entries at which an interrupt is generated. This interrupt is cleared by hardware when data are read from the receive FIFO buffer, bringing it below the threshold level. †
- Receive FIFO Overflow Interrupt – Set when the receive logic attempts to place data into the receive FIFO buffer after it has been completely filled. When set, newly received data are discarded. This interrupt remains set until you read the receive FIFO overflow interrupt clear register (RXOICR). †
- Receive FIFO Underflow Interrupt – Set when a system bus access attempts to read from the receive FIFO buffer when it is empty. When set, zeros are read back from the receive FIFO buffer. This interrupt remains set until you read the receive FIFO underflow interrupt clear register (RXUICR). †
- Combined Interrupt Request – ORed result of all the above interrupt requests after masking. To mask this interrupt signal, you must mask all other SPI interrupt requests. †

Transmit FIFO Overflow, Transmit FIFO Empty, Receive FIFO Full, Receive FIFO Underflow, and Receive FIFO Overflow interrupts can all be masked independently, using the Interrupt Mask Register (IMR). †

20.4.3. Transfer Modes

When transferring data on the serial bus, the SPI controller operates one of several modes. The transfer mode (TMOD) is set by writing to the TMOD field in control register 0 (CTRLR0).

Note: The transfer mode setting does not affect the duplex of the serial transfer. TMOD is ignored for Microwire transfers, which are controlled by the MWCR register. †

20.4.3.1. Transmit and Receive

When TMOD = 0, both transmit and receive logic are valid. The data transfer occurs as normal according to the selected frame format (serial protocol). Transmit data are popped from the transmit FIFO buffer and sent through the txd line to the target

device, which replies with data on the rxd line. The receive data from the target device is moved from the receive shift register into the receive FIFO buffer at the end of each data frame. †

20.4.3.2. Transmit Only

When TMOD = 1, any receive data are ignored. The data transfer occurs as normal, according to the selected frame format (serial protocol). Transmit data are popped from the transmit FIFO buffer and sent through the txd line to the target device, which replies with data on the rxd line. At the end of the data frame, the receive shift register does not load its newly received data into the receive FIFO buffer. The data in the receive shift register is overwritten by the next transfer. You should mask interrupts originating from the receive logic when this mode is entered. †

20.4.3.3. Receive Only

When TMOD = 2, the transmit data are invalid. In the case of the SPI slave, the transmit FIFO buffer is never popped in Receive Only mode. The txd output remains at a constant logic level during the transmission. The data transfer occurs as normal according to the selected frame format (serial protocol). The receive data from the target device is moved from the receive shift register into the receive FIFO buffer at the end of each data frame. You should mask interrupts originating from the transmit logic when this mode is entered. †

20.4.3.4. EEPROM Read

Note: This transfer mode is only valid for serial masters. †

When TMOD = 3, the transmit data is used to transmit an opcode, an address, or both to the EEPROM device. This takes three data frames (8-bit opcode followed by 8-bit upper address and 8-bit lower address). During the transmission of the opcode and address, no data is captured by the receive logic (as long as the SPI master is transmitting data on its txd line, data on the rxd line is ignored). The SPI master continues to transmit data until the transmit FIFO buffer is empty. You should ONLY have enough data frames in the transmit FIFO buffer to supply the opcode and address to the EEPROM. If more data frames are in the transmit FIFO buffer than are needed, then Read data is lost.

When the transmit FIFO buffer becomes empty (all control information has been sent), data on the receive line (rxd) is valid and is stored in the receive FIFO buffer; the txd output is held at a constant logic level. The serial transfer continues until the number of data frames received by the SPI master matches the value of the NDF field in the CTRLR1 register plus one. †

Note: EEPROM read mode is not supported when the SPI controller is configured to be in the SSP mode. †

20.4.4. SPI Master

The SPI master initiates and controls all serial transfers with serial-slave peripheral devices. †

The serial bit-rate clock, generated and controlled by the SPI controller, is driven out on the `sclk_out` line. When the SPI controller is disabled, no serial transfers can occur and `sclk_out` is held in "inactive" state, as defined by the serial protocol under which it operates. †

Related Information

SPI Block Diagram on page 609

20.4.4.1. RXD Sample Delay

The SPI master device is capable of delaying the default sample time of the `rxd` signal in order to increase the maximum achievable frequency on the serial bus.

Round trip routing delays on the `sclk_out` signal from the master and the `rxd` signal from the slave can mean that the timing of the `rxd` signal, as seen by the master, has moved away from the normal sampling time.

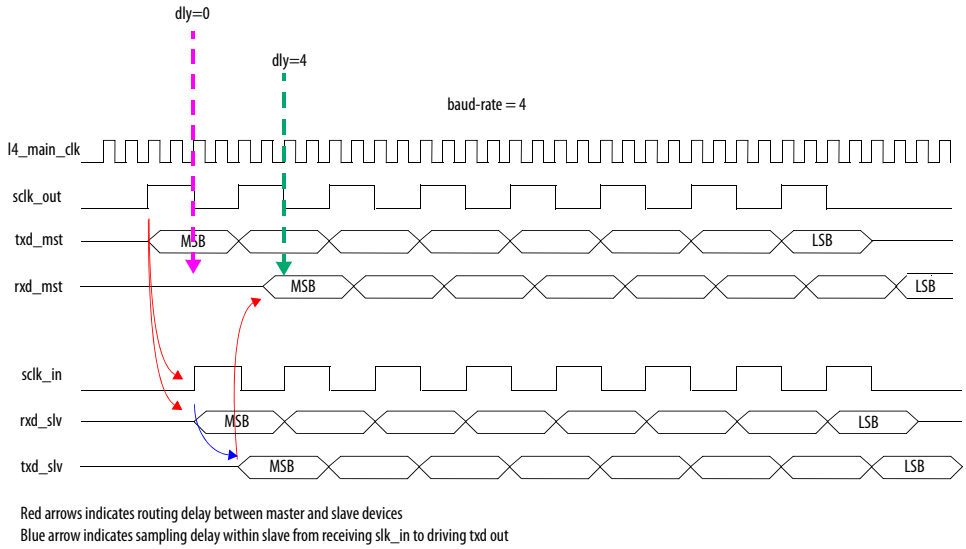
Without the RXD sample delay, you must increase the baud rate for the transfer in order to ensure that the setup times on the `rxd` signal are within range. This reduces the frequency of the serial interface.

Additional logic is included in the SPI master to delay the default sample time of the `rxd` signal. This additional logic can help to increase the maximum achievable frequency on the serial bus. †

By writing to the `rsd` field of the RXD sample delay register (`rx_sample_dly`), you specify an additional amount of delay applied to the `rxd` sample. The delay is in number of `l4_main_clk` clock cycles, with 64 maximum cycles allowed (zero is reserved). If the `rsd` field is programmed with a value exceeding 64, a zero delay is applied to the `rxd` sample.

The sample delay logic has a resolution of one `l4_main_clk` cycle. Software can "train" the serial bus by coding a loop that continually reads from the slave and increments the master's RXD sample delay value until the correct data is received by the master. †

Figure 109. Effects of Round Trip Routing Delays on sclk_out Signal



20.4.4.2. Data Transfers

The SPI master starts data transfers when all the following conditions are met:

- The SPI master is enabled
- There is at least one valid entry in the transmit FIFO buffer
- A slave device is selected

When actively transferring data, the busy flag (BUSY) in the status register (SR) is set. You must wait until the busy flag is cleared before attempting a new serial transfer. †

Note: The BUSY status is not set when the data are written into the transmit FIFO buffer. This bit gets set only when the target slave has been selected and the transfer is underway. After writing data into the transmit FIFO buffer, the shift logic does not begin the serial transfer until a positive edge of the `sclk_out` signal is present. The delay in waiting for this positive edge depends on the baud rate of the serial transfer. Before polling the BUSY status, you should first poll the Transit FIFO Empty (TFE) status (waiting for 1) or wait for (BAUDR * SPI clock) clock cycles. †

20.4.4.3. Master SPI and SSP Serial Transfers

"Motorola SPI Protocol" and "Texas Instruments Synchronous Serial Protocol (SSP)" describe the SPI and SSP serial protocols, respectively. †

When the transfer mode is "transmit and receive" or "transmit only" (TMOD = 0 or TMOD = 1, respectively), transfers are terminated by the shift control logic when the transmit FIFO buffer is empty. For continuous data transfers, you must ensure that the transmit FIFO buffer does not become empty before all the data have been transmitted. The transmit FIFO threshold level (TXFTLR) can be used to early interrupt (Transmit FIFO Empty Interrupt) the processor indicating that the transmit FIFO buffer is nearly empty. †

When the DMA is used in conjunction with the SPI master, the transmit data level (DMATDLR) can be used to early request the DMA Controller, indicating that the transmit FIFO buffer is nearly empty. The FIFO buffer can then be refilled with data to continue the serial transfer. The user may also write a block of data (at least two FIFO buffer entries) into the transmit FIFO buffer before enabling a serial slave. This ensures that serial transmission does not begin until the number of data frames that make up the continuous transfer are present in the transmit FIFO buffer. †

When the transfer mode is "receive only" (TMOD = 2), a serial transfer is started by writing one "dummy" data word into the transmit FIFO buffer when a serial slave is selected. The `txd` output from the SPI controller is held at a constant logic level for the duration of the serial transfer. The transmit FIFO buffer is popped only once at the beginning and may remain empty for the duration of the serial transfer. The end of the serial transfer is controlled by the "number of data frames" (NDF) field in control register 1 (CTRLR1). †

If, for example, you want to receive 24 data frames from a serial-slave peripheral, you should program the NDF field with the value 23; the receive logic terminates the serial transfer when the number of frames received is equal to the NDF value plus one. This transfer mode increases the bandwidth of the system bus as the transmit FIFO buffer never needs to be serviced during the transfer. The receive FIFO buffer should be read each time the receive FIFO buffer generates a FIFO full interrupt request to prevent an overflow. †

When the transfer mode is "eeprom_read" (TMOD = 3), a serial transfer is started by writing the opcode or address, or both, into the transmit FIFO buffer when a serial slave (EEPROM) is selected. The opcode and address are transmitted to the EEPROM device, after which read data is received from the EEPROM device and stored in the receive FIFO buffer. The end of the serial transfer is controlled by the NDF field in the control register 1 (CTRLR1).

Note: EEPROM read mode is not supported when the SPI controller is configured to be in the SSP mode. †

The receive FIFO threshold level (RXFTLR) can be used to give early indication that the receive FIFO buffer is nearly full. When a DMA is used, the receive data level (DMARDLR) can be used to early request the DMA Controller, indicating that the receive FIFO buffer is nearly full. †

Related Information

- Motorola SPI Protocol on page 624
- Texas Instruments Synchronous Serial Protocol (SSP) on page 626

20.4.4.4. Master Microwire Serial Transfers

"National Semiconductor Microwire Protocol" describes the Microwire serial protocol in detail. †

Microwire serial transfers from the SPI serial master are controlled by the Microwire Control Register (MWCR). The MHS bit field enables and disables the Microwire handshaking interface. The MDD bit field controls the direction of the data frame (the control frame is always transmitted by the master and received by the slave). The MWMOD bit field defines whether the transfer is sequential or nonsequential. †

All Microwire transfers are started by the SPI serial master when there is at least one control word in the transmit FIFO buffer and a slave is enabled. When the SPI master transmits the data frame (MDD =1), the transfer is terminated by the shift logic when the transmit FIFO buffer is empty. When the SPI master receives the data frame (MDD = 1), the termination of the transfer depends on the setting of the MWMOD bit field. If the transfer is nonsequential (MWMOD = 0), it is terminated when the transmit FIFO buffer is empty after shifting in the data frame from the slave. When the transfer is sequential (MWMOD = 1), it is terminated by the shift logic when the number of data frames received is equal to the value in the CTRLR1 register plus one. †

When the handshaking interface on the SPI master is enabled (MHS =1), the status of the target slave is polled after transmission. Only when the slave reports a *ready status* does the SPI master complete the transfer and clear its BUSY status. If the transfer is continuous, the next control/data frame is not sent until the slave device returns a *ready status*. †

Related Information

National Semiconductor Microwire Protocol on page 627

20.4.5. SPI Slave

The SPI slave handles serial communication with transfer initiated and controlled by serial master peripheral devices.

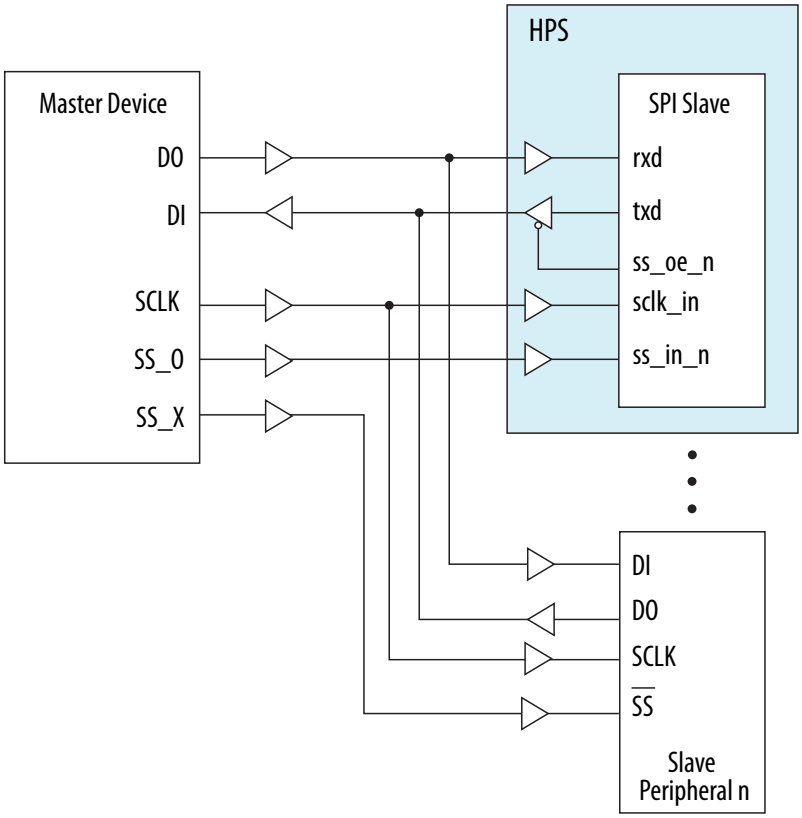
- sclk_in—serial clock to the SPI slave †
- ss_in_n—slave select input to the SPI slave †
- ss_oe_n—output enable for the SPI master or slave †
- txd—transmit data line for the SPI master or slave †
- rxd—receive data line for the SPI master or slave †

When the SPI serial slave is selected, it enables its txd data onto the serial bus. All data transfers to and from the serial slave are regulated on the serial clock line (sclk_in), driven from the SPI master device. Data are propagated from the serial slave on one edge of the serial clock line and sampled on the opposite edge. †

When the SPI serial slave is not selected, it must not interfere with data transfers between the serial-master and other serial-slave devices. When the serial slave is not selected, its txd output is buffered, resulting in a high impedance drive onto the SPI master rxd line. The buffers shown in the Figure 110 on page 621 figure are external to the SPI controller. spi_oe_n is the SPI slave output enable signal. †

The serial clock that regulates the data transfer is generated by the serial-master device and input to the SPI slave on sclk_in. The slave remains in an idle state until selected by the bus master. When not actively transmitting data, the slave must hold its txd line in a high impedance state to avoid interference with serial transfers to other slave devices. The SPI slave output enable (ss_oe_n) signal is available for use to control the txd output buffer. The slave continues to transfer data to and from the master device as long as it is selected. If the master transmits to all serial slaves, a control bit (SLV_OE) in the SPI control register 0 (CTRLR0) can be programmed to inform the slave if it should respond with data from its txd line. †

Figure 110. SPI Slave



The slv_oe bit in the control register is only valid if the SPI slave interface is routed to the FPGA. To use the SPI slave in a multi master system or in a system that requires the SPI slave TXD to be tri-stated, you can do the following:

- If you want the SPI slave to control the tri-state of TXD, it must be routed to the FPGA first and use the FPGA IO. HPS I/O can also be used via the Loan I/O interface (timing permitting).
- If you do not want to route to the FPGA, then software control of the TXD (tri-state) must be performed with the already included code to control via an HPS GPIO input. Please refer to the pin connection guidelines to find which GPIO pins correspond to which HPS SPI slave SS ports.

20.4.5.1. Slave SPI and SSP Serial Transfers

"Motorola SPI Protocol" and the "Texas Instruments Synchronous Serial Protocol (SSP)" contain a description of the SPI and SSP serial protocols, respectively. †

If the SPI slave is *receive only* (TMOD=2), the transmit FIFO buffer need not contain valid data because the data currently in the transmit shift register is resent each time the slave device is selected. The TXE error flag in the status register (SR) is not set when TMOD=2. You should mask the Transmit FIFO Empty Interrupt when this mode is used. †

If the SPI slave transmits data to the master, you must ensure that data exists in the transmit FIFO buffer before a transfer is initiated by the serial-master device. If the master initiates a transfer to the SPI slave when no data exists in the transmit FIFO buffer, an error flag (TXE) is set in the SPI status register, and the previously transmitted data frame is resent on `txd`. For continuous data transfers, you must ensure that the transmit FIFO buffer does not become empty before all the data have been transmitted. The transmit FIFO threshold level register (TXFTLR) can be used to early interrupt (Transmit FIFO Empty Interrupt) the processor, indicating that the transmit FIFO buffer is nearly empty. When a DMA Controller is used, the DMA transmit data level register (DMATDLR) can be used to early request the DMA Controller, indicating that the transmit FIFO buffer is nearly empty. The FIFO buffer can then be refilled with data to continue the serial transfer. †

The receive FIFO buffer should be read each time the receive FIFO buffer generates a FIFO full interrupt request to prevent an overflow. The receive FIFO threshold level register (RXFTLR) can be used to give early indication that the receive FIFO buffer is nearly full. When a DMA Controller is used, the DMA receive data level register (DMARDLR) can be used to early request the DMA controller, indicating that the receive FIFO buffer is nearly full. †

Related Information

- Motorola SPI Protocol on page 624
- Texas Instruments Synchronous Serial Protocol (SSP) on page 626

20.4.5.2. Serial Transfers

"National Semiconductor Microwire Protocol" describes the Microwire serial protocol in detail, including timing diagrams and information about how data are structured in the transmit and receive FIFO buffers before and after a serial transfer. The Microwire protocol operates in much the same way as the SPI protocol. There is no decode of the control frame by the SPI slave device. †

Related Information

National Semiconductor Microwire Protocol on page 627

20.4.5.3. Glue Logic for Master Port ss_in_n

When configured as a master, the SPI has an input, `ssi_in_n`, which can be used by the deciding logic in a multi-master system to disable one master when another has priority. The polarity of this signal depends on the serial protocol in use, and the protocol is dynamically selectable.

The table below lists the three protocols and the effect of `ss_in_n` on the ability of the master to transfer data. Note that for the SSP protocol the effect of `ss_in_n` is inverted with respect to the other protocols.

Protocol	ss_in_n value	Effect on Serial Transfer
Motorola SPI	1	Enabled
	0	Disabled
National Semiconductor Microwire	1	Enabled
	0	Disabled
Texas Instruments Serial Protocol (SSP)	1	Disabled
	0	Enabled

Figure 111. Arbitration Between Multiple Serial Masters

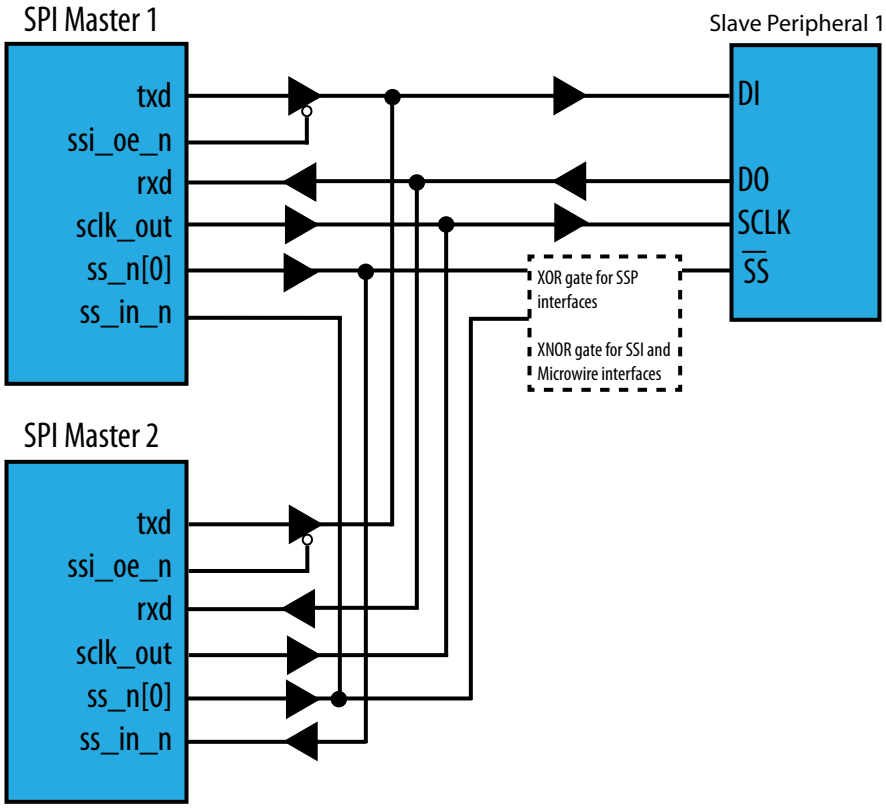
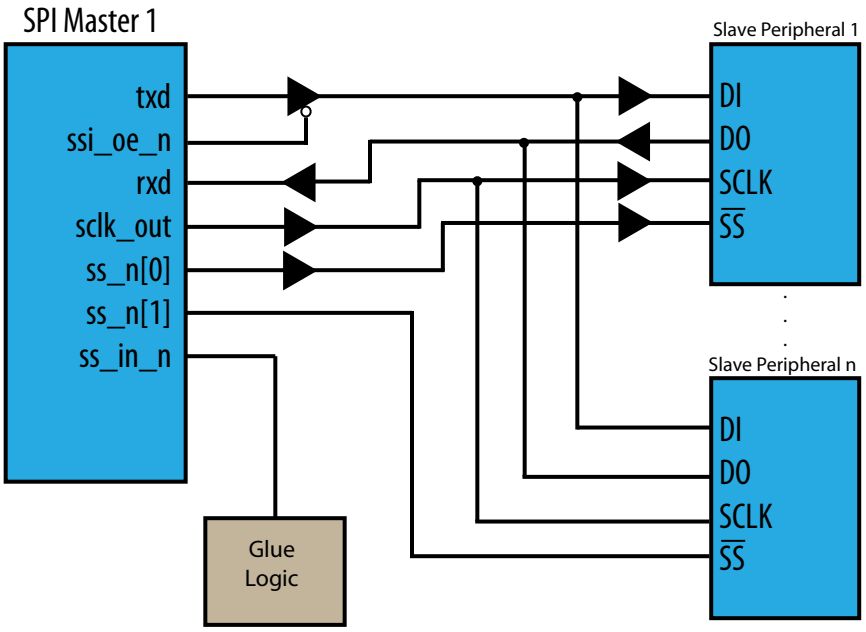


Figure 112. SPI Configured as Master Device



20.4.6. Partner Connection Interfaces

The SPI can connect to any serial-master or serial-slave peripheral device using one of several interfaces.

20.4.6.1. Motorola SPI Protocol

With SPI, the clock polarity (SCPOL) configuration parameter determines whether the inactive state of the serial clock is high or low. The data frame can be 4 to 16 bits in length. †

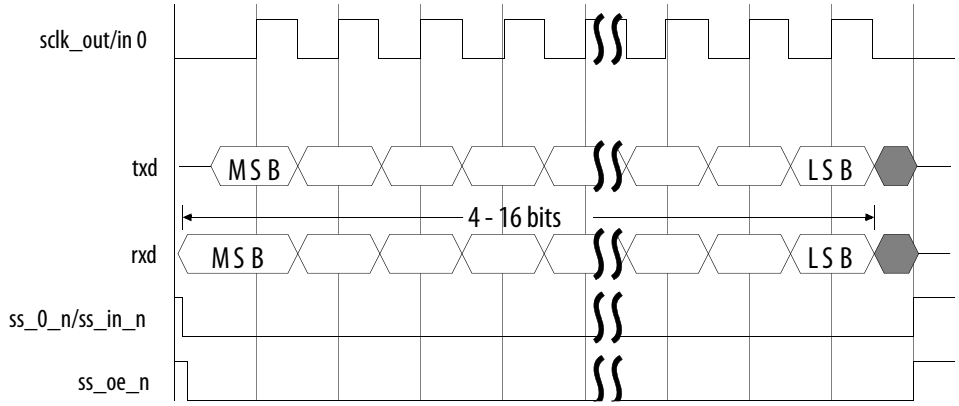
When the configuration parameter (SCPH = 0), data transmission begins on the falling edge of the slave select signal. The first data bit is captured by the master and slave peripherals on the first edge of the serial clock; therefore, valid data must be present on the `txd` and `rxd` lines prior to the first serial clock edge. †

The slave select signal takes effect only when used as slave SPI. For master SPI, the data transmission begins as soon as the output enable signal is deasserted.

The following signals are illustrated in the timing diagrams in this section: †

- `sclk_out`—serial clock from SPI master †
- `sclk_in`—serial clock from SPI slave †
- `ss_0_n`—slave select signal from SPI master †
- `ss_oe_n`—output enable for the SPI master or slave †
- `txd`—transmit data line for the SPI master or slave †
- `rxd`—receive data line for the SPI master or slave †

Figure 113. SPI Serial Format (SCPH = 0)



There are four possible transfer modes on the SPI controller for performing SPI serial transactions; refer to "Transfer Modes" . For *transmit and receive transfers* (transfer mode field (9:8) of the Control Register 0 = 0), data transmitted from the SPI controller to the external serial device is written into the transmit FIFO buffer. Data received from the external serial device into the SPI controller is pushed into the receive FIFO buffer. †

Note:

For *transmit only* transfers (transfer mode field (9:8) of the Control Register 0 = 1), data transmitted from the SPI controller to the external serial device is written into the transmit FIFO buffer. As the data received from the external serial device is deemed invalid, it is not stored in the SPI receive FIFO buffer. †

For *receive only* transfers (transfer mode field (9:8) of the Control Register 0 = 2), data transmitted from the SPI controller to the external serial device is invalid, so a single dummy word is written into the transmit FIFO buffer to begin the serial transfer. The `txd` output from the SPI controller is held at a constant logic level for the duration of the serial transfer. Data received from the external serial device into the SPI controller is pushed into the receive FIFO buffer. †

For EEPROM read transfers (transfer mode field [9:8] of the Control Register 0 = 3), the opcode or the EEPROM address, or both, are written into the transmit FIFO buffer. During transmission of these control frames, received data is not captured by the SPI master. After the control frames have been transmitted, receive data from the EEPROM is stored in the receive FIFO buffer.

SPI Serial Format

Two different modes of continuous data transfers are supported:

- When clock phase SCPH = 0 and clock polarity SCPOL = 0, the SPI Controller deasserts the slave select signal between each data word and the serial clock is held to its default value while the slave select signal is deasserted.†
- When SCPH = 1 and SCPOL = 1, the slave select is held asserted (active low) for the duration of the transfer.†

Figure 114. Serial Format Continuous Transfers (SCPH = 0 and SCPOL = 0)

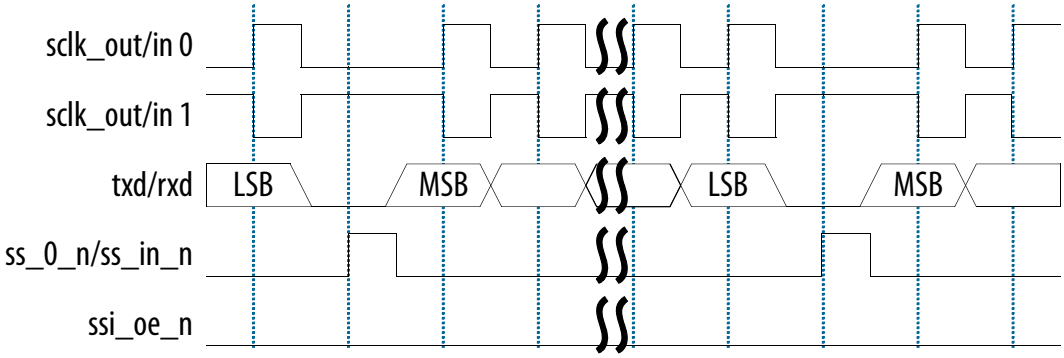
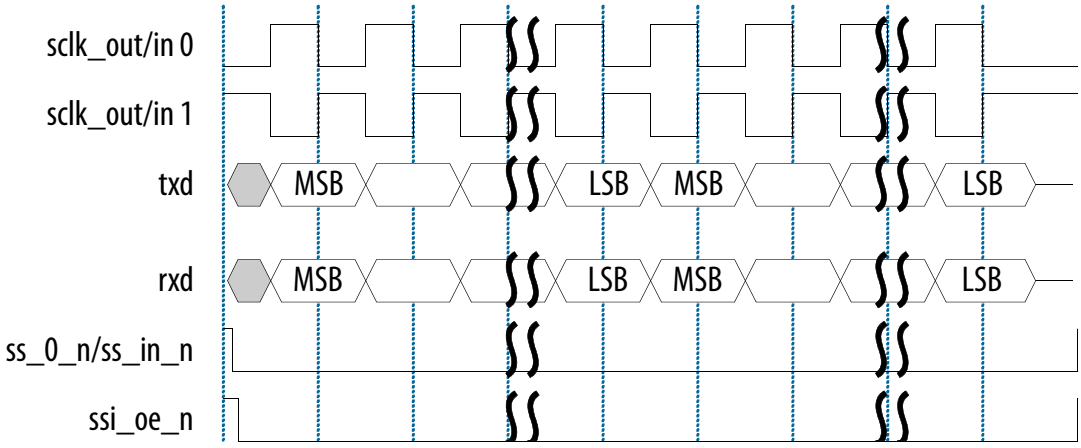


Figure 115. Serial Format Continuous Transfers (SCPH = 1 and SCPOL = 1)



Related Information

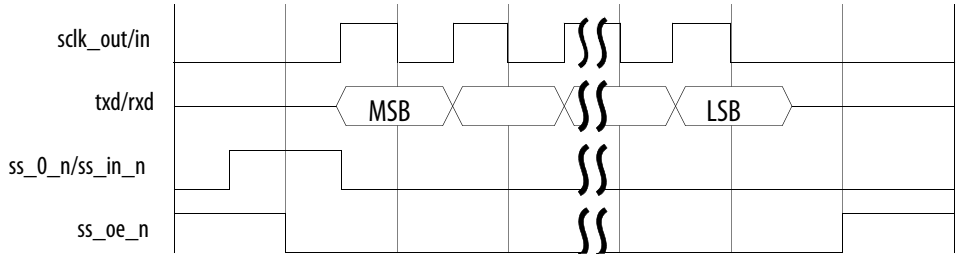
Transfer Modes on page 615

20.4.6.2. Texas Instruments Synchronous Serial Protocol (SSP)

Data transfers begin by asserting the frame indicator line (`ss_0_n`) for one serial clock period. Data to be transmitted are driven onto the `txd` line one serial clock cycle later; similarly data from the slave are driven onto the `rxd` line. Data are propagated on the rising edge of the serial clock (`sclk_out/sclk_in`) and captured on the falling edge. The length of the data frame ranges from 4 to 16 bits.

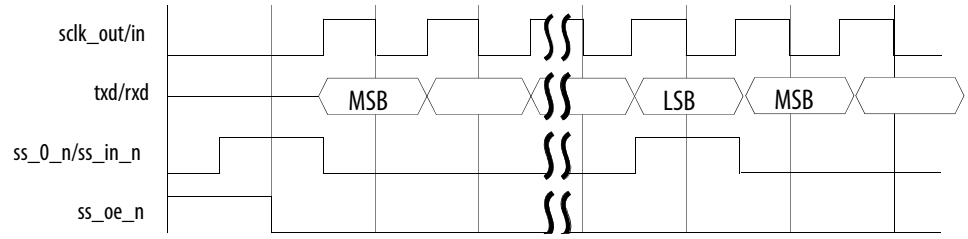
Note: The slave select signal (ss_0_n) takes effect only when used as slave SPI. For master SPI, the data transmission begins as soon as the output enable signal is deasserted.

Figure 116. SSP Serial Format



Continuous data frames are transferred in the same way as single data frames. The frame indicator is asserted for one clock period during the same cycle as the LSB from the current transfer, indicating that another data frame follows. †

Figure 117. SSP Serial Format Continuous Transfer



20.4.6.3. National Semiconductor Microwire Protocol

For the master SPI, data transmission begins as soon as the output enable signal is deasserted. One-half serial clock (sclk_out) period later, the first bit of the control is sent out on the txd line. The length of the control word can be in the range 1 to 16 bits and is set by writing bit field CFS (bits 15:12) in CTRLR0. The remainder of the control word is transmitted (propagated on the falling edge of sclk_out) by the SPI serial master. During this transmission, no data are present (high impedance) on the serial master's rxd line. †

The direction of the data word is controlled by the MDD bit field (bit 1) in the Microwire Control Register (MWCR). When MDD=0, this indicates that the SPI serial master receives data from the external serial slave. One clock cycle after the LSB of the control word is transmitted, the slave peripheral responds with a dummy 0 bit, followed by the data frame, which can be 4 to 16 bits in length. Data are propagated on the falling edge of the serial clock and captured on the rising edge. †

Continuous transfers from the Microwire protocol can be sequential or nonsequential, and are controlled by the MWMOD bit field (bit 0) in the MWCR. †

Nonsequential continuous transfers occur, with the control word for the next transfer following immediately after the LSB of the current data word. †

The only modification needed to perform a continuous nonsequential transfer is to write more control words into the transmit FIFO buffer. †

During sequential continuous transfers, only one control word is transmitted from the SPI master. The transfer is started in the same manner as with nonsequential read operations, but the cycle is continued to read further data. The slave device automatically increments its address pointer to the next location and continues to provide data from that location. Any number of locations can be read in this manner; the SPI master terminates the transfer when the number of words received is equal to the value in the CTRLR1 register plus one. †

When MDD = 1, this indicates that the SPI serial master transmits data to the external serial slave. Immediately after the LSB of the control word is transmitted, the SPI master begins transmitting the data frame to the slave peripheral. †

Note: The SPI controller does not support continuous sequential Microwire writes, where MDD = 1 and MWMOD = 1. †

Continuous transfers occur with the control word for the next transfer following immediately after the LSB of the current data word.

The Microwire handshaking interface can also be enabled for SPI master write operations to external serial-slave devices. To enable the handshaking interface, you must write 1 into the MHS bit field (bit 2) on the MWCR register. When MHS is set to 1, the SPI serial master checks for a ready status from the slave device before completing the transfer, or transmitting the next control word for continuous transfers. †

After the first data word has been transmitted to the serial-slave device, the SPI master polls the rxd input waiting for a ready status from the slave device. Upon reception of the ready status, the SPI master begins transmission of the next control word. After transmission of the last data frame has completed, the SPI master transmits a start bit to clear the ready status of the slave device before completing the transfer. †

In the SPI slave, data transmission begins with the falling edge of the slave select signal (ss_in_0). One-half serial clock (sclk_in) period later, the first bit of the control is present on the rxd line. The length of the control word can be in the range of 1 to 16 bits and is set by writing bit field CFS in the CTRLR0 register. The CFS bit field must be set to the size of the expected control word from the serial master. The remainder of the control word is received (captured on the rising edge of sclk_in) by the SPI serial slave. During this reception, no data are driven (high impedance) on the serial slave's txd line. †

The direction of the data word is controlled by the MDD bit field (bit 1) MWCR register. When MDD=0, this indicates that the SPI serial slave is to receive data from the external serial master. Immediately after the control word is transmitted, the serial master begins to drive the data frame onto the SPI slave rxd line. Data are propagated on the falling edge of the serial clock and captured on the rising edge. The slave-select signal is held active-low during the transfer and is deasserted one-half clock cycle later after the data are transferred. The SPI slave output enable signal is held inactive for the duration of the transfer. †

When MDD=1, this indicates that the SPI serial slave transmits data to the external serial master. Immediately after the LSB of the control word is transmitted, the SPI slave transmits a dummy 0 bit, followed by the 4- to 16-bit data frame on the txd line. †

Continuous transfers for a SPI slave occur in the same way as those specified for the SPI master. The SPI slave does not support the handshaking interface, as there is never a busy period. †

Figure 118. Single SPI Serial Master Microwire Serial Transfer (MDD=0)

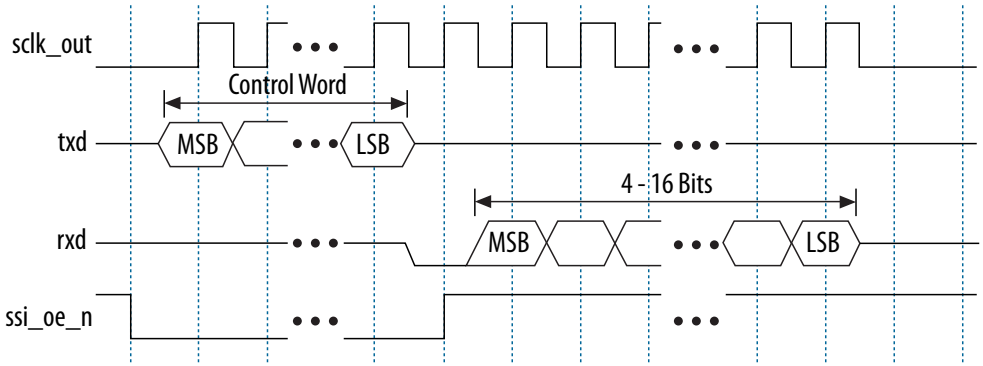
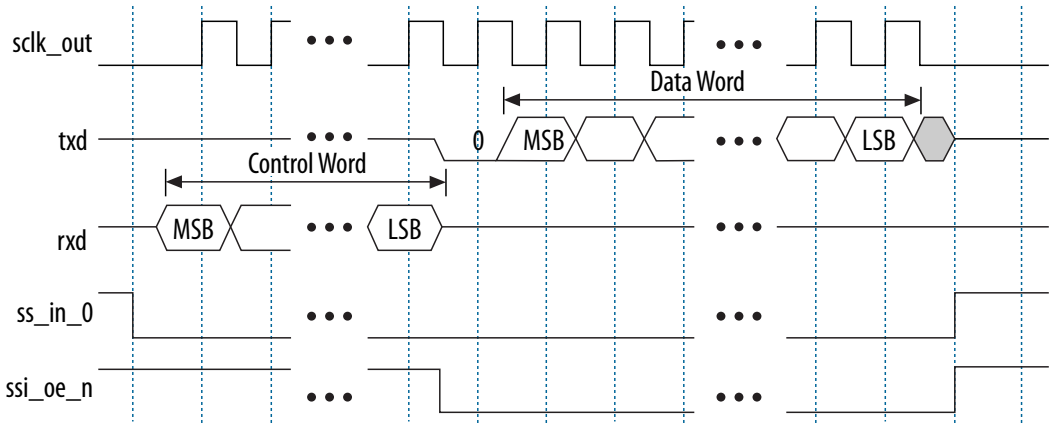


Figure 119. Single SPI Slave Microwire Serial Transfer (MDD=1)



20.4.7. DMA Controller Interface

The SPI controller supports DMA signaling to indicate when the receive FIFO buffer has data ready to be read or when the transmit FIFO buffer needs data. It requires two DMA channels, one for transmit data and one for receive data. The SPI controller can issue single or burst DMA transfers and accepts burst acknowledges from the DMA. System software can trigger the DMA burst mode by programming an appropriate value into the threshold registers. The typical setting of the threshold register value is half full.

To enable the DMA Controller interface on the SPI controller, you must write the DMA Control Register (DMACR). Writing a 1 into the TDMAE bit field of DMACR register enables the SPI transmit handshaking interface. Writing a 1 into the RDMAE bit field of the DMACR register enables the SPI receive handshaking interface. †

20.4.8. Slave Interface

The host processor accesses data, control, and status information about the SPI controller through the slave interface. The SPI supports a data bus width of 32 bits.

20.4.8.1. Control and Status Register Access

Control and status registers within the SPI controller are byte-addressable. The maximum width of the control or status register in the SPI controller is 16 bits. Therefore, all read and write operations to the SPI control and status registers require only one access. †

20.4.8.2. Data Register Access

The data register (DR) within the SPI controller is 16 bits wide in order to remain consistent with the maximum serial transfer size (data frame). A write operation to DR moves data from the slave write data bus into the transmit FIFO buffer. An read operation from DR moves data from the receive FIFO buffer onto the slave readback data bus. †

Note: The DR register in the SPI controller occupies sixty-four 32-bit locations of the memory map to facilitate burst transfers. There are no burst transactions on the system bus itself, but SPI supports bursts on the system interconnect. Writing to any of these address locations has the same effect as pushing the data from the slave write data bus into the transmit FIFO buffer. Reading from any of these locations has the same effect as popping data from the receive FIFO buffer onto the slave readback data bus. The FIFO buffers on the SPI controller are not addressable.

20.4.9. Clocks and Resets

The SPI controller uses the clock and reset signals shown in the following table.

Table 217. SPI Controller Clocks and Resets

	Master	Slave
SPI clock	`l4_main_clk`	`l4_main_clk`
SPI bit-rate clock	`sclk_out`	`sclk_in`
Reset	`spim_rst_n`	`spis_rst_n`

20.4.9.1. Clock Gating

You can locally clock gate the `l4_main_clk` to the Master SPI by programming the `spimclken` bit of the en register in the `perpllgroup`.

Note: This option is not available for SPI slaves.

Related Information

Clock Manager Address Map and Register Definitions on page 67

20.4.9.2. Taking the SPI Controller Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

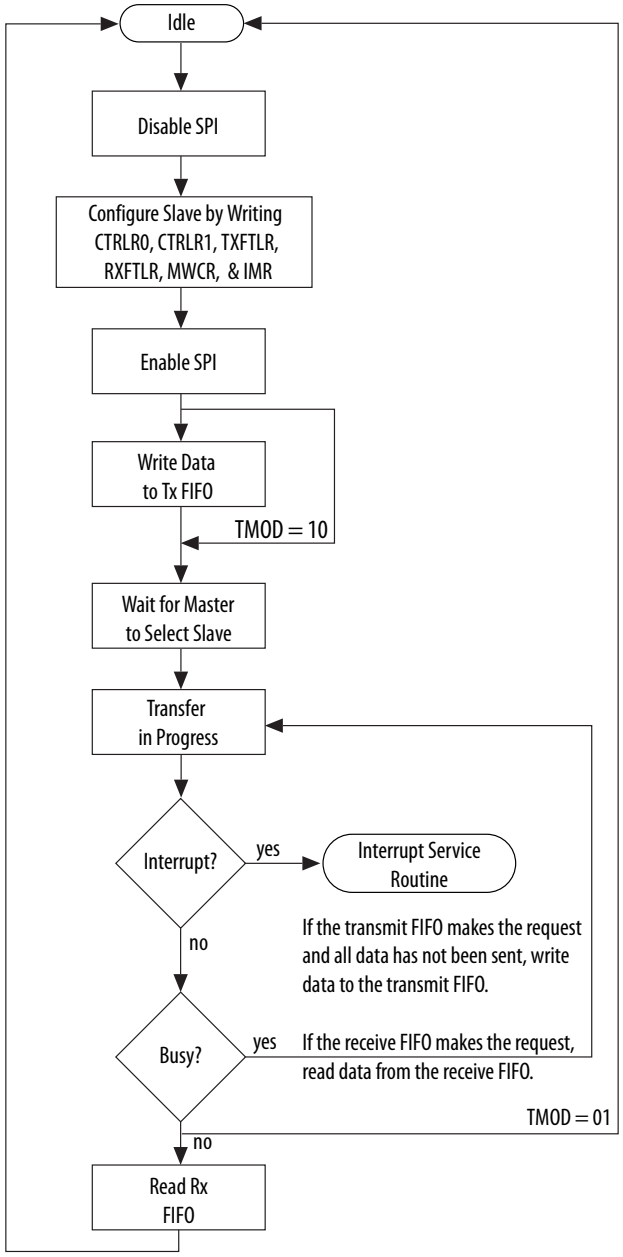
20.5. SPI Programming Model

This section describes the programming model for the SPI controller, for the following master and slave transfer types:

- Master SPI and SSP serial transfers
- Master Microwire serial transfers
- Slave SPI and SSP serial transfers
- Slave Microwire serial transfers
- Software Control for slave selection

20.5.1. Master SPI and SSP Serial Transfers

Figure 120. Master SPI or SSP Serial Transfer Software Flow



To complete an SPI or SSP serial transfer from the SPI master, follow these steps:

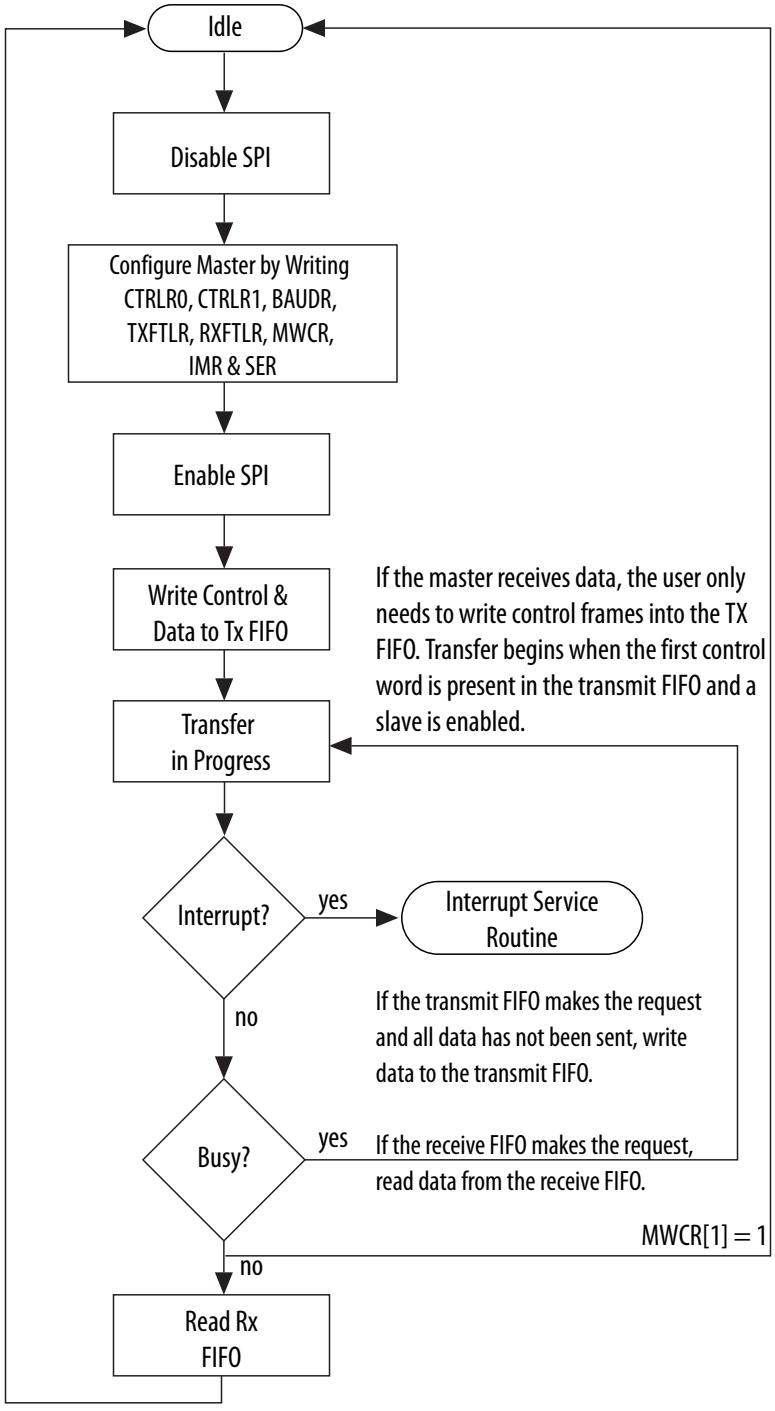
1. If the SPI master is enabled, disable it by writing 0 to the SSI Enable register (SSIENR).
2. Set up the SPI master control registers for the transfer. You can set these registers in any order.

- Write Control Register 0 (CTRLLR0). For SPI transfers, you must set the serial clock polarity and serial clock phase parameters identical to the target slave device.
- If the transfer mode is receive only, write Control Register 1 (CTRLR1) with the number of frames in the transfer minus 1. For example, if you want to receive four data frames, write 3 to this register.
- Write the Baud Rate Select Register (BAUDR) to set the baud rate for the transfer.
- Write the Transmit and Receive FIFO Threshold Level registers (TXFTLR and RXFTLR) to set FIFO buffer threshold levels.
- Write the IMR register to set up interrupt masks.
- Write the Slave Enable Register (SER) register to enable the target slave for selection. If a slave is enabled at this time, the transfer begins as soon as one valid data entry is present in the transmit FIFO buffer. If no slaves are enabled prior to writing to the Data Register (DR), the transfer does not begin until a slave is enabled.

3. Enable the SPI master by writing 1 to the SSIENR register.
4. Write data for transmission to the target slave into the transmit FIFO buffer (write DR). If no slaves were enabled in the SER register at this point, enable it now to begin the transfer.
5. Poll the BUSY status to wait for the transfer to complete. If a transmit FIFO empty interrupt request is made, write the transmit FIFO buffer (write DR). If a receive FIFO full interrupt request is made, read the receive FIFO buffer (read DR).
6. The shift control logic stops the transfer when the transmit FIFO buffer is empty. If the transfer mode is receive only (TMOD = 2), the shift control logic stops the transfer when the specified number of frames have been received. When the transfer is done, the BUSY status is reset to 0.
7. If the transfer mode is not transmit only (TMOD is not equal to 1), read the receive FIFO buffer until it is empty
8. Disable the SPI master by writing 0 to SSIENR.

20.5.2. Master Microwire Serial Transfers

Figure 121. Microwire Serial



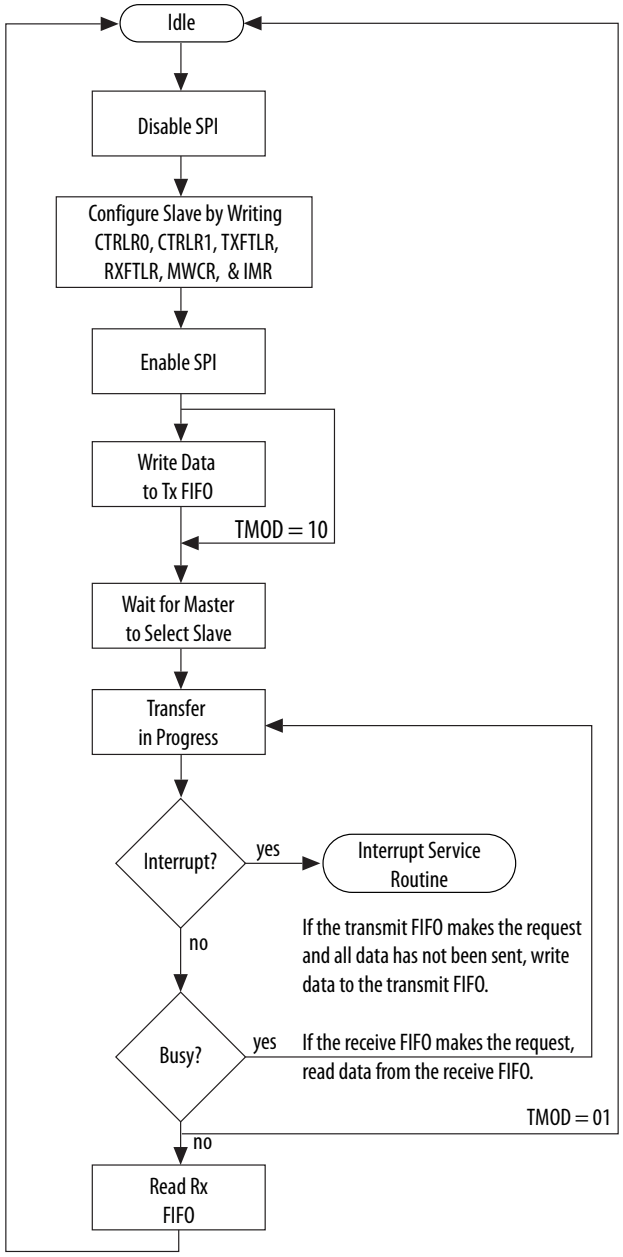
To complete a Microwire serial transfer from the SPI master, follow these steps:

1. If the SPI master is enabled, disable it by writing 0 to SSIENR.
2. Set up the SPI control registers for the transfer. You can set these registers in any order.
   - Write CTRLR0 to set transfer parameters. If the transfer is sequential and the SPI master receives data, write CTRLR1 with the number of frames in the transfer minus 1. For example, if you want to receive four data frames, write 3 to this register.
   - Write BAUDR to set the baud rate for the transfer.
   - Write TXFTLR and RXFTLR to set FIFO buffer threshold levels.
   - Write the IMR register to set up interrupt masks.

   You can write the SER register to enable the target slave for selection. If a slave is enabled here, the transfer begins as soon as one valid data entry is present in the transmit FIFO buffer. If no slaves are enabled prior to writing to the DR register, the transfer does not begin until a slave is enabled.
3. Enable the SPI master by writing 1 to the SSIENR register.
4. If the SPI master transmits data, write the control and data words into the transmit FIFO buffer (write DR). If the SPI master receives data, write the control word or words into the transmit FIFO buffer. If no slaves were enabled in the SER register at this point, enable now to begin the transfer.
5. Poll the BUSY status to wait for the transfer to complete. If a transmit FIFO empty interrupt request is made, write the transmit FIFO buffer (write DR). If a receive FIFO full interrupt request is made, read the receive FIFO buffer (read DR).
6. The shift control logic stops the transfer when the transmit FIFO buffer is empty. If the transfer mode is sequential and the SPI master receives data, the shift control logic stops the transfer when the specified number of data frames is received. When the transfer is done, the BUSY status is reset to 0.
7. If the SPI master receives data, read the receive FIFO buffer until it is empty.
8. Disable the SPI master by writing 0 to SSIENR.

20.5.3. Slave SPI and SSP Serial Transfers

Figure 122. Slave SPI or SSP Serial Transfer Software Flow



To complete a continuous serial transfer from a serial master to the SPI slave, follow these steps:

1. If the SPI slave is enabled, disable it by writing 0 to SSIENR.
2. Set up the SPI control registers for the transfer. You can set these registers in any order.

- Write CTRLR0 (for SPI transfers, set SCPH and SCPOL identical to the master device).
- Write TXFTLR and RXFTLR to set FIFO buffer threshold levels.
- Write the IMR register to set up interrupt masks.

3. Enable the SPI slave by writing 1 to the SSIENR register.
4. If the transfer mode is transmit and receive (TMOD= 0) or transmit only (TMOD= 1), write data for transmission to the master into the transmit FIFO buffer (write DR). If the transfer mode is receive only (TMOD= 2), you need not write data into the transmit FIFO buffer. The current value in the transmit shift register is retransmitted.
5. The SPI slave is now ready for the serial transfer. The transfer begins when a serial-master device selects the SPI slave.
6. When the transfer is underway, the BUSY status can be polled to return the transfer status. If a transmit FIFO empty interrupt request is made, write the transmit FIFO buffer (write DR). If a receive FIFO full interrupt request is made, read the receive FIFO buffer (read DR).
7. The transfer ends when the serial master removes the select input to the SPI slave. When the transfer is completed, the BUSY status is reset to 0.
8. If the transfer mode is not transmit only (TMOD != 1), read the receive FIFO buffer until empty.
9. Disable the SPI slave by writing 0 to SSIENR.

20.5.4. Slave Microwire Serial Transfers

For the SPI slave, the Microwire protocol operates in much the same way as the SPI protocol. The SPI slave does not decode the control frame.

20.5.5. Software Control for Slave Selection

When using software to select slave devices, the input select lines from serial slave devices is connected to a single slave select output on the SPI master.

20.5.5.1. Example: Slave Selection Software Flow for SPI Master

1. If the SPI master is enabled, disable it by writing 0 to SSIENR.
2. Write CTRLR0 to match the required transfer.
3. If the transfer is receive only, write the number of frames into CTRLR1.
4. Write BAUDR to set the transfer baud rate.
5. Write TXFTLR and RXFTLR to set FIFO buffer threshold levels.
6. Write IMR register to set interrupt masks.
7. Write SER register bit 0 to 1 to select slave 1 in this example.
8. Write SSIENR register bit 0 to 1 to enable SPI master.

20.5.5.2. Example: Slave Selection Software Flow for SPI Slave

1. If the SPI slave is enabled, disable it by writing 0 to SSIENR.
2. Write CTRLR0 to match the required transfer.
3. Write TXFTLR and RXFTLR to set FIFO buffer threshold levels.
4. Write IMR register to set interrupt masks.
5. Write SSIENR register bit 0 to 1 to enable SPI slave.
6. If the SPI slave transmits data, write data into TX FIFO buffer.

