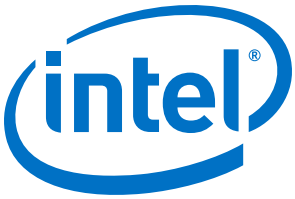


Intel® Agilex™ Hard Processor System Technical Reference Manual

Updated for Intel® Quartus® Prime Design Suite: **19.4**

Subscribe
Send Feedback

MNL-1100 | 2020.01.25
Latest document on the web: PDF | HTML

Contents

1. Intel® Agilex™ Hard Processor System Technical Reference Manual Revision History

Table 1. Intel® Agilex™ Hard Processor System Technical Reference Manual Revision History Summary

Chapter	Date of Last Update
Introduction to the Hard Processor System	September 30, 2019
Cortex-A53 MPCore* Processor	September 30, 2019
Cache Coherency Unit	September 30, 2019
System Memory Management Unit	September 30, 2019
System Interconnect	September 30, 2019
HPS-FPGA Bridges	September 30, 2019
DMA Controller	January 25, 2020
On-Chip RAM	September 30, 2019
Error Checking and Correction Controller	September 30, 2019
Clock Manager	September 30, 2019
Reset Manager	January 25, 2020
System Manager	September 30, 2019
Hard Processor Subsystem I/O Pin Multiplexing	September 30, 2019
NAND Flash Controller	January 25, 2020
SD/MMC Controller	January 25, 2020
Ethernet Media Access Controller	September 30, 2019
USB 2.0 OTG Controller	January 25, 2020
SPI Controller	September 30, 2019
I^2C Controller	September 30, 2019
UART Controller	September 30, 2019
General-Purpose I/O Interface	September 30, 2019
Timer	September 30, 2019
Watchdog Timer	September 30, 2019
CoreSight* Debug and Trace	September 30, 2019
Booting and Configuration	July 1, 2019
Accessing the SDM Quad SPI Flash Controller through HPS	September 30, 2019

Table 2. Introduction to the Hard Processor System Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Introduction to the Hard Processor System on page 18

Table 3. Cortex-A53 MPCore Processor Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Cortex-A53 MPCore Processor on page 32

Table 4. Cache Coherency Unit Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.07.01	Added the following sections: • *Reset and Initialization* • *Discovery Routine* • *Operational State* • *Maintenance Operations* • *Error Handling* • *OCRAM Firewall*
2019.04.02	Initial release.

Cache Coherency Unit on page 60

Table 5. System Memory Management Unit Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

System Memory Management Unit on page 71

Table 6. System Interconnect Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.07.01	• Added the missing data width for MPFE blocks in *Figure: Block Diagram*. • Corrected the *Figure: Generic Timestamp Connection*. • Corrected the GIC address region in *Figure: L3 Address Regions*. • Corrected the address range in *Figure: SDRAM Regions*. • Added a new section: *Peripheral Region Address Map*.
2019.04.02	Initial release.

System Interconnect on page 82

Table 7. HPS-FPGA Bridges Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.07.01	Added information about FPGA Fabric Bypass Mux in section: *FPGA-to-SoC Bridge.*
2019.04.02	Initial release.

HPS Bridges on page 108

Table 8. DMA Controller Revision History

Document Version	Changes
2020.01.25	Clarified reset information in section: *DMA Controller Block Diagram*.
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

DMA Controller on page 116

Table 9. On-Chip RAM Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

On-Chip RAM on page 124

Table 10. Error Checking and Correction Controller Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Error Checking and Correction Controller on page 128

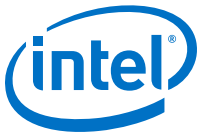
Table 11. Clock Manager Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Clock Manager on page 149

Table 12. Reset Manager Revision History

Document Version	Changes
2020.01.25	Added a new section: *SoC-to-FPGA Reset Sequence*.
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.07.01	Corrected steps in section: *Warm Reset Sequence*.
2019.04.02	Initial release.



Reset Manager on page 161

Table 13. System Manager Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

System Manager on page 171

Table 14. Hard Processor System I/O Pin Multiplexing Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Hard Processor System I/O Pin Multiplexing on page 178

Table 15. NAND Flash Controller Revision History

Document Version	Changes
2020.01.25	Clarified reset information in section: *Taking the NAND Flash Controller Out of Reset*.
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

NAND Flash Controller on page 185

Table 16. SD/MMC Controller Revision History

Document Version	Changes
2020.01.25	Clarified reset information in section: *Taking the SD/MMC Controller Out of Reset*.
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

SD/MMC Controller on page 221

Table 17. Ethernet Media Access Controller Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Ethernet Media Access Controller on page 308

Table 18. USB 2.0 OTG Controller Revision History

Document Version	Changes
2020.01.25	Clarified reset information in section: *Taking the USB 2.0 OTG Controller Out of Reset*.
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

USB 2.0 OTG Controller on page 381

Table 19. SPI Controller Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

SPI Controller on page 397

Table 20. I²C Controller Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

I2C Controller on page 430

Table 21. UART Controller Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

UART Controller on page 456

Table 22. General-Purpose I/O Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

General-Purpose I/O Interface on page 470

Table 23. Timers Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Timers on page 473

Table 24. Watchdog Timers Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Watchdog Timers on page 478

Table 25. CoreSight Debug and Trace Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

CoreSight Debug and Trace on page 485

Table 26. Booting and Configuration Revision History

Document Version	Changes
2019.07.01	Simplified information in the appendix. For more information, refer to the *Intel Agilex Configuration User Guide and Intel Agilex Boot User Guide*.
2019.04.02	Initial release.

Booting and Configuration on page 505

Table 27. Accessing the SDM Quad SPI Flash Controller through HPS Revision History

Document Version	Changes
2019.09.30	Added links to access the complete HPS address map and register definitions.
2019.04.02	Initial release.

Accessing the Secure Device Manager Quad SPI Flash Controller through HPS on page 511

2. Introduction to the Hard Processor System

The Intel Agilex system-on-a-chip (SoC) is composed of two distinct portions: a 64-bit quad core Arm* Cortex*-A53 MPCore 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.

The HPS communicates outside of the SoC through the following types of interfaces:

- Dedicated I/O interfaces
- FPGA fabric interfaces
- FPGA secure device manager (SDM) interfaces

Key modules in the HPS include:

- Quad core Arm Cortex-A53 MPCore processor
- Level 3 (L3) interconnect
- Cache Coherency Unit (CCU)
- System Memory Management Unit (SMMU)
- Multi-port front end (MPFE) subsystem, consisting of the hard memory controller adaptor and interface to the CCU interconnect
- DMA Controller
- On-chip RAM
- Debug components
- PLLs
- Flash memory controllers
- Support peripherals
- Interface peripherals

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

The FPGA portion of the device contains:

- FPGA fabric
- PLLs
- User I/O
- Hard memory controllers
- Secure Device Manager (SDM)

The HPS and FPGA portions of the device each have their own pins. The HPS has dedicated I/O pins. You can also route most of the HPS peripherals into the FPGA fabric to use the FPGA I/O. You can configure pin placement assignments when you instantiate the HPS component in Intel Platform Designer System Integration Tool.

You can boot the SoC from a power-on reset in one of two ways:

- FPGA configures first and then optionally boots the HPS (also called FPGA Configuration First).
- HPS boots first and then configures the FPGA (called HPS Boot First).

For more information, refer to the "Boot and Configuration" appendix.

2.1. Features of the HPS

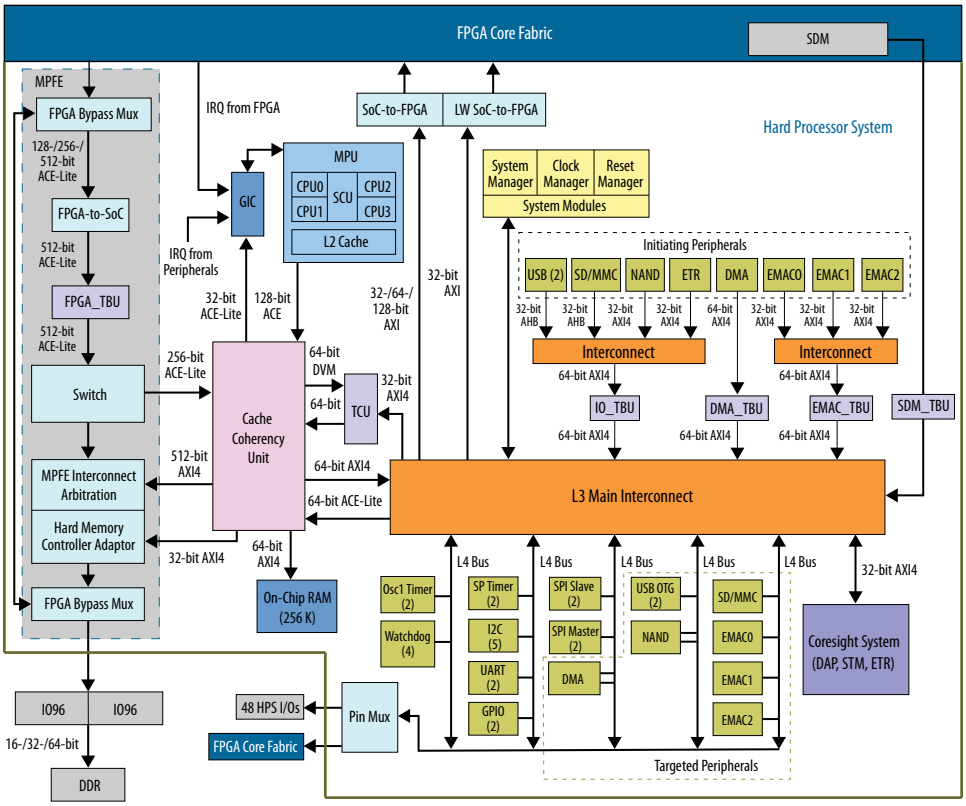
- Quad-core Arm Cortex-A53 MPCore processor
- Cache Coherency Unit (CCU)
- System Memory Management Unit (SMMU)
- System interconnect that includes:
  - L3 main interconnect
    - Provides high bandwidth routing from master to slave
    - Provides two memory-mapped SoC-to-FPGA interfaces:
      - SoC-to-FPGA bridge (32-, 64-, or 128-bit wide Arm Advanced Microcontroller Bus Architecture (AMBA*) Advanced eXtensible Interface (AXI*)-4)
      - Lightweight SoC-to-FPGA bridge: 32-bit wide AXI-4
  - MPFE interconnect
    - Routes transactions from FPGA translation buffer units (TBU) to SDRAM or SoC
- General-purpose direct memory access (DMA) controller
- 256 KB on-chip RAM
- Error checking and correction controllers for on-chip RAM and peripheral RAMs
- Clock manager
- Reset manager
- System manager
- Dedicated I/O pin multiplexer (MUX)
- NAND flash controller
- Secure digital/multimedia card (SD/MMC) controller
- Three Ethernet media access controllers (EMACs)
- Two USB 2.0 on-the-go (OTG) controllers
- Two serial peripheral interface (SPI) master controllers
- Two SPI slave controllers

- Five inter-integrated circuit (I^2C) controllers:
  - Three can provide support for EMAC
  - Two for general purpose
- Two UARTs
- Two general-purpose I/O (GPIO) interfaces with a total of 48 dedicated I/O
- Four system timers
- Four watchdog timers
- Arm CoreSight debug components:
  - Debug access port (DAP)
  - Trace port interface unit (TPIU)
  - System trace macrocell (STM)
  - Embedded trace macrocell (ETM)
  - Embedded trace router (ETR)
  - Embedded cross trigger (ECT)

2.2. HPS Block Diagram and System Integration

2.2.1. HPS Block Diagram

Figure 1. Intel Agilex HPS Block Diagram



2.2.2. Cortex-A53 MPCore Processor

The Intel Agilex SoC integrates a full-featured Arm Cortex-A53 MPCore Processor.

The Cortex-A53 MPCore supports high-performance applications and provides the capability for secure processing and virtualization. Each CPU in the processor has the following features:

- Support for 32- and 64-bit instruction sets
- In-order pipeline with symmetric dual-issue of most instructions
- Arm NEON* single instruction, multiple data (SIMD) co-processor with a floating-point unit (FPU)
  - Single- and double-precision IEEE-754 floating point math support
  - Integer and polynomial math support
- Symmetric multiprocessing (SMP) and asymmetric multiprocessing (AMP) modes
- Armv8 Cryptography Extension
- Level 1 (L1) cache
  - 32 KB two-way set associative instruction cache
  - Single Error Detect (SED) and parity checking support for L1 instruction cache
  - 32 KB four-way set associative data cache
  - Error checking and correction (ECC), Single Error Correct, Double Error Detect (SECDED) protection for L1 data cache
- Memory Management Unit (MMU) that communicates with the system MMU (SMMU)
- Generic timer
- Governor module that controls clock and reset
- Debug modules
  - Performance Monitor Unit
  - Embedded Trace Macrocell (ETMv4)
  - CoreSight cross trigger interface

The four CPUs share a 1 MB L2 cache with ECC, SECDED protection. A snoop control unit (SCU) maintains coherency between the CPUs and communicates with the system cache coherency unit (CCU).

At a system level, the Cortex-A53 MPCore interfaces to a generic interrupt controller (GIC), CCU, and system memory management unit (SMMU).

2.2.3. Cache Coherency Unit

The cache coherency unit allows I/O masters to maintain one-way coherency with the Cortex-A53 MPCore. It acts as an interconnect among the processor, FPGA-to-SoC bridge, system MMU, multiport front end (MPFE) subsystem and peripheral masters interfacing the system interconnect and supports weighted priority of memory accesses.

The CCU features include:

- Coherency directory to track the state of the L2 and L1 caches in the Arm Cortex-A53 MPCore
- Snoop filter support for tracking coherent lines and sending coherency transaction requests, including cache maintenance operations
- Support for distributed virtual memory (DVM) using the Arm AXI Coherency Extensions (ACE) protocol. Distributed virtual memory broadcast messages are sent to the Cortex-A53 MPCore and translation control unit (TCU) in the system memory management unit (SMMU)
- Quality-of-service (QoS) support for transaction prioritization using a weight bandwidth allocation
- Interconnect debug capability through master and slave bridge status registers
- Interrupt support for CCU transaction and counter events

2.2.4. System Memory Management Unit

The SMMU provides system-wide address translation for system bus masters. A two-stage translation supports memory virtualization. The module includes a single TCU that controls distributed translation buffer units (TBUs).

The system MMU features include:

- A central TCU that supports five distributed TBUs for the following masters:
  - FPGA
  - DMA
  - EMAC0-2, collectively
  - USB0-1, NAND, SD/MMC, ETR, collectively
  - Secure Device Manager (SDM)
- Caches for storing page table entries and intermediate table walk data:
  - 512-entry macro translation lookaside buffer (TLB) page table entry cache in the TCU
  - 128-entry micro TLB for table walk data in the FPGA TBU and 32-entry micro TLB for all other distributed TBUs
  - Single-bit error detection and invalidation on error detection for caches
- Communication with the MMU of the Arm Cortex-A53 MPCore
- System-wide address translation
- Address virtualization
- Support for 32 contexts
- Two stages of translation or combined (stage 1 and stage 2) translation
- Support for up to 49-bit virtual addresses and up to 48-bit physical and intermediate physical addresses
- Programmable QoS to support page table walk arbitration
- Fault handling, logging and interrupts for translation errors
- Debug support

2.2.5. HPS Interfaces

The Intel Agilex device family provides multiple communication channels between the FPGA, HPS, and SDRAM.

2.2.5.1. SoC-FPGA Memory-Mapped Interfaces

The SoC-FPGA memory-mapped interfaces provide the major communication channels among the HPS, the FPGA fabric, and SDRAM. The SoC-FPGA memory-mapped interfaces include:

- FPGA-to-SoC bridge—a high–performance bus with a configurable data width of 128, 256, or 512 bits, allowing the FPGA fabric to master transactions to the slaves in the HPS or access to SDRAM through the MPFE interconnect. This interface allows the FPGA fabric to have full visibility into the HPS address space. This interface supports single-direction I/O coherency with the HPS MPU.
- SoC-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 SoC-to-FPGA bridge—an interface with a 32–bit fixed data width, allowing the HPS to master transactions to slaves in the FPGA fabric. This bridge is primarily used for control and status register accesses.

2.2.5.2. Other HPS Interfaces

- TPIU trace—sends trace data created in the SoC-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 vice versa.
- DMA peripheral interface—multiple peripheral–request channels.
- 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–A53 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.

2.2.6. System Interconnect

The system interconnect supports the following features:

- Configurable Arm TrustZone*-compliant firewall and security support.
  - For each peripheral, implements secure or non-secure access.
  - Allows configuration of individual transactions as secure or non-secure at the initiating master.
- Multi-tiered bus structure to separate high bandwidth masters from lower bandwidth peripherals and control and status ports.
- Quality of service (QoS) with three programmable levels of service on a per masterbasis.
- On-chip debugging and tracing capabilities. The system interconnect is based on the Arteris® FlexNoC™ network-on-chip (NoC) interconnect technology.

2.2.6.1. MPFE Subsystem

The multiport front end (MPFE) subsystem connects the HPS to the hard memory controller adaptor (HMCA) that is located in the FPGA portion of the device. The MPFE subsystem includes an MPFE Interconnect, which is secured by firewalls. It supports AMBA AXI QoS for the FPGA fabric interfaces.

The MPFE Subsystem implements the following high-level features:

- Support for double data rate 4 (DDR4) devices
- Software-configurable priority scheduling per port
- 8-bit Single Error Correction, Double Error Detection (SECDED) ECC with 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

2.2.7. On-Chip RAM

The on-chip RAM offers the following features:

- 256 KB size
- 64-bit slave interface
- ECC support provides detection of single–bit and double–bit errors and correction for single-bit errors
- Memory scrambling on tamper events

2.2.8. Flash Memory Controllers

The Intel Agilex device family provides two flash memory controllers:

- NAND Flash Controller
- SD/MMC Controller

2.2.8.1. 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 two chip selects
- Integrated descriptor-based direct memory access (DMA) controller
- Supports Open NAND Flash Interface (ONFI) 1.0
- Programmable page sizes of 512 bytes, 2 KB, 4 KB, or 8 KB
- Supports 32, 64, or 128 pages per block
- Programmable hardware ECC
- Supports 8- and 16-bit data width

2.2.8.2. 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:

- Supports eMMC
- Integrated descriptor-based DMA
- Supports CE-ATA digital protocol commands
- Supports only 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 up to 50 MHz flash operating frequency

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

2.2.9. System Modules

2.2.9.1. 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

2.2.9.2. Reset Manager

The reset domains and sequences support several security features. The SDM brings the reset manager out of reset; and after that, 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 HPS sequencing during resets

2.2.9.3. System Manager

The System Manager provides configuration of system-level functions that are required by other modules.

The major areas of control registers are:

- Peripheral control registers
- ECC interrupt registers
- FPGA interface and general-purpose configuration signals
- Boot scratch registers

The System Manager provides the following functionalities:

- Combined ECC status and interrupts from different modules
- Memory-mapped control signals to other modules
- Watchdog stop functionality on debug request
- FPGA interface disable and enable control signals
- AXI/AHB* control signals (`hprot`, `awcache`, `arcache`) to master ports of SD/MMC, NAND, USB and EMAC

2.2.9.4. 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*) Timer peripheral and offer the following features:

- Free-running timer mode
- Supports a time-out period of up to 43 seconds when the timer clock frequency is 100 MHz
- Interrupt generation

2.2.9.5. Watchdog Timers

The HPS provides four watchdogs connected to the L4 buses in addition to the watchdogs built into the MPU. The four watchdog timers have a 32-bit timer resolution and are based on the Synopsys DesignWare APB Watchdog Timer peripheral.

A watchdog timer can be programmed to generate a reset request on a timeout. Alternatively, the watchdog can be programmed to assert an interrupt request on a timeout, and if the interrupt is not serviced by software before a second timeout occurs, generate a reset request.

2.2.9.6. 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 up to 32 peripheral request interfaces

2.2.9.7. 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 controllers
- SD/MMC controller
- EMAC controllers
- DMA controller
- NAND flash controller
- On-chip RAM

Features of the ECC controller:

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

2.2.10. Interface Peripherals

2.2.10.1. EMACs

The three EMACs are based on the Synopsys DesignWare 3504-0 Universal 10/100/1000 Ethernet MAC. Each of the EMACs offers the following features:

- IEEE 802.3-2008 compliant
- Supports 10, 100, and 1000 Mbps standard
- Supports full and half duplex modes
- IEEE 1588-2002 and 2008 precision networked clock synchronization
- IEEE 802.3-az, version D2.0 of Energy Efficient Ethernet (EEE)
- Supports IEEE 802.1Q Virtual local area network (VLAN) tag detection for reception frames
- VLAN insertion, replacement, or deletion

- Supports a variety of flexible address filtering modes
- Programmable frame length support for full jumbo frames up to 9000 Bytes
- The Gigabit media independent interface/Media independent interface (GMII/MII) interface includes optional FIFO loopback to support debugging
- Network statistics with RMON/MIB counters (RFC2819/RFC2665)
- PHY interface support for Reduced Gigabit Media Independent Interface (RGMII) and Reduced Media Independent Interface (RMII) on HPS I/O pins
- PHY interface support for GMII and MII on FPGA I/O pins:
  — Additional PHY interface support on FPGA I/O pins using adapter logic in the FPGA fabric to adapt the GMII/MII interface from the HPS to interfaces such as Serial Gigabit Media Independent Interface (SGMII), RGMII or RMII
- PHY Management control through Management data input/output (MDIO) interface or I^2C interface
- Integrated DMA controller

2.2.10.2. 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

2.2.10.3. I^2C Controllers

There are five I^2C controllers. Two controllers for general purpose usage. The remaining three controllers can be optionally used as a control interface for Ethernet PHY communication. The controllers are based on Synopsys DesignWare APB I^2C controller which offer the following features:

- 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

Note: When an I^2C controller is used for Ethernet, it takes the place of the EMAC MDIO pins.

2.2.10.4. UARTs

The HPS provides two UART controllers to provide asynchronous serial communications:

- 16550-compatible UART
- Support automatic flow control as specified in 16750 standard

The two UART controllers are based on Synopsys DesignWare APB Universal Asynchronous Receiver/ Transmitter peripheral and offer the following features:

- Direct access for host processor
- DMA controller may be used for large transfers
- Separate thresholds for DMA request and handshake signals to maximize throughput
- 128-byte transmit and receive FIFO buffers
- Programmable baud rate up to 6.25 MBaud (with 100MHz reference clock)
- Programmable character properties, such as number of data bits per character (5-8), optional parity bit (with odd or even select) and number of stop bits (1, 1.5 or 2)

2.2.10.5. SPI Master Controllers

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

- Programmable data frame size of 4 - 32 bits
- Supports full- and half-duplex modes
- Supports up to four chip selects
- Direct access for host processor
- DMA controller may be used for large transfers
- Programmable master serial bit rate
- Support for receive sample delay
- Choice of Motorola* SPI, Texas Instruments* Synchronous Serial Protocol or National Semiconductor* Microwire protocol

2.2.10.6. SPI Slave Controllers

There are two slave SPI controllers based on the Synopsys DesignWare Synchronous Serial Interface (SSI) controller that have a maximum bit rate of 33.33 Mbps each. The following features are offered:

- Programmable data frame size from 4 - 32 bits
- Support for full- and half-duplex modes
- Direct access for host processor
- DMA controller may be used for large transfers

2.2.10.7. GPIO Interfaces

The HPS provides two GPIO interfaces that are based on the Synopsys DesignWare APB General Purpose Programming I/O peripheral. Together, these interfaces support up to 48 dedicated GPIO pins with input and output capability. Each GPIO interface offers the following features:

- Digital de-bounce
- Configurable interrupt mode
- Configurable hardware and software control for each signal
- Level and edge interrupts

2.2.11. CoreSight Debug and Trace

The CoreSight Debug and Trace system offers the following features:

- Real-time program flow instruction trace through a separate Embedded Trace Macrocell (ETM) 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 L3 interconnect

2.2.12. Hard Processor System I/O Pin Multiplexing

The Intel Agilex SoC has a total of 48 flexible I/O pins that are used for HPS operation, external flash memories, and external peripheral communication. 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-SoC, SoC-to-FPGA, FPGA–to–SDRAM, and lightweight FPGA-to-SoC 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 DMA controller is software configurable to perform byte lane swapping during a transfer.

2.4. Introduction to the Hard Processor System Address Map

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

3. Cortex-A53 MPCore Processor

The Arm Cortex-A53 MPCore is composed of four Armv8-A architecture central processing units (CPUs), a level 2 (L2) cache, and debugging modules. Advanced functions, such as floating point operations and cryptographic extensions, are supported.

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

3.1. Features of the Cortex-A53 MPCore

The Arm Cortex-A53 MPCore Processor contains four CPUs that implement the Armv8-A architecture instruction set. Each CPU has identical integration.

- Support for 32- and 64-bit instruction sets
- In-order pipeline with symmetric dual-issue of most instructions
- ArmNEON single instruction, multiple data (SIMD) coprocessor with a floating point unit (FPU)
  - Single- and double-precision IEEE-754 floating point math support
  - Integer and polynomial math support
- Symmetric multiprocessing (SMP) and asymmetric multiprocessing (AMP) modes
- Armv8 Cryptography Extension
- Level 1 (L1) cache
  - 32 KB two-way set associative instruction cache
  - Single Error Detect (SED) and parity checking support for L1 instruction cache
  - 32 KB four-way set associative data cache
  - ECC, Single Error Correct, Double Error Detect (SECDED) protection for L1 data cache
- Memory Management Unit (MMU) that communicates with system MMU (SMMU)
  - 10-entry fully-associative instruction micro translation lookaside buffer (TLB)
  - 10-entry fully-associative data micro TLB
  - 512-entry unified TLB

- Generic timer
- Governor module that controls clock and reset
- Debug modules
  - Performance Monitor Unit
  - Embedded Trace Macrocell (ETMv4)
  - CoreSight cross trigger interface

Some integration is also shared among the four CPUs in the Cortex-A53 MPCore processor.

- 1 MB Arm L2 cache controller with ECC, SECDED protection
- Snoop Control Unit (SCU) that maintains coherency between CPUs and communicates with the system CCU
- Global timer

Modules that the Cortex-A53 MPCore interfaces to in the system include:

- Generic Interrupt Controller (GIC-400, version r0p1)
- System cache coherency unit (CCU)
- System memory management unit (SMMU, ARM MMU-500, version r2p0)

The table below lists the Cortex-A53 MPCore version.

Table 28. Cortex-A53 MPCore Module Version

Processor	Version
Cortex-A53 MPCore	r0p4

Related Information

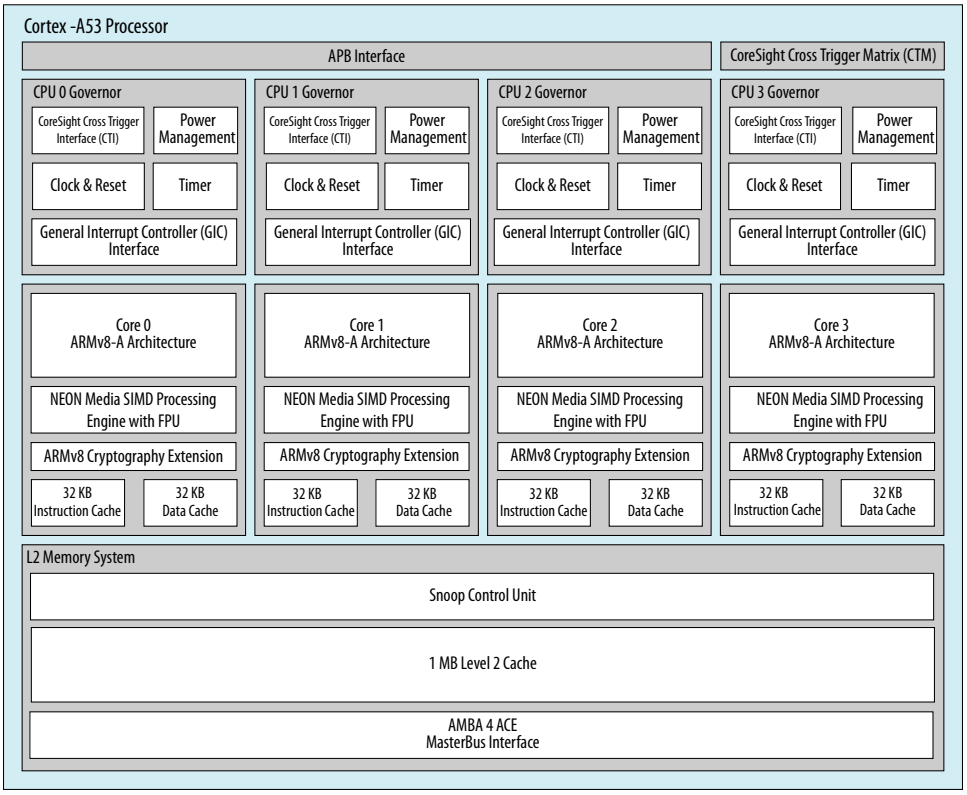
Arm Cortex-A53 MPCore Technical Reference Manual, Revision: r0p4

3.2. Advantages of Cortex-A53 MPCore

The Cortex-A53 MPCore processor seamlessly supports 32-bit and 64-bit instruction sets. It implements the full Armv8-A architecture and has a highly efficient 8-stage in-order pipeline enhanced with advanced fetch and data access techniques that provide high performance and low power.

3.3. Cortex-A53 MPCore Block Diagram

Figure 2. Cortex-A53 MPCore Block Diagram



3.4. Cortex-A53 MPCore System Integration

The Cortex-A53 MPCore is part of the MPU system complex. The system complex is comprised of the Cortex-A53 MPCore, system memory management unit (SMMU), cache coherency unit (CCU), on-chip RAM and generic interrupt controller (GIC). The primary interfaces to the Arm Cortex-A53 MPCore provide a read and write datapath and support for debug, power management and interrupts.

- Requests from the Cortex-A53 MPCore processor are sent to the cache coherency unit (CCU) by the 128-bit ACE bus master. The CCU supports memory read and write requests and I/O memory-mapped read and write requests. The CCU allows masters to maintain I/O coherency with the Cortex-A53 MPCore subsystem.
- The System MMU (SMMU) resides outside of the Cortex-A53 MPCore. It consists of a translation control unit (TCU) which controls and manages the address translations of each master's translation buffer unit (TBU). The TLB data of the Cortex-A53 MPCore is managed by the SMMU.
- The debug access port (DAP) interfaces directly to the processor and can perform invasive or non-invasive debug.
- The Generic Interrupt Controller (GIC) resides outside of the Cortex-A53 MPCore and sends interrupt requests to the processor through a dedicated bus.

Related Information

- System Memory Management Unit on page 71
- Cache Coherency Unit on page 60
- Generic Interrupt Controller on page 48
- System Interconnect on page 82

3.5. Cortex-A53 MPCore Functional Description

The Arm Cortex-A53 MPCore is a 64-bit processor that implements the Armv8-A architecture. The Cortex-A53 MPCore processor has four cores with an L1 memory system and a single, shared L2 cache.

Table 29. Arm Cortex-A53 MPCore Processor Configuration

This table shows the parameters for the Arm Cortex-A53 MPCore Processor.

Feature	Configuration
Armv8-A architecture, Cortex-A53 CPUs	4
Instruction cache size per CPU	32 KB, 2-way set associative with a line size of 64 bytes per line
Data cache size per CPU	32 KB, 4-way set associative with a line size of 64 bytes per line
L2 cache size shared among four CPUs	1 MB, 16-way set associative with a line size of 64 bytes per line
Media Processing Engine with NEON technology in each CPU	Included with support for floating-point operations
Armv8-A cryptographic extensions in each CPU	Included
Embedded Trace Macrocell (ETMv4) in each CPU	Included
Cache protection	Included for L1 and L2 cache. See "Cache Protection" section for more information.

Related Information

Cache Protection on page 46

3.5.1. Exception Levels

The Cortex-A53 MPCore CPUs support 4 exception levels:

- EL0 has the lowest software execution privilege, and execution in EL0 is called unprivileged execution. This execution level may be used for application software.
- EL1 provides support for operating systems.
- EL2 provides support for processor virtualization or hypervisor mode.
- EL3 provides support for the secure monitor.

As the exception level increases from 1 to 3, the software execution privilege increases.

3.5.1.1. Security State

The Arm Cortex-A53 CPUs provide the following security states, each with an associated memory address space:

- Secure state:
  - The processor can access both the secure memory address space and the non-secure memory address space.
  - When executing at EL3, the processor can access all the system control resources.
- Non-secure state:
  - The processor can access only the non-secure memory address space.
  - The processor cannot access the secure system control resources.

Depending on the security state, only certain exception levels are allowed.

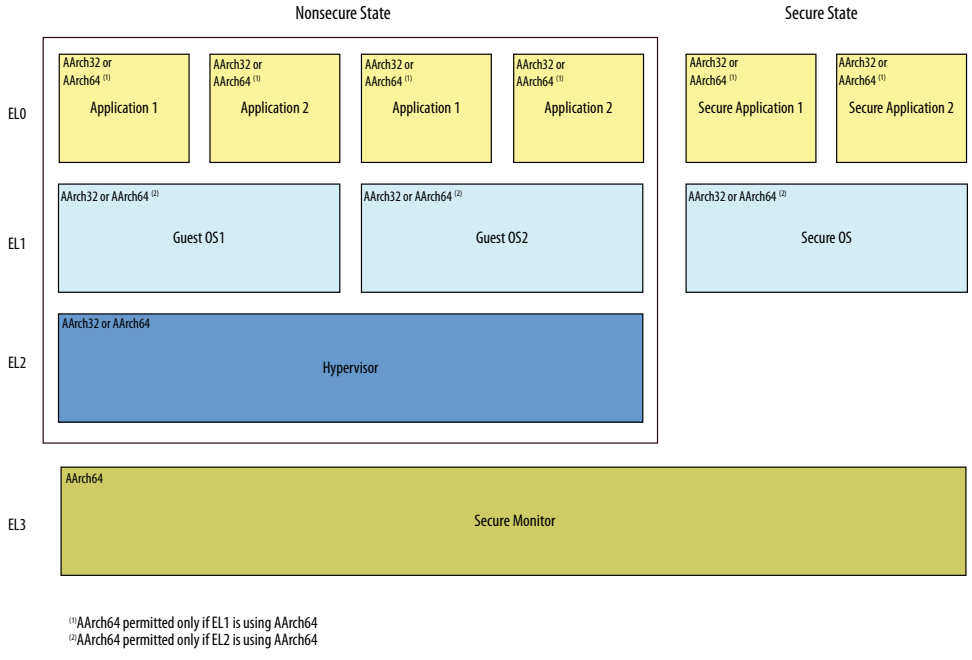
Table 30. Exception Level Implementation by Security State

Exception Level	Non-secure State	Secure State
EL0	Yes	Yes
EL1	Yes	Yes
EL2	Yes	No
EL3	No	Yes

3.5.1.2. Security Model

The Arm Cortex-A53 processor implements all of the exception levels. The EL3 exists only in the secure state and a change from the secure state to the non-secure state is made only by an exception return from EL3. EL2 exists only in non-secure state.