   Note: All other SPI slaves are disabled (SSIENR = 0) and therefore does not respond to an active level on their ss_in_n port.

The FIFO buffer depth (FIFO_DEPTH) for both the RX and TX buffers in the SPI controller is 256 entries.

20.5.6. DMA Controller Operation

To enable the DMA controller interface on the SPI controller, you must write the DMA Control Register (DMACR). Writing a 1 to the TDMAE bit field of DMACR register enables the SPI controller transmit handshaking interface. Writing a 1 to the RDMAE bit field of the DMACR register enables the SPI controller receive handshaking.†

Related Information

DMA Controller on page 512
For details about the DMA controller, refer to the *DMA Controller* chapter.

20.5.6.1. Transmit FIFO Buffer Underflow

During SPI serial transfers, transmit FIFO buffer requests are made to the DMA Controller whenever the number of entries in the transmit FIFO buffer is less or equal to the value in DMA Transmit Data Level Register (DMATDLR); also known as the watermark level. The DMA Controller responds by writing a burst of data to the transmit FIFO buffer, of length specified as DMA burst length.†

Note: Data should be fetched from the DMA often enough for the transmit FIFO buffer to perform serial transfers continuously, that is, when the FIFO buffer begins to empty, another DMA request should be triggered. Otherwise, the FIFO buffer runs out of data (underflow). To prevent this condition, you must set the watermark level correctly.†

Related Information

DMA Controller on page 512
For details about the DMA burst length microcode setup, refer to the *DMA Controller* chapter.

20.5.6.2. Transmit FIFO Watermark

Therefore, the goal in choosing a watermark level is to minimize the number of transactions per block, while at the same time keeping the probability of an underflow condition to an acceptable level. In practice, this is a function of the ratio of the rate at which the SPI transmits data to the rate at which the DMA can respond to destination burst requests. †

20.5.6.2.1. Example 1: Transmit FIFO Watermark Level = 64

Consider the example where the assumption is made: †

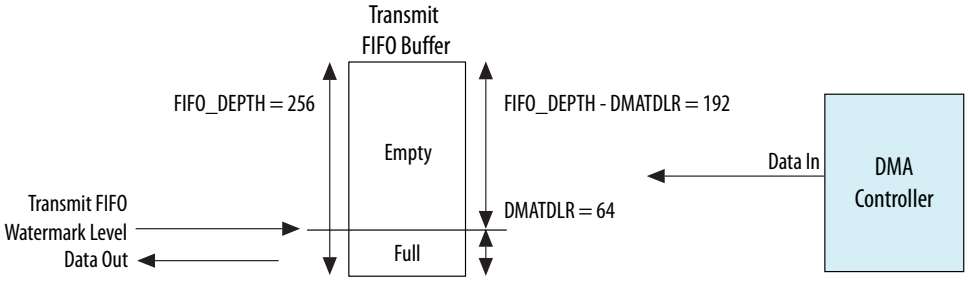
DMA burst length = FIFO_DEPTH - DMATDLR

Here the number of data items to be transferred in a DMA burst is equal to the empty space in the transmit FIFO buffer.

Consider the following:

- Transmit FIFO watermark level = DMATDLR = 64 †
- DMA burst length = FIFO_DEPTH - DMATDLR = 192 †
- SPI transmit FIFO_DEPTH = 256 †
- Block transaction size = 960 †

Figure 123. Transmit FIFO Watermark Level = 64



The number of burst transactions needed equals the block size divided by the number of data items per burst:

Block transaction size/DMA burst length = 960/192 = 5

The number of burst transactions in the DMA block transfer is 5. But the watermark level, DMATDLR, is quite low. Therefore, there is a high probability that the SPI serial transmit line needs to transmit data when there is no data left in the transmit FIFO buffer. This is a transmit underflow condition. This occurs because the DMA has not had time to service the DMA request before the FIFO buffer becomes empty.

20.5.6.2.2. Example 2: Transmit FIFO Watermark Level = 192

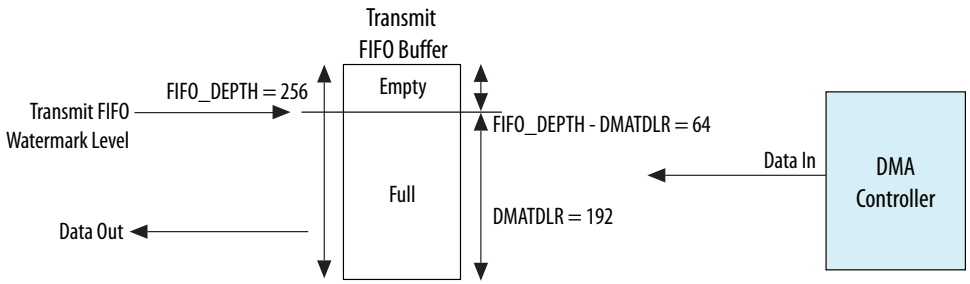
Consider the example where the assumption is made: †

DMA burst length = FIFO_DEPTH - DMATDLR

Here the number of data items to be transferred in a DMA burst is equal to the empty space in the transmit FIFO buffer. Consider the following:

- Transmit FIFO watermark level = DMATDLR = 192 †
- DMA burst length = FIFO_DEPTH - DMATDLR = 64 †
- SPI transmit FIFO_DEPTH = 256 †
- Block transaction size = 960 †

Figure 124. Transmit FIFO Watermark Level = 192



Number of burst transactions in block: †

Block transaction size/DMA burst length = 960/64 = 15 †

In this block transfer, there are 15 destination burst transactions in a DMA block transfer. But the watermark level, DMATDLR, is high. Therefore, the probability of SPI transmit underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the SPI transmit FIFO buffer becomes empty. †

This case has a lower probability of underflow at the expense of more burst transactions per block. This provides a potentially greater amount of bursts per block and worse bus utilization than the former case.

20.5.6.3. Transmit FIFO Buffer Overflow

Setting the DMA transaction burst length to a value greater than the watermark level that triggers the DMA request may cause overflow when there is not enough space in the transmit FIFO buffer to service the destination burst request. Therefore, the following equation must be adhered to in order to avoid overflow: †

DMA burst length <= FIFO_DEPTH - DMATDLR

In Example 2: Transmit Watermark Level = 192, the amount of space in the transmit FIFO buffer at the time of the burst request is made is equal to the DMA burst length. Thus, the transmit FIFO buffer may be full, but not overflowed, at the completion of the burst transaction.

Therefore, for optimal operation, DMA burst length should be set at the FIFO buffer level that triggers a transmit DMA request; that is: †

DMA burst length = FIFO_DEPTH - DMATDLR

Adhering to this equation reduces the number of DMA bursts needed for block transfer, and this in turn improves bus utilization. †

The transmit FIFO buffer is not be full at the end of a DMA burst transfer if the SPI controller has successfully transmitted one data item or more on the serial transmit line during the transfer. †

Related Information

Transmit FIFO Watermark on page 638

20.5.6.4. Receive FIFO Buffer Overflow

During SPI serial transfers, receive FIFO buffer requests are made to the DMA whenever the number of entries in the receive FIFO buffer is at or above the DMA Receive Data Level Register, that is DMARDLR + 1.This is known as the watermark level. The DMA responds by fetching a burst of data from the receive FIFO buffer. †

Data should be fetched by the DMA often enough for the receive FIFO buffer to accept serial transfers continuously, that is, when the FIFO buffer begins to fill, another DMA transfer is requested. Otherwise the FIFO buffer fills with data (overflow). To prevent this condition, the user must set the watermark level correctly. †

20.5.6.5. Choosing Receive Watermark Level

Similar to choosing the transmit watermark level, the receive watermark level, DMATDLR + 1, should be set to minimize the probability of overflow. It is a trade off between the number of DMA burst transactions required per block versus the probability of an overflow occurring. †

Related Information

Texas Instruments Synchronous Serial Protocol (SSP) on page 626

20.5.6.6. Receive FIFO Buffer Underflow

Setting the source transaction burst length greater than the watermark level can cause underflow where there is not enough data to service the source burst request. Therefore, the following equation must be adhered to avoid underflow: †

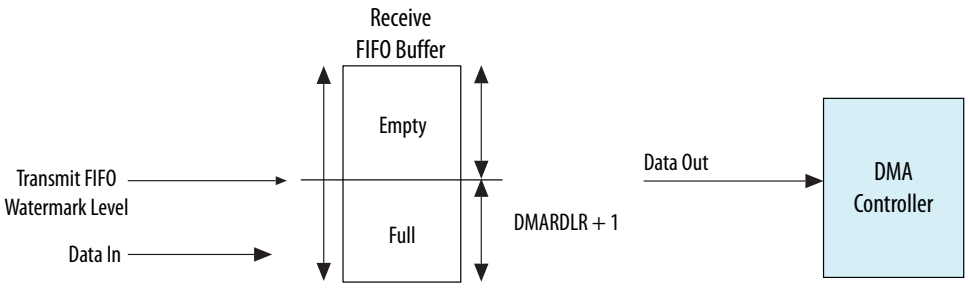
DMA burst length = DMATDLR + 1

If the number of data items in the receive FIFO buffer is equal to the source burst length at the time of the burst request is made, the receive FIFO buffer may be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA burst length should be set at the watermark level,DMATDLR + 1. †

Adhering to this equation reduces the number of DMA bursts in a block transfer, which in turn can improve bus utilization. †

Note: The receive FIFO buffer is not be empty at the end of the source burst transaction if the SPI controller has successfully received one data item or more on the serial receive line during the burst. †

Figure 125. Receive FIFO Buffer



20.6. SPI Controller Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

21. I²C Controller

The I²C controller provides support for a communication link between integrated circuits on a board. It is a simple two-wire bus which consists of a serial data line (SDA) and a serial clock (SCL) for use in applications such as temperature sensors and voltage level translators to EEPROMs, A/D and D/A converters, CODECs, and many types of microprocessors. †

The hard processor system (HPS) provides five I²C controllers to enable system software to communicate serially with I²C buses. Each I²C controller can operate in master or slave mode, and support standard mode of up to 100 Kbps or fast mode of up to 400 Kbps. These I²C controllers are instances of the Synopsys DesignWare APB I²C (DW_apb_i2c) controller.

Each I²C controller must be programmed to operate in either master or slave mode only. Operating as a master and slave simultaneously is not supported. † (51)

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

21.1. Features of the I²C Controller

The I²C controller has the following features:

- Maximum clock speed of up to 400 Kbps
- Standard clock speed 100 kbps
- One of the following I²C operations:
  - A master in an I²C system and programmed only as a master †
  - A slave in an I²C system and programmed only as a slave †
- 7- or 10-bit addressing †
- Mixed read and write combined-format transactions in both 7-bit and 10-bit addressing mode †

(51) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Bulk transmit mode †
- Transmit and receive buffers †
- Handles bit and byte waiting at all bus speeds †
- DMA handshaking interface †

Three of the five I²Cs, provide support for EMAC communication. They provide flexibility for the EMACs to use MDIO or I²C for PHY communication and can also be used as general purpose.

- `I2C_EMAC0`
- `I2C_EMAC1`
- `I2C_EMAC2`

The remaining two I²Cs are intended for general purpose.

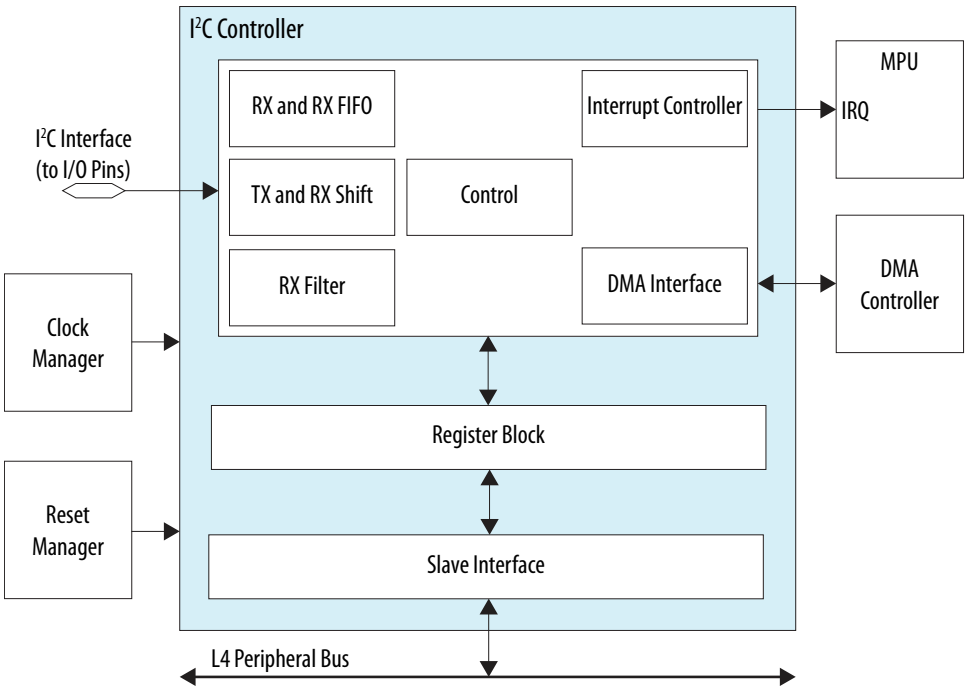
- `I2C0`
- `I2C1`

21.2. I²C Controller Block Diagram and System Integration

The I²C controller consists of an internal slave interface, an I²C interface, and FIFO logic to buffer data between the two interfaces. †

The host processor accesses data, control, and status information about the I²C controller through a 32-bit slave interface.

Figure 126. I²C Controller Block Diagram



The I²C controller consists of the following modules and interfaces:

- Slave interface for control and status register (CSR) accesses and DMA transfers, allowing a master to access the CSRs and the DMA to read or write data directly.
- Two FIFO buffers for transmit and receive data, which hold the Rx FIFO and Tx FIFO buffer register banks and controllers, along with their status levels. †
- Shift logic for parallel-to-serial and serial-to-parallel conversion
  - Rx shift – Receives data into the design and extracts it in byte format. †
  - Tx shift – Presents data supplied by CPU for transfer on the I²C bus. †
- Control logic responsible for implementing the I²C protocol.
- DMA interface that generates handshaking signals to the DMA controller in order to automate the data transfer without CPU intervention. †
- Interrupt controller that generates raw interrupts and interrupt flags, allowing them to be set and cleared. †
- Receive filter for detecting events, such as start and stop conditions, in the bus; for example, start, stop and arbitration lost. †

21.3. I²C Controller Signal Description

All instances of the I²C controller connect to external pins through pin multiplexers. Pin multiplexing allows all instances to function simultaneously and independently. The pins must be connected to a pull-up resistors and the I²C bus capacitance cannot exceed 400 pF.

There are five instances of the I²C which can be routed to the HPS I/O pins. Three of these I²C modules can be used for PHY management by the three Ethernet Media Access Controllers within the HPS. For more information about routing the I²C signals to the FPGA and HPS I/O pins, refer to the *HPS Component Interfaces* chapter.

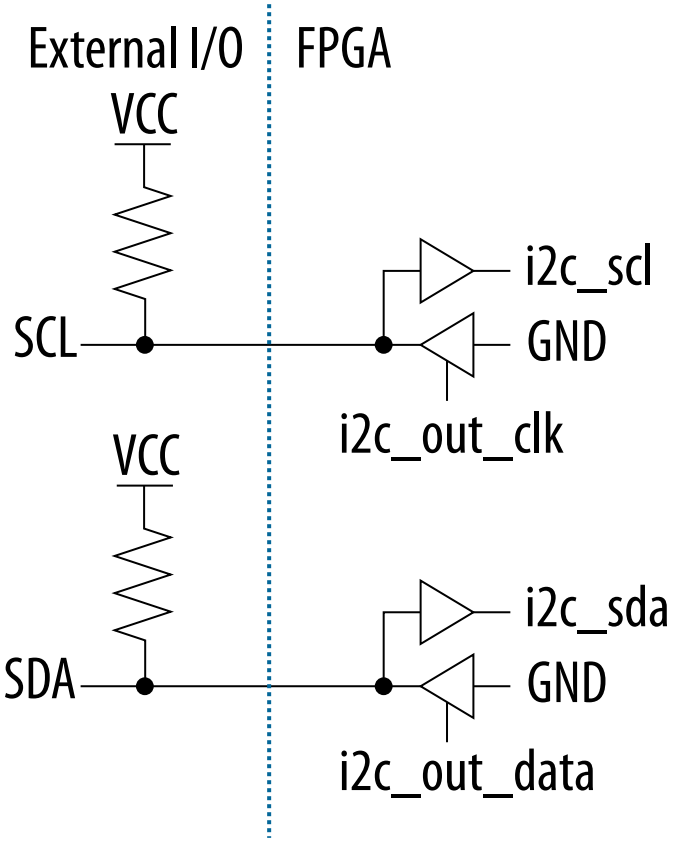
Table 218. I²C Controller Interface HPS I/O Pins

Pin Name	Signal Width	Direction	Description
SCL	1 bit	Bidirectional	Serial clock
SDA	1 bit	Bidirectional	Serial data

Table 219. HPS I²C Signals for FPGA Routing

Signal Name	Signal Width	Direction	Description
`i2c<#>_scl`	1 bit	Input	Incoming I²C clock source. This is the input SCL signal
`i2c<#>_out_clk`	1 bit	Output	Outgoing I²C clock enable. Output SCL signal. This signal is logically inverted and is synchronous to the HPS peripheral clock
`i2c<#>_sda`	1 bit	Input	Incoming I²C data. This is the input SDA signal.
`i2c<#>_out_data`	1 bit	Output	Outgoing I²C data enable. Output SDA signal. This signal is logically inverted and is synchronous to the HPS peripheral clock.

Figure 127. I2C Interface in FPGA Fabric



The figure above shows the typical connection on the I2C interface in FPGA fabric with `alt_iobuff`.

For both I2C clock and data, external IO pins are open drain connection. When output enables `i2c_out_data` and `i2c_out_clk` are asserted, external signal will be driven to ground.

Related Information

- HPS Component Interfaces on page 743
  For more information on how the I2C Signals are routed to the FPGA and HPS I/O pins, refer to this chapter.
- I/O Buffer (ALTIOBUF)
  For more information on configuring open drain I/O buffer to connect I2C signals to external I/O pins, please refer to Altera I/O Buffer (ALTIOBUF) Megafunction User Guide.

21.4. Functional Description of the I²C Controller

21.4.1. Feature Usage

The I²C controller can operate in standard mode (with data rates of up to 100 Kbps) or fast mode (with data rates less than or equal to 400 Kbps). Additionally, fast mode devices are downward compatible. For instance, fast mode devices can communicate with standard mode devices in 0 to 100 Kbps I²C bus system. However, standard mode devices are not upward compatible and should not be incorporated in a fast-mode I²C bus system as they cannot follow the higher transfer rate and therefore unpredictable states would occur. †

You can attach any I²C controller to an I²C-bus and every device can talk with any master, passing information back and forth. There needs to be at least one master (such as a microcontroller or DSP) on the bus and there can be multiple masters, which require them to arbitrate for ownership. Multiple masters and arbitration are explained later in this chapter. †

21.4.2. Behavior

You can control the I²C controller via software to be in either mode:

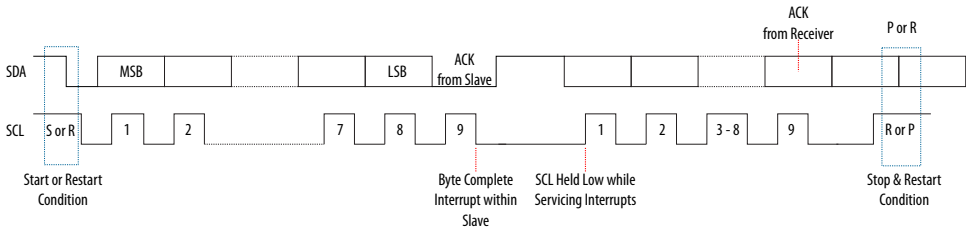
- An I²C master only, communicating with other I²C slaves.
- An I²C slave only, communicating with one or more I²C masters.

The master is responsible for generating the clock and controlling the transfer of data. The slave is responsible for either transmitting or receiving data to/from the master. The acknowledgement of data is sent by the device that is receiving data, which can be either a master or a slave. As mentioned previously, the I²C protocol also allows multiple masters to reside on the I²C bus and uses an arbitration procedure to determine bus ownership. †

Each slave has a unique address that is determined by the system designer. When a master wants to communicate with a slave, the master transmits a START/RESTART condition that is then followed by the slave's address and a control bit (R/W) to determine if the master wants to transmit data or receive data from the slave. The slave then sends an acknowledge (ACK) pulse after the address. †

If the master (master-transmitter) is writing to the slave (slave-receiver), the receiver receives one byte of data. This transaction continues until the master terminates the transmission with a STOP condition. If the master is reading from a slave (master-receiver), the slave transmits (slave-transmitter) a byte of data to the master, and the master then acknowledges the transaction with an ACK pulse. This transaction continues until the master terminates the transmission by not acknowledging (NACK) the transaction after the last byte is received, and then the master issues a STOP condition or addresses another slave after issuing a RESTART condition. †

Figure 128. Data Transfer on the I²C Bus †



The I²C controller is a synchronous serial interface. The SDA line is a bidirectional signal and changes only while the SCL line is low, except for STOP, START, and RESTART conditions. The output drivers are open-drain or open-collector to perform wire-AND functions on the bus. The maximum number of devices on the bus is limited by only the maximum capacitance specification of 400 pF. Data is transmitted in byte packages. †

21.4.2.1. START and STOP Generation

When operating as a master, putting data into the transmit FIFO causes the I²C controller to generate a START condition on the I²C bus. In order for the master to complete the transfer and issue a STOP condition it must find a transmit FIFO entry tagged with a stop bit. Allowing the transmit FIFO to empty without a stop bit set, the master will stall the transfer by holding the SCL line low. †

When operating as a slave, the I²C controller does not generate START and STOP conditions, as per the protocol. However, if a read request is made to the I²C controller, it holds the SCL line low until read data has been supplied to it. This stalls the I²C bus until read data is provided to the slave I²C controller, or the I²C controller slave is disabled by writing a 0 to bit 0 of IC_ENABLE register. †

21.4.2.2. Combined Formats

The I²C controller supports mixed read and write combined format transactions in both 7-bit and 10-bit addressing modes. †

The I²C controller does not support mixed address and mixed address format—that is, a 7-bit address transaction followed by a 10-bit address transaction or vice versa—combined format transactions. †

To initiate combined format transfers, the IC_RESTART_EN bit in the IC_CON register should be set to 1. With this value set and operating as a master, when the I²C controller completes an I²C transfer, it checks the transmit FIFO and executes the next transfer. If the direction of this transfer differs from the previous transfer, the combined format is used to issue the transfer. If the IC_RESTART_EN is 0, a STOP is issued followed by a START condition. Another way to generate the RESTART condition is to set the Restart bit [10] of the DATA_CMD register. Regardless if the direction of the transfer changes or not the RESTART condition is generated.†

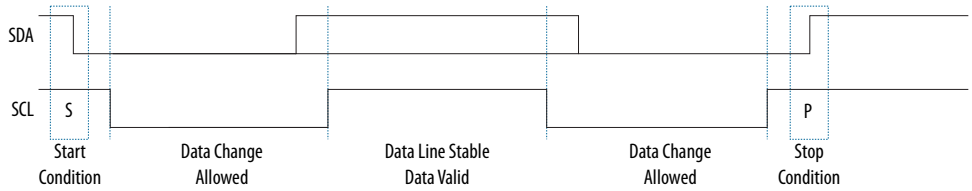
21.4.3. Protocol Details

21.4.3.1. START and STOP Conditions

When the bus is idle, both the SCL and SDA signals are pulled high through pull-up resistors on the bus. When the master wants to start a transmission on the bus, the master issues a START condition. This is defined to be a high-to-low transition of the SDA signal while SCL is 1. When the master wants to terminate the transmission, the master issues a STOP condition. This is defined to be a low-to-high transition of the SDA line while SCL is 1. †

The following figure shows the timing of the START and STOP conditions. When data is being transmitted on the bus, the SDA line must be stable when SCL is 1. †

Figure 129. Timing Diagram for START and STOP Conditions



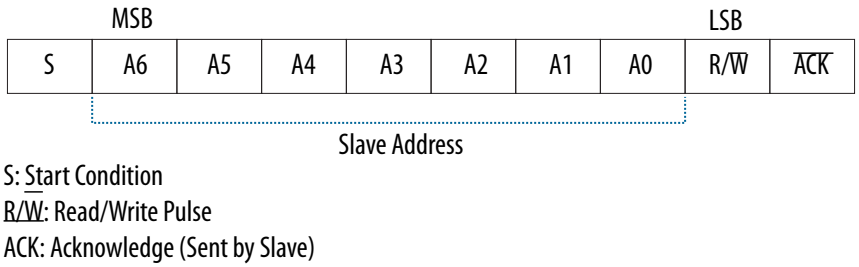
The signal transitions for the START or STOP condition, as shown in the figure, reflect those observed at the output signals of the master driving the I²C bus. Care should be taken when observing the SDA or SCL signals at the input signals of the slave(s), because unequal line delays may result in an incorrect SDA or SCL timing relationship. †

21.4.3.2. Addressing Slave Protocol

21.4.3.2.1. 7-Bit Address Format

During the 7-bit address format, the first seven bits (bits 7:1) of the first byte set the slave address and the LSB bit (bit 0) is the R/W bit as shown in the following figure. When bit 0 (R/W) is set to 0, the master writes to the slave. When bit 0 (R/W) is set to 1, the master reads from the slave. †

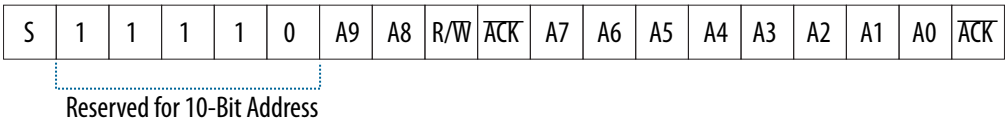
Figure 130. 7- Bit Address Format



21.4.3.2.2. 10-Bit Address Format

During 10-bit addressing, two bytes are transferred to set the 10-bit address. The transfer of the first byte contains the following bit definition. The first five bits (bits 7:3) notify the slaves that this is a 10-bit transfer followed by the next two bits (bits 2:1), which set the slaves address bits 9:8, and the LSB bit (bit 0) is the R/W bit. The second byte transferred sets bits 7:0 of the slave address. †

Figure 131. 10-Bit Address Format



S: Start Condition
R/W: Read/Write Pulse
ACK: Acknowledge (Sent by Slave)

The following table defines the special purpose and reserved first byte addresses. †

Table 220. I²C Definition of Bits in First Byte

Slave Address	R/W Bit	Description
0000 000	0	General call address. The I²C controller places the data in the receive buffer and issues a general call interrupt.
0000 000	1	START byte. For more details, refer to "START BYTE Transfer Protocol"
0000 001	X	CBUS address. The I²C controller ignores these accesses.
0000 010	X	Reserved
0000 011	X	Reserved
0000 1XX	X	Unused
1111 1XX	X	Reserved
1111 0XX	X	10-bit slave addressing.
Note to Table: 'X' indicates do not care.		

Related Information

START BYTE Transfer Protocol on page 652

21.4.3.3. Transmitting and Receiving Protocol

The master can initiate data transmission and reception to or from the bus, acting as either a master-transmitter or master-receiver. A slave responds to requests from the master to either transmit data or receive data to or from the bus, acting as either a slave-transmitter or slave-receiver, respectively. †

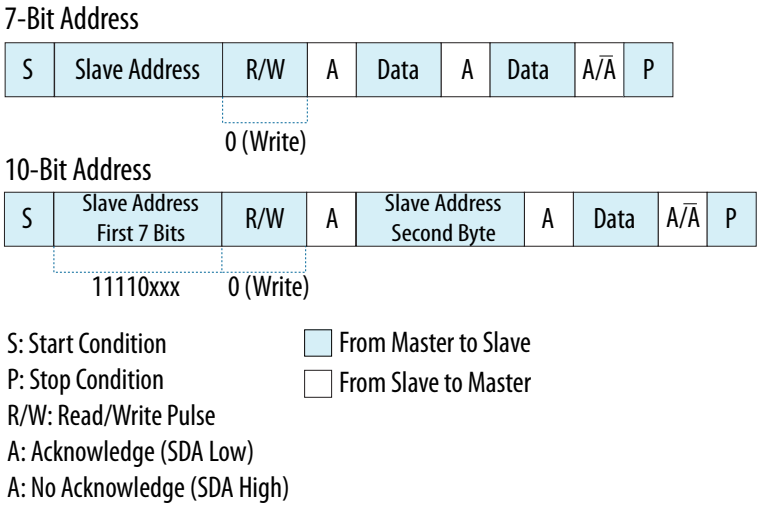
21.4.3.3.1. Master-Transmitter and Slave-Receiver

All data is transmitted in byte format, with no limit on the number of bytes transferred per data transfer. After the master sends the address and R/W bit or the master transmits a byte of data to the slave, the slave-receiver must respond with the

acknowledge signal (ACK). When a slave-receiver does not respond with an ACK pulse, the master aborts the transfer by issuing a STOP condition. The slave must leave the SDA line high so that the master can abort the transfer. †

If the master-transmitter is transmitting data as shown in the following figure, then the slave-receiver responds to the master-transmitter with an ACK pulse after every byte of data is received. †

Figure 132. Master-Transmitter Protocol †



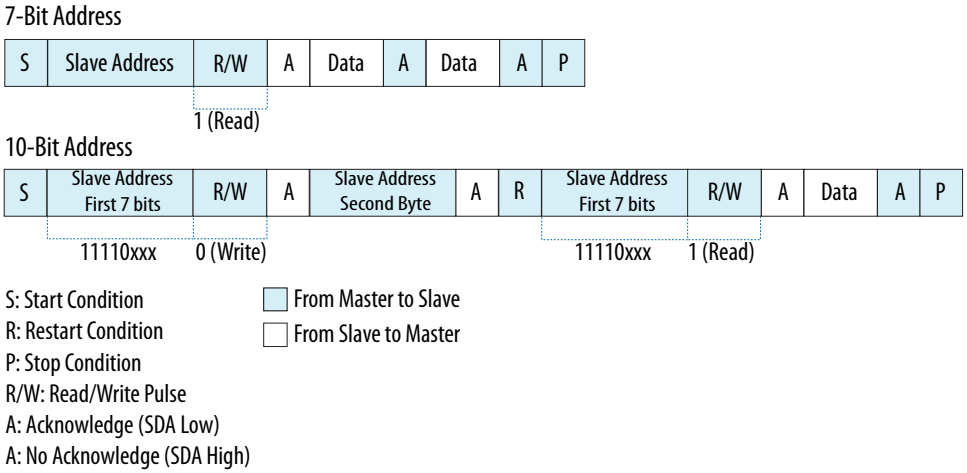
21.4.3.3.2. Master-Receiver and Slave-Transmitter

If the master is receiving data as shown in the following figure, then the master responds to the slave-transmitter with an ACK pulse after a byte of data has been received, except for the last byte. This is the way the master-receiver notifies the slave-transmitter that this is the last byte. The slave-transmitter relinquishes the SDA line after detecting the No Acknowledge (NACK) bit so that the master can issue a STOP condition. †

When a master does not want to relinquish the bus with a STOP condition, the master can issue a RESTART condition. This is identical to a START condition except it occurs after the ACK pulse. Operating in master mode, the I²C controller can then communicate with the same slave using a transfer of a different direction. For a description of the combined format transactions that the I²C controller supports, refer to "Combined Formats" section of this chapter. †

Note: The I²C controller must be inactive on the serial port before the target slave address register, IC_TAR, can be reprogrammed. †

Figure 133. Master-Receiver Protocol †



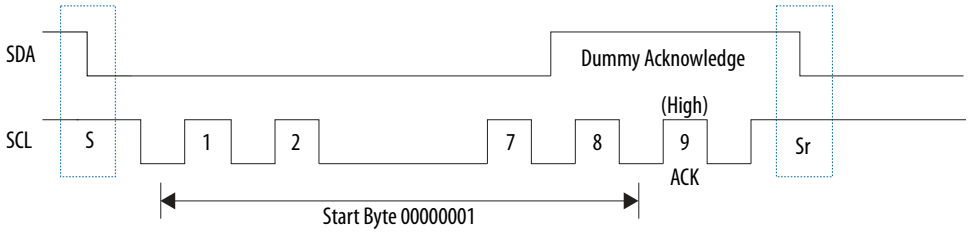
Related Information

Combined Formats on page 648

21.4.3.4. START BYTE Transfer Protocol

The START BYTE transfer protocol is set up for systems that do not have an on-board dedicated I²C hardware module. When the I²C controller is set as a slave, it always samples the I²C bus at the highest speed supported so that it never requires a START BYTE transfer. However, when I²C controller is set as a master, it supports the generation of START BYTE transfers at the beginning of every transfer in case a slave device requires it. This protocol consists of seven zeros being transmitted followed by a 1, as illustrated in the following figure. This allows the processor that is polling the bus to under-sample the address phase until the microcontroller detects a 0. Once the microcontroller detects a 0, it switches from the under sampling rate to the correct rate of the master. †

Figure 134. START BYTE Transfer †



The START BYTE has the following procedure: †

1. Master generates a START condition. †
2. Master transmits the START byte (0000 0001). †
3. Master transmits the ACK clock pulse. (Present only to conform with the byte handling format used on the bus) †
4. No slave sets the ACK signal to 0. †
5. Master generates a RESTART (R) condition. †

A hardware receiver does not respond to the START BYTE because it is a reserved address and resets after the RESTART condition is generated. †

21.4.4. Multiple Master Arbitration

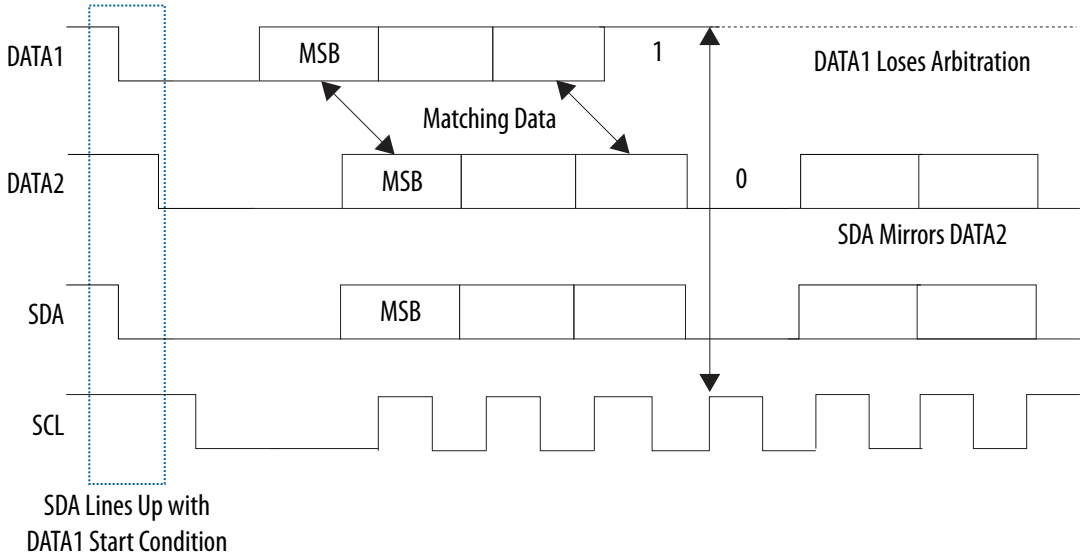
The I²C controller bus protocol allows multiple masters to reside on the same bus. If there are two masters on the same I²C-bus, there is an arbitration procedure if both try to take control of the bus at the same time by simultaneously generating a START condition. Once a master (for example, a microcontroller) has control of the bus, no other master can take control until the first master sends a STOP condition and places the bus in an idle state. †

Arbitration takes place on the SDA line, while the SCL line is 1. The master, which transmits a 1 while the other master transmits 0, loses arbitration and turns off its data output stage. The master that lost arbitration can continue to generate clocks until the end of the byte transfer. If both masters are addressing the same slave device, the arbitration could go into the data phase. †

Upon detecting that it has lost arbitration to another master, the I²C controller stops generating SCL. †

The following figure illustrates the timing of two masters arbitrating on the bus.

Figure 135. Multiple Master Arbitration †



The bus control is determined by address or master code and data sent by competing masters, so there is no central master nor any order of priority on the bus. †

Arbitration is not allowed between the following conditions: †

- A RESTART condition and a data bit †
- A STOP condition and a data bit †
- A RESTART condition and a STOP condition †

Slaves are not involved in the arbitration process. †

21.4.4.1. Clock Synchronization

When two or more masters try to transfer information on the bus at the same time, they must arbitrate and synchronize the SCL clock. All masters generate their own clock to transfer messages. Data is valid only during the high period of SCL clock. Clock synchronization is performed using the wired-AND connection to the SCL signal. When the master transitions the SCL clock to 0, the master starts counting the low time of the SCL clock and transitions the SCL clock signal to 1 at the beginning of the next clock period. However, if another master is holding the SCL line to 0, then the master goes into a HIGH wait state until the SCL clock line transitions to 1. †

All masters then count off their high time, and the master with the shortest high time transitions the SCL line to 0. The masters then counts out their low time and the one with the longest low time forces the other master into a HIGH wait state. Therefore, a synchronized SCL clock is generated, which is illustrated in the following figure. Optionally, slaves may hold the SCL line low to slow down the timing on the I²C bus. †

Figure 136. Multiple Master Clock Synchronization †

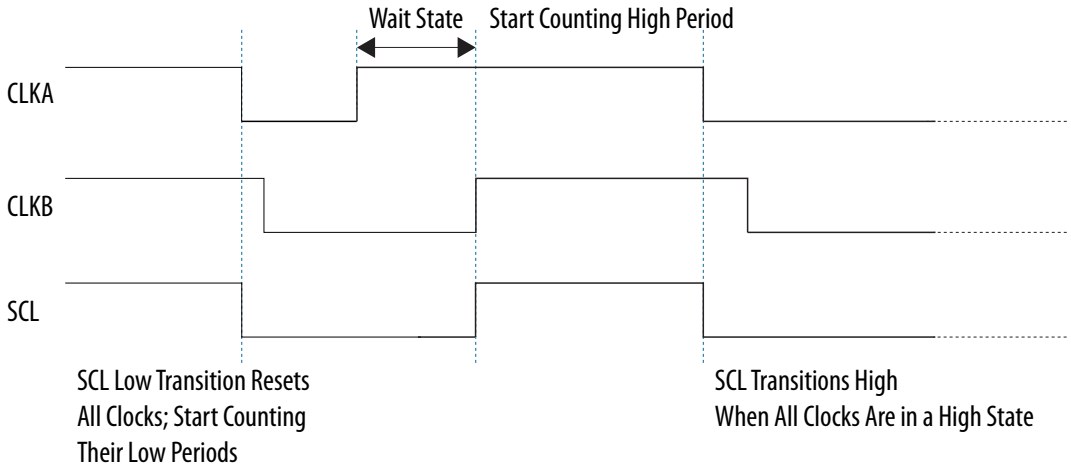
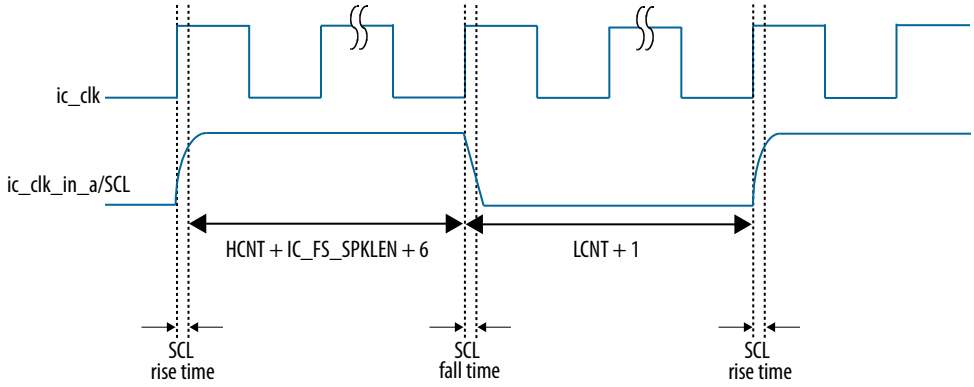


Figure 137. Impact of SCL Rise Time and Fall Time on Generated SCL



The following equations can be used to compute SCL high and low time:

```
SCL_High_time = [(HCNT + IC_FS_SPKLEN + 6)*ic_clk] +
SCL_Fall_time
```

```
SCL_Low_time = [(LCNT + 1)*ic_clk] - SCL_Fall_time +
SCL_Rise_time
```

21.4.5. Clock Frequency Configuration

When you configure the I2C controller as a master, the SCL count registers must be set before any I2C bus transaction can take place in order to ensure proper I/O timing. † There are four SCL count registers:

- Standard speed I2C clock SCL high count, `IC_SS_SCL_HCNT` †
- Standard speed I2C clock SCL low count, `IC_SS_SCL_LCNT` †
- Fast speed I2C clock SCL high count, `IC_FS_SCL_HCNT` †
- Fast speed I2C clock SCL low count, `IC_FS_SCL_LCNT` †

It is not necessary to program any of the SCL count registers if the I2C controller is enabled to operate only as an I2C slave, since these registers are used only to determine the SCL timing requirements for operation as an I2C master. †

21.4.5.1. Minimum High and Low Counts

When the I2C controller operates as an I2C master in both transmit and receive transfers, the minimum value that can be programmed in the SCL low count registers is 8 while the minimum value allowed for the SCL high count registers is 6. †

The minimum value of 8 for the low count registers is due to the time required for the I2C controller to drive SDA after a negative edge of SCL. The minimum value of 6 for the high count register is due to the time required for the I2C controller to sample SDA during the high period of SCL. †

The I2C controller adds one cycle to the low count register values in order to generate the low period of the SCL clock.

The I2C controller adds seven cycles to the high count register values in order to generate the high period of the SCL clock. This is due to the following factors: †

- The digital filtering applied to the SCL line incurs a delay of four `l4_sp_clk` cycles. This filtering includes metastability removal and a 2-out-of-3 majority vote processing on SDA and SCL edges. †
- Whenever SCL is driven 1 to 0 by the I2C controller—that is, completing the SCL high time—an internal logic latency of three `l4_sp_clk` cycles incurs. †

Consequently, the minimum SCL low time of which the I2C controller is capable is nine (9) `l4_sp_clk` periods (8+1), while the minimum SCL high time is thirteen (13) `l4_sp_clk` periods (6+1+3+3). †

Note: The `ic_fs_spklen` register must be set before any I2C bus transaction can take place to ensure stable operation. This register sets the duration measured in `ic_clk` cycles, of the longest spike in the SCL or SDA lines that is filtered out by the spike suppression logic.†

21.4.5.1.1. Calculating High and Low Counts

The calculations below show an example of how to calculate SCL high and low counts for each speed mode in the I2C controller.

The equation to calculate the proper number of l4_sp_clk clock pulses required for setting the proper SCL clocks high and low times is as follows: †

Table 221. Equation

IC_HCNT = ceil(MIN_SCL_HIGHtime*OSCFREQ)
IC_LCNT = ceil(MIN_SCL_LOWtime*OSCFREQ)

MIN_SCL_HIGHtime = minimum high period
MIN_SCL_HIGHtime =
4000 ns for 100 kbps
600 ns for 400 kbps
60 ns for 3.4 Mbs, bus loading = 100pF
160 ns for 3.4 Mbs, bus loading = 400pF

MIN_SCL_LOWtime = minimum low period
MIN_SCL_LOWtime =
4700 ns for 100 kbps
1300 ns for 400 kbps
120 ns for 3.4Mbs, bus loading = 100pF
320 ns for 3.4Mbs, bus loading = 400pF

OSCFREQ = l4_sp_clk clock frequency (Hz)

Calculating High and Low Counts

```
OSCFREQ = 100 MHz
I2Cmode = fast, 400 kbps
MIN_SCL_HIGHtime = 600 ns
MIN_SCL_LOWtime = 1300 ns

IC_HCNT = ceil(600 ns * 100 MHz) IC_HCNTSCL PERIOD = 60
IC_LCNT = ceil(1300 ns * 100 MHz) IC_LCNTSCL PERIOD = 130
Actual MIN_SCL_HIGHtime = 60*(1/100 MHz) = 600 ns
Actual MIN_SCL_LOWtime = 130*(1/100 MHz) = 1300 ns †
```

You may configure the SCL and SDA falling time using the parameters i2c-scl-falling-time-ns and i2c-sda-falling-time-ns respectively.

21.4.6. SDA Hold Time

The I2C protocol specification requires 300 ns of hold time on the SDA signal in standard and fast speed modes. Board delays on the SCL and SDA signals can mean that the hold time requirement is met at the I2C master, but not at the I2C slave (or vice-versa). As each application encounters differing board delays, the I2C controller contains a software programmable register, IC_SDA_HOLD, to enable dynamic adjustment of the SDA hold time. IC_SDA_HOLD effects both slave-transmitter and master mode.

21.4.7. DMA Controller Interface

The I2C controller supports DMA signaling to indicate when data is ready to be read or when the transmit FIFO needs data. This support requires 2 DMA channels, one for transmit data and one for receive data. The I2C controller supports both single and burst DMA transfers. System software can choose the DMA burst mode by programming an appropriate value into the threshold registers. The recommended setting of the FIFO threshold register value is half full.

To enable the DMA controller interface on the I2C controller, you must write to the DMA control register (DMACR) bits. Writing a 1 into the TDMAE bit field of DMACR register enables the I2C controller transmit handshaking interface. Writing a 1 into the RDMAE bit field of the DMACR register enables the I2C controller receive handshaking interface. †

Related Information

DMA Controller on page 512
For details about the DMA burst length microcode setup, refer to the *DMA controller* chapter.

21.4.8. Clocks

Each I2C controller is connected to the `l4_sp_clk` clock, which clocks transfers in standard and fast mode. The clock input is driven by the clock manager.

Related Information

Clock Manager on page 52
For more information, refer to *Clock Manager* chapter.

21.4.9. Resets

Each I2C controller has a separate reset signal. The reset manager drives the signals on a cold or warm reset.

Related Information

Reset Manager on page 134
For more information, refer to *Reset Manager* chapter.

21.4.9.1. Taking the I2C Controller Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

21.5. I2C Controller Programming Model

This section describes the programming model for the I2C controllers based on the two master and slave operation modes. †

Note: Each I2C controller should be set to operate only as an I2C master or as an I2C slave, never set both simultaneously. Ensure that bit 6 (`IC_SLAVE_DISABLE`) and 0 (`IC_MASTER_MODE`) of the `IC_CON` register are never set to 0 and 1, respectively. †

21.5.1. Slave Mode Operation

21.5.1.1. Initial Configuration

To use the I²C controller as a slave, perform the following steps: †

1. Disable the I²C controller by writing a 0 to bit 0 of the IC_ENABLE register. †
2. Write to the IC_SAR register (bits 9:0) to set the slave address. This is the address to which the I²C controller responds. †

   Note: The reset value for the I²C controller slave address is 0x55. If you are using 0x55 as the slave address, you can safely skip this step.
3. Write to the IC_CON register to specify which type of addressing is supported (7- or 10-bit by setting bit 3). Enable the I²C controller in slave-only mode by writing a 0 into bit 6 (IC_SLAVE_DISABLE) and a 0 to bit 0 (MASTER_MODE). †

   Note: Slaves and masters do not have to be programmed with the same type of addressing 7- or 10-bit address. For instance, a slave can be programmed with 7-bit addressing and a master with 10-bit addressing, and vice versa. †
4. Enable the I²C controller by writing a 1 in bit 0 of the IC_ENABLE register. †

   Note: It is recommended that the I²C Slave be brought out of reset only when the I²C bus is IDLE. De-asserting the reset when a transfer is ongoing on the bus causes internal flip-flops used to synchronize SDA and SCL to toggle from a reset value of 1 to the actual value on the bus. In this scenario, if SDA toggling from 1 to 0 while SCL is 1, thereby causing a false START condition to be detected by the I²C Slave by configuring the I²C with IC_SLAVE_DISABLE = 1 and IC_MASTER_MODE = 1 so that the Slave interface is disabled after reset. It can then be enabled by programming IC_CON[0] = 0 and IC_CON[6] = 0 after the internal SDA and SCL have synchronized to the value on the bus; this takes approximately 6 ic_clk cycles after reset de-assertion.†

21.5.1.2. Slave-Transmitter Operation for a Single Byte

When another I²C master device on the bus addresses the I²C controller and requests data, the I²C controller acts as a slave-transmitter and the following steps occur: †

1. The other I²C master device initiates an I²C transfer with an address that matches the slave address in the IC_SAR register of the I²C controller †
2. The I²C controller acknowledges the sent address and recognizes the direction of the transfer to indicate that it is acting as a slave-transmitter. †
3. The I²C controller asserts the RD_REQ interrupt (bit 5 of the IC_RAW_INTR_STAT register) and waits for software to respond. †

If the RD_REQ interrupt has been masked, due to bit 5 of the IC_INTR_MASK register (M_RD_REQ bit field) being set to 0, then it is recommended that you instruct the CPU to perform periodic reads of the IC_RAW_INTR_STAT register. †

- Reads that indicate bit 5 of the IC_RAW_INTR_STAT register (R_RD_REQ bit field) being set to 1 must be treated as the equivalent of the RD_REQ interrupt being asserted. †
- Software must then act to satisfy the I²C transfer. †
- The timing interval used should be in the order of 10 times the fastest SCL clock period the I²C controller can handle. For example, for 400 Kbps, the timing interval is 25 us. †

  Note: The value of 10 is recommended here because this is approximately the amount of time required for a single byte of data transferred on the I²C bus.†

4. If there is any data remaining in the TX FIFO before receiving the read request, the I²C controller asserts a TX_ABRT interrupt (bit 6 of the IC_RAW_INTR_STAT register) to flush the old data from the TX FIFO. †

   Note: Because the I²C controller's TX FIFO is forced into a flushed/reset state whenever a TX_ABRT event occurs, it is necessary for software to release the I²C controller from this state by reading the IC_CLR_TX_ABRT register before attempting to write into the TX FIFO. For more information, refer to the C_RAW_INTR_STAT register description in the register map.†

   If the TX_ABRT interrupt has been masked, due to of IC_INTR_MASK[6] register (M_TX_ABRT bit field) being set to 0, then it is recommended that the CPU performs periodic reads of the IC_RAW_INTR_STAT register. †

   - Reads that indicate bit 6 (R_TX_ABRT) being set to 1 must be treated as the equivalent of the TX_ABRT interrupt being asserted. †
   - There is no further action required from software. †
   - The timing interval used should be similar to that described in the previous step for the IC_RAW_INTR_STAT[5] register. †
5. Software writes to the DAT bits of the IC_DATA_CMD register with the data to be written and writes a 0 in bit 8.†
6. Software must clear the RD_REQ and TX_ABRT interrupts (bits 5 and 6, respectively) of the IC_RAW_INTR_STAT register before proceeding. †

   If the RD_REQ or TX_ABRT interrupt is masked, then clearing of the IC_RAW_INTR_STAT register has already been performed when either the R_RD_REQ or R_TX_ABRT bit has been read as 1.
7. The I²C controller transmits the byte. †
8. The master may hold the I²C bus by issuing a RESTART condition or release the bus by issuing a STOP condition. †

21.5.1.3. Slave-Receiver Operation for a Single Byte

When another I²C master device on the bus addresses the I²C controller and is sending data, the I²C controller acts as a slave-receiver and the following steps occur:†

1. The other I²C master device initiates an I²C transfer with an address that matches the I²C controller's slave address in the IC_SAR register. †
2. The I²C controller acknowledges the sent address and recognizes the direction of the transfer to indicate that the I²C controller is acting as a slave-receiver. †
3. I²C controller receives the transmitted byte and places it in the receive buffer. †

   Note: If the RX FIFO is completely filled with data when a byte is pushed, then an overflow occurs and the I²C controller continues with subsequent I²C transfers. Because a NACK is not generated, software must recognize the overflow when indicated by the I²C controller (by the R_RX_OVER bit in the IC_INTR_STAT register) and take appropriate actions to recover from lost data. Hence, there is a real time constraint on software to service the RX FIFO before the latter overflow as there is no way to reapply pressure to the remote transmitting master. †
4. I²C controller asserts the RX_FULL interrupt (IC_RAW_INTR_STAT[2] register). †

   If the RX_FULL interrupt has been masked, due to setting IC_INTR_MASK[2] register to 0 or setting IC_TX_TL to a value larger than 0, then it is recommended that the CPU does periodic reads of the IC_STATUS register. Reads of the IC_STATUS register, with bit 3 (RFNE) set at 1, must then be treated by software as the equivalent of the RX_FULL interrupt being asserted. †
5. Software may read the byte from the IC_DATA_CMD register (bits 7:0). †
6. The other master device may hold the I²C bus by issuing a RESTART condition or release the bus by issuing a STOP condition. †

21.5.1.4. Slave-Transfer Operation for Bulk Transfers

In the standard I²C protocol, all transactions are single byte transactions and the programmer responds to a remote master read request by writing one byte into the slave's TX FIFO. When a slave (slave-transmitter) is issued with a read request (RD_REQ) from the remote master (master-receiver), at a minimum there should be at least one entry placed into the slave-transmitter's TX FIFO. The I²C controller is designed to handle more data in the TX FIFO so that subsequent read requests can receive that data without raising an interrupt to request more data. Ultimately, this eliminates the possibility of significant latencies being incurred between raising the interrupt for data each time had there been a restriction of having only one entry placed in the TX FIFO. †

This mode only occurs when I²C controller is acting as a slave-transmitter. If the remote master acknowledges the data sent by the slave-transmitter and there is no data in the slave's TX FIFO, the I²C controller raises the read request interrupt (RD_REQ) and waits for data to be written into the TX FIFO before it can be sent to the remote master. †

If the RD_REQ interrupt is masked, due to bit 5 (M_RD_REQ) of the IC_INTR_STAT register being set to 0, then it is recommended that the CPU does periodic reads of the IC_RAW_INTR_STAT register. Reads of IC_RAW_INTR_STAT that return bit 5 (R_RD_REQ) set to 1 must be treated as the equivalent of the RD_REQ interrupt referred to in this section.†

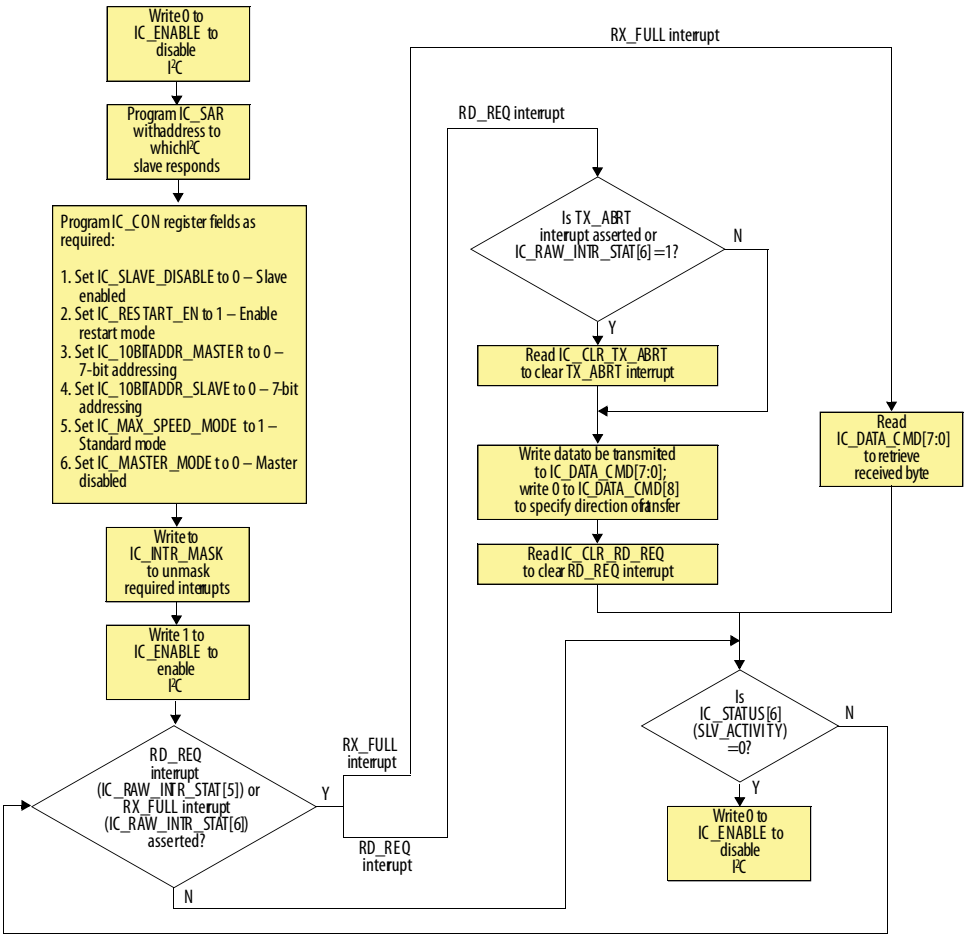
The RD_REQ interrupt is raised upon a read request, and like interrupts, must be cleared when exiting the interrupt service handling routine (ISR). The ISR allows you to either write 1 byte or more than 1 byte into the TX FIFO. During the transmission of these bytes to the master, if the master acknowledges the last byte then the slave must raise the RD_REQ again because the master is requesting for more data. †

If the programmer knows in advance that the remote master is requesting a packet of n bytes, then when another master addresses the I²C controller and requests data, the TX FIFO could be written with n number bytes and the remote master receives it as a continuous stream of data. For example, the I²C controller slave continues to send data to the remote master as long as the remote master is acknowledging the data sent and there is data available in the TX FIFO. There is no need to issue RD_REQ again. †

If the remote master is to receive n bytes from the I²C controller but the programmer wrote a number of bytes larger than n to the TX FIFO, then when the slave finishes sending the requested n bytes, it clears the TX FIFO and ignores any excess bytes. †

The I²C controller generates a transmit abort (TX_ABRT) event to indicate the clearing of the TX FIFO in this example. At the time an ACK/NACK is expected, if a NACK is received, then the remote master has all the data it wants. At this time, a flag is raised within the slave's state machine to clear the leftover data in the TX FIFO. This flag is transferred to the processor bus clock domain where the FIFO exists and the contents of the TX FIFO are cleared at that time. †

21.5.1.5. Slave Programming Model



21.5.2. Master Mode Operation

21.5.2.1. Initial Configuration

For master mode operation, the target address and address format can be changed dynamically without having to disable the I²C controller. This feature is only applicable when the I²C controller is acting as a master because the slave requires the component to be disabled before any changes can be made to the address. To use the I²C controller as a master, perform the following steps: †

For multiple I²C transfers, perform additional writes to the Tx FIFO such that the Tx FIFO does not become empty during the I²C transaction. IF the Tx FIFO is completely emptied at any stage, then the master stalls the transfer by holding the SCL line low because there was no stop bit indicating the master to issue a STOP. The master completes the transfer when it finds a Tx FIFO entry tagged with a Stop bit.