Figure 3. Security Model when EL3 is using AArch64



3.5.2. Virtualization

EL2 supports virtualization of the non-secure state. A virtualized system typically includes:

- A hypervisor, running in EL2, that is responsible for switching between virtual machines. A virtual machine is comprised of non-secure EL1 and non-secure EL0.
- A number of guest operating systems, that each run in non-secure EL1, on a virtual machine
- For each guest operating system, applications that usually run in non-secure EL0 on a virtual machine

Note: The Cortex-A53 MPCore processor supports systems where the guest OS is unaware that it or any other guest OS is running on a virtual machine and systems where the guest OS is aware it is running on a virtual machine with other guest OSs.

The hypervisor assigns a virtual machine identifier (VMID) to each virtual machine. For guest OS management, EL2 is implemented only in non-secure state. EL2 provides controls to:

- Virtual values of a small number of identification registers. A read of one of these registers by a guest OS or the applications for a guest OS returns the virtual value.
- Trap various operations, including memory management operations and accesses other registers. A trapped operation generates an exception that is taken to EL2.
- Route interrupts to:
  - The current guest OS
  - A guest OS that is not currently running
  - The hypervisor

In the non-secure state, an independent translation regime exists for memory accesses from EL2. For the EL0 and EL1 translation regime, address translation occurs in two stages:

- Stage 1 maps the virtual address (VA) to an intermediate physical address (IPA). This translation is managed at EL1, usually by a guest OS. The guest OS believes that the IPA is the physical address (PA).
- Stage 2 maps the IPA to the PA. This translation is managed at EL2. The guest OS might be completely unaware of this stage. For more information on the translation regimes, see the *System Memory Management Unit* chapter.

EL2 implements the following exceptions:

- Hypervisor call (HVC) exception
- Traps to EL2
- All of the virtual interrupts:
  - Virtual SError
  - Virtual IRQ
  - Virtual FIQ

HVC exceptions are always taken to EL2. All virtual interrupts are always taken to EL1, and can only be taken from the non-secure EL1 or EL0. You can independently enable each of the virtual interrupts using controls at EL2.

The Cortex-A53 MPCore processor contains virtualization registers that allow you to configure translation tables, hypervisor operations, exception levels, and virtual interrupts, For more information, please refer to the *Arm Cortex-A53 MPCore Processor Technical Reference Manual*.

Related Information

- System Memory Management Unit on page 71
- Arm Cortex-A53 MPCore Technical Reference Manual, Revision: r0p4

3.5.2.1. Virtual Interrupts

Each virtual interrupt corresponds to a physical interrupt.

When a virtual interrupt is enabled, its corresponding physical exception is taken to EL2, unless EL3 has configured that physical exception to be taken to EL3.

Table 31. Virtual Interrupts

Physical Interrupt	Corresponding Virtual Interrupt
SError	Virtual SError
IRQ	Virtual IRQ
FIQ	Virtual FIQ

Software executing in EL2 can use virtual interrupts to signal physical interrupts to non-secure EL1 and non-secure EL0.

An example of a virtual interrupt model follows:

1. Software executing at EL2 routes a physical interrupt to EL2.
2. When a physical interrupt of that type occurs, the exception handler executing in EL2 determines whether the interrupt can be handled in EL2 or requires routing to a guest OS in EL1. If an interrupt requires routing to a guest OS and the guest OS is running, the hypervisor asserts the appropriate virtual interrupt to signal the physical interrupt to the guest OS. If the guest OS is not running, the physical interrupt is marked as pending for the guest OS. When the hypervisor next switches to the virtual machine that is running that guest OS, the hypervisor uses the appropriate virtual interrupt type to signal the physical interrupt to the guest OS.

A hypervisor can prevent non-secure EL1 and non-secure EL0 from distinguishing a virtual interrupt from a physical interrupt.

3.5.3. Memory Management Unit

Each CPU of the Cortex-A53 MPCore contains a memory management unit (MMU) that translates virtual addresses to physical addresses. Address mappings and memory attributes are held in page tables that are loaded into the translation lookaside buffer (TLB) when a location is accessed.

The MMUs support 40-bit physical address size and two stages of translation. You can enable or disable each stage of address translation independently.

The Cortex-A53 MPCore communicates with the system memory management unit (SMMU) when pages are invalidated in a CPU's MMU.

For more information regarding the SMMU, refer to the *System Memory Management Unit* chapter.

Related Information

System Memory Management Unit on page 71

3.5.3.1. Translation Lookaside Buffers

Each CPU in the Cortex-A53 MPCore contains micro and main translation lookaside buffers (TLBs).

Table 32. MMU Features of Each CPU in the Cortex-A53 MPCore

TLB Type	Memory Type	Number of Entries	Associativity
Micro TLB	Instruction	10	Fully associative
Micro TLB	Data	10	Fully associative
Main TLB	Instruction and Data	512	Four-way set-associative

Each CPU also includes:

- 4-way set associative 64-entry walk cache that holds the result of a stage 1 translation. The walk cache holds entries fetched from the secure and non-secure state.
- 4-way set associative 64-entry intermediate physical address (IPA) cache. This cache holds map points between intermediate physical addresses and physical addresses. Only non-secure exception level 1 (EL1) and exception level 0 (EL0) stage 2 translations use this cache.

TLB entries include global and application specific identifiers to prevent context switch TLB flushes. The architecture also supports virtual machine identifiers (VMIDs) to prevent TLB flushes on virtual machine switches by hypervisor.

The micro TLBs are the first level of caching for the translation table information. The unified main TLB handles misses from the micro TLBs.

When the main TLB performs maintenance operations it flushes both the instruction and data micro TLBs.

3.5.3.2. Translation Match Process

The ARMv8-A architecture supports multiple mappings of the virtual address space, which are translated differently. The TLB entries store all the required context information to facilitate a match and avoid a TLB flush or a context or virtual machine switch.

Each TLB entry contains a virtual address, block size, physical address, and a set of memory properties that include the memory type and access permissions. Each entry is associated with a particular application space ID (ASID), or is global for all application spaces.

The TLB entry also contains a field to store the virtual memory ID (VMID) for accesses made from the non-secure EL0 and EL1. A memory space identifier in the TLB entry records whether the request occurred at the:

- EL3, if EL3 is in the AArch64 execution state
- Non-secure EL2 exception level
- Secure and non-secure EL0 or EL1 and EL3, when EL3 is in AArch32 execution state

A TLB entry match occurs when:

- The virtual address matches that of the requested address
- The memory space matches the memory space state of the requests. The memory space can be one of four values:
  - Secure EL3, when EL3 is in the AArch64 execution state
  - Non-secure EL2
  - Secure EL0 or EL1, and EL3 when EL3 is in the AArch32 execution state
  - Non-secure EL0 or EL1
- The ASID matches the current application space ID held in the `CONTEXTIDR`, `TTBR0`, or `TTBR1` register or the entry is marked global.
- The VMID matches the current VMID held in the VTTBR register.

Note: For a request originating from EL2 or AArch64 EL3, the ASID and VMID match are ignored.

Note: For a request not originating from non-secure EL0 or EL1, the VMID match is ignored.

3.5.4. Level 1 Caches

3.5.4.1. Instruction Cache

The instruction cache is 2-way set associative. The instruction cache provides parity checking to detect single-bit errors. If an error is detected, the line is invalidated and fetched again. The instruction cache is designed to reduce instruction fetching latency by implementing prefetch and branch prediction logic.

- Instruction fetches are sequential
- A two-instruction transparent target instruction cache and 256-entry branch target address cache provides reduced branch latency
- An 8-entry return stack accelerates branch returns
- The read interface to the 1 MB L2 cache is 128-bits wide

A cache line is 64 bytes and only holds one instruction type. Different instruction types cannot be mixed in the same cache line.

Each cache line can hold the following:

- 16—A32 instructions
- 16—32-bit T32 instructions
- 16—A64 instructions
- 32—16-bit T32 instructions

The instruction cache supports single error detection (SED) parity checking.

3.5.4.2. Data Cache

The data cache is 4-way set associative with a cache line length of 64 bytes. It is organized as a physically indexed and physically tagged cache. The micro TLB for the data cache converts virtual addresses to physical addresses before it executes a cache access.

- Supports 256-bit writes and 128-bit reads to L2 cache
- Utilizes prefetch engine and read buffer
- Supports three outstanding data cache misses
- Provides error checking and correction (ECC) on L1 data and parity checking on control bits

3.5.4.2.1. ACE Transactions

The L1 ACE Data Transactions that are supported are listed in the table below Refer to the *Arm Cortex-A53 MPCore Processor Technical Reference Manual* for more details.

Table 33. L1 ACE Data Transactions

Attribute		ACE Transaction				
Memory Type	Shareability	Domain	Load	Store	Load Exclusive	Store Exclusive
Device	N/A	System	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop and ARLOCKM set to HIGH	WriteNoSnoop and AWLOCKM set to HIGH
Normal, inner Non-cacheable, outer Non-cacheable	Non-shared	System	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop and ARLOCKM set to HIGH	WriteNoSnoop and AWLOCKM set to HIGH
	Inner-shared					
	Outer-shared					
Normal, inner Non-cacheable, outer Write-Back or Write-Through	Non-shared	System	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop	ReadNoSnoop
	Inner-shared	System	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop and ARLOCKM set to HIGH	WriteNoSnoop and AWLOCKM set to HIGH
	Outer-shared					
Normal, inner Write-Through, outer Write-Back, Write-Through	Non-shared	System	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop	ReadNoSnoop
	Inner-shared	System	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop and ARLOCKM set to HIGH	WriteNoSnoop and AWLOCKM set to HIGH
	Outer-shared					
Non-cacheable, or Normal inner Write-Back outer Non-cacheable or Write-Through	Non-shared	System	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop	ReadNoSnoop
	Inner-shared	System	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop with ARLOCKM set to HIGH	WriteNoSnoop with ARLOCKM set to HIGH
	Outer-shared					

continued...

Attribute	ACE Transaction					
Memory Type	Shareability	Domain	Load	Store	Load Exclusive	Store Exclusive
Normal, inner Write-Back, outer Write-Back	Non-shared	Non-shareable	ReadNoSnoop	WriteNoSnoop	ReadNoSnoop	WriteNoSnoop
	Inner-shared	Inner Shareable	ReadShared	ReadUnique or CleanUnique if required, then a WriteBack when the line is evicted	ReadShared with ARLOCKM set to HIGH	CleanUnique with ARLOCKM set to HIGH if required, then a WriteBack when the line is evicted
	Outer-shared	Outer Shareable				

Note: It is recommended that no load or store instructions are placed between the exclusive load and the exclusive store because these additional instructions can cause a cache eviction. Any data cache maintenance instruction can also clear the exclusive monitor.

Related Information

Arm Cortex-A53 MPCore Technical Reference Manual, Revision: r0p4

3.5.4.2.2. Data Prefetching

The prefetcher can detect cache misses and begin linefills in the background.

The prefetcher is enabled by default at reset. You may configure the sequence length that triggers the prefetcher or the number of outstanding requests the prefetcher can make by programming the CPU Auxiliary Control (CPUACTLR) register.

3.5.4.3. Initializing the Instruction and Data Caches

Follow these steps for each execution level (EL) that you use in the system.

1. Enable data coherency by setting the SMPEN bit in the CPU Extended Control Register (CPUTECTR).
2. Invalidate all instruction cache entries by setting the I bit in the System Control Registers (SCTLR_ELx), where x indicates the execution level.
3. Invalidate all entries in the data cache.
4. Invalidate the TLB contents.
5. Configure the CPU MMUs (the data cache is not active unless the MMUs are also active).
   a. Configure the page tables.
   b. Configure the Translation Table Base Registers (TTBR) for each execution level.
6. Set the M bit in the SCTRL_ELx register to enable the MMU.
7. Set the C bit in the SCTRL_ELx register to enable the data cache.

Related Information

- Cortex-A Series Programmer's Guide for Armv8-A
  For more information about the Armv8-A Registers
- Arm Cortex-A53 MPCore Technical Reference Manual, Revision: r0p4

3.5.5. Level 2 Memory System

- 1 MB L2 cache, shared among four processors
  - 16-way set associative cache structure
  - 64 bytes per line
- Snoop Control Unit (SCU) that provides data coherency and ECC protection
- Interfaces to system through a 128-bit AMBA 4 ACE bus

3.5.6. Snoop Control Unit

The Snoop Control Unit (SCU) maintains L1 data cache coherency between the four CPUs within the Cortex-A53 MPCore processor.

- 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.
- The SCU reduces latency by using buffers to execute cache-to-cache transfers between CPUs without accessing external memory.
- The SCU can accept up to eight requests from the system.

The SCU communicates with the system-level cache coherency unit (CCU) to maintain coherency between the two modules.

3.5.6.1. 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.

If multiple CPUs attempt simultaneous access to the L2 cache, the SCU arbitrates among them.

3.5.7. Cryptographic Extensions

Cryptographic functions are an extension of the SIMD support and operate on the vector register file. This extension provides instructions for the acceleration of encryption and decryption to support:

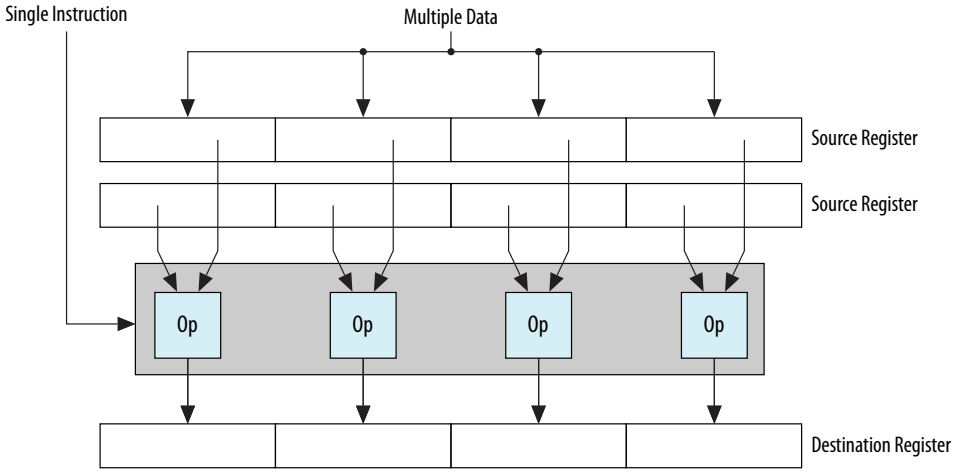
- AES
- SHA1
- SHA2-256

The cryptographic extension also provides multiply instructions that operate on long polynomials.

3.5.8. 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.

3.5.8.1. Single Instruction, Multiple Data (SIMD) Processing



3.5.8.2. 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-A53 MPCore 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

3.5.9. Floating Point Unit

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.

3.5.10. ACE Bus Interface

The ACE bus interface operates in the `mpu_ccu_clk` domain, which is `mpu_clk/2`. This bus provides an AXI3 compatibility mode with support for privilege level accesses through the `ARPROTM[0]` and `AWPROTM[0]` signals.

The Cortex-A53 MPCore processor does not generate any FIXED bursts and all WRAP bursts fetch a complete cache line starting with the critical word first. A burst does not cross a cache line boundary. The cache linefill fetch length is always 64 bytes. The Cortex-A53 generates only a subset of all possible AXI transactions on the master interface.

For WriteBack transfers the supported transfers are:

- WRAP 4 128-bit for read transfers (linefills).
- INCR 4 128-bit for write transfers (evictions).
- INCR N (N:1, 2, or 4) 128-bit write transfers (read allocate).

For non-cacheable transactions:

- INCR N (N:1, 2, or 4) 128-bit for write transfers.
- INCR N (N:1, 2, or 4) 128-bit for read transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit for read transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit for write transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit for exclusive write transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit for exclusive read transfers.

For Device transactions:

- INCR N (N:1, 2, or 4) 128-bit read transfers.
- INCR N (N:1, 2, or 4) 128-bit write transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit read transfers.

- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit write transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit exclusive read transfers.
- INCR 1 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit exclusive write transfers.

For translation table walk transactions INCR 1 32-bit, and 64-bit read transfers.

The following characteristics apply to AXI transactions:

- WRAP bursts are only 128-bit.
- INCR 1 can be any size for read or write.
- INCR burst, more than one transfer, are only 128-bit.
- No transaction is marked as FIXED.
- Write transfers with all, some or no byte strobes HIGH can occur.

3.5.11. Abort Handling

The following list details items you should take into consideration about abort handling.

- All load accesses synchronously abort.
- All STREX, STREXB, STREXH, STREXD, STXR, STXRB, STXRH, STXP, STLXR, STLXRB, STLXRH and STLXP instructions use the synchronous abort mechanism.
- All store accesses to device memory, or normal memory that is inner non-cacheable, inner write-through, outer non-cacheable, or outer write-through use the asynchronous abort mechanism, except for STREX, STREXB, STREXH, STREXD, STXR, STXRB, STXRH, STXP, STLXR, STLXRB, STLXRH, and STLXP.
- All store accesses to normal memory that is both inner cacheable and outer cacheable and any evictions from L1 or L2 cache do not cause an abort in the processor. Instead, an nEXTERRIRQ interrupt is asserted because the access that aborts might not relate directly back to a specific CPU in the cluster.
- L2 linefills triggered by an L1 Instruction fetch assert the nEXTERRIRQ interrupt if the data is received from the interconnect in a dirty state. Instruction data can be marked as dirty as a result of self-modifying code or a line containing a mixture of data and instructions. If an error response is received on any part of the line, the dirty data might be lost.

Note: When nEXTERRIRQ is asserted it remains asserted until the error is cleared by a write of 0 to the AXI asynchronous error bit of the L2ECTLR register.

3.5.12. Cache Protection

The L1 instruction cache provides parity checking with single error detection (SED). Double bit errors are not detected or corrected.

The L1 data cache and L2 cache provide single error correction and double error detection (SECDED). If a single-bit error is detected, the access that caused the error is stalled while the correction takes place. After correction, the access that was stalled continues or is retried.

Correction behavior varies depending on RAM type.

Table 34. Cache Protection Behavior

RAM	Protection Type	Protection Granule	Correction Behavior
L1 Instruction cache tag	Parity, SED	31 bits	Both lines in the cache set are invalidated, and then the line requested is refetched from L2 cache or external memory.
L1 Instruction cache data	Parity, SED	20 bits	Both lines in the cache set are invalidated, and then the line requested is refetched from L2 cache or external memory.
TLB	Parity, SED	52 bits	The entry is invalidated, and a new pagewalk is started to refetch it.
L1 Data cache tag	Parity, SED	32 bits	The line is cleaned and invalidated from the L1 cache. SCU duplicate tags are used to get the correct address. The line is refetched from L2 cache or external memory.
L1 Data cache data	ECC, SECDED	32 bits	The line is cleaned and invalidated from the L1 cache, with single bit errors corrected as part of the eviction. The line is refetched from L2 cache or external memory.
L1 data cache dirty bit	Parity, SED with correction by re-loading data	1 bit	The line is cleaned and invalidated from the L1 cache, with detection of dirty bit corruption through parity checking. Only the dirty bit is protected. The other bits are performance hints, therefore do not cause a functional failure if they are incorrect. Error is corrected by reloading the data.
SCU L1 duplicate tag	ECC, SECDED	33 bits	The tag is rewritten with the correct value, and the access is retried. If the error is uncorrectable then the tag is invalidated.
L2 tag	ECC, SECDED	33 bits	The tag is rewritten with the correct value, and the access is retried. If the error is uncorrectable then the tag is invalidated.
L2 data	ECC, SECDED	64 bits	Data is corrected inline, and the access might stall for an additional cycle or two while the correction takes place. After correction, the line might be evicted from the processor.

3.5.12.1. Error Reporting

Detected errors are reported in the CPUMERRSR or L2MERRSR registers and also signaled on the PMUEVENT bus. Detected errors include errors that are successfully corrected, and those that cannot be corrected. If multiple errors occur on the same clock cycle then only one of them is reported.

Errors that cannot be corrected, and therefore might result in data corruption, also cause an abort. Your software can register this error and can either attempt to recover or can restart the system.

When an L1 data or L2 dirty cache line with an error on the data RAMs is evicted from the processor, the write on the master interface still takes place. However, if the error is uncorrectable, then the incorrect data is not written externally.

3.5.13. Generic Interrupt Controller

The Arm Generic Interrupt Controller (GIC-400) resides within the system complex outside of the Cortex-A53 processor. The GIC is shared by all of the Cortex-A53 CPUs.

The GIC has software-configurable settings to detect, manage and distribute interrupts in the SoC.

- Interrupts are enabled or disabled and prioritized through control registers.
- Interrupts can be prioritized and signaled to different processors.
- You can configure interrupts as secure or non-secure by assigning them to group 0 or group1, respectively.
- Virtualization extensions within the GIC allow you to manage virtualized interrupts.

The GIC provides 205 shared interrupt sources, including dedicated peripherals and IP implemented in the FPGA fabric. Each CPU also has six external private peripheral interrupts and one internal private peripheral interrupt. All four CPUs share 16 banked software-generated interrupts (SGIs).

Note: Legacy IRQs are not supported by the GIC.

The GIC detects private peripheral interrupts (PPIs) and shared peripheral interrupts (SPIs) from interrupt signals. Software-generated interrupts are detected through the register interface. Each CPU generates a signal for every private peripheral interrupt ID (PPI ID). There is only one input signal for each SPI interrupt ID shared among the four CPUs. The GIC supports virtual interrupts as well.

The GIC notifies each CPU of an interrupt or virtual interrupt through output signals sent to the Cortex-A53 MPCore.

The configuration and control for the GIC is memory-mapped and accessed through the cache coherency unit (CCU).

3.5.13.1. GIC Clock

The GIC operates in the `mpu_periph_clk` domain which is `mpu_clk/4`.

3.5.13.2. GIC Reset

The GIC is reset on a cold or warm reset. All interrupt configurations are cleared upon a cold or warm reset.

3.5.13.3. GIC Interrupt Map for the 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 35. GIC Interrupt Map

GIC Interrupt Number	Source Block	Interrupt Name	Description
32	Secure Device Manager (SDM)	SDM_IRQ0 (mailbox_intr)	SDM Mailbox Interrupt
35	Secure Device Manager (SDM)	SDM_IRQ3 (sdm_qspi_intr)	SDM Quad SPI Interrupt
37	Secure Device Manager (SDM)	SDM_IRQ5 (sdm_sdmmc_irq)	SDM SD/MMC Interrupt
47	System Manager	SERR_Global	Global System Error Interrupt
48	CCU	interrupt_ccu	CCU Combined Interrupt
49	FPGA	F2S_FPGA_IRQ0	F2S FPGA Interrupt 0
50	FPGA	F2S_FPGA_IRQ1	F2S FPGA Interrupt 1
51	FPGA	F2S_FPGA_IRQ2	F2S FPGA Interrupt 2
52	FPGA	F2S_FPGA_IRQ3	F2S FPGA Interrupt 3
53	FPGA	F2S_FPGA_IRQ4	F2S FPGA Interrupt 4
54	FPGA	F2S_FPGA_IRQ5	F2S FPGA Interrupt 5
55	FPGA	F2S_FPGA_IRQ6	F2S FPGA Interrupt 6
56	FPGA	F2S_FPGA_IRQ7	F2S FPGA Interrupt 7
57	FPGA	F2S_FPGA_IRQ8	F2S FPGA Interrupt 8
58	FPGA	F2S_FPGA_IRQ9	F2S FPGA Interrupt 9
59	FPGA	F2S_FPGA_IRQ10	F2S FPGA Interrupt 10
60	FPGA	F2S_FPGA_IRQ11	F2S FPGA Interrupt 11
61	FPGA	F2S_FPGA_IRQ12	F2S FPGA Interrupt 12
62	FPGA	F2S_FPGA_IRQ13	F2S FPGA Interrupt 13
63	FPGA	F2S_FPGA_IRQ14	F2S FPGA Interrupt 14
64	FPGA	F2S_FPGA_IRQ15	F2S FPGA Interrupt 15
65	FPGA	F2S_FPGA_IRQ16	F2S FPGA Interrupt 16
66	FPGA	F2S_FPGA_IRQ17	F2S FPGA Interrupt 17
67	FPGA	F2S_FPGA_IRQ18	F2S FPGA Interrupt 18
68	FPGA	F2S_FPGA_IRQ19	F2S FPGA Interrupt 19
69	FPGA	F2S_FPGA_IRQ20	F2S FPGA Interrupt 20
70	FPGA	F2S_FPGA_IRQ21	F2S FPGA Interrupt 21
71	FPGA	F2S_FPGA_IRQ22	F2S FPGA Interrupt 22
72	FPGA	F2S_FPGA_IRQ23	F2S FPGA Interrupt 23
73	FPGA	F2S_FPGA_IRQ24	F2S FPGA Interrupt 24
74	FPGA	F2S_FPGA_IRQ25	F2S FPGA Interrupt 25

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Description
75	FPGA	F2S_FPGA_IRQ26	F2S FPGA Interrupt 26
76	FPGA	F2S_FPGA_IRQ27	F2S FPGA Interrupt 27
77	FPGA	F2S_FPGA_IRQ28	F2S FPGA Interrupt 28
78	FPGA	F2S_FPGA_IRQ29	F2S FPGA Interrupt 29
79	FPGA	F2S_FPGA_IRQ30	F2S FPGA Interrupt 30
80	FPGA	F2S_FPGA_IRQ31	F2S FPGA Interrupt 31
81	FPGA	F2S_FPGA_IRQ32	F2S FPGA Interrupt 32
82	FPGA	F2S_FPGA_IRQ33	F2S FPGA Interrupt 33
83	FPGA	F2S_FPGA_IRQ34	F2S FPGA Interrupt 34
84	FPGA	F2S_FPGA_IRQ35	F2S FPGA Interrupt 35
85	FPGA	F2S_FPGA_IRQ36	F2S FPGA Interrupt 36
86	FPGA	F2S_FPGA_IRQ37	F2S FPGA Interrupt 37
87	FPGA	F2S_FPGA_IRQ38	F2S FPGA Interrupt 38
88	FPGA	F2S_FPGA_IRQ39	F2S FPGA Interrupt 39
89	FPGA	F2S_FPGA_IRQ40	F2S FPGA Interrupt 40
90	FPGA	F2S_FPGA_IRQ41	F2S FPGA Interrupt 41
91	FPGA	F2S_FPGA_IRQ42	F2S FPGA Interrupt 42
92	FPGA	F2S_FPGA_IRQ43	F2S FPGA Interrupt 43
93	FPGA	F2S_FPGA_IRQ44	F2S FPGA Interrupt 44
94	FPGA	F2S_FPGA_IRQ45	F2S FPGA Interrupt 45
95	FPGA	F2S_FPGA_IRQ46	F2S FPGA Interrupt 46
96	FPGA	F2S_FPGA_IRQ47	F2S FPGA Interrupt 47
97	FPGA	F2S_FPGA_IRQ48	F2S FPGA Interrupt 48
98	FPGA	F2S_FPGA_IRQ49	F2S FPGA Interrupt 49
99	FPGA	F2S_FPGA_IRQ50	F2S FPGA Interrupt 50
100	FPGA	F2S_FPGA_IRQ51	F2S FPGA Interrupt 51
101	FPGA	F2S_FPGA_IRQ52	F2S FPGA Interrupt 52
102	FPGA	F2S_FPGA_IRQ53	F2S FPGA Interrupt 53
103	FPGA	F2S_FPGA_IRQ54	F2S FPGA Interrupt 54
104	FPGA	F2S_FPGA_IRQ55	F2S FPGA Interrupt 55
105	FPGA	F2S_FPGA_IRQ56	F2S FPGA Interrupt 56
106	FPGA	F2S_FPGA_IRQ57	F2S FPGA Interrupt 57
107	FPGA	F2S_FPGA_IRQ58	F2S FPGA Interrupt 58

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Description
108	FPGA	F2S_FPGA_IRQ59	F2S FPGA Interrupt 59
109	FPGA	F2S_FPGA_IRQ60	F2S FPGA Interrupt 60
110	FPGA	F2S_FPGA_IRQ61	F2S FPGA Interrupt 61
111	FPGA	F2S_FPGA_IRQ62	F2S FPGA Interrupt 62
112	FPGA	F2S_FPGA_IRQ63	F2S FPGA Interrupt 63
113	DMA	dma_IRQ0	DMA Interrupt 0
114	DMA	dma_IRQ1	DMA Interrupt 1
115	DMA	dma_IRQ2	DMA Interrupt 2
116	DMA	dma_IRQ3	DMA Interrupt 3
117	DMA	dma_IRQ4	DMA Interrupt 4
118	DMA	dma_IRQ5	DMA Interrupt 5
119	DMA	dma_IRQ6	DMA Interrupt 6
120	DMA	dma_IRQ7	DMA Interrupt 7
121	DMA	dma_irq_abort	DMA Abort Interrupt
122	EMAC0	emac0_IRQ	EMAC0 Interrupt
123	EMAC1	emac1_IRQ	EMAC1 Interrupt
124	EMAC2	emac2_IRQ	EMAC2 Interrupt
125	USB0	usb0_IRQ	USB0 Interrupt
126	USB1	usb1_IRQ	USB1 Interrupt
127	MPFE	HMC_error	DDR4 Protocol Error Interrupt
128	SDMMC	sdmmc_IRQ	SD/MMC Interrupt
129	NAND	nand_IRQ	NAND Interrupt
130	Reserved	Reserved	-
131	SPI0 master	spim0_IRQ	SPI0 Master Interrupt
132	SPI1 master	spim1_IRQ	SPI1 Master Interrupt
133	SPI0 slave	spis0_IRQ	SPI0 Slave Interrupt
134	SPI1 slave	spis1_IRQ	SPI1 Slave Interrupt
135	I^2C0	i2c0_IRQ	I^2C0 Interrupt
136	I^2C1	i2c1_IRQ	I^2C1 Interrupt
137	I^2C2	i2c2_IRQ	I^2C2 Interrupt (I^2C2 can be used with EMAC0)
138	I^2C3	i2c3_IRQ	I^2C3 Interrupt (I^2C3 can be used with EMAC1)
139	I^2C4	i2c4_IRQ	I^2C4 Interrupt (I^2C4 can be used with EMAC2)
140	UART0	uart0_IRQ	UART0 Interrupt

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Description
141	UART1	uart1_IRQ	UART1 Interrupt
142	GPIO0	gpio0_IRQ	GPIO 0 Interrupt
143	GPIO1	gpio1_IRQ	GPIO 1 Interrupt
144	Reserved	-	-
145	Timer0	timer_l4sp_0_IRQ	Timer0 Interrupt
146	Timer1	timer_l4sp_1_IRQ	Timer1 Interrupt
147	Timer2	timer_osc1_0_IRQ	Timer 2 Interrupt
148	Timer3	timer_osc1_1_IRQ	Timer 3 Interrupt
149	Watchdog0	wdog0_IRQ	Watchdog0 Interrupt
150	Watchdog1	wdog1_IRQ	Watchdog1 Interrupt
151	Clock Manager	clkmgr_IRQ	Clock Manager Interrupt
152	SDRAM MPFE	seq2core	Calibration Interrupt
153	CoreSight CPU0 CTI	CTIIRQ[0]	Cortex-A53 MPCore Processor CPU 0 Cross Trigger Interface Interrupt
154	CoreSight CPU1 CTI	CTIIRQ[1]	Cortex-A53 MPCore Processor CPU 1 Cross Trigger Interface Interrupt
155	CoreSight CPU2 CTI	CTIIRQ[2]	Cortex-A53 MPCore Processor CPU 2 Cross Trigger Interface Interrupt
156	CoreSight CPU3 CTI	CTIIRQ[3]	Cortex-A53 MPCore Processor CPU 3 Cross Trigger Interface Interrupt
157	Watchdog2	wdog2_IRQ	Watchdog 2 Interrupt
158	Watchdog3	wdog3_IRQ	Wathdog 3 Interrupt
159	Cortex-A53	nEXTERRIRQ	Cortex-A53 MPCore External Error Interrupt
160	System MMU	gbl_flt_irpt_s	Global Secure Fault Interrupt
161	System MMU	gbl_flt_irpt_ns	Global Non-secure Fault Interrupt
162	System MMU	perf_irpt_FPGA_TBU	FPGA TBU Performance Counter Interrupt
163	System MMU	perf_irpt_DMA_TBU	DMA TBU Performance Counter Interrupt
164	System MMU	perf_irpt_EMAC_TBU	EMAC TBU Performance Counter Interrupt
165	System MMU	perf_irpt_IO_TBU	Peripheral I/O Master TBU Performance Counter Interrupt
167	Reserved	Reserved	-
168	System MMU	comb_irpt_ns	System MMU Combined Non-secure Interrupt
169	System MMU	comb_irpt_s	System MMU Combined Secure Interrupt
170	System MMU	cxt_irpt_0	System MMU Non-secure Context Interrupt 0
171	System MMU	cxt_irpt_1	System MMU Non-secure Context 1 Interrupt

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Description
172	System MMU	cxt_irpt_2	System MMU Non-secure Context 2 Interrupt
173	System MMU	cxt_irpt_3	System MMU Non-secure Context 3 Interrupt
174	System MMU	cxt_irpt_4	System MMU Non-secure Context 4 Interrupt
175	System MMU	cxt_irpt_5	System MMU Non-secure Context 5 Interrupt
176	System MMU	cxt_irpt_6	System MMU Non-secure Context 6 Interrupt
177	System MMU	cxt_irpt_7	System MMU Non-secure Context 7 Interrupt
178	System MMU	cxt_irpt_8	System MMU Non-secure Context 8 Interrupt
179	System MMU	cxt_irpt_9	System MMU Non-secure Context 9 Interrupt
180	System MMU	cxt_irpt_10	System MMU Non-secure Context 10 Interrupt
181	System MMU	cxt_irpt_11	System MMU Non-secure Context 11 Interrupt
182	System MMU	cxt_irpt_12	System MMU Non-secure Context 12 Interrupt
183	System MMU	cxt_irpt_13	System MMU Non-secure Context 13 Interrupt
184	System MMU	cxt_irpt_14	System MMU Non-secure Context 14 Interrupt
185	System MMU	cxt_irpt_15	System MMU Non-secure Context 15 Interrupt
186	System MMU	cxt_irpt_16	System MMU Non-secure Context 16 Interrupt
187	System MMU	cxt_irpt_17	System MMU Non-secure Context 17 Interrupt
188	System MMU	cxt_irpt_18	System MMU Non-secure Context 18 Interrupt
189	System MMU	cxt_irpt_19	System MMU Non-secure Context 19 Interrupt
190	System MMU	cxt_irpt_20	System MMU Non-secure Context 20 Interrupt
191	System MMU	cxt_irpt_21	System MMU Non-secure Context 21 Interrupt
192	System MMU	cxt_irpt_22	System MMU Non-secure Context 22 Interrupt
193	System MMU	cxt_irpt_23	System MMU Non-secure Context 23 Interrupt
194	System MMU	cxt_irpt_24	System MMU Non-secure Context 24 Interrupt
195	System MMU	cxt_irpt_25	System MMU Non-secure Context 25 Interrupt
196	System MMU	cxt_irpt_26	System MMU Non-secure Context 26 Interrupt
197	System MMU	cxt_irpt_27	System MMU Non-secure Context 27 Interrupt
198	System MMU	cxt_irpt_28	System MMU Non-secure Context 28 Interrupt
199	System MMU	cxt_irpt_29	System MMU Non-secure Context 29 Interrupt
200	System MMU	cxt_irpt_30	System MMU Non-secure Context 30 Interrupt
201	System MMU	cxt_irpt_31	System MMU Non-secure Context 31 Interrupt
202	Cortex-A53	nPMUIRQ[0]	Cortex-A53 Processor CPU 0 Performance Monitor Interrupt

continued...

GIC Interrupt Number	Source Block	Interrupt Name	Description
203	Cortex-A53	nPMUIRQ[1]	Cortex-A53 Processor CPU 1 Performance Monitor Interrupt
204	Cortex-A53	nPMUIRQ[2]	Cortex-A53 Processor CPU 2 Performance Monitor Interrupt
205	Cortex-A53	nPMUIRQ[3]	Cortex-A53 Processor CPU 3 Performance Monitor Interrupt

3.5.14. Generic Timers

The Arm Cortex-A53 MPCore provides a generic timer within each CPU.

The generic timer of each CPU contains a set of timer registers to capture a variety of events:

- non-secure physical events
- secure physical events
- physical events
- virtual events

The four timers provided in each CPU are:

- EL1 non-secure physical timer register
- EL1 secure physical timer register
- EL2 virtual physical timer register
- Hypervisor timer register

You can configure the generic timers as count-up or count-down timers and they can operate in real-time and during virtual memory operation. You can also program a starting value for each generic timer.

Each of these timers has a 64-bit comparator that generates a private interrupt when the counter reaches the specified value. These interrupts are sent as a private peripheral interrupt with separate PPI ID.

Table 36. Private Peripheral Interrupt (PPI) ID Assignments

Timer	PPI ID
EL1 non-secure physical timer	30
EL1 secure physical timer	29
EL2 virtual physical timer	27
Hypervisor timer	26

For more information about the generic timers, please refer to the *Arm Cortex-A53 MPCore Processor Technical Reference Manual*, and the *Arm Architecture Reference Manual ARMv8, for ARMv8-A Architecture*.

Related Information

- Arm Cortex-A53 MPCore Technical Reference Manual, Revision: r0p4

- Arm Architecture Reference Manual ARMv8, for ARMv8-A Architecture Profile

3.5.14.1. System Counter

The CoreSight SoC-400 Timestamp Generator module provides a system counter to the Cortex-A53 MPCore processor generic timers.

This system counter measures the passing of time in real-time. A 64-bit bus interface carries the system timer value to each CPU and this value is used as a basis for the four generic timers. The system timer operates in the `mpu_periph_clk` domain, which is `mpu_clk/4`.

Related Information

Arm CoreSight SoC-400 Technical Reference Manual, Revision: r3p2

3.5.15. Debug Modules

The Cortex-A53 MPCore provides the following assistance for debug:

- Support for JTAG interface
- Embedded trace interface, which includes program and event trace
- Cross-trigger interface (CTI) that communicates between processors and other HPS debug modules

3.5.15.1. ARMv8 Debug

Core debug support allows you to debug your hardware and OS as well as debug of application development.

Each of the four CPUs in the Arm Cortex-A53 MPCore supports self-hosted debug and external debug. When you use self-hosted debug, you can use debug instructions to move the CPU into a debug state. When you use external debug, you can configure debug events to trigger the CPU to enter a debug state that is controlled by an external debugger.

3.5.15.2. Interactive Debugging Features

Each Cortex-A53 MPCore CPU 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.

3.5.15.3. Performance Monitor Unit

Each Armv8-A CPU has a Performance Monitoring Unit (PMU) that enables events such as cache misses and executed instructions to be counted over a period of time. The PMU supports 58 events to gather statistics on the operation of the processor and memory system. You can use up to six counters in the PMU to count and record events in real time. The PMU counters are 32-bit and are enabled based on events.

You can access each CPU's PMU counters through the system interface or from an external debugger. The events are also supplied to the Embedded Trace Macrocell (ETM) and can be used for trigger or trace.

Related Information

Arm Cortex-A53 MPCore Technical Reference Manual, Revision: r0p4

3.5.15.4. Embedded Trace Macrocell

You can perform real-time instruction flow tracing with Arm's Embedded Trace Macrocell (ETM) component. Filtering logic within the ETM can be configured to customize the amount of trace data to analyze. The ETM has a FIFO buffer to hold trace data which can be read by the external debugger.

- Support for:
  - 8-byte instruction size
  - 1-byte virtual machine ID size
  - 4-byte context ID size
- Cycle counting in the instruction trace
- Branch broadcast tracing
- Three exception levels in secure state
- Three exception levels in non-secure state
- Four events in trace
- Return stack support
- Tracing OS Error exception support
- 7-bit trace ID
- 64-bit global timestamp size
- ATB trigger support
- Low-power behavior override
- Stall control support

3.5.15.4.1. Embedded Trace Macrocell Reset

The ETM is reset on a cold reset. If you reset the ETM, tracing stops until the ETM is reconfigured. When a warm reset occurs, the ETM is not reset so that the debugger can continue to trace. If you perform a warm reset on the processor, the last few instructions executed by the processor before the reset may not be traced.

3.5.15.5. 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 37 on page 57.

Table 37. 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 485

3.5.15.6. 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.

3.5.15.7. Cross Trigger Interface

The Cortex-A53 processor has a cross trigger interface (CTI) that communicates with ARM's CoreSight module. The CTI enables debug logic, ETM and PMU to interact with each other and with other HPS debugging components including the FPGA fabric. The ETM can export trigger events and perform actions on trigger inputs. Also, a breakpoint in one CPU can trigger a break in the other.

Related Information

CoreSight Debug and Trace on page 485
For detailed information about cross-triggering and about debugging hardware in the MPU, refer to the *CoreSight Debug and Trace* chapter.

3.5.16. Cache Coherency Unit

The Cache Coherency Unit (CCU) resides outside of the Cortex-A53 MPCore processor and maintains data coherency within the SoC system. Masters in the system, including HPS peripheral and user logic in the FPGA, can access coherent memory through the CCU. The FPGA interfaces to the CCU through the FPGA-to-HPS bridge.

The CCU provides I/O coherency. I/O coherency, also called one-way coherency, allows a CCU master to see the coherent memory visible to the Cortex-A53 processor but does not allow the Cortex-A53 processor to see memory changes outside of its cache.

The masters that communicate with the CCU can read coherent memory from the L1 and L2 caches, but cannot write directly to the L1 cache. If a master performs a cacheable write to the CCU, the L2 cache updates. Any of the cacheable write locations that reside in the L1 data cache are invalidated because the L2 cache has the latest copy of those addresses.

The CCU communicates with the SCU within the Cortex-A53 MPCore to provide coherency with the SCU.

For more information, please refer to the *Cache Controller Unit* Chapter.

Related Information

Cache Coherency Unit on page 60

3.5.17. Clock Sources

Four clock inputs exist for the Cortex-A53 MPCore.

Table 38. Cortex-A53 Clock Inputs

System Clock Name	Use
mpu_clk	Main clock for the Arm Cortex-A53 MPCore processor. This synchronous clock drives each CPU including the L1 cache, the L2 cache contoller and the snoop control unit clock.
mpu_ccu_clk	Synchronous clock for the L2 RAM. The L2 RAM is clocked at ½ of the mpu_clk frequency. The 128-bit Cortex-A53 MPCore ACE bus and the system cache coherency unit (CCU), also operate in the mpu_ccu_clk domain.
mpu_periph_clk	Synchronous clock for the peripherals internal to the Arm Cortex-A53 MPCore MPU system complex. The peripherals include the generic interrupt controller and internal timers. They are clocked at ¼ of the mpu_clk frequency.
cs_pdbg_clk	Asynchronous clock for debug and performance monitor counters.

Related Information

Clock Manager on page 149

3.6. Cortex-A53 MPCore Programming Guide

3.6.1. Enabling Cortex-A53 MPCore Clocks

After the Cortex-A53 MPCore comes out of reset, the mpuclken bit in the mainpllgrp of the Clock Manager is set to 1 by default and the processor clock group is enabled. To disable the processor clock group at any time you can write a 1 to the mpuclken bit of the enr register in privileged mode.

When the processor comes out of reset, the secure internal oscillator is enabled. To use a different source, set the mpu bit in the bypassr register of the mainpllgrp of registers in the Clock Manager. Next, select the source and frequency by programming the mpuclk register in the mainpllgrp of register in the Clock Manager.

3.6.2. Bringing the Cortex-A53 MPCore out of Reset

When a cold or warm reset is issued to the ArmCortex-A53 MPCore Processor, CPU0 is released from reset automatically. CPU1, CPU2 and CPU3 reset signals remain asserted when a cold or warm reset is issued. After CPU0 comes out of reset, you can deassert the other CPU reset signals by clearing the CPUn bits in the MPU Module Reset (`mpumodrst`) register in the Reset Manager.

A cold reset, resets the entire ArmCortex-A53 MPCore, including any debug functionality. A warm reset, resets all of the MPCore, except for the debug logic.

Table 39. Reset Combinations

Reset Type	Description
HPS cold reset	The ArmCortex-A53 MPCore Processor is held in reset and powered down.
HPS cold reset with active debug	Each of the four cores in the ArmCortex-A53 MPCore Processor are held in reset. The L2 cache is held in reset but powered. Debug is enabled.
Individual Armv8-A core cold reset with active debug	One of the four cores is in held in reset so that it can be powered. The L2 cache and debug are released from reset. This configuration enables external debug over power down for the core that is held in reset.
Individual Armv8-A core warm reset with trace enabled and active debug	One of the four cores is held in reset and debug is active.

3.6.3. Enabling and Disabling Cache

You can enable the instruction and data caches in the `System Control Register (SCTLR)`.

If you disable the instruction cache, all instruction fetches are treated as non-cacheable. Only instruction cache maintenance operations continue to be maintained when the instruction cache is disabled.

You cannot enable and disable the L1 and L2 data caches separately because they are controlled by the same enable. If you disable the data cache, loads and stores are treated as non-cacheable. Only cache maintenance operations continue to be maintained in the data caches.

3.6.4. Entering Low Power Modes

The Cortex-A53 supports dynamic retention of each core, the L2 cache and the SIMD/floating-point modules. You can configure the amount of time before entering retention through the CPUECTLR and the L2ECTLR registers.

The processor will dynamically enter a dormant mode with optional L2 cache retention when not actively executing instructions.

3.7. Cortex-A53 MPCore Address Map

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

4. Cache Coherency Unit

The CCU comprises a coherency interconnect, cache coherency controller (CCC), and support for distributed virtual memory (DVM).

The Intel Agilex Hard Processor System (HPS) cache coherency unit (CCU) ensures consistency of shared data. Dedicated master peripherals in the HPS and those built in FPGA logic access coherent memory through the CCU. Cacheable transactions from the system interconnect route to the CCU.

The CCU provides I/O coherency with the Arm Cortex-A53 MPCore cache subsystem. I/O coherency, also called one-way coherency, allows HPS peripheral and FPGA masters (I/O masters) to see the same coherent view of system memory as the Cortex-A53 MPCore processor cores. The CCU also contains error protection logic and logic for optimal performance during coherent accesses. The Coherent Agent Interface (CAI) within the CCU forwards non-coherent accesses to the addressed slave port via non-coherent interconnect.

The following master ports interface to the CCU:

- Cortex-A53 MPCore processor
- FPGA-to-SoC bridge
- Translation Control Unit (TCU) (part of the SMMU)
- HPS peripheral I/O master ports interfacing to the system interconnect:
  - EMAC0/1/2
  - USB0/1
  - DMA
  - SD/MMC
  - NAND
  - Embedded Trace Router (ETR)

The CCU interfaces to the following HPS slave ports:

- S0C2FPGA port
- External SDRAM memory
- On-chip RAM
- Generic Interrupt Controller (GIC)
- Peripheral slaves and master CSR slave ports
- SDRAM register group

Related Information

- System Interconnect on page 82

- Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
  For details on the document revision history of this chapter

4.1. Supported Features

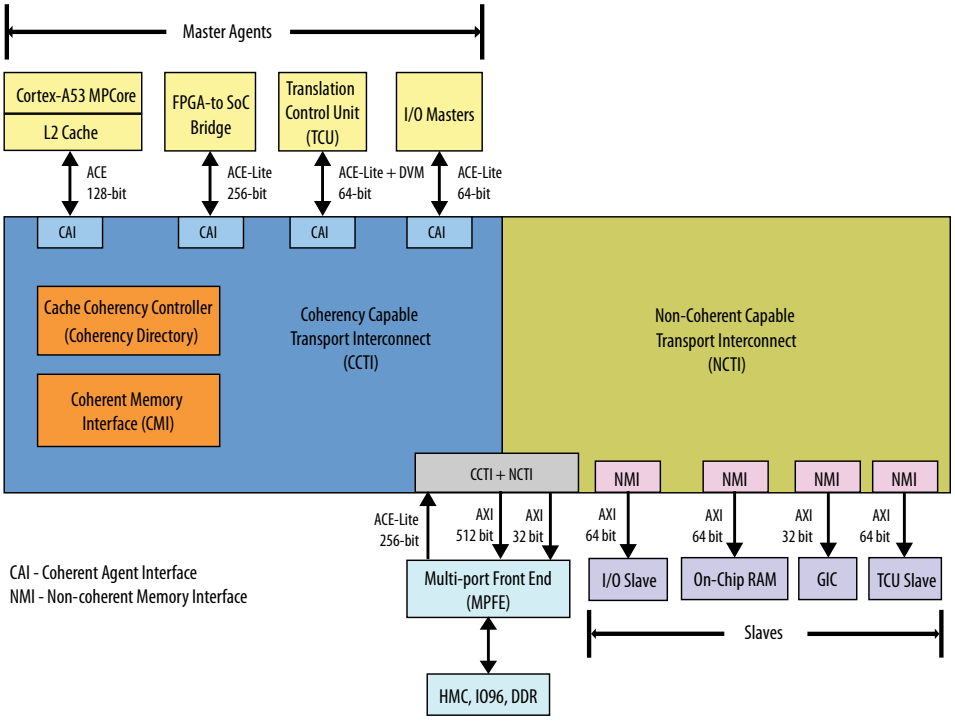
- Coherency directory to track the state of the L1 and L2 cache in the Arm Cortex-A53 MPCore
- Snoop filter support
- Single-bit error correction and double-bit error detection (SECDED) in the coherency directory
- Support for distributed virtual memory (DVM) using the Arm Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) Coherency Extensions, also known as the ACE protocol. The CCU sends distributed virtual memory broadcast messages to the Cortex-A53 MPCore and the TCU in the SMMU.
- Quality of service (QoS) support for transaction prioritization using a weight bandwidth allocation
- Interconnect debug capability through master and slave bridge status registers
- Interrupt support for CCU transaction and counter events

4.2. Functional Description

The CCU's coherency interconnect routes master agent transactions to the cache coherency components within the interconnect and ultimately, to the slave agents.

The CCU manages one-way coherency with the Cortex-A53 MPCore. The CCU allows the master agents (masters connected to the CCU) to see the coherent memory of the Cortex-A53 MPCore processor cores but does not allow the processor cores to be coherent with any caches external to the Cortex-A53 MPCore processor.

Figure 4. CCU Block Diagram



The CCU Block Diagram shows the master agents of the CCU, the CCU components and the slaves that connect to it.

Master agent ports interface to the CCU

- The Cortex-A53 MPCore port:
  - Connects the Cortex-A53 MPCore subsystem to the CCU
  - Supports memory read and write requests, as well as I/O memory-mapped read and write requests
  - Includes read and write channels and their corresponding response channels
  - Supports channels for snoop requests, snoop responses and signals used as part of the coherency protocol to indicate response arrival.
- The FPGA-to-SoC ACE-lite port connects the FPGA-to-SoC bridge to the CCU and supports I/O coherent requests to the CCU.
- The peripheral master port supports I/O coherent and non-coherent requests to the CCU from masters connected to the level 3 (L3) interconnect.
- The TCU port provides a page table walk interface to transfer requests to the CCU. This interface includes a DVM interface to send translation look-aside buffer (TLB) control information between the Cortex-A53 MPCore and the system MMU.

Slave bus ports interface to the CCU

- The MPFE port sends read and write transactions from the CCU to external memory through the MPFE interconnect.
- The MPFE register port is a dedicated interface to the MPFE interconnect, and hard memory controller registers.
- The RAM port is a dedicated interface to the on-chip RAM.
- The GIC port is a dedicated interface to the general interrupt controller (GIC).
- The peripheral slave I/O port sends memory-mapped read and write requests to slave peripherals connected to the L3 interconnect.

In the CAI, the coherency bridge accepts requests from the ACE, ACE-lite + DVM and ACE-lite buses of the master agent ports. The coherency bridge sends these requests to the cache coherency controller.

The CCU directory tracks the state of the L1 and L2 cache in the Arm Cortex-A53 MPCore.

The CAI controls address range and QoS, and tracks the transmitting logic and FIFO status. You can control and view these features through registers in the CCU.

Routers within the CCU coherency interconnect send transactions to the appropriate coherency components within the CCU or to the appropriate slave port bridge where they are de-packetized and converted to the appropriate slave agent bus protocol.

Cacheable/Non-cacheable accesses from the Cortex-A53 MPCore processor route directly to the CCU where the coherency directory is updated. The Coherent Memory Interface (CMI) with CCU forwards non-cacheable accesses to non-coherent transport interconnect (NCTI) and cacheable accesses to the directory.

Master agents with ACE-lite and ACE-lite + DVM bus interfaces send transactions to the CMI. The CMI sends coherent requests to the cache coherency controller (CCC) where a directory lookup determines if the address resides within a cache line of the MPU L2 cache.

The distributed virtual memory (DVM) controller supports the AMBA ACE DVM protocol. The DVM controller broadcasts and synchronizes control packets for TLB invalidations, cache invalidations and similar requests.

4.2.1. Connectivity

The Cortex-A53 MPCore, FPGA-to-SoC bridge, TCU and peripheral masters are connected coherently to memory and slave agents through the coherency interconnect.

The CCU supports communication between different protocols by packetizing accesses into a common protocol, routing an access to a specific port and depacketizing the transaction before it reaches the slave agent. Not all master agents have access to all five slave agents interfacing to the CCU.

Table 40. CCU Connectivity

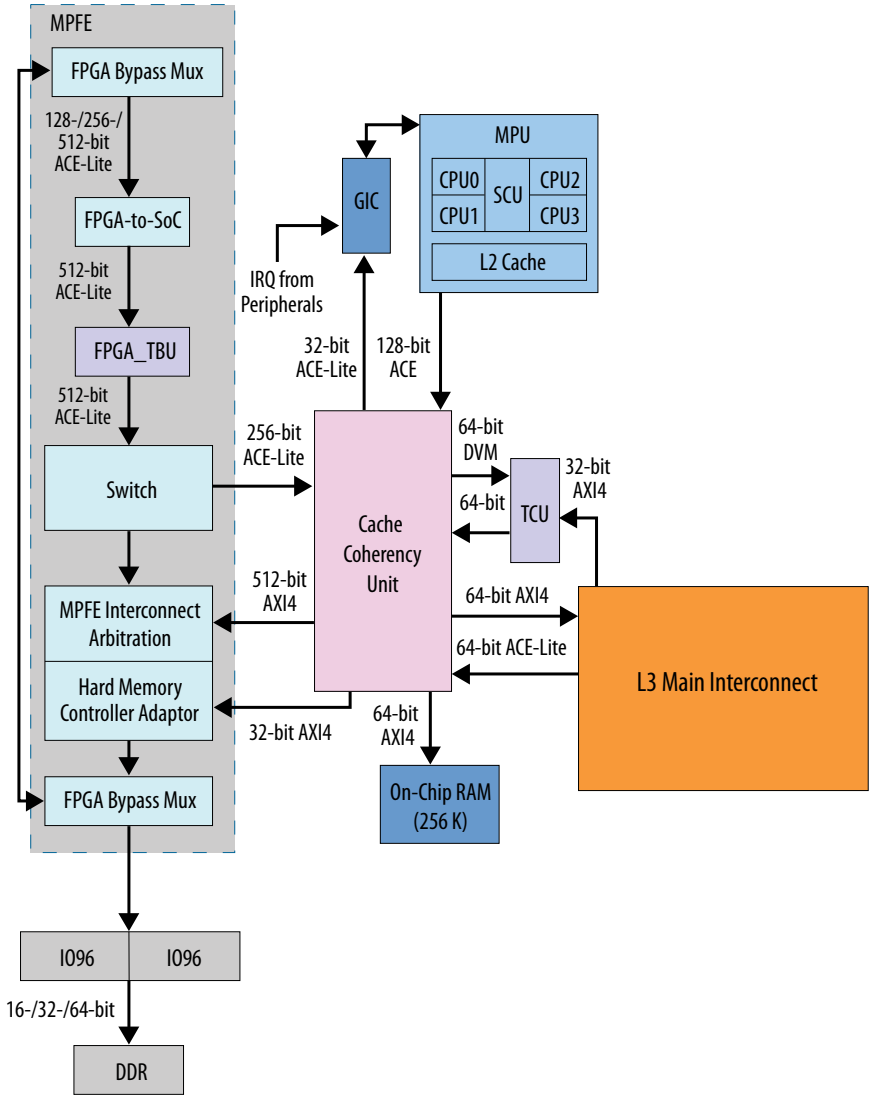
An "X" in the table indicates that the slave agent is connected to the master agent through the coherency interconnect. A blank entry indicates that there is no connection between the slave and master agent.

Slave Agents	Master Agents			
	Cortex-A53 MPCore	FPGA-to-SoC Bridge	I/O Master	Translation Control Unit (TCU)
External SDRAM memory	X	X	X	X
On-chip RAM	X	X	X	X
I/O slaves	X	X	X	
SDRAM registers	X	X	X	
Generic Interrupt Controller	X		X	

Note: The Cortex-A53 MPCore also has access to the CCU Control and Status Registers (CSRs).

4.2.2. System Integration

Figure 5. Cache Coherency Unit Integration Within System



The coherency interconnect in the CCU accepts both coherent and non-coherent transactions from masters in the system. The coherency interconnect routes non-coherent transactions to the appropriate target.

All accesses from the Cortex-A53 MPCore are routed through the CCU so the coherency directory can be updated. TCU and FPGA-to-SoC bridge accesses and peripheral master accesses coming from the L3 interconnect are routed to the CCU if they are cacheable. Non-cacheable accesses route directly to the slave.

Note: As part of the SMMU, translation buffer units (TBUs) sit between the master peripherals and the L3 interconnect. The FPGA-to-SoC bridge interface also passes through a TBU before interfacing with the CCU. The system TCU manages the TBUs and performs page table walks on translation misses. A DVM interface on the TCU allows the Cortex-A53 MPCore processor to send TLB control information to the TCU.

The CCU interfaces with the L3 interconnect and the MPFE. The MPFE provides a 32-bit register bus interface to the CCU for accessing the MPFE interconnect and hard memory controller. The CCU accesses external memory through a 512-bit interface to the MPFE.

4.2.3. Reset and Initialization

Software is responsible for transitioning each structural unit into the operational state, and then responsible for transitioning into the online state.

- Transition the directory to the online state.
- Transition the Coherent Memory Interface (CMI) to the online state.
- Transition the non-coherent bridge to the online state.
- Transition the Coherent Agent Interface (CAI) to the online state.

4.2.4. Discovery Routine

Once all structural units are in the operational state, software may perform a discovery routine to determine the number of snoop filters, the number of each type of unit, and the various unit associations. With this discovery routine, software can determine the size and organization of each snoop filter, of each proxy cache, and of each coherent memory cache; all to be able to perform maintenance operations.

4.2.5. Operational State

Once a structural unit is in the operational and online state, then it can either be in the active state or inactive state, with respect to processing transactions. Each functional unit implements a single status bit to indicate whether the unit is in the active or inactive state.

4.2.6. Maintenance Operations

Software uses maintenance operations to initialize caching structures in the coherent subsystem and to manage those caching structures relative to non-coherent agents.

- Coherent Memory Cache Maintenance
- Snoop Filter Maintenance
- Proxy Cache Maintenance

4.2.7. Error Handling

In the CCU, there are two kinds of errors reported, uncorrectable errors and correctable errors. Uncorrectable errors are errors that the hardware is unable to correct (for example: double-bit ECC). Correctable errors are errors that hardware can correct. Hardware reports the correctable error to software.

The CCU detects an error logs information about the error and signals as interrupts (`correctible_error_irq` and `uncorrectible_error_irq`). Each CCU component implements two sets of registers, `control` for enabling error reporting and `status` for logging errors. The error control registers consist of an error detection enable bit, an error interrupt enable bit, and error threshold field. Software should enable the error detection, correction, and logging for all the active components. The error threshold field in the correctable error control register determines the number of correctable errors that occur before a correctable error is logged. Once the set threshold is reached, the next detected correctable error is logged.

Software is responsible for programming the error registers and handling any errors logged by the hardware. At reset, all the error control registers are initialized to zero, disabling detection, correction and logging of error's and masking interrupts. Software must set the enable bits to one and program the threshold field to the desired value. In addition, the error valid bit and the error overflow bit are initialized to zero at reset. As errors are detected, these bits are set, causing error interrupt signals to be asserted, software must write a **1** to each bit to clear the error.

Note: Writing **1** to a bit that is not set is UNDEFINED and may result in a loss of errors. In addition, writing **1** to the error valid bit also clears the error count field.

In Intel Agilex HPS the correctable and uncorrectable interrupts and are combined to generate a single CCU_INTERRUPT. The error ISR should execute the following steps:

1. The ISR reads the Coherent Subsystem Correctable Error Interrupt Status Registers (CSCEISRn) and the Coherent Subsystem Uncorrectable Error Interrupt Status Registers (CSUEISRn), to determine which unit detected the error.
2. In the case that multiple errors have occurred, the ISR prioritizes the errors and chooses one to handle.
3. The ISR reads the correctable error status register or the uncorrectable error status register in the unit with the highest-priority error.
4. If the error valid bit is set in the error status register, the ISR reads the appropriate error location registers.
5. The ISR acknowledges the error by writing '1' to either or both error valid and error overflow bits.
6. Using the information from the error status register and error location registers, the ISR performs the desired action to handle the error and returns.

4.2.8. OCRAM Firewall

The CCU implements a firewall on the OCRAM slave. The firewall is implemented as four secure regions in the 256 kbytes addressable OCRAM space (`OCRAM_SPACE_0`), each region can be programmed to be as small as 64 Kbytes or as big as 256 kbytes. By default all regions are secure.

Table 41. OCRAM Address Ranges For Firewall

Memory Space Name	Address Range
OCRAM_SPACE_0	0x000_FFE0_0000 to 0x000_FFE3_FFFF

Note: The total reserved address space for OCRAM is 1MB. Only 256K of the address space is populated while the rest is disabled.

4.3. Cache Coherency Unit Transactions

The coherency interconnect in the CCU accepts both coherent and non-coherent transactions. These transactions are routed to the CMI.

The CCU handles transactions from the FPGA-to-SoC interface, TCU, and peripheral masters in the L3 interconnect as follows:

- Coherent read: The CMI sends the read to the coherency directory in the CCC to perform a lookup and issue a snoop to the Cortex-A53 MPCore processor if required.
  - If the access is a cache hit, data is routed from the cache.
  - If the access is a cache miss, data is routed from the appropriate slave agent after cache operations have completed.
- Coherent write: The CMI sends the write to the coherency directory in the CCC to perform a lookup and issue a snoop.
  - If the access is a cache hit, the cache is updated with the new data and the coherency directory continues to track the cache line.
  - If the access is a cache miss, then the new data is written to the appropriate slave agent.

Note: You must configure the FPGA and I/O master TBUs to prevent coherent master transactions from accessing the HPS-to-SDM mailbox address range. Please refer to the *System Memory Management Unit* chapter for more details.

- Non-coherent transactions are handled differently depending on the master agent issuing the transaction.
  - If the FPGA or TCU send a non-coherent access to the CCU, the CMI routes the access directly to the slave agent.
  - If an HPS peripheral master issues a non-cacheable memory access to on-chip RAM or SDRAM, then the L3 interconnect routes the access to the CMI of the CCU. In turn, the CCU routes the access directly to the corresponding memory.
  - If an HPS peripheral master issues a non-cacheable memory access to a peripheral slave agent, then the L3 interconnect routes the access directly to the slave, bypassing the CCU.

Some key points to remember about CCU transactions:

- You can program address ranges to be disabled, read-only, or write-only. During address decode, the CCU compares the transaction ARPROT or AWPROT with the access privilege programmed for an address range. A failed access check results in a decode error response for the transaction.
- Each address range can also be associated with hash functions that are used in the route lookup process.
- Master agents have no pre-defined priority. A master's L3 interconnect QoS level determines the associated coherency interconnect QoS priority for the L3 masters and slaves, as well as the SDRAM memory interface. The Cortex-A53 MPCore and FPGA-to-SoC interface priorities are configured in the System Manager and FPGA, respectively. You can configure the coherency interconnect QoS weights through the `QoS Profile Data Register (p_0)` registers.

- Fixed transactions are split into multiple single beat increments (INCRs).
- The CCU only accepts 16-, 32- or 64-byte WRAP transactions. All other cache line sizes generate a fatal error interrupt.
- Master and slave ports queue outstanding requests. The table below shows the maximum number of outstanding requests each agent supports.

Table 42. Maximum Outstanding Request Support

Agent	Outstanding Reads	Outstanding Writes
Cortex-A53 MPCore processor	33	21
FPGA-to-SoC Interface	8	8
TCU	16	1
Peripheral masters	32	32
External SDRAM Memory	8	8
On-chip RAM	16	1
GIC	1	1
Peripheral slaves	33	21
SDRAM register group	32	32

Certain errors or stalls can occur when unsupported accesses occur:

- An unknown address or access privilege violation on the AR or AW channels causes a decode error. This error stalls the command channels until the decode error (DECERR) response can be issued on the R or B channel, respectively.

Related Information

System Memory Management Unit on page 71

4.3.1. Command Mapping

The CCU sends transactions to different locations depending on the ACE or ACE-lite command variant.

Table 43. Command Mapping

X values denote a don't care.

ARSNOOP[3:0]	ARDOMAIN[1:0]	ARBAR[1:0]	ACE/ ACE-Lite Transaction	Transaction Type
4'b0000	2'b00, 2'b11	2'bX0	ReadNoSnoop	Non-snooping
4'b0000	2'b01, 2'b10	2'bX0	ReadOnce	Coherent
4'b1000	2'b00, 2'b01, 2'b10	2'bX0	CleanShared	Cache maintenance
4'b1001	2'b00, 2'b01, 2'b10	2'bX0	CleanInvalid	Cache maintenance
4'b1101	2'b00, 2'b01, 2'b10	2'bX0	MakeInvalid	Cache maintenance

continued...

ARSNOOP[3:0]	ARDOMAIN[1:0]	ARBAR[1:0]	ACE/ ACE-Lite Transaction	Transaction Type
3'b000	2'b00, 2'b11	2'bX0	WriteNoSnoop	Non-snooping
3'b000	2'b01, 2'b10	2'bX0	WriteUnique	Coherent
3'b001	2'b01, 2'b10	2'bX0	WriteLineUnique(1)	Coherent

4.4. Cache Coherency Unit Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

(1) WriteLine Unique is only supported on the FPGA2S0C interface.

5. System Memory Management Unit

The system memory management unit (SMMU) provides memory management services to system bus masters. The SMMU translates input addresses to output addresses based on address mapping and memory attribute information in the SMMU registers and translation tables. The SMMU also provides caching attributes for physical pages. A single translation control unit (TCU) manages distributed translation buffer units (TBUs) and performs page table walks (PTWs) on translation misses.

The SMMU conforms to the Arm SMMU v2 Specification.

Table 44. SMMU IP Description

Description	Revision Number
Arm CoreLink MMU-500	r2p4

Related Information

- Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
  For details on the document revision history of this chapter
- Arm CoreLink MMU-500 System Memory Management Unit Technical Reference Manual
  For details on the functionality of the Arm CoreLink MMU-500
- Arm Cortex-A Series: Programmer's Guide for the Armv8-A
  For programming reference information

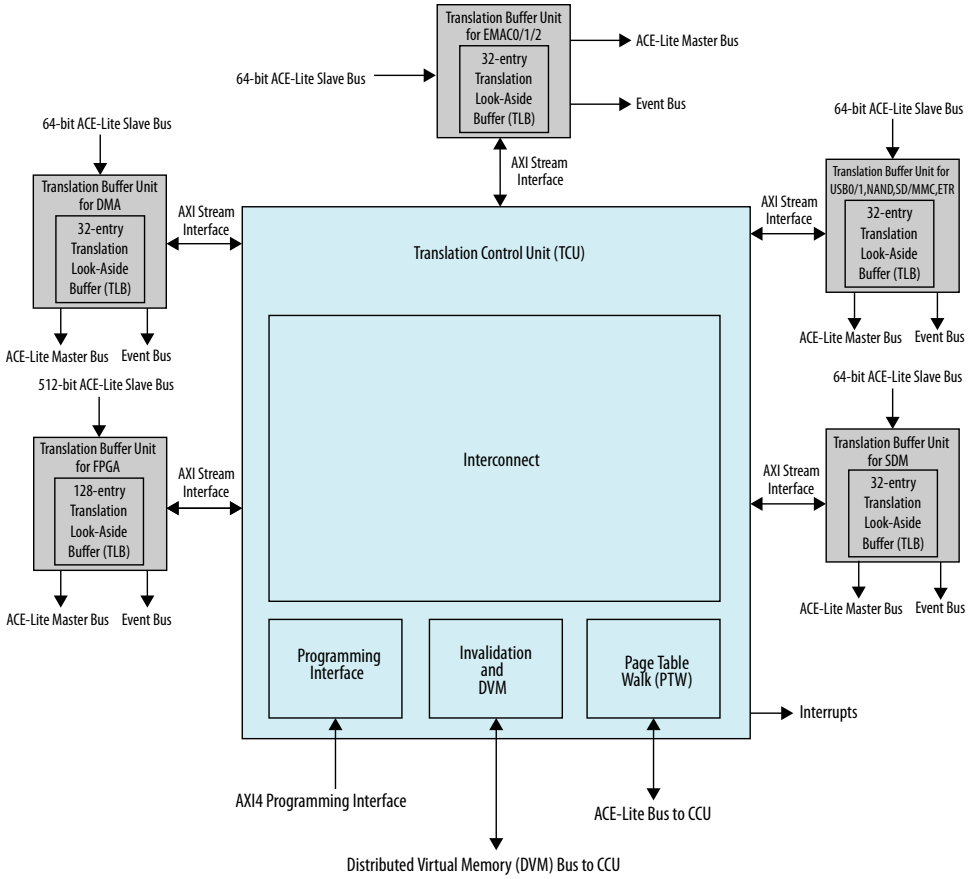
5.1. System Memory Management Unit Features

- Central TCU that supports five distributed TBUs for the following masters:
  - FPGA
  - DMA
  - EMAC0-2, collectively
  - USB0-1, NAND, SD/MMC, ETR, collectively
  - Secure Device Manager (SDM)
- Integrates caches for storing page table entries and intermediate table walk data
  - 512-entry macro TLB page table entry cache in the TCU
  - 128-entry micro TLB for table walk data in the FPGA TBU and 32-entry micro TLB for all other distributed TBUs
  - Single-bit error detection and invalidation on error detection for caches
- Communicates with the MMU of ArmCortex-A53 MPCore
- System-wide address translation

- Address virtualization
- Support for 32 contexts
- Allows two stages of translation or combined (stage 1 and stage 2) translation
  - Secure or non-secure translation capability in stage 1
  - Support for modifying attributes from stage 1 to stage 2 translation
  - Capable of multiple levels of address lookup
  - Allows bypassing or disabling stages
- Supports up to 49-bit virtual addresses and up to 48-bit physical and intermediate physical addresses
- Provides programmable Quality of Service (QoS) to support page table walk arbitration
- Provides fault handling, logging and interrupts for translation errors
- Supports debug

5.2. System MMU Block Diagram

Figure 6. System MMU Block Diagram



At the memory system level, the system MMU controls the following functions when performing an address translation:

- TLB operation
- Security state determination
- Context determination
- Memory access permissions and determination of memory attributes
- Memory attribute checks

5.2.1. System Memory Management Unit Interfaces

The TCU contains the following interfaces:

- AXI Programming Interface: The Cortex-A53 MPCore configures the SMMU through this interface.
- ACE-Lite Interface: The TCU uses this interface for page table walk memory requests to the system interconnect.
- DVM Interface: The Cortex-A53 MPCore uses this interface to send TLB control information to the SMMU TLBs.
- Interrupt Interface: The TCU sends context and system monitor interrupts to the generic interrupt controller (GIC) through this interface.