1. Disable the I²C controller by writing 0 to bit 0 of the IC_ENABLE register. †
2. Write to the IC_CON register to set the maximum speed mode supported for slave operation (bits 2:1) and to specify whether the I²C controller starts its transfers in 7/10 bit addressing mode when the device is a slave (bit 3). †
3. Write to the IC_TAR register the address of the I²C device to be addressed. It also indicates whether a General Call or a START BYTE command is going to be performed by I²C. The desired speed of the I²C controller master-initiated transfers, either 7-bit or 10-bit addressing, is controlled by the IC_10BITADDR_MASTER bit field (bit 12). †
4. Enable the I²C controller by writing a 1 in bit 0 of the IC_ENABLE register. †
5. Now write the transfer direction and data to be sent to the IC_DATA_CMD register. If the IC_DATA_CMD register is written before the I²C controller is enabled, the data and commands are lost as the buffers are kept cleared when the I²C controller is not enabled. †

21.5.2.2. Dynamic IC_TAR or IC_10BITADDR_MASTER Update

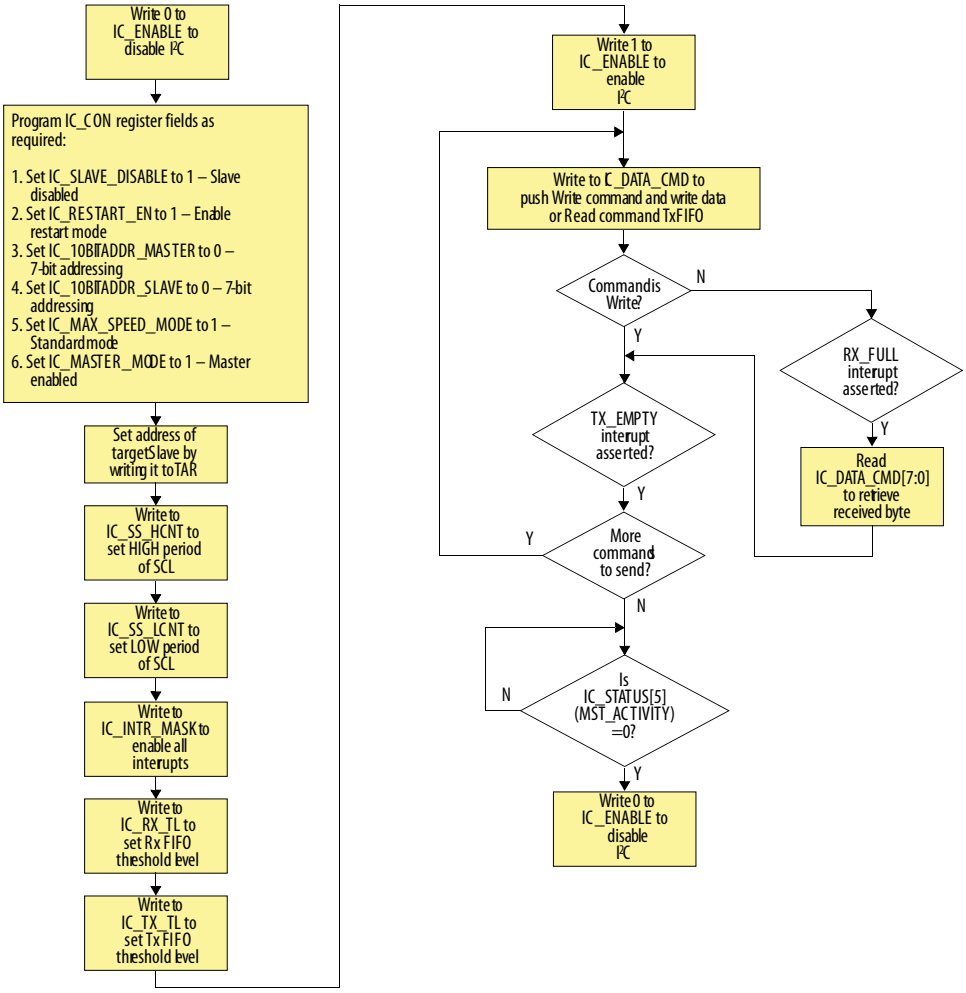
The I²C controller supports dynamic updating of the IC_TAR (bits 9:0) and IC_10BITADDR_MASTER (bit 12) bit fields of the IC_TAR register. You can dynamically write to the IC_TAR register provided the following conditions are met: †

- The I²C controller is not enabled (IC_ENABLE=0); †
- The I²C controller is enabled (IC_ENABLE=1); AND I²C controller is NOT engaged in any Master (TX, RX) operation (IC_STATUS[5]=0); AND I²C controller is enabled to operate in Master mode (IC_CON[0]=1); AND there are no entries in the TX FIFO (IC_STATUS[2]=1) †

21.5.2.3. Master Transmit and Master Receive

The I²C controller supports switching back and forth between reading and writing dynamically. To transmit data, write the data to be written to the lower byte of the I²C Rx/Tx Data Buffer and Command Register (IC_DATA_CMD). The CMD bit [8] should be written to 0 for I²C write operations. Subsequently, a read command may be issued by writing "don't cares" to the lower byte of the IC_DATA_CMD register, and a 1 should be written to the CMD bit.†

21.5.2.4. Master Programming Model



21.5.3. Disabling the I²C Controller

The register IC_ENABLE_STATUS is added to allow software to unambiguously determine when the hardware has completely shutdown in response to the IC_ENABLE register being set from 1 to 0. †

1. Define a timer interval (ti2c_poll) equal to the 10 times the signaling period for the highest I²C transfer speed used in the system and supported by the I²C controller. For example, if the highest I²C transfer mode is 400 Kbps, then ti2c_poll is 25 us. †
2. Define a maximum time-out parameter, MAX_T_POLL_COUNT, such that if any repeated polling operation exceeds this maximum value, an error is reported. †
3. Execute a blocking thread/process/function that prevents any further I²C master transactions to be started by software, but allows any pending transfers to be completed.

— This step can be ignored if the I²C controller is programmed to operate as an I²C slave only. †

4. The variable POLL_COUNT is initialized to zero. †
5. Set IC_ENABLE to 0. †
6. Read the IC_ENABLE_STATUS register and test the IC_EN bit (bit 0). Increment POLL_COUNT by one. If POLL_COUNT >= MAX_T_POLL_COUNT, exit with the relevant error code. †
7. If IC_ENABLE_STATUS[0] is 1, then sleep for ti2c_poll and proceed to the previous step. Otherwise, exit with a relevant success code. †

21.5.4. Abort Transfer

The ABORT control bit of the IC_ENABLE register allows the software to relinquish the I2C bus before completing the issued transfer commands from the Tx FIFO. In response to an ABORT request, the controller issues the STOP condition over the I2C bus, followed by Tx FIFO flush. Aborting the transfer is allowed only in master mode of operation.†

1. Stop filling the Tx FIFO (IC_DATA_CMD) with new commands.†
2. When operating in DMA mode, disable the transmit DMA by setting TDMAE to 0. †
3. Set bit 1 of the IC_ENABLE register (ABORT) to 1.†
4. Wait for the M_TX_ABRT interrupt.†
5. Read the IC_TX_ABRT_SOURCE register to identify the source as ABRT_USER_ABRT.†

21.5.5. DMA Controller Operation

To enable the DMA controller interface on the I²C controller, you must write the DMA Control Register (IC_DMA_CR). Writing a 1 to the TDMAE bit field of IC_DMA_CR register enables the I²C controller transmit handshaking interface. Writing a 1 to the RDMAE bit field of the IC_DMA_CR register enables the I²C controller receive handshaking interface.†

The FIFO buffer depth (FIFO_DEPTH) for both the RX and TX buffers in the I²C controller is 64 entries.

Related Information

DMA Controller on page 512
For details about the DMA burst length microcode setup, refer to the *DMA controller* chapter.

21.5.5.1. Transmit FIFO Underflow

During I²C serial transfers, transmit FIFO requests are made to the DMA controller whenever the number of entries in the transmit FIFO is less than or equal to the value in DMA Transmit Data Level Register (IC_DMA_TDLR), also known as the watermark level. The DMA controller responds by writing a burst of data to the transmit FIFO buffer, of length specified as DMA burst length. †

Note: Data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously, that is, when the FIFO begins to empty, another DMA request should be triggered. Otherwise, the FIFO runs out of the data (underflow) causing the maser to stall the transfer by holding the SCL line low. To prevent this condition, you must set the watermark level correctly.†

Related Information

DMA Controller on page 512
For details about the DMA burst length microcode setup, refer to the *DMA controller* chapter.

21.5.5.2. Transmit Watermark Level

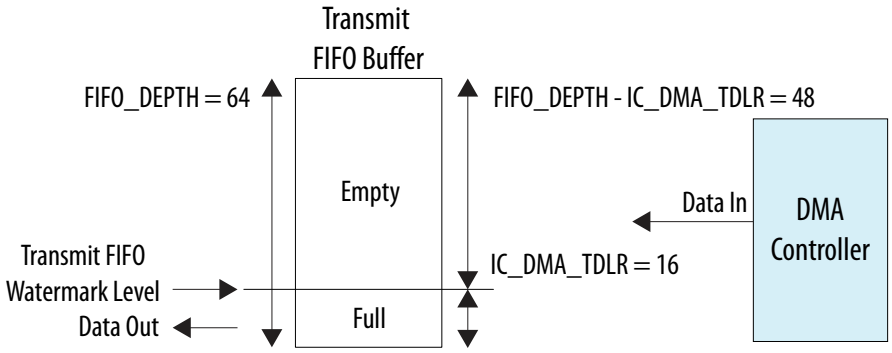
Consider the example where the assumption is made: †

DMA burst length = FIFO_DEPTH - IC_DMA_TDLR †

Here the number of data items to be transferred in a DMA burst is equal to the empty space in the transmit FIFO. Consider the following two different watermark level settings: †

- Case 1: IC_DMA_TDLR = 16: †
  - Transmit FIFO watermark level = IC_DMA_TDLR = 16: †
  - DMA burst length = FIFO_DEPTH - IC_DMA_TDLR = 48: †
  - I²C transmit FIFO_DEPTH = 64: †
  - Block transaction size = 240: †

Figure 138. Transmit FIFO Watermark Level = 16



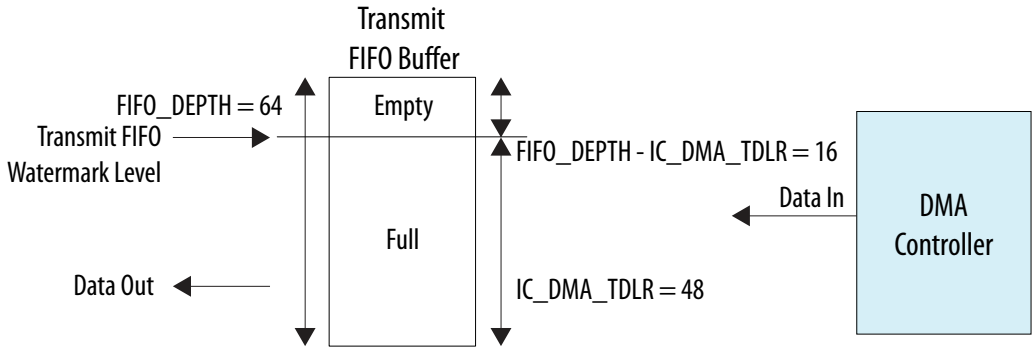
The number of burst transactions needed equals the block size divided by the number of data items per burst:

Block transaction size/DMA burst length = 240/48 = 5

The number of burst transactions in the DMA block transfer is 5. But the watermark level, IC_DMA_TDLR, is quite low. Therefore, the probability of transmit underflow is high where the I²C serial transmit line needs to transmit data, but there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the FIFO becomes empty.

- Case 2: IC_DMA_TDLR = 48 †
  - Transmit FIFO watermark level = IC_DMA_TDLR = 48 †
  - DMA burst length = FIFO_DEPTH - IC_DMA_TDLR = 16 †
  - I²C transmit FIFO_DEPTH = 64 †
  - Block transaction size = 240 †

Figure 139. Transmit FIFO Watermark Level = 48



Number of burst transactions in block: †

Block transaction size/DMA burst length = 240/16 = 15 †

In this block transfer, there are 15 destination burst transactions in a DMA block transfer. But the watermark level, IC_DMA_TDLR, is high. Therefore, the probability of I²C transmit underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the I²C transmit FIFO becomes empty. †

Thus, the second case has a lower probability of underflow at the expense of more burst transactions per block. This provides a potentially greater amount of bursts per block and worse bus utilization than the former case. †

Therefore, the goal in choosing a watermark level is to minimize the number of transactions per block, while at the same time keeping the probability of an underflow condition to an acceptable level. In practice, this is a function of the ratio of the rate at which the I²C transmits data to the rate at which the DMA can respond to destination burst requests. †

21.5.5.3. Transmit FIFO Overflow

Setting the DMA burst length to a value greater than the watermark level that triggers the DMA request might cause overflow when there is not enough space in the transmit FIFO to service the destination burst request. Therefore, the following equation must be adhered to in order to avoid overflow: †

DMA burst length <= FIFO_DEPTH - IC_DMA_TDLR

In case 2: IC_DMA_TDLR = 48, the amount of space in the transmit FIFO at the time of the burst request is made is equal to the DMA burst length. Thus, the transmit FIFO may be full, but not overflowed, at the completion of the burst transaction. †

Therefore, for optimal operation, DMA burst length should be set at the FIFO level that triggers a transmit DMA request; that is: †

DMA burst length = FIFO_DEPTH - IC_DMA_TDLR

Adhering to this equation reduces the number of DMA bursts needed for block transfer, and this in turn improves bus utilization. †

The transmit FIFO is not be full at the end of a DMA burst transfer if the I2C controller has successfully transmitted one data item or more on the I2C serial transmit line during the transfer. †

21.5.5.4. Receive FIFO Overflow

During I2C serial transfers, receive FIFO requests are made to the DMA whenever the number of entries in the receive FIFO is at or above the DMA Receive Data Level Register, that is IC_DMA_RDLR + 1. This is known as the watermark level. The DMA responds by fetching a burst of data from the receive FIFO. †

Data should be fetched by the DMA often enough for the receive FIFO to accept serial transfers continuously, that is, when the FIFO begins to fill, another DMA transfer is requested. Otherwise the FIFO fills with data (overflow). To prevent this condition, the user must set the watermark level correctly. †

21.5.5.5. Receive Watermark Level

Similar to choosing the transmit watermark level described earlier, the receive watermark level, IC_DMA_RDLR + 1, should be set to minimize the probability of overflow, as shown in the Receive FIFO Buffer diagram. It is a trade off between the number of DMA burst transactions required per block versus the probability of an overflow occurring. †

21.5.5.6. Receive FIFO Underflow

Setting the source transaction burst length greater than the watermark level can cause underflow where there is not enough data to service the source burst request. Therefore, the following equation must be adhered to avoid underflow: †

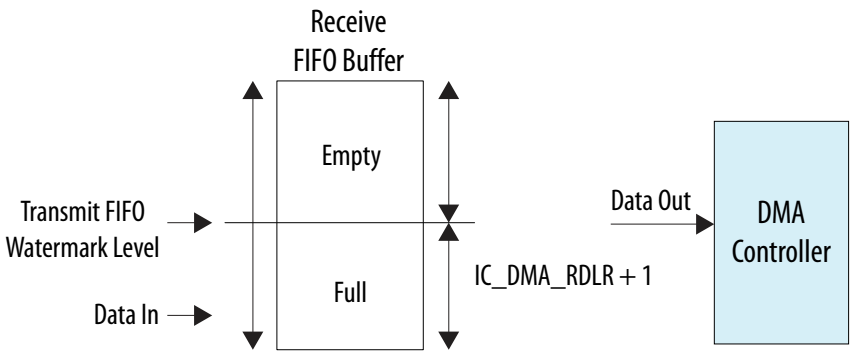
DMA burst length = IC_DMA_RDLR + 1

If the number of data items in the receive FIFO is equal to the source burst length at the time of the burst request is made, the receive FIFO may be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA burst length should be set at the watermark level, IC_DMA_RDLR + 1. †

Adhering to this equation reduces the number of DMA bursts in a block transfer, which in turn can avoid underflow and improve bus utilization. †

Note: The receive FIFO is not be empty at the end of the source burst transaction if the I2C controller has successfully received one data item or more on the I2C serial receive line during the burst. †

Figure 140. Receive FIFO Buffer



21.6. I 2 C Controller Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

22. UART Controller

The hard processor system (HPS) provides two UART controllers for asynchronous serial communication. The UART controllers are based on an industry standard 16550 UART controller. The UART controllers are instances of the Synopsys DesignWare APB Universal Asynchronous Receiver/Transmitter (DW_apb_uart) peripheral. (52)

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

22.1. UART Controller Features

The UART controller provides the following functionality and features:

- Programmable character properties, such as number of data bits per character, optional parity bits, and number of stop bits †
- Line break generation and detection †
- DMA controller handshaking interface
- Prioritized interrupt identification †
- Programmable baud rate
- False start bit detection †
- Automatic flow control mode per 16750 standard †
- Internal loopback mode support
- 128-byte transmit and receive FIFO buffers
  - FIFO buffer status registers †
  - FIFO buffer access mode (for FIFO buffer testing) enables write of receive FIFO buffer by master and read of transmit FIFO buffer by master †

(52) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Shadow registers reduce software overhead and provide programmable reset †
- Transmitter holding register empty (THRE) interrupt mode †
- Separate thresholds for DMA request and handshake signals to maximize throughput

22.2. UART Controller Block Diagram and System Integration

Figure 141. UART Block Diagram

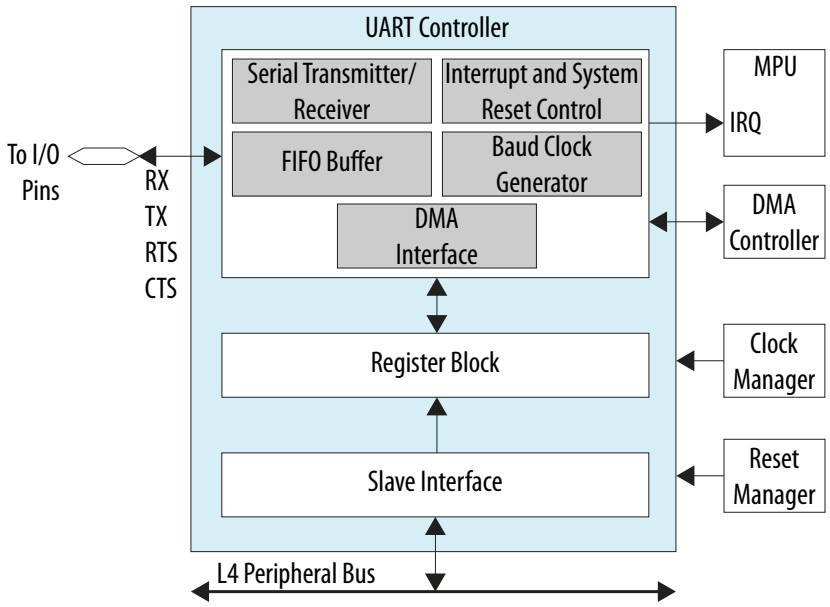


Table 222. UART Controller Block Descriptions

Block	Description
Slave interface	Slave interface between the component and L4 peripheral bus.
Register block	Provides main UART control, status, and interrupt generation functions.†
FIFO buffer	Provides FIFO buffer control and storage. †
Baud clock generator	Generates the transmitter and receiver baud clock. With a reference clock of 100 MHz, the UART controller supports transfer rates of 95 baud to 6.25 Mbaud. This supports communication with all known 16550 devices. The baud rate is controlled by programming the interrupt enable or divisor latch high (IER_DLH) and receive buffer, transmit holding, or divisor latch low (RBR_THR_DLL) registers.
Serial transmitter	Converts parallel data written to the UART into serial data and adds all additional bits, as specified by the control register, for transmission. This makeup of serial data, referred to as a character, exits the block in serial UART. †
Serial receiver	Converts the serial data character (as specified by the control register) received in the UART format to parallel form. Parity error detection, framing error detection and line break detection is carried out in this block. †
DMA interface	The UART controller includes a DMA controller interface to indicate when received data is available or when the transmit FIFO buffer requires data. The DMA requires two channels, one for transmit and one for receive. The UART controller supports single and burst transfers. You can use DMA in FIFO buffer and non-FIFO buffer mode.

Related Information

DMA Controller on page 512
For more information, refer to the DMA Controller chapter.

22.3. UART Controller Signal Description

22.3.1. HPS I/O Pins

There are two UARTs available in the HPS. Signals from both sets of UARTs can be routed to the HPS I/O. For more information on routing UART signals to the HPS I/O, refer to the *HPS Component Interfaces* chapter.

Table 223. HPS I/O UART Pin Descriptions

Pin	Width	Direction	Description
RX	1 bit	Input	Serial Input
TX	1 bit	Output	Serial Output
CTS	1 bit	Input	Clear to send
RTS	1 bit	Output	Request to send

Related Information

HPS Component Interfaces on page 743
For more information on how to route the UART signals to the FPGA and HPS I/O, refer to this chapter.

22.3.2. FPGA Routing

There are two UARTs provided in the HPS. Both sets of UART signals can be routed to the FPGA. For more information on routing UART signals to the FPGA, refer to the *HPS Component Interfaces* chapter.

Table 224. Signals for FPGA Routing

Signal	Width	Direction	Description
uart_rxd	1 bit	Input	Serial input
uart_txd	1 bit	Output	Serial output
uart_cts	1 bit	Input	Clear to send
uart_rts	1 bit	Output	Request to send
uart_dsr	1 bit	Input	Data set ready
uart_dcd	1 bit	Input	Data carrier detect
uart_ri	1 bit	Input	Ring indicator
uart_dtr	1 bit	Output	Data terminal ready
uart_out1_n	1 bit	Output	User defined output 1
uart_out2_n	1 bit	Output	User defined output 2

Related Information

HPS Component Interfaces on page 743
For more information on how to route the UART signals to the FPGA and HPS I/O, refer to this chapter.

22.4. Functional Description of the UART Controller

The HPS UART is based on an industry-standard 16550 UART. The UART supports serial communication with a peripheral, modem (data carrier equipment), or data set. The master (CPU) writes data over the slave bus to the UART. The UART converts the data to serial format and transmits to the destination device. The UART also receives serial data and stores it for the master (CPU). †

The UART's registers control the character length, baud rate, parity generation and checking, and interrupt generation. The UART's single interrupt output signal is supported by several prioritized interrupt types that trigger assertion. You can separately enable or disable each of the interrupt types with the control registers. †

22.4.1. FIFO Buffer Support

The UART controller includes 128-byte FIFO buffers to buffer transmit and receive data. FIFO buffer access mode allows the master to write the receive FIFO buffer and to read the transmit FIFO buffer for test purposes. FIFO buffer access mode is enabled with the FIFO access register (FAR). Once enabled, the control portions of the transmit and receive FIFO buffers are reset and the FIFO buffers are treated as empty. †

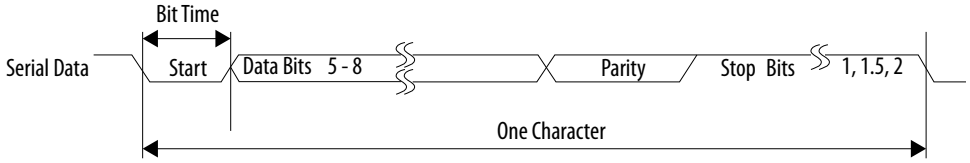
When FIFO buffer access mode is enabled, you can write data to the transmit FIFO buffer as normal; however, no serial transmission occurs in this mode and no data leaves the FIFO buffer. You can read back the data that is written to the transmit FIFO buffer with the transmit FIFO read (TFR) register. The TFR register provides the current data at the top of the transmit FIFO buffer. †

Similarly, you can also read data from the receive FIFO buffer in FIFO buffer access mode. Since the normal operation of the UART is halted in this mode, you must write data to the receive FIFO buffer to read it back. The receive FIFO write (RFW) register writes data to the receive FIFO buffer. The upper two bits of the 10-bit register write framing errors and parity error detection information to the receive FIFO buffer. Bit 9 of RFW indicates a framing error and bit 8 of RFW indicates a parity error. Although you cannot read these bits back from the receive buffer register, you can check the bits by reading the line status register (LSR), and by checking the corresponding bits when the data in question is at the top of the receive FIFO buffer. †

22.4.2. UART(RS232) Serial Protocol

Because the serial communication between the UART controller and the selected device is asynchronous, additional bits (start and stop) are added to the serial data to indicate the beginning and end. Utilizing these bits allows two devices to be synchronized. This structure of serial data accompanied by start and stop bits is referred to as a character, as shown in below.†

Figure 142. Serial Data Format



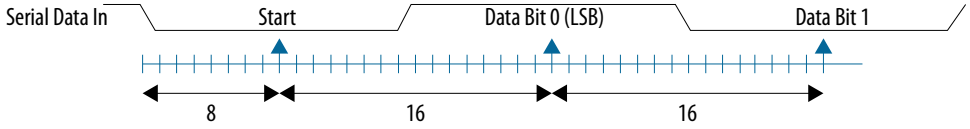
An additional parity bit may be added to the serial character. This bit appears after the last data bit and before the stop bit(s) in the character structure to provide the UART controller with the ability to perform simple error checking on the received data.†

The Control Register is used to control the serial character characteristics. The individual bits of the data word are sent after the start bit, starting with the least-significant bit (LSB). These are followed by the optional parity bit, followed by the stop bit(s), which can be 1, 1.5 or 2.†

All the bits in the transmission (with exception to the half stop bit when 1.5 stop bits are used) are transmitted for exactly the same time duration. This is referred to as a Bit Period or Bit Time. One Bit Time equals 16 baud clocks. To ensure stability on the line, the receiver samples the serial input data at approximately the midpoint of the Bit Time once the start bit has been detected. Because the exact number of baud clocks that each bit transmission is known, calculating the midpoint for sampling is not difficult. That is, every 16 baud clocks after the midpoint sample of the start bit.†

Together with serial input debouncing, this feature also contributes to avoid the detection of false start bits. Short glitches are filtered out by debouncing, and no transition is detected on the line. If a glitch is wide enough to avoid filtering by debouncing, a falling edge is detected. However, a start bit is detected only if the line is sampled low again after half a bit time has elapsed. †

Figure 143. Receiver Serial Data Sample Points



The baud rate of the UART controller is controlled by the serial clock and the Divisor Latch Register (DLH and DLL).†

22.4.3. Automatic Flow Control

The UART includes 16750-compatible request-to-send (RTS) and clear-to-send (CTS) serial data automatic flow control mode. You enable automatic flow control with the modem control register (MCR.AFCE). †

22.4.3.1. RTC Flow Control Trigger

RTC is an RX FIFO Almost-Full Trigger, where "almost full" refer to two available slots in the FIFO.

The UART controller uses two separate trigger levels for a DMA request and handshake signal (rts_n) in order to maximize throughput on the interface.

22.4.3.2. Automatic RTS mode

Automatic RTS mode becomes active when the following conditions occur: †

- RTS (MCR.RTS bit and MCR.AFCE bit are both set)
- FIFO buffers are enabled (FCR.FIFOE bit is set)

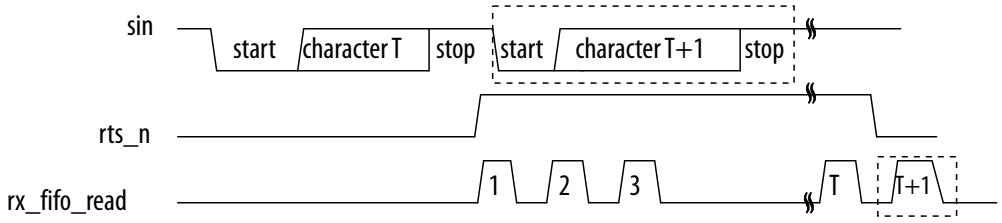
With automatic RTS enabled, the rts_n output pin is forced inactive (high) when the FIFO is almost full; where "almost full" refers to two available slots in the FIFO. When rts_n is connected to the cts_n input pin of another UART device, the other UART stops sending serial data until the receive FIFO buffer has available space (until it is completely empty). †

The selectable receive FIFO buffer threshold values are 1, ¼, ½, and 2 less than full. Because one additional character may be transmitted to the UART after rts_n is inactive (due to data already having entered the transmitter block in the other UART), setting the threshold to 2 less than full allows maximum use of the FIFO buffer with a margin of one character. †

Once the receive FIFO buffer is completely emptied by reading the receiver buffer register (RBR_THR_DLL), rts_n again becomes active (low), signaling the other UART to continue sending data.†

Even when you set the correct MCR bits, if the FIFO buffers are disabled through FCR.FIFOE, automatic flow control is also disabled. When auto RTS is not implemented or disabled, rts_n is controlled solely by MCR.RTS. In the Automatic RTS Timing diagram, the character T is received because rts_n is not detected prior to the next character entering the sending UART transmitter. †

Figure 144. Automatic RTS Timing



22.4.3.3. Automatic CTS mode

Automatic CTS mode becomes active when the following conditions occur: †

- AFCE (MCR.AFCE bit is set)
- FIFO buffers are enabled (through FIFO buffer control register IIR_FCR.FIFOE) bit

When automatic CTS is enabled (active), the UART transmitter is disabled whenever the cts_n input becomes inactive (high). This prevents overflowing the FIFO buffer of the receiving UART. †

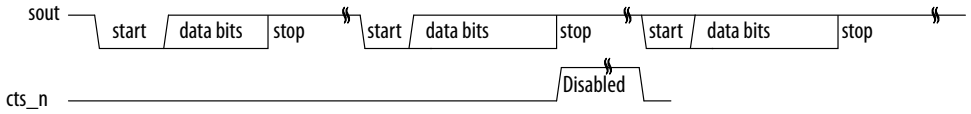
If the cts_n input is not deactivated before the middle of the last stop bit, another character is transmitted before the transmitter is disabled. While the transmitter is disabled, you can continue to write and even overflow to the transmit FIFO buffer. †

Automatic CTS mode requires the following sequence:

1. The UART status register are read to verify that the transmit FIFO buffer is full (UART status register USR.TFNF set to zero). †
2. The current FIFO buffer level is read via the transmit FIFO level (TFL) register. †
3. Programmable THRE interrupt mode must be enabled to access the FIFO buffer full status from the LSR. †

When using the FIFO buffer full status, software can poll this before each write to the transmit FIFO buffer. When the cts_n input becomes active (low) again, transmission resumes. If the FIFO buffers are disabled with the FCR.FIFOE bit, automatic flow control is also disabled regardless of any other settings. When auto CTS is not implemented or disabled, the transmitter is unaffected by cts_n.†

Figure 145. Automatic CTS Timing



22.4.4. Clocks

The UART controller is connected to the l4_sp_clk clock. The clock input is driven by the clock manager.

Related Information

Clock Manager on page 52
For more information, refer to the *Clock Manager* chapter.

22.4.5. Resets

The UART controller is connected to the uart_rst_n reset signal. The reset manager drives the signal on a cold or warm reset.

22.4.5.1. Taking the UART Controller Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

Related Information

Reset Manager on page 134
For more information, refer to the *Reset Manager* chapter.

22.4.6. Interrupts

The assertion of the UART interrupt output signal occurs when one of the following interrupt types are enabled and active: †

Table 225. Interrupt Types and Priority †

Interrupt Type	Priority	Source	Interrupt Reset Control
Receiver line status	Highest	Overrun, parity and framing errors, break condition.	Reading the line status Register.
Received data available	Second	Receiver data available (FIFOs disabled) or RCVR FIFO trigger level reached (FIFOs enabled).	Reading the receiver buffer register (FIFOs disabled) or the FIFO drops below the trigger level (FIFOs enabled)
Character timeout indication	Second	No characters in or out of the Receive FIFO during the last 4 character times and there is at least 1 character in it during this Time.	Reading the receiver buffer Register.
Transmit holding register empty	Third	Transmitter holding register empty (Programmable THRE Mode disabled) or Transmit FIFO at or below threshold (Programmable THRE Mode enabled).	Reading the `IIR` register (if source of interrupt); or, writing into THR (FIFOs or Programmable THRE Mode not enabled) or Transmit FIFO above threshold (FIFOs and Programmable THRE Mode enabled).
Modem Status	Fourth	Clear to send or data set ready or ring indicator or data carrier detect. If auto flow control mode is enabled, a change in CTS (that is, DCTS set) does not cause an interrupt.	Reading the Modem status Register.

You can enable the interrupt types with the interrupt enable register (`IER_DLH`).

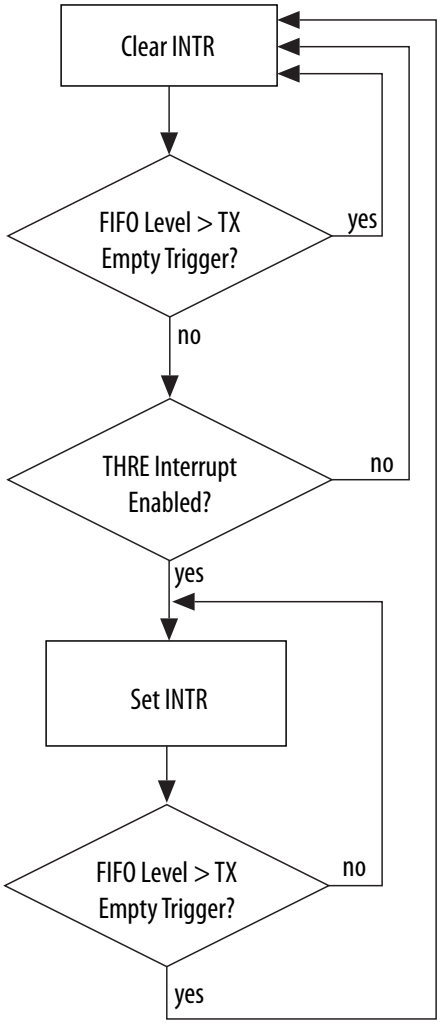
Note: Received Data Available" and "Character Timeout Indication" are enabled by a single bit in the IER_DLH register, because they have the same priority.

Once an interrupt is signaled, you can determine the interrupt source by reading the Interrupt Identity Register (IIR).

22.4.6.1. Programmable THRE Interrupt

The UART has a programmable THRE interrupt mode to increase system performance. You enable the programmable THRE interrupt mode with the interrupt enable register (`IER_DLH.PTIME`). When the THRE mode is enabled, THRE interrupts and the `dma_tx_req` signal are active at and below a programmed transmit FIFO buffer empty threshold level, as shown in the flowchart. †

Figure 146. Programmable THRE Interrupt

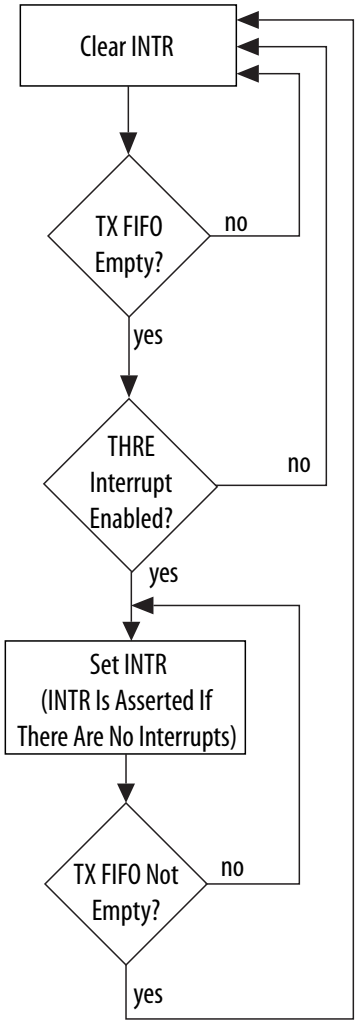


The threshold level is programmed into FCR.TET. The available empty thresholds are empty, 2, ¼, and ½. The optimum threshold value depends on the system's ability to begin a new transmission sequence in a timely manner. However, one of these thresholds should prove optimum in increasing system performance by preventing the transmit FIFO buffer from running empty.

In addition to the interrupt change, line status register (LSR.THRE) also switches from indicating that the transmit FIFO buffer is empty, to indicating that the FIFO buffer is full. This change allows software to fill the FIFO buffer for each transmit sequence by polling LSR.THRE before writing another character. This directs the UART to fill the transmit FIFO buffer whenever an interrupt occurs and there is data to transmit, instead of waiting until the FIFO buffer is completely empty. Waiting until the FIFO buffer is empty reduces performance whenever the system is too busy to respond immediately. You can increase system efficiency when this mode is enabled in combination with automatic flow control.

When not selected or disabled, THRE interrupts and LSR.THRE function normally, reflecting an empty THR or FIFO buffer.

Figure 147. Interrupt Generation without Programmable THRE Interrupt Mode



22.5. DMA Controller Operation

The UART controller includes a DMA controller interface to indicate when the receive FIFO buffer data is available or when the transmit FIFO buffer requires data. The DMA requires two channels, one for transmit and one for receive. The UART controller supports both single and burst transfers.

The FIFO buffer depth (FIFO_DEPTH) for both the RX and TX buffers in the UART controller is 128 entries.

Related Information

DMA Controller on page 512
For more information, refer to the DMA Controller chapter.

22.5.1. Transmit FIFO Underflow

During UART serial transfers, transmit FIFO requests are made to the DMA controller whenever the number of entries in the transmit FIFO is less than or equal to the decoded level of the Transmit Empty Trigger (TET) field in the FIFO Control Register (FCR), also known as the watermark level. The DMA controller responds by writing a burst of data to the transmit FIFO buffer, of length specified as DMA burst length. †

Data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously, that is, when the FIFO begins to empty, another DMA request should be triggered. Otherwise, the FIFO runs out of data (underflow) causing a STOP to be inserted on the UART bus. To prevent this condition, you must set the watermark level correctly. †

Related Information

DMA Controller on page 512
For more information, refer to the DMA Controller chapter.

22.5.2. Transmit Watermark Level

Consider the example where the following assumption is made: †

DMA burst length = FIFO_DEPTH - decoded watermark level of IIR_FCR.TET †

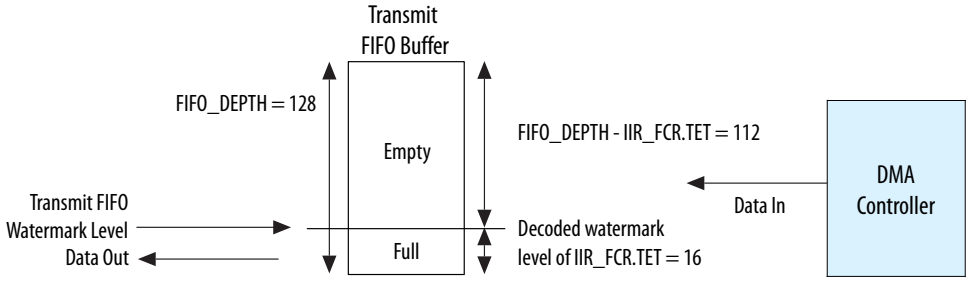
Here the number of data items to be transferred in a DMA burst is equal to the empty space in the transmit FIFO. Consider the following two different watermark level settings: †

22.5.2.1. IIR_FCR.TET = 1

IIR_FCR.TET = 1 decodes to a watermark level of 16.

- Transmit FIFO watermark level = decoded watermark level of IIR_FCR.TET = 16 †
- DMA burst length = FIFO_DEPTH - decoded watermark level of IIR_FCR.TET = 112 †
- UART transmit FIFO_DEPTH = 128 †
- Block transaction size = 448†

Figure 148. Transmit FIFO Watermark Level = 16



The number of burst transactions needed equals the block size divided by the number of data items per burst:

Block transaction size/DMA burst length = 448/112 = 4

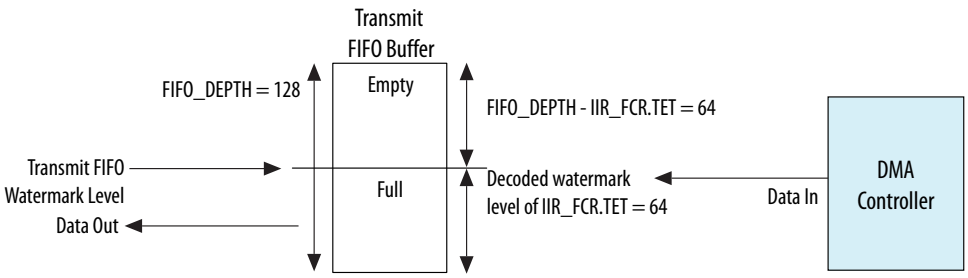
The number of burst transactions in the DMA block transfer is 4. But the watermark level, decoded level of IIR_FCR.TET, is quite low. Therefore, the probability of transmit underflow is high where the UART serial transmit line needs to transmit data, but there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the FIFO becomes empty.

22.5.2.2. IIR_FCR.TET = 3

IIR_FCR.TET = 3 decodes to a watermark level of 64.

- Transmit FIFO watermark level = decoded watermark level of IIR_FCR.TET = 64 †
- DMA burst length = FIFO_DEPTH - decoded watermark level of IIR_FCR.TET = 64†
- UART transmit FIFO_DEPTH = 128 †
- Block transaction size = 448 †

Figure 149. Transmit FIFO Watermark Level = 64



Number of burst transactions in block: †

Block transaction size/DMA burst length = 448/64 = 7 †

In this block transfer, there are 15 destination burst transactions in a DMA block transfer. But the watermark level, decoded level of IIR_FCR.TET, is high. Therefore, the probability of UART transmit underflow is low because the DMA controller has plenty of time to service the destination burst transaction request before the UART transmit FIFO becomes empty. †

Thus, the second case has a lower probability of underflow at the expense of more burst transactions per block. This provides a potentially greater amount of bursts per block and worse bus utilization than the former case. †

Therefore, the goal in choosing a watermark level is to minimize the number of transactions per block, while at the same time keeping the probability of an underflow condition to an acceptable level. In practice, this is a function of the ratio of the rate at which the UART transmits data to the rate at which the DMA can respond to destination burst requests. †

22.5.3. Transmit FIFO Overflow

Setting the DMA burst length to a value greater than the watermark level that triggers the DMA request might cause overflow when there is not enough space in the transmit FIFO to service the destination burst request. Therefore, the following equation must be adhered to in order to avoid overflow: †

DMA burst length <= FIFO_DEPTH - decoded watermark level of IIR_FCR.TET

In case 2: decoded watermark level of IIR_FCR.TET = 64, the amount of space in the transmit FIFO at the time of the burst request is made is equal to the DMA burst length. Thus, the transmit FIFO may be full, but not overflowed, at the completion of the burst transaction. †

Therefore, for optimal operation, DMA burst length must be set at the FIFO level that triggers a transmit DMA request; that is: †

DMA burst length = FIFO_DEPTH - decoded watermark level of IIR_FCR.TET

Adhering to this equation reduces the number of DMA bursts needed for block transfer, and this in turn improves bus utilization. †

The transmit FIFO is not be full at the end of a DMA burst transfer if the UART controller has successfully transmitted one data item or more on the UART serial transmit line during the transfer. †

22.5.4. Receive FIFO Overflow

During UART serial transfers, receive FIFO requests are made to the DMA whenever the number of entries in the receive FIFO is at or above the decoded level of Receive Trigger (RT) field in the FIFO Control Register (IIR_FCR). This is known as the watermark level. The DMA responds by fetching a burst of data from the receive FIFO. †

Data should be fetched by the DMA often enough for the receive FIFO to accept serial transfers continuously, that is, when the FIFO begins to fill, another DMA transfer is requested. Otherwise the FIFO fills with data (overflow). To prevent this condition, the user must set the watermark level correctly. †

22.5.5. Receive Watermark Level

Similar to choosing the transmit watermark level described earlier, the receive watermark level, decoded watermark level of IIR_FCR.RT, should be set to minimize the probability of overflow, as shown in the Receive FIFO Buffer diagram. It is a tradeoff between the number of DMA burst transactions required per block versus the probability of an overflow occurring. †

22.5.6. Receive FIFO Underflow

Setting the source transaction burst length greater than the watermark level can cause underflow where there is not enough data to service the source burst request. Therefore, the following equation must be adhered to avoid underflow: †

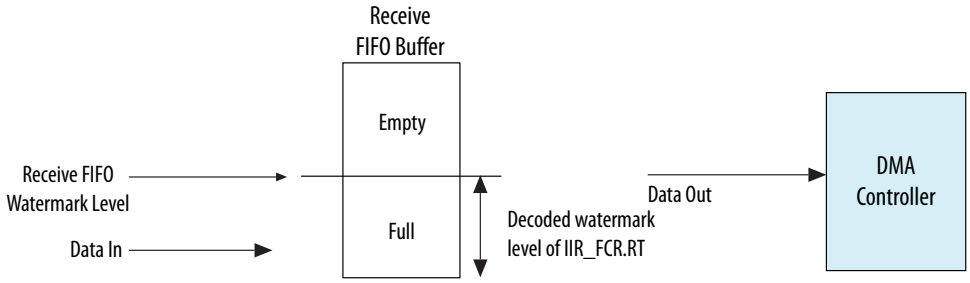
DMA burst length = decoded watermark level of IIR_FCR.RT + 1

If the number of data items in the receive FIFO is equal to the source burst length at the time of the burst request is made, the receive FIFO may be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA burst length should be set at the watermark level, decoded watermark level of `IIR_FCR.RT`. †

Adhering to this equation reduces the number of DMA bursts in a block transfer, which in turn can avoid underflow and improve bus utilization. †

The receive FIFO is not be empty at the end of the source burst transaction if the UART controller has successfully received one data item or more on the UART serial receive line during the burst. †

Figure 150. Receive FIFO Buffer



22.6. UART Controller Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

23. General-Purpose I/O Interface

The hard processor system (HPS) provides three general-purpose I/O (GPIO) interface modules. The GPIO modules are instances of the Synopsys DesignWare APB General Purpose Programming I/O (DW_apb_gpio) peripheral.† (53)

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

23.1. Features of the GPIO Interface

The GPIO interface offers the following features:

- Supports digital debounce
- Configurable interrupt mode
- Supports up to 62 I/O pins

(53) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

23.2. GPIO Interface Block Diagram and System Integration

The figure below shows a block diagram of the GPIO interface. The following table shows a pin table of the GPIO interface:

Figure 151. Arria 10 SoC GPIO

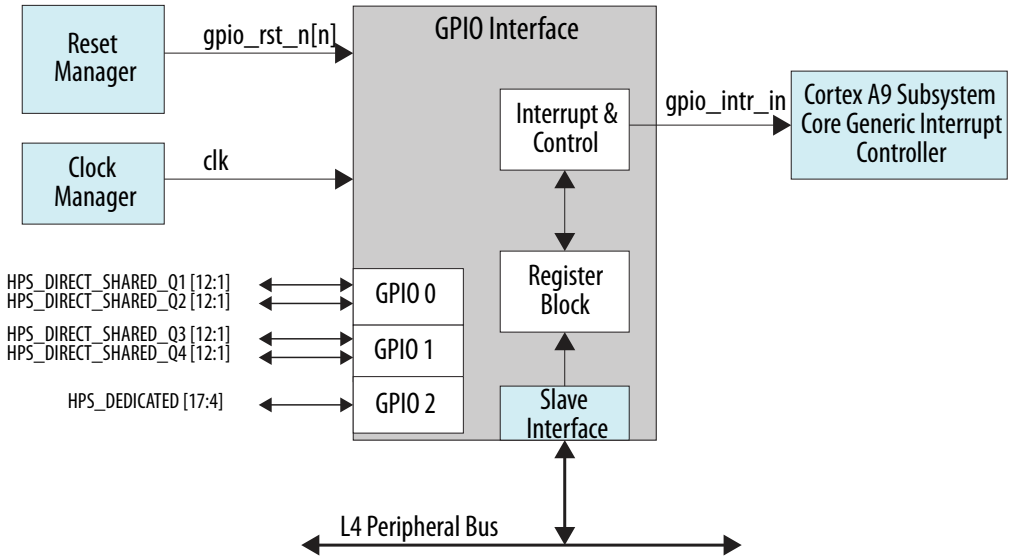


Table 226. GPIO Interface pin table

Pin Name	Mapped to GPIO Signal Name	Comments
HPS_DIRECT_SHARED_Q1 [12:1]	GPIO 0 [11:0]	Shared Input / Output
HPS_DIRECT_SHARED_Q2 [12:1]	GPIO 0 [23:12]	Shared Input / Output
HPS_DIRECT_SHARED_Q3 [12:1]	GPIO 1 [11:0]	Shared Input / Output
HPS_DIRECT_SHARED_Q4 [12:1]	GPIO 1 [23:12]	Shared Input / Output
HPS_DEDICATED [17:4]	GPIO 2 [13:0]	Dedicated Input / Output

Related Information

Arria 10 Device Handbook Volume 1: Device Interfaces and Integration
For more information about I/O banks locations on the Arria 10 device, refer to the *I/O Banks Locations in Arria 10 Devices* section.

23.3. Functional Description of the GPIO Interface

23.3.1. Debounce Operation

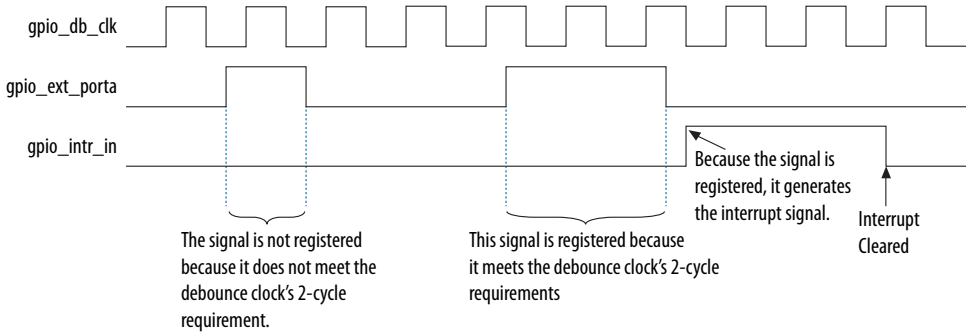
The GPIO modules provided in the HPS include optional debounce capabilities. The external signal can be debounced to remove any spurious glitches that are less than one period of the external debouncing clock, `gpio_db_clk`. †

When input signals are debounced using the `gpio_db_clk` debounce clock, the signals must be active for a minimum of two cycles of the debounce clock to guarantee that they are registered. Any input pulse widths less than a debounce clock

period are filtered out. If the input signal pulse width is between one and two debounce clock widths, it may or may not be filtered out, depending on its phase relationship to the debounce clock. If the input pulse spans two rising edges of the debounce clock, it is registered. If it spans only one rising edge, it is not registered. †

The figure below shows a timing diagram of the debounce circuitry for both cases: a bounced input signal, and later, a propagated input signal.

Figure 152. Debounce Timing With Asynchronous Reset Flip-Flops



Note: Enabling the debounce circuitry increases interrupt latency by two clock cycles of the debounce clock.

23.3.2. Pin Directions

All GPIO pins can be configured to be either input or output signals.

23.3.3. Taking the GPIO Interface Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

Related Information

Module Reset Signals on page 140

23.4. GPIO Interface Programming Model

Debounce capability for each of the input signals can be enabled or disabled under software control by setting the corresponding bits in the `gpio_debounce` register, accordingly. The debounce clock must be stable and operational before the debounce capability is enabled.

Under software control, the direction of the external I/O pad is controlled by a write to the `gpio_swportx_ddr` register. When configured as input mode, reading `gpio_ext_porta` would read the values on the signal of the external I/O pad. When configured as output mode, the data written to the `gpio_swporta_dr` register drives the output buffer of the I/O pad. The same pins are shared for both input and output modes, so they cannot be configured as input and output modes at the same time. †

23.5. General-Purpose I/O Interface Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

24. Timer

The hard processor system (HPS) provides four 32-bit general-purpose timers connected to the level 4 (L4) peripheral bus.The timers optionally generate an interrupt when the 32-bit binary count-down timer reaches zero. The timers are instances of the Synopsys DesignWare APB Timers (DW_apb_timers) peripheral. (54)

Related Information

- Cortex-A9 Microprocessor Unit Subsystem on page 282
  The MPU subsystem provides additional timers. For more information about the timers in the MPU, refer to the *Cortex-A9 MPU* chapter.
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

24.1. Features of the Timer

- Supports interrupt generation
- Supports free-running mode
- Supports user-defined count mode

24.2. Timer Block Diagram and System Integration

Each timer includes a slave interface for control and status register (CSR) access, a register block, and a programmable 32-bit down counter that generates interrupts on reaching zero. The timer operates on a single clock domain driven by the clock manager.

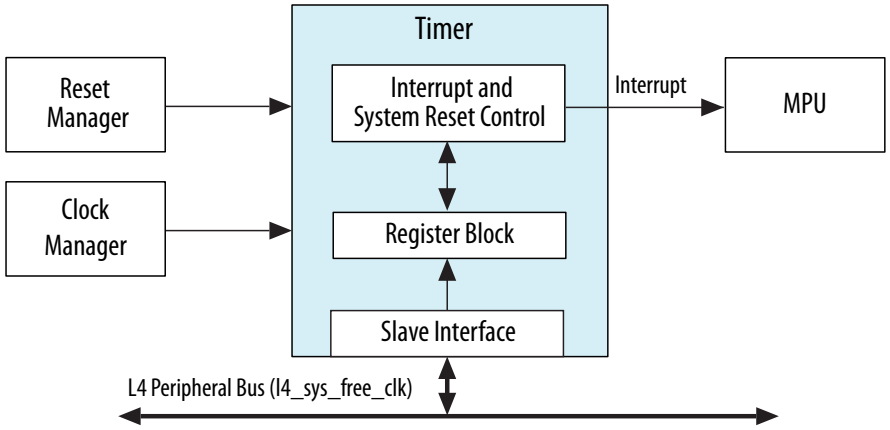
(54) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 153. Timer Block Diagram



24.3. Functional Description of the Timer

The 32-bit timer counts down from a programmed value and generates an interrupt when the count reaches zero. The timer has an independent clock input connected to the system clock signal or to an external clock source. †

The timer supports the following modes of operation:

- Free-running mode—decrementing from the maximum value (0xFFFFFFFF). Reloads maximum value upon reaching zero.
- User-defined count mode—generates a periodic interrupt. Decrements from the user-defined count value loaded from the timer1 load count register (`timer1loadcount`). Reloads the user-defined count upon reaching zero.

The initial value for the timer (that is, the value from which it counts down) is loaded into the timer by the `timer1loadcount` register. The following events can cause a timer to load the initial count from the `timer1loadcount` register: †

- Timer is enabled after being reset or disabled
- Timer counts down to 0

24.3.1. Clocks

Table 227. Timer Clock Characteristics

Timer		System Clock	Notes
System timer 0	`sys_timer0`	`l4_sys_free_clk`	_
System timer 1	`sys_timer1`		
SP timer 0	`sp_timer0`	`l4_sp_clk`	Timer must be disabled if clock frequency changes
SP timer 1	`sp_timer1`		

The timers above are labeled according to the clock they receive. The system timers are connected to the L4_SYS bus and clocked by the `l4_sys_free_clk`. The SP timers are connected to the L4_SP bus and clocked by `l4_sp_clk`.

SP timer 0 and SP timer 1 must be disabled before `l4_sp_clk` is changed to another frequency. You can then re-enable the timer once the clock frequency change takes effect.

Related Information

Clock Manager on page 52
For more information about clock performance, refer to the *Clock Manager* chapter.

24.3.2. Resets

The timers are reset by a cold or warm reset. Resetting the timers produces the following results in the following order:

1. The timer is disabled.
2. The interrupt is enabled.
3. The timer enters free-running mode.
4. The timer count load register value is set to zero.

24.3.3. Interrupts

The timer1 interrupt status (`timer1intstat`) and timer1 end of interrupt (`timer1eoi`) registers handle the interrupts. The `timer1intstat` register allows you to read the status of the interrupt. Reading from the `timer1eoi` register clears the interrupt. †

The timer1 control register (`timer1controlreg`) contains the timer1 interrupt mask bit (`timer1_interrupt_mask`) to mask the interrupt. In both the free-running and user-defined count modes of operation, the timer generates an interrupt signal when the timer count reaches zero and the interrupt mask bit of the control register is high.

If the timer interrupt is set, then it is cleared when the timer is disabled.

24.4. Timer Programming Model

24.4.1. Initialization

To initialize the timer, perform the following steps: †

1. Initialize the timer through the `timer1controlreg` register: †
   - Disable the timer by writing a 0 to the timer1 enable bit (`timer1_enable`) of the `timer1controlreg` register. †

     Note: Before writing to a timer1 load count register (`timer1loadcount`), you must disable the timer by writing a 0 to the `timer1_enable` bit of the `timer1controlreg` register to avoid potential synchronization problems. †
   - Program the timer mode—user-defined count or free-running—by writing a 0 or 1, respectively, to the timer1 mode bit (`timer1_mode`) of the `timer1controlreg` register. †
   - Set the interrupt mask as either masked or not masked by writing a 1 or 0, respectively, to the `timer1_interrupt_mask` bit of the `timer1controlreg` register. †
2. Load the timer counter value into the `timer1loadcount` register. †
3. Enable the timer by writing a 1 to the `timer1_enable` bit of the `timer1controlreg` register. †

24.4.2. Enabling the Timer

When a timer transitions to the enabled state, the current value of `timer1loadcount` register is loaded into the timer counter. †

1. To enable the timer, write a 1 to the `timer1_enable` bit of the `timer1controlreg` register.

24.4.3. Disabling the Timer

When the timer enable bit is cleared to 0, the timer counter and any associated registers in the timer clock domain, are asynchronously reset. †

1. To disable the timer, write a 0 to the `timer1_enable` bit. †

24.4.4. Loading the Timer Countdown Value

When a timer counter is enabled after being reset or disabled, the count value is loaded from the `timer1loadcount` register; this occurs in both free-running and user-defined count modes. †

When a timer counts down to 0, it loads one of two values, depending on the timer operating mode: †

- User-defined count mode—timer loads the current value of the `timer1loadcount` register. Use this mode if you want a fixed, timed interrupt. Designate this mode by writing a 1 to the `timer1_mode` bit of the `timer1controlreg` register. †
- Free-running mode—timer loads the maximum value (0xFFFFFFFF). The timer max count value allows for a maximum amount of time to reprogram or disable the timer before another interrupt occurs. Use this mode if you want a single timed interrupt. Enable this mode by writing a 0 to the `timer1_mode` bit of the `timer1controlreg` register. †

24.4.5. Servicing Interrupts

24.4.5.1. Clearing the Interrupt

An active timer interrupt can be cleared in two ways.

1. If you clear the interrupt at the same time as the timer reaches 0, the interrupt remains asserted. This action happens because setting the timer interrupt takes precedence over clearing the interrupt. †
2. To clear an active timer interrupt, read the `timer1eoi` register or disable the timer. When the timer is enabled, its interrupt remains asserted until it is cleared by reading the `timer1eoi` register. †

24.4.5.2. Checking the Interrupt Status

You can query the interrupt status of the timer without clearing its interrupt.

1. To check the interrupt status, read the `timer1intstat` register. †

24.4.5.3. Masking the Interrupt

The timer interrupt can be masked using the `timer1controlreg` register.

To mask an interrupt, write a 1 to the `timer1_interrupt_mask` bit of the `timer1controlreg` register. †

24.5. Timer Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

25. Watchdog Timer

The watchdog timers are peripherals you can use to recover from system lockup that might be caused by software or system related issues. The hard processor system (HPS) provides two programmable watchdog timers, which are connected to the level 4 (L4) peripheral bus. The watchdog timers are instances of the Synopsys DesignWare APB Watchdog Timer (DW_apb_wdt) peripheral. (55)

The microprocessor unit (MPU) subsystem provides two additional watchdog timers.

Related Information

- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter
- Cortex-A9 Microprocessor Unit Subsystem on page 282
  For more information about the watchdog timers in the MPU, refer to *Cortex A9 Microprocessor Unit Subsystem* chapter.

25.1. Features of the Watchdog Timer

The following list describes the features of the watchdog timer:

- Programmable 32-bit timeout range
- Timer counts down from a preset value to zero, then performs one of the following user-configurable operations:
  - Generates a system reset †
  - Generates an interrupt, restarts the timer, and if the timer is not cleared before a second timeout occurs, generates a system reset
- Dual programmable timeout period, used when the time to wait after the first start is different than that required for subsequent restarts †
- Prevention of accidental restart of the watchdog counter †
- Prevention of accidental disabling of the watchdog counter †
- Pause mode for debugging

(55) Portions © 2017 Synopsys, Inc. Used with permission. All rights reserved. Synopsys & DesignWare are registered trademarks of Synopsys, Inc. All documentation is provided "as is" and without any warranty. Synopsys expressly disclaims any and all warranties, express, implied, or otherwise, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, and any warranties arising out of a course of dealing or usage of trade.

†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

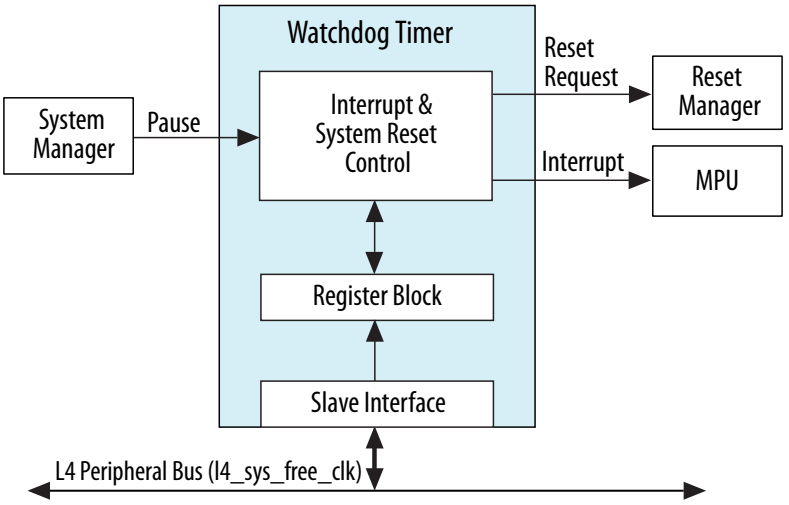
ISO 9001:2015 Registered

25.2. Watchdog Timer Block Diagram and System Integration

Each watchdog timer consists of a slave interface for control and status register (CSR) access, a register block, and a 32-bit down counter that operates on the slave interface clock (`l4_sys_free_clk`). A pause input, driven by the system manager, optionally pauses the counter when a CPU is being debugged.

The watchdog timer drives an interrupt request to the MPU and a reset request to the reset manager.

Figure 154. Watchdog Timer Block Diagram



Related Information

- Reset Manager on page 134
  For more information, refer to the *Reset Manager* chapter.
- Cortex-A9 Microprocessor Unit Subsystem on page 282
  For more information about the watchdog timers in the MPU, refer to *Cortex A9 Microprocessor Unit Subsystem* chapter.

25.3. Functional Description of the Watchdog Timer

25.3.1. Watchdog Timer Counter

Each watchdog timer is a programmable, little-endian down counter that decrements by one on each clock cycle. The watchdog timer supports 16 fixed timeout period values, Software chooses which timeout periods are desired. A timeout period is 2<*n*>`l4_sys_free_clk` clock periods, where *n* is an integer from 16 to 31 inclusive.

Software must regularly restart the timer (which reloads the counter with the restart timeout period value) to indicate that the system is functioning normally. Software can reload the counter at any time by writing to the restart register. If the counter reaches zero, the watchdog timer has timed out, indicating an unrecoverable error has occurred and a system reset is needed.

Software configures the watchdog timer to one of the following output response modes:

- On timeout, generate a reset request.
- On timeout, assert an interrupt request and restart the watchdog timer. Software must service the interrupt and reset the watchdog timer before a second timeout occurs. Otherwise, generate a reset request.

If a restart occurs at the same time the watchdog counter reaches zero, an interrupt is not generated.

Note: After the watchdog timer reaches zero and generates a reset or interrupt, the counter resets and continues to count.

Related Information

- Watchdog Timer Clocks on page 695
- Setting the Timeout Period Values on page 696
- Selecting the Output Response Mode on page 696
- Reloading a Watchdog Counter on page 697

25.3.2. Watchdog Timer Pause Mode

The watchdog timers can be paused during debugging. The watchdog timer pause mode is controlled by the system manager. The following options are available:

- Pause the timer while either CPU0 or CPU1 is in debug mode
- Pause the timer while only CPU1 is in debug mode
- Pause the timer while only CPU0 is in debug mode
- Do not pause the timer

When pause mode is enabled, the system manager pauses the watchdog timer while debugging. When pause mode is disabled, the watchdog timer runs while debugging.

At reset, the watchdog pausing feature is enabled for both CPUs by default.

Related Information

Pausing a Watchdog Timer on page 697

25.3.3. Watchdog Timer Clocks

Each watchdog timer is connected to the `l4_sys_free_clk` clock so that timer operation is not dependent on the phase-locked loops (PLLs) in the clock manager and so that it is always running. This independence allows recovery from software that inadvertently programs the PLLs in the clock manager incorrectly.

Table 228. Watchdog Timer Clocks

Timer	System Clock
`watchdog0`	`l4_sys_free_clk`
`watchdog1`	`l4_sys_free_clk`

Related Information

Clock Manager on page 52
For more information, refer to the *Clock Manager* chapter.

25.3.4. Watchdog Timer Resets

Watchdog timers are reset by a cold or warm reset from the reset manager, and are disabled when exiting reset. †

Related Information

Reset Manager on page 134
For more information, refer to the *Reset Manager* chapter.

25.3.4.1. Taking the Watchdog Timer Out of Reset

When a cold or warm reset is issued in the HPS, the reset manager resets this module and holds it in reset until software releases it.

- Cold reset—HPS I/O are in frozen state (tri-state with a weak pull-up). Pin Mux and GPIO/LoanIO Mux are in the default state.
- Warm reset—HPS I/O retain their configuration. The Pin Mux and GPIO/LoanIO Mux do not change during warm reset, meaning it does not reset to the default/reset value and maintain the current settings. Peripheral IP using the HPS I/O performs proper reset of the signals driving the IO.

After the Cortex-A9 MPCore CPU boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For details about reset registers, refer to "Module Reset Signals".

25.4. Watchdog Timer Programming Model

25.4.1. Setting the Timeout Period Values

The watchdog timers have a dual timeout period. The counter uses the initial start timeout period value the first the timer is started. All subsequent restarts use the restart timeout period. The valid values are $2^{(16+<i>)}$ – 1 clock cycles, where i is an integer from 0 to 15. To set the programmable timeout periods, perform the following actions in no specific order:

Note: Set the timeout values before enabling the timer.

- To set the initial start timeout period, write i to the timeout period for the initialization field (`top_init`) of the watchdog timeout range register (`wdt_torr`).
- To set the restart timeout period, write i to the timeout period field (`top`) of the `wdt_torr` register

25.4.2. Selecting the Output Response Mode

The watchdog timers have two output response modes. To select the desired mode, perform one of the following actions:

- To generate a system reset request when a timeout occurs, write 0 to the output response mode bit (`rmod`) of the watchdog timer control register (`wdt_cr`).
- To generate an interrupt and restart the timer when a timeout occurs, write 1 to the `rmod` field of the `wdt_cr` register.

If a restart occurs at the same time the watchdog counter reaches zero, a system reset is not generated. †

Related Information

Watchdog Timer Counter on page 694

25.4.3. Enabling and Initially Starting a Watchdog Timer

To enable and start a watchdog timer, write the value 1 to the watchdog timer enable bit (`wdt_en`) of the `wdt_cr` register.

25.4.4. Reloading a Watchdog Counter

To reload a watchdog counter, write the value 0x76 to the counter restart register (`wdt_crr`). This unique 8-bit value is used as a safety feature to prevent accidental restarts.

25.4.5. Pausing a Watchdog Timer

Pausing the watchdog timers is controlled by the L4 watchdog debug register (`wddbg`) in the system manager.

Related Information

Features of the System Manager on page 159
For more information, refer to the *System Manager* chapter.

25.4.6. Disabling and Stopping a Watchdog Timer

The watchdog timers are disabled and stopped by resetting them from the reset manager.

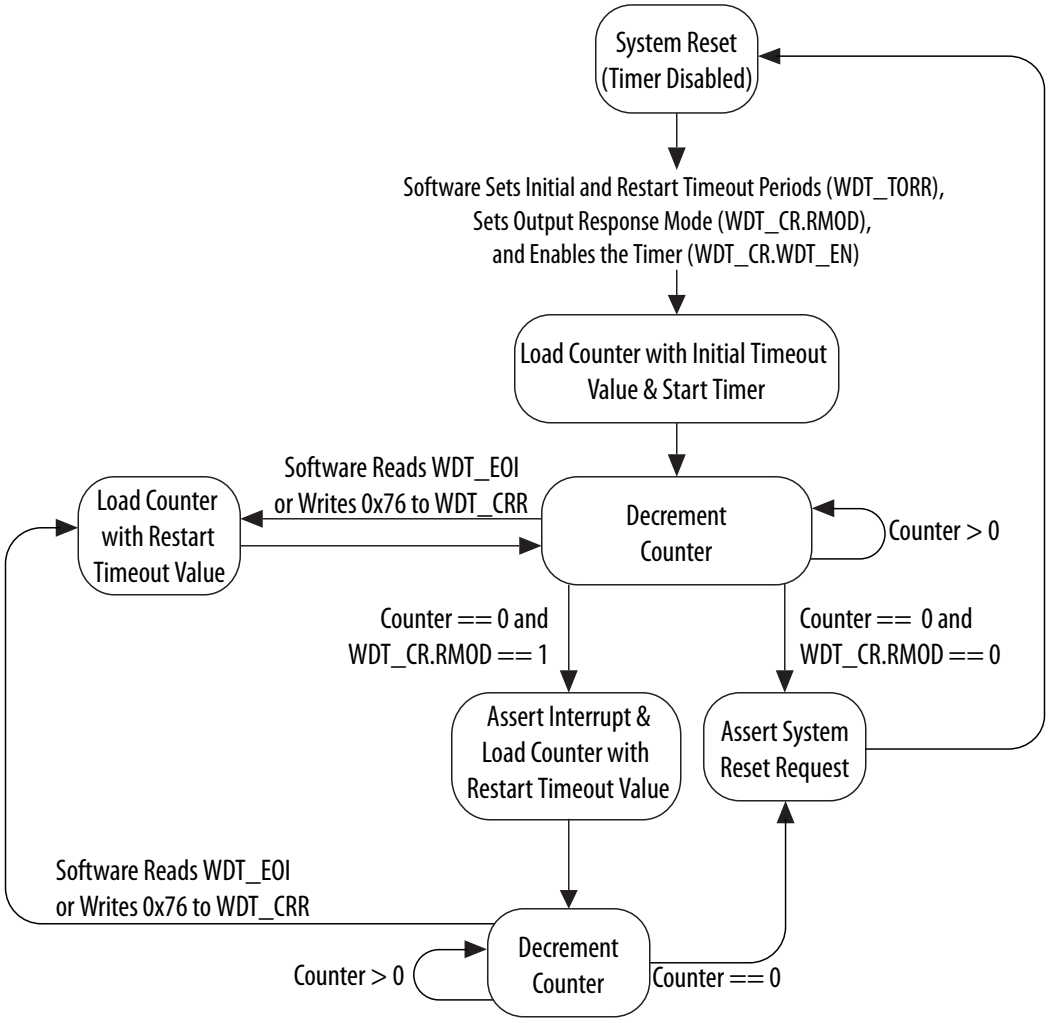
Related Information

Reset Manager on page 134
For more information, refer to the *Reset Manager* chapter.

25.4.7. Watchdog Timer State Machine

The following figure illustrates the behavior of the watchdog timer, including the behavior of both output response modes. Once initialized, the counter decrements at every clock cycle. The state machine remains in the Decrement Counter state until the counter reaches zero, or the watchdog timer is restarted. If software reads the interrupt clear register (`wdt_eoi`), or writes 0x76 to the `wdt_crr` register, the state changes from Decrement Counter to Load Counter with Restart Timeout Value. In this state, the watchdog counter gets reloaded with the restart timeout value, and then the state changes back to Decrement Counter.

Figure 155. Watchdog Timer State Machine



If the counter reaches zero, the state changes based on the value of the output response mode setting defined in the `rmod` bit of the `wdt_cr` register. If the `rmod` bit of the `wdt_cr` register is 0, the output response mode is to generate a system reset request. In this case, the state changes to Assert System Reset Request. In response, the reset manager resets and disables the watchdog timer, and gives software the opportunity to reinitialize the timer.

If the `rmod` bit of the `wdt_cr` register is `1`, the output response mode is to generate an interrupt. In this case, the state changes to Assert Interrupt and Load Counter with Restart Timeout Value. An interrupt to the processor is generated, and the watchdog counter is reloaded with the restart timeout value. The state then changes to the second Decrement Counter state, and the counter resumes decrementing. If software reads the `wdt_eoi` register, or writes `0x76` to the `wdt_crr` register, the state changes from Decrement Counter to Load Counter with Restart Timeout Value. In this state, the watchdog counter gets reloaded with the restart timeout value, and then the state changes back to the first Decrement Counter state. If the counter again

reaches zero, the state changes to Assert System Reset Request. In response, the reset manager resets the watchdog timer, and gives software the opportunity to reinitialize the timer.

25.5. Watchdog Timer Address Map and Register Definitions

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

26. Hard Processor System I/O Pin Multiplexing

The Arria 10 SoC has two banks of flexible I/O pins that are used for hard processor system (HPS) operation, external memory and external peripheral connection. Some of the pins are dedicated to the HPS, and others can be shared with the FPGA fabric. A pin multiplexing mechanism allows the SoC to use the flexible I/O pins in a wide range of configurations.

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

26.1. Features of the HPS I/O Block

The I/O block provides the following functionality and features:

- I/O pins
  - Dedicated I/O: 17 pins supporting clock, resets, boot devices, and other key peripherals. These pins cannot be used by soft logic in the FPGA.
  - Shared I/O: 48 pins available for HPS external memory and peripherals. Shared I/O pins can also be used by FPGA logic.

  Note: The HPS also interfaces with an SDRAM memory controller. This interface is separate from the dedicated and shared I/O pins discussed in this chapter.
- I/O multiplexing
  - Selects pins used by each HPS peripheral
  - Can assign shared I/O pins to FPGA logic
  - Can expose HPS peripheral interfaces to FPGA logic

    Note: When routed to the FPGA, some HPS peripherals require additional pipeline support in the connected soft logic. Refer to the relevant HPS peripheral chapter for details.

  I/O multiplexing is configured when you instantiate the HPS component.

Related Information

Intel Arria 10 External Memory Interfaces IP User Guide
For details about memory I/O pins in the SoC hard memory controller, refer to "Intel Arria 10 EMIF for Hard Processor Subsystem" in the "Intel Arria 10 EMIF IP Product Architecture" chapter of the *Intel Arria 10 External Memory Interfaces IP User Guide*.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

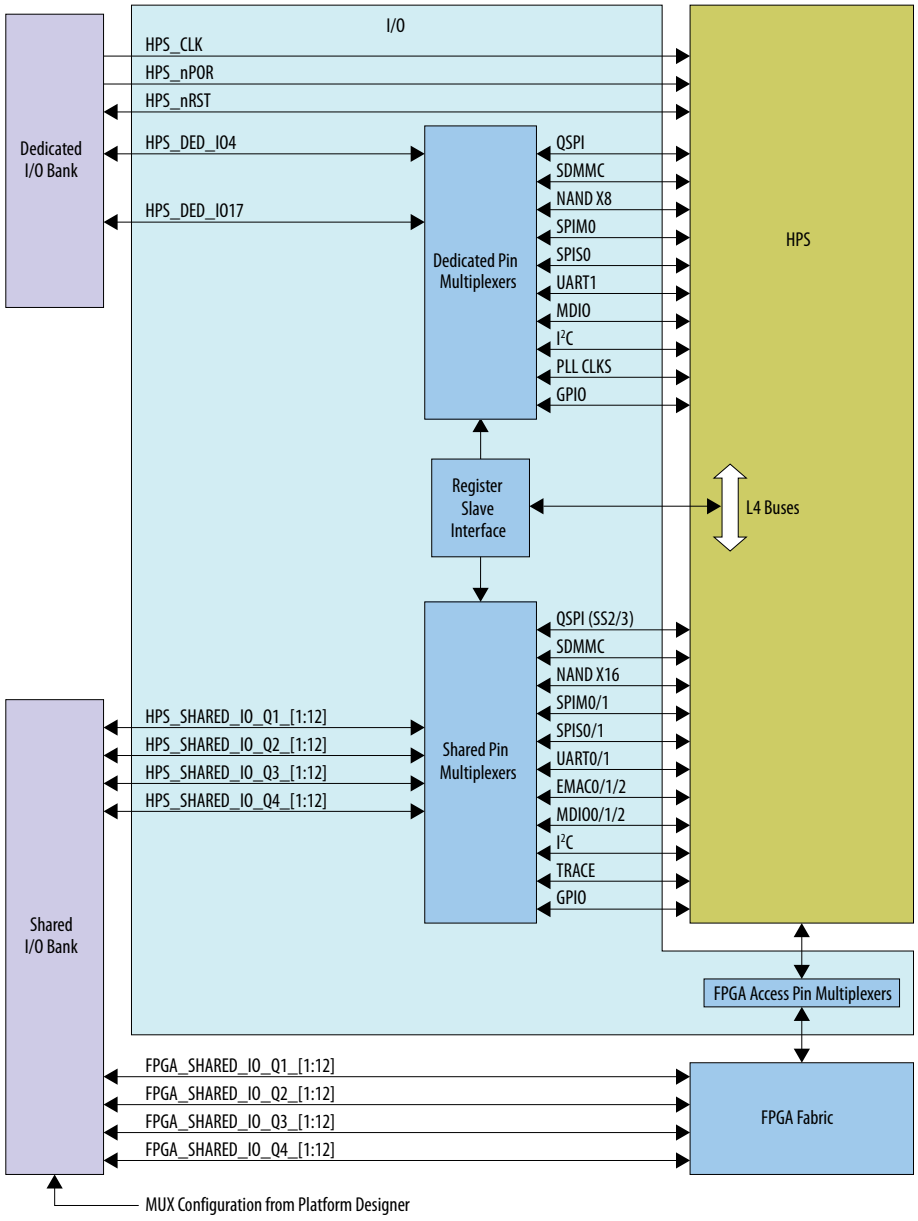
ISO 9001:2015 Registered

26.2. HPS I/O Block Diagram and System Integration

The HPS I/O block consists of the following sub-blocks:

- Dedicated pin multiplexers (MUXes) – MUXes for the dedicated I/O bank
- Shared pin multiplexers – MUXes for the shared I/O bank
- FPGA access pin multiplexers – MUXes for HPS peripheral connections to the FPGA fabric
- Register slave interface – Provides access to control registers, which allow the bootloader to initialize I/O pins and HPS peripheral interfaces at system startup

Figure 156. HPS I/O Block Diagram



Related Information

Control Registers on page 705

26.3. Functional Description of the HPS I/O

26.3.1. Dedicated I/O Pins

The HPS has 17 dedicated I/O pins. Three are used as clock, cold reset and warm reset. The remaining 14 dedicated I/O pins are dedicated to boot devices and other commonly-used peripherals. The following HPS peripherals can be served by dedicated I/O pins:

- GPIO2
- NAND
- SD/MMC
- QSPI
- CM_PLL
- UART1
- SPI0
- EMAC0 MDIO interface
- EMAC1 MDIO interface
- EMAC2 MDIO interface

Each of these peripherals can alternatively be routed through the FPGA. You can configure this routing when you instantiate the HPS component.

Related Information

- Booting and Configuration on page 771
  Details about boot select pins
- FPGA Access on page 705
  Information about routing HPS peripheral interfaces to the FPGA
- Configuring HPS I/O Multiplexing on page 708
  Information about configuring the HPS I/O MUXes

26.3.1.1. Warm Reset Pin

The warm reset signal is a bidirectional signal; a warm reset event can drive into the HPS, or a warm reset event can be generated by the HPS and drive out of this pin.

26.3.1.2. Boot Pins

The following dedicated I/O pins convey boot source information:

- `HPS_DEDICATED_6/BOOTSEL2`
- `HPS_DEDICATED_10/BOOTSEL1`
- `HPS_DEDICATED_11/BOOTSEL0`

Typically, software samples these pins at cold reset.

Related Information

Arria 10 Hard Processor System Pin Information spreadsheet
Details of available HPS pin MUXing options

26.3.2. Shared I/O Pins

There are 48 HPS peripheral pins that are shared with the FPGA core. They are divided into four quadrants of 12 signals per quadrant. Each quadrant can be assigned to the HPS or the FPGA fabric.

In each shared I/O quadrant, all 12 I/Os are assigned either to the FPGA or to the HPS. It is not possible to divide the I/Os in a quadrant between the FPGA and HPS.

When a quadrant is assigned to the FPGA fabric, you can connect FPGA soft logic to the shared I/O pins with the assignment editor, just as for any other FPGA I/O pins. Soft logic can use the shared I/O pins to interface to off-chip resources. However, pins in that quadrant are no longer directly available to HPS peripherals. Enabled HPS peripherals can be assigned to another quadrant, or else their interfaces can be routed to the FPGA fabric.

The following HPS peripherals can be assigned to shared I/O pins:

- EMAC0
- EMAC1
- EMAC2
- GPIO0
- GPIO1
- I2C0
- I2C1
- NAND
- QSPI
- SD/MMC
- SPI0
- SPI1
- TRACE
- UART0
- UART1
- USB0
- USB1

Each of these peripherals except USB0 and USB1 can alternatively be routed through the FPGA. You configure this routing when you instantiate the HPS component in Platform Designer.

Note: Although the shared I/O pins are configured through the control registers, software cannot reconfigure the shared I/O pins after I/O configuration is complete. There is no support for dynamically changing the pin MUX selections for shared pins.

Related Information

- FPGA Access on page 705
  Information about routing HPS peripheral interfaces to the FPGA
- HPS I/O Block Diagram and System Integration on page 701
  Refer to the HPS I/O block diagram for an overview of how the HPS and FPGA share the shared I/O bank.
- Configuring HPS I/O Multiplexing on page 708
  Information about configuring the HPS I/O MUXes

26.3.3. FPGA Access

Most HPS peripheral interfaces can be connected into the FPGA fabric, instead of to the dedicated or shared I/O pins.

HPS peripherals connect to the FPGA fabric through the FPGA access pin MUX. When connected to the FPGA fabric, peripheral interfaces are exposed as ports of the HPS component.

Connecting HPS peripherals to the FPGA fabric can be a strategy to make optimal use of the I/O pins available to the HPS. For example, you can route HPS peripherals through the FPGA if your design requires more I/Os than the HPS I/O block provides.

All HPS peripherals except the USB 2.0 OTG controllers can interface to the FPGA fabric.

Related Information

Configuring HPS I/O Multiplexing on page 708
Information about configuring the HPS I/O MUXes

26.3.4. Control Registers

The HPS provides control registers that allow the system to initialize the following I/O parameters at system startup:

- Pins assigned to each HPS peripheral
- Shared I/O pins optionally assigned to FPGA logic
- HPS peripheral interfaces optionally exposed to FPGA logic
- I/O cell configuration

The HPS I/O control registers can only be accessed in secure mode.

Control registers can be divided into the following groups:

- Dedicated pin MUX registers
- Dedicated configuration registers
- Shared pin MUX registers
- FPGA access MUX registers

You set up the control registers when you instantiate the HPS component at the time of system generation. When you configure the HPS component, Platform Designer determines the correct register settings, and places them in the Intel Quartus Prime I/O programming file.

Related Information

- I/O Pin MUX Address Map and Register Definitions for Arria 10 on page 709
- Configuring HPS I/O Multiplexing on page 708
  Information about configuring the HPS I/O MUXes

26.3.4.1. Dedicated Pin MUX Registers

Dedicated pin MUX registers control the functions of the dedicated pins. One register is provided for each dedicated I/O pin.

The HPS provides pin MUX registers, `pinmux_dedicated_io_4` through `pinmux_dedicated_io_17`, for each of the dedicated pins HPS_DEDICATED_4 through HPS_DEDICATED_17. Each pin MUX register contains a 4-bit MUX select field to select the function of the dedicated pin. At a cold reset event these fields are reset to 15, selecting a general purpose I/O function. A warm reset event does not affect these registers.

The registers for dedicated I/O pins HPS_DEDICATED_1 through HPS_DEDICATED_3 provide no functionality, because these pins are always used as the HPS clock and reset pins.

Note: Although the dedicated I/O pins are configured through the control registers, Intel recommends against reconfiguring the dedicated I/O pins after I/O configuration is complete.

Related Information

- I/O Pin MUX Address Map and Register Definitions for Arria 10 on page 709
- Arria 10 Hard Processor System Pin Information spreadsheet
  Details of available HPS pin MUXing options

26.3.4.2. Dedicated Configuration Registers

Configuration registers for each dedicated I/O pin allow software to control the corresponding I/O cell. These registers, `configuration_dedicated_io_1` through `configuration_dedicated_io_17`, allow software to set the following characteristics:

- Termination
- Slew rate
- Input buffer settings

These registers are not affected by a warm reset.

One configuration register, `configuration_dedicated_io_bank`, is also provided to select the I/O voltage for the dedicated I/O bank. At cold reset this register defaults to 0x2 (2.5-3.0 volt operation). This register is not affected by a warm reset.

Typically, the boot code programs this register when the system boots up. The boot code reads the BOOTSEL0, BOOTSEL1, and BOOTSEL2 pins and then sets the dedicated I/O bank to either 1.8 or 2.5-3.0 volt operation, depending on the pin status.

Note: Although the dedicated I/O pins are configured through the control registers, Intel recommends against reconfiguring the dedicated I/O pins after I/O configuration is complete.

Related Information

- I/O Pin MUX Address Map and Register Definitions for Arria 10 on page 709
- Configuring HPS I/O Multiplexing on page 708
  Information about configuring the HPS I/O MUXes

26.3.4.3. Shared Pin MUX Registers

There are 48 shared pin MUX registers, one for each shared I/O pin. The pin MUX register names correspond to the signal names, as listed in the following table.

Table 229. Shared Pin MUX Register Names and Signals

Shared Pin MUX Register Name	HPS Signal Name	FPGA Signal Name
pinmux_shared_io_q1_1	HPS_SHARED_IO_Q1_1	FPGA_SHARED_IO_Q1_1
...	...	...
pinmux_shared_io_q1_12	HPS_SHARED_IO_Q1_12	FPGA_SHARED_IO_Q1_12
pinmux_shared_io_q2_1	HPS_SHARED_IO_Q2_1	FPGA_SHARED_IO_Q2_1
...	...	...
pinmux_shared_io_q2_12	HPS_SHARED_IO_Q2_12	FPGA_SHARED_IO_Q2_12
pinmux_shared_io_q3_1	HPS_SHARED_IO_Q3_1	FPGA_SHARED_IO_Q3_1
...	...	...
pinmux_shared_io_q3_12	HPS_SHARED_IO_Q3_12	FPGA_SHARED_IO_Q3_12
pinmux_shared_io_q4_1	HPS_SHARED_IO_Q4_1	FPGA_SHARED_IO_Q4_1
...	...	...
pinmux_shared_io_q4_12	HPS_SHARED_IO_Q4_12	FPGA_SHARED_IO_Q4_12

Each shared pin MUX register contains a 4-bit MUX select field to select the function of the shared I/O pin. At a cold reset event these fields default to 15, selecting a general purpose I/O function. A warm reset event does not affect these registers.

Note: Although the shared I/O pins are configured through the control registers, software cannot reconfigure the shared I/O pins after I/O configuration is complete. There is no support for dynamically changing the pin MUX selections for shared pins.

Related Information

- I/O Pin MUX Address Map and Register Definitions for Arria 10 on page 709
- Arria 10 Hard Processor System Pin Information spreadsheet
  Details of available HPS pin MUXing options

26.3.4.4. FPGA Access MUX Registers

The FPGA access ("use FPGA") MUX registers select whether each HPS peripheral uses HPS I/O pins or is routed to the FPGA fabric.

All peripherals except the USBs can be routed to the FPGA. The following FPGA access registers are available:

- `pinmux_emac0_usefpga`
- `pinmux_emac1_usefpga`
- `pinmux_emac2_usefpga`
- `pinmux_i2c0_usefpga`
- `pinmux_i2c1_usefpga`
- `pinmux_i2c_emac0_usefpga`
- `pinmux_i2c_emac1_usefpga`
- `pinmux_i2c_emac2_usefpga`
- `pinmux_nand_usefpga`
- `pinmux_qspi_usefpga`
- `pinmux_sdmmc_usefpga`
- `pinmux_spim0_usefpga`
- `pinmux_spim1_usefpga`
- `pinmux_spis0_usefpga`
- `pinmux_spis1_usefpga`
- `pinmux_uart0_usefpga`
- `pinmux_uart1_usefpga`

At cold reset, the FPGA access registers default to 0, selecting the HPS I/O pins. These registers are not affected by a warm reset event.

Note: Although the FPGA access MUX is configured through the control registers, Intel recommends against reconfiguring the FPGA access MUX after I/O configuration is complete.

Related Information

- I/O Pin MUX Address Map and Register Definitions for Arria 10 on page 709
- Configuring HPS I/O Multiplexing on page 708
  Information about configuring the HPS I/O MUXes

26.3.5. Configuring HPS I/O Multiplexing

HPS I/O multiplexing is configured when you generate the system with Platform Designer.

26.3.5.1. Configuring Multiplexing at System Generation

There are 48 HPS peripheral pins that are shared with the FPGA core. They are divided into four quadrants of 12 signals per quadrant. Each quadrant can be configured with Platform Designer to be assigned to the HPS or the FPGA fabric.

The select for this multiplexer is in the I/O configuration shift registers (IOCSRs) which are configured by the Intel Quartus Prime I/O programming file.

When you configure the HPS component, Platform Designer determines the correct register settings, and creates a device tree for the boot loader. When the system boots up, the boot loader configures the registers before configuring the I/O chain.

26.3.5.2. Configuring Multiplexing at Boot Time

Typically, the boot code programs the `configuration_dedicated_io_bank` register when the system boots up. The boot code reads the BOOTSEL0, BOOTSEL1, and BOOTSEL2 pins and then sets the dedicated I/O bank to either 1.8 V or 2.5-3.0 V operation, depending on the pin status.

Related Information

- I/O Pin MUX Address Map and Register Definitions for Arria 10 on page 709
- Control Registers on page 705

26.4. Test Considerations

The dedicated I/O pins are chained into the full chip JTAG boundary scan chain. However, in some power modes the HPS can be off or disabled. The SoC device provides a bypass MUX to remove the dedicated I/O pins from the scan chain in this situation.

The boundary scan phase is not required to include I/O configuration shift register (IOCSR) configuration for the CONFIG_IO instruction. The only requirement is that no software be executing on the HPS during boundary scan.

If CONFIG_IO mode is active during the boundary scan phase, the HPS is in cold reset, preventing any software interference with the boundary scan. However, if the I/Os are configured in user mode, CONFIG_IO mode is not active during the scan phase, and you must implement safeguards to prevent software from executing during scan.

Related Information

Arria 10 Core Fabric and General Purpose I/Os Handbook
For information about JTAG boundary-scan testing in Arria 10 devices, refer to "JTAG Boundary-Scan Testing" in the *Arria 10 Devices* chapter of the *Arria 10 Core Fabric and General Purpose I/O Handbook*.

26.5. I/O Pin MUX Address Map and Register Definitions for Arria 10

For complete HPS address map and register definitions, refer to the Intel Arria 10 HPS Register Address Map and Definitions.

27. Introduction to the HPS Component

The hard processor system (HPS) component is a wrapper that interfaces logic in the user design to the HPS hard logic, simulation models, BFMs, and software handoff files. It instantiates the HPS hard logic in the user design; and enables other soft components to interface with the HPS hard logic. The HPS component itself has a small footprint in the FPGA fabric, because its only purpose is to enable soft logic to connect to the extensive hard logic in the HPS. You can connect soft logic to the HPS.

After the soft logic is connected to the HPS, Platform Designer (Standard) ensures the following features:

- Interoperability by adapting Avalon Memory-Mapped (Avalon-MM) interfaces to AXI
- Handle data width mismatches
- Clock domain transfer crossing

This allows you to integrate IP from Intel, 3rd party IP cores, and custom IP cores to the HPS without having to create integration logic.

For a description of the HPS and its integration into the system on a chip (SoC), refer to the *Intel Intel Arria 10 Device Datasheet*.

For a description of the HPS system architecture and features, refer to the "Introduction to the Hard Processor" and the CoreSight Debug and Trace chapters in the *Intel Intel Arria 10 Device Datasheet*.

For more information about instantiating the HPS component, refer to the *Instantiating the HPS Component* chapter in the *Hard Processor System Technical Reference Manual*.

For more information about the HPS component interfaces, refer to the *HPS Component Interfaces* chapter in the *Hard Processor System Technical Reference Manual*.

For more information about simulating the HPS component, refer to the *Simulating the HPS Component* chapter in the *Hard Processor System Technical Reference Manual*.

Related Information

- Intel Intel Arria 10 Device Datasheet
- Simulating the HPS Component on page 747
- HPS Component Interfaces on page 731
- Instantiating the HPS Component on page 716
- Introduction to the Hard Processor System on page 28
- CoreSight Debug and Trace on page 322

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter

27.1. MPU Subsystem

The MPU subsystem features the dual Arm Cortex-A9 MPCore processors.

Related Information

Cortex-A9 Microprocessor Unit Subsystem on page 282

27.2. Arm CoreSight Debug Components

The following lists the Arm CoreSight debug components:

- Debug Access Port (DAP)
- System Trace Macrocell (STM)
- Trace Funnel
- Embedded Trace FIFO (ETF)
- AMBA Trace Bus Replicator (Replicator)
- Embedded Trace Router (ETR)
- Trace Port Interface Unit (TPIU)
- Embedded Cross Trigger (ECT)
- Program Trace Macrocell (PTM)

Related Information

CoreSight Debug and Trace on page 322

27.3. Interconnect

The interconnect consists of the L3 interconnect, SDRAM L3 interconnect, and level 4 (L4) buses. The L3 interconnect is an network-on-chip (NOC) interconnect module.

Related Information

System Interconnect on page 200

27.4. HPS-to-FPGA Interfaces

The HPS-to-FPGA interfaces provide a variety of communication channels between the HPS and the FPGA fabric. The HPS is highly integrated with the FPGA fabric, resulting in thousands of connecting signals. Some of the HPS-to-FPGA interfaces include:

- FPGA-to-HPS port
- HPS-to-FPGA port
- Lightweight HPS-to-FPGA port
- FPGA-to-SDRAM interface

Related Information

HPS-FPGA Bridges on page 248

27.5. Memory Controllers

The following lists the memory controller peripherals:

- SDRAM L3 Interconnect
- NAND Flash Controller
- Quad SPI Controller
- SD/MMC Controller

Related Information

- System Interconnect on page 200
- NAND Flash Controller on page 370
- SD/MMC Controller on page 405
- Quad SPI Flash Controller on page 490

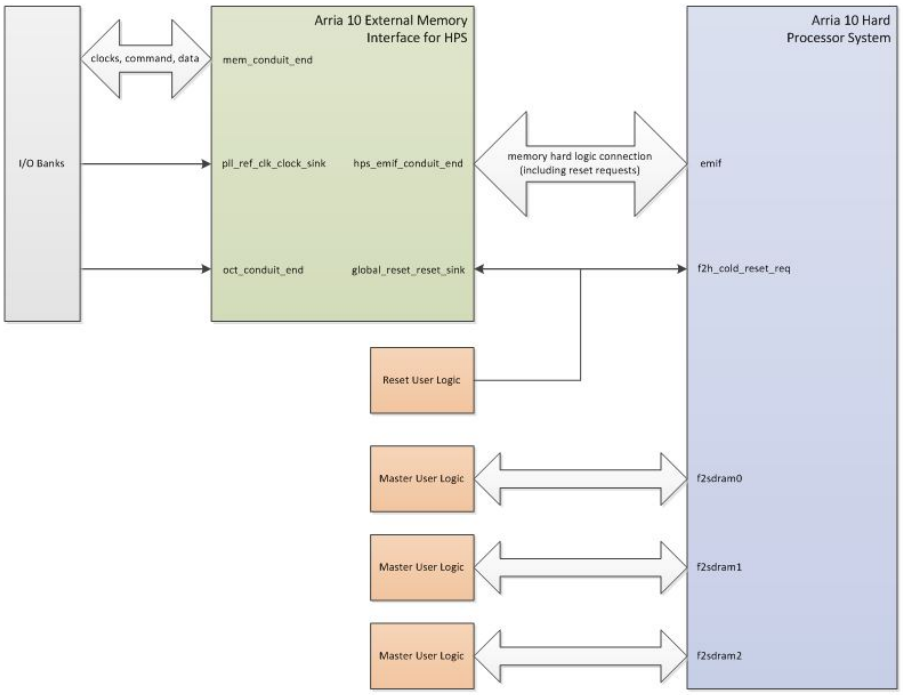
27.5.1. HPS SDRAM Controller

There are several memory interfaces connected, but only one connection to each interface is allowed at one time. You must instantiate a separate SDRAM controller and connect it to the HPS.

27.5.1.1. HMC and HPS Connectivity

The following figure shows that the HMC clock comes from the I/O itself, the HPS can reset the memory through the dedicated hard-logic connection between them, and any logic in the FPGA that resets the HMC must also be connected to the HPS.

Figure 157. HMC and HPS Connectivity



27.5.1.2. Clocks

You must connect the SDRAM controller to a reference clock that comes from the I/O column where the RAM controller is located.

27.5.1.3. Resets

There are two ways that the SDRAM controller can be connected to reset.

- One way - Reset and status connection is automatic by just connecting the SDRAM Hard Memory Controller (HMC) to the HPS. You access these wires in the reset manager so that you can reset the HMC and find out when the reset is complete.
- Second way - The HMC reset input from the FPGA fabric. This signal is called `global_reset_n` and is one of the signals exposed to the FPGA fabric from the HMC when you instantiate it in Platform Designer (Standard). The purpose of this signal is to reset the HMC from the FPGA side of the fabric. Before the FPGA fabric is configured, the memory controller ignores this signal which is why the memory controller can become live before the fabric is configured. This also means that the user design that gets configured into the fabric must drive it because once the fabric is configured, the HMC uses `global_reset_n` as an input source.

Note: You must connect the input source for `global_reset_n` to the cold reset input for the HPS. If you put the SDRAM into reset but that same reset source does not connect to the HPS, the memory is then wiped out while the HPS is still trying to access it. This causes the software to crash.

Related Information

Functional Description - HPS Memory Controller

27.6. Support Peripherals

The following lists the support peripherals:

- Clock Manager
- Reset Manager
- Security Manager
- System Timers
- System Watchdog Timers
- Direct Memory Access (DMA) Controller
- FPGA Manager

Related Information

- Clock Manager on page 52
- Reset Manager on page 134
- FPGA Manager on page 154
- System Manager on page 159
- SoC Security on page 168
  For more information, refer to the *Security Manager* chapter in the *Hard Processor System Technical Reference Manual*.
- DMA Controller on page 512
- Timer on page 688
- Watchdog Timer on page 693

27.7. Interface Peripherals

The following lists the interface peripherals:

- Ethernet Media Access Controller
- USB 2.0 On-The-Go (OTG) Controllers
- I^2C Controllers
- UART
- SPI Master Controllers
- SPI Slave Controllers
- GPIO Interfaces

Related Information

- Ethernet Media Access Controller on page 520
- USB 2.0 OTG Controller on page 591
- SPI Controller on page 608
- I2C Controller on page 643

- UART Controller on page 670
- General-Purpose I/O Interface on page 684

27.8. On-Chip Memories

The following lists the on-chip memories:

- On-Chip RAM
- Boot ROM

Related Information

On-Chip Memory on page 364

28. Instantiating the HPS Component

This chapter describes the parameters available in the hard processor system (HPS) component parameter editor, which opens when you add or edit an HPS component. You instantiate the HPS component in Platform Designer (Standard).

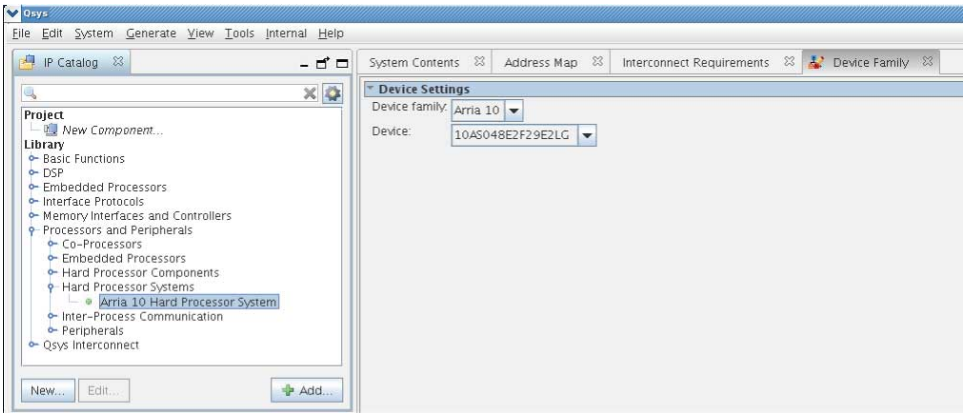
Related Information

- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter
- Intel Intel Arria 10 Device Datasheet
- Intel® Quartus® Prime Standard Edition Handbook Volume 1: Design and Synthesis
  For general information about using Platform Designer (Standard), refer to the *Creating a System with Platform Designer (Standard)* chapter in the *Quartus® Prime* Handbook.

28.1. Using the HPS Parameter Editor

1. Install the Arria 10 ACDS version 17.0.
2. Open the Quartus Prime software.
3. Open Platform Designer (Standard) by selecting **Tools ➤ Platform Designer (Standard)**.
4. In the **Device Family** tab, select a 10ASxxx device from the **Device** pull-down menu.
5. In the **IP Catalog** tab, under Library, select **Processors and Peripherals ➤ Hard Processor Systems ➤ Arria 10 Hard Processor System**.

Figure 158. Platform Designer (Standard) Device Family Tab



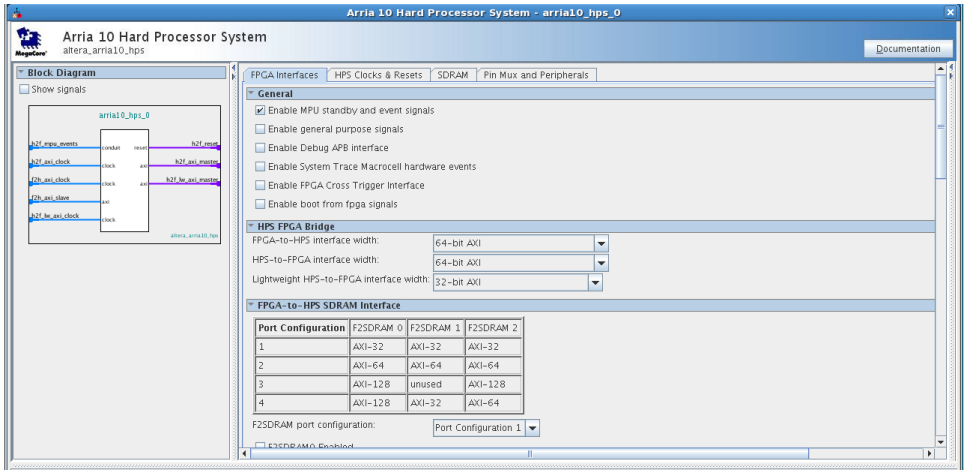
Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

28.2. FPGA Interfaces

The **FPGA Interfaces** tab is one of four tabs on the HPS Component. This tab contains several groups with the parameters shown in the following figure.

Figure 159. FPGA Interfaces Tab



Related Information

Introduction to the HPS Component on page 710

28.2.1. General Interfaces

When enabled, the interfaces described in the following table become visible in the HPS component.

Table 230. General Parameters

Parameter Name	Parameter Description	Interface Name
Enable MPU standby and event signals	Enables interfaces that perform the following functions: • Notify the FPGA fabric that the microprocessor unit (MPU) is in standby mode. • Wake up an MPCore processor from a wait for event (WFE) state.	`h2f_mpu_events`
	Input for FPGA to signal events to both processors. This signal must be asserted high for at least two MPU clock cycles to be visible to either Cortex-A9 processor core. This signal is used to wake either processor core from standby mode.	`h2f_mpu_eventi`
	Output event from either processor sent to the FPGA. The output event is a multiple cycle pulse so logic in the FPGA should detect it using a rising edge detector circuit to detect the occurrence of the event. This signal is asserted each time either processor executes the SEV instruction.	`h2f_mpu_evento`

continued...

Parameter Name	Parameter Description	Interface Name
	Output per processor that indicates if the processor is in WFE standby mode. When high the processor is in WFE standby mode.	h2f_mpu_standbywfe [1..0]
	Output per processor that indicates if the processor is in WFI standby mode. When high the processor is in WFI standby mode.	h2f_mpu_standbywfi [1..0]
Enable general purpose signals	Enables a pair of 32-bit unidirectional general-purpose interfaces between the FPGA fabric and the FPGA manager in the HPS portion of the SoC device.	h2f_gp • h2f_gp_in [31..0] • h2f_gp_out [31..0]
Enable Debug APB interface	Enables debug interface to the FPGA, allowing access to debug components in the HPS. For more information, refer to the *CoreSight Debug and Trace* chapter.	h2f_debug_apb • h2f_dbg_apb_PADDR [17..0] • h2f_dbg_apb_PADDR31 • h2f_dbg_apb_PENABLE • h2f_dbg_apb_PRDATA [31..0] • h2f_dbg_apb_PREADY • h2f_dbg_apb_PSEL • h2f_dbg_apb_PSLVERB • h2f_dbg_apb_PWDATA [31..0] • h2f_dbg_apb_PWRITE h2f_debug_apb_sideband • h2f_debug_apb_PCLKEN • h2f_debug_apb_DBG_APB_DISABLE h2f_debug_apb_clock • h2f_abg_apb_clk h2f_debug_apb_reset • h2f_dbg_apb_rst_n
Enable System Trace Macrocell hardware events	Enables system trace macrocell (STM) hardware events, allowing logic inside the FPGA to insert messages into the trace stream. For more information, refer to the *CoreSight Debug and Trace* chapter.	f2h_stm_hw_events • f2h_stm_hwevents [27..0]
Enable FPGA Cross Trigger interface	Enables the cross trigger interface (CTI), which allows trigger sources and sinks to interface with the embedded cross trigger (ECT). For more information, refer to the *CoreSight Debug and Trace* chapter. If this interface has to be connected to a SignalTap II instance in the FPGA fabric, then it has to be left disabled in Platform Designer (Standard).	h2f_cti • h2f_cti_trig_in [7..0] • h2f_cti_trig_out_ack [7..0] • h2f_cti_trig_out [7..0] • h2f_cti_trig_in_ack [7..0]
Enable boot from FPGA signals	Enables an input to the HPS indicating whether a preloader is available in the on-chip memory of the FPGA. This option also enables a separate input to the HPS indicating a fallback preloader is available in the FPGA memory. A fallback preloader is used when there is no valid preloader image found in flash memory.	f2h_boot_from_fpga

continued...

Parameter Name	Parameter Description	Interface Name
	For more information, refer to *Appendix A: Booting and Configuration*.	
	This is an active high signal which the Boot ROM polls to determine when the FPGA is configured and the memory located at offset 0x0 from the FPGA to HPS bridge is ready to be written to. When the FPGA is not configured the hardware drives this signal low. You will need to drive this signal high when the memory in the FPGA is ready to accept memory mapped transactions. The `f2h_boot_from_fpga_ready` signal is used by the Boot ROM when accessing the public key stored in the FPGA. The Boot ROM will only use the signal if asserted to boot with the public key. For more information, refer to *Appendix A: Booting and Configuration*.	`f2h_boot_from_fpga_ready`
	This is an active high signal which the Boot ROM polls when all the preloaders fail to load. This signal is driven low when the FPGA is not configured. You will need to drive the signal high in your design. The Boot ROM will continuously poll both **f2h_boot_from_fpga** signals until both are set. For more information, refer to *Appendix A: Booting and Configuration*.	`f2h_boot_from_fpga_on_failure`

Related Information

- CoreSight Debug and Trace on page 322
  Enabling the TPIU exposes trace signals to the device pins. Refer to the *CoreSight Debug and Trace* for more information.
- Booting and Configuration on page 771
  For detailed information about the HPS boot sequence, refer to the *Booting and Configuration*.

28.2.2. FPGA-to-HPS SDRAM Interface

In the FPGA-to-HPS SDRAM interface table, use the F2SDRAM port configuration pull-down menu to choose bus widths for each SDRAM. You can add one to three SDRAM ports that make the HPS SDRAM subsystem accessible to the FPGA fabric.

The following table shows the width options for the SDRAM ports in Platform Designer (Standard).

Table 231. FPGA-to-HPS SDRAM Port and Interface Names

Port configuration	F2SDRAM 0	F2SDRAM 1	F2SDRAM 2
1	AXI-32	AXI-32	AXI-32
2	AXI-64	AXI-64	AXI-64
3	AXI-128	unused	AXI-128
4	AXI-128	AXI-32	AXI-64

Table 232. FPGA-to-HPS SDRAM Port and Interface Names

Port Name	Interface Name
F2SDRAM 0	f2sdram0_data f2sdram0_clock
F2SDRAM 1	f2sdram1_data f2sdram1_clock
F2SDRAM 2	f2sdram2_data f2sdram2_clock

Note: You can configure the slave interface to a data width of 32-, 64-, 128-bits. To facilitate accessing this slave from a memory-mapped master with a smaller address width, you can use the *Intel Address Span Extender*.

Warning: To avoid memory aliasing issues, ensure that Avalon-MM burst transactions into the HPS do not cross the 4 KB address boundary restriction specified by the AXI protocol.

Related Information

Using the Address Span Extender Component on page 728
For information about Address sfo1410144425160 extender

28.2.3. DMA Peripheral Request

You can enable each direct memory access (DMA) controller peripheral request ID individually. Each request ID enables an interface for FPGA soft logic to request one of eight logical DMA channels to the FPGA.

Related Information

DMA Controller on page 512
For more information, refer to the *DMA Controller* chapter in the *Intel Arria 10 Hard Processor System Technical Reference Manual* .

28.2.4. Security Manager

Table 233. Anti Tamper signals

Interface Name	Description	Associated Signal
h2f_security	The security manager anti-tamper interface provides communication between the anti-tamper logic in the FPGA and the HPS security manager.	f2h_security_anti_tamper_in f2h_security_anti_tamper_out

Related Information

SoC Security on page 168

28.2.5. Interrupts

Table 234. FPGA-to-HPS Interrupts Interface

Parameter Name	Parameter Description	Interface Name
Enable FPGA-to-HPS Interrupts	Enables interface for FPGA interrupt signals to the MPU (in the HPS).	f2h_irq0 f2h_irq1

You can enable the interfaces for each HPS peripheral interrupt to the FPGA.

Table 235. HPS-to-FPGA Interrupts Interface

The following table lists the available HPS to FPGA interrupt interfaces and the corresponding parameters to enable them.