Each TBU contains the following interfaces:

- ACE-Lite Slave Interface: Creates a connection between the I/O device and the SMMU
- ACE-Lite Master Interface: Creates a connection between the SMMU and the system interconnect
- Event interface: Generates performance event signals

5.3. System Integration

The SMMU comprises the translation control unit (TCU) that interfaces to five distributed translation buffer units (TBUs).

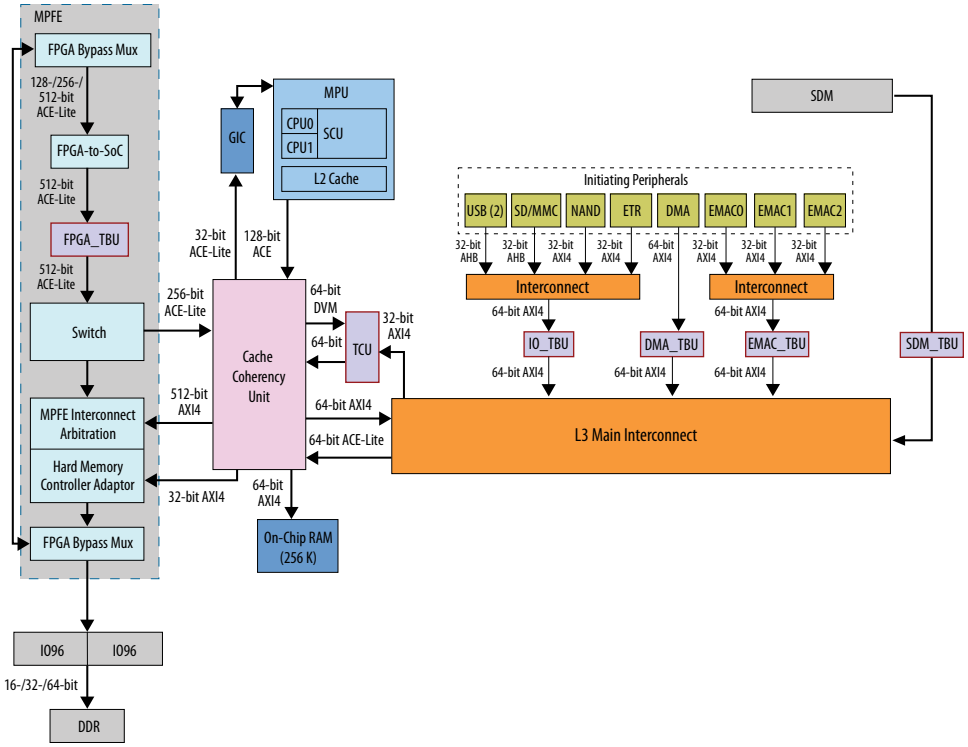
The TBUs interface to the following masters:

- FPGA
- DMA
- EMAC0-2, collectively
- USB0-1, NAND controller, SD/MMC controller, Arm Embedded Trace Router (ETR), collectively
- Secure Device Manager (SDM)

Each of the TLBs within the TBUs cache frequently used address ranges. By having multiple TBUs, the frequently cached addresses in the TLBs are localized to the masters connected to them. The TCU performs the page table walks on address misses.

The Cortex-A53 MPCore has its own main and micro translation lookaside buffers (TLBs) for address translation but communicates with the SMMU so that its translation tables remain coherent. For more information about the Cortex-A53 MPCore MMU, refer to the *Cortex-A53 MPCore* chapter.

Figure 7. System Integration



Related Information

Cortex-A53 MPCore Processor on page 32

5.4. System Memory Management Unit Functional Description

When a master issues a read or write transaction, the SMMU performs the following steps:

1. Observes the security state of the transaction that originates the request.
2. Maps the incoming transaction to one of the 32 contexts using the incoming stream ID.
3. Caches frequently used address ranges using the TLB in that master's TBU.
4. Performs a memory page table walk automatically on a TLB address lookup miss.
5. Applies memory attributes and translates the incoming address. This step is explained in the following *Translation Stages* section.
6. Applies required fault handling for every transaction.

5.4.1. Translation Stages

The SMMU supports two stages of address translation. This design allows multiple guest operating systems to run on a processor while a hypervisor manages translation tables that can translate addresses for a specific guest operating system to physical addresses.

- In stage 1 translations, the virtual address (VA) input is translated to a physical address (PA) or intermediate physical address (IPA) output. Both secure and non-secure translation contexts use stage 1 translations. Typically, an OS defines translations tables in memory for the stage 1 translations of a given security state. The OS also configures the SMMU for the stage 1 translations before enabling the SMMU to accept transactions.

  An example of a stage 1 translation could be a guest OS that translates addresses on a system that supports multiple OSs. In this case, the translation from virtual address to physical address is actually a translation from virtual address to intermediate physical address that is managed along with other OS IPAs by a virtual machine manager.
- In stage 2 translations, an IPA input is translated to a PA output. Only non-secure translation contexts can use stage 2 translations. An example of stage 2 translation could be a hypervisor translating a particular guest OS IPA to a PA.
- Stage 1 and stage 2 translations may be combined so that a VA input is translated to an IPA output and then an IPA input is translated to a PA output. The translation control unit (TCU) of the SMMU performs translation table walks for each stage of translation. An example of a combined translation could be:
  - A non-secure operating system defines the stage 1 translations for application level and operating system level operation. It does this assignment assuming it is mapping from the VAs used by the processors to the PAs in the physical memory system. However, it actually maps from VAs to IPAs.
  - The hypervisor defines the stage 2 address translations that map the IPAs to PAs. It does this as part of its virtualization of one or more non-secure guest operating systems.

Each stage of translation can require multiple translation table lookups or levels of address lookups. The SMMU can also modify memory attributes from stage 1 to stage 2 translation. You can also program the SMMU to disable or bypass a stage translation and modify the memory attributes of that disabled or bypassed stage.

5.4.2. Exception Levels

The Cortex-A53 MPCore CPUs support 4 exception levels:

- EL0 has the lowest software execution privilege, and execution in EL0 is called unprivileged execution. This execution level may be used for application software.
- EL1 provides support for operating systems.
- EL2 provides support for processor virtualization or hypervisor mode.
- EL3 provides support for the secure monitor.

As the exception level increases from 1 to 3, the software execution privilege increases.

5.4.2.1. Security State

The Arm Cortex-A53 CPUs provide the following security states, each with an associated memory address space:

- Secure state:
  - The processor can access both the secure memory address space and the non-secure memory address space.
  - When executing at EL3, the processor can access all the system control resources.
- Non-secure state:
  - The processor can access only the non-secure memory address space.
  - The processor cannot access the secure system control resources.

Depending on the security state, only certain exception levels are allowed.

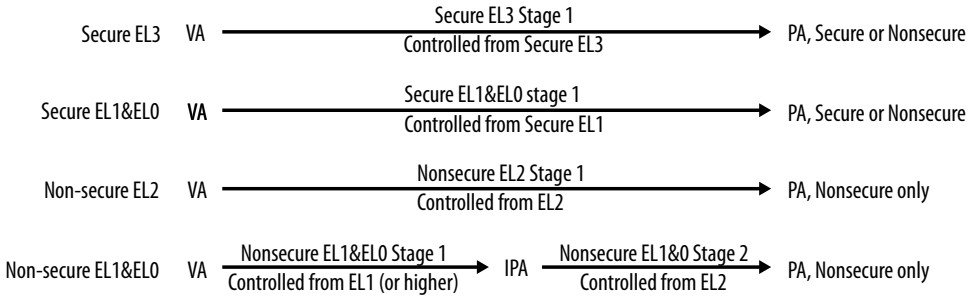
Table 45. Exception Level Implementation by Security State

Exception Level	Non-secure State	Secure State
EL0	Yes	Yes
EL1	Yes	Yes
EL2	Yes	No
EL3	No	Yes

5.4.3. Translation Regimes

The figure below shows the supported translation regimes when EL3 is using AArch64. The non-secure EL1and EL0 translation regime comprises two stages of translation. All other translation regimes shown below comprise only a single stage of translation.

Figure 8. Translation Regimes



5.4.4. Translation Buffer Unit

The FPGA TBU caches page table walk results for FPGA-issued accesses to the FPGA-to-SoC bridge. Details of the FPGA TBU configuration are shown in the table below.

The remaining TBUs have the configuration shown in the "Peripheral Master TBU" column of the *Translation Buffer Unit Configurations* table below. The DMA and SDM each have their own dedicated peripheral master TBU. EMAC0 through EMAC2 share a peripheral master TBU. The USBs, NAND, SD/MMC, and Embedded Trace Router (ETR) share a peripheral master TBU.

Table 46. Translation Buffer Unit Configurations

Parameter	FPGA TBU	Peripheral Master TBUs
AXI data bus width	512 bits	64 bits
Write buffer depth	16 entries	8 entries
TLB depth	128 entries	32 entries
TBU queue depth	8 entries	8 entries

The Cortex-A53 MPCore has its own TBU configuration. Details on this TBU can be found in the *Cortex-A53 MPCore* chapter.

5.4.4.1. Micro Translation Lookaside Buffer

The micro TLB in the TBU caches the page table walk (PTW) results returned by the TCU. The TBU compares the PTW results of incoming transactions with the entries in the micro TLB before performing a TCU PTW.

5.4.5. Translation Control Unit

The TCU cache consists of macro TLB, prefetch buffers, IPA to PA support and PTW caches.

The prefetch buffer fetches pages up to 16 KB in size. The prefetch buffer is a single four-way associative cache that you can enable or disable depending on the context.

5.4.5.1. Macro Translation Lookaside Buffer

The macro TLB caches PTW results in the TCU. The macro TLB is 64 bits wide by 512 entries deep.

5.4.6. Security State Determination

There are two concepts of security in the SMMU:

- A transaction is either secure or non-secure depending on the value of the APROT[1] signal.
- The stream has an assigned security state determination (SSD) that determines whether secure or non-secure software controls the stream.

Each transaction is classified through a security state determination (SSD) as either SSD secure or SSD non-secure. The current bus transaction provides an SSD_index that points to a bit in the `smmu_ssd_reg_*` registers. For a given transaction, the device is either SSD secure or SSD non-secure. This bit determines the SSD security state.

For an SSD secure transaction, the APROT[1] signal can indicate whether it is secure or non-secure and the information is generally passed downstream. However, an SSD non-secure transaction is forced by the SMMU to indicate non-secure transaction in

the APROT[1] signal on the downstream. For each SSD, set the SMMU_SCR0.CLIENTPD bit field if you want all transactions to bypass the translation process of the SMMU.

5.4.7. Stream ID

Each transaction is also classified by a 10-bit stream ID. The stream ID represents a set or stream of transactions from a particular master device. All transactions in a stream are subject to the same translation process. For example, the DMA controller may have multiple independent threads of execution that each form a different stream and can be subject to different translations. Alternatively, the peripheral masters on the system interconnect may share a single stream ID. Transactions from these devices can only be translated as a single entity. The TCU matches the stream ID against a set of stream match registers, SMMU_SMRx. The SSD determines the set of registers that are used. The secure software can partition the set into a non-secure set for use by SSD non-secure transactions and a secure set for use by SSD secure transactions. The stream matching process results in the following possible outcomes:

- No matches: If no matches are found, you can select whether transactions bypass the SMMU.
- Multiple match: If multiple SMMU_SMRn matches are found, the SMMU faults the transactions. The fault detection for these transactions is imprecise.
- Single match: If only a single match is found, the corresponding SMMU_S2CRn for the SMMU_SMRn that matched is used to determine the required additional processing steps.

For each SMMU_SMRn, there is a corresponding SMMU_S2CRn that is used when only a single SMMU_SMRn matches. The SMMU_S2CRn.TYPE bit field determines one of the following results:

- Fault All transactions generate a fault. A client device receives a bus abort if SMMU_sCR0.GFRE == 1, otherwise the transaction acts as read-as-zero/write-ignored (RAZ/WI).
- Bypass transactions bypass the SMMU.
- Translate transactions are mapped to a context bank for additional processing. The SMMU_S2CRn.CBNDX bit field specifies the context bank to be used by the SMMU.

The second stage boot loader configures the stream ID for the SDM-to-HPS TBU interface. The FPGA-to-SoC interface provides its stream ID. You can specify the FPGA-to-SoC stream ID value in Intel Quartus® Prime Pro Edition.

You can configure the stream ID for each HPS peripheral master through registers in the System Manager. The table below lists the peripheral masters, the corresponding System Manager register used to configure the stream ID and the specific bitfields that represent the stream ID. During a master access the stream ID source is provided as a part of the AxUSER[12:3] signals.

Table 47. HPS Master Stream ID

Master	System Manager Register	Register Bitfields Corresponding to stream ID[9:0]
EMAC0	emac0_ace	awsid[29:20]

continued...

Master	System Manager Register	Register Bitfields Corresponding to stream ID[9:0]
		arsid[17:8]
EMAC1	emac1_ace	awsid[29:20] arsid[17:8]
EMAC2	emac2_ace	awsid[29:20] arsid[17:8]
USB0	usb0_l3master	hauser22_13[25:16]
USB1	usb1_l3master	hauser22_13[25:16]
DMA	dma_l3master	aruser[25:16] awuser[9:0]
NAND	nand_axuser	aruser[25:16] awuser[9:0]
ETR	etr_l3master	aruser[25:16] awuser[9:0]

5.4.8. Quality of Service Arbitration

The TCU generates page table walks for all the TBUs. If there are multiple outstanding transactions in the TCU, the TBU with the highest quality of service (QoS) is given priority. For individual prefetch accesses, the SMMU uses the QoS value of the hit transaction. For transactions with the same QoS value, the SMMU translates the transactions in the order they occur. The QoS for each TBU is programmed in the System MMU TBU Quality of Service 0 (SMMU_TBUQOS0) register at offset 0x2100.

5.4.9. System Memory Management Unit Interrupts

Table 48. SMMU Interrupt Descriptions

Interrupt Type	GIC Interrupt Name(s)	Description
Global Fault Interrupt	gbl_flt_irpt_s gbl_flt_irpt_ns	The SMMU asserts the global fault interrupt when a fault is identified in the translation process before a context is mapped. The SMMU provides both a secure (gbl_flt_irpt_s) and non-secure (gbl_flt_irpt_ns) global fault interrupt signal to the generic interrupt controller (GIC).
Performance Monitoring Interrupt	perf_irpt_FPGA_TBU perf_irpt_DMA_TBU perf_irpt_EMAC_TBU perf_irpt_IO_TBU perf_irpt_SDM_TBU	The SMMU asserts this interrupt when a performance counter overflows.
Combined Interrupt	comb_irpt_ns comb_irpt_s	The non-secure combined interrupt is the logical OR of glb_flt_irpt_ns, perf_irpt_<tbu_name> and cxt_irpt_<number>

continued...

Interrupt Type	GIC Interrupt Name(s)	Description
		The secure combined interrupt is the logical OR of `glb_flt_irpt_s`, `perf_irpt_<tbu_name>` and `cxt_irpt_<number>`
Context Interrupt	`cxt_irpt_0` through `cxt_irpt_31`	The SMMU asserts one of these interrupts when a context fault is detected.

5.4.10. System Memory Management Unit Reset

The SMMU resets on a power-on, cold, or warm reset. On any one of these resets:

- TCU caches are cleared
- TLB entries are invalidated
- System configuration registers return to their reset state, which may by undefined

  Note: You must reconfigure the SMMU for each transaction client after reset.

5.4.11. System Memory Management Unit Clocks

- `l3_main_free_clk` is the clock source for:
  - EMAC 0/1/2 TBU
  - The TBU that services USB 0/1, NAND, SD/MMC and ETR
- `l4_main_clk` is the clock source for the DMA TBU

5.5. System Memory Management Unit Configuration

You must configure the SMMU TBUs to prevent coherent master transactions from accessing the HPS-to-SDM mailbox address range. Only the Cortex-A53 CPUs have access to the 256 byte HPS-to-SDM mailbox range.

- The 256-byte HPS-to-SDM mailbox that must be protected starts at address 0xFFA30000 and ends at 0xFFA300FF.
- Enable the SMMU to translate from virtual to physical address (stage 1 translation).
- Configure the page tables for your TBU so that it issues a context fault if a master attempts to access the HPS-to-SDM mailbox range. There are two ways you can communicate a page table context fault:
  - Use interrupts that route through the generic interrupt controller (GIC). Set the CFIE bit of the TBU's context bank system control register (SMMU_CB*_SCTLR) to enable interrupt reporting of a context fault. Program your software to sample the corresponding context interrupt, `cxt_irpt_*`. Note that the CFIE bit clears on reset. The SMMU contains 32 context banks and 32 corresponding interrupts in the GIC.
  - Generate a slave error on the AXI bus as the response sent back to the master. Set the CFRE bit of the context bank system control register (SMMU_CB*_SCTLR) to enable an abort bus error when a context fault occurs. Note that CFRE bit clears on reset.

5.6. System Memory Management Unit Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

6. 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:

- Layer 3 (L3) Interconnect

  It is a high performance tier of the interconnect that provides high-bandwidth routing between masters and slaves in the HPS.
- Layer 4 (L4) Interconnect

  It is a lower performance tier of the interconnect that handles data traffic for low- to mid-level bandwidth slave peripherals.

The system interconnect is a highly efficient packet-switched network that supports high-throughput traffic. The system interconnect is the main communication bus for the MPU and all hard IP cores in the SoC device.

The system interconnect supports the following features:

- Configurable Arm TrustZone-compliant firewall and security support.
  - For each peripheral, implements secure or non-secure access.
  - Allows configuration of individual transactions as secure or non-secure at the initiating master.
- Multi-tiered bus structure to separate high bandwidth masters from lower bandwidth peripherals and control and status ports.
- Quality of service (QoS) with three programmable levels of service on a per-master basis.
- On-chip debugging and tracing capabilities.

The system interconnect is based on the Arteris® FlexNoC™ network-on-chip (NoC) interconnect technology.

Related Information

- Arteris Website
  For information about the FlexNoC Network-on-Chip Interconnect, refer to the Arteris website.
- Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
  For details on the document revision history of this chapter

6.1. Functional Description

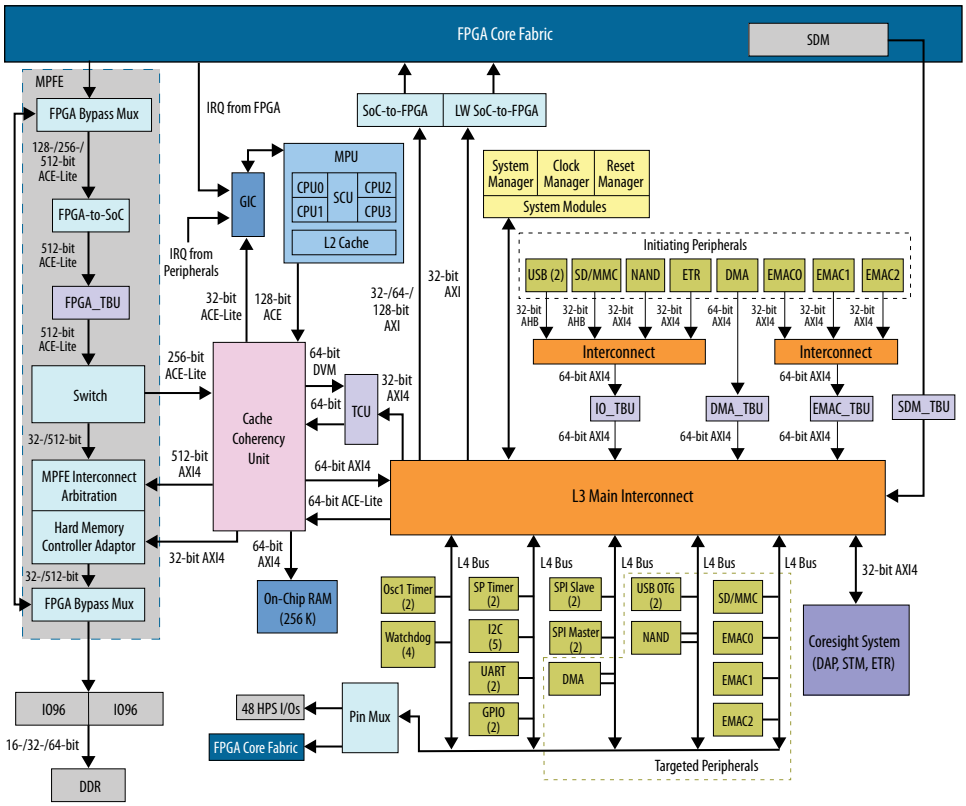
The system interconnect consists of connection points, datapaths, and the service network.

- Connection points interface the Interconnect to masters and slaves of other HPS components.
- Datapath switches transport data across the network, from initiator connection points to target connection points.
- Service network allows you to update master and slave peripheral security features and access Interconnect registers.

The system interconnect is also connected to the Cache Coherency Unit (CCU). The CCU provides additional routing between the MPU, FPGA-to-SoC bridge, MPFE, GIC, and on-chip RAM.

In addition to providing routing connectivity and arbitration between masters and slaves in the HPS, the system interconnect also features firewall security, QoS mechanisms, and observation probe points.

Figure 9. Block Diagram



The system interconnect has a transaction-based architecture that functions as a partially-connected fabric. Not all masters can access all slaves.

Each system interconnect 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 interfaces:

- SoC-to-FPGA bridge
- Lightweight SoC-to-FPGA bridge
- FPGA-to-SoC bridge

6.1.1. Masters and Slaves Connectivity Matrix

The following table shows the connectivity of all the master and slave interfaces in the system interconnect.

Table 49. Master-to-Slave Connectivity

Slaves	Masters				
	DAP	CCU Master (2)	DMAC (3)	EMAC 0/1/2	Peripheral Master (4)
CCU Slaves (5)	•		•	•	•
TCU	•	•			
L4 Main Bus Slaves	•	•	•		
L4 MP Bus Slaves	•	•			
L4 AHB Bus Slaves	•	•			
L4 SP Bus Slaves	•	•	•		
L4 SYS Bus Slaves	•	•	•		
Secure/Non-Secure Timestamp System Counters	•	•	•		
L4 ECC Bus Slaves	•	•			
L4 SHR Bus (Clock, Reset and System Manager)	•	•			
DAP	•	•			•(6)
STM		•	•		
Lightweight SoC-to-FPGA Bridge	•	•	•	•	•
SoC-to-FPGA Bridge	•	•	•	•	•
Service Network	•	•			
SoC-to-SDM – Peripheral Access (QSPI, SDMMC)	•	•	•		
SoC-to-SDM – Mailbox Access	•	•			

(2) CCU Master Agent: Cortex-A53 MPCore, FPGA-to SoC, HPS peripheral masters, TCU

(3) Direct Memory Access Controller

(4) Peripheral Master TBU, including:
- TBU for EMAC 0/1/2
- TBU for USB 0/1, NAND, SD/MMC, and ETR
- TBU for DMAC

(5) CCU Slaves: MPFE, on-chip RAM, GIC, HPS peripheral slaves

(6) ETR access only.

6.1.1.1. Connections

Figure 10. Master Connections

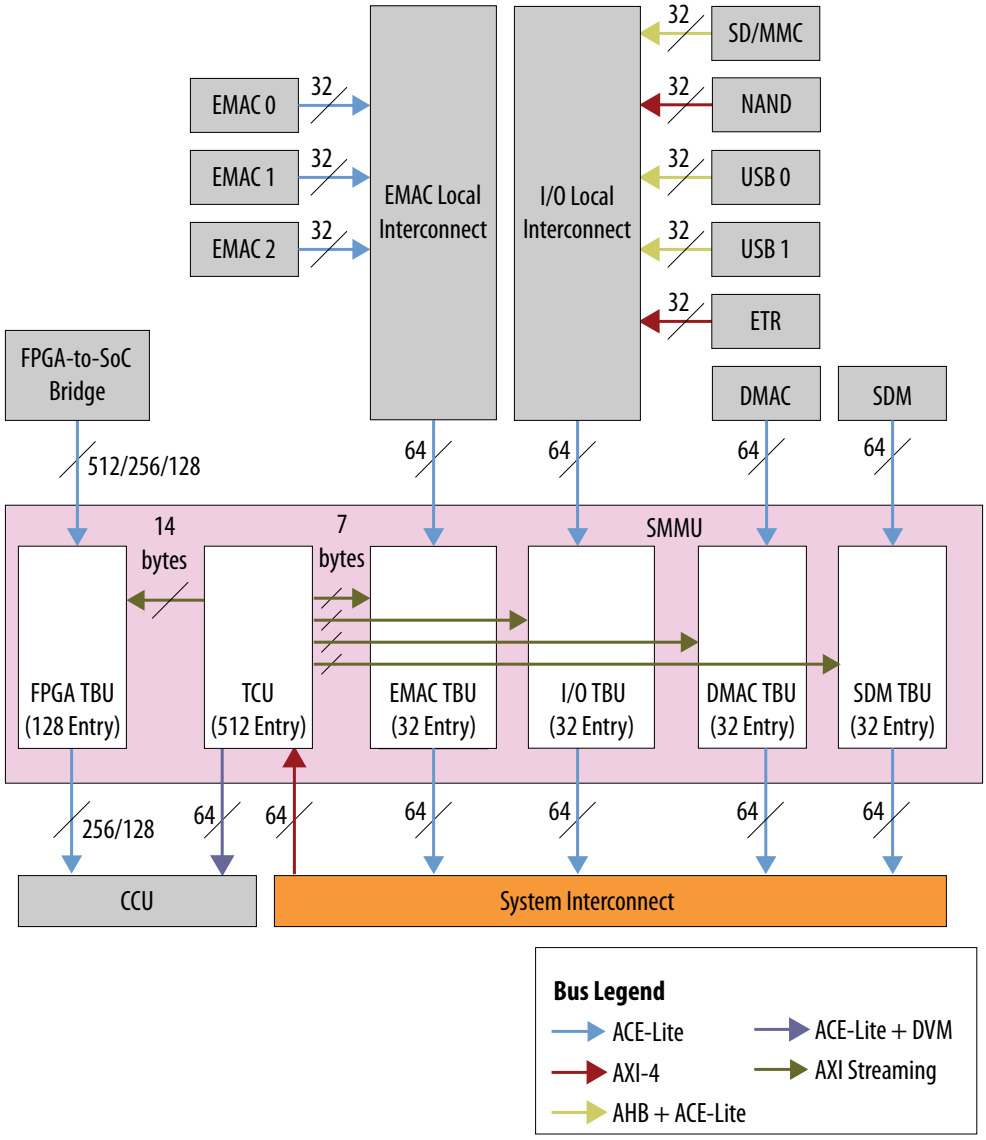


Figure 11. Peripheral Connections

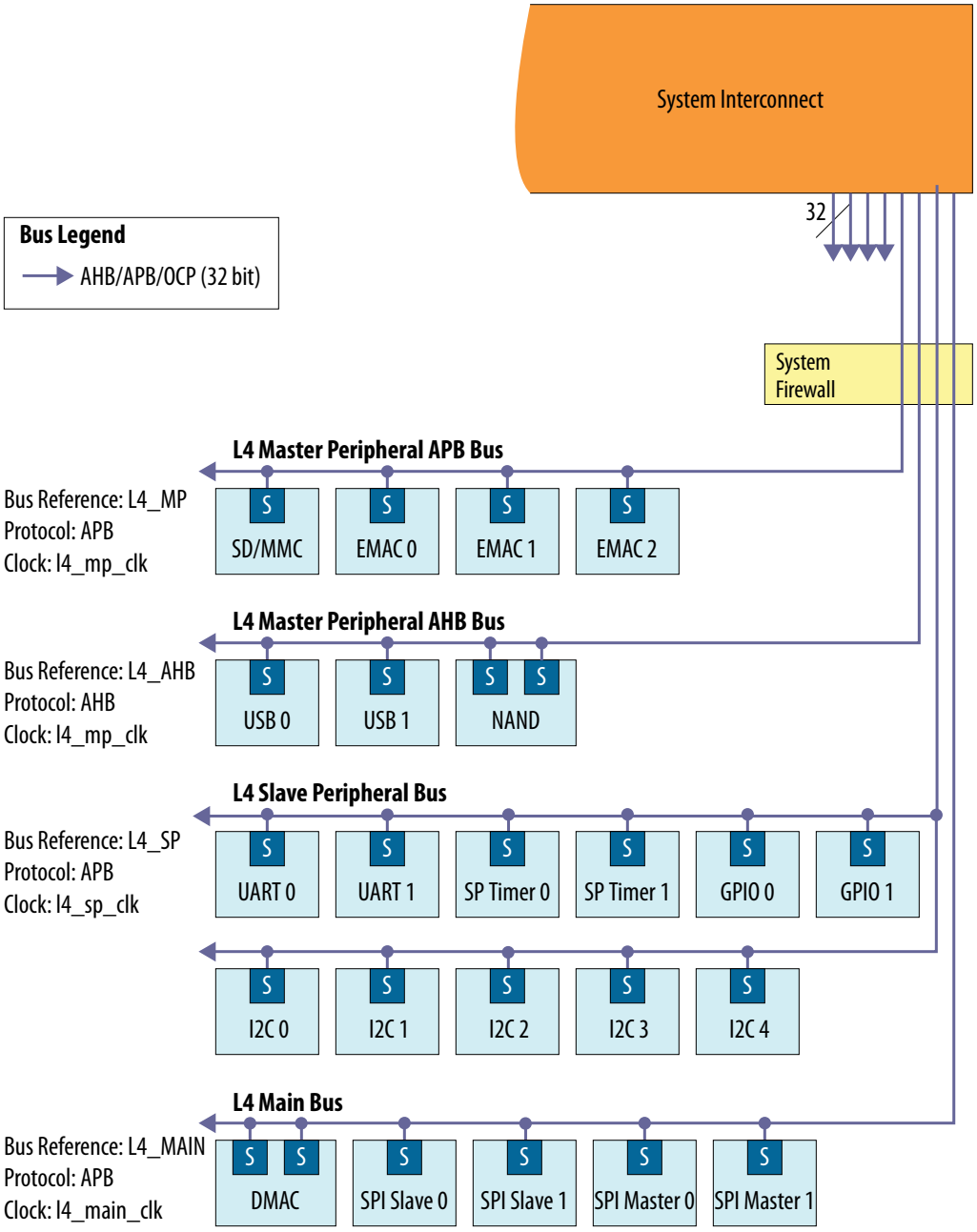


Figure 12. System Connections

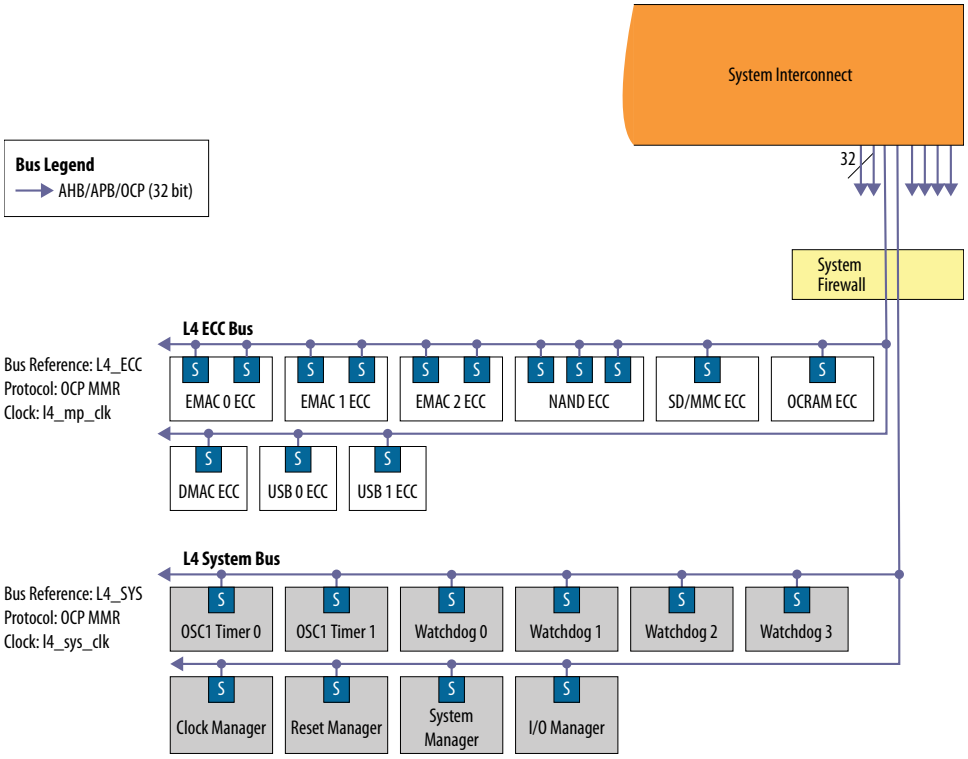


Figure 13. DAP Connection

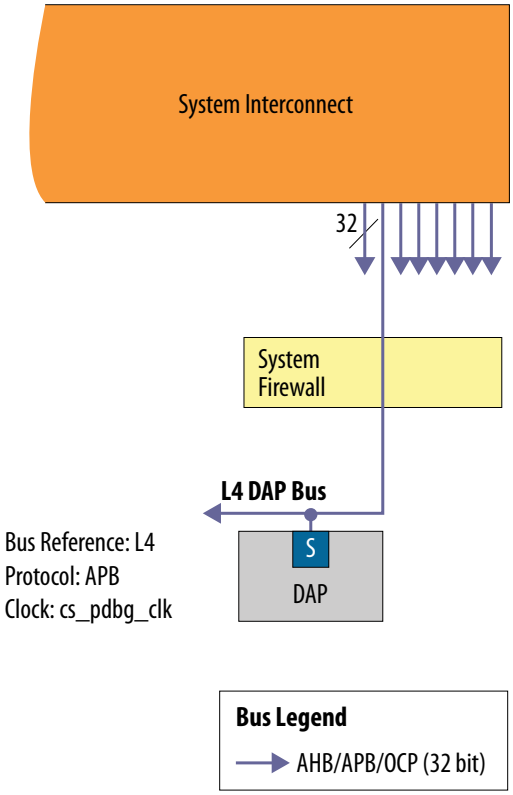
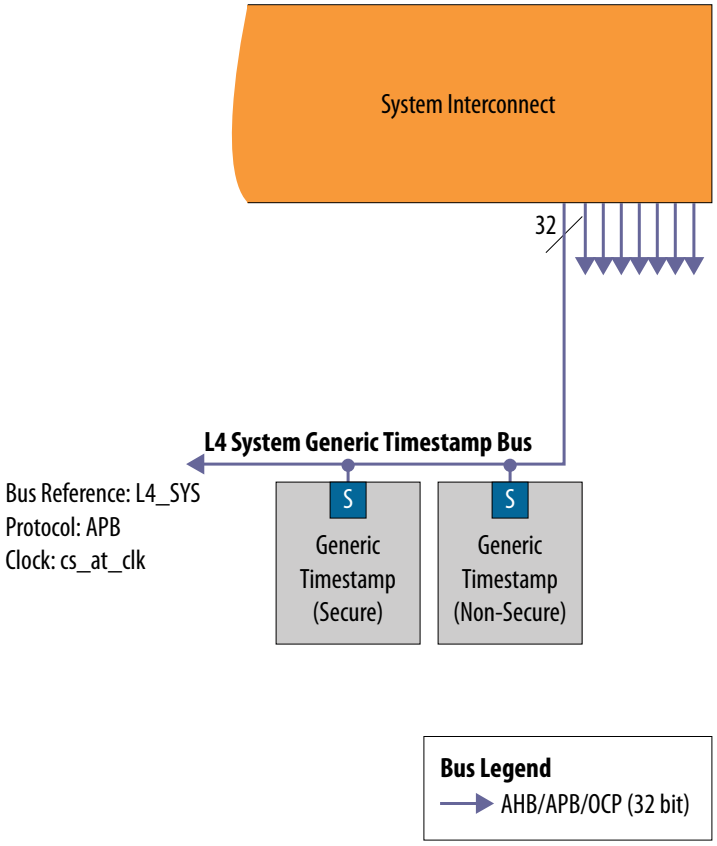


Figure 14. Generic Timestamp Connection



6.1.2. Secure Transaction Protection

The system interconnect provides two levels of secure transaction protection:

- Security firewalls (Arm TrustZone compliant)—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.