Parameter Name	Parameter Description	Interface Name
Enable Clock Peripheral Interrupts	Enables interface for HPS clock manager and MPU wake-up interrupt signals to the FPGA	`h2f_clkmgr_interrupt` `h2f_mpuwakeup_interrupt`
Enable CTI Interrupts	Enables interface for HPS cross-trigger interrupt signals to the FPGA	`h2f_ncti_interrupt0` `h2f_ncti_interrupt1`

Enable DMA Interrupts	Enables interface for HPS DMA channels interrupt and DMA abort interrupt to the FPGA	`h2f_dma_interrupt0`
		`h2f_dma_interrupt1`
		`h2f_dma_interrupt2`
		`h2f_dma_interrupt3`
		`h2f_dma_interrupt4`
		`h2f_dma_interrupt5`
		`h2f_dma_interrupt6`
		`h2f_dma_interrupt7`
		`h2f_dma_abort_interrupt`

Enable EMAC Interrupts	Enables interface for HPS Ethernet MAC controller interrupt to the FPGA. EMAC must be enabled in Pin Mux Tab before enabling interrupt.	`h2f_emac0_interrupt`
		`h2f_emac1_interrupt`
		`h2f_emac2_interrupt`

Enable FPGA manager Interrupts	Enables interface for the HPS FPGA manager interrupt to the FPGA	`h2f_fpga_man_interrupt`

Enable GPIO Interrupts	Enables interface for the HPS general purpose IO (GPIO) interrupt to the FPGA	`h2f_gpio0_interrupt`
		`h2f_gpio1_interrupt`
		`h2f_gpio2_interrupt`

Enable HMC Interrupts	Enable the HPS peripheral interrupt for HMC to be driven into the FPGA fabric.	`h2f_hmc_interrupt`

Enable I2C-EMAC Interrupts	Enable the HPS peripheral interrupt for I2CEMAC to be driven into the FPGA fabric	`h2f_i2c_emac0_interrupt`
		`h2f_i2c_emac1_interrupt`
		`h2f_i2c_emac2_interrupt`

Enable I2C Peripherals Interrupts	Enable the HPS peripheral interrupt for I2C0 to be driven into the FPGA fabric. The I2C must be enabled in the Pin Mux Tab before enabling interrupt.	`h2f_i2c0_interrupt` `h2f_i2c1_interrupt`
Enable L4 Timer Interrupts	Enables the HPS peripheral interrupt for L4TIMER to be driven into the FPGA fabric.	`h2f_timer_l4sp_0_interrupt` `h2f_timer_l4sp_1_interrupt`
Enable NAND Interrupts	Enables interface for the HPS NAND controller interrupt to the FPGA. The NAND IP Block must be enabled in Pin Mux Tab before enabling interrupt.	`h2f_nand_interrupt`
Enable Quad SPI Interrupts	Enables interface for the HPS QSPI controller interrupt to the FPGA. The Quad SPI IP Block must be enabled in Pin Mux Tab before enabling interrupt.	`h2f_qspi_interrupt`

continued...

Parameter Name	Parameter Description	Interface Name
Enable SYS Timer Interrupts	Enables the HPS peripheral interrupt for SYSTIMER to be driven into the FPGA fabric.	h2f_timer_sys_0_interrupt h2f_timer_sys_1_interrupt
Enable SD/MMC Interrupts	Enables interface for the HPS SD/MMC controller interrupt to the FPGA. The SD/MMC IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_sdmmc_interrupt
Enable SPI Master Interrupts	Enables interface for the HPS SPI master controller interrupt to the FPGA. The SPI Master IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_spim0_interrupt h2f_spim1_interrupt
Enable SPI Slave Interrupts	Enables interface for the HPS SPI slave controller interrupt to the FPGA. The SPI IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_spis0_interrupt h2f_spis1_interrupt
Enable ECC/Parity_L1 Interrupts	Enables the HPS peripheral interrupt for ECC single and double bit error and L1 parity error to be driven into the FPGA fabric.	h2f_ecc_serr_interrupt h2f_ecc_derr_interrupt h2f_parity_l1_interrupt
Enable UART Interrupts	Enables interface for the HPS UART controller interrupt to the FPGA. The UART IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_uart0_interrupt h2f_uart1_interrupt
Enable USB Interrupts	Enables interface for the HPS USB controller interrupt to the FPGA. The USB IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_usb0_interrupt s2f_usb1_interrupt
Enable Watchdog Interrupts	Enables interface for the HPS watchdog interrupt to the FPGA	h2f_wdog0_interrupt h2f_wdog1_interrupt

28.2.6. AXI Bridges

Table 236. Bridge Parameters

Parameter Name	Parameter Description	Interface Name
FPGA-to-HPS interface width	Enable or disable the FPGA-to-HPS interface; if enabled, set the data width to 32, 64, or 128 bits.	f2h_axi_slave f2h_axi_clock
HPS-to-FPGA interface width	Enable or disable the HPS-to-FPGA interface; if enabled, set the data width to 32, 64, or 128 bits.	h2f_axi_master h2f_axi_clock
Lightweight HPS-to-FPGA interface width	Enable or disable the lightweight HPS-to-FPGA interface. When enabled, the data width is 32 bits.	h2f_lw_axi_master h2f_lw_axi_clock

Note: To facilitate accessing these slaves from a memory-mapped master with a smaller address width, you can use the Intel Platform Designer (Standard) Address Span Extender.

Related Information

Using the Address Span Extender Component on page 728
For information about Address sfo1410144425160 extender

28.3. Configuring HPS Clocks and Resets

The **HPS Clocks and Reset** tab is one of four tabs on the HPS Component. This tab contains several groups with the following parameters:

Alternate Clock Source from FPGA on page 723
User Clocks on page 723
Reset Interfaces on page 724
Peripheral FPGA Clocks on page 725

28.3.1. Alternate Clock Source from FPGA

Table 237. Alternate Clock Source Parameters

Parameter Name	Parameter Description	Clock Interface Name
Enable Alternate Clock Source from FPGA	Drives the HPS Clock source to originate from the FPGA fabric instead of the dedicated HPS input clock pin.	`f2h_free_clk`

28.3.2. User Clocks

When you enable a HPS-to-FPGA user clock, you must manually enter its maximum frequency for timing analysis. The Timing Analyzer has no other information about how software running on the HPS configures the phase-locked loop (PLL) outputs. Each possible clock, including clocks that are available from peripherals, has its own parameter for describing the clock frequency.

Note: Use the `set_global_assignment -name ENABLE_HPS_INTERNAL_TIMING ON` to enable timing analysis.

28.3.2.1. User Clock Parameters

The frequencies that you provide are the maximum expected frequencies. The actual clock frequencies can be modified through the register interface, for example by software running on the microprocessor unit (MPU). For further details, refer to *Selecting PLL Output Frequency and Phase*.

Parameter Name	Parameter Description	Clock Interface Name
Enable HPS-to-FPGA user 0 clock	Enable main PLL from HPS-to-FPGA	`h2f_user0_clock`
User 0 clock frequency	Specify the maximum expected frequency for the main PLL	
Enable HPS-to-FPGA user 1 clock	Enable peripheral PLL from HPS-to-FPGA	`h2f_user1_clock`
User 1 clock frequency	Specify the maximum expected frequency for the peripheral PLL	

28.3.2.2. Clock Frequency Usage

The clock frequencies you provide are reported in a Synopsys Design Constraints File (**.sdc**) generated by Platform Designer (Standard). The **.sdc** file is referenced in the system **.qip** file when the system is generated.

Related Information

Clock Manager on page 52
For general information about clock signals, refer to the *Clock Manager* chapter in the *Intel Arria 10 Hard Processor System Technical Reference Manual* .

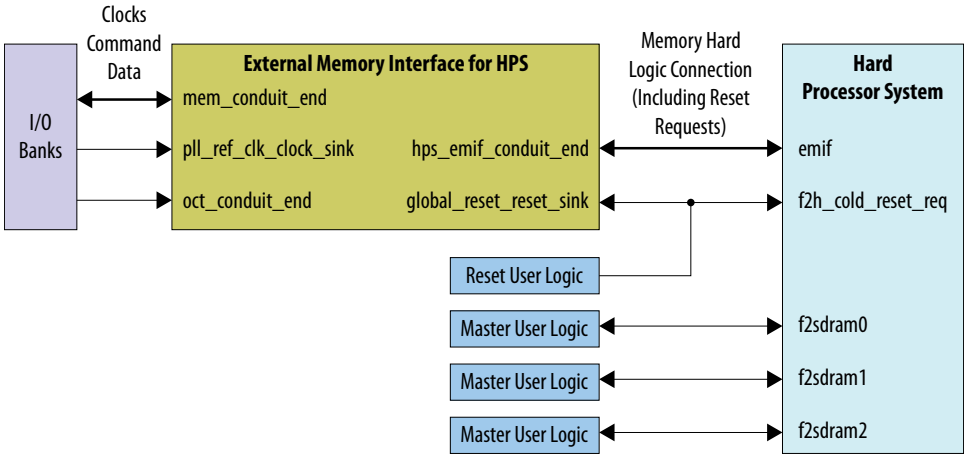
28.3.3. Reset Interfaces

You can enable the resets on an individual basis through the **HPS Clocks and resets** tab under the **Resets** sub tab.

Table 238. Reset Parameters

Parameter Name	Parameter Description	Interface Name
Enable HPS-to-FPGA cold reset output	Enable interface for HPS-to-FPGA cold reset output	`h2f_cold_reset`
Enable HPS warm reset handshake signals	Enable an additional pair of reset handshake signals allowing soft logic to notify the HPS when it is safe to initiate a warm reset in the FPGA fabric.	`h2f_warm_reset_handshake`
Enable FPGA-to-HPS debug reset request	Enable interface for FPGA-to-HPS debug reset request	`f2h_debug_reset_req`
Enable FPGA-to-HPS warm reset request	Enable interface for FPGA-to-HPS warm reset request	`f2h_warm_reset_req`
Enable FPGA-to-HPS cold reset request	Enable interface for FPGA-to-HPS cold reset request	`f2h_cold_reset_req`

Figure 160. Recommended SDRAM Reset Connections



The SDRAM reset relies on the `f2h_cold_reset_req`to be enabled. The reset can be enabled through the Platform Designer (Standard) **Resets** tab.

Note: SDRAM and the EMIF conduit can be enabled through the Platform Designer (Standard) **FPGA Interface** and **SDRAM** tab.

Related Information

Reset Manager on page 134
For more information about the reset interfaces, refer to *Functional Description of the Reset Manager* in the *Reset Manager* chapter in the *Intel Arria 10 Hard Processor System Technical Reference Manual* .

28.3.4. Peripheral FPGA Clocks

Parameter Name	Parameter Description
EMAC 0 (emac0_md_clk clock frequency)	If EMAC 0 peripheral is routed to FPGA, use the input field to specify EMAC 0 MDIO clock frequency
EMAC 0 (emac0_gtx_clk clock frequency)	If EMAC 0 peripheral is routed to FPGA, use the input field to specify EMAC 0 transmit clock frequency
EMAC 1 (emac1_md_clk clock frequency)	If EMAC 1 peripheral is routed to FPGA, use the input field to specify EMAC 1 MDIO clock frequency
EMAC 1 (emac1_gtx_clk clock frequency)	If EMAC 1 peripheral is routed to FPGA, use the input field to specify EMAC 1 transmit clock frequency
EMAC 2 (emac2_md_clk clock frequency)	If EMAC 2 peripheral is routed to FPGA, use the input field to specify EMAC 2 MDIO clock frequency
EMAC 2 (emac2_gtx_clk clock frequency)	If EMAC 2 peripheral is routed to FPGA, use the input field to specify EMAC 2 transmit clock frequency
QSPI (qspi_sclk_out clock frequency)	If QSPI peripheral is routed to FPGA, use the input field to specify QSPI serial clock frequency
SDMMC (sdmmc_cclk)	If this peripheral pin multiplexing is configured to route to FPGA fabric, use the input field to specify the SDMMC sdmmc_cclk clock frequency
SPIM 0 (spim0_sclk_out clock frequency)	If SPI master 0 peripheral is routed to FPGA, use the input field to specify SPI master 0 output clock frequency
SPIM 1 (spim1_sclk_out clock frequency)	If SPI master 1 peripheral is routed to FPGA, use the input field to specify SPI master 1 output clock frequency
I^2C0 (i2c0_clk clock frequency)	If I^2C 0 peripheral is routed to FPGA, use the input field to specify I^2C 0 output clock frequency
I^2C1 (i2c1_clk clock frequency)	If I^2C 1 peripheral is routed to FPGA, use the input field to specify I^2C 1 output clock frequency
I2CEMAC0 (i2cemac0_clk)	If this peripheral pin multiplexing is configured to route to the FPGA fabric, use the input field to specify the I2CEMAC0 i2cemac0_clk clock frequency
I2CEMAC1 (i2cemac1_clk)	If this peripheral pin multiplexing is configured to route to the FPGA fabric, use the input field to specify the I2CEMAC1 i2cemac1_clk clock frequency
I2CEMAC2 (i2cemac2_clk)	If this peripheral pin multiplexing is configured to route to the FPGA fabric, use the input field to specify the I2CEMAC2 i2cemac2_clk clock frequency

28.4. Configuring Peripheral Pin Multiplexing

The **Pin Mux and Peripherals** tab is one of four tabs on the HPS Component. This tab contains several groups with the following parameters:

Configuring Peripherals on page 726

Connecting Unassigned Pins to GPIO on page 727
Peripheral Signals Routed to FPGA on page 727

28.4.1. Configuring Peripherals

The **Pin MUX and Peripherals** tab contains three sub tabs **IP Selection**, **Advanced Pin Placement**, and **Advanced FPGA Placement**. The **IP Selection** tab contains a list of peripherals that can be enabled and either routed to the HPS I/Os or to the FPGA. You can enable one or more instances of each peripheral type by using the drop down menu next to each peripheral. When enabled, some peripherals also have mode settings specific to their functions.

Note: Once you have selected a peripheral, you must click the **Apply Selections** button in order to enable the selected peripherals.

Under the the **IP Selection** tab, Platform Designer (Standard) allows you to boot from the following flash device sources if selected.

- NAND
- SD/MMC
- QSPI

In the **Advanced Pin Placement** tab you can be more specific about the placement of each peripheral in the HPS dedicated I/O and HPS shared I/O quadrant space. Each location has a pull down IP selection menu where you can select a peripheral for the location.

Pin placement for peripherals cannot span both dedicated I/O and HPS shared I/O, except for the following situations:

- QSPI signals `qspi_ss_o[2]` and `qspi_ss_o[3]` are available to the HPS shared I/O and can be routed to any quadrant selected.
- If the SD/MMC signals are routed to the HPS shared I/O, and the `SDMMC_PWR_ENA` pin is required, then it is only available to the HPS dedicated I/O.
- If the signals for all three EMACs are routed to the HPS shared I/O, the MDIO signals for all three EMACs can be routed to the HPS dedicated I/Os.
- If the NAND Flash Controller pins are routed to the HPS dedicated I/O, `NAND_WP_N` is only available to the HPS shared I/O.

In addition, peripheral pins cannot span multiple quadrants within the HPS shared I/O. All pins of an instance must be self-contained in one quadrant, except for NANDx16 and the GPIO modules because of the number of pins required for these peripherals. For example, NANDx16 may be routed to either quadrant 1 and quadrant 2 or quadrant 3 and quadrant 4.

The **Advanced FPGA Placement** tab allows you to route peripherals to the FPGA. Aside from the **IP Selection** tab this tab allows you to focus on the FPGA routing only. There are also SDMMC and NAND Bit-width options. You can select the EMAC interface and PHY options on this tab if peripheral is selected.

Note: Changes in the **IP Selection** tab carry over to the **Advanced Pin Placement** tab, **Advanced FPGA Placement** tab, and vise versa.

You can enable the following types of peripherals:

Related Information

- CoreSight Debug and Trace on page 322
  Enabling the TPIU exposes trace signals to the device pins. Refer to the *CoreSight Debug and Trace* for more information.
- NAND Flash Controller on page 370
- SD/MMC Controller on page 405
  Secure Digital / MultiMediaCard (SD/MMC) Controller chapter in the *Intel Arria 10 Hard Processor System Technical Reference Manual* .
- Quad SPI Flash Controller on page 490
  Quad serial peripheral interface (SPI) Flash Controller chapter in the *Intel Arria 10 Hard Processor System Technical Reference Manual* .
- Ethernet Media Access Controller on page 520
  Ethernet Media Access Controller chapter in the *Intel Arria 10 Hard Processor System Technical Reference Manual* .
- USB 2.0 OTG Controller on page 591
- SPI Controller on page 608
- I2C Controller on page 643
- UART Controller on page 670

28.4.1.1. Boot Source

Under the **IP Selection** tab you have the option to choose a nonvolatile flash-based memory to be the boot source. The following flash controller peripherals under the IP block list in the **IP Selection** tab have a **Boot** enable button beside them where you can make this selection.

- NAND
- SD/MMC
- QSPI

Related Information

Booting and Configuration on page 771
For detailed information about the HPS boot sequence, refer to the *Booting and Configuration*.

28.4.2. Connecting Unassigned Pins to GPIO

For pins that are not assigned to any peripherals, you can assign them to the GPIO peripheral by clicking on the corresponding GPIO push button in the table. Click on the selected push button to remove GPIO pin assignment. The table also shows the GPIO bit number that correlates to the I/O pins.

28.4.3. Peripheral Signals Routed to FPGA

You can route the peripheral signals to the FPGA fabric and assign them to the FPGA I/O pins.

The following steps show you how to enable the peripheral signals:

1. Select a peripheral in the IP Selection list.
2. Choose the To FPGA drop-down box next to the IP you have selected and choose the amount of instances for the peripheral
3. Click the Apply Selections button at the bottom of the peripherals list.

28.5. Configuring the External Memory Interface

The **SDRAM** tab is one of four tabs on the HPS component. Enable the conduit to connect to the Arria 10 External Memory Interface. This option enables a conduit interface for the HPS to talk to the Arria 10 External Memory interfaces for the HPS.

The HPS supports one memory interface implementing double data rate 3 (DDR3), double data rate 3 Low Voltage (DDR3L), and double data rate 4 (DDR4) protocols.

Related Information

- External Memory Interface Handbook Volume 2: Design Guidelines
  The Implementing and Parameterizing Memory IP chapter in the *External Memory Interface* Handbook.
- External Memory Interface Handbook Volume 3: Reference Material
  The Functional Description--Hard Memory Interface chapter in the *External Memory Interface* Handbook. "EMI-Related HPS Features in SoC Devices" describes features specific to the HPS SDRAM controller.
- External Memory Interface In Arria 10 Devices
  For more information on the A10 External Memory Interface, refer to the External Memory Interfaces in Arria 10 Devices in the Arria 10 Core Fabric and General Purpose I/O Handbook.

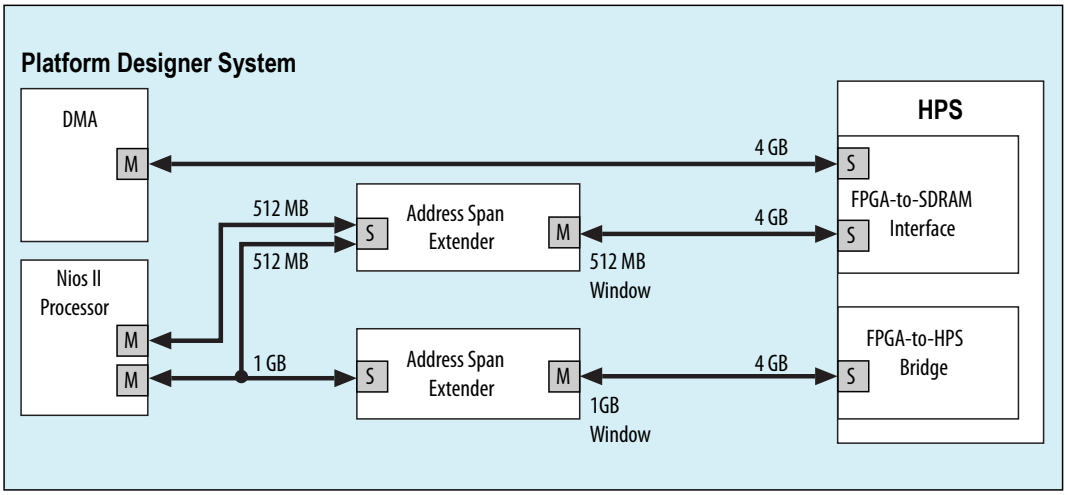
28.6. Using the Address Span Extender Component

The FPGA-to-HPS bridge and FPGA-to-HPS SDRAM memory-mapped interfaces expose their entire 4-GB address spaces to the FPGA fabric. The address span extender component provides a memory-mapped window into the address space that it masters. Using the address span extender, you can expose portions of the HPS memory space without needing to expose the entire 4 GB address space.

You can use the address span extender between a soft logic master and an FPGA-to-HPS bridge or FPGA-to-HPS SDRAM interface. This component reduces the number of address bits required for a master to address a memory-mapped slave interface located in the HPS.

Figure 161. Address Span Extender Components

Two address span extender components used in a system with the HPS.



You can also use the address span extender in the HPS-to-FPGA direction, for slave interfaces in the FPGA. In this case, the HPS-to-FPGA bridge exposes a limited, variable address space in the FPGA, which can be paged in using the address span extender.

For example, suppose that the HPS-to-FPGA bridge has a 1-GB span, and the HPS needs to access three independent 1-GB memories in the FPGA portion of the device. To achieve this, the HPS programs the address span extender to access one SDRAM (1-GB) in the FPGA at a time. This technique is commonly called paging or windowing.

Related Information

- Intel Quartus Prime Standard Edition User Guide: Platform Designer
  For more information about the Address Span Extender, refer to the Address Span Extender section.
- Platform Designer (Standard) Interconnect and System Design Components
  For more information about the address span extender, refer to *Bridges* in the *Platform Designer (Standard)* Interconnect and System Design Components chapter in the *Quartus® Prime* Handbook.

28.7. Generating and Compiling the HPS Component

The process of generating and compiling an HPS design is very similar to the process for any other Platform Designer (Standard) project. Perform the following steps:

1. Generate the design with Platform Designer (Standard). The generated files include an **.sdc** file containing clock timing constraints. If simulation is enabled, simulation files are also generated.
2. Add *<system_name>.qip* to the Quartus Prime project. *<system_name>.qip* is the Quartus Prime IP File for the HPS component, generated by Platform Designer (Standard).

Note: Platform Designer (Standard) generates pin assignments in the **qip** file.

3. Perform analysis and synthesis with the Quartus Prime software.
4. Compile the design with the Quartus Prime software.
5. Optionally back-annotate the SDRAM pin assignments, to eliminate pin assignment warnings the next time you compile the design.

Related Information

- Configuring Peripheral Pin Multiplexing on page 725
- Intel® Quartus® Prime Standard Edition Handbook Volume 1: Design and Synthesis
  For general information about using Platform Designer (Standard), refer to the *Creating a System with Platform Designer (Standard)* chapter in the *Quartus® Prime* Handbook.
- External Memory Interface Handbook Volume 2: Design Guidelines
  The Implementing and Parameterizing Memory IP chapter in the *External Memory Interface* Handbook.
- Setting Up the HPS Component for Simulation on page 749
  For a description of the simulation files generated, refer to "Simulation Flow" in the Simulating the HPS Component chapter of the *Intel Arria 10 Hard Processor System Technical Reference Manual* .

29. HPS Component Interfaces

This chapter describes the interfaces, including clocks and resets, implemented by the hard processor system (HPS) component.

The majority of the resets can be enabled on an individual basis. The exception is the `h2f_reset` interface, which is always enabled.

You must declare the clock frequency of each HPS-to-FPGA clock for timing purposes. Each possible clock, including ones that are available from peripherals, has its own parameter for describing the clock frequency. Declaring the clock frequency for HPS-to-FPGA clocks specifies how you plan to configure the PLLs and peripherals, to enable the Timing Analyzer to accurately estimate system timing. It has no effect on PLL settings.

Related Information

- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter
- Avalon Interface Specifications
  For Avalon protocol timing, refer to *Avalon Interface Specifications* .
- HPS Component Interfaces on page 731
  For information about instantiating the HPS component, refer to the *Instantiating the HPS Component* chapter.
- ARM Infocenter
  For Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) protocol timing, refer to the AMBA AXI Protocol Specification v1.0, which you can download from the ARM info center website.

29.1. Memory-Mapped Interfaces

29.1.1. FPGA-to-HPS Bridge

Interface Name	Description	Associated Clock Interface
`f2h_axi_slave`	FPGA-to-HPS AXI slave interface	`f2h_axi_clock`

The FPGA-to-HPS interface is a configurable data width AXI slave allowing FPGA masters to issue transactions to the HPS. This interface allows the FPGA fabric to access the majority of the HPS slaves. This interface also provides a coherent memory interface.

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Other interface standards in the FPGA fabric, such as connecting to Avalon® Memory-Mapped (Avalon-MM) interfaces, can be supported through the use of soft logic adapters. The Platform Designer (Standard) system integration tool automatically generates adapter logic to connect AXI to Avalon-MM interfaces.

The FPGA-to-HPS interface is an AXI-3 compliant interface with the following features:

- Configurable data width: 32, 64, or 128 bits
- Ready latency support data widths: 32, 64, or 128 bits

Related Information

- Clocks on page 734
- Features of the HPS-FPGA Bridges on page 248
  For more information, refer to the *HPS FPGA AXI Bridges Component* chapter.
- HPS Component Interfaces on page 731
  For information about the address span extender, refer to "Using the Address Span Extender Component" in the *Instantiating the HPS Component* chapter.

29.1.2. HPS-to-FPGA and Lightweight HPS-to-FPGA Bridges

Table 239. HPS-to-FPGA and Lightweight HPS-to-FPGA Bridges and Clocks

Interface Name	Description	Associated Clock Interface
`h2f_axi_master`	HPS-to-FPGA AXI master interface	`h2f_axi_clock`
`h2f_lw_axi_master`	HPS-to-FPGA lightweight AXI master interface	`h2f_lw_axi_clock`

The HPS-to-FPGA interface is a configurable data width AXI master (32-, 64-, or 128-bit) that allows HPS masters to issue transactions to the FPGA fabric.

The lightweight HPS-to-FPGA interface is a 32-bit AXI master that allows HPS masters to issue transactions to the FPGA fabric.

Other interface standards in the FPGA fabric, such as connecting to Avalon-MM interfaces, can be supported through the use of soft logic adapters. The Platform Designer (Standard) system integration tool automatically generates adaptor logic to connect AXI to Avalon-MM interfaces.

Each AXI bridge accepts a clock input from the FPGA fabric and performs clock domain crossing internally. The exposed AXI interface operates on the same clock domain as the clock supplied by the FPGA fabric.

Latency Support for (32-,64, or 128-bit)

Related Information

- Clocks on page 734
- Features of the HPS-FPGA Bridges on page 248
  For more information, refer to the *HPS FPGA AXI Bridges Component* chapter.

29.1.3. FPGA-to-HPS SDRAM Interface

The FPGA-to-HPS SDRAM interface is a direct connection between the FPGA fabric and the HPS SDRAM controller. The interface is highly configurable, allowing a mix between the four port configurations and the port width. The FPGA-to-HPS SDRAM interface supports AMBA AXI interface standards.

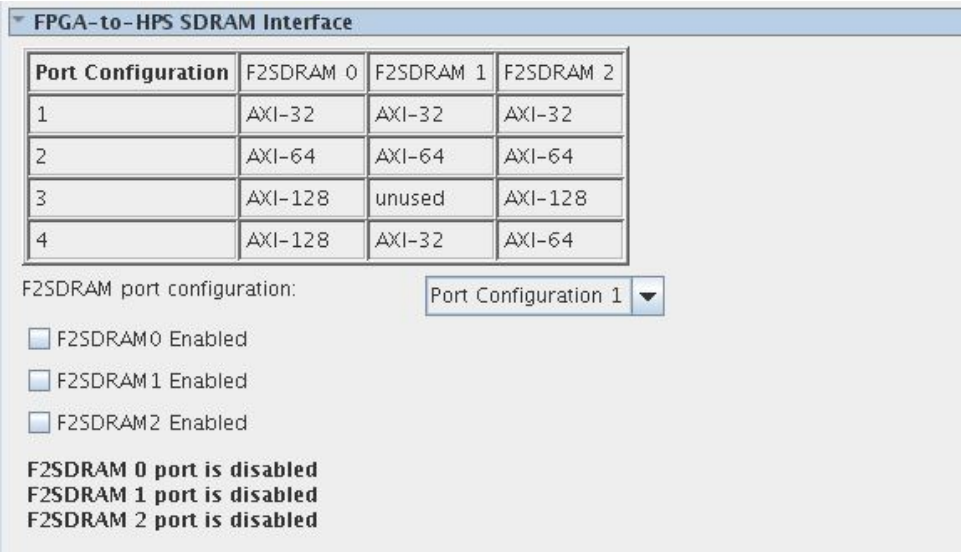
Table 240. FPGA-to-HPS SDRAM Interfaces and Clocks

Interface Name	Associated Data Interface	Associated Clock Interface
F2SDRAM0	f2sdram0_data	f2sdram0_clock
F2SDRAM1	f2sdram1_data	f2sdram1_clock
F2SDRAM2	f2sdram2_data	f2sdram2_clcok

The FPGA-to-HPS SDRAM interface is a configurable interface to the multi-port SDRAM controller. There are four port configurations that are a combination of the SDRAM ports 0 - 2 that you are able to select. Once a port configuration is selected, you can choose to enable the ports that you need.

The total data width of all interfaces is limited to a maximum of 256 bits in the read direction and 256 bits in the write direction. The HPS SDRAM controller supports up to 3 masters (command ports), 3x 64-bit read data ports and 3x 64-bit write data ports.

Figure 162. FPGA-to-HPS SDRAM Platform Designer (Standard) Interface



Related Information

- Clocks on page 734
- HPS Component Interfaces on page 731
  For information about the address span extender, refer to "Using the Address Span Extender Component" in the *Instantiating the HPS Component* chapter.

29.1.4. EMIF Conduit

Under the SDRAM tab in Platform Designer (Standard), the EMIF conduit can be enabled. This enables the HPS dedicated conduit to the Arria 10 External Interface for the HPS. This conduit cannot connect to any other External Memory Interface (EMIF). Only IP generated by the Arria 10 External Memory Interface for HPS Platform Designer (Standard) library should be used.

29.2. Clocks

The HPS-to-FPGA clock interface supplies physical clocks to the FPGA. These clocks and resets are generated in the HPS.

29.2.1. Alternative Clock Inputs to HPS PLLs

This section lists alternative clock inputs to HPS PLLs.

- `f2h_free_clk`—FPGA-to-HPS free clock. Drives the HPS clock source to originate from the FPGA fabric instead of the dedicated HPS input clock pin.

29.2.2. User Clocks

A user clock is a PLL output that is connected to the FPGA fabric rather than the HPS. You can connect a user clock to logic that you instantiate in the FPGA fabric.

- `h2f_user0_clock`—HPS-to_FPGA user clock, driven from main PLL
- `h2f_user1_clock`—HPS-to-FPGA user clock, driven from peripheral PLL

29.2.3. AXI Bridge FPGA Interface Clocks

The AXI interface has an asynchronous clock crossing in the FPGA-to-HPS bridge. The FPGA-to-HPS and HPS-to-FPGA interfaces are synchronized to clocks generated in the FPGA fabric. These interfaces can be asynchronous to one another.

- `f2h_axi_clock`—AXI slave clock for FPGA-to-HPS bridge, generated in FPGA fabric
- `h2f_axi_clock`—AXI master clock for HPS-to-FPGA bridge, generated in FPGA fabric
- `h2f_lw_axi_clock`—AXI master clock for lightweight HPS-to-FPGA bridge, generated in FPGA fabric

29.2.4. SDRAM Clocks

You can configure the HPS component with up to three FPGA-to-HPS SDRAM clocks.

Each command channel to the SDRAM controller has an individual clock source from the FPGA fabric. The interface clock is always supplied by the FPGA fabric, with clock crossing occurring on the HPS side of the boundary.

The FPGA-to-HPS SDRAM clocks are driven by soft logic in the FPGA fabric.

- `f2h_sdram0_clock`—SDRAM clock for port 0
- `f2h_sdram1_clock`—SDRAM clock for port 1
- `f2h_sdram2_clock`—SDRAM clock for port 2

29.2.5. Peripheral FPGA Clocks

Table 241. Peripheral FPGA Clocks

Clock Name	Description
`emac_md_clk`	Ethernet PHY management interface clock
`emac_gtx_clk`	Ethernet transmit clock that is used by the PHY in GMII mode
`emac_rx_clk_in`	Ethernet MAC reference clock from the PHY
`emac_tx_clk_in`	Ethernet MAC uses this clock input for TX reference
`emac_ptp_ref_clock`	Ethernet timestamp precision time protocol (PTP) reference clock
`qspi_sclk_out`	QSPI master clock output
`qspi_s2f_clk`	QSPI serial clock output (connects to the FPGA clock network)
`sdmmc_clk_in`	Clock for SD/MMC controller
`sdmmc_cclk`	Generated output clock for card
`spim_sclk_out`	SPI master serial clock output
`spis_sclk_in`	SPI slave serial clock input
`i2c_clk`	I^2C outgoing clock (part of the SCL bidirectional pin signals)
`i2c_scl_in`	I^2C incoming clock (part of the SCL bidirectional pin signals)
`i2cemac_clk`	I^2C EMAC outgoing clock

Note: The Peripheral FPGA Clocks can only be accessed once the corresponding IP Block has been selected and routed to the FPGA through the Pin Mux and Peripherals tab.

29.3. Resets

This section describes the reset interfaces to the HPS component.

Related Information

Reset Manager on page 134
For details about the HPS reset sequences, refer to the *Functional Description of the Reset Manager* in the *Reset Manager* chapter .

29.3.1. HPS-to-FPGA Reset Interfaces

The following interfaces allow the HPS to reset soft logic in the FPGA fabric:

- `h2f_reset`—HPS-to-FPGA warm reset
- `h2f_cold_reset`—HPS-to-FPGA cold reset
- `h2f_warm_reset_handshake`—Warm reset request and acknowledge interface between HPS and FPGA

29.3.2. HPS External Reset Request

The following interfaces allow soft logic in the FPGA fabric to request for different types of HPS resets:

- `f2h_cold_reset_req`—FPGA-to-HPS cold reset request
- `f2h_warm_reset_req`—FPGA-to-HPS warm reset request
- `f2h_dbg_reset_req`—FPGA-to-HPS debug reset request

29.3.3. Peripheral Reset Interfaces

The following are Ethernet reset interfaces, that can be used when the Ethernet is routed to the FPGA:

- `emac_tx_reset`— Ethernet transmit clock reset output used to reset external PHY TX clock domain logic
- `emac_rx_reset`— Ethernet receive clock reset output used to reset external PHY RX clock domain logic

29.4. Debug and Trace Interfaces

29.4.1. FPGA System Trace Macrocell Events Interface

The system trace macrocell (STM) hardware events allow logic in the FPGA to insert messages into the trace stream.

- `f2h_stm_hw_events`

29.4.2. FPGA Cross Trigger Interface

The cross trigger interface (CTI) allows trigger sources and sinks to interface with the embedded cross trigger (ECT).

- `h2f_cti`

29.4.3. Debug APB Interface

The debug Advanced Peripheral Bus (APB) interface allows debug components in the FPGA fabric to debug components on the CoreSight debug APB.

- h2f_debug_apb
- h2f_debug_apb_sideband
- h2f_debug_apb_reset
- h2f_debug_apb_clock

29.5. Peripheral Signal Interfaces

29.5.1. Platform Designer (Standard) Peripheral Port Interface Mapping

For many of the HPS component peripherals, the Platform Designer (Standard) ports can be routed to the HPS pins, FPGA pins or both. The following tables show the available mappings.

29.5.1.1. NAND Flash Controller Interface

Table 242. NAND Flash Controller Interface Platform Designer (Standard) Port Mappings

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
nand_adq_i[15:0]	Yes	Yes	NAND_ADQ[15:0]
nand_adq_oe	Yes	Yes	
nand_adq_o[15:0]	Yes	Yes	
nand_ale_o	Yes	Yes	NAND_ALE
nand_ce_o[3:0]	Yes, 4 chip enables	Yes, 1 chip enable	NAND_CE_N
nand_cle_o	Yes	Yes	NAND_CLE
nand_re_o	Yes	Yes	NAND_RE_N
nand_rdy_busy_i[3:0]	Yes, 4 ready/busy signals	Yes, 1 ready/busy signal	NAND_RB
nand_we_o	Yes	Yes	NAND_WE_N
nand_wp_o	Yes	Yes	NAND_WP_N

Related Information

NAND Flash Controller on page 370
For more information about the NAND Flash Controller features and interface.

29.5.1.2. SD/MMC Controller Interface

Table 243. SD/MMC Controller Interface Platform Designer (Standard) Port Mappings

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
sdmmc_vs_o	Yes	No	-
sdmmc_pwr_ena_o	Yes	Yes	SDMMC_PWR_ENA
sdmmc_wp_i	Yes	No	-
sdmmc_cdn_i	Yes	No	-
sdmmc_rstn_o	Yes	No	-
sdmmc_card_intn_i	Yes	No	-
sdmmc_clk_in_i	Yes	No	-
sdmmc_cclk_out	Yes	Yes	SDMMC_CCLK
sdmmc_cmd_i	Yes	Yes	SDMMC_CMD
sdmmc_cmd_o	Yes	Yes	
sdmmc_cmd_oe	Yes	Yes	
sdmmc_data_i[7:0]	Yes	Yes	SDMMC_DATA[7:0]
sdmmc_data_o[7:0]	Yes	Yes	
sdmmc_data_oe	Yes	Yes	

Related Information

Features of the SD/MMC Controller on page 405
For more information about the SD/MMC Controller features and interface.

29.5.1.3. Quad SPI Flash Controller Interface

Table 244. Quad SPI Flash Controller Platform Designer (Standard) Port Mappings

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
qspi_sclk_out	Yes	Yes	QSPI_CLK
qspi_s2f_clk	Yes	No	-
qspi_io0_i	Yes	Yes	QSPI_IO0
qspi_io0_o	Yes	Yes	
qspi_mo_oe0	Yes	Yes	
qspi_io1_i	Yes	Yes	QSPI_IO1
qspi_io1_o	Yes	Yes	
qspi_mo_oe1	Yes	Yes	
qspi_io2_i	Yes	Yes	QSPI_IO2_WPN
qspi_io2_wpn_o	Yes	Yes	
qspi_mo_oe2	Yes	Yes	

continued...

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
qspi_io3_i	Yes	Yes	QSPI_IO3_HOLD
qspi_io3_hold_o	Yes	Yes	
qspi_mo_oe3	Yes	Yes	
qspi_ss_o[3:0]	Yes	Yes	QSPI_SS[3:0]

Related Information

Quad SPI Flash Controller on page 490
For more information about the Quad SPI Flash Controller features and interface.

29.5.1.4. Ethernet Media Access Controller Interface

Table 245. Ethernet Media Access Controller Platform Designer (Standard) Port Mappings

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
emac0_phy_txd_o[7:0]	Yes, 8 bits	Yes, 4 bits	EMAC0_TXD[3:0]
emac0_phy_mac_speed_o	Yes	No	-
emac0_phy_txen_o	Yes	Yes	EMAC0_TX_CTL
emac0_phy_txer_o	Yes	No	-
emac0_phy_rxdv_i	No	Yes	EMAC0_RX_CTL
emac0_phy_rxer_i	Yes	No	-
emac0_phy_rxd_i[7:0]	Yes, 8 bits	Yes, 4 bits	EMAC0_RXD[3:0]
emac0_phy_col_i	Yes	No	-
emac0_phy_crs_i	Yes	No	-
emac0_gmii_mdc_o	Yes	Yes	EMAC0_MDC
emac0_gmii_mdo_o	Yes	Yes	EMAC0_MDIO
emac0_gmii_mdo_o_e	Yes	Yes	
emac0_gmii_mdi_i	Yes	Yes	
emac0_ptp_pps_o	Yes	No	-
emac0_ptp_aux_ts_trig_i	Yes	No	-
emac0_clk_rx_i	Yes	Yes	EMAC0_RX_CLK
emac0_clk_tx_i	Yes	No	-
emac0_phy_txclk_o	Yes	Yes	EMAC0_TX_CLK
emac0_rst_clk_tx_n_o	Yes	No	-
emac0_rst_clk_rx_n_o	Yes	No	-
emac1_phy_txd_o[7:0]	Yes, 8 bits	Yes, 4 bits	EMAC1_TXD[3:0]
emac1_phy_mac_speed_o	Yes	No	-

continued...

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
emac1_phy_txen_o	Yes	Yes	EMAC1_TX_CTL
emac1_phy_txer_o	Yes	No	-
emac1_phy_rxdv_i	Yes	Yes	EMAC1_RX_CTL
emac1_phy_rxer_i	Yes	No	-
emac1_phy_rxd_i[7:0]	Yes, 8 bits	Yes, 4 bits	EMAC1_RXD[3:0]
emac1_phy_col_i	Yes	No	-
emac1_phy_crs_i	Yes	No	-
emac1_gmii_mdc_o	Yes	Yes	EMAC1_MDC
emac1_gmii_mdo_o	Yes	Yes	EMAC1_MDIO
emac1_gmii_mdo_o_e	Yes	Yes	
emac1_gmii_mdi_i	Yes	Yes	
emac1_ptp_pps_o	Yes	No	-
emac1_ptp_aux_ts_trig_i	Yes	No	-
emac1_clk_rx_i	Yes	Yes	EMAC1_RX_CLK
emac1_clk_tx_i	Yes	No	-
emac1_phy_txclk_o	Yes	Yes	EMAC1_TX_CLK
emac1_rst_clk_tx_n_o	Yes	No	-
emac1_rst_clk_rx_n_o	Yes	No	-
emac2_phy_txd_o[7:0]	Yes, 8 bits	Yes, 4 bits	EMAC2_TXD[3:0]
emac2_phy_mac_speed_o	Yes	No	-
emac2_phy_txen_o	Yes	Yes	EMAC2_TX_CTL
emac2_phy_txer_o	Yes	No	-
emac2_phy_rxdv_i	Yes	Yes	EMAC2_RX_CTL
emac2_phy_rxer_i	Yes	-	No
emac2_phy_rxd_i[7:0]	Yes, 8 bits	Yes, 4 bits	EMAC2_RXD[3:0]
emac2_phy_col_i	Yes	No	-
emac2_phy_crs_i	Yes	No	-
emac2_gmii_mdc_o	Yes	Yes	EMAC2_MDC
emac2_gmii_mdo_o	Yes	Yes	EMAC2_MDIO
emac2_gmii_mdo_o_e	Yes	Yes	
emac2_gmii_mdi_i	Yes	Yes	
emac2_ptp_pps_o	Yes	No	-
emac2_ptp_aux_ts_trig_i	Yes	No	_
emac2_clk_rx_i	Yes	Yes	EMAC2_RX_CLK

continued...

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
emac2_clk_tx_i	Yes	No	-
emac2_phy_txclk_o	Yes	Yes	EMAC2_TX_CLK
emac2_rst_clk_tx_n_o	Yes	No	-
emac2_rst_clk_rx_n_o	Yes	No	-

Related Information

Ethernet Media Access Controller on page 520
For more information about the Ethernet Media Access Controller features and interface.

29.5.1.5. USB 2.0 OTG Controller Interface

Table 246. USB 2.0 OTG Controller Platform Designer (Standard) Port Mappings

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
usb0_ulpi_clk	No	Yes	USB0_CLK
usb0_ulpi_dir	No	Yes	USB0_DIR
usb0_ulpi_nxt	No	Yes	USB0_NXT
usb0_ulpi_stp	No	Yes	USB0_STP
usb0_ulpi_data_i[7:0]	No	Yes	USB0_DATA[7:0]
usb0_ulpi_data_o[7:0]	No	Yes	
usb0_ulpi_data_oe[7:0]	No	Yes	
usb1_ulpi_clk	No	Yes	USB1_CLK
usb1_ulpi_dir	No	Yes	USB1_DIR
usb1_ulpi_nxt	No	Yes	USB1_NXT
usb1_ulpi_stp	No	Yes	USB1_STP
usb1_ulpi_data_i[7:0]	No	Yes	USB1_DATA[7:0]
usb1_ulpi_data_o[7:0]	No	Yes	
usb1_ulpi_data_oe[7:0]	No	Yes	

29.5.1.6. SPI Controller Interface

Table 247. SPI Controller Interface Platform Designer (Standard) Port Mappings

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
spim0_sclk_out	Yes	Yes	SPIM0_CLK
spim0_mosi_o	Yes	Yes	SPIM0_MOSI
spim0_mosi_oe	Yes	No	

continued...

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
spim0_miso_i	Yes	Yes	SPIM0_MISO
spim0_ss_in_n	Yes	No	-
spim0_mosi_oe	Yes	No	-
spim0_ss0_n_o	Yes	Yes	SPIM0_SS0_N
spim0_ss1_n_o	Yes	Yes	SPIM0_SS1_N
spim0_ss2_n_o	Yes	No	-
spim0_ss3_n_o	Yes	No	-
spim1_sclk_out	Yes	Yes	SPIM1_CLK
spim1_mosi_o	Yes	Yes	SPIM1_MOSI
spim1_mosi_oe	Yes	Yes	
spim1_miso_i	Yes	Yes	SPIM1_MISO
spim1_ss_in_n	Yes	No	-
spim1_ss0_n_o	Yes	Yes	SPIM1_SS0_N
spim1_ss1_n_o	Yes	Yes	SPIM1_SS1_N
spim1_ss2_n_o	Yes	No	-
spim1_ss3_n_o	Yes	No	-
spis0_sclk_in	Yes	Yes	SPIS0_CLK
spis0_mosi_i	Yes	Yes	SPIS0_MOSI
spis0_ss_in_n	Yes	Yes	SPIS0_SS0_N
spis0_miso_o	Yes	Yes	SPIS0_MISO
spis0_miso_oe	Yes	Yes	
spis1_sclk_in	Yes	Yes	SPIS1_CLK
spis1_mosi_i	Yes	Yes	SPIS1_MOSI
spis1_ss_in_n	Yes	Yes	SPIS1_SS0_N
spis1_miso_o	Yes	Yes	SPIS1_MISO
spis1_miso_oe	Yes	Yes	

Related Information

SPI Controller on page 608
For more information about the SPI Controller features and interface.

29.5.1.7. I2C Controller Interface

Table 248. I²C Controller Platform Designer (Standard) Port Mappings

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
i2c0_scl_i	Yes	Yes	I2C0_SCL
i2c0_scl_oe	Yes	Yes	
i2c0_sda_i	Yes	Yes	I2C0_SDA
i2c0_sda_oe	Yes	Yes	
i2c1_scl_i	Yes	Yes	I2C1_SCL
i2c1_scl_oe	Yes	Yes	
i2c1_sda_i	Yes	Yes	I2C1_SDA
i2c1_sda_oe	Yes	Yes	
i2c_emac0_scl_i	Yes	Yes	I2C_EMAC0_SCL
i2c_emac0_scl_oe	Yes	Yes	
i2c_emac0_sda_i	Yes	Yes	I2C_EMAC0_SDA
i2c_emac0_sda_oe	Yes	Yes	
i2c_emac1_scl_i	Yes	Yes	I2C_EMAC1_SCL
i2c_emac1_scl_oe	Yes	Yes	
i2c_emac1_sda_i	Yes	Yes	I2C_EMAC1_SDA
i2c_emac1_sda_oe	Yes	Yes	
i2c_emac2_scl_i	Yes	Yes	I2C_EMAC2_SCL
i2c_emac2_scl_oe	Yes	Yes	
i2c_emac2_sda_i	Yes	Yes	I2C_EMAC2_SDA
i2c_emac2_sda_oe	Yes	Yes	

Note: When routing the I²C signals to the FPGA I/O, the I²C output signal should be connected to ground. Refer to the *I²C Controller* chapter for more information.

Related Information

I2C Controller on page 643
For more information about the I²C Controller features and interface.

29.5.1.8. UART Interface

Table 249. UART Interface Platform Designer (Standard) Port Mappings

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
uart0_cts_n	Yes	Yes	UART0_CTS_N
uart0_dsr_n	Yes	No	-

continued...

Platform Designer (Standard) Port Name	Routed to FPGA	Routed to HPS I/O	HPS Pin Name
uart0_dcd_n	Yes	No	-
uart0_ri_n	Yes	No	-
uart0_rx	Yes	Yes	UART0_RX
uart0_dtr_n	Yes	No	-
uart0_rts_n	Yes	Yes	UART0_RTS_N
uart0_out1_n	Yes	No	-
uart0_out2_n	Yes	No	-
uart0_tx	Yes	Yes	UART0_TX
uart1_cts_n	Yes	Yes	UART1_CTS_N
uart1_dsr_n	Yes	No	-
uart1_dcd_n	Yes	No	-
uart1_ri_n	Yes	No	-
uart1_rx	Yes	Yes	UART1_RX
uart1_dtr_n	yes	No	-
uart1_rts_n	Yes	Yes	UART1_RTS_N
uart1_out1_n	Yes	No	-
uart1_out2_n	Yes	No	-
uart1_tx	Yes	Yes	UART1_TX

Related Information

UART Controller on page 670
For more information about the UART Controller features and interface.

29.5.2. DMA Controller Interface

The DMA controller interface allows soft IP in the FPGA fabric to communicate with the DMA controller in the HPS. You can configure up to eight separate interface channels.

- f2h_dma0—FPGA DMA controller peripheral request interface 0
- f2h_dma1—FPGA DMA controller peripheral request interface 1
- f2h_dma2—FPGA DMA controller peripheral request interface 2
- f2h_dma3—FPGA DMA controller peripheral request interface 3
- f2h_dma4—FPGA DMA controller peripheral request interface 4
- f2h_dma5—FPGA DMA controller peripheral request interface 5
- f2h_dma6—FPGA DMA controller peripheral request interface 6
- f2h_dma7—FPGA DMA controller peripheral request interface 7

Each of the DMA peripheral request interface contains the following three signals:

- `f2h_dma_req`—This signal is used to request burst transfer using the DMA
- `f2h_dma_single`—This signal is used to request single word transfer using the DMA
- `f2h_dma_ack`—This signal indicates the DMA acknowledgment upon requests from the FPGA

Related Information

DMA Controller on page 512
For details about the DMA Controller, refer to *DMA controller* chapter.

29.6. Other Interfaces

Related Information

Cortex-A9 MPCore on page 0
For more information, refer to *Cortex-A9 MPU Subsystem* .

29.6.1. MPU Standby and Event Interfaces

MPU standby signals are notification signals to the FPGA fabric that the MPU is in standby. Event signals are used to wake up the Cortex-A9 processors from a wait for event (WFE) state. The following shows the signals in the interface:

- `h2f_mpu_eventi`—Sends an event from logic in the FPGA fabric to the MPU. This FPGA-to-HPS signal is used to wake up a processor that is in a Wait For Event state. Asserting this signal has the same effect as executing the SEV instruction in the Cortex-A9. This signal must be de-asserted until the FPGA fabric is powered-up and configured.
- `h2f_mpu_evento`—Sends an event from the MPU to logic in the FPGA fabric. This HPS-to-FPGA signal is asserted when an SEV instruction is executed by one of the Cortex-A9 processors.
- `h2f_mpu_standbywfe[1:0]`—Indicates which Cortex-A9 processor is in the WFE state
- `h2f_mpu_standbywfi[1:0]`—Indicates which Cortex-A9 processor is in the wait for interrupt (WFI) state

The MPU provides signals to indicate when it is in a standby state. These signals are available to custom hardware designs in the FPGA fabric.

Related Information

Cortex-A9 MPCore on page 0
For more information, refer to *Cortex-A9 MPU Subsystem* .

29.6.2. General Purpose Signals

Table 250. General Purpose Interfaces

Signal	Description	Associated Signal
h2f_gp	General purpose interface. Enables a pair of 32-bit unidirectional general-purpose interfaces between the FPGA manager in HPS and the FPGA fabric.	h2f_gp_in h2f_gp_out

29.6.3. FPGA-to-HPS Interrupts

You can configure the HPS component to provide 64 general purpose FPGA-to-HPS interrupts, allowing soft IP in the FPGA fabric to trigger interrupts to the MPU's generic interrupt controller (GIC). The interrupts are implemented through the following 32-bit interfaces:

- f2h_irq0—FPGA-to-HPS interrupts 0 through 31
- f2h_irq1—FPGA-to-HPS interrupts 32 through 63

The FPGA-to-HPS interrupts are asynchronous on the FPGA interface. Inside the HPS, the interrupts are synchronized to the MPU's internal peripheral clock (periphclk).

Related Information

HPS Component Interfaces on page 731
There are various HPS peripherals to FPGA interrupt interfaces that you can configure. To learn the complete list of interfaces, refer to the *Instantiating the HPS Component* chapter.

29.6.4. Boot from FPGA Interface

You can enable the boot from FPGA interface to indicate the availability of preloader software in the FPGA memory. This interface is used to indicate the availability of a fallback preloader software in FPGA memory as well. The fallback preloader is used when there is no valid preloader image found in HPS flash memories.

29.6.5. Security Manager

Table 251. Anti Tamper signal

Interface Name	Description
h2f_security	HPS to FPGA input to trigger memory scrambling of all internal memories.

30. Simulating the HPS Component

This section describes the simulation support for the hard processor system (HPS) component. The HPS simulation models provide bus functional models of the HPS and FPGA fabric and a simulation model of the interface to SDRAM memory.

The HPS simulation support does not include modules implemented in the HPS, such as the Arm Cortex-A9 MPCore processor.

The simulation support files are specified when the HPS component is instantiated in the Platform Designer (Standard) system integration tool. When you enable a particular HPS-FPGA interface, Platform Designer (Standard) provides the corresponding model during the generation process.

The HPS simulation support enables you to develop and verify your own FPGA user logic or intellectual property (IP) that interfaces to the HPS component.

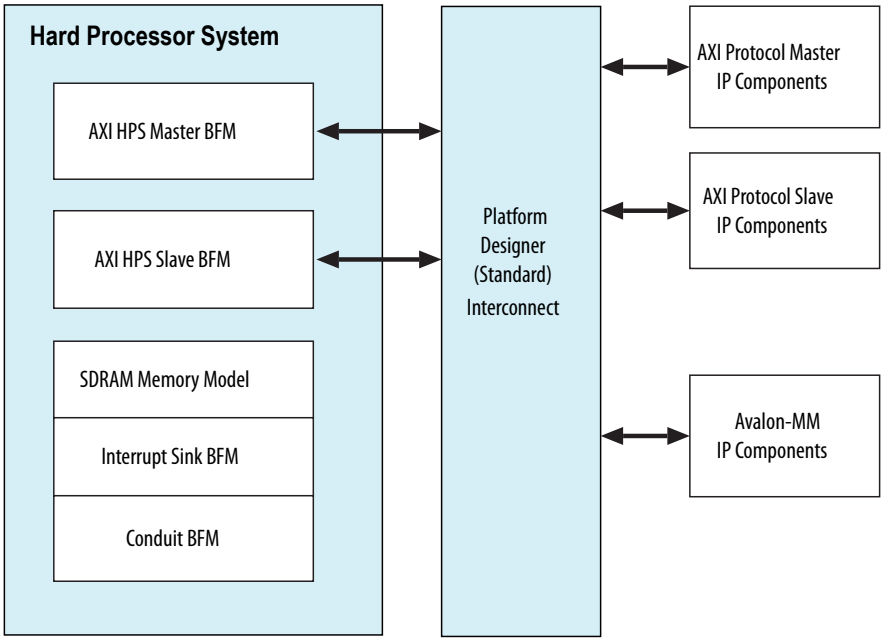
The simulation model supports the following interfaces:

- Clock and reset interfaces
- FPGA-to-HPS Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) slave interface
- HPS-to-FPGA AXI master interface
- Lightweight HPS-to-FPGA AXI master interface
- Microprocessor unit (MPU) general-purpose I/O (GPIO) interface
- MPU standby and event interface
- Interrupts interface
- Direct memory access (DMA) controller peripheral request interface
- Debug Advanced Peripheral Bus (APB) interface
- System Trace Macrocell (STM) hardware event
- FPGA cross trigger interface (CTI)
- FPGA trace port interface unit (TPIU)
- Boot from FPGA interface

Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Figure 163. HPS BFM Block Diagram



The HPS BFMs use standard function calls from the BFM application programming interface (API), as detailed in the remainder of this chapter.

HPS simulation supports only Verilog HDL or SystemVerilog simulation environments. Users with VHDL Custom IP can run BFM simulations so long as a mixed-language simulation license is available on their chosen simulator.

Related Information

- Simulation Flows on page 748
- Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
  For details on the document revision history of this chapter
- Instantiating the HPS Component on page 716
- Avalon Verification IP Suite User Guide
- Mentor Graphics* Verification IP Altera Edition AMBA AXI3 and AXI4 User Guide

30.1. Simulation Flows

Intel provides a functional register transfer level (RTL) simulation and a post–fitter gate–level simulation flow. Both simulation flows involve the following major steps, which is defined in the following sections:

1. Setting up the HPS component for simulation.
2. Generating the HPS simulation model in Platform Designer (Standard).
3. Running the simulation.

Related Information

Simulating Intel FPGA Designs
For general information about simulation, refer to *Simulating Intel FPGA Designs* in the *Intel Quartus Prime Standard Edition User Guide: Third-party Simulation*.

30.1.1. Setting Up the HPS Component for Simulation

The following steps outline how to set up the HPS component for simulation.

1. Add the HPS component from the Platform Designer (Standard) Component Library.
2. Configure the component based on your application needs by selecting or deselecting the HPS-FPGA interfaces.
3. Connect the appropriate HPS interfaces to other components in the system. For example, connect the FPGA-to-HPS AXI slave interface to an AXI or Avalon-MM master interface in another component in the system.

When you create your component, make sure the conduit interfaces have the correct role names and widths. Also make sure the conduit interfaces are opposite in direction to what is shown in the HPS Conduit Interfaces table.

Related Information

Instantiating the HPS Component on page 716

30.1.1.1. HPS Conduit Interfaces Connecting to the FPGA

The following tables define the HPS Conduit interfaces that connect to the FPGA.