6.1.2.1. System Interconnect 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:

- For each peripheral, can implement secure or non-secure access.
- Allows configuration of individual transactions as secure or non-secure at the initiating master.
- For certain peripherals, can implement two levels of access: privileged or user.

You can program the security configuration registers (SCRs) to set security policies and define which transactions the firewall allows. Transactions that the firewall blocks result in a bus error.

The HPS has the following firewalls:

- Peripheral
- System
- SoC-to-FPGA
- Lightweight SoC-to-FPGA
- Debug access port
- TCU

6.1.2.1.1. Firewall Description

TrustZone is enforced by firewalls implemented on the slave datapath. After reset, every slave on the system interconnect is in the secure state. This feature is referred to as Boot Secure.

To change the security state of a slave requires a secure write to the appropriate SCR.

Firewalls check the secure bit of a transaction against the secure state of the slave. A transaction that passes the Firewall proceeds normally to the slave. A transaction that fails the Firewall results an error response with data set to 0. Transactions that fail the firewall are never presented to the slave interface.

The SCRs, implemented in the system interconnect, control the security state of each slave. The SCR is an internal target on the system interconnect, accessed through the service network. You can configure the slave security state on a per-master basis. This means that the SCR associated with each slave contains multiple secure state bits, one for each master allowed to access it.

Firewalls work in the following order:

1. Based on the transaction's destination slave, fetch the entire slave SCR.
2. Based on the transaction's originating master, read the master-specific secure bit in the SCR.
3. Compare the secure bit with the transaction's secure attribute to determine if the transaction should pass the firewall.

The table below shows how the secure state of a slave is used with the transaction security bit to determine if a transaction passes or fails.

Table 50. Slave Security Decision Table

Transaction Security Bit	Slave Security State (SCR)	Result
0–Non-Secure	0–Secure	Fail
1–Secure	0–Secure	Pass: transaction sent to target
0–Non-Secure	1–Non-Secure	Pass: transaction sent to target
1–Secure	1–Non-Secure	Pass: transaction sent to target

6.1.2.1.2. Master Security

All masters on the system interconnect are expected to drive the `Secure` bit attribute for every transaction. In addition to `Secure` bit, each master is assigned a Unique ID that identifies the source of a transaction. Accesses to secure slaves by non-secure masters result in a bus error.

Table 51. Master Security Bit

Master	Secure bit	Secure State	Non Secure State	Source
AXI-AP	A*PROT[1]	0	1	Driven by AXI-AP
CCU_IOS	A*PROT[1]	0	1	Driven by CCU (transported from MPU and FPGA2SOC)
DMAC	A*PROT[1]	0	1	Driven by DMAC
EMACx	A*PROT[1]	0	1	Driven by System Manager
EMAC_TBU	A*PROT[1]	0	1	Driven by TBU (transported from EMAC or page table attribute)
ETR	A*PROT[1]	0	1	Driven by ETR
ETR_TBU	A*PROT[1]	0	1	Driven by TBU (transported from ETR or page table attribute)
NAND	A*PROT[1]	0	1	Driven by System Manager
SD/MMC	HA*USER[1]	0	1	Driven by System Manager
USB	HA*USER[1]	0	1	Driven by System Manager
IO_TBU	A*PROT[1]	0	1	Driven by TBU (transported or page table attribute)
SDM_TBU	A*PROT[1]	0	1	Driven by TBU (transported from page table attribute)

6.1.2.1.3. Slave Security

The system interconnect enforces security through the slave settings. The slave settings are controlled by the Interconnect Security Control Register (SCR) in the service network.

Firewalls protect certain L3 and L4 slaves. Each of these slaves has its own security check and programmable security settings. After reset, every slave of the system interconnect is in a secure state. This feature is called boot secure. Only secure masters can access secure slaves.

The system interconnect implements seven 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 52. System Interconnect Firewalls

The main system interconnect contains firewalls configured as shown in the following table.

Name	Description
Peripherals Firewall	Filter access to slave peripherals (SPs) in the following buses: • 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 • System Trace Macrocell (STM) • L4 hard memory controller (HMC) • L4 bus registers (SCR firewall, and probes)
Lightweight SoC-to-FPGA Firewall	Controls access through the lightweight HPS-to-FPGA bridge
TCU Firewall	Controls access to the TCU. The system interconnect interfaces to the TCU through a 64-bit AXI bus.
DAP Firewall	Controls access to the CoreSight APB DAP
SoC-to-FPGA Firewall	Filter access to FPGA through the SoC-to-FPGA bridge.
DDR L3 Firewalls	Filter access to DDR and HMC Configuration Register

In addition to the firewalls listed above, the following slaves are protected by firewalls implemented outside the system interconnect:

Table 53. Firewalls Outside the System Interconnect

Slave Name	Comment
On-chip RAM Module - 256KB	Firewall in CCU

Note: At reset, the privilege filters are configured to allow certain L4 slaves to receive only secure transactions. Software must either configure bridge secure at startup, or reconfigure the privilege filters to accept non-secure transactions.

To change the security state, you must perform a secure write to the appropriate SCR register of a secure slave. A non-secure access to the SCR register of a secure slave triggers a bus error.

The following slaves are not protected by firewalls:

Table 54. Slaves Without Firewalls

Slave Name	Comment
GIC	GIC implements its own security extensions
STM	STM implements its own master security through master IDs
L4_Generic Timestamp	Fixed Secure/Non-Secure by interconnect, no configuration required.
DMA	DMA implements its own security extensions

6.1.2.2. Transaction Privilege

The system interconnect supports two levels of privilege: Privileged (or Supervisor), Non-Privileged (or User) for a transaction. Privilege is supported for all masters on the system interconnect. Privilege is enforced only on Writes. A write from a non-privileged master to a privileged slave will result in an error. Reads have no privilege requirement.

AXI supports Privilege via A*PROT[0] bit and other bus protocols may also have equivalents.

Table 55. Master Privilege

All Masters on the system interconnect is expected to drive the Privilege attribute for every transaction.

Master	Privilege bit	Privileged State	Non Privileged State	Source
AXI-AP	A*PROT[0]	1	0	Driven by AXI-AP
CCU_IOS	A*PROT[0]	1	0	Driven by CCU (transported from MPU and FPGA2SOC)
DMAC	A*PROT[0]	1	0	Driven by DMA Controller
EMACx	A*PROT[0]	1	0	Driven by System Manager
EMAC_TBU	A*PROT[0]	1	0	Driven by TBU (transported from EMAC or page table attribute)
ETR	A*PROT[0]	1	0	Driven by ETR
ETR_TBU	A*PROT[0]	1	0	Driven by TBU (transported from ETR or page table attribute)
NAND	A*PROT[0]	1	0	Driven by System Manager
SD/MMC	H*PROT[0]	1	0	Driven by System Manager
USB	H*PROT[0]	1	0	Driven by System Manager
IO_TBU	A*PROT[0]	1	0	Driven by TBU (transported or page table attribute)
SDM_TBU	A*PROT[0]	1	0	Driven by TBU (transported from page table attribute)

Table 56. Slave Privilege

Slave	Level	Programmable (Yes/No)
L4_AHB	Privileged	Y
L4_MAIN	Privileged	Y
L4_MP	Privileged	Y
L4_SP	Privileged	Y
L4_ECC	Privileged	N
L4_SEC	Privileged	N
L4_SHR	Privileged	N
L4_SYS	Privileged	N
L4_SYS_GENTS	Privileged	Y
TCU_s	Privileged	Y
CCU_IOM	Non Privileged	N
APB-DAP	Non Privileged	N
L4_NOC	Privileged	N
LWSOC2FPGA	Privileged	Y
SOC2FPGA	Privileged	Y
STM	Non Privileged	N

6.1.3. Rate Adapter

The system interconnect implements a rate adapter to buffer data packets carrying requests from L3 master peripherals to the L3 interconnect. It transfers low-bandwidth channels data to high-bandwidth channels.

The rate adapter module, `noc_mpu_m0_L4_MP_rate_ad_main_RateAdapter`, is positioned between datapaths clocked by `l3_main_free_clk` and datapaths clocked by the divided-down clocks `l4_mp_clk`, `l4_sp_clk`, and `l4_sys_clk`. At these bandwidth discontinuities, the rate adapter ensures efficient use of interconnect data pathways. You can configure the rate adapter using the `L4_MP_rate_ad_main_RateAdapter_Rate` register.

6.1.4. 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.

6.1.5. Observation Network

The observation network connects probes to the CoreSight trace funnel through the debug channel. It is physically separate from the Interconnect 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 AMBA Trace Bus (ATB) interface. Software can enable probes and retrieve probe data through the interconnect observation registers.

6.1.5.1. Interconnect Probes

The system interconnect includes the probes shown in the following table.

Table 57. System Interconnect Probe Types and Locations

Probe Name	Location (Connection Point Name)	Packet Tracing/Statistic	Transaction Profiling	Probe ID
CCU	ccu_ios_p1Resp	Yes	No	3
SOC2FPGA	soc2fgpa_probe_linkResp	Yes	No	2

continued...

Probe Name	Location (Connection Point Name)	Packet Tracing/Statistic	Transaction Profiling	Probe ID
	lwsoc2fgpa_probe_linkResp	Yes	No	
EMAC	emac_probe_linkResp	Yes	No	1
	emac_tbu_m	No	Yes	

The probe ID is available on the ATB trace bus.

6.1.5.2. Packet Tracing, Profiling, Statistics, Alarms, and Error Logs

Packet probes perform both tracing and statistic collection over packets passing through links connected to the probe.

Tracing can be fine-tuned through a Filter or a combination of Filters. Packets matching filter criteria will be sent to the observer (to be forwarded to the ATB bus). Trace alarms can also be raised when packets match filter criteria. Trace Alarms are software readable registers on the system interconnect.

Statistics collection is done by setting up counters that track the number of packets matching certain criteria going through a link. Alarms can also be set when a statistic count hit a certain level.

The following table shows how each packet probe is configured.

Table 58. Probe Configuration

Probe	nFilter	Filter On Enabled Bytes	Payload Tracing	nStatistics Counter	wStatistics Counter	Statistics Counter Alarm	Cross Trigger
CCU	2	FALSE	FALSE	4	16	TRUE	CoreSight
SOC2FPGA	2	FALSE	FALSE	4	16	TRUE	CoreSight
EMAC	2	FALSE	FALSE	4	16	TRUE	CoreSight

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

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.

EMAC Transaction Profiling

A transaction probe is available on the Ethernet MACs. You can use the transaction probe to measure either the transaction latency or the number of pending packets on the EMAC. Data are collected as a histogram

The EMAC0 transaction probe is configured as shown in the following table.

Table 59. EMAC0 Transaction Probe Configuration

Parameter	Value
Width of counters	10 bits
Available delay thresholds	64, 128, 256, 512
Available pending transaction count thresholds	2, 4, 8
Number of comparators	3

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 EMAC Transactions

In pending transaction mode, three transaction count threshold values are available for each comparator. The threshold values represent the number of requests pending on the EMACs.

Packet Alarms

You can configure the hardware to trigger a software interrupt on a packet alarms.

The following types of alarms are available:

- Trace alarms—You can examine trace alarms through the system interconnect registers.
- Statistics alarms

Error Logs

The error probe logs errors on initiators, targets, and firewalls.

6.2. System Interconnect Clocks

The clock manager drives the system interconnect clocks. The system interconnect's clocks are part of the Interconnect clock group, which is hardware-sequenced. All clocks within a domain are synchronous with each other.

The main domain is largest synchronous domain in the interconnect, containing most of the datapath. The main domain generally consists of a single free-running clock and divided clocks with enables. Resets in the main domain depend on clock groups. Each clock group in the table below uses a single reset. Paths crossing different groups also cross asynchronous reset domains.

Table 60. Clocks in the Main Clock Domain

Group	Clock	Clock Divider	Reset	Usage
main	l3_main_free_clk	-	l3_rst_n	clocks most of the interconnect datapath
	l4_main_clk	1		DMAC and SPI

continued...

Group	Clock	Clock Divider	Reset	Usage
	l4_mp_clk	2		EMAC, SDMMC, NAND, USB, ECC
	l4_sp_clk	4		L4_SP bus
	l4_sys_clk	4		L4_SYS bus
syscfg	l4_sys_clk	4	syscfg_rst_n	L4_SHR and L4_SEC buses
dbg	cs_at_clk	1	dbg_rst_n	CoreSight
	cs_pdbg_clk	2		CoreSight

The SoC-to-FPGA domain is used purely by the SoC-to-FPGA bridge. The FPGA drives the SoC-to-FPGA clock, which is asynchronous to all other clocks.

Table 61. SoC-to-FPGA Domain

Clock	Reset
soc2fpga_clk	soc2fgpa_bridge_rst_n

The Lightweight SoC-to-FPGA domain is used purely by the Lightweight SoC-to-FPGA bridge. The FPGA drives the Lightweight SoC-to-FPGA clock, which is asynchronous to all other clocks.

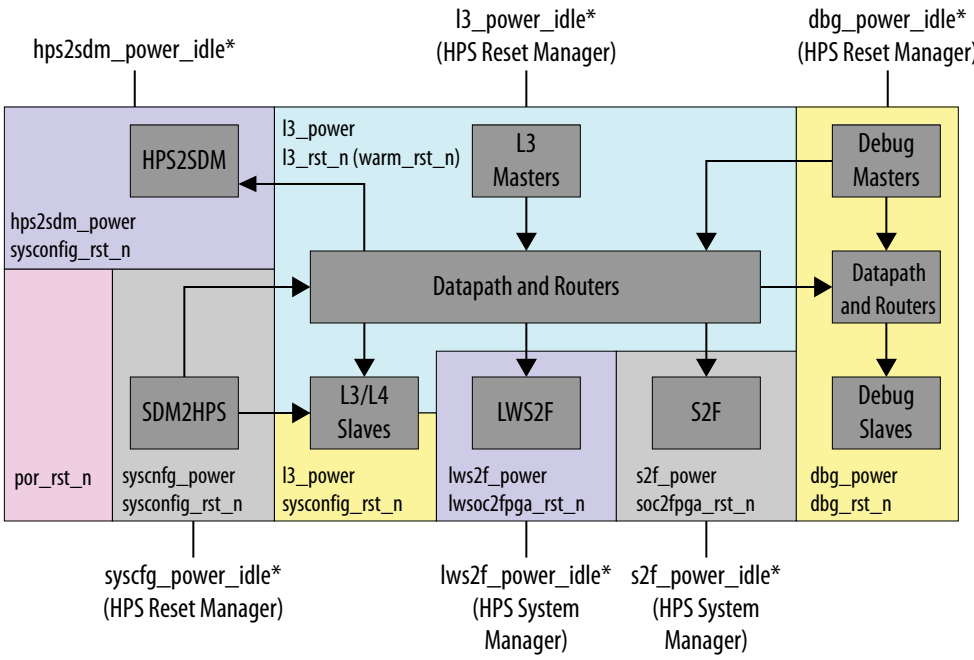
Table 62. Lightweight SoC-to-FPGA Domain

Clock	Reset
lws2f_clk	lws2f_bridge_rst_n

6.3. System Interconnect Resets

The diagram below shows the reset domains of the system interconnect along with all the idle handshake signals that control the state of each domain. The driver of each idle handshake signal is also indicated in brackets.

Figure 15. System Interconnect Reset Domains



** Handshake Signals*

The majority of the system interconnect (most masters, slaves, datapaths and routers) are reset by `l3_rst_n`. Almost all transactions in the system interconnect are routed through the `l3_rst_n` domain. For full functionality of the system interconnect, `l3_rst_n` must be out of reset.

6.4. System Interconnect Address Spaces

The system interconnect supports multiple address spaces.

Each address space uses some or all of a 1 TB address range. Depending on the configuration, different address spaces are visible in different regions for each master.

There are several address spaces that overlap each other, giving masters access to common space such as shared memory or CSRs. Within a given address map, the space is contiguous and non-overlapping. No peripheral mappings overlap makes it unnecessary to segment the space.

6.4.1. L3 Address Space

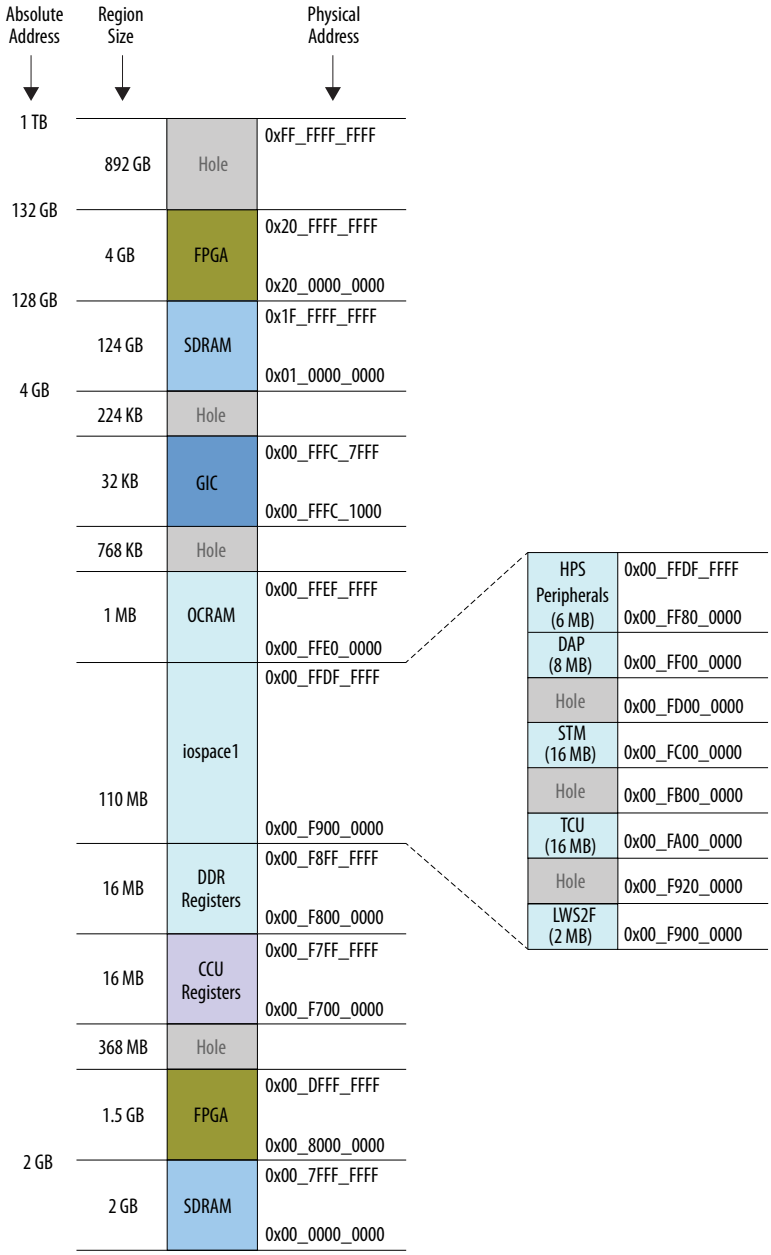
The L3 address space is 1 TB with the SMMU enabled. This address space applies to all L3 masters.

All L3 address space configurations have the following characteristics:

- The peripheral region matches the peripheral region in the MPU address space, except that MPU private registers (SCU and L2) and the GIC are inaccessible.
- The FPGA slaves region is the same as the FPGA slaves region in the MPU address space.
- The DDR Memory region is the same as the memory region in the MPU address space

The L3 address space configurations contain the regions shown in the following figure:

Figure 16. L3 Address Regions



The figure above abbreviates memory regions as follows:

- lws2f: Lightweight SoC-to-FPGA slaves region
- hps_per: Peripherals region

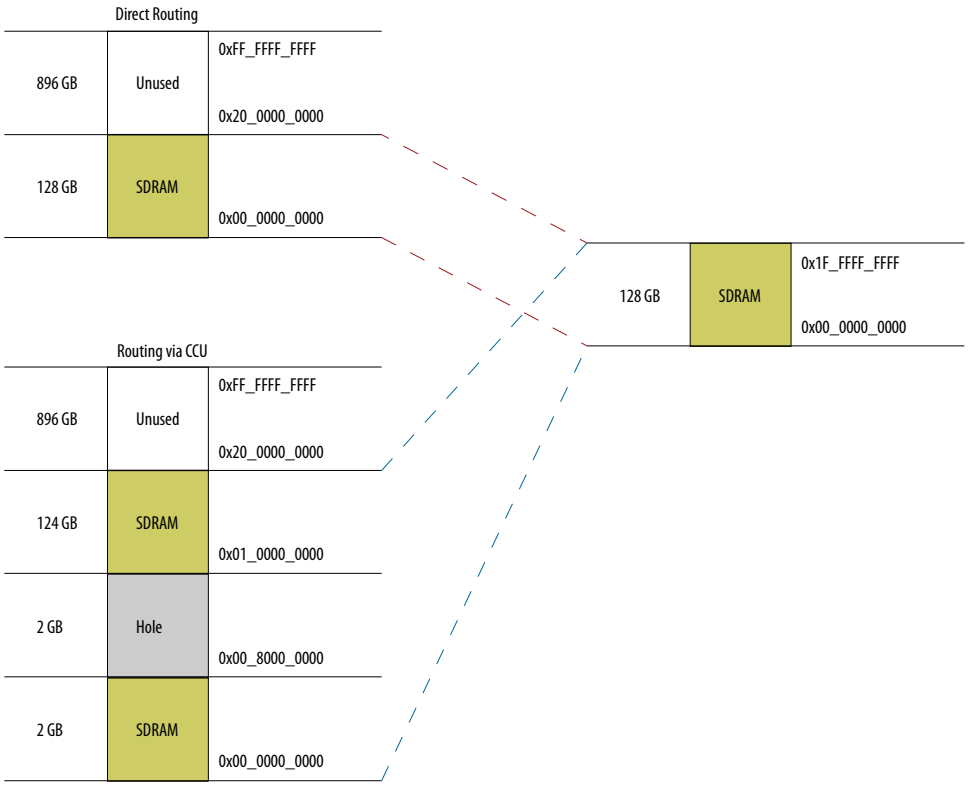
Internal MPU registers (SCU and L2) are not accessible to L3 masters.

Cache coherent memory accesses have the same view of memory as the MPU.

SDRAM Window Regions

The L3 address map includes two SDRAM window regions when the FPGA-to-SoC trafffic is routed via CCU, a 2-GB window and a 124-GB window. Otherwise, when routed directly to SDRAM, the entire 128GB address space is visible.

Figure 17. SDRAM Regions



SoC-to-FPGA Slaves Region

The SoC-to-FPGA slaves region provides access to 4 GB of slaves in the FPGA fabric through the SoC-to-FPGA bridge.

Lightweight SoC-to-FPGA Slaves Region

The lightweight SoC-to-FPGA slaves region provide access to slaves in the FPGA fabric through the lightweight SoC-to-FPGA bridge.

Peripherals Region

The peripherals region includes slaves connected to the L3 interconnect and L4 buses.

On-Chip RAM Region

The on-chip RAM region provides access to on-chip RAM. Although the on-chip RAM region is 1 MB, the physical on-chip RAM is only 256 KB.

6.4.2. MPU Address Space

The MPU address space is 1 TB, and applies to addresses generated by the MPU. MPU private registers (SCU and L2) and the GIC are visible only to the MPU. The MPU address map covers the entire HPS address map.

The MPU address space contains the following regions:

- The boot region, starting at 0x_FFE0_0000 in RAM
- The FPGA slaves window region, including the SoC-to-FPGA and lightweight SoC-to-FPGA regions
- The peripheral region

The FPGA-to-SoC bridge sees the same address space as the MPU, except for private registers (SCU and L2) and the GIC, which are visible only to the MPU.

SoC-to-FPGA Slaves Region

The SoC-to-FPGA slaves region provides access to slaves in the FPGA fabric through the SoC-to-FPGA bridge.

Lightweight SoC-to-FPGA Slaves Region

The lightweight SoC-to-FPGA slaves provide access to slaves in the FPGA fabric through the lightweight SoC-to-FPGA bridge.

Peripherals Region

The peripheral region addresses 144 MB at the top of the first 4 GB address space. The peripheral region includes all slaves connected to the L3 Interconnect, L4 buses, and MPU registers (SCU and L2). The on-chip RAM is mapped into the peripheral region.

This region provides access to internally-decoded MPU registers (SCU and L2).

Generic Interrupt Controller Region

The GIC region provides access to the GIC control and status registers.

SCU and L2 Registers Region

The SCU and L2 registers region provides access to internally-decoded MPU registers (SCU and L2).

6.4.3. SoC-to-FPGA Bridge Address Space

FPGA Slave Address Space

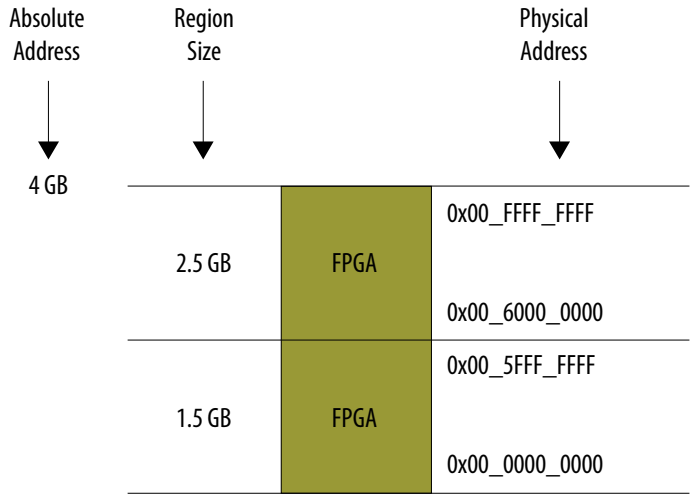
The FPGA slave address space provides access to soft components implemented in the FPGA core, through the SoC-to-FPGA bridge. The soft logic in the FPGA performs address decoding.

The L3 and MPU regions provide windows of 4 GB into the FPGA slave address space.

The lower 1.5 GB is accessible from 0x00_8000_0000 to 0x00_E000_0000 in the HPS system memory map.

The full 4 GB space is accessible starting at 0x20_0000_0000 in the HPS system memory map. Therefore, the lower 1.5 GB is mapped to two separate addresses in the HPS address space.

Figure 18. FPGA Slave Address Map



Lightweight FPGA Slave Address Map

The lightweight FPGA slave address space provides access to soft components implemented in the FPGA core through the lightweight SoC-to-FPGA bridge. The soft logic in the FPGA performs address decoding.

A portion of the peripheral region provides a window of 2 MB into the FPGA slave address space. The base address of the lightweight FPGA slaves window is mapped to address 0x0 in the FPGA slave address space.

6.4.4. Peripheral Region Address Map

Table 63. Peripheral Region Address Map

Identifier	Slave Description(s)	Base Address(es)	Size(s)	Privilege/ Security	Bus
FPGASLAVES	FPGA Slaves via HPS-to-FPGA Bridge	0x8000_0000	1.5 GB	P/S	L3
CCU	Cache Coherency Unit Register bus	0xF700_0000	16MB	-	CCU
DDRREG	DDR Scheduler and Hard Memory Controller Configuration Register	0xF800_0000	16 MB	P/S	CCU
LWFPGASLAVES	FPGA Slaves Accessed Via Lightweight HPS-to-FPGA Bridge	0xF900_0000	2 MB	P/S	L3
LWFPGASLAVES	Cache Cleaning Slaves	0xF9C0_0000	4 MB	P/S	L3
TCU	TCU Configuration	0xFA00_0000	16 MB	P/S	L3
STM	STM Module	0xFC00_0000	16 MB	-	L3
DAP	DAP Module	0xFF00_0000	8 MB	P/S	L3
EMAC0	EMAC0 Module	0xFF80_0000	8 KB	-	L4 MP

continued...

Identifier	Slave Description(s)	Base Address(es)	Size(s)	Privilege/ Security	Bus
EMAC1	EMAC1 Module	0xFF80_2000	8 KB	-	L4 MP
EMAC2	EMAC2 Module	0xFF80_4000	8 KB	-	L4 MP
SDMMC	SD/MMC Module	0xFF80_8000	4 KB	-	L4 MP
EMAC0RXECC	EMAC0 RX ECC	0xFF8C_0000	1 KB	P/S	L4 ECC
EMAC0TXECC	EMAC0 TX ECC	0xFF8C_0400	1 KB	P/S	L4 ECC
EMAC1RXECC	EMAC1 RX ECC	0xFF8C_0800	1 KB	P/S	L4 ECC
EMAC1TXECC	EMAC1 TX ECC	0xFF8C_0C00	1 KB	P/S	L4 ECC
EMAC2RXECC	EMAC2 RX ECC	0xFF8C_1000	1 KB	P/S	L4 ECC
EMAC2TXECC	EMAC2 TX ECC	0xFF8C_1400	1 KB	P/S	L4 ECC
USB0ECC	USB0 ECC	0xFF8C_4000	1 KB	P/S	L4 ECC
USB1ECC	USB1 ECC	0xFF8C_4400	1 KB	P/S	L4 ECC
NANDECC	NAND ECC	0xFF8C_8000	1 KB	P/S	L4 ECC
NANDREADECC	NAND READ ECC	0xFF8C_8400	1 KB	P/S	L4 ECC
NANDWRITEECC	NAND WRITE ECC	0xFF8C_8800	1 KB	P/S	L4 ECC
SDMMCECC	SDMMC ECC	0xFF8C_8C00	1 KB	P/S	L4 ECC
DMAECC	DMAC ECC	0xFF8C_9000	1 KB	P/S	L4 ECC
OCRAMECC	OCRAM ECC	0xFF8C_C000	1 KB	P/S	L4 ECC
USB0	USB0 OTG Controller Module Registers	0xFFB0_0000	256 KB	-	L4 AHB
USB1	USB1 OTG Controller Module Registers	0xFFB4_0000	256 KB	-	L4 AHB
NANDREGS	NAND Controller Module Registers	0xFFB8_0000	64 KB	-	L4 AHB
NANDDATA	NAND Controller Module Data	0xFFB9_0000	64 KB	-	L4 AHB
UART0	UART0 Module	0xFFC0_2000	256 B	-	L4 SP
UART1	UART1 Module	0xFFC0_2100	256 B	-	L4 SP
I2C0	I^2C0 Module	0xFFC0_2800	256 B	-	L4 SP
I2C1	I^2C1 Module	0xFFC0_2900	256 B	-	L4 SP
I2C2	I^2C2 Module	0xFFC0_2A00	256 B	-	L4 SP
I2C3	I^2C3 Module	0xFFC0_2B00	256 B	-	L4 SP
I2C4	I^2C4 Module	0xFFC0_2C00	256 B	-	L4 SP
SPTIMER0	SP Timer0 Module	0xFFC0_3000	256 B	-	L4 SP
SPTIMER1	SP Timer1 Module	0xFFC0_3100	256 B	-	L4 SP
GPIO0	GPIO0 Module	0xFFC0_3200	256 B	-	L4 SP
GPIO1	GPIO1 Module	0xFFC0_3300	256 B	-	L4 SP
OSC1TIMER0	OSC1 Timer0 Module	0xFFD0_0000	256 B	P/S	L4 sys
OSC1TIMER1	OSC1 Timer1 Module	0xFFD0_0100	256 B	P/S	L4 sys
L4WD0	Watchdog0 Module	0xFFD0_0200	256 B	P/S	L4 sys

continued...

Identifier	Slave Description(s)	Base Address(es)	Size(s)	Privilege/ Security	Bus
L4WD1	Watchdog1 Module	0xFFD0_0300	256 B	P/S	L4 sys
L4WD2	Watchdog2 Module	0xFFD0_0400	256 B	P/S	L4 sys
L4WD3	Watchdog3 Module	0xFFD0_0500	256 B	P/S	L4 sys
GENTSSEC	Generic Timestamp, Secure	0xFFD0_1000	4 KB	P/S	L4 sys
GENTSNSEC	Generic Timestamp, Non-secure	0xFFD0_2000	4 KB	P	L4 sys
CLKMGR	Clock Manager Module	0xFFD1_0000	4 KB	P/S	L4 sys
RSTMGR	Reset Manager Module	0xFFD1_1000	4 KB	P/S	L4 sys
SYSMGR	System Manager Module	0xFFD1_2000	4 KB	P/S	L4 sys
IOMGR	I/O Manager Module	0xFFD1_3000	4 KB	P/S	L4 sys
L4FRW	L4 Interconnect Firewall CSR	0xFFD2_1000	4 KB	P/S	L4 noc
L4PRB	L4 Interconnect Probes CSR	0xFFD2_2000	8 KB	P/S	L4 noc
L4QOS	L4 Interconnect QoS	0xFFD2_4000	8 KB	P/S	L4 noc
DMANONSECURE	DMAC Non-Secure Module Registers	0xFFDA_0000	4 KB	-	L4 main
DMASECURE	DMAC Secure Module Registers	0xFFDA_1000	4 KB	S	L4 main
SPI0	SPI 0 Module slave	0xFFDA_2000	4 KB	-	L4 main
SPI1	SPI 1 Module slave	0xFFDA_3000	4 KB	-	L4 main
SPI2	SPI 2 Module master	0xFFDA_4000	4 KB	-	L4 main
SPI3	SPI 3 Module master	0xFFDA_5000	4 KB	-	L4 main
OCRAM	On-chip RAM Module - 256KB	0xFFE0_0000	1 MB	-	CCU
GIC	GIC	0xFFFC_1000	32 KB	-	CCU
FPGASLAVES	FPGA Slaves via HPS-to-FPGA Bridge	0x20_0000_0000	4 GB	P/S	L3

6.5. System Interconnect Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

7. HPS Bridges

This chapter describes the bridges in the 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 bridges:

- FPGA-to-SoC Bridge
- SoC-to-FPGA Bridge
- Lightweight SoC-to-FPGA Bridge

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

7.1. Features of the HPS Bridges

The HPS bridges allow masters in the FPGA fabric to communicate with slaves in the HPS logic and vice versa. For example, you can instantiate additional memories or peripherals in the FPGA fabric, and master interfaces belonging to components in the HPS logic can access them. You can also instantiate components such as a Nios® II processor in the FPGA fabric and their master interfaces can access memories or peripherals in the HPS logic.

Table 64. HPS Bridge Features

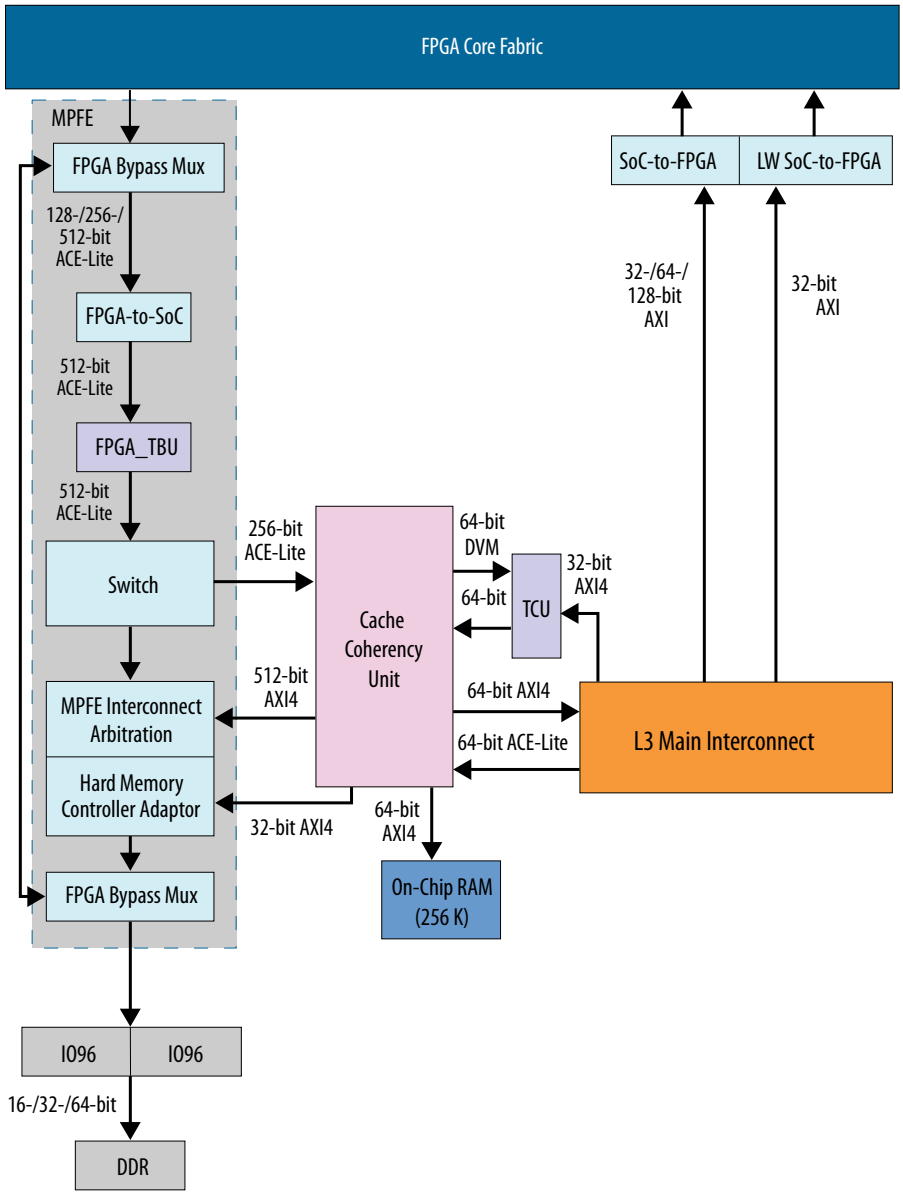
Feature	FPGA-to-SoC	SoC-to-FPGA	Lightweight SoC-to-FPGA
Interface support	ACE Lite	AMBA AXI4	AMBA AXI4
Implements clock crossing and manages the transfer of data across the clock domains in the HPS logic and the FPGA fabric	Y	Y	Y
Performs data width conversion between the HPS logic and the FPGA fabric	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.

7.2. HPS Bridges Block Diagram

Figure 19. Block Diagram

The following figure shows the HPS bridges in the context of the FPGA fabric to the HPS.



7.3. FPGA-to-SoC Bridge

The FPGA-to-SoC 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 ACE-Lite protocol, with a data width of 128/256/512 bits.

The FPGA-to-SoC bridge is configurable in the HPS component parameter editor, available in Platform Designer and the IP Catalog. The FPGA master selects either CCU or SDRAM as the target of the transaction by using a user bit on the AXI bus or selecting the interface target in Platform Designer.

Table 65. FPGA-to-SoC Bridge Properties

The following table lists the properties of the FPGA-to-SoC bridge, including the configurable slave interface exposed to the FPGA fabric.

Bridge Property	Value
Data width(7)	128, 256, or 512 bits
Clock domain	`fpga2soc_clk`
Address width	40 bits
ID width	5 bits
Read acceptance	16 transactions
Write acceptance	16 transactions
Total acceptance	16 transactions

Note: If the FPGA Fabric Bypass Mux is enabled, then

- FPGA-to-SoC bridge is not available for FPGA to CCU and FPGA to SDRAM traffic.
- MPFE remains in reset.
- SDRAM ECC is not available. But, SDRAM traffic can still be ECC protected using the soft logic
- FPGA to SDRAM access is managed in a similar manner like any other IO96 or IO96 pair from the FPGA.
- SoC to SDRAM path is routed through SoC-to-FPGA port to FPGA. It allows FPGA to control the SDRAM bandwidth allocation as well as in-line encryption for SDRAM traffic.

7.3.1. FPGA-to-SoC Bridge Signals

Table 66. FPGA-to-SoC Bridge Signals

Name	Direction	Description
`fpga2soc_clk`	Input	Clock source from FPGA.
`fpga2soc_bridge_rst_n`	Input	Module reset signal from Reset Manager.
`fpga2soc_enable`	Input	To enable the FPGA-to-SoC Bridge from MPFE Interconnect.
`fpga2soc_cmd_idle`	Output	Indicates that the AW and AR channels are idle.
`fpga2soc_force_drain`	Input	Forces B and R channels to be flushed.
`fpga2soc_resp_idle`	Output	Indicates that the B channel and R channel are idle.
`fpga2soc_port_size_config[1:0]`	Input	Port width configuration signal from FPGA:
		continued...

(7) The bridge master data width is user-configurable at the time you instantiate the HPS component in your system.

Name	Direction	Description
		• 00: 128-bit • 01: 256-bit • 10: 512-bit • 11: Reserved

7.4. SoC-to-FPGA Bridge

The SoC-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 size of the address space is 4 GB. You can configure the bridge master exposed to the FPGA fabric for 32/64/128-bit data.

The SoC-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.

Table 67. SoC-to-FPGA Bridge Properties

The following table lists the properties of the SoC-to-FPGA bridge, including the configurable master interface exposed to the FPGA fabric.

Bridge Property	Value
Data width [8]	32, 64, or 128 bits
Clock domain	`soc2fpga_clk`
Address width	32 bits
ID width	4 bits
Read acceptance	16 transactions
Write acceptance	16 transactions
Total acceptance	16 transactions

The SoC-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.

7.4.1. SoC-to-FPGA Bridge Signals

All the SoC-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 SoC-to-FPGA bridge incorporates the Arm TrustZone technology by providing the `ARPROT[1]` and `AWPROT[1]` signals, which specify whether a transaction is secure or non-secure. The firewalls use these signals to determine whether each bus access is valid.

(8) The bridge master 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 table lists signals exposed by the SoC-to-FPGA master interface to the FPGA fabric.

Table 68. SoC-to-FPGA Bridge Bridge Signals

Name	Direction	Description
soc2fpga_clk	Input	Clock source from FPGA.
soc2fpga_bridge_rst_n	Input	Module reset signal from Reset Manager.
soc2fpga_port_size_config[1:0]	Input	Port width configuration signal from FPGA: • 00: 32-bit • 01: 64-bit • 10: 128-bit • 11: Reserved

Table 69. SoC-to-FPGA Bridge Master Write Address Channel Signals

Signal	Width	Direction	Description
AWID	4 bits	Output	Write address ID
AWADDR	32 bits	Output	Write address
AWLEN	8 bits	Output	Burst length
AWSIZE	3 bits	Output	Burst size

7.5. Lightweight SoC-to-FPGA Bridge

The lightweight SoC-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 Cortex-A53 MPCore processor, direct memory access (DMA) controller, and debug access port (DAP) can use the lightweight SoC-to-FPGA bridge to access the FPGA fabric or NoC registers.

The lightweight SoC-to-FPGA bridge has a fixed data width of 32 bits.

Use the lightweight SoC-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 SoC-to-FPGA bridge, and can improve both register access latency and overall system performance.

Table 70. Lightweight SoC-to-FPGA Bridge Properties

This table lists the properties of the lightweight SoC-to-FPGA bridge, including the master interface exposed to the FPGA fabric.

Bridge Property	Value
Data width	32 bits
Clock domain	lwsoc2fpga_clk

continued...

Bridge Property	Value
Address width	32 bits
ID width	4 bits
Read acceptance	16 transactions
Write acceptance	16 transactions
Total acceptance	16 transactions

The lightweight SoC-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, to match the FPGA bitstream.

7.5.1. Lightweight SoC-to-FPGA Bridge Signals

Table 71. Lightweight SoC-to-FPGA Bridge Signals

Name	Direction	Description
`lwsoc2fpga_clk`	Input	Clock source from FPGA.
`lwsoc2fpga_bridge_rst_n`	Input	Module reset signal from Reset Manager.
`lwsoc2fpga_port_size_config[0]`	Input	Port width configuration signal from FPGA: • 0: 32-bit • 1: Reserved

7.6. Clocks and Resets

7.6.1. FPGA-to-SoC Bridge Clocks and Resets

The master interface of the bridge in the HPS logic operates in the `ccu_clk` clock domain, which is `mpu_clk` / 2. The slave interface exposed to the FPGA fabric operates in the `fpga2soc_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-SoC bridge has one reset signal, `fpga2soc_bridge_rst_n`. The reset manager asserts this signal to the FPGA-to-SoC bridge on a cold or warm reset.

Related Information

- Clock Manager on page 149
- Reset Manager on page 161

7.6.2. SoC-to-FPGA Bridge Clocks and Resets

The master interface into the FPGA fabric operates in the `soc2fpga_clk` clock domain. 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 SoC-to-FPGA bridge has one reset signal, `soc2fpga_bridge_rst_n`. The reset manager asserts this signal to the SoC-to-FPGA bridge on a cold or warm reset.

Related Information

- Clock Manager on page 149
- Reset Manager on page 161

7.6.3. Lightweight SoC-to-FPGA Bridge Clocks and Resets

The master interface into the FPGA fabric operates in the `lwsoc2fpga_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 SoC-to-FPGA bridge has one reset signal, `lwsoc2fpga_bridge_rst_n`. The reset manager asserts this signal to the lightweight SoC-to-FPGA bridge on a cold or warm reset.

Related Information

- Clock Manager on page 149
- Reset Manager on page 161

7.6.4. Taking HPS 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.

After the Cortex-A53 MPCore 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 section: *Reset Signals and Registers* in the *Reset Manager* chapter.

7.7. Data Width Sizing

The SoC-to-FPGA bridge allows 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.

7.8. HPS Bridges Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

8. 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), Revision r1p2_rel.

- 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 Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

8.1. 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 System Complex. 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
- Software programmable with dedicated register field
- Supports multiple transfer types:
  - Memory-to-memory
  - Memory-to-peripheral
  - Peripheral-to-memory
  - Scatter-gather
- Supports eight DMA channels
- Supports eight outstanding AXI read and eight outstanding AXI write transactions

- Enables software to schedule up to 16 outstanding read and 16 outstanding write instructions
- Supports nine interrupt lines into the MPU System Complex:
  - One for DMA thread abort
  - Eight for external events
- Supports up to 32 peripheral request interfaces[9]:
  - Eight for FPGA[10]:
    - FPGA_0 - FPGA_5
    - FPGA_6 is multiplexed with I^2C_EMAC2_TX
    - FPGA_7 is multiplexed with I^2C_EMAC2_RX
  - Ten for I^2C:
    - I^2C_EMAC2 TX is multiplexed with FPGA_6
    - I^2C_EMAC2 RX is multiplexed with FPGA_7
    - I^2C0 (TX and RX) - I^2C1 (TX and RX)
    - I^2C_EMAC0 (TX and RX) - I^2C_EMAC1 (TX and RX)
  - Eight for SPI
  - One for System Trace Macrocell (STM)
  - Four for UART

The following peripheral interface protocols are supported:

- Synopsys protocol, which is used by the following peripheral interfaces:
  - Serial peripheral interface (SPI)
  - Universal asynchronous receiver transmitter (UART)
  - Inter-integrated circuit (I^2C)
  - FPGA interface
- Arm protocol, which is used by the STM peripherals.
  - System trace macrocell (STM) peripherals[11]

The DMA controller provides:

- Linux drivers for DMA transfers
- An Arm Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface (AXI) master interface unit
- A multi-FIFO (MFIFO) data buffer that it uses to store data that it reads, or writes, during a DMA transfer

---

(9) Three of the interfaces are Reserved.

(10) The HPS requires a total of 33 peripheral request interfaces, while the DMAC supports a maximum of 32 interfaces; therefore, FPGA_6 and FPGA_7 are controlled by the system manager software control registers.

(11) Supports the same Arm protocol and does not need an adapter.

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 provide access to status registers and are used to directly issue and execute instructions in the DMA controller.

8.2. DMA Controller Block Diagram

The following figure shows a block diagram of the DMAC and how it integrates into the rest of the HPS system.

Figure 20. DMA Controller Block Diagram

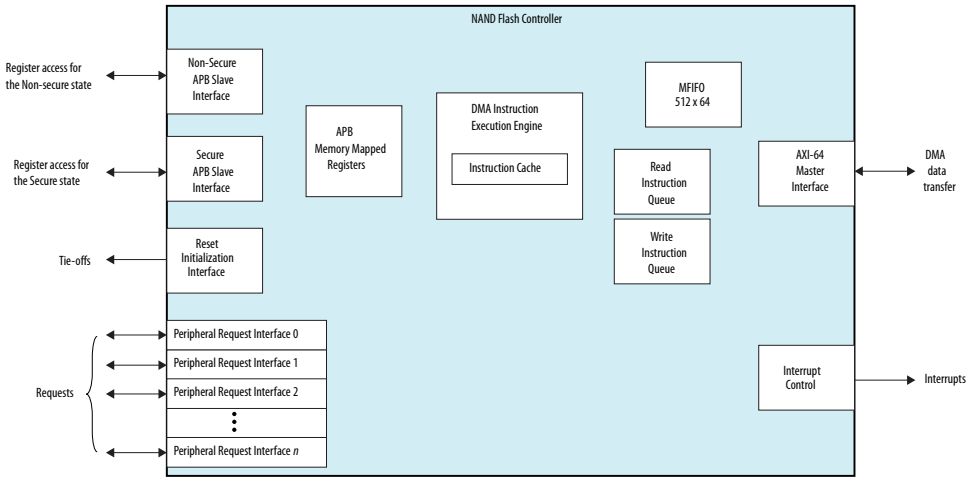
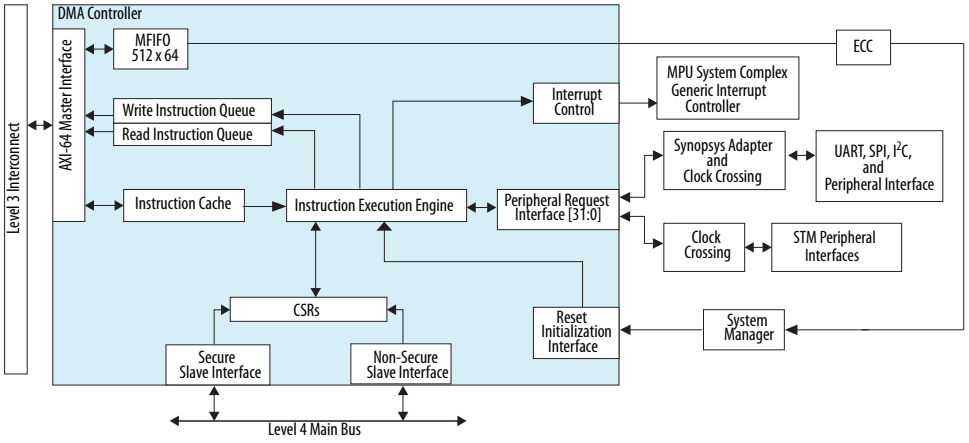


Figure 21. DMA Controller Connectivity

The following figure shows the 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 21 on page 118. 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.

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.

After the Cortex-A53 MPCore 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 section: *Reset Signals and Registers* in the *Reset Manager* chapter.

You should ensure that both the DMA ECC RAM and the DMA Module resets are deasserted before beginning transactions. Program the `dmaocp` bits and the `dma` bits in the `per0modrst` register of the Reset Manager to deassert reset in the DMA ECC RAM and the DMA module, respectively.

Related Information

Error Checking and Correction Controller on page 128

8.2.1. Distributed Virtual Memory Support

The system memory management unit (SMMU) in the HPS supports distributed virtual memory transactions initiated by masters.

As part of the SMMU, a translation buffer unit (TBU) sits between the DMA Controller and the L3 interconnect. The TBU contains a micro translation lookaside buffer (TLB) that holds cached page table walk results from a translation control unit (TCU) in the SMMU. For every virtual memory transaction that this master initiates, the TBU compares the virtual address against the translations stored in its buffer to see if a physical translation exists. If a translation does not exist, the TCU performs a page table walk. This SMMU integration allows the DMA driver to pass virtual addresses directly to the DMA without having to perform virtual to physical address translations through the operating system.

For more information about distributed virtual memory support and the SMMU, refer to the *System Memory Management Unit* chapter.

Related Information

System Memory Management Unit on page 71

8.3. Functional Description of the DMA Controller

This section describes the major interfaces and components of the DMAC and its operation.

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.

For more information, refer to the *CoreLink DMA-330 DMA Controller Technical Reference Manual* on the Arm Infocenter website.

The DMAC includes a 16-line instruction cache to improve the instruction fetch performance. Each instruction cache line contains eight, 4-byte words for a total cache line size of 32 bytes. The DMAC instruction cache size is, 16 lines times 32 bytes per line which equals 512 bytes. There is no mechanism to preload the program code into the DMA cache before the corresponding thread starts executing. The initial latency to bring the code into cache and start executing is dependent on where the code resides. This inital latency should be taken into consideration at system level to optimize the DMA transfers and expected performance. When a thread requests an instruction from an address, the cache performs a lookup. If a cache hit occurs, 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 fill 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 bus. 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 nine 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.

Related Information

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.

8.3.1. Error Checking and Correction

There is a dual port SRAM local memory buffer that provides ECC capability and is 64 by 512 bits, providing 4096 bytes of memory. One of the ports is always used for writes, and the other is always used for reads. The ECC block is integrated around the SRAM local memory buffer and provides the following features:

- Output to notify the system manager when single-bit correctable errors are detected and corrected
- Output to notify the system manager when double-bit uncorrectable errors are detected
- Provision for the injection of single-bit and double-bit errors for test purposes

The system manager provides registers to set or mask single-bit or double-bit ECC error interrupts.

Related Information

Error Checking and Correction Controller on page 128

8.3.1.1. Initializing and Clearing of Memory before Enabling ECC

Due to the DMA controller FIFO implementation, you must initialize and clear the FIFO before you enable the ECC to avoid a single event upset (SEU).

The following describes the initialization requirements:

- You must write a known pattern for data and ECC syndrome bits in memory, which involves the initialization of data to zero and corresponding nonzero ECC in hardware.
- Software must wait for the initialization process to complete before memory access is allowed. The initialization process cannot be interrupted nor stopped.

8.3.2. Peripheral Request Interface

The DMAC provides 32 peripheral request interfaces, which can be enabled on an individual basis. The HPS makes eight of these interfaces available to the FPGA, which allows for FPGA soft logic to request a DMA transfer. Two of the eight interfaces are shared by the HPS I^2C EMAC2 peripheral under software control. The eight DMAC peripheral request interfaces to the FPGA can be individually enabled using the HPS Platform Designer IP component. For DMA transfers to or from the FPGA, this feature is only necessary if your design requires transfer flow control.

Each FPGA peripheral request interface enabled using the HPS Platform Designer IP component contains the following set of signals exported to the FPGA, where <n> corresponds to a specific request interface enabled in Platform Designer:

- `f2h_dma<n>_req`—FPGA peripheral request to the HPS DMAC for a DMA transfer
- `f2h_dma<n>_ack`—HPS DMAC acknowledgment to the FPGA peripheral's request for a DMA transfer
- `f2h_dma<n>_single`—FPGA peripheral request to the HPS DMAC for a single, non-burst transfer

8.3.2.1. Peripheral Request Interface Mapping

You can assign a peripheral request interface to any of the DMA channels.

The DMAC supports 32 peripheral request interfaces. Each request interface can receive up to one outstanding request and is assigned a specific peripheral device ID. The following table lists the peripheral device ID assignments.

Table 72. Peripheral Request Interface Mapping

Peripheral	Request Interface ID
FPGA 0	0
FPGA 1	1
FPGA 2	2
FPGA 3	3
FPGA 4	4
FPGA 5	5
FPGA 6/I2C EMAC2 Tx(12)	6
FPGA 7/I2C EMAC2 Rx(12)	7
I2C0 Tx	8
I2C0 Rx	9
I2C1 Tx	10
I2C1 Rx	11
I2C EMAC0 Tx	12
I2C EMAC0 Rx	13
I2C EMAC1 Tx	14
I2C EMAC1 Rx	15
SPI0 Master Tx	16
SPI0 Master Rx	17
SPI0 Slave Tx	18
SPI0 Slave Rx	19
SPI1 Master Tx	20
SPI1 Master Rx	21
SPI1 Slave Tx	22
SPI1 Slave Rx	23
Reserved	24
Reserved	25
STM	26

continued...

(12) These interfaces are MUXed and controlled by software; since the HPS requires a total of 33 peripheral request interfaces and the DMAC supports a maximum of 32 interfaces.

Peripheral	Request Interface ID
Reserved	27
UART0 Tx	28
UART0 Rx	29
UART1 Tx	30
UART1 Rx	31

8.4. DMA Controller Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

9. On-Chip RAM

The hard processor system (HPS) contains a synchronous single-port RAM with an AXI interface. The on-chip RAM provides 256 KB of general-purpose memory.

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

9.1. Features of the On-Chip RAM

The on-chip RAM offers the following features:

- 64-bit slave interface
- 256 KB of synchronous single-port RAM
- Memory read acceptance is two and write acceptance is two with a total acceptance of four
- Read latency is four clock cycles and write latency is two clock cycles
- Supports Normal-exclusive accesses
- Error correction code (ECC) support(13)

Related Information

- Clock Manager on page 149
- Error Checking and Correction Controller on page 128

9.2. On-Chip RAM Interfaces

The on-chip RAM interfaces to the following:

- Clock manager
- Reset manager
- System manager
- CCU
- L3 interconnect

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

9.3. Functional Description of the On-Chip RAM

The on-chip RAM uses a 64-bit slave interface which consists of a single-ported SRAM.

The boot software copies initial software (pre-Bootloader) from the boot device into the on-chip RAM and executes the initial software by jumping to a known address in the on-chip RAM. The on-chip RAM also serves as a general-purpose memory that allows the FPGA fast access.

9.3.1. Read and Write Double-Bit Bus Errors

The integrated ECC Controller provides ECC protection to the on-chip RAM. The ECC controller detects and corrects single-bit errors. Double-bit errors are detected, but not corrected.

There are two types of double-bit bus errors:

- Read—Double-bit errors that occur during a read access from the slave interface are reported back on the slave interface using the read bus error.
- Write—Double-bit errors that occur during a write access from the slave interface are reported back on the slave interface using the read bus error.

This error response is true only if the corresponding double-bit error generation is enabled in the CTRL register of the ECC controller.

9.3.2. On-Chip RAM Controller

The on-chip RAM is a single-port memory that is composed of:

- Five FIFOs that register inputs and outputs on the AXI4 bus. Each of the five FIFOs correspond to an AXI4 channel:
  - Write address
  - Write data
  - Write response
  - Read address
  - Read data
- Each FIFO holds two entries to support the on-chip RAM's memory acceptance. The FIFO indicates to the slave bus when it is ready to accept another entry.
- Arbiter—The SRAM is a single-port design which means that only one read or one write can be executed in any given clock. The Arbiter grants either read or write access to the memory in a round-robin fashion.

9.3.3. On-Chip RAM Burst Support

The on-chip RAM AXI bus interface supports INCR and WRAP burst types for both reads and writes. The on-chip RAM does not support fixed bursts greater than a length of one beat. If a fixed burst greater than 1 is attempted a SLVERR is returned on the bus.

The on-chip RAM supports the following burst features:

Table 73. Burst Features

Feature	Description
Burst types	• FIXED—Only the one-beat fixed burst is supported and has a length of 1 • INCR—1 to 256 • WRAP—2, 4, 8, or 16
Burst size	1, 2, 4, 8 bytes For any access lower than 8 bytes, the controller determines which bytes are valid.
Burst lengths	1 to 16 beats
Latency	Supports back to back single beat bursts. This applies to reads, writes, and combined reads and writes.
Error response	If the fixed burst length is greater than 1, a SLVERR is returned.

9.3.4. Exclusive Access Support

The On-chip RAM is connected to the CCU. The CCU provides an exclusive monitor to the OCRAM for accesses to cached memory.

9.3.5. Sub-word Accesses

The on-chip RAM supports sub-word accesses. Sub-word accesses are performed using a read-modify-write access independent of whether the ECC is enabled or not.

9.3.5.1. Pipeline and Timing

The RAM controller is a pipelined design, where:

- The Write path (full width) pipeline is two stages deep; therefore providing a two clock latency.
- The Write path (subword access) pipeline is three stages deep to account for the read-modify-write.
- The Read path pipeline is four stages deep for RAM.

9.3.6. On-Chip RAM Clocks

The on-chip RAM is driven by the `mpu_ccu_clk` interconnect clock.

Source	Functional Usage	Value
`mpu_ccu_clk`	RAM AXI interconnect clock	`mpu_ccu_clk` is 1/2 of the `mpu_clk`

The on-chip RAM ECC registers operate on a different clock domain.

Source	Functional Usage	Value
`mpu_periph_clk`	ECC Register interface clock	`mpu_periph_clk` is 1/4 of the `mpu_clk`

9.3.7. On-Chip RAM Resets

During a cold or warm reset, the contents of the RAM remain unchanged. The reset only clears the state on the AXI bus.

Name	Functional Usage	Comments
ram_rst_n	RAM AXI interconnect reset	Asynchronously asserted, synchronously de-asserted to osc1_clk

9.3.8. On-Chip RAM Initialization

You must initialize the on-chip RAM before you enable the ECC. Failure to do so triggers spurious interrupts.

9.3.9. ECC Protection

The ECC controller operation and functionality is programmable through the ECC register slave interface. 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 128

9.4. On-Chip RAM Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

10. 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 Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

10.1. 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

10.2. 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
- DMA controller
- NAND flash controller

Note: The L2 cache and the SDRAM interface have their own dedicated ECC support.

Related Information

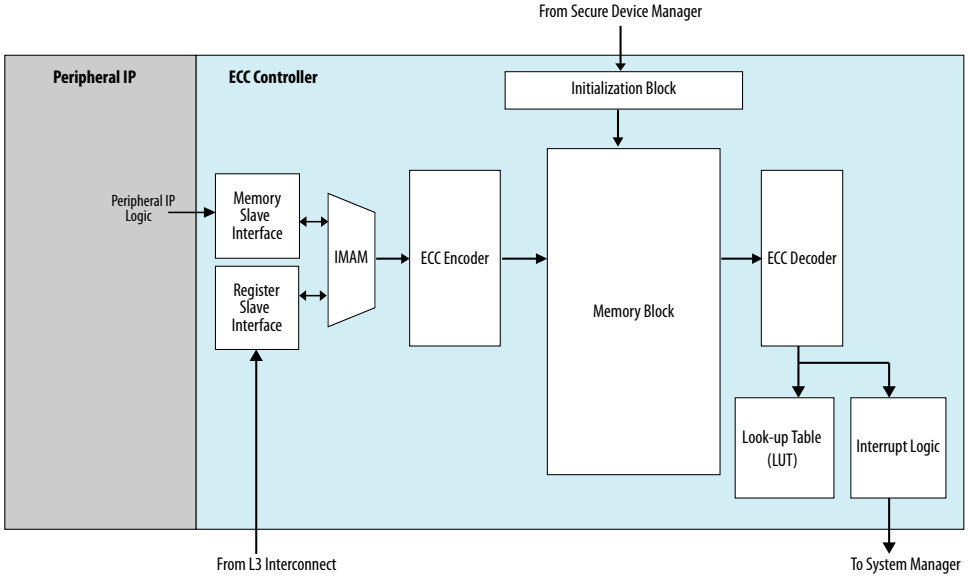
System Interconnect on page 82
For more information regarding SDRAM interface ECC support.

10.3. 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 22. 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) system complex 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 Reset Manager when memory clearing is required.

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 133

10.4. ECC Controller Functional Description

10.4.1. 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.

10.4.2. ECC Structure

The ECC is calculated based on a Hamming code for the corresponding data word length.

Table 74. 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 75. 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 only the DMA is 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.

The on-chip RAM is word-addressable. It supports sub-word accesses, however, through a read-modify-write operation. For example, accessing byte 2 of an on-chip RAM word causes a data read of the whole word with ECC. If the word passes the syndrome check, then the byte 2 data is concatenated with the other three bytes of the original data. ECC is recalculated and data is written to memory.

Peripheral Memory	Data Size	Memory	ECC Bits	Data Width + ECC Bits	Hamming Code Word (length in bits)	Type [14]
On-chip RAM	64 x 32768	Word-addressable	8	64+7[15]	72	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 4096	Word-addressable	7	35+7	42	Simple dual port
DMA FIFO	64 x 512	Byte-addressable	5 per byte lane	64+40[16]	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

10.4.2.1. RAM and ECC Memory Organization Example

The DMA has 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.

(14) 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.

(15) Uses read-modify-write for subword accesses

(16) This is the same as 8 byte lanes with 5 ECC bits per lane.

Table 76. 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]

Table 77. 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]

10.4.3. 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 pre-fetches 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. The Cortex-A53 MPCore and secure device manager (SDM) can directly access the register slave interface and initialize the memory

When a tamper event occurs, the SDM uses the initialization block to scramble all of the ECC memories. The SDM can initiate this memory scrambling as part of a secure boot, secure configuration or authentication and at any time during functional or non-functional mode.

10.4.4. 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.

10.4.4.1. 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.

10.4.4.2. 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.

10.4.4.3. 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.

10.4.4.4. Memory Testing

You can perform all memory diagnostic testing through the register slave interface. Additionally, the DMA can also be tested through the peripheral slave interface. You can run ECC diagnostics when the ECC register interface is out of reset and idle or when the peripheral is in reset

Each peripheral with ECC RAM has two separate reset control bits. One bit controls the peripheral reset and one bit controls the peripheral's ECC register interface reset.

It is recommended to run ECC diagnostics of peripheral memories that are indirectly addressable before operating the peripheral in functional mode. Diagnostics can only be run if the peripheral ECC register interface is out of reset but idle or if the peripheral itself is in reset. The peripheral ECC register interface is out of reset and idle when:

- The peripheral's *ocp bit in the per0modrst register of the Reset Manager is clear
- The ECC_EN bit in the CTRL register of the peripheral's ECC register set is clear

When the ECC diagnostics have completed, software can bring the ECC register interface out of reset if it is still in reset and configure the ECC registers.

10.4.4.4.1. 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 for DMA ECC RAM

This sequence tests the single-bit error detection and correction in the ECC decoder of the DMA ECC RAM.

1. Write data to the ECC_WData3bus through ECC_WData0bus registers.
2. Set the ECC_EN bit in the CTRL register to enable the ECC detection and correction logic.
3. Set the DBEN bit in the ECC_dbytectrl register.
4. Select the address bus to write the data to by programming the ECC_Addrbus register.
5. In the ECC_accctrl register, program the following bits:
   - RDWR=1
   - ECCOVR=0
   - DATAOVR=1

6. Set the ENBUS* bit in the ECC_startacc register to trigger an indirect write access.
7. Clear the ECC_EN bit in the CTRL register to disable the ECC detection and correction logic.
8. Write a data value that has one bit altered in the ECC_WData3bus through ECC_WData0bus registers to the same address.
9. Set the ENBUS* bit in the ECC_startacc register to trigger an indirect write access.
10. In the ECC_accctrl register, program the following bits:
    - RDWR=0
    - ECCOVR=1
    - DATAOVR=1
11. Set the ECC_EN bit in the CTRL register to enable the ECC detection and correction logic.
12. Set the ENBUS* bit in the ECC_startacc register to trigger an indirect write access.
    If you have configured an interrupt to trigger for a single-bit error, then expect it to trigger after these steps have completed. If you read back the data at the same address using the ECC_RData*bus register, expect to see a corrected data result from the memories.

Single-Bit Error Test for Word-Writeable Memories

This sequence tests the single-bit error detection and correction in the ECC decoder of the word-writeable ECC RAMs.