Table 252. h2f_warm_reset_handshake

Role Name	Direction	Width
h2f_pending_rst_req_n	Output	1
f2h_pending_rst_ack_n	Input	1

Table 253. h2f_gp

Role Name	Direction	Width
h2f_gp_in	Input	32
h2f_gp_out	Output	32

Table 254. h2f_mpu_events

Role Name	Direction	Width
h2f_mpu_eventi	Input	1
h2f_mpu_evento	Output	1
h2f_mpu_standbywfe	Output	2
h2f_mpu_standbywfi	Output	2

Table 255. f2h_dma0 to f2h_dma7

Role Name	Direction	Width
f2h_dma_req<0-7>_req	Input	1
f2h_dma_req<0-7>_single	Input	1
f2h_dma_req<0-7>_ack	Output	1

Table 256. h2f_debug_apb_sideband

Role Name	Direction	Width
h2f_dbg_apb_PCLKEN	Input	1
h2f_dbg_apb_DBG_APB_DISABLE	Input	1

Table 257. f2h_stm_hw_events

Role Name	Direction	Width
f2h_stm_hwevents	Input	28

Table 258. h2f_cti

Role Name	Direction	Width
h2f_cti_trig_in	Input	8
h2f_cti_trig_in_ack	Output	8
h2f_cti_trig_out	Output	8
h2f_cti_trig_out_ack	Input	8
h2f_cti_fpga_clk_en	Input	1

Table 259. h2f_tpiu

Role Name	Direction	Width
h2f_tpiu_clk_ctrl	Input	1
h2f_tpiu_data	Output	32

Table 260. f2h_boot_from_fpga

Role Name	Direction	Width
f2h_boot_from_fpga_ready	Input	1
f2h_boot_from_fpga_on_failure	Input	1

30.1.1.1.1. Setting Up the HPS Component for Simulation

The following steps outline how to set up the HPS component for simulation.

1. Add the HPS component from the Platform Designer (Standard) Component Library.
2. Configure the component based on your application needs by selecting or deselecting the HPS-FPGA interfaces.
3. Connect the appropriate HPS interfaces to other components in the system. For example, connect the FPGA-to-HPS AXI slave interface to an AXI or Avalon-MM master interface in another component in the system.

When you create your component, make sure the conduit interfaces have the correct role names and widths. Also make sure the conduit interfaces are opposite in direction to what is shown in the HPS Conduit Interfaces table.

Related Information

Instantiating the HPS Component on page 716

30.1.2. Generating the HPS Simulation Model in Platform Designer (Standard)

The following steps outline how to generate the simulation model:

1. In Platform Designer (Standard), click **Generate HDL** under the Generate menu.
2. Choose between RTL and post–fit simulation

   For RTL simulation, perform the following steps:

   a. Set **Create simulation model** to Verilog.

   b. Click **Generate**.(56)

   For post–fit simulation, perform the following steps:

   a. Turn on the **Create HDL design files for synthesis** option.

   b. Turn on the **Create block symbol file (.bsf)** option.(57)(58)
3. Click **Generate**.

Related Information

- Instantiating the HPS Component on page 716
- Creating a System with Platform Designer (Standard)
  For more information about Platform Designer (Standard) simulation, refer to "Simulating a Platform Designer (Standard) System" in the *Intel Quartus Prime Standard Edition User Guide: Platform Designer (Standard)*.

30.1.3. Running the Simulations

The steps to run a simulation depend on whether you are running an RTL simulation or a post-fit simulation.

(56) VHDL is supported for HPS simulation and it requires a mix language simulator. However, the BFMs always need to be in verilog. Custom components can be in VHDL.

(57) A **.bsf** file is only needed for schematic entry.

(58) This is not a requirement for simulation or implementation unless a schematic is used.

30.1.3.1. Running HPS RTL Simulation

Platform Designer (Standard) generates scripts for several simulators that you can use to complete the simulation process, as listed in the following table.

Table 261. Platform Designer (Standard)-Generated Scripts for Supported Simulators

Simulator	Script Name	Directory
Mentor Graphics* ModelSim	**msim_setup.tcl**	*<project directory>*/*<design name>***/simulation/mentor**
Cadence NC-Sim	**ncsim_setup.sh**	*<project directory>*/*<design name>***/simulation/cadence**
Synopsys VCS	**vcs_setup.sh**	*<project directory>*/*<design name>***/simulation/synopsys/vcs**
Synopsyss VCS-MX	**vcsmx_setup.sh**	*<project directory>*/*<design name>***/simulation/synopsys/vcsmx**
Aldec* RivieraPro™	**rivierapro_setup.tcl**	*<project directory>*/*<design name>***/simulation/aldec**

Related Information

- Avalon Verification IP Suite User Guide
- Mentor Graphics Verification IP Altera Edition AMBA AXI3 and AXI4 User Guide

30.1.3.2. Running HPS Post-Fit Simulation

To run HPS post-fit simulation after successful Platform Designer (Standard) generation, perform the following steps:

1. Add the generated synthesis file set to your Intel Quartus Prime project by performing the following steps:
   a. In the Quartus Prime software, click **Settings** in the Assignments menu.
   b. In the **Settings** *<your system name>* dialog box, on the **Files** tab, browse to *<your project directory>*/*<your system name>***/synthesis/** and select *<your system name>***.qip**.
   c. Click **Open**. The **Select File** dialog box closes.
   d. Click **OK**. The **Settings** dialog box closes.
2. Optionally instantiate your HPS system as the top-level entity in your Intel Quartus Prime project.
3. Compile the design by clicking **Start Compilation** in the Processing menu.
4. Change the EDA Netlist Writer settings, if necessary, by performing the following steps:
   a. Click **Settings** in the Assignment menu.
   b. On the **Simulation** tab, under the **EDA Tool Settings** tab, you can specify the following EDA Netlist Writer settings:

- **Tool name**—The name of the simulation tool
- **Format for output netlist**
- **Output directory**

c. Click **OK**.

5. To create the post-fitter simulation model with Quartus Prime EDA Netlist Writer, perform the following steps:
   a. Click **Start** in the Processing menu.
   b. Click **Start EDA Netlist Writer**.

Related Information

Welcome to the Intel Quartus Prime Pro Edition Software Help

30.1.3.2.1. Post-Fit Simulation Files

Post-fit simulation is the simulation of the netlist generated from the original RTL design after it has been mapped, synthesized, and fit. The netlist represents the actual hardware and its connections as they appear in the FPGA. Quartus Prime generates the netlist and can generate a Standard Delay Format (`.sdf`) file with the timing information for all connections. The simulation can be functional only (without the timing information) where all wires and gates take zero time, or it can be a timing simulation where the time for all transitions is based on the SDF information.

Post-fit simulation can serve a number of different purposes. It can be used to perform a dynamic verification of the timing of the design, or it can be used to verify the functional correctness of either the design, the compilation flow (in particular, the fitter), or both.

This table uses the following symbols:

- *<ACDS install>* = Intel SoC FPGA Embedded Development Suite installation path
- *<Avalon Verification IP>* = *<ACDS install>*`/ip/altera/sopc_builder_ip/verification`
- *<AXI Verification IP>* = *<ACDS install>*`/ip/altera/mentor_vip_ae`
- *<HPS Post-fit Sim>* = *<ACDS install>*`/ip/altera/hps/postfitter_simulation`
- *<Device Sim Lib>* = *<ACDS install>*`/quartus/eda/sim_lib`

Table 262. Post-Fit Simulation Files

Library	Directory	File

Altera Verification IP Library	*<Avalon Verification IP>*`/lib/`	`verbosity_pkg.sv`
		`avalon_mm_pkg.sv`
		`avalon_utilities_pkg.sv`

Avalon Clock Source BFM	*<Avalon Verification IP>*`/altera_avalon_clock_source/`	`altera_avalon_clock_source.sv`
Avalon Reset Source BFM	*<Avalon Verification IP>*`/altera_avalon_reset_source/`	`altera_avalon_reset_source.sv`

continued...

Library	Directory	File
Avalon MM Slave BFM	*<Avalon Verification IP>*/altera_avalon_mm_slave_bfm/	altera_avalon_mm_slave_bfm.sv
Avalon Interrupt Sink BFM	*<Avalon Verification IP>*/altera_avalon_interrupt_sink/	altera_avalon_interrupt_sink.sv
Mentor AXI Verification IP Library	*<AXI Verification IP>*/common/	questa_mvc_svapi.svh
Mentor AXI3 BFM	*<AXI Verification IP>*/axi3/axi3/bfm/	mgc_common_axi.sv mgc_axi_master.sv mgc_axi_slave.sv
HPS Post-Fit Simulation Library	*<HPS Post-fit Sim>*/	All the files in the directory
Device Simulation Library[59]	*<Device Sim Lib>*/	altera_primitives.v 220model.v sgate.v altera_mf.v altera_lnsim.sv cyclonev_atoms.v arriav_atoms.v mentor/cyclonev_atoms_ncrypt.v mentor/arriav_atoms_ncrypt.v
EDA Netlist Writer Generated Post-Fit Simulation Model	*<User project directory>*/	*.vo *.vho (Mixed-language simulator is needed for Verilog HDL and VHDL mixed design)
User testbench files	*<User project directory>*/	*.v *.sv *.vhd (Mixed-language simulator is needed for Verilog HDL and VHDL mixed design)

30.1.3.2.2. BFM API Hierarchy Format

For post-fit simulation, you must call the BFM API in your test program with a specific hierarchy. The hierarchy format is:
<DUT>.*<HPS>*|fpga_interfaces|
<interface><space>.*<BFM>*.*<API function>*

[59] The device simulation library is not needed with ModelSim* - Intel FPGA Edition.

Where:

- <*DUT*> is the instance name of the design under test that you instantiated in your test bench . The design under test is the HPS component.
- <*HPS*> is the HPS component instance name that you use in your Platform Designer (Standard) system.
- <*interface*> is the instance name of a specific FPGA-to-HPS or HPS-to-FPGA interface. This name can be found in the **fpga_interfaces.sv** file located in <*project directory*>**/**<*design name*>**/synthesis/submodules**.
- <*space*>—You must insert one space character after the interface instance name.
- <*BFM*> is the BFM instance name. To identify the BFM instance name, in <*ACDS install*>**/ip/altera/hps/postfitter_simulation**, find the SystemVerilog file corresponding to the interface type that you are using. This SystemVerilog file contains the BFM instance name.

For example, a path for the Lightweight HPS-to-FPGA master interface hierarchy could be formed as follows:

```
top.dut.\my_hps_component|fpga_interface|
hps2fpga_light_weight .h2f_lw_axi_master
```

Notice the space after `hps2fpga_light_weight`. Omitting this space would cause simulation failure because the instance name `hps2fpga_light_weight` , including the space, is the name used in the post-fit simulation model generated by the Intel Quartus Prime software.

30.2. Clock and Reset Interfaces

30.2.1. Clock Interface

Platform Designer (Standard) generates the clock source BFM for the FPGA-to-HPS alternate clock source.

Table 263. HPS Clock Input Interface Simulation Model

The clock source BFM application programming interface (API) applies to the BFM listed in this table. Your Verilog interfaces use the same API.

Interface Name	BFM Instance Name
`f2h_free_clk`	`f2h_free_clock_inst`

Platform Designer (Standard) generates the clock source BFM for each clock output interface from the HPS component. For HPS-to-FPGA user clocks, specify the BFM clock rate in the **User clock frequency field** in the **HPS Clocks** page when instantiating the HPS component in Platform Designer (Standard).

The HPS-to-FPGA debug APB interface generates a clock output to the FPGA, named `h2f_debug_apb_clock`. In simulation, the clock source BFM also represents this clock output's behavior.

Table 264. HPS Clock Output Interface Simulation Model

The Intel clock source BFM application programming interface (API) applies to all the BFMs listed in this table. Your Verilog interfaces use the same API.

Interface Name	BFM Instance Name
h2f_user0_clock	h2f_user0_clock_inst
h2f_user1_clock	h2f_user1_clock_inst
h2f_user2_clock	h2f_user2_clock_inst

Related Information

Memory-Mapped Interfaces on page 731

30.2.2. Reset Interface

The HPS reset request and handshake interfaces are connected to conduit BFMs for simulation.

Table 265. HPS Reset Input Interface Simulation Model

You can monitor the reset request interface state changes or set the interface by using the API listed.

Interface Name	BFM Instance Name	API Function Names
f2h_cold_reset_req	f2h_cold_reset_req_inst	get_f2h_cold_rst_req_n()
f2h_debug_reset_req	f2h_debug_reset_req_inst	get_f2h_dbg_rst_req_n()
f2h_warm_reset_req	f2h_warm_reset_req_inst	get_f2h_warm_rst_req_n()
h2f_warm_reset_handshake	h2f_warm_reset_handshake_inst	set_h2f_pending_rst_req_n() get_f2h_pending_rst_ack_n()

Table 266. HPS Reset Output Interface Simulation Model

The reset source BFM application programming interface applies to all the BFMs listed.

Interface Name	BFM Instance Name
h2f_reset	h2f_reset_inst
h2f_cold_reset	h2f_cold_reset_inst
h2f_debug_apb_reset	h2f_debug_apb_reset_inst

Table 267. Configuration of Reset Source BFM for HPS Reset Output Interface

The HPS reset output interface is connected to a reset source BFM. Platform Designer (Standard) configures the BFM as shown in the following table. The parameter value of the instantiated BFM is configured for HPS simulation.

Parameter	BFM Value	Meaning
Assert reset high	Off	This parameter is off, specifying an active-low reset signal from the BFM.
Cycles of initial reset	0	This parameter is 0, specifying that the BFM does not assert the reset signal automatically.

Related Information

- Memory-Mapped Interfaces on page 731
- Avalon Verification IP Suite User Guide

30.3. FPGA-to-HPS AXI Slave Interface

The FPGA-to-HPS AXI slave interface, `f2h_axi_slave`, is connected to a Mentor Graphics AXI slave BFM for simulation with an instance name of `f2h_axi_slave_inst`. Platform Designer (Standard) configures the BFM as shown in the following table. The BFM clock input is connected to `f2h_axi_clock` clock.

Table 268. Configuration of FPGA-to-HPS AXI Slave BFM

Parameter	Value
AXI Address Width	32
AXI Read Data Width	32, 64, or 128
AXI Write Data Width	32, 64, or 128
AXI ID Width	8

You control and monitor the AXI slave BFM by using the BFM API.

Related Information

- Memory-Mapped Interfaces on page 731
- Mentor Graphics Verification IP Altera Edition AMBA AXI3 and AXI4 User Guide
  The Mentor Graphics Verification IP User guide provides details of the API and connection guidelines for the AXI3 and AXI4 BFMs.

30.4. HPS-to-FPGA AXI Master Interface

The HPS-to-FPGA AXI master interface, `h2f_axi_master`, is connected to a Mentor Graphics AXI master BFM for simulation with an instance name of `h2f_axi_master_inst`. In Platform Designer (Standard), you can configure the HPS-to-FPGA interface with the following address, data, and ID widths. The BFM clock input is connected to `h2f_axi_clock` clock.

Table 269. Configuration of HPS-to-FPGA AXI Master BFM

Parameter	Value
AXI Address Width	30
AXI Read and Write Data Width	32, 64, or 128
AXI ID Width	12

You control and monitor the AXI master BFM by using the BFM API.

Related Information

- Memory-Mapped Interfaces on page 731
- Mentor Graphics Verification IP Altera Edition AMBA AXI3 and AXI4 User Guide
  The Mentor Graphics Verification IP User guide provides details of the API and connection guidelines for the AXI3 and AXI4 BFMs.

30.5. Lightweight HPS-to-FPGA AXI Master Interface

The lightweight HPS-to-FPGA AXI master interface, `h2f_lw_axi_master`, is connected to a Mentor Graphics AXI master BFM for simulation with an instance name of `h2f_lw_axi_master_inst`. Platform Designer (Standard) configures the BFM as shown in the following table. The BFM clock input is connected to `h2f_lw_axi_clock` clock.

Table 270. Configuration of Lightweight HPS-to-FPGA AXI Master BFM

Parameter	Value
AXI Address Width	21
AXI Read and Write Data Width	32
AXI ID Width	12

You control and monitor the AXI master BFM by using the BFM API.

Related Information

- Memory-Mapped Interfaces on page 731
- Mentor Graphics Verification IP Altera Edition AMBA AXI3 and AXI4 User Guide
  The Mentor Graphics Verification IP User guide provides details of the API and connection guidelines for the AXI3 and AXI4 BFMs.

30.6. HPS-to-FPGA MPU Event Interface

The HPS-to-FPGA MPU event interface is connected to a conduit BFM for simulation. The following table lists the name of each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API listed.

Table 271. HPS-to-FPGA MPU Event Interface Simulation Model

The usage of conduit `get_*()` and `set_*()` API functions is the same as with the general Avalon conduit BFM.

Interface Name	BFM Instance Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names

`h2f_mpu_events`	`h2f_mpu_events_inst`	`get_h2f_mpu_eventi()`
		`set_h2f_mpu_evento()`
		`set_h2f_mpu_standbywfe()`
		`set_h2f_mpu_standbywfi()`
			`get_eventi()`
			`set_evento()`
			`set_standbywfe()`
			`set_standbywfi()`

Related Information

Avalon Verification IP Suite User Guide

30.7. Interrupts Interface

The FPGA-to-HPS interrupts interface is connected to an Avalon interrupt sink BFM for simulation.

Table 272. FPGA-to-HPS Interrupts Interface Simulation Model

Interface Name	BFM Instance Name
f2h_irq0	f2h_irq0_inst
f2h_irq1	f2h_irq1_inst

The HPS-to-FPGA peripheral interfaces are connected to conduit BFMs for simulation. When you enable the peripheral interrupt, the corresponding peripheral signal to the FPGA is exposed.

Table 273. HPS-to-FPGA Peripherals Interrupt Interface Simulation Model

Interface Name	BFM Instance Name
h2f_clkmgr_interrupt	h2f_clkmgr_interrupt_inst
h2f_mpuwakeup_interrupt	h2f_mpuwakeup_interrupt_inst
h2f_ncti_interrupt0	h2f_ncti_interrupt0_inst
h2f_ncti_interrupt1	h2f_ncti_interrupt1_inst
h2f_dma_interrupt0	h2f_dma_interrupt0_inst
h2f_dma_interrupt1	h2f_dma_interrupt1_inst
h2f_dma_interrupt2	h2f_dma_interrupt2_inst
h2f_dma_interrupt3	h2f_dma_interrupt3_inst
h2f_dma_interrupt4	h2f_dma_interrupt4_inst
h2f_dma_interrupt5	h2f_dma_interrupt5_inst
h2f_dma_interrupt6	h2f_dma_interrupt6_inst
h2f_dma_interrupt7	h2f_dma_interrupt7_inst
h2f_dma_abort_interrupt	h2f_dma_abort_interrupt_inst
h2f_fpga_man_interrupt	h2f_fpga_man_interrupt_inst
h2f_gpio0_interrupt	h2f_gpio0_interrupt_inst
h2f_gpio1_interrupt	h2f_gpio1_interrupt_inst
h2f_gpio2_interrupt	h2f_gpio2_interrupt_inst
h2f_hmc_interrupt	h2f_hmc_interrupt_inst
h2f_timer_l4sp_0_interrupt	h2f_timer_l4sp_0_interrupt_inst
h2f_timer_l4sp_1_interrupt	h2f_timer_l4sp_1_interrupt_inst
h2f_timer_sys_0_interrupt	h2f_timer_sys_0_interrupt_inst
h2f_timer_sys_1_interrupt	h2f_timer_sys_1_interrupt_inst
h2f_ecc_serr_interrupt	h2f_ecc_serr_interrupt_inst
h2f_ecc_derr_interrupt	h2f_ecc_derr_interrupt_inst
h2f_parity_l1_interrupt	h2f_parity_l1_interrupt_inst
h2f_wdog0_interrupt	h2f_wdog0_interrupt_inst
h2f_wdog1_interrupt	h2f_wdog1_interrupt_inst

30.8. HPS-to-FPGA Debug APB Interface

The HPS-to-FPGA debug APB interface is connected to an Intel conduit BFM for simulation. The following table lists the name of each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API functions listed.

Table 274. HPS-to-FPGA Debug APB Interface Simulation Model

Interface Name	BFM Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
h2f_debug_apb	h2f_debug_apb	set_h2f_dbg_apb_PADDR() set_h2f_dbg_apb_PADDR_31() set_h2f_dbg_apb_PENABLE() get_h2f_dbg_apb_PRDATA() get_h2f_dbg_apb_PREADY() set_h2f_dbg_apb_PSEL() get_h2f_dbg_apb_PSLVERR() set_h2f_dbg_apb_PWDATA() set_h2f_dbg_apb_PWRITE()	set_PADDR() set_PADDR_31() set_PENABLE() get_PRDATA() get_PREADY() set_PSEL() get_PSLVERR() set_PWDATA() set_PWRITE()
h2f_debug_apb_sideband	h2f_debug_apb_sideband	get_h2f_dbg_apb_PCLKEN() get_h2f_dbg_apb_DBG_APB_DISABLE()	get_PCLKEN() get_DBG_APB_DISABLE()

30.9. FPGA-to-HPS System Trace Macrocell Hardware Event Interface

The FPGA-to-HPS STM hardware event interface is connected to a conduit BFM for simulation. The following table lists the name of each interface, along with the API function name for each type of simulation. You can monitor the interface state changes or set the interface by using the API functions listed.

Table 275. FPGA-to-HPS STM Hardware Event Interface Simulation Model

Interface Name	BFM Name	RTL Simulation API Function Name	Post-Fit Simulation API Function Name
f2h_stm_hw_events	f2h_stm_hw_events_inst	get_f2h_stm_hwevents()	get_stm_events()

30.10. HPS-to-FPGA Cross-Trigger Interface

The HPS-to-FPGA cross-trigger interface is connected to a conduit BFM for simulation. The following table lists the name of each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API functions listed.

Table 276. HPS-to-FPGA Cross-Trigger Interface Simulation Model

Interface Name	BFM Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
h2f_cti	h2f_cti_inst	get_h2f_cti_trig_in()	get_trig_in()
		set_h2f_cti_trig_in_ack()	set_trig_inack()

continued...

Interface Name	BFM Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
		set_h2f_cti_trig_out()	set_trig_out()
		get_h2f_cti_trig_out_ack()	get_trig_outack()
		get_h2f_cti_fpga_clk_en()	get_clk_en()

30.11. FPGA-to-HPS DMA Handshake Interface

The FPGA-to-HPS DMA handshake interface is connected to a conduit BFM for simulation. The following table lists the name for each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API listed.

Table 277. FPGA-to-HPS DMA Handshake Interface Simulation Model

The usage of conduit `get_*()` and `set_*()` API functions is the same as with the general Avalon conduit BFM.

Interface Name	BFM Instance Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
f2h_dma0	f2h_dma0_inst	get_f2h_dma0_req() get_f2h_dma0_single() set_f2h_dma0_ack()	get_channel0_req() get_channel0_single() set_channel0_xx_ack()
f2h_dma1	f2h_dma1_inst	get_f2h_dma1_req() get_f2h_dma1_single() set_f2h_dma1_ack()	get_channel1_req() get_channel1_single() set_channel1_xx_ack()
f2h_dma2	f2h_dma2_inst	get_f2h_dma2_req() get_f2h_dma2_single() set_f2h_dma2_ack()	get_channel2_req() get_channel2_single() set_channel2_xx_ack()
f2h_dma3	f2h_dma3_inst	get_f2h_dma3_req() get_f2h_dma3_single() set_f2h_dma3_ack()	get_channel3_req() get_channel3_single() set_channel3_xx_ack()
f2h_dma4	f2h_dma4_inst	get_f2h_dma4_req() get_f2h_dma4_single() set_f2h_dma4_ack()	get_channel4_req() get_channel4_single() set_channel4_xx_ack()
f2h_dma5	f2h_dma5_inst	get_f2h_dma5_req() get_f2h_dma5_single() set_f2h_dma5_ack()	get_channel5_req() get_channel5_single() set_channel5_xx_ack()
f2h_dma6	f2h_dma6_inst	get_f2h_dma6_req() get_f2h_dma6_single() set_f2h_dma6_ack()	get_channel6_req() get_channel6_single() set_channel6_xx_ack()
f2h_dma7	f2h_dma7_inst	get_f2h_dma7_req() get_f2h_dma7_single() set_f2h_dma7_ack()	get_channel7_req() get_channel7_single() set_channel7_xx_ack()

Related Information

Avalon Verification IP Suite User Guide

30.12. Boot from FPGA Interface

The boot from FPGA interface is connected to a conduit BFM for simulation. You can monitor the interface state changes or set the interface by using the API functions in the table below.

Table 278. Boot from FPGA Interface Simulation Model

Interface Name	BFM Name	RTL simulation API Function Names	Post-fit Simulation API Function Names
f2h_boot_from_fpga	f2h_boot_from_fpga_inst	get_f2h_boot_from_fpga_ready() get_f2h_boot_from_fpga_on_failure()	get_boot_fpga_ready() get_boot_from_fpga_on_failure()

30.13. Security Manager Anti-Tamper Signals Interface

The security manager anti-tamper signals interface is connected to a conduit BFM for simulation. You can monitor the interface state changes or set the interface by using the API listed.

Table 279. Security Manager Anti-Tamper Signals Interface Simulation Model

Interface Name	BFM Instance Name	RTL Simulation API Function Names
h2f_security	h2f_security_inst	get_f2h_security_anti_tamper_in set_h2f_security_anti_tamper_out

30.14. EMIF Conduit

Enables the HPS dedicated conduit to the Arria 10 External memory Interface for HPS. This conduit cannot connect to any other External memory Interface (EMIF). Only IP generated by the Arria 10 External memory Interface for HPS Platform Designer library should be used.

Table 280. EMIF Conduit Interface Simulation Model

Interface Name	BFM Instance Name	RTL Simulation API Function Names
emif	emif_inst	emif_emif_to_hps emif_hps_to_emif

30.15. Pin MUX and Peripherals

Under Pin MUX and Peripherals, the **Pin Mux GUI** has three folder tabs with the following functions:

- IP Selection
- Advanced Pin Placement
- Advanced FPGA Placement

The **IP Selection** tab contains two sub-windows. In the left sub-window, you are able to select the following:

- Boot source — SD/MMC, NAND, and QSPI
- Pin routing decisions between:
  - IP and HPS I/O—where your selections are reflected on the **Advanced Pin Placement**
  - IP and FPGA—where your selections are reflected on the **Advanced FPGA Placement**
- NAND bit-width when the NAND IP is selected
- SD/MMC bit-width and SD/MMC Power Enable when the SD/MMC IP is selected
- RGMII and PHY Options for each of the EMACs when the EMAC IP is selected
- Additional QSPI Slave Selects when the QSPI IP is selected

After you have made your selection, you must click on the **Apply Selections** button and select "Show signals" so that you can see your selection reflected in the right sub-window. Depending on if you are routing to the HPS I/O or the FPGA, your selection is reflected on the **Advanced Pin Placement** or **Advanced FPGA Placement** tab, respectively.

Related Information

Configuring Peripherals on page 726

30.15.1. HPS Conduit Interfaces Connecting to the HPS I/O

The **Pin Mux and Peripherals** interface, `hps_io`, is connected to an Intel Platform Designer (Standard) conduit BFM for simulation. Platform Designer (Standard) configures the BFM as shown in the following tables:

Table 281. PLL Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1 .. 5	N/A	`hps_io_phery_cm_PLL_CLK0 ()` .. `hps_io_phery_cm_PLL_CLK4 ()`

Table 282. SD/MMC Interface Simulation Model

This peripheral is a boot source.

To HPS I/Os	Options	Signal(s)

1	**SDMMC bit-width** = Default (1)
	SDMMC Power Enable = Yes
		`hps_io_phery_sdmmc_CMD()`
		`hps_io_phery_sdmmc_PWR_ENA()`, only when **SDMMC Power Enable** option is set to "Yes"
		`hps_io_phery_sdmmc_D0()`
		`hps_io_phery_sdmmc_CCLK ()`

	SDMMC bit-width = 4 **SDMMC Power Enable** = Yes[60]	The same signal list from above with **SDMMC bit-width** set to "default", plus the following signals: `hps_io_phery_sdmmc_D1()`

continued...

(60) When **SDMMC Power Enable** is set to Default, the `hps_io_phery_sdmmc_PWR_ENA()` signal is not available.

To HPS I/Os	Options	Signal(s)
		hps_io_phery_sdmmc_D2() hps_io_phery_sdmmc_D3()
	SDMMC bit-width = 8 **SDMMC Power Enable** = Yes(60)	The same signal list from above with **SDMMC bit-width** set to 4, plus the following signals: hps_io_phery_sdmmc_D4() hps_io_phery_sdmmc_D5() hps_io_phery_sdmmc_D6() hps_io_phery_sdmmc_D7()

Table 283. USB Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1	N/A	hps_io_phery_usb1_DATA0 () hps_io_phery_usb1_DATA1 () hps_io_phery_usb1_DATA2 () hps_io_phery_usb1_DATA3 () hps_io_phery_usb1_DATA4 () hps_io_phery_usb1_DATA5 () hps_io_phery_usb1_DATA6 () hps_io_phery_usb1_DATA7 () hps_io_phery_usb1_CLK () hps_io_phery_usb1_STP () hps_io_phery_usb1_DIR () hps_io_phery_usb1_NXT ()
2		hps_io_phery_usb0_DATA0 () hps_io_phery_usb0_DATA1 () hps_io_phery_usb0_DATA2 () hps_io_phery_usb0_DATA3 () hps_io_phery_usb0_DATA4 () hps_io_phery_usb0_DATA5 () hps_io_phery_usb0_DATA6 () hps_io_phery_usb0_DATA7 () hps_io_phery_usb0_CLK () hps_io_phery_usb0_STP () hps_io_phery_usb0_DIR () hps_io_phery_usb0_NXT () Add the same signal list from above to this list of signals.

Table 284. EMAC Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1	EMAC A is available for selecting options. See the signals in the "Signal(s)" column.	**RGMII?** = RMII and **PHY Options** = None (Default List) *Note:* Where <i> is "To HPS I/Os" - 1. hps_io_phery_emac<i>_TX_CLK () hps_io_phery_emac<i>_TXD0 () hps_io_phery_emac<i>_TXD1 () hps_io_phery_emac<i>_RX_CTL () hps_io_phery_emac<i>_TX_CTL () hps_io_phery_emac<i>_RX_CLK () hps_io_phery_emac<i>_RXD0 ()

continued...

To HPS I/Os	Options	Signal(s)
		hps_io_phery_emac<i>_RXD1 () When **RGMII?** = RGMII Add the following signals to the Default List: hps_io_phery_emac<i>_TXD2 () hps_io_phery_emac<i>_TXD3 () hps_io_phery_emac<i>_RXD2 () hps_io_phery_emac<i>_RXD3 () When **PHY Options** = MDIO Add the following signals to the Default List: hps_io_phery_emac<i>_MDIO () hps_io_phery_emac<i>_MDC () When **PHY Options** = I2C Add the following signals to the Default List: hps_io_phery_i2cemac<i>_SDA () hps_io_phery_i2cemac<i>_SCL ()
2	EMAC A and EMAC B are available for selecting options. See the signals in the "Signal(s)" column.	This signal list is comprised of the list for when "To HPS I/Os" is equal to 1, taking in to consideration the various options, plus the following: *Note:* Where <i> is "To HPS I/Os" - 1. **RGMII?** = RMII and **PHY Options** = None (Default List) hps_io_phery_emac<i>_TX_CLK () hps_io_phery_emac<i>_TXD0 () hps_io_phery_emac<i>_TXD1 () hps_io_phery_emac<i>_RX_CTL () hps_io_phery_emac<i>_TX_CTL () hps_io_phery_emac<i>_RX_CLK () hps_io_phery_emac<i>_RXD0 () hps_io_phery_emac<i>_RXD1 () When **RGMII?** = RGMII Add the following signals to the Default List: hps_io_phery_emac<i>_TXD2 () hps_io_phery_emac<i>_TXD3 () hps_io_phery_emac<i>_RXD2 () hps_io_phery_emac<i>_RXD3 () When **PHY Options** = MDIO Add the following signals to the Default List: hps_io_phery_emac<i>_MDIO () hps_io_phery_emac<i>_MDC () When **PHY Options** = I2C Add the following signals to the Default List: hps_io_phery_i2cemac<i>_SDA () hps_io_phery_i2cemac<i>_SCL ()
3	EMAC A, EMAC B, and EMAC C are available for selecting options. See the signals in the "Signal(s)" column.	This signal list is comprised of the list for when "To HPS I/Os" is equal to 1 and 2, taking in to consideration the various options, plus the following: *Note:* Where <i> is "To HPS I/Os" - 1. **RGMII?** = RMII and **PHY Options** = None (Default List) hps_io_phery_emac<i>_TX_CLK () hps_io_phery_emac<i>_TXD0 () hps_io_phery_emac<i>_TXD1 () hps_io_phery_emac<i>_RX_CTL () hps_io_phery_emac<i>_TX_CTL () hps_io_phery_emac<i>_RX_CLK ()

continued...

To HPS I/Os	Options	Signal(s)
		hps_io_phery_emac<i>_RXD0 () hps_io_phery_emac<i>_RXD1 () When **RGMII?** = RGMII Add the following signals to the Default List: hps_io_phery_emac<i>_TXD2 () hps_io_phery_emac<i>_TXD3 () hps_io_phery_emac<i>_RXD2 () hps_io_phery_emac<i>_RXD3 () When **PHY Options** = MDIO Add the following signals to the Default List: hps_io_phery_emac<i>_MDIO () hps_io_phery_emac<i>_MDC () When **PHY Options** = I2C Add the following signals to the Default List: hps_io_phery_i2cemac<i>_SDA () hps_io_phery_i2cemac<i>_SCL ()

Table 285. SPIM Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1	N/A	hps_io_phery_spim0_CLK () hps_io_phery_spim0_MOSI () hps_io_phery_spim0_MISO () hps_io_phery_spim0_SS0_N ()
2		The same list as above plus the following signals: hps_io_phery_spim1_CLK () hps_io_phery_spim1_MOSI () hps_io_phery_spim1_MISO () hps_io_phery_spim1_SS0_N ()

Table 286. SPIS Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1	N/A	hps_io_phery_spis0_CLK () hps_io_phery_spis0_MOSI () hps_io_phery_spis0_MISO () hps_io_phery_spis0_SS0_N ()
2		The same list as above plus the following signals: hps_io_phery_spis1_CLK () hps_io_phery_spis1_MOSI () hps_io_phery_spis1_MISO () hps_io_phery_spis1_SS0_N ()

Table 287. UART Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1	N/A	hps_io_phery_uart0_RX () hps_io_phery_uart0_TX () hps_io_phery_uart0_CTS_N ()

continued...

To HPS I/Os	Options	Signal(s)
		hps_io_phery_uart0_RTS_N ()
2		The same list as above plus the following signals: hps_io_phery_uart1_RX () hps_io_phery_uart1_TX () hps_io_phery_uart1_CTS_N () hps_io_phery_uart1_RTS_N ()

Table 288. I²C Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1	N/A	hps_io_phery_i2c0_SDA () hps_io_phery_i2c0_SCL ()
2		The same list as above plus the following signals: hps_io_phery_i2c1_SDA () hps_io_phery_i2c1_SCL ()

Table 289. NAND Interface Simulation Model

This peripheral is a boot source.

To HPS I/Os	Options	Signal(s)
1	**NAND bit-width** = 8	hps_io_phery_nand_ALE () hps_io_phery_nand_CE_N () hps_io_phery_nand_CLE () hps_io_phery_nand_RE_N () hps_io_phery_nand_RB () hps_io_phery_nand_ADQ0 () hps_io_phery_nand_ADQ1 () hps_io_phery_nand_ADQ2 () hps_io_phery_nand_ADQ3 () hps_io_phery_nand_ADQ4 () hps_io_phery_nand_ADQ5 () hps_io_phery_nand_ADQ6 () hps_io_phery_nand_ADQ7 () hps_io_phery_nand_WP_N () hps_io_phery_nand_WE_N ()
	NAND bit-width = 16	The same list as above plus the following signals: hps_io_phery_nand_ADQ8 () hps_io_phery_nand_ADQ9 () hps_io_phery_nand_ADQ10 () hps_io_phery_nand_ADQ11 () hps_io_phery_nand_ADQ12 () hps_io_phery_nand_ADQ13 () hps_io_phery_nand_ADQ14 () hps_io_phery_nand_ADQ15 ()

Table 290. TRACE Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1	N/A	hps_io_phery_trace_CLK () hps_io_phery_trace_D0 ()

To HPS I/Os	Options	Signal(s)
		hps_io_phery_trace_D1 () hps_io_phery_trace_D2 () hps_io_phery_trace_D3 ()

Table 291. GPIO Pins Interface Simulation Model

To HPS I/Os	Options	Signal(s)
1 .. 62	N/A	hps_io_gpio_gpio2_io0 () .. hps_io_gpio_gpio2_io62 ()

Table 292. QSPI Interface Simulation Model

This peripheral is a boot source.

To HPS I/Os	Options	Signal(s)
1	N/A *Note:* Although there are no options for this peripheral, "Boot" must be selected also for the signals to become available.	hps_io_phery_qspi_IO0() hps_io_phery_qspi_IO1() hps_io_phery_qspi_IO2_WPN() hps_io_phery_qspi_IO3_HOLD() hps_io_phery_qspi_CLK () hps_io_phery_qspi_SS0()

30.15.2. HPS Conduit Interfaces Connecting to the FPGA

When connecting to the FPGA, the **Pin Mux and Peripherals** interface is indicative to the peripheral name and represented in the "Conduit" column in the following table. The following tables for each of the peripherals, lists the conduits and their interfaces. In order to see the signals, you must select "Show Signals".

Table 293. SD/MMC Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	sdmmc	sdmmc_reset sdmmc_clk

Table 294. EMAC Pins Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	emac0 i2cemac0 — when **PHY Options** is set to "I2C"	emac_ptp_ref_clock emac0_rx_clk_in emac0_tx_clk_in emac0_gtx_clk emac0_tx_reset emac0_rx_reset emac0_md_clk — when **PHY Options** is set to "MDIO" i2cemac0_scl_in — when **PHY Options** is set to "I2C"
2	emac1 i2cemac1 — when **PHY Options** is set to "I2C"	emac_ptp_ref_clock emac1_rx_clk_in emac1_tx_clk_in emac1_gtx_clk

continued...

To FPGA	Conduit	Interface(s)
		emac1_tx_reset emac1_rx_reset emac1_md_clk — when **PHY Options** is set to "MDIO" i2cemac1_scl_in — when **PHY Options** is set to "I2C"
3	emac2 i2cemac2 — when **PHY Options** is set to "I2C"	emac_ptp_ref_clock emac2_rx_clk_in emac2_tx_clk_in emac2_gtx_clk emac2_tx_reset emac2_rx_reset emac2_md_clk — when **PHY Options** is set to "MDIO" i2cemac2_scl_in — when **PHY Options** is set to "I2C"

Table 295. SPIM Pins Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	spim0	spim0_sclk_out
2	spim0 spim1	spim0_sclk_out spim1_sclk_out

Table 296. SPIS Pins Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	spis0	spis0_sclk_out
2	spis0 spis1	spis0_sclk_out spis1_sclk_out

Table 297. UART Pins Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	uart0	None
2	uart0 uart1	

Table 298. I²C Pins Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	i2c0	i2c0_scl_in i2c0_clk
2	i2c0 i2c1	i2c0_scl_in i2c0_clk i2c1_scl_in i2c1_clk

Table 299. NAND Pins Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	nand	None

Table 300. Trace Pins Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	trace	trace_s2f_clk

Table 301. QSPI Pins Interface Simulation Model

To FPGA	Conduit	Interface(s)
1	qspi	None

A. Booting and Configuration

The Intel system-on-a-chip (SoC) device provides a variety of ways to boot the hard processor system (HPS) and configure the FPGA portion.

Related Information

Intel Arria 10 Hard Processor System Technical Reference Manual Revision History on page 14
For details on the document revision history of this chapter

A.1. Boot Overview

The boot flow of the HPS is a multi-stage process. Each stage is responsible for loading the next stage.

The first stage is the boot ROM execution. The boot ROM code, located in the HPS, brings the processor out of reset, puts the processor into a known and stable state, finds the second-stage boot loader and passes control to the next stage. The boot ROM code is only aware of the second-stage boot loader and not aware of any potential subsequent software stages. During this time, the boot ROM also seamlessly handles any error conditions that may occur.

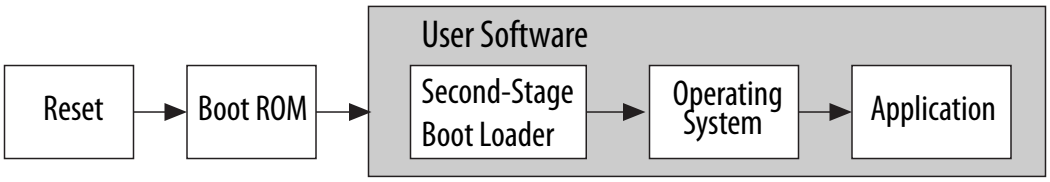
The next stage is when control passes to the second-stage boot loader. The second-stage boot loader is located external to the HPS, either in off-chip flash memory or within the FPGA. If the FPGA is used, the second stage boot loader can execute directly from the FPGA without the need to copy it to on-chip RAM. The second stage-boot loader locates and loads the next stage software and so on.

Before the control is passed to the second stage boot loader, it can be decrypted and authenticated if a secure boot is enabled.

After a warm reset, the user can program the HPS to instruct the boot ROM to find an image in the on-chip RAM and execute directly from that. In this case, the image that resides in RAM is unauthenticated and clear text, although it may have been imported into on-chip RAM as authenticated code initially.

The figure below illustrates the typical boot flow. However, there may be more or less software stages in the user software than shown and the roles of the software stages may vary.

Figure 164. Typical Boot Flow



Intel Corporation. All rights reserved. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.
*Other names and brands may be claimed as the property of others.

ISO 9001:2015 Registered

Related Information

SoC Security on page 168
For more information about secure boot, refer to the *SoC Security* chapter.

A.2. FPGA Configuration Overview

After power is applied to the FPGA and HPS portion of the SoC, configuration may begin. The FPGA may be configured through a variety of ways, such as through the HPS, JTAG, or an external host.

The Configuration Subsystem (CSS) in the FPGA portion of the device is responsible for obtaining an FPGA configuration image and configuring the FPGA fabric, as well as the IOCSRs (I/O Configuration and Status Registers). The I/O configuration file programs the following in the IOCSRs:

- FPGA I/O
- Shared I/O
- Hard memory controller I/O
- Hard memory controller settings, such as memory type, frequency and timings
- FPGA PLL settings
- Transceiver settings

The FPGA configuration ends when the configuration image has been fully loaded and the FPGA enters user mode. The FPGA configuration image is typically stored in external non-volatile flash-based memory. The FPGA Configuration Subsystem can obtain a configuration image from the HPS through the FPGA Manager or from any of the sources supported by the FPGA family.

Related Information

- FPGA Manager on page 154
  For more information regarding programming the FPGA through the HPS, refer to the *FPGA Manager* chapter.
- Configuration, Design Security, and Remote System Upgrades in Arria 10 Devices

A.3. Booting and Configuration Options

SoC initialization includes the booting of the HPS and the configuration of the FPGA fabric and I/O.

The I/O provided in the SoC are:

- HPS dedicated I/O that are configured by the HPS
- Shared I/O used by the HPS or FPGA and configured by the Intel Quartus Prime I/O configuration file.
- FPGA I/O configured by the Intel Quartus Prime I/O configuration file.
- Hard memory controller I/O shared by the HPS and FPGA and configured by the Intel Quartus Prime I/O configuration file.

Depending on the initialization option you choose, the I/O configuration is handled differently. You can choose one of the three initialization options:

- The HPS boot and FPGA configuration occur separately.
- The HPS boots first and then configures the FPGA.
- The HPS boots from the FPGA after the FPGA is configured.

Note: The HPS and FPGA fabric must be powered at the same time. You must not power-down the HPS or FPGA independently in the middle of device operation.

The following three figures illustrate the possible HPS boot and FPGA configuration schemes. The arrows in the figures denote the data flow direction.

Figure 165. Separate FPGA Configuration and HPS Booting

In the figure below, the FPGA configuration and HPS boot can occur separately. The FPGA obtains its configuration image from a non-HPS source. The source is sent to the CSS block which configures the FPGA fabric, FPGA I/O, shared I/O, hard memory controller I/O and other settings. If the hard memory controller or shared I/O are required by the HPS, then the FPGA fabric and I/O (FPGA, shared and hard memory controller) configuration must be complete before the HPS can boot.

The HPS boot ROM obtains its second-stage boot loader from a non-FPGA fabric source. The second-stage boot loader should contain the configuration for the HPS dedicated I/O.

For more information about FPGA configuration, refer to the "FPGA Configuration" section.

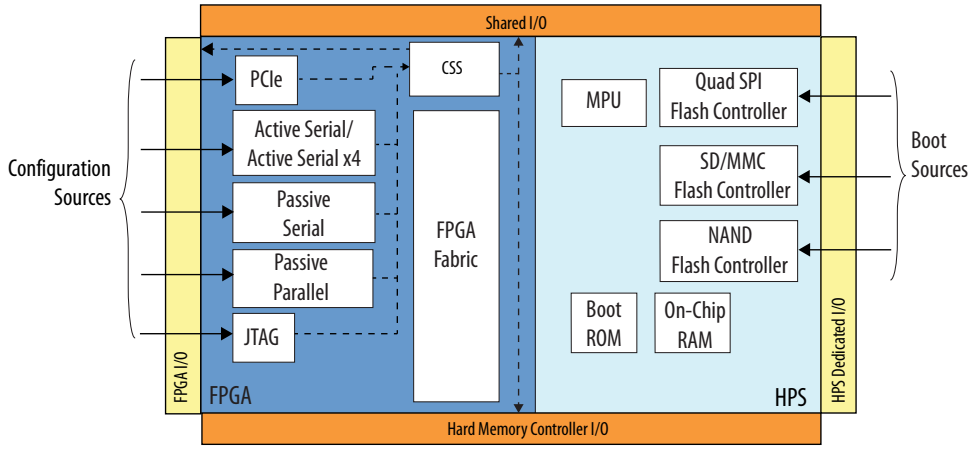


Figure 166. HPS Boots First and then Configures the FPGA

In the figure below, the HPS boots first through one of its non-FPGA fabric boot sources. If the hard memory controller or shared I/O are required by the HPS during booting then you can either execute a full FPGA configuration flow or an early I/O release configuration flow. The FPGA must be in a power-on state for the HPS to reset properly and for the second stage boot loader to initiate configuration through the FPGA Manager. The software executing on the HPS obtains the FPGA configuration image from any of its flash memory devices. For more information about full FPGA configuration or early I/O release configuration after HPS booting, refer to the "FPGA Configuration" section.

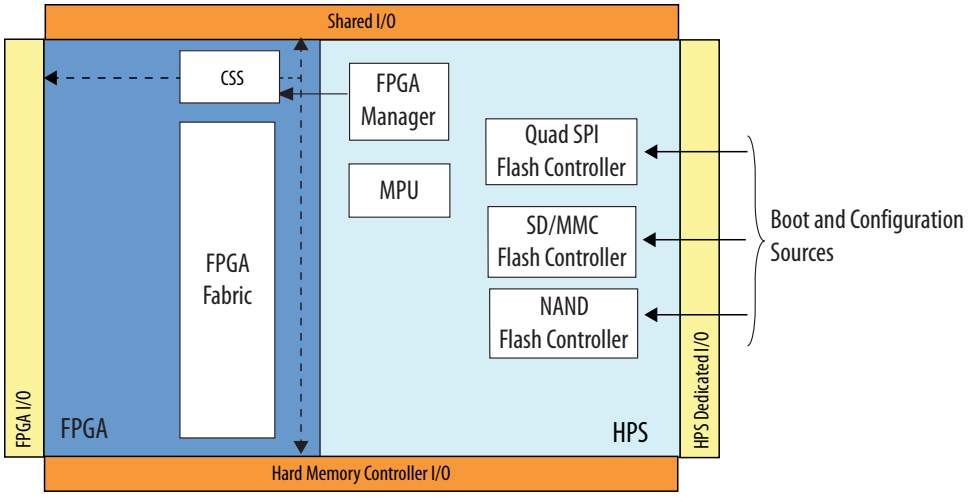
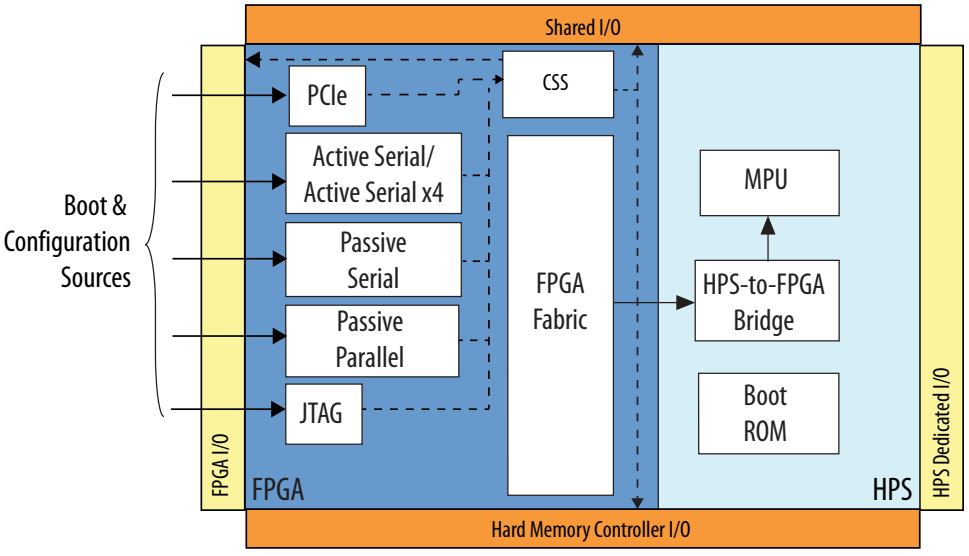


Figure 167. HPS Boots From FPGA

In the figure below, the FPGA is configured first through one of its non-HPS configuration sources. The CSS block configures the FPGA fabric as well as the FPGA I/O, shared I/O and hard memory controller I/O. The HPS boot ROM code executes the second-stage boot loader from the FPGA fabric over the HPS-to-FPGA bridge.

You can select a boot from FPGA by setting the BSEL value to 0x1. If the fpga_boot_f fuse is blown then the FPGA is always selected as the boot source. In both cases, when the FPGA is selected as the boot source, the CSEL fuses are ignored and clock configuration is controlled through the second-stage boot loader code in the FPGA.



Related Information

- FPGA Configuration on page 813
  For more information about full and early I/O release FPGA configuration
- SoC Security on page 168
  For more information about CSEL encodings and secure clock configuration, refer to the *SoC Security* chapter.

A.4. Boot Definitions

The following sections contain basic terms and definitions that are part of the boot process.

A.4.1. Reset

Reset precedes the boot stages and is an important part of device initialization. There are two different reset types: cold reset and warm reset.

The FPGA portion of the SoC can trigger a warm or cold reset on completion of configuration.

The boot process begins when CPU0 in the MPU exits from the reset state. When CPU0 exits from reset, it starts executing code at the reset exception address where the boot ROM code is located. CPU1 remains in reset during this time and is brought out of reset by user software.

With warm reset, some software registers are preserved and the boot process may skip some steps depending on software settings. In addition, on a warm reset, the second-stage boot loader has the ability to be executed from on-chip RAM.

Note: It is important that the HPS is not held in cold or warm reset indefinitely, otherwise the CSS cannot be accessible to FPGA configuration sources.

A.4.2. Boot ROM

The boot ROM code is 128 KB in size and located in on-chip ROM at address range 0xFFFC0000 to 0xFFFDFFFF. The function of the boot ROM code is to determine the boot source, initialize the HPS after a reset, and jump to the second-stage boot loader. In the case of indirect execution, the boot ROM code loads the second-stage boot loader image from the flash memory to on-chip RAM. The boot ROM performs the following actions to initialize the HPS:

- Enable instruction cache, branch predictor, floating point unit, NEON vector unit of CPU0
- Sets up the level 4 (L4) watchdog 0 timer
- Initializes the flash controller to default settings
- Configures HPS dedicated I/O

When booting from flash memory, the boot ROM code uses the top 32 KB of the on-chip RAM as data workspace. This area is reserved for the boot ROM code after a reset until the boot ROM code passes software control to second-stage boot loader. The maximum second-stage boot loader size is 208 KB with authentication and 224 KB

without. For a warm reset or cold reset with a boot from FPGA, the boot ROM code does not reserve the top 32 KB of the on-chip RAM, and the user may place user data in this area without being overwritten by the boot ROM.

The boot process begins when CPU0 exits from the reset state. The boot ROM code only executes on CPU0. CPU1 is held in reset while boot ROM executes on CPU0. When a CPU0 exits from reset, it starts executing code at the reset exception address.

During boot ROM execution, the clock control fuse information is automatically sent to the Clock Manager, the memory control fuse information is automatically sent to the Reset Manager and all other fuse functions (authentication, encryption, private and public key source, hash functions) are stored in a memory-mapped location for boot code to read. In normal operation, the boot ROM is mapped at the reset exception address so code starts executing in the boot ROM.

When CPU0 exits the boot ROM code and starts executing user software, the boot ROM access is disabled. The user software in CPU0 must map the user software exception vectors at 0x0 (which is previously mapped to boot ROM exception vectors). The user software also has the option of releasing CPU1 from reset. If CPU1 is released from reset, CPU1 executes the user software exception instead of boot ROM.

A.4.3. Boot Select

The boot select (BSEL) pins offer multiple methods to obtain the second-stage boot image. On a cold reset, the boot source is determined by a combination of secure boot fuses and BSEL pins. These fuse values and BSEL pin values are sent to the Security Manager module of the HPS when the cold reset occurs. When the HPS is released from reset, the boot ROM reads the `bootinfo` register of the System Manager to determine the source of the boot.

Note: If the fpga_boot_f fuse is blown, the BSEL pins are bypassed and the HPS can only boot from the FPGA. Additionally, the clock select (CSEL) fuse values are ignored and clock configuration is controlled through the FPGA. This configuration allows the HPS to boot from encrypted user code in the FPGA. If the boot source is the FPGA, the boot ROM code does not configure any of the boot-specific HPS I/Os for booting from flash memory. If the fpga_boot_f fuse is not blown, then the boot source is determined according to the BSEL pins. If the BSEL pins are used for determining the boot source, then the following table shows the flash devices assigned to each encoding.

Note: When booting from FPGA is selected (`BSEL[2:0]=0x1`), the Boot ROM waits until the FPGA is in user mode, and then it queries the handshake signals, `f2h_boot_from_fpga_ready` and `f2h_boot_from_fpga_on_failure`, from the FPGA to the HPS. The `f2h_boot_from_fpga_ready` signal must be pulled up to indicate readiness. Refer to the *Instantiating the HPS Component* chapter for more information about the FPGA boot handshake signals.

Note: The acronyms BSEL and BOOTSEL are used interchangeably to define the boot select pins.

Table 311. BSEL Values for Boot Source Selection

BSEL[2:0] Value	Flash Device
0x0	Reserved
0x1	FPGA (HPS-to-FPGA bridge)
0x2	1.8 V NAND flash memory
0x3	3.0 V NAND flash memory
0x4	1.8 V SD/MMC flash memory with external transceiver
0x5	3.0 V SD/MMC flash memory with internal transceiver
0x6	1.8 V quad SPI flash memory
0x7	3.0 V quad SPI flash memory

Note: During the HPS boot phase, the boot ROM only recognizes 1.8 V or 3.0 V dedicated I/O operation. If your dedicated I/O are operating at 2.5 V, then your BSEL value must be set to 3.0 V to support your operating I/O voltage at boot. After booting has completed, you may configure your dedicated I/O voltage settings to the true level by programming the `configuration_dedicated_io_bank` and `configuration_dedicated_io_*` registers in the `io48_pin_mux_dedicated_io_grp` address block.

Note: If the BSEL value is set to 0x4 or 0x5, an external translation transceiver may be required to supply level-shifting and isolation if the SD cards interfacing to the SD/MMC controller must operate at a different voltage than the controller interface. Please refer to the *SD/MMC Controller* chapter for more information.

The typical boot flow is for the boot ROM code to find the second-stage boot loader image on a flash device, load that into on-chip RAM and execute it. After a warm reset, the boot ROM code can be instructed to find the image in RAM and execute that.

The HPS flash sources can store various file types, such as:

- FPGA programming files
- Second-stage boot loader binary file (up to four copies)
- Operating system binary files
- Application file system

The second-stage boot loader image in flash can be authenticated and decrypted by the HPS. A boot directly from the HPS on-chip RAM is always unauthenticated and in clear text, although it may have an optional CRC if required.

When the BSEL value is 0x1, the FPGA is selected as the boot source for that boot. This selection is not permanent as it is when the fpga_boot_f fuse is enabled. In both cases, the CSEL fuses are also ignored and the HPS must be held in reset until the FPGA is powered on and programmed to prevent the boot ROM from misinterpreting the boot source.

If an HPS flash interface has been selected to load the boot image, then the boot ROM enables and configures that interface before loading the boot image into on-chip RAM, verifying it and passing software control to the second-stage boot loader.

If the FPGA fabric is the boot source, the boot ROM code waits until the FPGA portion of the device is in user mode, and is ready to execute code and then passes software control to the second-stage boot loader in the FPGA RAM.

Related Information

- SD/MMC Controller on page 405
  Refer to the *SD/MMC Controller* chapter for more information regarding features and functionality.
- SoC Security on page 168
  For more information about secure boot, refer to the *SoC Security* chapter.
- General Interfaces on page 717
  For more information about FPGA boot handshake signals, refer to the "General Interfaces" section of the *Instantiating HPS Component* chapter.

A.4.3.1. Boot Source I/O Pins

The HPS has 17 dedicated I/O pins. Three are used as a clock, cold reset and warm reset pin. The warm reset signal is a bidirectional signal; a warm reset event can drive into the HPS, or a warm reset event can be generated by the HPS and drive out of this pin.

The remaining 14 dedicated I/O pins are used for boot devices as well as other peripherals. Three of these 14 dedicated I/O pins are sampled by software at either cold or warm reset and convey boot source information. The following table identifies the pin mux values for the boot source options.

Table 312. Boot Source MUX Selects

This table identifies the dedicated HPS signals that are mapped to each boot interface and the mux select value each boot interface signal requires. Note that Clock Manager clock inputs are also assigned to MUX select 4 and are documented here for thoroughness.

Signal	MUX Select		
	14	8	4
HPS_DEDICATED_4	NAND_ADQ0	SDMMC_DATA0	QSPI_CLK
HPS_DEDICATED_5	NAND_ADQ1	SDMMC_CMD	QSPI_IO0
HPS_DEDICATED_6/BOOTSEL2	NAND_WE_N	SDMMC_CCLK	QSPI_SS0
HPS_DEDICATED_7	NAND_RE_N	SDMMC_DATA1	QSPI_IO1
HPS_DEDICATED_8	NAND_ADQ2	SDMMC_DATA2	QSPI_IO2_WPN
HPS_DEDICATED_9	NAND_ADQ3	SDMMC_DATA3	QSPI_IO3_HOLD
HPS_DEDICATED_10/BOOTSEL1	NAND_CLE	SDMMC_PWR_ENA	–
HPS_DEDICATED_11/BOOTSEL0	NAND_ALE	QSPI_SS1	–
HPS_DEDICATED_12	NAND_RB	SDMMC_DATA4	–
HPS_DEDICATED_13	NAND_CE_N	SDMMC_DATA5	–
HPS_DEDICATED_14	NAND_ADQ4	SDMMC_DATA6	–
HPS_DEDICATED_15	NAND_ADQ5	SDMMC_DATA7	–
HPS_DEDICATED_16	NAND_ADQ6	QSPI_SS2	–
HPS_DEDICATED_17	NAND_ADQ7	QSPI_SS3	–

Note: The MUX selects for the QSPI interface signals include both mux select 4 and mux select 8.

Table 313. Boot Source MUX Selects (Alternate View)

This table displays the same information as the prior table but as an alternative view per interface selected. The number in each cell shows the mux select value needed to select the correct interface signal on the pin

Signal	Boot Interface		
	NAND	SDMMC	QSPI
HPS_DEDICATED_4	14	8	4
HPS_DEDICATED_5	14	8	4
HPS_DEDICATED_6/BOOTSEL2	14	8	4
HPS_DEDICATED_7	14	8	4
HPS_DEDICATED_8	14	8	4
HPS_DEDICATED_9	14	8	4
HPS_DEDICATED_10/BOOTSEL1	14	8	Not Used
HPS_DEDICATED_11/BOOTSEL0	14	Not Used	8
HPS_DEDICATED_12	14	8	Not Used
HPS_DEDICATED_13	14	8	Not Used
HPS_DEDICATED_14	14	8	Not Used
HPS_DEDICATED_15	14	8	Not Used
HPS_DEDICATED_16	14	Not Used	8
HPS_DEDICATED_17	14	Not Used	8

Related Information

- Pin-Out Files for Arria 10 Devices
- Intel Intel Arria 10 Device Datasheet
  For information on BSEL sampling after reset deassertion

A.4.3.2. Boot Fuses

During boot ROM execution, the boot ROM reads user-programmed fuses that configure the boot state of the HPS.

These fuse values are read by the Configuration Subsystem (CSS) after power-up and to the Security Manager where they are read by the boot ROM. The following list describes the basic fuses that affect the state of the HPS:

- Clock select fuses: Determine the clock frequencies of the MPU clock, the interconnect clocks and the peripheral clocks.
- Debug fuses: Determine the state of debug during the boot process and during hand-off to the second-stage boot loader.
- RAM fuses: Determine whether or not the various peripheral RAMs are cleared during a warm reset and how they are cleared (series or parallel).

- Boot fuses: Determines whether the second-stage boot loader is from on-chip RAM, whether an internal or external clock is used for booting and whether the boot source is solely from FPGA.
- Security fuses: Determines whether the second-stage boot loader is authenticated or decrypted. If authentication is used, then there are fuses that are read by the boot ROM to determine where the key authorization key (KAK) resides and what its length is.
- User fuses: The user fuses may either hold proprietary user information or may hold the value of the KAK.

Table 314. HPS_fusesec Register Description

Bits	Name	Description
31:27	Reserved	Bit values in this field are undefined.
26:23	csel_f	This field indicates the value of the clock select fuses that are available for configuring the clock for the boot interface and for the PLLs. Refer to the *Clock Configuration* section for more information on CSEL encodings.
22	dbg_access_f	This fuse determines the initial state of the debug access domains.
21	dbg_lock_JTAG	This field indicates if the HPS JTAG access level can be changed through software when the HPS is released from reset. • 0x0= HPS JTAG access level can be changed through the sec_jtagdbg register. • 0x1= HPS JTAG access level cannot be changed (locked).
20	dbg_lock_DAP	This field indicates if the DAP access level can be changed through software when the HPS is released from reset. • 0x0= The DAP access level can be changed through the sec_dapdbg register. • 0x1= The DAP access level cannot be changed (locked).
19	dbg_lock_CPU0	This field indicates if the CPU0 debug access level can be changed through software when the HPS is released from reset. • 0x0= CPU0 debug access level can be changed through the sec_cpu0dbg register. • 0x1= CPU0 debug access level cannot be changed (locked).
18	dbg_lock_CPU1	This field indicates if the CPU1 debug access level can be changed through software when the HPS is released from reset. • 0x0= The CPU1 debug access level can be changed through the sec_cpu1dbg register. • 0x1= The CPU1 debug access level cannot be changed (locked).

continued...

Bits	Name	Description
17	dbg_lock_CS	This field indicates if the CoreSight debug access level can be changed through software when the HPS is released from reset. • 0x0= The CoreSight debug access level can be changed through the sec_csdbg register. • 0x1= The CoreSight debug access level cannot be changed (locked).
16	dbg_lock_FPGA	This field indicates if the FPGA debug access level can be changed through software when the HPS is released from reset. • 0x0= The FPGA debug access level can be changed through the sec_fpgadbg register. • 0x1= The FPGA debug access level cannot be changed (locked).
15:12	Reserved	Bit values in this field are undefined.
11	clr_ram_order_f	This fuse value determines how RAMs are cleared during a tamper event. • 0x0= All RAMs are cleared in parallel. • 0x1= All RAMs are cleared in series.
10	clr_ram_cold_f	This fuse value indicates what happens to the RAM on a cold reset. • 0x0= All RAMs are not cleared on a cold reset. • 0x1= All RAMs are cleared on a cold reset.
9	clr_ram_warm_f	This fuse value indicates what happens to the RAMs on a warm reset. • 0x0= All RAMs are not cleared on a warm reset. • 0x1= All RAMs are cleared on a warm reset.
8	oc_boot_f	This fuse value determines if the second-stage boot code is allowed to boot from on-chip RAM. • 0x0= Second-stage boot can be from on-chip RAM if enabled by the System Manager. • 0x1= Second-stage boot is not from on-chip RAM.
7	hps_clk_f	This fuse value selects the clock used for the boot process and in the case of a tamper event, memory scrambling. • 0x0= The external oscillator, HPS_CLK1, is used for boot. • 0x1= The internal oscillator, cb_intosc_ls_clk, is used for boot.

continued...

Bits	Name	Description
6	fpga_boot_f	If blown, this fuse value allows the FPGA to configure independently and allows the HPS to boot from an encrypted next-stage boot source that was decrypted into the FPGA. • 0x0= Booting is dependent on the BSEL pins. • 0x1= HPS only boots from the FPGA; BSEL options are ignored and CSEL fuse options are ignored.
5	aes_en_f	This fuse value indicates if a decryption of the flash image is always performed. • 0x0= An AES decryption of the flash image is determined from the second stage boot loader header. • 0x1= An AES decryption of the flash image is always performed.
4:2	kak_src_f	This bit field indicates the source of the Key Authorization Key (KAK) which can be in: • Proprietary ROM • FPGA logic elements • User fuses
1	kak_len_f	This fuse value indicates the Key Authorization Key (KAK) length: • 0x0= 256 bits • 0x1= 384 bits
0	authen_en_f	This fuse value determines whether authentication of flash images is required prior to execution. • 0x0= No authentication of the flash image is required prior to execution. • 0x1= HPS authentication of all flash images is required prior to execution.

For more information about the fuses and how they work, refer to the *SoC Security* chapter.

Related Information

SoC Security on page 168
For more information about secure boot, refer to the *SoC Security* chapter.

A.4.4. Flash Memory Devices for Booting

The memory controllers and devices that contain the boot loader image have configuration requirements for proper boot from flash.

On all flash devices, there is an area of memory, called the boot area that contains up to four second-stage boot loader images. For the QSPI and SD/MMC devices, the boot area is 1 MB in size. For NAND devices the boot area is four device blocks in size and may be larger than 1 MB if the NAND erase block is larger than 256 KB.

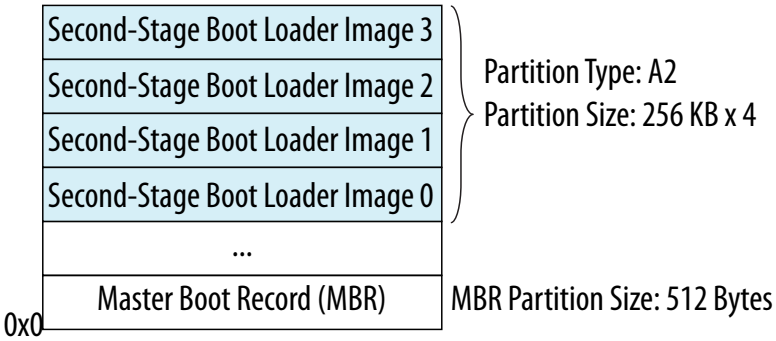
The SD/MMC, Quad SPI and NAND flash devices all support raw and MBR (partition) mode. In raw mode, the boot image is located at the start of the flash memory device, at offset 0x0. In MBR mode:

- The boot image is read from a custom partition (0xA2)
- The first image is located at the beginning of the partition, at offset 0x0
- Start address = partition start address

A.4.4.1. SD/MMC Flash Devices

The following figure shows an SD/MMC flash image layout example for boot. The master boot record (MBR) is located in the first 512 bytes of the memory. The MBR contains information about the partitions (address and size of partition). The second-stage boot loader image is stored in partition A2. Partition A2 is a custom raw partition with no file system.

Figure 168. SD/MMC Flash Image Layout



The SD/MMC controller supports two booting modes:

- MBR (partition) mode
  - The boot image is read from a custom partition (0xA2)
  - The first image is located at the beginning of the partition, at offset 0x0
  - Start address = partition start address
- Raw mode
  - If the MBR signature is not found, SD/MMC driver assumes it is in raw mode
  - The boot image data is read directly from sectors in the user area and is located at the first sector of the SD/MMC
  - The first image is located at the start of the memory card, at offset 0x0
  - Start address = 0x0

The MBR contains the partition table, which is always located in the first sector (LBA0) with a memory size of 512 bytes. The MBR consists of executable code, four partition entries, and the MBR signature. A MBR can be created by specific tools like the FDISK program.

Table 315. MBR Structure

Offset	Size (In Bytes)	Description
0x000	446	Code area
0x1BE	16	Partition entry for partition 1
0x1CE	16	Partition entry for partition 2
0x1DE	16	Partition entry for partition 3
0x1EE	16	Partition entry for partition 4
0x1FE	2	MBR signature: 0xAA55

The standard MBR structure contains a partition with four 16-byte entries. Thus, memory cards using this standard table cannot have more than four primary partitions or up to three primary partitions and one extended partition.

Each partition type is defined by the partition entry. The boot images are stored in a primary partition with custom partition type (0xA2). The SD/MMC flash driver does not support a file system, so the boot images are located in partition A2 at fixed locations.

Table 316. Partition Entry

Offset	Size (In Bytes)	Description
0x0	1	Boot indicator. 0x80 indicates that it is bootable.
0x1	3	Starting CHS value
0x4	1	Partition type
0x5	3	Ending CHS value
0x8	4	LBA of first sector in partition
0xB	4	Number of sectors in partition

The boot ROM code configures the SD/MMC controller to default settings for the supported SD/MMC flash memory.

Note: Please refer to the following Knowledge Base article for guidelines on partitioning your SD card image for booting the Intel Arria 10 SoC FPGA.

Related Information

SD/MMC Controller on page 405
Refer to the *SD/MMC Controller* chapter for more information regarding features and functionality.

A.4.4.1.1. Default Settings of the SD/MMC Controller

Table 317. SD/MMC Controller Default Settings

Parameter	Default	Register Value
Card type	1 bit	The card type register (`ctype`) in the SD/MMC controller registers (`sdmmc`) = 0x0
Bus mode	—	SD/MMC(61)

continued...

Parameter		Default	Register Value
Timeout		Maximum	The timeout register (`tmout`) = 0xFFFFFFFF
FIFO threshold RX watermark level		1	The RX watermark level field (`rx_wmark`) of the FIFO threshold watermark register (`fifoth`) = 0x1
Clock source		0	The clock source register (`clksrc`) = 0x0
Block size		512	The block size register (`blksiz`) = 0x200
Clock divider	Identification mode	32	The clock divider register (`clkdiv`)= 0x10 (2*16=32)
	Data transfer mode	Bypass	The clock divider register (`clkdiv`)= 0x00
External Device Power enable		Disabled (Power Off)	The `power_enable` bit in the `pwren` register is programmed to 0x0 out of reset. In Arria 10, the SD/MMC power enable is inverted. To compensate for this active low polarity, you can implement one of three options: • Force the power enable high on the board. • Use a GPIO to control the power enable. • Invert the power enable line on the board so that when software disables the power (`SDMMC_PWR_ENA_HPS` is high), the board inverts the signal to turn off the card.

(61) SPI cards are not supported for boot.

A.4.4.1.2. CSEL Settings for the SD/MMC Controller

Table 318. SD/MMC Controller Clock Options Based on CSEL and HPS_CLK fuse settings

Note: The `osc1_clk` signal is sourced from the external oscillator input, HPS_CLK1.

CSEL [3:0] Fuse Values	HPS CLK Fuse Value	Required Input Clock Range	Controller Clock	Controller Clock Frequency	ID Mode: Baud Rate Divisor	ID Mode: Device Clock	Data Transfer Mode: Baud Rate Divisor	Data Transfer Mode: Device Clock	PLL Status
0x0-0x1 and 0x3-0xF	1	60-200 MHz (Secure Bypass)	`cb_intosc_hs_clk/4`	15-50 MHz	128	29-97.75 KHz	4	937.5 KHz-3.125 MHz	Bypass
0x2	1	30-100 MHz (Secure PLL)	`cb_intosc_ls_clk/8`	2.75-12.5 MHz	32	29-97.75 KHz	1 (Bypass)	687.5 KHz-3.125 MHz	Locked
0x1	0	10-50 MHz (Untouched PLL)	`osc1_clk/4`	2.5-12.5 MHz	32	19.5-97.75 KHz	1 (Bypass)	625 KHz-3.125 MHz	Untouched
0x0 and 0x2-0x6	0	10-50 MHz (PLL Bypass)	`osc1_clk/4`	2.5-12.5 MHz	32	19.5-97.75 KHz	1 (Bypass)	625 KHz-3.125 MHz	Bypass
0x7	0	10-15 MHz	`osc1_clk*0.8333`	8.333-12.5 MHz	32	65-97.75 KHz	1 (Bypass)	2.08-3.125 MHz	Locked
0x8	0	15-20 MHz	`osc1_clk*0.625`	9.375-12.5 MHz	32	73-97.75 KHz	1 (Bypass)	2.34-3.125 MHz	Locked
0x9	0	20-25 MHz	`osc1_clk/2`	10-12.5 MHz	32	78.13-97.75 KHz	1 (Bypass)	2.5-3.125 MHz	Locked
0xA	0	25-30 MHz	`osc1_clk*0.4166`	10.4175-12.5 MHz	32	81.25-97.75 KHz	1 (Bypass)	2.60-3.125 MHz	Locked
0xB	0	30-35 MHz	`osc1_clk*0.3571`	10.715-12.5 MHz	32	83.5-97.75 KHz	1 (Bypass)	2.68-3.125 MHz	Locked
0xC	0	35-40 MHz	`osc1_clk*0.3125`	10.9375-12.5 MHz	32	85.25-97.75 KHz	1 (Bypass)	2.75-3.125 MHz	Locked
0xD	0	40-45 MHz	`osc1_clk*0.2777`	11.111-12.5 MHz	32	86.75-97.75 KHz	1 (Bypass)	2.78-3.125 MHz	Locked
0xE	0	45-50 MHz	`osc1_clk/4`	11.25-12.5 MHz	32	87.75-97.75 KHz	1 (Bypass)	2.81-3.125 MHz	Locked

Related Information

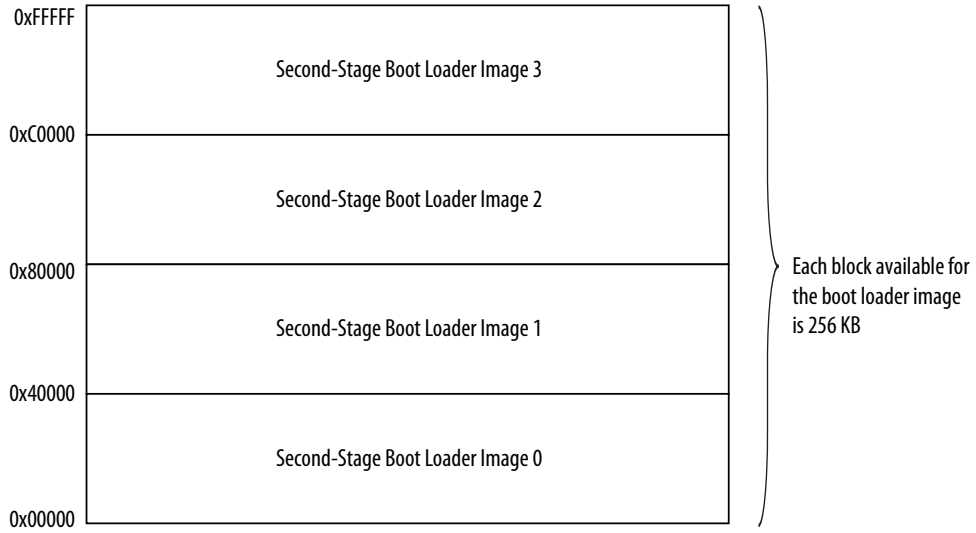
SoC Security on page 168
For more information about CSEL encodings and secure clock configuration, refer to the *SoC Security* chapter.

A.4.4.2. NAND Flash Devices

The NAND subsystem reserves at least the first 1 MB on the NAND device. If the NAND flash device has blocks greater than 256 KB, then the NAND subsystem reserves the first four blocks on the device. For a NAND device with less than 256 KB block size, the second-stage boot loader image must be placed in multiple blocks. The NAND subsystem expects to find up to four second-stage boot loader images on the

NAND device. You may have less than four images if required. The second-stage boot loader image should always be at the start of a physical page. Because a block is the smallest area used for erase operation, any update to a particular image does not affect other images.

Figure 169. NAND Flash Image Layout for 256 KB Memory Blocks



Related Information

NAND Flash Controller on page 370
Refer to the *NAND Flash Controller* chapter for more information regarding features and functionality.

A.4.4.2.1. NAND Flash Driver Features Supported in the Boot ROM Code

Table 319. NAND Flash Support Features

Feature	Driver Support
Device	Open NAND Flash Interface (ONFI) 1.0 raw NAND or electronic signature devices, single layer cell (SLC) and multiple layer cell (MLC)
Chip select	CS0 only. Only CS0 is available to the HPS, the other three chip selects are routed out to the FPGA portion of the device
Bus width	8-bit only
Page size	2 KB, 4 KB, or 8 KB
Page per block	32, 64, 128
ECC	512-bytes with 8-bit correction

A.4.4.2.2. CSEL Settings for the NAND Controller

Table 320. NAND Controller Clock Options Based on CSEL and HPS_CLK fuse settings

Note: The `osc1_clk` signal is sourced from the external oscillator input, HPS_CLK1.

CSEL [3:0] Fuse Values	HPS CLK Fuse Value	Required Input Clock Range	l4_mp_clk	l4_mp_clk Frequency	nand_clk	PLL Status
0x0-0x1 and 0x3-0xF	1	60-200 MHz (Secure bypass)	cb_intosc_hs_clk	60-200 MHz	15-50 MHz	Bypassed
0x2	1	30-100 MHz (Secure PLL)	cb_intosc_ls_clk*2	60-200 MHz	15-50 MHz	Locked
0x1	0	10-50 MHz (Untouched PLL)	osc1_clk	10-50 MHz	2.50-12.5 MHz	Untouched
0x0 and 0x2-0x6	0	10-50 MHz (PLL bypass)	osc1_clk	10-50 MHz	2.50-12.5 MHz	Bypassed
0x7	0	10-15 MHz	osc1_clk*13.33	133.33-200 MHz	33.33-50 MHz	Locked
0x8	0	15-20 MHz	osc1_clk*10	150-200 MHz	37.50-50 MHz	Locked
0x9	0	20-25 MHz	osc1_clk*8	160-200 MHz	40-50 MHz	Locked
0xA	0	25-30 MHz	osc1_clk*6.66	166.66-200 MHz	41.66-50 MHz	Locked
0xB	0	30-35 MHz	osc1_clk*5.71	171.43-200 MHz	42.85-50 MHz	Locked
0xC	0	35-40MHz	osc1_clk*5	175-200 MHz	43.75-50 MHz	Locked
0xD	0	40-45MHz	osc1_clk*4.44	177.78-200 MHz	44.45-50 MHz	Locked
0xE	0	45-50MHz	osc1_clk*4	180-200 MHz	45-50 MHz	Locked

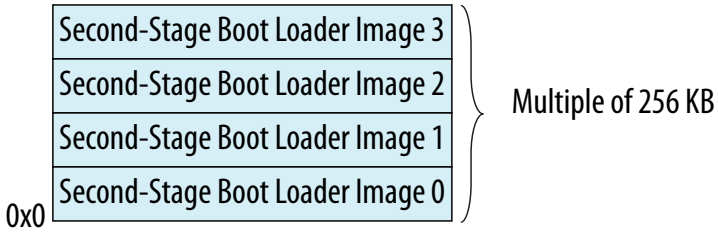
Related Information

SoC Security on page 168
For more information about CSEL encodings and secure clock configuration, refer to the *SoC Security* chapter.

A.4.4.3. Quad SPI Flash Devices

The figure below shows the quad SPI flash image layout. The second-stage boot loader image is always located at offsets that are multiples of 256 KB.

Figure 170. Quad SPI Flash Image Layout



The boot ROM code configures the quad SPI controller to default settings for the supported SPI or quad SPI flash memory.

Related Information

Quad SPI Flash Controller on page 490
Refer to the *Quad SPI Flash Controller* chapter for more information regarding features and functionality.

A.4.4.3.1. Quad SPI Controller Default Settings

Table 321. Quad SPI Controller Default Settings

Parameter	Default Setting	Register Value
SPI baud rate	Divide by 4	The master mode baud rate divisor field (`bauddiv`) of the quad SPI configuration register (`cfg`) in the quad SPI controller registers (`qspiregs`) = 1.
Read opcode	Refer to the "QSPI Controller Clock Options Based on CSEL and HPS_CLK Fuse Settings" table to determine the correct configuration of the HPS_CLK and CSEL fuses for Normal or Fast Reads.	The read opcode in non-XIP mode field (`rdopcode`) in the device read instruction register (`devrd`) = 0x3 (for normal read) and 0xB (for fast read).
Instruction type	Single I/O (1 bit wide)	The address transfer width field (`addrwidth`) and data transfer width field (`datawidth`) of the `devrd` register = 0.
Number of address bytes	3 bytes	The number of address bytes field (`numaddrbytes`) of the device size register (`devsz`) = 2. *Note:* Before a reset, you must ensure that the QSPI flash device is configured to 3 byte address mode for the boot ROM to function properly.
Delay in terms of `l4_main_clk` for the length that the master mode chip select outputs are deasserted between words when the clock phase is zero	The default setting in clock cycles must equal 200 ns	The clock delay for chip select deassert (field `nss` in the quad SPI device delay register (`delay`)). Refer to `delay[31:24]` in the "Quad SPI Flash Delay Configuration" table in the "Quad SPI Flash Delay Configuration" section for calculations.

continued...

Parameter	Default Setting	Register Value
Delay in terms of l4_main_clk clock cycles between one chip select being deactivated and the activation of another. This delay ensures a quiet period between the selection of two different slaves and requires the transmit FIFO to be empty	The default setting is 0 clock cycles.	The clock delay for chip select deactivation (field btwn in the delay register = 0x0).
Delay in terms of l4_main_clk clock cycles between the last bit of the current transaction and the first bit of the next transaction. If the clock phase is zero, the first bit of the next transaction refers to the cycle in which the chip select is deselected	The default setting in clock cycles must equal 20 ns.	The clock delay for last transaction bit (field after in the delay register). Refer to delay[15:8] in the "Quad SPI Flash Delay Configuration" table in the "Quad SPI Flash Delay Configuration" section for calculations.
Added delay in terms of l4_main_clk clock cycles between setting qspi_n_ss_out low and first bit transfer	The default setting in clock cycles must equal 20 ns.	The clock delay of qspi_n_ss_out (field init in the delay register). Refer to delay[7:0] in the "Quad SPI Flash Delay Configuration" table in the "Quad SPI Flash Delay Configuration" section for calculations.

A.4.4.3.2. Quad SPI Flash Delay Configuration

The delay register in the quad SPI controller configures relative delay of the generation of the master output signals. All timings are defined in cycles of l4_main_clk.

The quad SPI flash memory must meet the following timing requirements:

- T_{SLCH}: 20 ns
  - T_{SLCH} is used to calculate the init field (delay[7:0]) in the delay register. The init field represents the delay in l4_main_clk clocks between pulling the device chip select (qspi_n_ss_out) low and the first bit transfer.
- T_{CHSH}: 20 ns
  - T_{CHSH} is used to calculate the after field (delay[15:8]) in the delay register. The after field represents the delay in the l4_main_clk clocks between last bit of the current transaction and the deassertion of the device chip select (qspi_n_ss_out).
- T_{SHSL}: 200 ns
  - T_{SHSL} is used to calculate the nss (delay[31:24]) field in the delay register and is the delay in the l4_main_clk clocks for the length that the master mode chip select outputs are deasserted between transactions.
- $T_{qspi_ref_clk}$ is the master reference clock/external clock, l4_main_clk.

The formulas to calculate the fields in the delay register are:

delay[7:0]= init = $T_{SLCH}/T_{qspi_ref_clk}$

delay[15:8]= after = $T_{CHSH}/T_{qspi_ref_clk}$

delay[31:24]= nss = $(T_{SHSL} - T_{qspi_clk})/T_{qspi_ref_clk}$

Table 322. Quad SPI Flash Delay Configuration for Device Delay (delay) Register

CSEL [3:0] Fuse Values	HPS CLK Fuse Value	Input Clock Range	$T_{qspi_ref_clk}$ (ns)	T_{qspi_clk} (ns)	delay[7:0] (init)	delay[15:8] (after)	delay[31:24] (nss)
0x0-0x1 and 0x3-0xF	1	60-200 MHz (Secure Bypass)	5	20	4	4	36
0x2	1	30-100 MHz (Secure PLL)	2.5	10	8	8	76
0x1	0	10-50 MHz (Untouched PLL)	20	80	1	1	6
0x0 and 0x2-0x6, 0xF	0	10-50 MHz (PLL Bypass)	20	80	1	1	6
0x7	0	10-15 MHz	2.5	10	8	8	76
0x8	0	15-20 MHz	2.5	10	8	8	76
0x9	0	20-25 MHz	2.5	10	8	8	76
0xA	0	25-30 MHz	2.5	10	8	8	76
0xB	0	30-35 MHz	2.5	10	8	8	76
0xC	0	35-40 MHz	2.5	10	8	8	76
0xD	0	40-45 MHz	2.5	10	8	8	76
0xE	0	45-50 MHz	2.5	10	8	8	76

Read data capture delay is also configured when booting from QSPI. Depending on the CSEL and HPS_CLK fuse values, the Boot ROM configures the `delay` field of the `rddatacap` register in the QSPI differently. When you configure the CSEL and HPS_CLK fuses to enable the PLL, the boot ROM calibrates the interface by reading the QSPI signal for all delay values of the `rddatacap` register. The Boot ROM analyzes all of the delay values that return a valid signature and uses the delay value in the middle of the valid window as the value it programs into the `delay` field of the `rddatacap` register. If the PLL is not enabled, then the Boot ROM leaves the `rddatacap` register untouched.

A.4.4.3.3. Quad SPI Controller CSEL Settings

Table 323. QSPI Controller Clock Options Based on CSEL and HPS_CLK Fuse Settings

Note: The `osc1_clk` signal is sourced from the external oscillator input, HPS_CLK1.

CSEL [3:0] Fuse Values	HPS CLK Fuse Value	Required Input Clock Range	Controller clock	Controller Clock Frequency (`l4_main_clock`)	Baud Rate Divisor	External Device Clock (Divided down reference clock)	Default Flash Read Instruction	PLL Status
0x0-0x1 and 0x3-0xF	1	60-200 MHz (Secure Bypass)	`cb_intosc_hs_clk`	60-200 MHz	16	3.75-12 MHz	READ	Bypassed
0x2	1	30-100 MHz (Secure PLL)	`cb_intosc_ls_clk*4`	120-400 MHz	8	15-50 MHz	FAST_READ	Locked
0x1	0	10-50 MHz (Untouched PLL)	`osc1_clk`	10-50 MHz	4	2.50-12.5 MHz	READ	Untouched
0x0-0x6, 0xF	0	10-50 MHz (PLL Bypass)	`osc1_clk`	10-50 MHz	4	2.50-12.5 MHz	READ	Bypassed
0x7	0	10-15 MHz	`osc1_clk*26.75`	266.67-400 MHz	8	33.33-50 MHz	FAST_READ	Locked
0x8	0	15-20 MHz	`osc1_clk*20`	300-400 MHz	8	37.50-50 MHz	FAST_READ	Locked
0x9	0	20-25 MHz	`osc1_clk*16`	320-400 MHz	8	40-50 MHz	FAST_READ	Locked
0xA	0	25-30 MHz	`osc1_clk*13.33`	333.33-400 MHz	8	41.66-50 MHz	FAST_READ	Locked
0xB	0	30-35 MHz	`osc1_clk*11.42`	42.85-50 MHz	8	42.85-50 MHz	FAST_READ	Locked
0xC	0	35-40 MHz	`osc1_clk*10`	350-400 MHz	8	43.75-50 MHz	FAST_READ	Locked
0xD	0	40-45 MHz	`osc1_clk*8.88`	355.55-400 MHz	8	44.45-50 MHz	FAST_READ	Locked
0xE	0	45-50 MHz	`osc1_clk*3`	360-400 MHz	8	45-50 MHz	FAST_READ	Locked

Related Information

SoC Security on page 168
For more information about CSEL encodings and secure clock configuration, refer to the *SoC Security* chapter.

A.4.5. Clock Select

The boot ROM reads the clock select values to determine what frequency has been selected for the CPU clock and any interface clock during boot.

If the FPGA boot fuse is not blown, the clock select (CSEL) fuses are used to configure the main PLL and peripheral PLL. The Clock Manager samples the clock configuration on a cold and warm reset. If the hps_clk_f fuse is blown, the internal oscillator divided

by 2, `cb_intosc_hs_div2_clk`, is used as the boot clock. If the fuse is not blown, an external oscillator is used. During boot ROM execution, the boot code configures the device clock based on the CSEL fuse settings or software code. The `cb_intosc_hs_div2_clk` can be considered the secure reference clock option.

Note: The terms CSEL and CLKSEL are used interchangeably in Intel documentation to refer to clock select.

The CSEL settings are not used to configure the PLLs for the following conditions:

- A boot from RAM (warm reset)
- A boot from FPGA
- The clock select is set to 0x1, indicating the clocks should not be touched.

If a bypass condition is encountered, the boot ROM checks to make sure that the boot clock has a proper source according to the security level set and if it does not, the boot ROM stalls. ROM code bypasses the PLLs at warm or cold reset under the following conditions:

- The hps_clk_f fuse is not blown and the CSEL fuse value is 0xF. This setting is an unsecure bypass that uses the external oscillator (HPS_CLK1) as the clock source.
- The hps_clk_f fuse is not blown and the CSEL fuses values are not 0xF, 0x1 or one of the values from 0x7 to 0xE that indicates PLL use. These are mapped to unsecure bypass using the external oscillator (HPS_CLK1) as the clock source.
- The hps_clk fuse is set to secure clock, but the CSEL fuse value is not 0x2. This setting is considered secure bypass, using the internal oscillator as the clock source.

CSEL Fuse Encodings During Boot

The intent of the CSEL fuses is to give the user more control over the CPU clock in both non-secure and secure mode. For example, if you select CSEL=0x0 (the default) in non-secure mode, then the boot ROM configures all the clocks back to their default bypass (boot) mode and the MPU is clocked by the HPS_CLK1 (10-50 MHz). However, if you have a 10 MHz external oscillator (HPS_CLK1) and want to operate faster, then the CSEL fuses can be programmed to 0x7 for a MPU clock of 553 MHz. Similarly, if the input clock is 25 MHz and the CSEL fuses are programmed to 0x9 then the MPU clock is 800 MHz (25 * 32). If CSEL=0xA is chosen, then the same clock gives a MPU clock of 650 MHz (25 * 26). Finally, if some set of CSEL fuses have been blown, you can return to the default bypass state (CSEL=0x0) by blowing all the fuses (CSEL=0xF).

The following tables represent the clock frequencies that are available during the boot process according the boot clock that is selected. As noted previously, when the internal secure clock, `cb_intosc_ls_clk`, is used as the boot clock, the meaning of the CSEL fuses is slightly different.

To configure the clock speeds for second-stage boot loader execution, you need to select the clock speeds in the HPS component of Platform Designer (Standard). The tables below only apply to the boot process.

Table 324. CSEL Encodings for hps_clk_f = 0 (Non-secure Operation)

CSEL[3:0] Fuse Value	Description	MPU Clock Value
0x0	When no fuses are blown, the boot ROM returns clocks back to their default (bypass) boot mode and the CPU is driven by the external oscillator (HPS_CLK1), which must be in the range of 10 to 50 MHz. No PLL is enabled.	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x1	All the clock codes are bypassed and the clock source is user selected.	User-selected clock source
0x2	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x3	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x4	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x5	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x6	Reserved	External Oscillator, HPS_CLK1 (10 to 50 MHz)
0x7	External oscillator input clock, HPS_CLK1, is in the range of 10 to 15 MHz	533 to 800 MHz
0x8	External oscillator input clock, HPS_CLK1, is in the range 15 to 20 MHz.	600 to 800 Mhz
0x9	External oscillator input clock, HPS_CLK1, is in the range 20 to 25 MHz.	640 to 800 Mhz
0xA	External oscillator input clock, HPS_CLK1, is in the range 25 to 30 MHz.	666 to 800 Mhz
0xB	External oscillator input clock, HPS_CLK1, is in the range 30 to 35 MHz.	685 to 800 Mhz
0xC	External oscillator input clock, HPS_CLK1, is in the range 35 to 40 MHz.	700 to 800 Mhz
0xD	External oscillator input clock, HPS_CLK1, is in the range 40 to 45 MHz.	711 to 800 Mhz
0xE	External oscillator input clock, HPS_CLK1, is in the range 45 to 50 MHz.	720 to 800 Mhz
0xF	The boot ROM returns clocks back to their default (bypass) boot mode and the CPU is driven by the external oscillator (HPS_CLK1). No PLL is enabled.	External Oscillator, HPS_CLK1 (10 to 50 MHz)

Table 325. CSEL Encodings for hps_clk_f = 1 (Secure Operation)

CSEL[3:0] Fuse Value	Description	MPU Clock Value
0x0	Bypass mode; in this mode all clocks are reset and boot mode is ensured active; `cb_intosc_hs_div2_clk` (`cb_intosc_hs` clock divided by 2) is used.	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x1	All the clock codes are bypassed and the clock is user selected.	User-selected clock source
0x2	PLL is used with the internal oscillator (30 to 100 MHz)	120 to 800 MHz
0x3	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x4	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x5	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x6	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x7	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x8	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0x9	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0xA	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0xB	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0xC	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0xD	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0xE	Reserved	`cb_intosc_hs_div2_clk` (60 to 200 MHz)
0xF	Bypass mode; reset all clocks and ensure boot mode is active; `cb_intosc_hs_div2_clk` is used.	`cb_intosc_hs_div2_clk` (60 to 200 MHz)

Related Information

SoC Security on page 168
For more information about CSEL encodings and secure clock configuration, refer to the *SoC Security* chapter.

A.4.6. I/O Configuration

The flash devices needed for booting must be connected to specific I/Os. The Boot ROM configures these I/Os depending on the flash device selected by boot select setting. To configure the I/Os, the boot ROM performs pin muxing and pin configuration on these I/Os.

The boot ROM is allocated 14 dedicated HPS I/Os. There are three separate I/O tables in the boot ROM, one for each flash device.

On cold reset, the boot ROM always configures the pin muxes and pin configuration. On warm reset, the user has the capability to specify whether or not the boot ROM code configures the I/Os and pin muxes for boot pins after a warm reset. This configuration on warm reset is enabled by programming the `romcode_ctrl` register in the System Manager.

Refer to the "Boot Source I/O Pins" section for detail on mux select values.

Related Information

Boot Source I/O Pins on page 778

A.4.6.1. I/O State

The state of the device I/O pins changes depending on the current operation mode of the device. The table below describes the changes in state per I/O type for each operating mode of the device.

Table 326. I/O State

I/O Type	Operation Modes from the time the FPGA is Unconfigured (POR) to User Mode				Operation Modes after the FPGA is Configured		
	Power up	HPS Boot ROM Mode	Early I/O Release Mode	User Mode	HPS Cold Reset	FPGA Partial Reconfiguration	FPGA Full Reconfiguration
Dedicated HPS I/O used during Boot	Input tri-state	Select I/O are configured by Boot ROM; unused I/O remain as input tri-state.	Select I/O are configured by software. Unused I/O remain as input tri-state.	Select I/O are configured by software. Unused I/O remain as input tri-state.	Input tri-state	Configured I/O continue to operate. Unused I/O remain as input tri-state.	Configured I/O continue to operate. Unused I/O remain as input tri-state.
HPS Shared I/O	Input tri-state[62]	Input tri-state unless FPGA is being configured.	Select I/O are configured by configuration bitstream. Unused I/O remain as input tri-state.	Select I/O are configured by configuration bitstream. Unused I/O remain as input tri-state.	Pin multiplexer is reset and thus I/O pins can be input tri-state.	Configured I/O continue to operate. Unused I/O remain as input tri-state.	When full reconfiguration begins, I/O state reverts to input tristate. Once configuration is complete, unused I/O remain as input-tristate while selected I/O are configured by bitstream.

continued...

[62] Depending on the `nIO_PULLUP` pin value, the weak pull up of the pins may be enabled.

I/O Type	Operation Modes from the time the FPGA is Unconfigured (POR) to User Mode				Operation Modes after the FPGA is Configured		
	Power up	HPS Boot ROM Mode	Early I/O Release Mode	User Mode	HPS Cold Reset	FPGA Partial Reconfiguration	FPGA Full Reconfiguration
							Note: In this mode, any HPS peripheral connected to the shared I/O loses connectivity until configuration is complete.
HPS DDR I/O	Input tri-state(62)	Input tri-state unless FPGA is being configured.	Select I/O are configured by configuration bitstream. Unused I/O remain as input tri-state.	Select I/O are configured by configuration bitstream. Unused I/O remain as input tri-state.	Configured I/O continue to operate.	Configured I/O continue to operate. Unused I/O remain as input tri-state.	When full reconfiguration begins, I/O state reverts to input tristate. Once configuration is complete, unused I/O remain as input-tristate while selected I/O are configured by bitstream. *Note:* In this mode HPS loses connectivity to HPS DDR and may need to be reset to reboot the system.
FPGA I/O	Input tri-state(62)	Input tri-state unless FPGA is being configured.	Select I/O are configured by configuration bitstream. Unused I/O remain as input tri-state.	Select I/O are configured by configuration bitstream. Unused I/O remain as input tri-state.	Configured I/O continue to operate.	Configured I/O continue to operate. Unused I/O remain as input tri-state.	When full reconfiguration begins, I/O state reverts to input tristate. Once configuration is complete, unused I/O remain as input-tristate while selected I/O are configured by bitstream.

A.4.7. L4 Watchdog 0 Timer

The boot ROM enables the L4 Watchdog 0 Timer early in the boot process.

This watchdog is reserved for the boot ROM until the second-stage boot loader indicates that it has started correctly and taken control of the exception vectors. The timeout is at least one second, depending on the clock select setting. Because the watchdog is reset just before the control passes to second-stage boot loader, the second-stage boot loader must reset the watchdog when it begins execution.

The L4 watchdog 0 timer is reserved for boot ROM use. While booting, if a watchdog reset happens before software control passes to the second-stage boot loader, the boot ROM code attempts to load the last valid second-stage boot loader image, identified by the `initswlastld` register in the System Manager.

If the watchdog reset happens after the second-stage boot loader has started executing but before it writes a valid value to `initswstate` register, the boot ROM increments `initswlastld` and attempts to load that image. If the watchdog reset happens after the second-stage boot loader writes a valid value to `initswstate` register, the boot ROM code attempts to load the image indicated by `initswlastld` register.

A.4.8. Second-Stage Boot Loader

The function of the second-stage boot loader is user-defined. The Intel-provided second-stage boot loader is a combination of initialization, configuration and U-Boot code and contains features such as:

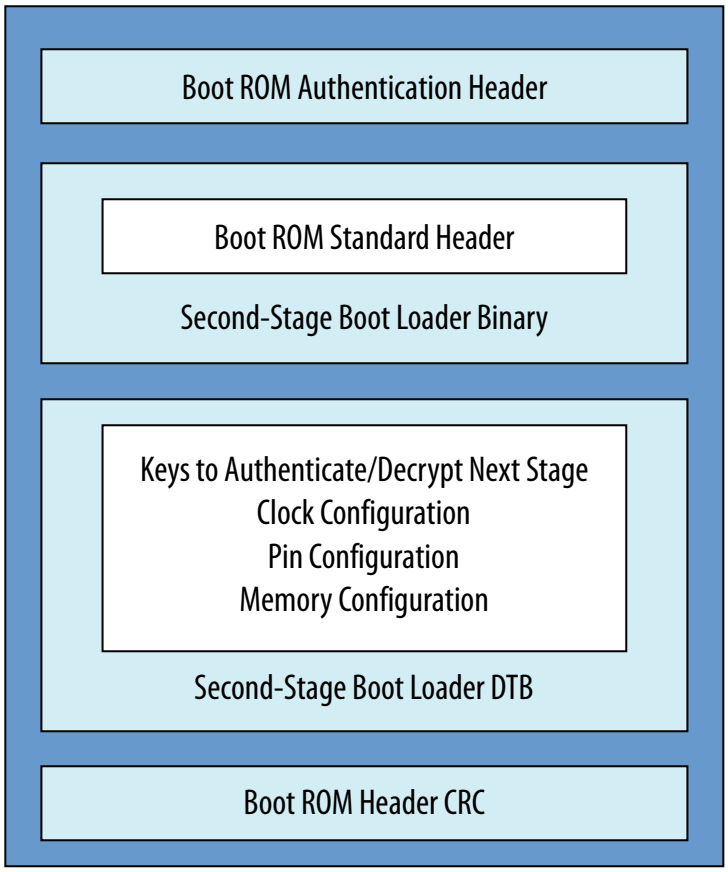
- SD/MMC controller driver
- QSPI controller driver
- Ethernet driver plus protocol support
- Drivers for system-level IP, such as Clock Manager, System Manager, and FPGA Manager
- Cache memory drivers
- UART, timer and watchdog drivers
- FAT file system support
- Flat Image Tree (FIT) image processing
- U-Boot console support including basic essential debug commands
- Cryptographic library
- Flattened device tree (FDT) processing library
- System and memory firewall configuration
- Initialization code for the HPS and FPGA I/O and the interface that loads the next stage of software

If a secure boot is required, the second-stage boot loader can be used to increase the level of security and to authenticate and initiate decryption of the next boot image if necessary.

If the hard memory controller I/O in the SoC device have been configured through the FPGA or HPS and the SDRAM firewall access has been lowered, the second-stage boot loader can be used to load the next stage of the boot software into SDRAM. The maximum length for a second-stage boot loader to fit into on-chip RAM is 208 KB with authentication and 224 KB without authentication. A typical next software stage is loading the application OS software. The second-stage boot loader is allowed to load the next stage boot software from any device available to the HPS. Typical sources include the same flash device that contains the second stage boot loader, a different flash device, or a communication interface such as an EMAC.

If the second-stage boot loader must be authenticated, it must store a public key. Below is a figure which depicts the second-stage boot loader image presented to the boot ROM, during a secure, authenticated boot.

Figure 171. High-Level Diagram of Second-Stage Boot Loader Image



A.4.9. Secure Boot

The boot ROM supports both non-secure and secure boot.

Secure boot allows the HPS to release from reset into a known state and then validate the authenticity and integrity of the second-stage boot loader code prior to executing it. Secure boot can ensure that only authorized code is executed on the system. In addition, the system has the option to configure the FPGA from the HPS, which provides a secure boot mechanism for the FPGA portion of the SoC. In this mode, the HPS boot code can authenticate the FPGA POF prior to loading it.

The boot ROM determines the security level based on user fuse values that are stored in the Security Manager during initialization. The second-stage boot loader may provide a security header to indicate authentication or decryption for the image. Based on the merged data from security manager and the security header, the boot ROM can load the following images:

- Clear text: No authentication or decryption is required
- Secure, Authenticated - Authentication but no decryption is required on the image
- Secure, Encrypted - Decryption, but no authentication is required. During the decryption process, the boot ROM looks for an encrypted image on the flash device and decrypts it into the on-chip RAM
- Full Secure - Both authentication and decryption are required

The authentication process is independent of the decryption process. However, if authentication and decryption are both required, authentication is done before the decryption process.

For more information regarding Secure Boot, please refer to the *SoC Security* Chapter.

Related Information

SoC Security on page 168
For more information about secure boot, refer to the *SoC Security* chapter.

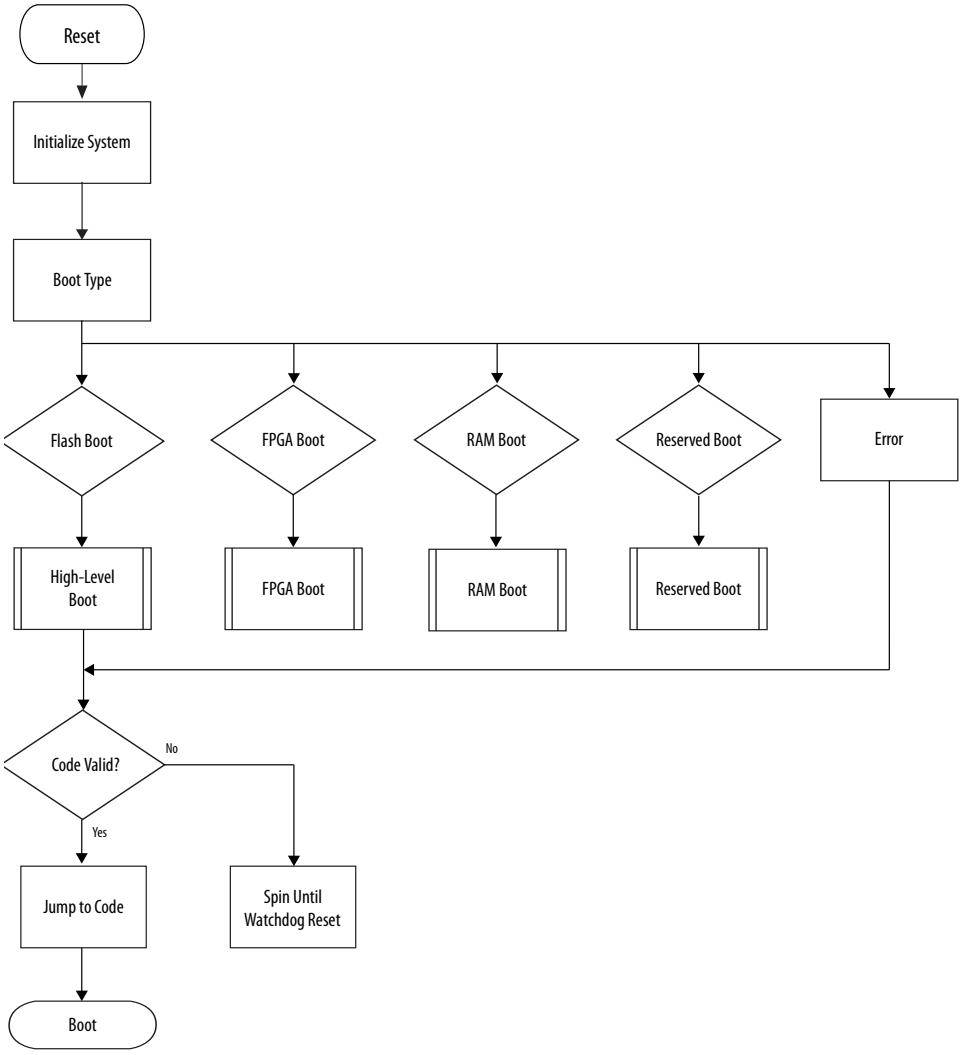
A.5. Boot ROM Flow

On a cold reset, the HPS boot process starts when CPU0 is released from reset (for example, on a power up) and executes code in the internal boot ROM at the reset exception address, 0x00000000. The boot ROM code brings the HPS out of reset and into a known state. After boot ROM code is exited, control passes to the next stage of the boot software, referred to as the second-stage boot loader. The second-stage boot loader can be customized and is typically stored external to the HPS in a nonvolatile flash-based memory or in on-chip RAM within the FPGA. The second-stage boot loader can then load an OS, BareMetal application or potentially a third-stage boot loader.

This section describes the software flow from reset until the boot ROM code passes software control to the second-stage boot loader.

The code starts, initializes the system and then depending on the type of boot requested, it may attempt to load the code into the on-chip RAM. An on-chip RAM boot can only occur on a warm reset. Warm boot from on-chip RAM has the highest priority to execute if the `warmram_*` registers in the system manager has been configured to support booting from on-chip RAM on a warm reset. If the `warmram_enable` register in the system manager is set to 0xAE9EFEBC, then the boot ROM code attempts to boot from on-chip RAM on a warm reset and the boot ROM does not configure boot I/Os, pin-muxes or clocks. The `warmram_datastart` and `warmram_length` registers in System Manager allow you to program the offset of the beginning of code and the length of the region of on-chip RAM for CRC validation. If the `warmram_length` register is clear, then the boot ROM does not perform a CRC calculation in the on-chip RAM. If the `warmram_length` register is non-zero, then CRC checking is performed. If CRC checking is successful, then the boot ROM jumps to global address programmed in the `warmram_execution` register. If for three subsequent load attempts, the boot ROM fails to find code or the CRC check fails, the boot ROM spins, waiting for a watchdog reset.

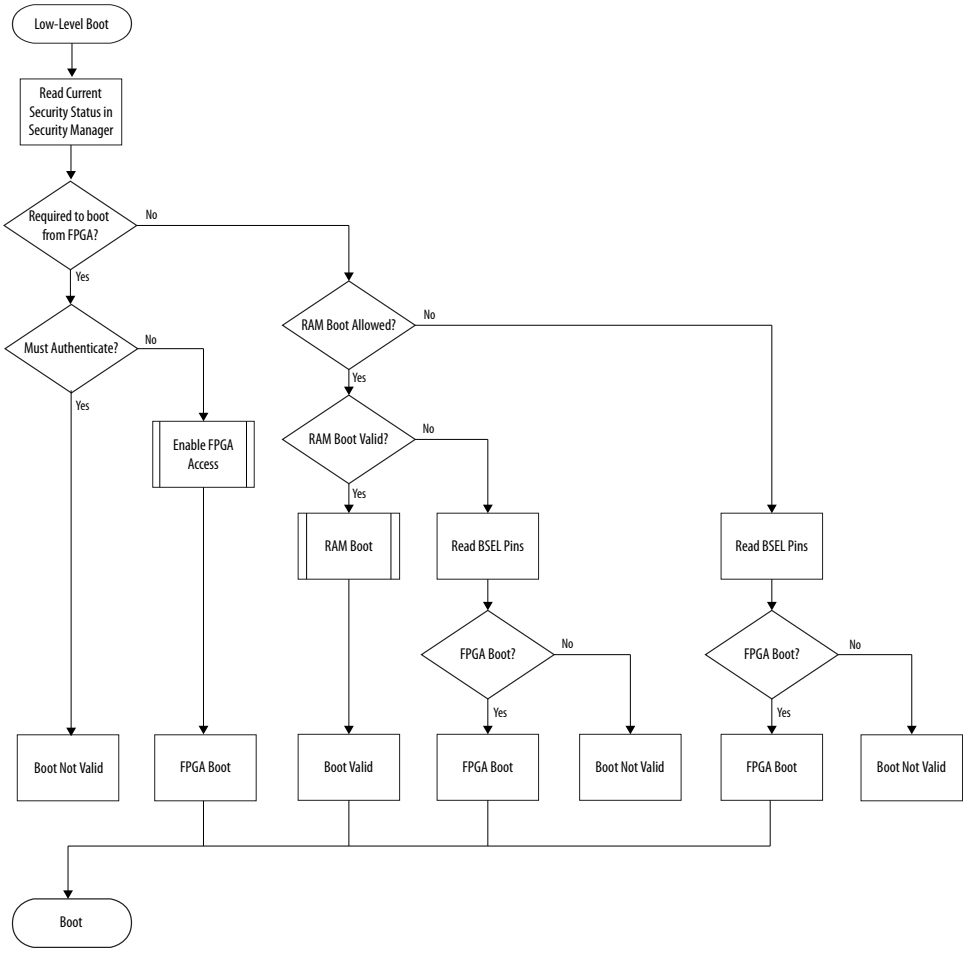
Figure 172. Main Boot ROM Flow



The boot ROM always executes on CPU0. CPU1 is always held in reset while the main boot ROM code is executing and is only released when required by system software.

As part of determining the boot type, the boot ROM executes a low-level boot flow. The boot ROM code reads the security fuse to determine if the source of the second-stage boot is forced to be the FPGA. If a non-authenticated FPGA boot or a non-CRC on-chip RAM boot is requested or the boot is invalid, it is processed within the low-level boot flow. All other boot types are processed within the high-level boot flow.

Figure 173. Low-Level Boot Flow

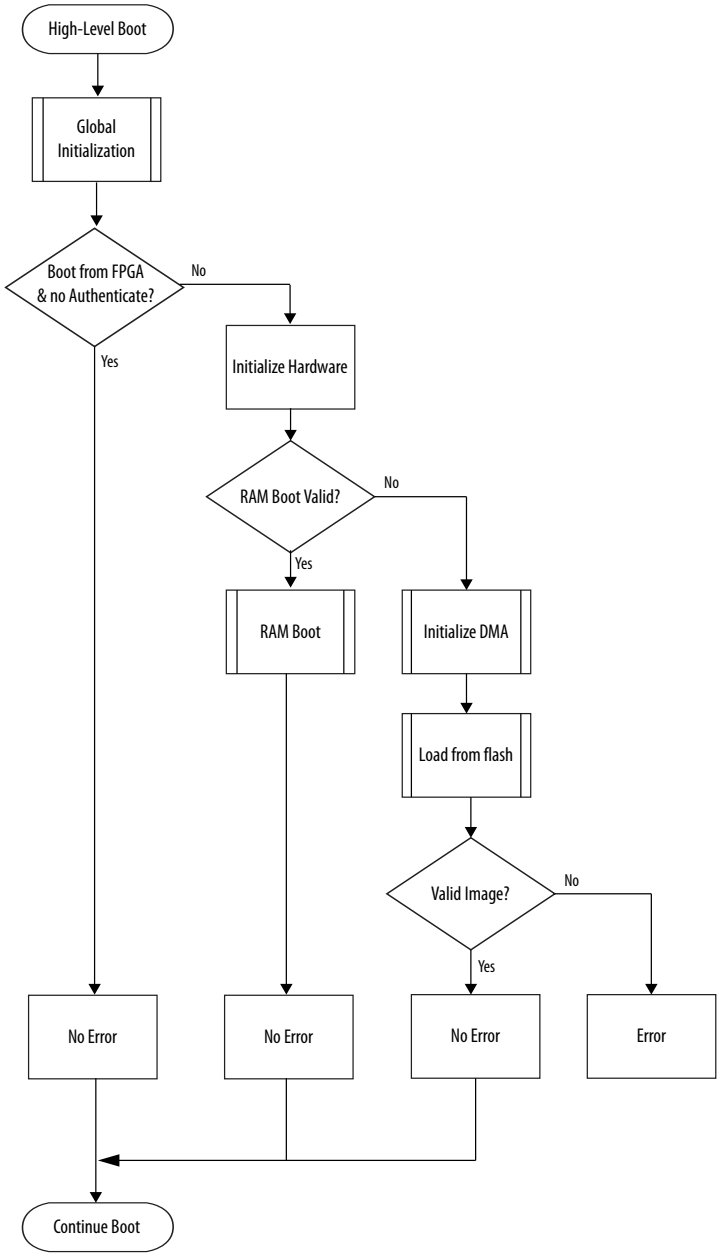


During the low-level portion of the boot ROM flow, the boot ROM reads the security fuses to determine if an FPGA-only boot is required. If so, then the boot ROM must also determine if the fuses indicate that authentication of the POF is needed. If no authentication is required, then a standard FPGA configuration occurs.