1. Write data to the ECC_WData3bus through ECC_WData0bus registers.
2. Set the ECC_EN bit in the CTRL register to enable the ECC detection and correction logic.
3. Set the DBEN bit in the ECC_dbytectrl register.
4. Select the address bus to write the data to by programming the ECC_Addrbus register.
5. In the ECC_accctrl register, program the following bits:
    - RDWR=1
    - ECCOVR=0
    - DATAOVR=1
6. Set the ENBUS* bit in the ECC_startacc register to trigger an indirect write access.
7. In the ECC_accctrl register, program the following bits:
    - RDWR=0
    - ECCOVR=1
    - DATAOVR=0
8. Set the ENBUS* bit in the ECC_startacc register to trigger an indirect write access.

9. Write a data value that has one bit altered in the ECC_WData3bus through ECC_WData0bus registers to the same address.
10. Read the resultant data from the ECC_RDataecc*bus registers at the same address.
11. Write the value from the ECC_RDataecc*bus registers into the ECC_WDataecc*bus registers.
12. In the ECC_accctrl register, program the following bits:
    - RDWR=1
    - ECCOVR=1
    - DATAOVR=1
13. Set the ENBUS* bit in the ECC_startacc register to trigger an indirect write access.
14. In the ECC_accctrl register, program the following bits:
    - RDWR=0
    - ECCOVR=1
    - DATAOVR=1
15. Set the ENBUS* bit in the ECC_startacc register to trigger an indirect write access.
    If you have configured an interrupt to trigger for a single-bit error, then expect it to trigger after these steps have completed. If you read back the data at the same address using the ECC_RData*bus register, expect to see a corrected data result from the memories.

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. The correct ECC data should 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 without data correction. Refer to the *Error Logging* section for more details about identifying errors.

Related Information

Error Logging on page 140

10.4.4.4.2. Peripheral Slave Interface Tests for DMA ECC RAM

Only the DMA ECC RAM can be tested using the peripheral slave interface. 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 through the register bus interface. Expect the data read to match the data originally written (with or without a single-bit error) or a double-bit error to be logged. 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. Expect no ECC value to be generated and no interrupt or error logging to occur.
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. Expect no ECC value to be generated and no interrupt or error logging to occur.
3. Enable the ECC by setting the ECC_EN bit in the CTRL register.
4. Read data from the 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

Error Logging on page 140

10.4.4.5. 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', you 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 23. 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 24. 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 25. 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	1	1	1	1	0	0	0	0	1	-	-	1	-	-	-	-
S3	0	1	1	1	0	0	0	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 26. 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	0	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	0	1	0	0	1	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 27. 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	0	1	1	0	0	0	1	0	0	1	0	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	1	0	1	0	1	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	0	1	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

10.4.5. 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.

10.4.5.1. 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.

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 130
Refer to the "ECC Structure" section for more information regarding ECC Structure and Hamming Code word lengths.

10.4.5.2. 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.

10.4.5.3. 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 78. LUT Depth for HPS ECC Memories

Peripheral	LUT Depth (entries)
On-chip RAM	16
USB	4
SD/MMC	4 x 2 [17]
Ethernet MAC (Rx FIFO)	4
Ethernet MAC (Tx FIFO)	4
DMA	4
NAND (ECC Buffer)	4
NAND (Write FIFO)	4
NAND (Read FIFO)	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 142
  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 148

10.4.6. 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 ArmCortex-A53 MPCore. The following steps outline the interrupt generation process when interrupts have been enabled through the Error Interrupt Enable (ERRINTEN) register.

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.

(17) 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.

10.4.6.1. 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.

Single-bit errors that occur during a read-modify-write cycle for a sub-word access are flagged in the MODSTAT register in addition to triggering an interrupt.

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.

Note: Because the DMA has eight individual decoders for each byte lane of its byte-accessible memory, the DECODERSTAT register provides extra information to the INSTAT register that indicates which of the individual decoders is flagging a single-bit error. All other ECC RAMs supported only have one decoder.

Related Information

ECC Controller Address Map and Register Descriptions on page 148

10.4.6.1.1. 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.

10.4.6.1.2. 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 79. 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.

10.4.6.1.3. 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. If 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 can run out again before the ISR exits. If this happens, and you clear the interrupt and exit the ISR, then the new counter match condition is never detected. 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 148

10.4.6.2. 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 occured. 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.

Double-bit errors thta occur during a read-modify-write cycle for a sub-word access are flagged in the MODSTAT register in addition to triggering an interrupt.

Note: Because the DMA has eight individual decoders for each byte lane of its byte-accessible memory, the DECODERSTAT register provides extra information to the INSTAT register that indicates which of the individual decoders is flagging a single-bit error. All other ECC RAMs supported only have one decoder.

Related Information

ECC Controller Address Map and Register Descriptions on page 148

10.4.6.3. 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.

10.4.7. ECC Controller Initialization and Configuration

You can initialize memory and run ECC diagnostics when the ECC register interface is out of reset and idle or when the IP is in reset. Peripherals that access memories with ECC enabled must run hardware initialization prior to using the peripheral memory.

Note: Software must not perform read or write accesses to memory during hardware intialization.

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.

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 133
- ECC Structure on page 130
- ECC Controller Interrupts on page 142
  For details of single-bit, count, and compare-match interrupts.

10.4.8. 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 80. 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`

Related Information

Clock Manager on page 149
For details of the clock signals for each ECC controller.

10.4.9. ECC Controller Reset

To access a peripheral's RAM and ECC configuration registers, both the peripheral and the ECC register interface must be out of reset.

- Bring the peripheral out of reset by clearing the peripheral's corresponding reset bit in the `per0modrst` register of the Reset Manager.
- Bring the peripheral's ECC register port out of reset by clearing the `*ocp` bit in the `per0modrst` register of the Reset Manager.

A cold or warm reset does not change the contents in the ECC RAM. These resets only clear the state associated with the bus slave.

Related Information

Reset Manager on page 161
For more information regarding reset, refer to the *Reset Manager* chapter.

10.5. ECC Controller Address Map and Register Descriptions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

11. 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 which may be synchronous to each other. The Clock Manager has two phase-locked loop (PLL) clock group where the clock source is a common PLL voltage-controlled oscillator (VCO). A clock group which is independent and asynchronous to other clocks may only have single clock, also known as clock slice. Peripheral clocks are a group of independent clock slices.

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

11.1. Features of the Clock Manager

The *Clock Manager* offers the following features:

- Generates and manages clocks in the HPS
- Contains the following clock groups:
  - MPU clock group:
    - Cortex A-53 MPCore, CCU, GIC, and SMMU components
  - Interconnect clock group:
    - L3 clocks
    - CoreSight clocks
    - L4 clocks
  - Peripheral clock group:
    - GPIO clocks
    - EMAC0/1/2 clocks
    - SDMMC clocks
    - SoC-to-FPGA clocks
  - Other peripherals (NAND, SPI, USB) connect to Interconnect clocks (L3/L4).
- Contains two flexible PLL blocks to drive any of the above clocks:
  - Main PLL
  - Peripheral PLL
- 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 4 post-scale counters for each PLL
  - VCO calibration for each PLL
  - Additional post-scale counters in 9 of 10 clock groups (MPU clock group excluded)
  - Bypass modes for each PLL
  - Gate of individual clocks in all PLL clock groups and clock slices.
  - Boot mode for hardware-managed clocks
  - General-purpose I/O (GPIO) debounce clock divide
- Supports interrupting the Cortex-A53 MPCore on PLL-lock and loss-of-lock.

You must use Platform Designer to configure HPS clock functionality, sources, outputs and frequency values. Platform Designer checks your HPS clock configuration and generate handoff information for boot firmware generation tools to ensure the following requirements are met:

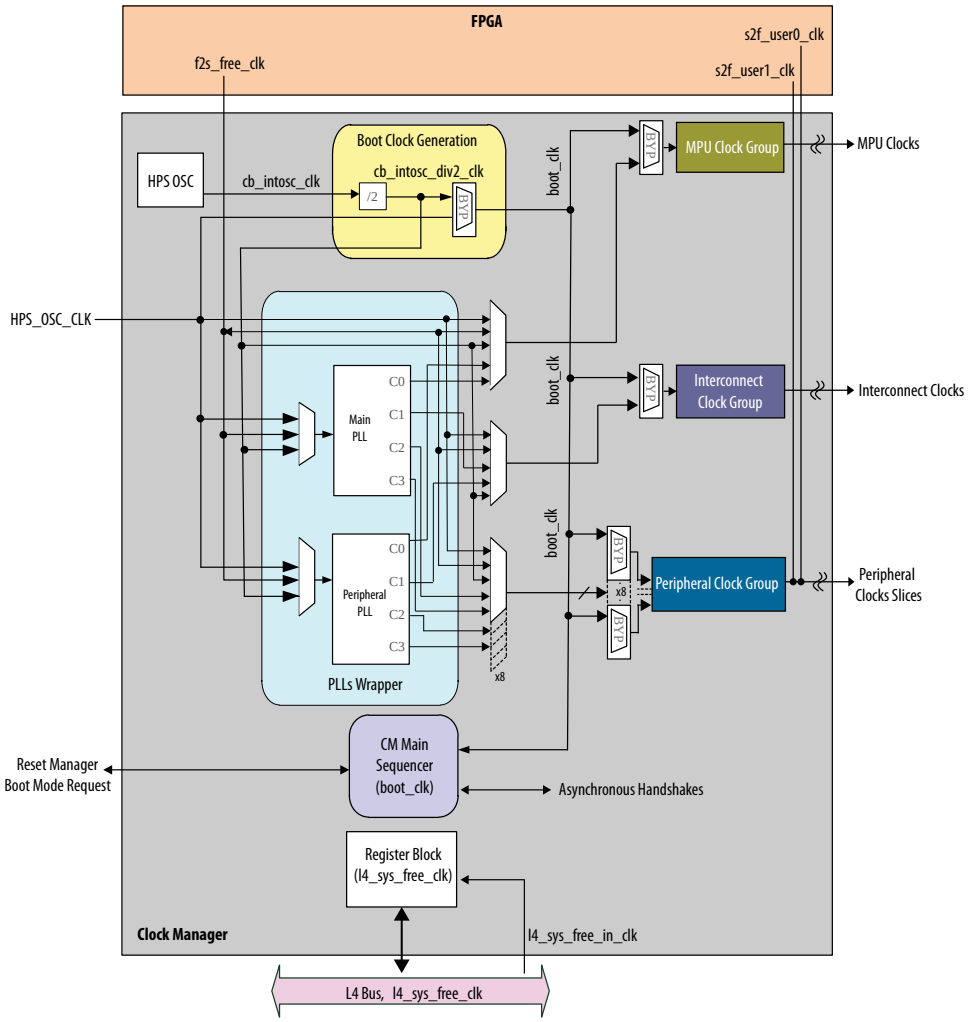
- Routing of FPGA-to-SoC and SoC-to-FPGA clocks. Platform Designer is responsible for routing and configuring clocks between the HPS. Only the SoC-to-FPGA clock are managed within the clock manager.
- Software must not program the clock manager with illegal values. If it does, the behavior of the clock 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 155

11.2. Top Level Clocks

Figure 28. Clock Manager Block Diagram



The Clock Manager contains 2 PLLs, the Main PLL and Peripheral PLL. Inputs into these two PLLs can come from the input pin HPS_OSC_CLK, the internal oscillator, `cb_intosc_div2_clk` or the `f2s_free_clk` FPGA clock input. Both PLLs generate clock outputs to be used by the output clock blocks shown in the diagram. Output clock blocks include the MPU clock block, the Interconnect clock block and the peripheral clock block. The peripheral clock block is comprised of GPIO, EMAC, SDMMC, and SoC-to-FPGA clocks.

Note: You must select one of the 48 HPS Dedicated I/O pin in Platform Designer to function as HPS_OSC_CLK

The clock from each of these output clock blocks is sourced from either the bypass clock (`boot_clk`) or a non-bypass clock. A non-bypass clock can be one of five sources.

Table 81. Non-Bypass Clock Sources

Source	Description
HPS_OSC_CLK	Pin for external oscillator (selected from one of the 48 HPS Dedicated I/O)
f2s_free_clk	FPGA fabric PLL clock reference
cb_intosc_div2_clk	Internal ring oscillator divided by 2 (230 MHz maximum)
PLL0 Counter Output	Main PLL counter outputs
PLL1 Counter Output	Peripheral PLL counter outputs

Table 82. Top Level Clocks

Clock Name	Source/Destination	Description
mpu_free_clk	Clock manager To MPU complex	Source clock from clock manager for both the MPU clock groups.
mpu_clk	Internal to MPU complex	MPU main clock
mpu_ccu_clk	Main clock for CCU. 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 Interconnect/ Peripherals	Interconnect L3 main switch clock. Always free running.
l4_sys_free_clk	Clock manager to Interconnect/ Peripherals	Interconnect L4 system clock. Always free running.
l4_main_clk	Clock manager to Interconnect/ Peripherals	L4 Interconnect clock for fast peripherals including DMA, SPIM, SPIS and TCM.
l4_mp_clk	Clock manager to Interconnect/ Peripherals	Interconnect L4 peripheral clock for peripherals including NAND, USB, and SDMMC.
l4_sp_clk	Clock manager to Interconnect/ Peripherals	Interconnect L4 slow peripheral clock for peripherals including Timer, I^2C, and UART.
cs_at_clk	Clock manager to CoreSight	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 will be independent and defaults to a low frequency (25 MHz) for lower speed debuggers.
s2f_user0_clk	SoC-to-FPGA fabric	General purpose interface clock to FPGA.
s2f_user1_clk	SoC-to-FPGA fabric	General purpose interface clock to FPGA.

11.2.1. 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_div2_clk` or the unsecure external oscillator. The `boot_clk` source is only updated coming out of cold reset or a warm reset (boot mode request) and is not changed at any other time. For normal operation, Intel recommends to use external oscillator as the internal oscillator is high variable and has slow speed.

All clocks are bypassed to boot clock while coming out of reset. Intel Quartus Prime may configure and lock the PLLs in boot mode. On exiting boot mode, all the clocks are gracefully transitioned to functional clocks.

The MPU and Interconnect (includes debug clocks) blocks contain enable outputs to define clock frequency ratios to the MPU, Interconnect and CoreSight logic.

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

11.3. Functional Description of the Clock Manager

11.3.1. Clock Manager Building Blocks

11.3.1.1. 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, output lock signal generation and configurable M/N VCO w/o fractional counter
- Four output dividers with a range of 1 to 2047 to further subdivide the clock
- A PLL can be configured to bypass all outputs to the input clock for glitch-free transitions

Related Information

- PLL Integration on page 154
- Hardware Sequenced Clock Groups on page 155
- Software Sequenced Clocks on page 157

11.3.1.2. Clock Gating

Clock gating enables and disables clock signals. Refer to the Main Group and 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 160

11.3.2. PLL Integration

The two PLLs contain exactly the same set of output clocks. PLL0 is intended to be used for the MPU and Interconnect clocks. PLL1 outputs are routed to the HPS master peripherals.

Figure 29. PLL Integration in Clock Manager

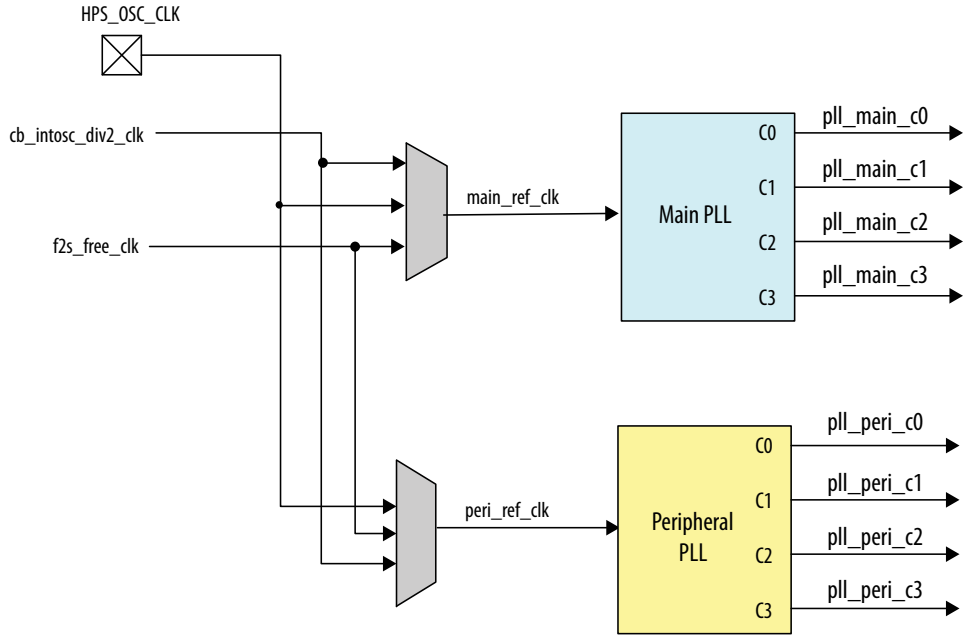


Table 83. PLL Direct Outputs

For Boot mode, the maximum and minimum frequency is 200 MHz and 10 MHz respectively.

PLL	Output Counter	Clock Name
Main PLL	C0	pll_main_c0
	C1	pll_main_c1
	C2	pll_main_c2
	C3	pll_main_c3
Peripheral PLL	C0	pll_peri_c0
	C1	pll_peri_c1
	C2	pll_peri_c2
	C3	pll_peri_c3

Note: The clock slice outputs of both the Main and Peripheral PLL are disabled out of reset.

11.3.3. 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.

The hardware-managed clocks are:

- `mpu_periph_clk`
- `mpu_ccu_clk`
- `mpu_clk`
- `l3_main_free_clk`
- `l4_sys_free_clk`
- `l4_main_clk`
- `l4_mp_clk`
- `l4_sp_clk`
- `cs_at_clk`
- `cs_pdbg_clk`
- `cs_trace_clk`

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

Note: During boot mode, all clocks are bypassed including both hardware-managed and software-managed clocks. Individual software bypasss controls are available for each set of clocks.

11.3.4. Hardware Sequenced Clock Groups

The hardware sequenced clock groups consists of the MPU clocks and the Interconnect 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 Interconnect has five and MPU has two software enables.

Table 84. Interconnect Clock Software Enables

Software Enable	Access	Description
csclken	RW	Enables Debug clock output (`cs_at_clk`, `cs_pdbg_clk`, `cs_trace_clk`)
l4spclken	RW	Enables clock `l4_sp_clk` output
l4mpclken	RW	Enables clock `l4_mp_clk` output
l4mainclken	RW	Enables clock `l4_main_clk` output

Table 85. MPU Clock Software Enables

Software Enable	Access	Description
mpuclken	RW	Enables mpu_clk, mpu_periph_clk, and mpu_ccu_clk to MPU interface

Figure 30. Hardware Clock Groups

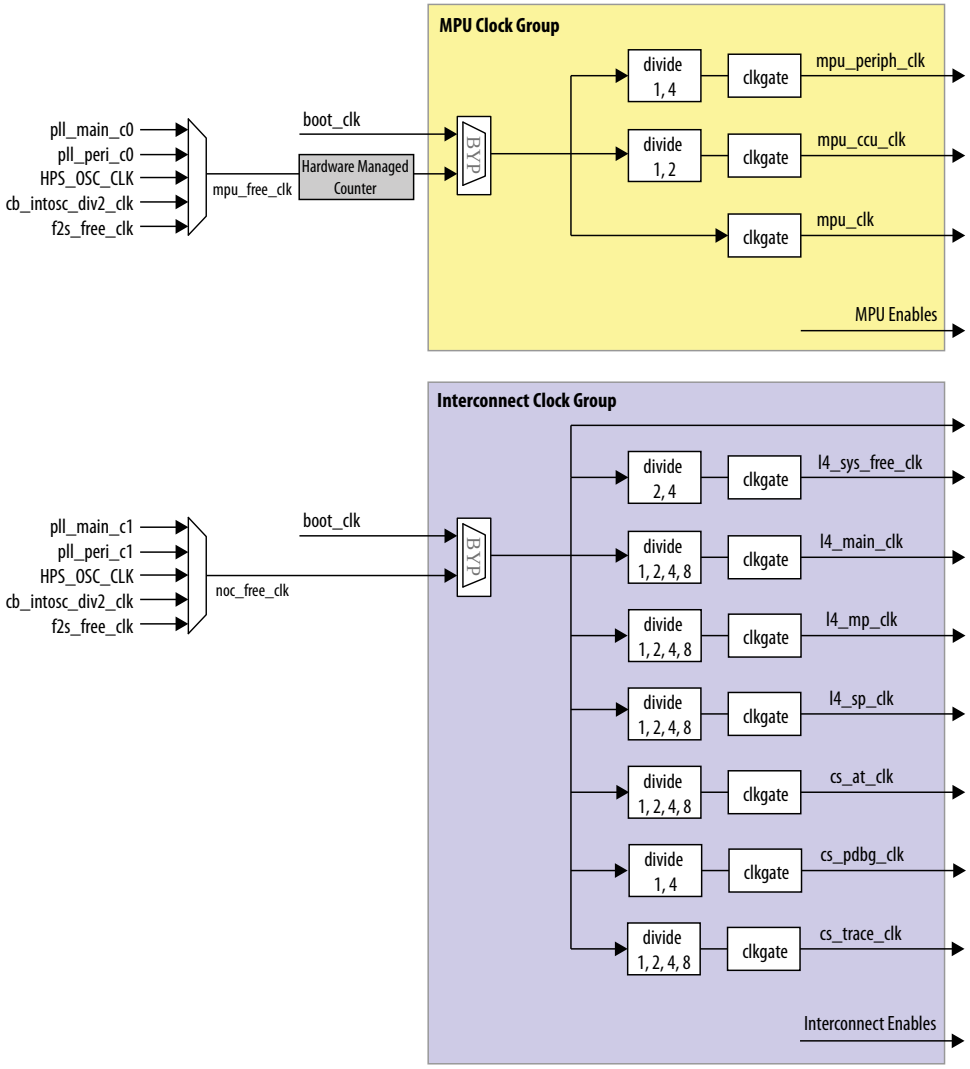


Table 86. The Hardware Sequenced Clocks Feature Summary

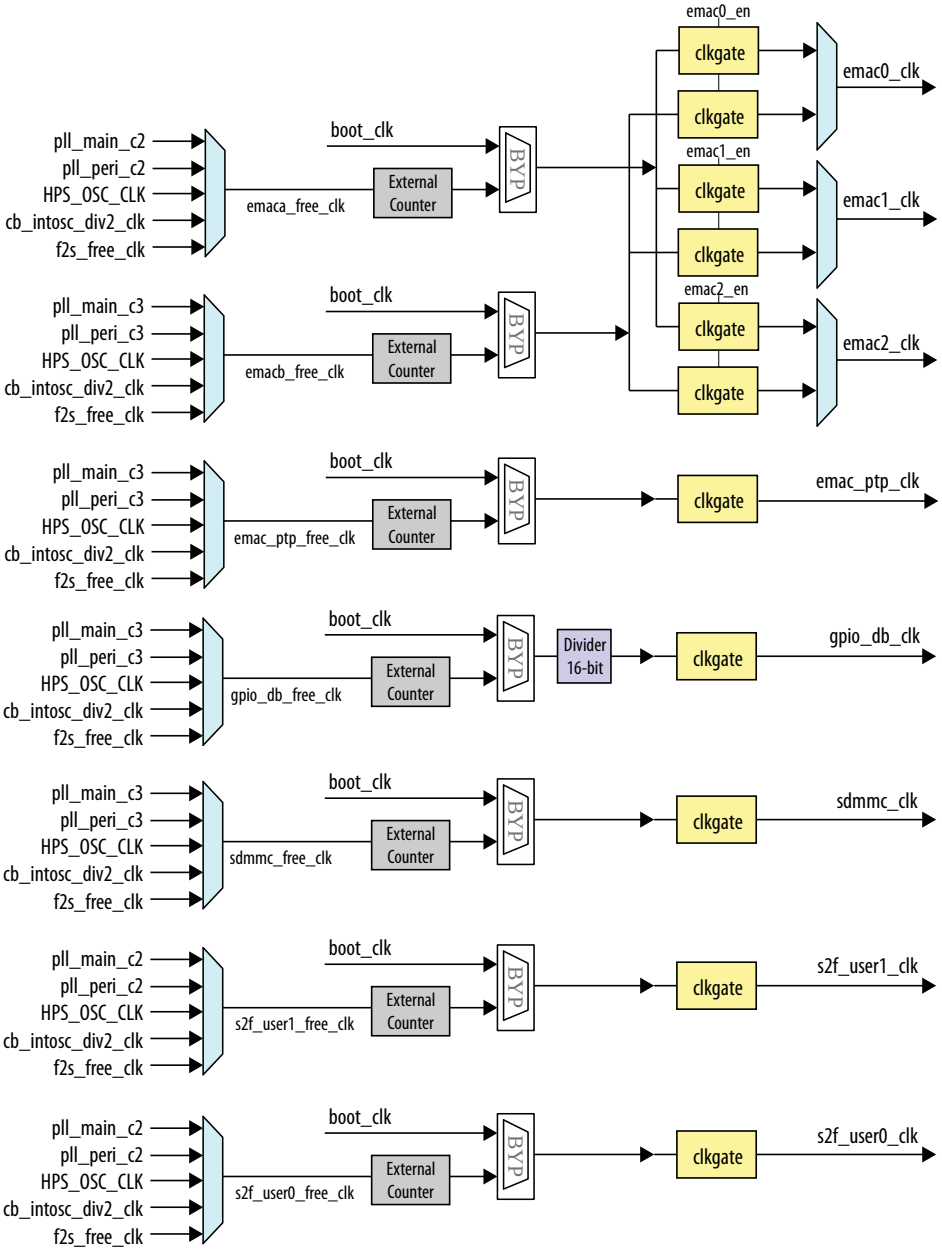
Clock Output Group	System Clock Name	Frequency[18] value.	Boot Frequency	Uses
MPU	mpu_clk	PLL C0	boot_clk	MPU system complex, including CPU0-3
	mpu_ccu_clk	mpu_clk/2	boot_clk	MPU level 2 (L2) RAM
	mpu_periph_clk	mpu_clk/4	boot_clk	MPU peripherals such as interrupts, timers, and watchdog
Interconnect	l3_main_free_clk	PLL C1	boot_clk	L3 interconnect
	l4_sys_free_clk	l3_main_free_clk/{2,4}	boot_clk/2	L4 interconnect
	l4_main_clk	l3_main_free_clk/{1,2,4,8}	boot_clk	L4 main bus
	l4_mp_clk	l3_main_free_clk/{1,2,4,8}	boot_clk	L4 MP bus
	l4_sp_clk	l3_main_free_clk/{1,2,4,8}	boot_clk/2	L4 SP bus
	cs_at_clk	l3_main_free_clk/{1,2,4,8}	boot_clk	CoreSight debug trace bus
	cs_pdbg_clk	l3_main_free_clk/{1,4}	boot_clk/2	Debug Access Port (DAP) and debug peripheral bus
	cs_trace_clk	l3_main_free_clk/{1,2,4,8}	boot_clk/4	CoreSight debug trace port Interface Unit (TPIU)

11.3.5. Software Sequenced Clocks

The software sequenced clock groups include additional clocks for peripherals not covered by the MPU and Interconnect clocks. The following diagram shows the external bypass muxes, hardware-managed external counters and dividers, and clock gates.

(18) All clock frequencies must be less than the F_{max}

Figure 31. 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 87. 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 C3	boot_clk	Clock for EMAC PTP timestamp clock
gpio_db_clk	125 Hz to PLL C3	boot_clk	Clock for GPIO debounce clock
sdmmc_clk	PLL C3	boot_clk	Clock for SDMMC
s2f_user0_clk	PLL C2	boot_clk	Clock reference for FPGA
s2f_user1_clk	PLL C2	boot_clk	Clock reference for FPGA

11.3.6. Resets

When the POR to the Clock Manager is de-asserted all the other modules in the system are still in reset. This ensures that the Clock Manager is the first module to come out of POR reset. Once the Clock Manager is out of reset, boot_clk is propagated on all the clocks going out from the clock manager.

Therefore, POR is the main reset domain for the clock manager. The Reset Manager also sends POR to the Cortex-A53 MPCore processor.

After a POR reset:

- All hardware-managed clocks are in Boot Mode and default to the boot_clk (cb_intosc_div2_clk) with all external dividers set to 1 (with the exceptions).
- All software-managed clocks will be in Boot Mode, bypassed to boot_clk.
- Clock gates are set to enable state, counters are set at their default value, and all external bypasses are set to boot_clk.

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.

When Reset manager issues a Boot Mode request to clock manager, these steps are followed:

1. Based on the status of the secure clock fuse during POR, Secure Device Manager (SDM) indicates if the boot clock should be secure.
   a. If secure clocks are enabled, boot_clk transitions gracefully to cb_intosc_div2_clk.
   b. If secure clocks are not enabled, boot_clk transitions gracefully to HPS_OSC_CLK.

Note: The security fuse is only sampled during cold reset and warm reset. The security fuse HPS CLK allows the user to enable secure clocks. If clearing RAM on a Cold or Warm reset, the user should enable secure clocks (`cb_intosc_clk` divide by 2).

2. 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.
3. After Hardware Managed Clocks have transitioned, the Clock Manager acknowledges the Reset Manager.

11.3.7. 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 Secure Device Manager (SDM).

The following table defines the boot clock sources.

Table 88. Security Input Clocks

Clock Name	Source	Description
`HPS_OSC_CLK`	Pin	External Oscillator Clock Reference. Non-secure clock reference.
`cb_intosc_clk`	HPS	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.

11.3.8. Interrupts

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

11.4. Clock Manager Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

12. Reset Manager

The reset manager generates module reset signals based on reset requests from the various sources in the HPS, and software writing to the module-reset control registers.

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

Table 89. HPS Reset Domains

Reset Domain	Reset Source	Description
POR (Power-on Reset)	Secure Device Manager (SDM)	SDM requests reset manager to assert POR reset. During a voltage tampering or out-of-range event, the SDM asserts POR. When voltage returns to operating range, the POR is de-asserted. During POR, the entire HPS and FPGA is reset. When the device is released from POR, SDM begins initialization.
System Cold Reset	• SDM (HPS mailbox message) • HPS_COLD_nRESET pin(19)	SDM requests reset manager to assert or de-assert cold reset.
System Warm Reset	• Software requests a warm reset through the EL3 register • Request from Cortex-A53 MPCore.	Reset manager asserts warm reset provided that the Cortex-A53 MPCore is idle. During warm reset, the CoreSight logic is not in reset, therefore the debug/trace can continue immediately after reset manager de-asserts warm reset. *Note:* An L2 reset must be performed before requesting a warm reset. Before you request L2 reset via software, you must flush L2 using the `l2flushen` bit of the `hdsken` register.
Watchdog Reset	Watchdog Timeout Event	Reset manager asserts watchdog reset based on the watchdog timer register. As the CoreSight logic is not reset, the debug/trace can continue immediately after reset manager de-asserts watchdog reset.
MPU Cold Reset	Software requests a cold reset through the COLDMODRST register	Reset manager asserts cold reset to the MPU provided that all four cores are idle. *Note:* Before you request MPU cold reset via software, you must idle all four cores using a WFI instruction and flush L2 using the `l2flushen` bit of the `hdsken` register.

continued...

(19) You can assign HPS_COLD_nRESET to an available SDM I/O pin. This pin serves both as an input to reset the HPS and as an output to assert reset to the system when the HPS is in reset. You can configure this pin using the Intel Quartus Prime Pro Edition, under **Device and Pin options > Configuration > Configuration pin** option.

Reset Domain	Reset Source	Description
CPU Cold Reset		Reset manager asserts cold reset to the requested core provided that that the core and L2 is idle (execute a WFI instruction).
CPU Warm Reset	Software requests a warm reset through the MPUMODRST register	Reset manager asserts warm reset to the requested core provided that that core is idle (execute a WFI instruction).
Debug Reset	Software requests a debug reset through the DBGMODRST register	The DBGMODRST register has two dedicated bits, one each for DAP and debug logic. Reset manager asserts reset for both DAP and debug logic. Software must clear debug reset bit to resume debugging.

Related Information

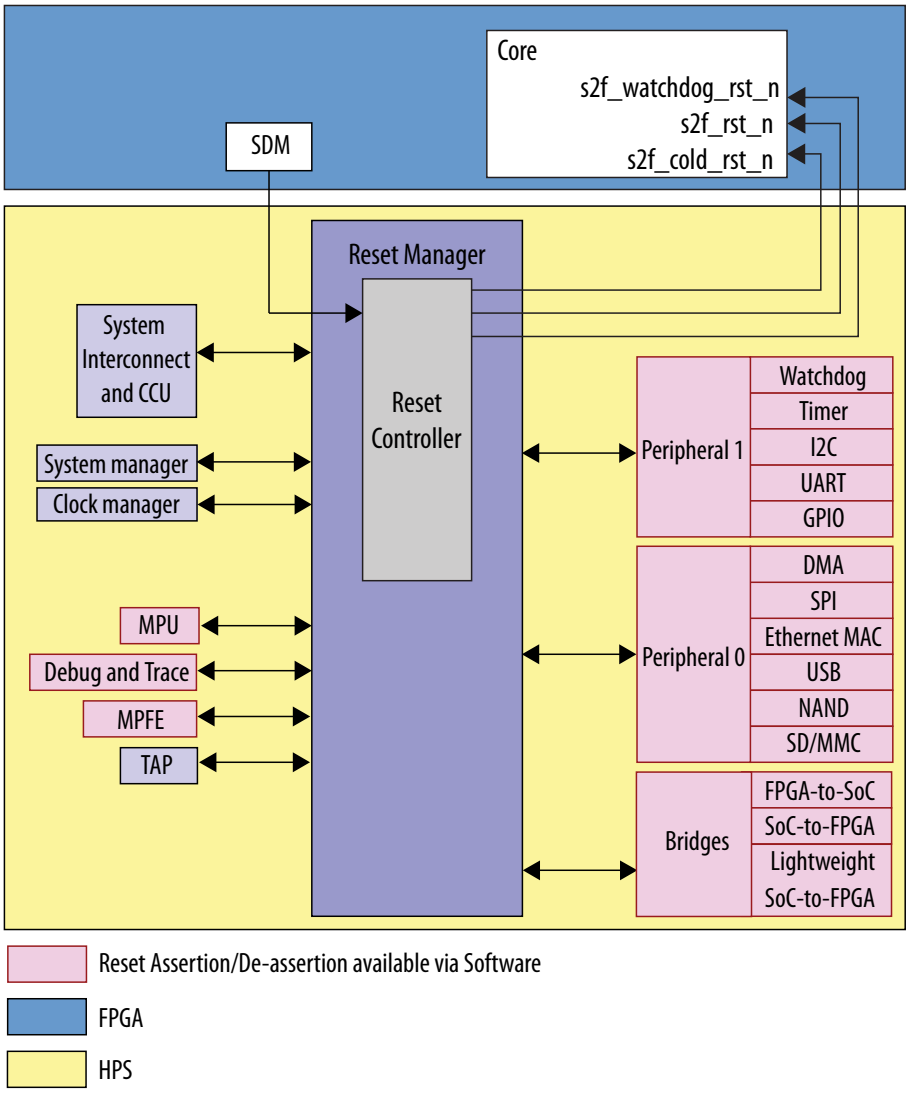
Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

12.1. Functional Description

The reset manager performs the following functions:

- Accepts reset requests from the SDM, and software.
- Generates reset signals to modules in the HPS and to the FPGA fabric. The following actions generate reset signals:
  - Using software to write the MPUMODRST, PER0MODRST, PER1MODRST, BRGMODRST, COLDMODRST, or DBGMODRST module reset control registers.
  - Asserting the HPS_COLD_nRESET signal triggers the reset controller and s2f_cold_rst_n signal.
- Provides reset handshaking signals to support system reset behavior.