If an FPGA-only boot is not required, then the boot ROM checks if an on-chip RAM boot is allowed. If it is, then the boot ROM checks to see if the code is valid. If the code is invalid, the boot ROM reads the BSEL pins to determine if it indicated an FPGA boot.

If the secure fuses indicate that authentication is required for the boot image, then a high-level boot (executed in C code) must be performed.

Figure 174. High-Level Boot Flow



During the boot process, authentication and decryption can be performed on the boot image. Authentication is independent of decryption; however, if both authentication and decryption are required, then authentication always occurs first. If an authenticated boot is required, then the boot ROM must have a root key to start the authentication process. This key can be implemented in the user fuses, in the FPGA logic elements or as part of the second-stage boot image header. The device configuration fuses determine the source of the key.

During a cold boot from flash memory, the boot ROM code attempts to load the first second-stage boot loader image from flash memory to on-chip RAM and pass control to the second-stage boot loader. If this initial image is invalid, the boot ROM code indexes the `romcode_initswlastld` register and attempts to load the next stored image. The boot ROM attempts three subsequent loads after the initial one. If there is still no valid image found after the subsequent loads, the boot ROM code checks the FPGA portion of the device for a fallback image.

Note: During the boot process, the boot ROM enables all of the caches (L1 data and instruction caches and L2 cache). If the second-stage boot loader is not loaded from a boot flash device (SD/MMC, QSPI, NAND) properly, the caches may be left on when the boot ROM checks the FPGA portion the device for a fallback image. This situation can lead to issues of coherency when loading code, so caches must be flushed and disabled in the fallback image.

If the warm RAM boot has failed or if a cold reset has occurred, then the boot ROM reads the BSEL value in the `bootinfo` register of the System Manager. If the FPGA is selected as the boot source, then the boot ROM code attempts to execute code at address 0xC0000000 across the HPS-to-FPGA bridge (offset 0x00000000 from bridge). No error conditions are generated if the FPGA does not initialize properly and the watchdog is not enabled for time-out. Instead, the boot ROM continues to wait until the FPGA is available.

If the BSEL bits indicate a boot from external flash, then the boot ROM code attempts to load an image from a flash device into the on-chip RAM, verify and execute it. If the BSEL is invalid or the boot ROM code cannot find a valid image in the flash, then the boot ROM code checks if there is a fallback image in the FPGA. If there is, then the boot ROM executes the fallback image. If there is no fallback image then the boot ROM performs a post-mortem dump of information into the on-chip RAM and awaits a reset.

Note: The acronyms BSEL and BOOTSEL are used interchangeably to define the boot select pins.

The boot ROM code verifies the second-stage boot loader in several ways to ensure a corrupted image is not executed. The first test is of the image header, which identifies the magic number, version, block length, and CRC of the image that protects the block. If any of these are invalid, an error occurs.

A.6. Second-Stage Boot Flow

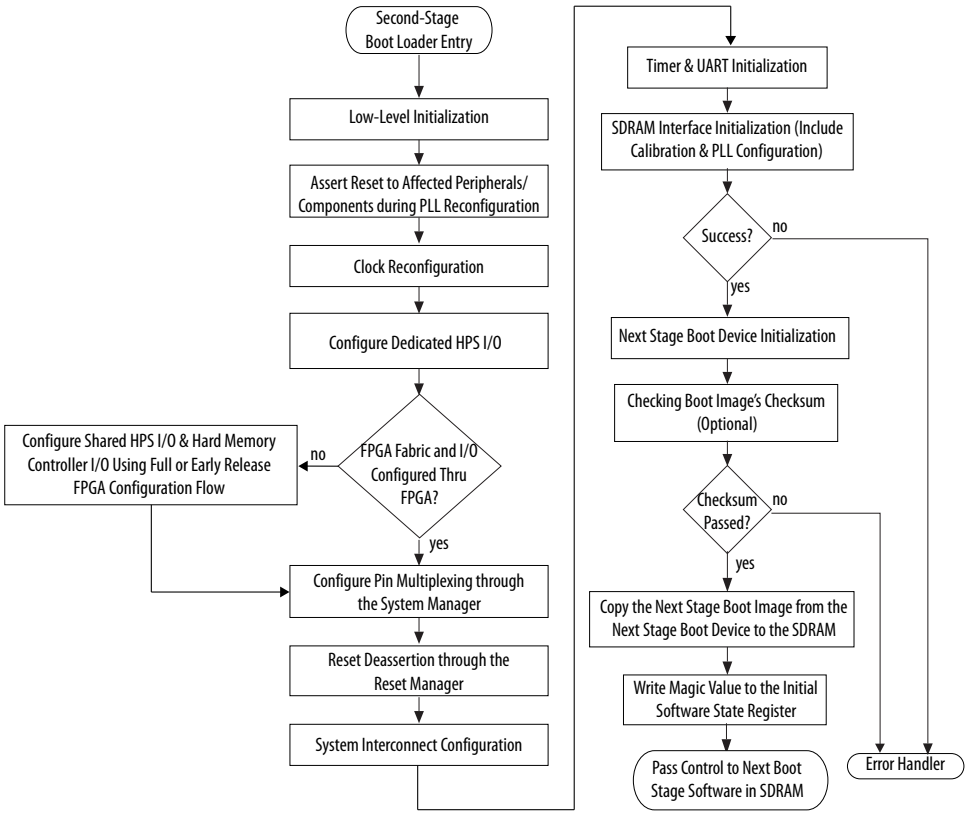
The second stage boot loader has the capability of supporting the following types of boot:

- Non-secure clear text boot
- Secure boot with:
  - Authentication only (also called verified boot)
  - Decryption only
  - Authentication and decryption

A.6.1. Typical Boot Flow (Non-Secure)

A non-secure second-stage boot process typically follows a flow as in the following figure.

Figure 175. Typical Second-Stage Boot Loader Flow (Non-Secure)



Low-level initialization steps include reconfiguring or disabling the L4 watchdog 0 timer, invalidating the instruction cache and branch predictor, remapping the on-chip RAM to the lowest memory region, and setting up the data area.

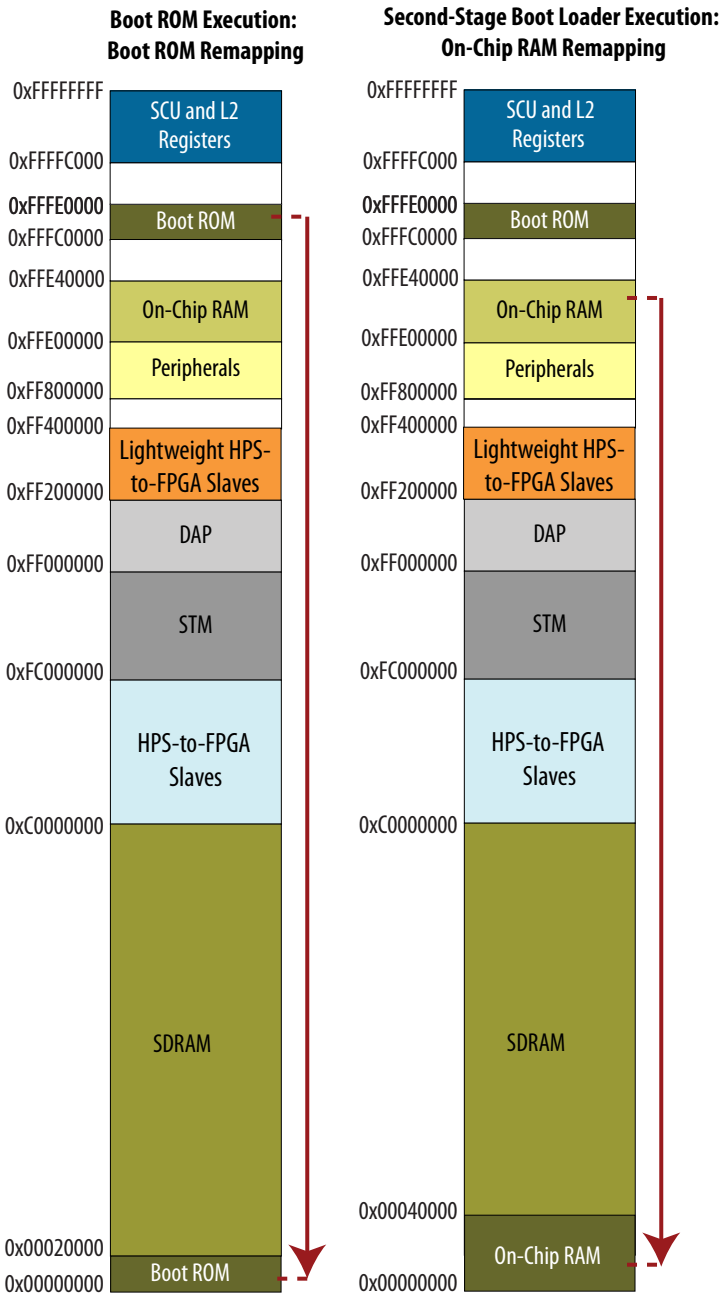
Upon entering the second-stage boot loader, the L4 watchdog 0 timer is active. The second stage boot loader can disable, reconfigure, or leave the watchdog timer unchanged. Once enabled after reset, the watchdog timer cannot be disabled, only paused or reset.

The instruction cache and branch predictor, which were previously enabled by the boot ROM code, must be invalidated. If the second-stage boot loader enables and uses the data cache, it must initialize all levels of the data cache before enabling.

The second-stage boot loader must remap the exception vector table because the exception vectors are still pointing to the exception handler in the boot ROM when it starts executing. By setting the system interconnect remap bit 0 to one, the on-chip RAM mirrors to the lowest region of the memory map. After this remap, the exception vectors use the exception handlers in the boot loader image.

The figure below shows the memory map before and after remap. The first mapping depicts boot ROM execution. The second mapping shows the remap of the on-chip RAM when the remap bit is set during second-stage boot loader execution.

Figure 176. Remapping the On-Chip RAM



The second-stage boot loader can reconfigure all HPS clocks. During clock reconfiguration, the boot loader asserts reset to the peripherals in the HPS affected by the clock changes.

The I/O assignments for the HPS are configured as part of the IOCSR configuration in the second-stage boot loader. Effectively, a bitstream containing the I/O assignments is sent to the device as part of the initialization code in the second-stage boot loader. If the FPGA fabric and I/O have not yet been configured through the FPGA and the HPS needs to access SDRAM, you should program the HPS to use the full or early I/O release configuration method to configure the shared and hard memory controller I/O. Refer to "FPGA Configuration" section of the "Booting and Configuration" appendix in the *Arria 10 Hard Processor System Technical Reference Manual* for details on full and early I/O release configuration.

The second-stage boot loader looks for a valid next-stage boot image in the next-stage boot device by checking the boot image validation data and checksum in the mirror image. Once validated, the second-stage boot loader copies the next-stage boot image (OS or application image) from the next-stage boot device to the SDRAM.

Before software passes control to the next-stage boot software, the second-stage boot loader can write a valid value (0x49535756) to the `romcode_initswstate` register in the System Manager. This value indicates that there is a valid boot image in the on-chip RAM. The `romcode_initiswlastld` register holds the index of the last second-stage boot loader software image loaded by the Boot ROM from the boot device. When a warm reset occurs, the Boot ROM loads the image indicated by the `romcode_initswlastld` register if the BSEL value is the same as the last boot.

Related Information

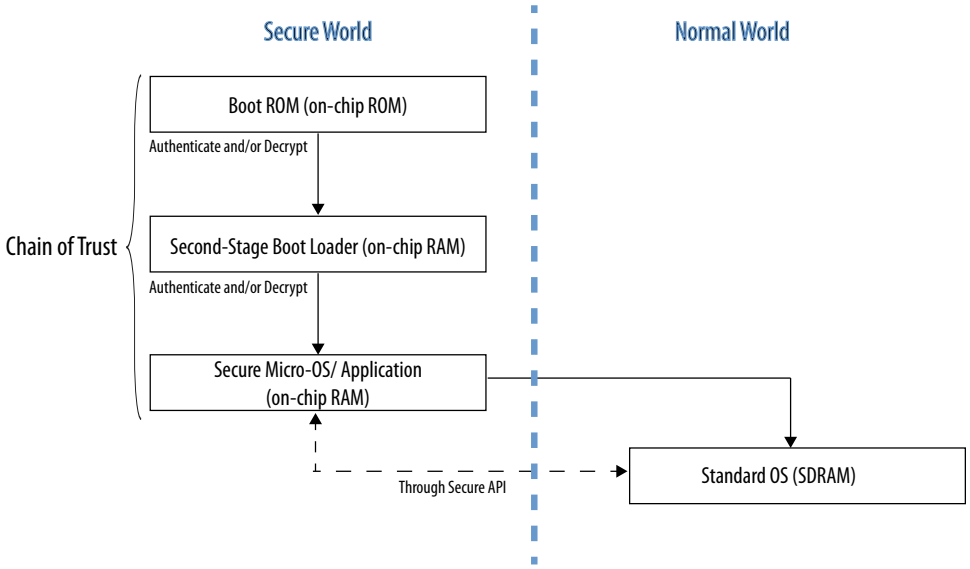
- FPGA Configuration on page 813
  For more information about full and early I/O release configuration methods
- Arria 10 SoC Boot User Guide
  For more information on the steps for booting the Arria 10 SoC

A.6.2. Secure Boot Flow

The main purpose of secure boot is to pass the chain of trust to the subsequent boot software. During a secure boot, the second-stage boot loader may authenticate or decrypt the subsequent boot image, depending on the current state registers in the

Security Manager. In addition, the second-stage boot loader must ensure that the subsequent boot image is executed from secure memory such as on-chip RAM. The second-stage boot loader fits into the chain of trust as such:

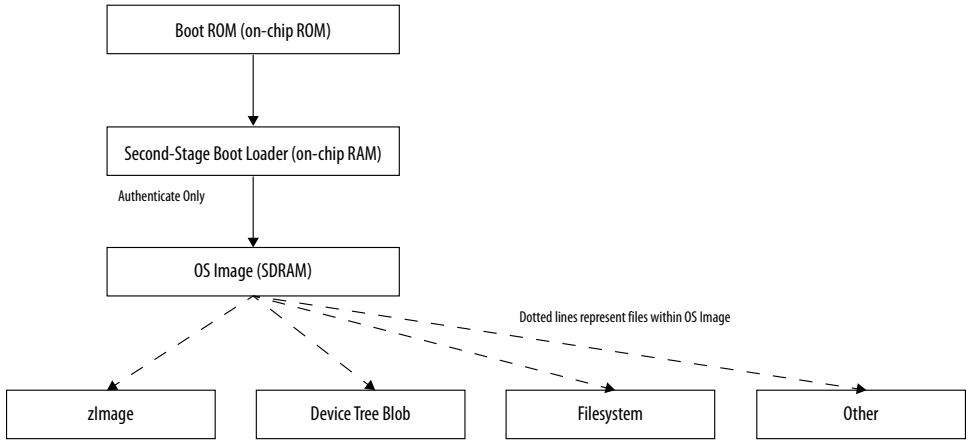
Figure 177. Secure Boot Flow



The micro OS provides secure APIs to allow the application in the normal world OS to establish trusted services.

During a verified boot, the second-stage boot loader only authenticates the OS image and other images required by the OS. A flow for a verified boot is shown below.

Figure 178. Verified (Authenticated) Boot Flow



For both the secure and verified boot, the subsequent boot image must be executed in on-chip RAM while the second-stage boot loader is still executing from on-chip RAM. In order to accommodate this requirement, the authentication and decryption process might follow the following steps depicted in the next three diagrams, depending on the type of secure boot chosen.

Figure 179. Second-Stage Boot Loader Authentication Process

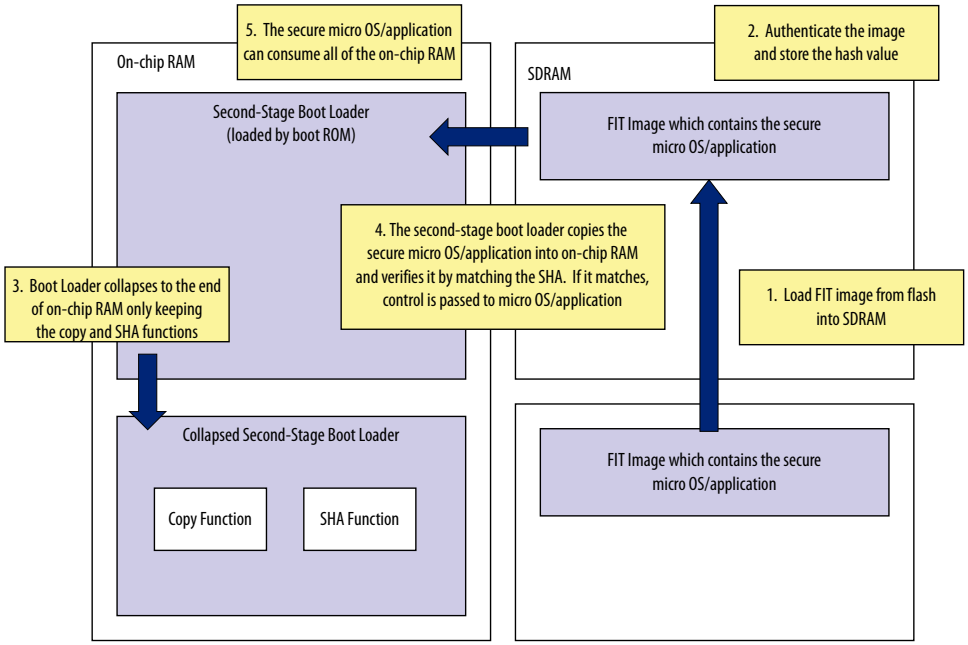
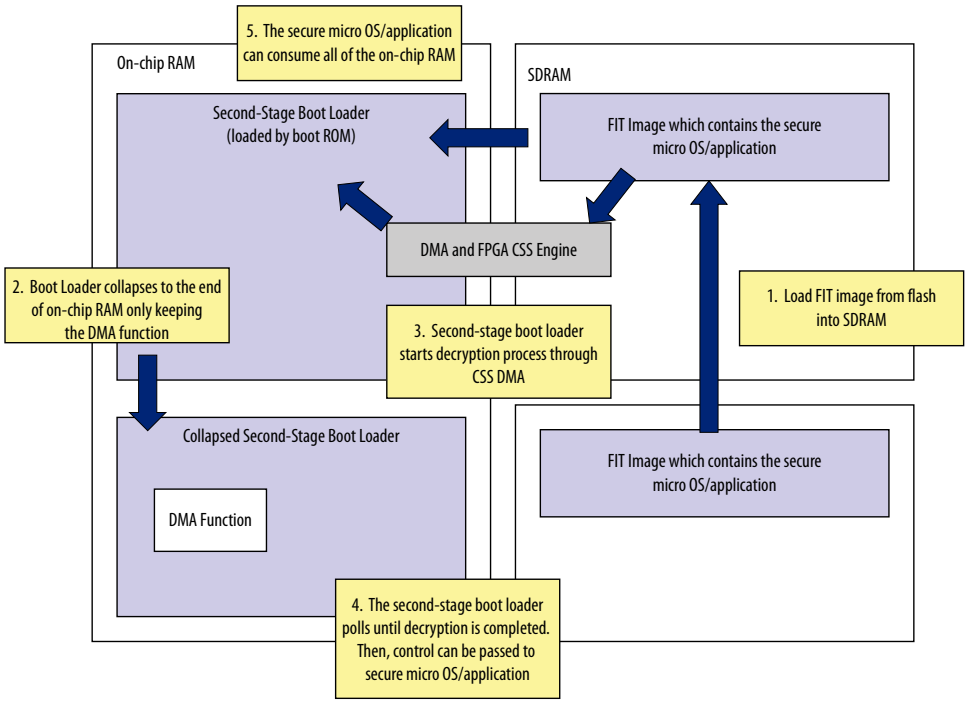
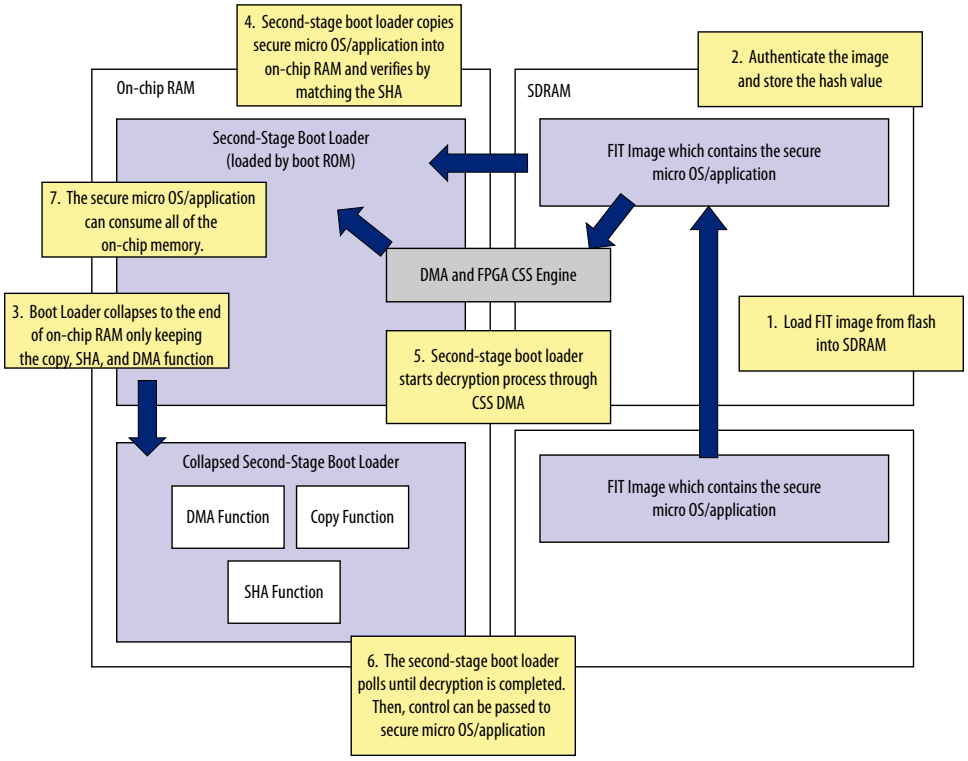


Figure 180. Second-Stage Boot Loader Decryption Process



Decryption is optional and is not required for secure boot. Upon entry into the second-stage boot loader, the CSS engine is enabled. The second-stage boot loader decrypts the subsequent boot image and disables the CSS engine upon exit.

Figure 181. Second-Stage Boot Loader Authentication and Decryption Process



A.6.3. HPS State on Entry to the Second-Stage Boot Loader

When the boot ROM code is ready to pass control to the second-stage boot loader, the processor (CPU0) is in the following state:

- Instruction cache is enabled
- Branch predictor is enabled
- Data cache is disabled
- MMU is disabled
- Floating point unit is enabled
- NEON vector unit is enabled
- Processor is in Arm secure supervisor mode

The boot ROM code sets the Arm Cortex-A9 MPCore registers to the following values:

- r0—contains the pointer to the shared memory block, which is used to pass information from the boot ROM code to the second-stage boot loader. The shared memory block is located in the top 4 KB of on-chip RAM.
- r1—contains the length of the shared memory.
- r2—unused and set to 0x0.
- r3—points to the version block.

All other MPCore registers are undefined.

Note: When booting CPU0 using the FPGA boot, or when booting CPU1 using any boot source, all MPCore registers, caches, the MMU, the floating point unit, and the NEON vector unit are undefined. HPS subsystems and the PLLs are undefined.

When the boot ROM code passes control to the second-stage boot loader, the following conditions also exist:

- CPU0 is in a secure system mode.
- The boot ROM is still mapped to address 0x0.
- The L4 watchdog 0 timer is active and has been toggled.
- The shared L2 cache is left in an uninitialized state.

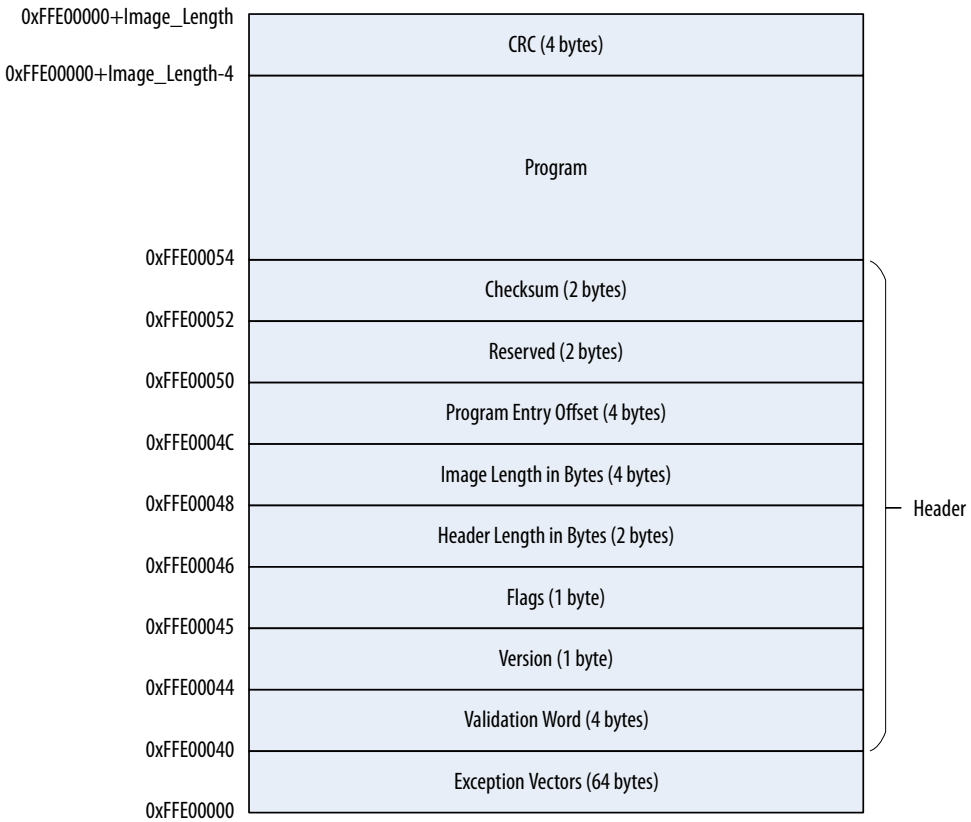
The boot ROM also saves the state of the reset cause in the `stat` register of the Reset Manager, and this information is available for the Second-Stage Boot Loader to read.

A.6.4. Loading the Second-Stage Boot Loader Image

The boot ROM code loads the image from flash memory into the on-chip RAM and passes control to the second-stage boot loader. The boot ROM code checks for a valid image by verifying the header and cyclic redundancy check (CRC) in the second-stage boot loader image.

The maximum second-stage boot loader size is 208 KB with authentication and 224 KB without. This size is limited by the on-chip RAM size of 256 KB, where 48KB or 32 KB is reserved as a workspace for the boot ROM data and stack depending on whether authentication is enabled or not. The second-stage boot loader can use this reserved region (for its stack and data, for example) after the boot ROM code passes control to the second-stage boot loader. This reserved region is overwritten by the boot ROM code on a subsequent reset. The following figure shows the second-stage boot loader image layout in the on-chip RAM after being loaded from the boot ROM.

Figure 182. Second-Stage Boot Loader Image



- **Exception vectors**—Exception vectors are located at the start of the on-chip RAM. Typically, the second-stage boot loader remaps the lowest region of the memory map to the on-chip RAM (from the boot ROM) to create easier access to the exception vectors.
- **Header**—contains information such as validation word, version, flags, program length, security settings,and checksum for the boot ROM code to validate the second-stage boot loader image before passing control to the second-stage boot loader.
- **Version**—This byte describes the version and layout of the program header. This value must be 0x1.
- **Flags**— Not used.
- **Header length**— The length of the header in bytes.
- **Image length** — The total length of the image (including the exception vectors and the CRC field)
- **Program entry offset**— The boot ROM jumps to this offset. The offset value is in bytes from the start of the program header (0xFFE00040).
- **Reserved** —Not used. Must be set to 0x0.

- **Checksum**— A simple checksum of all the bytes from 0xFFE00040 to 0xFFE00051, inclusive
- **Program** — Program contents which typically includes U-Boot binary and U-Boot Device Tree Blob (DTB).
- **CRC**—There is a CRC value for the entire image. The CRC consists of data located from address 0xFFFE0000 to 0xFFFE0000+(Image Length)–0x0004. The polynomial used to validate the image is:

  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$

  There is no reflection of the bits. The initial value of the remainder is 0xFFFFFFFF and the final value is XORed with 0xFFFFFFFF.

A.7. FPGA Configuration

You can configure the FPGA portion of the SoC device with non-HPS sources or by utilizing the HPS.

When the HPS handles the initial FPGA configuration, software executing on the HPS writes the configuration image to the FPGA Manager in the HPS. Software can control the configuration process and monitor the FPGA status by accessing the control and status register (CSR) interface in the FPGA Manager.

For the HPS to configure the FPGA, the mode select (MSEL) pins must be set to passive mode (fast or slow). The following table shows the pin encodings for MSEL[2:0] that are available to the HPS. Refer to the *Configuration, Design Security, and Remote System Upgrades in Arria 10 Devices* document for a comprehensive list of MSEL encodings.

Table 327. MSEL Pin Settings Options for FPGA Configuration through HPS

Configuration Scheme allowed through HPS	Power-On Reset (POR) Delay	Valid MSEL[2:0]
FPP (x16, and x32)	Fast	000
	Standard	001

Note: You must also select the configuration scheme in the **Configuration** page of the **Device and Pin Options** dialog box in the Intel Quartus Prime software. Based on your selection, the option bit in the programming file is set accordingly.

There are two types of initial FPGA configuration flows you can choose from when using the HPS to configure the FPGA.

- Full FPGA configuration flow: In this flow, the FPGA fabric, the shared I/O and hard memory controller I/O are configured. After full configuration is complete, the hard memory controller I/O, shared I/O and FPGA I/O are available to the HPS.
- Early I/O release configuration flow: In this flow, all I/O are configured and the HPS shared I/O and hard memory controller I/O are released so that the HPS has immediate access to them. At a later time, the FPGA fabric is configured and the FPGA I/O are released and available to the user design.

Note: If you are using the early I/O release configuration flow, you cannot initially use SmartVID to power your device. Instead, you can use a fixed power supply until after the FPGA is configured. When the FPGA is configured, you can then enable SmartVID.

Note: It is important that the HPS is not held in cold or warm reset indefinitely, otherwise the CSS cannot be accessible to FPGA configuration sources.

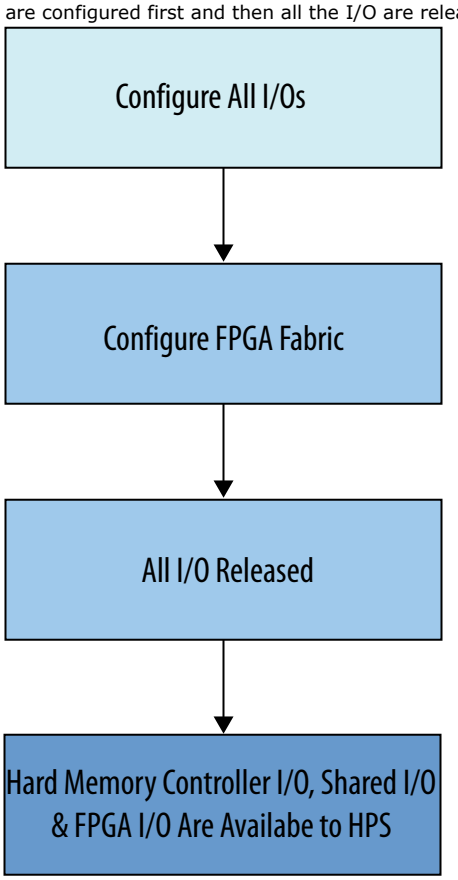
Related Information

- FPGA Manager on page 154
  For more information regarding programming the FPGA through the HPS, refer to the *FPGA Manager* chapter.
- Configuration, Design Security, and Remote System Upgrades in Arria 10 Devices

A.7.1. Full FPGA Configuration Flow Through HPS

Figure 183. Full FPGA Configuration Flow Through HPS

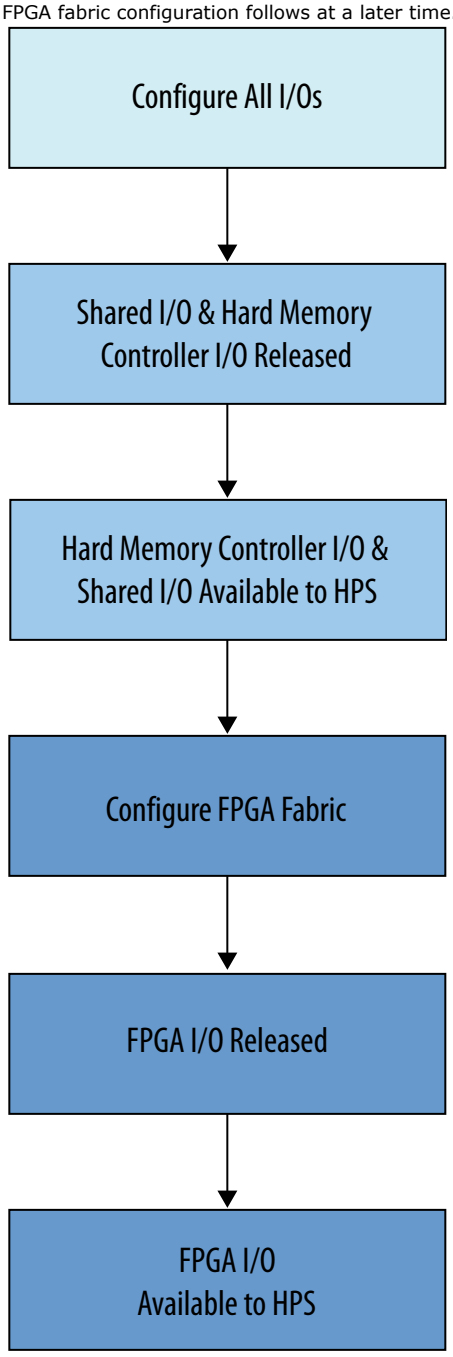
The FPGA fabric and the all I/Os are configured first and then all the I/O are released for use by the HPS.



A.7.2. Early I/O Release FPGA Configuration Flow Through HPS

Figure 184. Early I/O Release FPGA Configuration Flow Through HPS

The I/Os are configured and the shared and hard memory controller I/Os are released so that the HPS has immediate access to them. The FPGA fabric configuration follows at a later time.



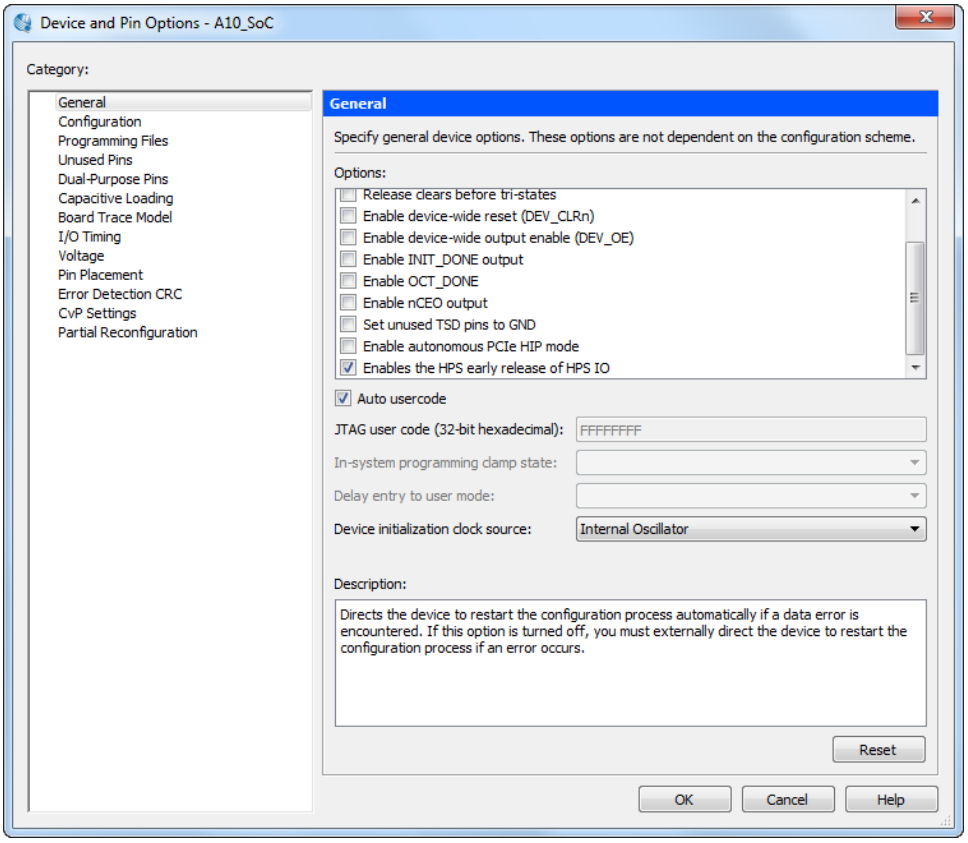
Note: The FPGA fabric is not required to be programmed immediately after the first I/O configuration. You can also program the I/Os, re-program the I/Os and then program the FPGA fabric as long as the early I/O release configuration option is enabled in each I/O configuration file.

Note: If you are using the early I/O release configuration flow, you cannot initially use SmartVID to power your device. Instead, you can use a fixed power supply until after the FPGA is configured. When the FPGA is configured, you can then enable SmartVID.

You can enable early I/O configuration in Intel Quartus Prime:

1. Click on the the **Assignments** menu and select **Device**.
2. When the **Device settings** window opens, click the **Device and Pin Options** button.
3. In the **General** category, click on the option labeled "Enables the HPS early release of HPS IO".

Figure 185. Intel Quartus Prime Early I/O Release Setting



A.7.2.1. Handling an FPGA Configuration Failure after Early I/O Release

If an FPGA configuration failure occurs after early I/O release, the HPS shared I/O and hard memory controller I/O enter input tri-state mode. In this situation, any accesses to peripherals or memory interfacing these I/O fails. HPS dedicated I/O are not affected by FPGA configuration failures.

If you are using early I/O release and issuing an HPS-initiated FPGA configuration afterward, you must ensure that your software routine includes a way to avoid FPGA configuration failure or a recovery mechanism that does not rely on the HPS shared I/O or hard memory controller I/O.

You must implement one of the following solutions in software.

- To prevent FPGA configuration failure, load the FPGA core configuration file into SDRAM and perform an FPGA integrity check before initiating configuration. For example, you can run a custom integrity check, such as a CRC, on the `.rbf` file to validate that there are no errors in the file. Intel also recommends that you enable ECC capabilities to avoid bit-stream corruption issues.
- To recover from an FPGA configuration failure, execute the HPS-initiated configuration from on-chip RAM. If the HPS is executing entirely from on-chip RAM, you can recover your application by reconfiguring the peripheral `.rbf` file to reenable the HPS shared I/O and hard memory controller I/O. After these I/O are recovered, your application can either attempt to configure the same FPGA image that caused the original failure or use a fallback core `.rbf` file to prevent the same failure from occurring repeatedly.

Note: If you are using full FPGA configuration, you do not need to make any of the adjustments listed above. For full FPGA configuration, the .rbf file contains both the core and peripheral configuration and your application can only access the HPS I/O after the device is successfully configured.

A.7.3. Arria 10 SoC FPGA Configuration Sequence Through FPGA Manager

The FPGA Manager in the HPS has the capability to execute an initial FPGA configuration using the full or early I/O release flow.

The steps below detail the programming of the FPGA Manager to initiate configuration.

1. Read the `f2s_msel[2:0]` field in the `imgcfg_stat` register to verify that the configuration mode is the passive fast (000) or passive slow (001).
2. Write the `cfgwidth` bit of the `imgcfg_ctrl_02` register in the FPGA manager to match the characteristics of the configuration image. For full, standard configuration, the `cfgwidth` bit must be set to 1. You can only initiate a full, standard configuration in 32-bit passive parallel mode.
3. Configure the `cdratio` field. The `cdratio` is a function of the datawidth and whether the POF in encrypted or compressed. This information should be provided through the Intel Quartus Prime configuration. For example,if `cfgwidth` is 32-bits and the POF is compressed, you must program the `cdratio` field to 0x8. If the POF is encrypted, then program the `cdratio` to 0x4.
4. Ensure there are no other external devices interfering with the FPGA programming. Poll the `f2s_nconfig_pin` and `f2s_nstatus_pin` bits from the `imgcfg_stat` register until they are both 1.
5. Program the following bits to deassert signal drives from the HPS before overriding the signals.
   - `s2f_nce` =1 in the `imgcfg_ctrl_01` register
   - `s2f_pr_request`= 0 in the `imgcfg_ctrl_01` register
   - `en_cfg_ctrl`= 0 in the `imgcfg_ctrl_02` register

- s2_nconfig = 1 in the imgcfg_ctrl_00 register
- s2f_nstatus_oe = 0 in the imgcfg_ctrl_00 register
- s2f_condone_oe = 0 imgcfg_ctrl_00 register

6. Enable overrides for the DATA, DCLK, nCE, PR_REQUEST and nCONFIG signals.
   - s2f_nenable_config = 0 in the imgcfg_ctrl_01 register
   - s2f_nenable_nconfig = 0 in the imgcfg_ctrl_00 register
7. Disable overrides for the nSTATUS and CONF_DONE signals.
   - s2f_nenable_nstatus = 1 in the imgcfg_ctrl_00 register
   - s2f_nenable_condone = 1 in the imgcfg_ctrl_00 register
8. Assert the chip select signal.
   - s2f_nce=0 in the imgcfg_ctrl_01 register
9. Repeat step 4. Ensure there are no other external devices interfering with the FPGA programming. Poll the f2s_nconfig_pin and f2s_nstatus_pin bits from the imgcfg_stat register until they are both 1.
10. Reset the configuration.
    a. Write a 0 to the s2f_nconfig bit in the imgcfg_ctrl_00 register.
    b. Poll the f2s_nstatus_pin bit in the imgcfg_stat register until it is clear.
    c. Write a 1 to the s2f_nconfig bit in the imgcfg_ctrl_00 register.
    d. Poll the f2s_nstatus_pin bit in the imgcfg_stat register until it is set. In addition, read and confirm the f2s_condone_pin is clear and the f2s_condone_oe is set in the imgcfg_stat register.
11. Enable DCLK and the DATA path by setting the en_cfg_ctrl bit in the imgcfg_ctrl_02 register.
12. Write the bitstream data to the img_data_w register. When sending the bitstream data, you can periodically read and confirm that the f2s_nstatus_pin bit is set in the imgcfg_stat register. If it is not set, configuration has failed and you must restart with step 1.
13. When you have completed writing the data, continuously poll the f2s_condone_pin bit in the imgcfg_stat register until it reads as 1. When this bit is 1, it indicates configuration has completed. If the routine times out with f2s_nstatus_pin=0, configuration has failed and you must restart from step 1.
14. Write the dclkcnt register with 0xF and continuously poll until the dclkcntstat register reads as 0x0. A minimum of 2 DCLK cycles are needed after the CONFIG_DONE assertion for initialization to start.
15. Wait for the initialization sequence to complete. Poll the f2s_usermode bit in the imgcfg_stat register until it reads as 1.
16. Disable the DATA path and DCLK by clearing the en_cfg_ctrl bit in the imgcfg_ctrl_02 register.
17. Disable the chip select. Set the s2f_nce bit in the imgcfg_ctrl_01 register.
18. Disable overrides to the nCONFIG, DATA and DCLK signals.
    - s2f_nenable_config=1

- `s2f_nenable_nconfig=1`

19. Check that user mode is enabled and the configuration is done by checking that the bits below have the following values:
    - `f2s_usermode=1` in the `imgcfg_stat` register
    - `f2s_nstatus_pin=1` in the `imgcfg_stat` register
    - `f2s_condone_pin=1` in the `imgcfg_stat` register

A.8. FPGA Reconfiguration

The FPGA Manager also supports FPGA reconfiguration through two methods: full or partial reconfiguration.

A.8.1. Full FPGA Reconfiguration

A full FPGA reconfiguration requires a system reboot. The following table details the supported reconfiguration options:

Table 328. Supported Reconfiguration Options

Reconfiguration Option	Configuration Method	HPS Reboot Required
FPGA Core Only	Partial Reconfiguration	No
FPGA andHPS Shared I/O(63)	Full Reconfiguration	Yes
FPGA(64)(63)	Full Reconfiguration	Yes
HPS Shared I/O(65)(63)	Full Reconfiguration	Yes
FPGA Core + FPGA I/O + HPS Shared I/O(63)	Full Reconfiguration	Yes

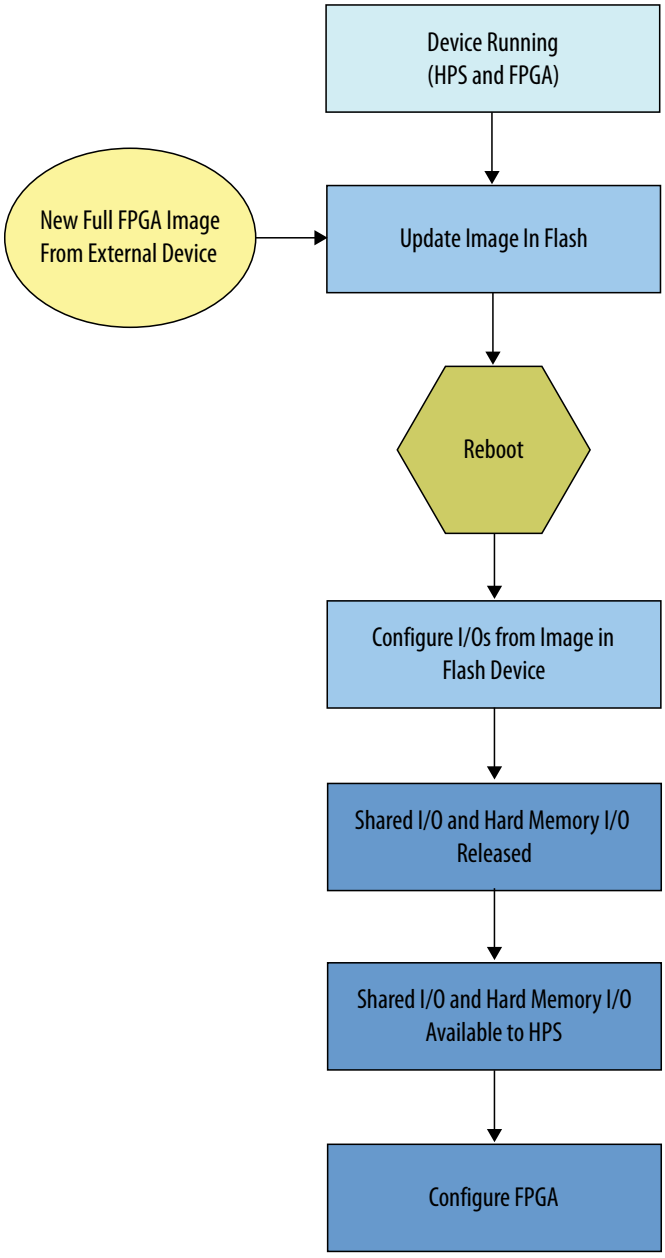
If a new FPGA fabric image is required and performing a system reboot is acceptable, then you can update the FPGA image in flash and perform a full system reboot of both hardware and software before following the steps found in the "Arria 10 SoC FPGA Configuration Sequence Through FPGA Manager."

(63) HPS dedicated I/O are not affected by full reconfiguration.

(64) HPS dedicated I/O are not affected by full reconfiguration.

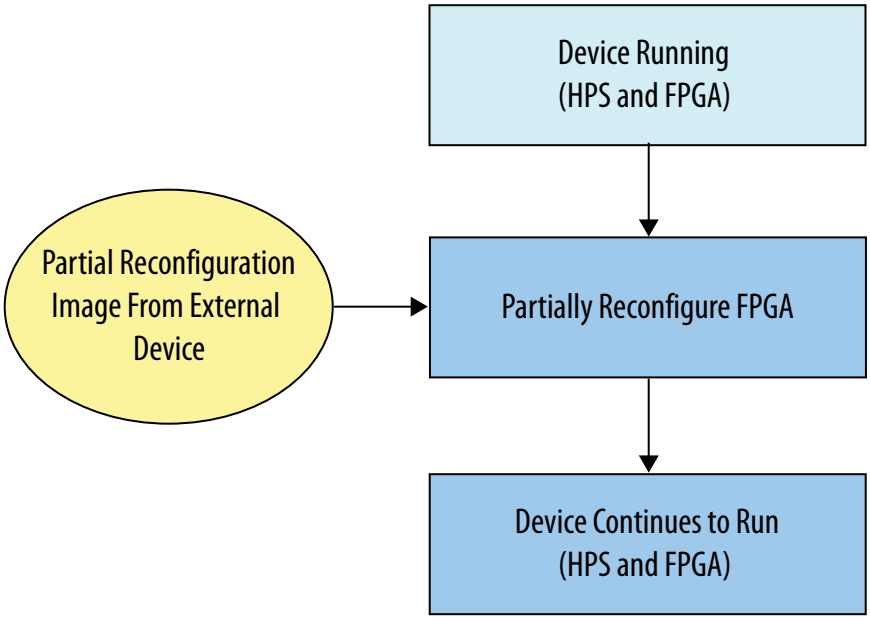
(65) HPS dedicated I/O are not affected by full reconfiguration.

Figure 186. Full FPGA Fabric Reconfiguration with System Reboot



If a new FPGA fabric image is required and performing a system reboot is unacceptable because the hard memory controller or shared I/O are being used by the HPS, then you can perform a reconfiguration using the available partial reconfiguration support. For the partial reconfiguration sequence, follow the steps found in the "Arria 10 SoC FPGA Partial Reconfiguration Sequence Through FPGA Manager" section.

Figure 187. Full FPGA Fabric Reconfiguration without System Reboot



Note: It is important that the HPS is not held in cold or warm reset indefinitely, otherwise the CSS cannot be accessible to FPGA configuration sources.

Related Information

- Arria 10 SoC FPGA Configuration Sequence Through FPGA Manager on page 817
- Arria 10 SoC FPGA Partial Reconfiguration Sequence Through FPGA Manager on page 821

A.8.2. Arria 10 SoC FPGA Partial Reconfiguration Sequence Through FPGA Manager

Partial reconfiguration allows you to reconfigure part of the device while other sections remain running. This implementation divides the design in to a static region and a partial reconfiguration region.

- Static Region: Comprised of hard memory controller I/O, shared I/O, FPGA I/O and a portion of your FPGA design
- Dynamic Region: Remaining portion of FPGA design

The HPS performs partial reconfiguration while the FPGA portion of the device is in user mode. The following sequence suggests one way for software to perform a partial configuration. When using this sequence:

- If `f2s_pr_ready=1` and `f2s_pr_error=0` in the `imgcfg_stat` register of the FPGA Manager, begin at step 9.
- If the `f2s_pr_ready` bit in the `imgcfg_stat` register of the FPGA Manager is clear, then begin the sequence at step 1.

If an HPS warm reset occurs in the middle of a partial reconfiguration, software must repeat the steps for partial configuration. After an HPS cold reset, software must repeat the steps for Arria 10 SoC FPGA Configuration Sequence Through FPGA Manager on page 817.

1. Read the `f2s_usermode` bit of the `imgcfg_stat` register in FPGA Manager to ensure that the FPGA is in user mode.
2. Read the `f2s_msel[2:0]` field in the `imgcfg_stat` register to verify that the configuration mode is the passive fast (000) or passive slow (001). The FPGA manager only supports passive parallel programming.
3. Set the `cdratio` bit of the `imgcfg_ctrl_02` register to match the characteristics of the partial reconfiguration image and configure the `cfgwidth` bit of the `imgcfg_ctrl_02` register to 0. Partial reconfiguration only supports 16-bit passive parallel programming.
4. Prepare for partial reconfiguration. Set the `en_cfg_ctrl` bit of the `imgcfg_ctrl_02` register to give the FPGA manager control of the configuration input signals.
   a. Read and confirm that `s2f_pr_request=0` in the `imgcfg_ctrl_01` register and `s2f_nconfig=1` in the `imgcfg_ctrl_00` register.
   b. Drive the chip select signal by clearing the s2f_nce bit in the `imgcfg_ctrl_01` register.
   c. Enable overrides for configuration data and control by setting the `en_cfg_ctrl` bit in the `imgcfg_ctrl_02` to 1.
   d. Disable overrides that are not used for partial reconfiguration by programming the following bits in the `imgcfg_ctrl_00` register:
      - `s2f_condone_oe = 0`
      - `s2f_nstatus_oe = 0`
      - `s2_nconfig = 1`
      - `s2f_nenable_condone = 1`
      - `s2f_nenable_nstatus = 1`
      - `s2f_nenable_nconfig = 0`
5. Enable HPS to override the DATA, DCLK, nCE and PR_REQUEST signals to the CSS by clearing the `s2f_nenable_config` bit in the `imgcfg_ctrl_01` register.
6. Run DCLK to clear any errors. Write a value of 0x100 to the `dclkcnt` register to generate 256 DCLK cycles. Poll the `dclkstat` register until `dcntdone=1`. You must implement a timeout in your software loop that either exits with an error or restarts at step 1 if `dcntdone` is not set after a period of time.
7. Initiate a partial reconfiguration request.
   a. Set the `s2f_pr_request` bit of the `imgcfg_ctrl_01` register to assert PR_REQUEST
   b. Write 0x7FF to the `dclkcnt` register.
   c. Poll the `dclkstat` register until `dcntdone=1` .Again, you must implement a timeout in your software loop that either exits with an error or restarts at step 1 if `dcntdone` is not set after a period of time.

8. Poll the f2s_pr_ready bit and the f2s_pr_error bit in the imgcfg_stat register. When f2s_pr_ready=1, the PR_READY signal is asserted. Alternately, the FPGA Manager can be configured to generate an interrupt when the PR_READY signal is asserted. If f2s_pr_error=1, you cannot continue with partial reconfiguration because security is preventing the software from updating the configuration. Intel recommends that your software routine implement an exit with a timeout error when f2s_pr_ready and f2s_pr_error continue to read as 0 after ten register reads.
9. When the PR_READY signal is asserted, write the partial reconfiguration image to the img_data_w register in the FPGA Manager. You can also choose to use a DMA to transfer the configuration image from a peripheral device to the FPGA manager.
10. Poll the imgcfg_stat register to observe the f2s_pr_done and f2s_pr_error bits. When the f2s_pr_done bit is set, partial FPGA reconfiguration is complete. If f2s_pr_done or f2s_pr_error is not set to 1 after reading the imgcfg_stat register ten times, generate a timeout error and continue on to the next step.
11. Clear the s2f_pr_request bit of the imgcfg_ctrl_01 register to deassert PR_REQUEST.
12. Poll the dcntdone bit of the dclkstat register until it reads as 1, which indicates that all the DCLKs have been sent.
13. Write a 1 to the dcntdone bit of the dclkstat register to clear the completed status flag.
14. Disable the DATA path and DCLK by clearing the en_cfg_ctrl bit of the imgcfg_ctrl_02 register.
15. Disable the chip select. Set the s2f_nce bit in the imgcfg_ctrl_01 register.
16. Disable overrides to the nCONFIG, DATA and DCLK signals by setting the s2f_nenable_config in the imgcfg_ctrl_01 register.
17. Check that user mode is enabled and the configuration is done by checking that the bits below have the following values:
    - f2s_usermode=1 in the imgcfg_stat register
    - f2s_nstatus_pin=1 in the imgcfg_stat register
    - f2s_condone_pin=1 in the imgcfg_stat register

Related Information

Arria 10 SoC FPGA Configuration Sequence Through FPGA Manager on page 817