Figure 32. Reset Manager Block Diagram



Multiple reset requests can be driven to the reset manager at the same time. Higher priority reset requests can preempt lower priority requests if the lower priority request has not been committed, that is if the reset acknowledgment process is incomplete. If a lower priority request is committed, then a higher priority request is delayed until the lower priority reset completes. There is no priority difference among reset requests within the same domain.

Table 90. Reset Priority

Ongoing Reset	Start of New Reset	Action Taken by Reset Manager
Cold reset	Cold reset	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

continued...

Ongoing Reset	Start of New Reset	Action Taken by Reset Manager
		manager is removing other modules out of the reset state, the reset manager returns those modules back to the reset state.
Warm reset	Watchdog reset	If warm reset is not committed: • Execute watchdog reset. • Complete the watchdog reset assertion and de-assertion. If warm reset is committed, queue the watchdog reset and • Execute warm reset. • Then, complete the pending watchdog reset.
Warm reset	Cold reset	If warm reset is not committed: • Execute cold reset. • Complete the cold reset assertion and de-assertion. If warm reset is committed, queue the cold reset and • Execute warm reset. • Then, complete the pending cold reset.
Warm reset	Any other reset initiated by software	Continue warm reset regardless of whether warm reset is committed or not.
Watchdog reset	Cold reset	If watchdog reset is not committed: • Execute cold reset. • Complete the cold reset assertion and de-assertion. If watchdog reset is committed, queue the cold reset and • Execute watchdog reset. • Then, complete the pending cold reset.
Watchdog reset	Warm reset	Continue watchdog reset.
Software initiated CPU warm reset	Warm reset	First, complete software initiated reset and then execute warm reset.
Software initiated POR reset / L2 reset	Warm reset	First, complete software initiated reset and then execute warm reset.
Software initiated CPU warm reset	Watchdog reset	Stop software initiated reset, and execute watchdog reset.
Software initiated POR reset	Watchdog reset	Stop software initiated reset, and execute watchdog reset.
Software initiated CPU warm reset	Cold reset	Stop software initiated reset, and execute cold reset.
Software initiated L2 reset	Cold reset	Stop software initiated reset, and execute cold reset.

The reset manager contains the `stat` register that indicates which reset source caused a reset. After a cold reset completes, the reset manager clears all bits 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.

After a warm reset is complete, the bit(s) that indicate the source of the warm reset are set to 1. A warm reset does not clear any bits in the `stat` register, therefore you may want clear them after determining the reset source. Any bit can be manually cleared by writing a 1 to it.

12.2. Modules Under Reset

This table depicts which modules undergo reset during different reset scenarios.

Table 91. Modules Under Reset

Modules/ Resources	POR	System Cold Reset	System Warm Reset	Watchdog Reset	MPU Cold Reset	CPU Cold Reset	CPU Warm Reset	Debug Reset
System Interconnect, CCU	X	X	X	X	-	-	-	-
Reset Manager, Clock Manager, System Manager	X	X(20)	X(20)	X(20)	-	-	-	-
Peripherals	X	X	X	X	-	-	-	-
L2/SCU	X	X	X	X	X	-	-	-
FPGA-to-SoC Bridge	X	X	X(21)	X	-	-	-	-
SoC-to-FPGA Bridge	X	X	X(21)	X	-	-	-	-
Lightweight SoC-to-FPGA Bridge	X	X	X(21)	X	-	-	-	-
MPU cores	X	X	X	X	X	X(22)	X(22)	-
MPU Debug	X	X	-	-	X	X(22)	-	X
Non-MPU Debug/Trace	X	X	-	-	-	-	-	X
JTAG TAP	X	-	-	-	-	-	-	-

Related Information

- Clock Manager Address Map and Register Definitions on page 160
- Reset Manager Address Map and Register Definitions on page 170

12.3. Reset Handshaking

The reset manager participates in several reset handshaking protocols to ensure that the interfaces to other modules are precisely shut-down before the reset is applied.

(20) Only clock and reset manager registers are reset. For more information about the specific register, refer to *Clock/Reset Manager Address Map and Register Definitions*.

(21) Only if enabled.

(22) Only the CPUs that are selected through the COLDMODRST/MPUMODRST register is reset.

- Handshake with Clock Manager:
  - After assertion of a cold or warm reset, the reset manager requests clock manager to put clocks in boot mode.
- Before the module reset signals are triggered by a warm reset, the reset manager performs handshakes with these modules to allow them to prepare for a warm reset. For example, debug AXI buses are made idle before asserting warm reset signal to non-debug logic. The handshake logic ensures the following conditions:
  - The embedded trace router (ETR) master has no pending master transactions to the L3 interconnect
  - Preserve SDRAM contents during warm reset by issuing self-refresh mode request
  - 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.

Handshake Scenarios

1. When the reset request is only for debug logic and not for system interconnect.
   - Reset manager asserts an idle request, indicating system interconnect to flush or complete all accesses, which is followed by a debug reset assertion.
2. When the reset request is only for system interconnect and not for debug logic.
   - Reset manager asserts an idle request, indicating system interconnect to flush or complete all accesses, which is followed by a system interconnect reset assertion.

This handshaking applies to all debug logic and debug peripherals residing outside the warm reset domain.

When performing debug domain reset, the reset manager performs other debug related handshakes (ETR) before resetting the debug domains.

12.4. 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.

In secure mode, the source of `boot_clk` clock is an internal oscillator, while in non-secure mode, the source is HPS_OSC_CLK clock.

Related Information

Clock Manager on page 149

12.4.1. SoC-to-FPGA Reset Sequence

During any reset condition that requires the SDM (for example: POR, system cold reset or mailbox message to SDM), the SDM holds the Reset Manager in reset until all reset requests to the SDM have been removed, or for a minimum of 128 boot clocks at 200 MHz. During this time, the Reset Manager asserts `s2f_cold_rst_n`, `s2f_rst_n`, and `s2f_watchdog_rst_n` signals. Thereafter, the Reset Manager releases signals according to the *Table: Reset Priority*.

During any reset condition that does not require the SDM (for example: system warm reset or watchdog reset), the Reset Manager asserts `s2f_rst_n` or `s2f_watchdog_rst_n` signal for a minimum of 128 boot clocks at 200 MHz. Thereafter, the Reset Manager releases signals according to the *Table: Reset Priority*.

Note: The Reset Manager does not release the MPU at the same time as releasing `s2f_cold_rst_n`, `s2f_rst_n`, or `s2f_watchdog_rst_n`. To release the MPU cores out of reset, the Reset Manager waits until the `ocramload.done` bit is set.

12.4.2. Warm Reset Sequence

1. You can assert warm reset request using the EL3 register via software. You must ensure that all CPUs enter WFI mode (for example, consider CPU0 is the master CPU):
   a. CPU3/2/1 interrupt routine:
      i. Pause all transaction prior to interrupt.
      ii. Idle the CPU3/2/1 with the WFI mode.
   b. CPU0 interrupt routine:
      i. Pause all transaction prior to interrupt.
      ii. Perform L2FLUSH.
      iii. Set the DDR self-refresh enable bit to preserve SDRAM contents during warm reset.
      iv. Write to EL3 register to reset.
      v. Idle the CPU0 with the WFI mode.
2. Reset Manager performs the following handshakes:
   a. HMC handshaking, if enabled using the `hdsken` register.
   b. FPGA handshaking, if enabled using the `hdsken` register.
   c. ETR handshaking, if enabled using the `hdsken` register.
3. Reset Manager initiates boot mode request handshake with Clock Manager.
4. Reset Manager waits for an acknowledgement signal from Clock Manager that indicates completion of the boot mode handshake before proceeding any further.
   - A cold or watchdog reset request that occurs before the completion of this step takes precedence over the warm reset sequence.
   - A cold or watchdog reset request that occurs after the completion of this step is delayed until the warm reset is completed.
5. Reset Manager asserts System Warm Reset. After a definite time-period, Reset Manager de-asserts all modules in reset except MPU.

6. Reset Manager waits until the `ocramload.done` bit is set.
7. Reset Manager de-asserts L2/SCU using the `coldmodrst.l2` register bit.
8. Reset Manager de-asserts MPU cores using the `mpumodrst.core[3:0]` and `coldmodrst.cpupor[3:0]` register bits.
9. You can de-assert peripheral modules using the `per0modrst` and `per1modrst` registers.

12.4.3. Watchdog Reset Sequence

A watchdog timeout event triggers this reset sequence. Reset Manager asserts watchdog reset based on the `watchdog timer` register.

1. Reset Manager performs L2FLUSH, if enabled.
   a. Use the `MPUL2FLUSH_timeout` register to timeout L2FLUSH event. The default value of the register is 0x00100000.
2. Reset Manager performs the following handshakes:
   a. HMC handshaking, if enabled using the `hdsken` register.
   b. FPGA handshaking, if enabled using the `hdsken` register.
   c. ETR handshaking, if enabled using the `hdsken` register.
3. Reset Manager initiates boot mode request handshake with Clock Manager.
4. Reset Manager waits for an acknowledgement signal from Clock Manager, that indicates completion of the boot mode handshake before proceeding any further.
   - A cold reset request that occurs before the completion of this step takes precedence over the watchdog reset sequence.
   - A cold reset request that occurs after the completion of this step is delayed until the watchdog reset is completed.
5. Reset Manager asserts Watchdog reset. After a definite time-period, Reset Manager de-asserts all modules in reset except MPU.
6. Reset Manager waits until the `ocramload.done` bit is set.
7. Reset Manager de-asserts L2/SCU using the `coldmodrst.l2` register bit.
8. Reset Manager de-asserts MPU cores using the `mpumodrst.core[3:0]` and `coldmodrst.cpupor[3:0]` register bits.
9. You can de-assert peripheral modules using the `per0modrst` and `per1modrst` registers.

12.5. Reset Signals and Registers

The reset manager uses the following module reset signals to assert reset for the respective modules during different reset domain . Most of these signals are driven internally, and you do not have any control over them. These signals are solely listed to explain the reset manager functionality.

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

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

Modules	Module Reset Signal	Register
FPGA fabric	s2f_rst_n	-
	s2f_cold_rst_n	-
	s2f_watchdog_rst_n	-
Debug domain with CoreSight and Trace	dbg_rst_n	dbgmodrst.dbg_rst dbgmodrst.csdap_rst
MPU	corereset_n [3:0]	mpumodrst.core[3:0]
	cpuporreset_n [3:0]	coldmodrst.cpupor[3:0]
	l2reset_n	coldmodrst.l2
DMA	dma_rst_n	per0modrst.dma
	dma_ecc_rst_n	per0modrst.dmaocp
	dma_periph_if_rst_n [7:0]	per0modrst.dmaif[7:0]
SPI Master and Slave	spim_rst_n [1:0]	per0modrst.spim[1:0]
	spis_rst_n [1:0]	per0modrst.spis[1:0]
Ethernet MAC	emac_rst_n [2:0]	per0modrst.emac[2:0]
	emac_ecc_rst_n [2:0]	per0modrst.emac[2:0]ocp
	emac_ptp_rst_n	per0modrst.emacptp
USB	usb_rst_n [1:0]	per0modrst.usb[1:0]
	usb_ecc_rst_n [1:0]	per0modrst.usb[1:0]ocp
NAND Flash	nand_flash_rst_n	per0modrst.nand
	nand_flash_ecc_rst_n	per0modrst.nandocp
SD/MMC	sdmmc_rst_n	per0modrst.sdmmc
	sdmmc_ecc_rst_n	per0modrst.sdmmcocp
Watchdog	watchdog_rst_n [3:0]	per1modrst.watchdog[3:0]
Timer	l4sys_timer_rst_n [1:0]	per1modrst.l4systimer[1:0]
	sp_timer_rst_n [1:0]	per1modrst.sptimer[1:0]
I^2C	i2c_rst_n [4:0]	per1modrst.i2c[4:0]
UART	uart_rst_n [1:0]	per1modrst.uart[1:0]
GPIO	gpio_rst_n [1:0]	per1modrst.gpio[1:0]
SoC-to-FPGA Bridge	soc2fpga_bridge_rst_n	brgmodrst.soc2fpga
FPGA-to-SoC Bridge	fpga2soc_bridge_rst_n	brgmodrst.fpga2soc
Lightweight SoC-to-FPGA Bridge	lwsoc2fpga_bridge_rst_n	brgmodrst.lwsoc2fpga
MPFE	mpfe_rst_n	brgmodrst.mpfe

12.6. Reset Manager Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

13. 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) system complex
- NAND flash controller
- Secure Digital/MultiMediaCard (SD/MMC) controller
- USB 2.0 On-The-Go (OTG) controllers (USB0 and USB1)
- Watchdog timers

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

13.1. 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:

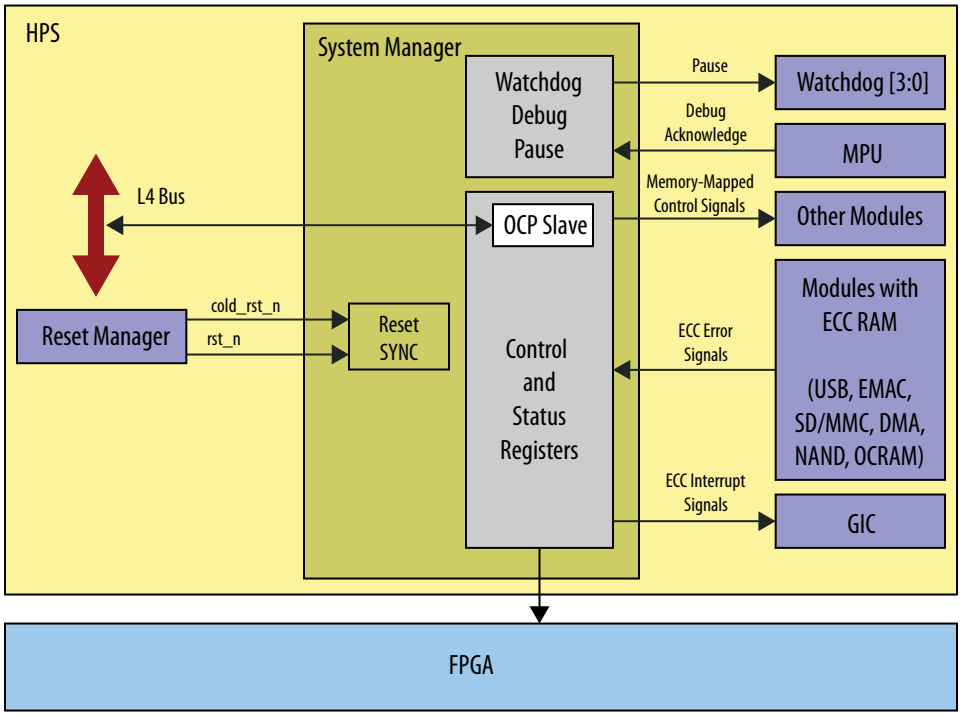
- Sends pause signals to pause the watchdog timers when the processors in the MPU system complex are in debug mode
- Selects the EMAC system interconnect master access options and other EMAC clock and interface options.
- Selects the SD/MMC controller clock options and system interconnect master access 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 the capability to enable or disable an interface to the FPGA.
- 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.

13.2. System Manager Block Diagram

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 33. 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
- Watchdog debug pause—accepts the debug mode status from the MPU system complex and pauses the L4 watchdog timers.
- Reset Manager— system manager receives the reset signals from reset manager.

13.3. Functional Description of the System Manager

The system manager serves the following purposes:

- 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
- The system manager provides ten 32-bit registers to store handoff information between the preloader and the operating system.

13.3.1. 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.

13.3.1.1. 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 ns bits of the dma_periph register determine if a peripheral request interface is secure or non-secure.

Note: The ns bits of the dma_periph register must be configured before the DMA is released from global reset.

Related Information

- DMA Controller on page 116
- System Manager Address Map and Register Definitions on page 177

13.3.1.2. 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 nand_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 185
- System Manager Address Map and Register Definitions on page 177

13.3.1.3. 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 program the system manager's emac* register to control the EMAC's ARCACHE and AWCACHE signals. 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 in the emac* registers 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.

Note: EMAC0 must be set to internal timestamp.

Related Information

- Clock Manager on page 149
- Ethernet Media Access Controller on page 308

13.3.1.4. 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 381

13.3.1.5. 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` and `sdmmc_smplsel` 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 221

13.3.1.6. 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 system complex is paused when its associated CPU enters debug mode.

Related Information

Watchdog Timers on page 478

13.3.2. FPGA Interface Enables

The system manager can enable or disable 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:

- Boundary scan interface
- Debug interface
- Trace interface
- System Trace Macrocell (STM) interface
- Cross-trigger interface (CTI)
- NAND interface
- SD/MMC interface
- SPI Master interface
- EMAC interfaces

13.3.3. 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
- SD/MMC controller RAM
- DDR interfaces

System manager provides combined ECC status and interrupt from each of these HPS modules. Each modules generates single or double bit error, which the system manager combines to generate interrupts.

Single bit ECC errors are maskable at system manager level by using the `ecc_intmask` register. Double bit errors are non-maskable.

ECC interrupt status is captured in the system manager register, which has the accessibility as **RO** (read only). Therefore, you need to clear the status at the actual source of the interrupt to release it.

13.3.4. 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.

13.3.5. Clocks

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

Related Information

Clock Manager on page 149

13.3.6. Resets

The system manager receives two reset signals from the reset manager. The `sys_config_rst_n` signal is driven on a cold or warm reset and the `sys_config_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 161

13.4. System Manager Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

14. Hard Processor System I/O Pin Multiplexing

The Intel Agilex 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. A pin multiplexing mechanism allows the SoC to use the flexible I/O pins in a wide range of configurations.

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

14.1. Features of the Intel Agilex HPS I/O Block

The I/O block provides the following functionality and features:

- Dedicated HPS I/O pins
  - 48 pins available for HPS clock, external flash memories and peripherals.

  Note: The HPS also interfaces with an SDRAM memory controller. This interface is separate from the dedicated pins discussed in this chapter.
- I/O multiplexing
  - Selects pins used by each HPS peripheral
  - 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.

  You configure I/O multiplexing when you instantiate the HPS component in Platform Designer.

Related Information

External Memory Interface Handbook
For details about memory I/O pins in the SoC hard memory controller, refer to the *Functional Description - HPS Memory Controller* chapter of the *External Memory Interface Handbook*.

14.2. Intel Agilex HPS I/O System Integration

The HPS I/O block consists of the following sub-blocks:

- Dedicated pin multiplexers (MUXes) – MUXes for the dedicated 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

Related Information

Intel Agilex I/O Control Registers on page 180

14.3. Functional Description of the HPS I/O

14.3.1. I/O Pins

The HPS has 48 dedicated I/O pins. They are divided into four quadrants of 12 signals per quadrant. When you instantiate the HPS component in Platform Designer, you must assign one of the 48 pins as the HPS clock. You can then use the remaining dedicated I/O pins for other common peripherals.

You can alternatively route most HPS peripherals (except USB) through the FPGA. Select this routing when you instantiate the HPS Component. For more information, refer to the *Intel Stratix® 10 HPS Component Reference Manual*.

Note: When assigning an HPS peripheral to HPS dedicated pins, you must assign all peripheral I/O pins to the same quadrant, except for NANDx16, Trace, and GPIO.

Note: Although the HPS dedicated I/O pins are configured through the control registers, software cannot reconfigure the pins after I/O configuration is complete. There is no support for dynamically changing the pin MUX selections for HPS dedicated I/O pins.

Related Information

- Booting and Configuration on page 505
  Details about the boot up process for the Intel Agilex HPS
- FPGA Access on page 179
  Information about routing HPS peripheral interfaces to the FPGA
- Configuring HPS I/O Multiplexing on page 183
  Information about configuring the HPS I/O MUXes

14.3.2. FPGA Access

Most HPS peripheral interfaces can be connected into the FPGA fabric, instead of to the dedicated 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 and GPIO controllers can interface to the FPGA fabric.

Related Information

Configuring HPS I/O Multiplexing on page 183
Information about configuring the HPS I/O MUXes

14.3.3. Intel Agilex I/O Control Registers

The HPS provides control registers that allow the system to initialize the following I/O parameters at system startup:

- Pin assignment for external oscillator clock input
- Pin assignment for each HPS peripheral
- HPS peripheral interfaces optionally exposed to FPGA logic
- I/O cell configuration

Note: Software can only access the HPS I/O control registers in secure mode.

Control registers can be divided into the following groups:

- Dedicated pin MUX registers
- Dedicated configuration registers
- FPGA access MUX registers
- HPS oscillator clock input register
- HPS JTAG pin MUX register

You program 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 boot loader code.

Related Information

Configuring HPS I/O Multiplexing on page 183
Information about configuring the HPS I/O MUXes

14.3.3.1. Intel Agilex Dedicated Pin MUX Registers

The HPS provides pin MUX registers, `pin0sel` through `pin47sel`, for each of the dedicated pins HPS_IOA_0 to HPS_IOA_23 and HPS_IOB_0 to HPS_IOB_23. Each pin MUX register contains a 4-bit MUX select field to select the function of the dedicated pin. A cold reset event sets these fields to 9 (reserved). Before a pin can connect to an HPS peripheral, the bootloader must reconfigure the MUX select field.

A warm reset event does not affect the dedicated pin MUX registers.

Platform Designer determines the values of the pin MUX registers automatically when you configure the HPS component.

Note: Although the HPS dedicated I/O pins are configured through the control registers, software cannot reconfigure the pins after I/O configuration is complete. There is no support for dynamically changing the pin MUX selections for HPS dedicated I/O pins.

Note: Platform Designer automatically places the values of the pin MUX registers in the Platform Designer handoff folder when you compile your design with the HPS component. The generator tool uses the handoff folder when generating the boot loader. The boot loader configures the pins during boot. You cannot select the oscillator clock input through the pin MUX (`pin*sel`) registers. To select the oscillator clock input you must program the HPS Oscillator Clock Input register (`hps_osc_clk`) and then set the corresponding `pin*sel` register to a value of 0x9.

Related Information

HPS Oscillator Clock Input Register on page 182

14.3.3.2. Intel Agilex Dedicated Configuration Registers

Configuration registers for each dedicated I/O pin allow software to control the corresponding I/O cell. These registers, `io0ctrl` through `io47ctrl`, allow software to set the following characteristics:

- Drive strength discrete values set to 2, 4, 6, or 8 mA
- Slow/Fast Slew rate control
- Internal weak pullup
- internal weak pulldown
- Open drain
- Schmitt trigger/TTL input

A warm reset event does not affect these registers.

In addition, registers `io0_delay` through `io47_delay` allow software to set the delay chains in each of the dedicated I/Os.

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

- Configuring HPS I/O Multiplexing on page 183
  Information about configuring the HPS I/O MUXes
- HPS Programmable I/O Timing Characteristics

14.3.3.3. FPGA Access MUX Registers

The FPGA access MUX registers (sometimes called "use FPGA" registers) select whether each HPS peripheral uses HPS I/O pins or is routed to the FPGA fabric. Platform Designer determines the values of the FPGA access MUX registers automatically when you configure the HPS component.

You can route most peripherals (except USB and GPIO) 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_sdmmc_usefpga`
- `pinmux_spim0_usefpga`
- `pinmux_spim1_usefpga`
- `pinmux_spis0_usefpga`
- `pinmux_spis1_usefpga`
- `pinmux_uart0_usefpga`
- `pinmux_uart1_usefpga`
- `pinmux_mdio0_usefpga`
- `pinmux_mdio1_usefpga`
- `pinmux_mdio2_usefpga`

At cold reset, the FPGA access registers default to 0, selecting the HPS I/O pins. A warm reset event does not affect these registers.

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

Configuring HPS I/O Multiplexing on page 183
Information about configuring the HPS I/O MUXes

14.3.3.4. HPS Oscillator Clock Input Register

Register `hps_osc_clk` selects the I/O for the external oscillator connection. The HPS routes this input clock from the oscillator to the HPS clock manager. Platform Designer determines the value of the HPS Oscillator Clock Input Register automatically when you configure the HPS component.

At cold reset, `hps_osc_clk` defaults to value 0x3F, and none of the I/Os are selected. A warm reset event does not affect this register.

Note: Although the HPS Oscillator Clock Input Register can be configured through the control registers, Intel recommends against reconfiguring this register after I/O configuration is complete.

When a pin is assigned as the oscillator clock input, it cannot support any other peripheral. For this reason, Platform Designer sets the pin MUX register to 8, indicating "not connected to any peripheral".

14.3.3.5. HPS JTAG Pin MUX Register

Register `pinmux_jtag_usefpga` selects whether HPS JTAG is accessed from the HPS pins or the FPGA interface. Platform Designer determines the values of the HPS JTAG pin MUX registers automatically when you configure the HPS component.

At cold reset, `pinmux_jtag_usefpga` defaults to 0 and selects the HPS JTAG access from HPS Pins. A warm reset event does not affect this register.

Note: Although the HPS JTAG Pin MUX Register is configured through the control registers, Intel recommends against reconfiguring this register after I/O configuration is complete.

14.3.4. Configuring HPS I/O Multiplexing

You can configure HPS I/O multiplexing when you generate the system in Platform Designer.

14.3.4.1. Configuring Intel Agilex I/O Multiplexing at System Generation

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 HPS I/O control registers.

Related Information

Intel Agilex I/O Control Registers on page 180

14.4. Intel Agilex Pin MUX Test Considerations

The HPS dedicated I/O pins are chained into the full chip JTAG boundary scan chain.

In some power modes the HPS can be off or disabled. However, the boundary scan chain still includes the HPS dedicated I/O pins, even when the HPS is inactive.

While the boundary scan is taking place, you must ensure that no software is executing in the HPS.

Note: You can only perform boundary scan with the FPGA JTAG. HPS JTAG does not support boundary scan.

14.5. Intel Agilex I/O Pin MUX Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

15. 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 Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

15.1. NAND Flash Controller Features

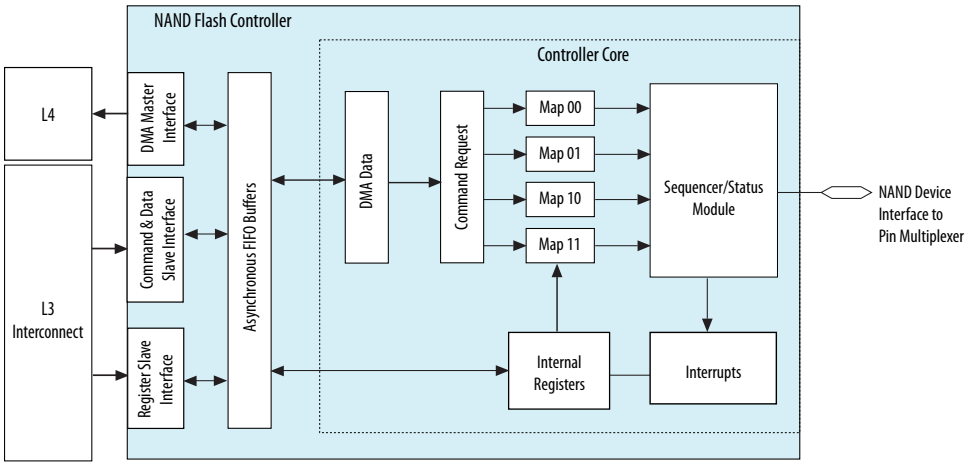
The NAND flash controller provides the following functionality and features:

- Supports Open NAND Flash Interface (ONFI) 1.0
- Provides support for 8- and 16-bit flash devices
- Provides support for up to four chip selects

  Note: Only the first chip select is exposed to the HPS I/O.
- 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 multi-plane devices
- Supports up to 50 MHz flash operating frequency
- Provides programmable access timing
- Supports page sizes of 512 bytes, 2 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)
- 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

15.2. NAND Flash Controller Block Diagram and System Integration

Figure 34. NAND Flash Controller Block Diagram



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-A53 MPCore processor generic interrupt controller (GIC)
- The DMA master interface provides accesses to and from the flash controller through the controller's built-in DMA.

15.2.1. Distributed Virtual Memory Support

The system memory management unit (SMMU) in the HPS supports distributed virtual memory transactions initiated by masters.

As part of the SMMU, a translation buffer unit (TBU) sits between the NAND Flash Controller and the L3 interconnect. The NAND shares this TBU with the USB, SD/MMC and ETR. An intermediate interconnect arbitrates accesses among the multiple masters before they are sent to the TBU. The TBU contains a micro translation lookaside buffer (TLB) that holds cached page table walk results from the translation control unit (TCU) in the SMMU. For every virtual memory transaction that this master initiates, the TBU compares the virtual address against the translations stored in its buffer to see if a physical translation exists. If a translation does not exist, the TCU performs a page table walk. The SMMU allows the NAND driver to pass virtual addresses directly to the NAND controller without having to perform virtual to physical address translations through the operating system.

For more information about distributed virtual memory support and the SMMU, refer to the "System Memory Management Unit" section.

Related Information

System Memory Management Unit on page 71

15.3. NAND Flash Controller Signal Descriptions

All NAND pins have to be from one of the following categories:

- HPS I/O
- FPGA I/O

The following table lists all NAND Flash pin options available to both the HPS and FPGA.

Table 92. NAND Flash Pin Options

Pins	Supported Data Width	Supported Number of CE and R/B
HPS Pins	8-bit or 16-bit	1
FPGA Pins	8-bit or 16-bit	1 – 4

If you are required to connect multiple NAND devices, you must route the NAND interface to FPGA logic. If you use HPS 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 pins, and route the CE and R/B signals to FPGA pins.

Table 93. NAND Flash Interface Signals

Platform Designer 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

Connected to FPGA	Connected to HPS I/O	HPS Pin Name
Yes	Yes	NAND_ADQ[15:0]
Yes	Yes	
Yes	Yes	
Yes	Yes	NAND_ALE

continued...

Connected to FPGA	Connected to HPS I/O	HPS Pin Name
Yes, 4 chip enables	Yes, 1 chip enable	NAND_CE_N
Yes	Yes	NAND_CLE
Yes	Yes	NAND_RE_N
Yes, 4 ready/busy signals	Yes, 1 ready/busy signal	NAND_RB
Yes	Yes	NAND_WE_N
Yes	Yes	NAND_WP_N

15.4. Functional Description of the NAND Flash Controller

This section describes the functionality of the NAND flash controller.

15.4.1. 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.

When the NAND controller is taken out of reset, the NAND flash controller samples the following configuration signals which are driven by user-writable registers in the System Manager:

Signal	Default Value	Description
bootstrap_inhibit_init	0	Inhibit the NAND controller from any initialization. The controller does not perform a query of the device and does not issue a `Page Load` command for block0, page0 for SLC devices. When this signal is asserted, it is expected that the software programs all registers pertaining to device parameters like: page size and width.
bootstrap_512B_device	0	Inform the NAND controller that 512-byte devices are connected.

continued...

Signal	Default Value	Description
bootstrap_512B_x16_device	0	Inform the NAND controller that 512-byte page size devices are connected and the device I/O width is 16 bits. This start should be asserted in case of 512-byte devices only.
bootstrap_two_row_addr_cycles	0	The connected device requires only two address cycles instead of the normal three row address cycles.
bootstrap_inhibit_b0p0_load	1	Bootstrap pin to inform the NAND controller not to issue a Page Load command for block0, page0, of the device as a part of the initialization procedure.

To support 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 7 on page 189.
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 5 on page 189.
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 read from that page. The processor can start reading from the first page of the flash memory.

   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.

15.4.2. 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 the controller is 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 171
For more information about the bootstrap interface control bits.

15.4.2.1. Bootstrap Setting Bits

Table 94. Bootstrap Setting Bits

Bit	Example Value for 512-Byte Page
noinit	1[23]
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 190

15.4.3. 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.

15.4.3.1. Recommended Bootstrap Settings for 512-Byte Page Device

Table 95. Recommended Bootstrap Settings for an 8-bit, 512-Byte Page Device

Register[24]	Value
devices_connected	1
device_width	0 indicating an 8-bit NAND flash device
	continued...

(23) 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".

(24) All registers are in the `config` group.

Register[24]	Value
number_of_planes	1 indicating a single-plane device
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.

15.4.4. Local Memory Buffer

The NAND flash controller has three FIFO memories implemented using dual-ported SRAM.

- Write FIFO—The read data from the host memory resides in the Write FIFO before being flushed to the memory.
- Read FIFO—The data from the device is read and stored in the FIFO before being forwarded to the host memory.
- ECC FIFO—This buffer holds data for applying the ECC correction while the logic computes error locations and mask.

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.

For more information about ECC, refer to the Error Checking and Correction Controller chapter.

Related Information

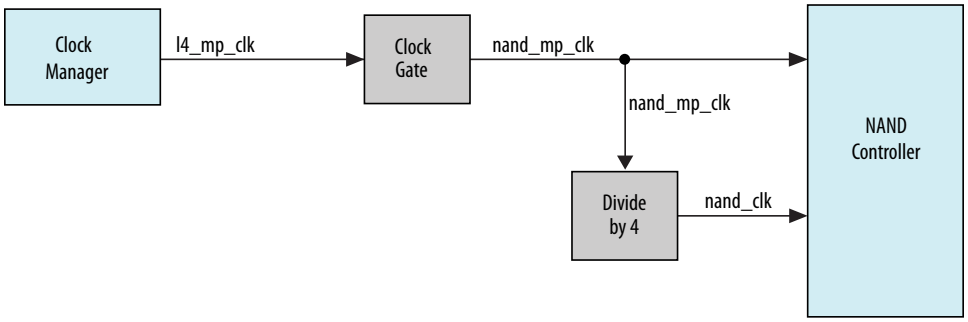
Error Checking and Correction Controller on page 128

15.4.5. Clocks

The software enable for NAND is `nand_clk_en` and is set to ENABLE by default. Also, during the automatic initialization performed after getting out of reset, `nand_clk_en` is active to ensure that all clocks are active if RAM is cleared for security.

(24) All registers are in the `config` group.

Figure 35. NAND Clocking Diagram



Note: When routing the NAND interface to the FPGA it may be necessary to increase the value of `max_rd_delay` to compensate for the additional delay between the controller and the FPGA I/O.

Related Information

Clock Manager on page 149

15.4.5.1. 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 generated clocks are 50 MHz and called `nand_clk`.

15.4.5.2. Clock Enable

The `nand_mp_clk` and `nand_clk` clocks have enables.

15.4.5.3. 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.

15.4.6. Resets

The NAND flash controller has one external reset signal, `nand_flash_rst_n`, that resets it. Once a reset is initiated, access to the NAND flash controller should not be attempted until after 20 `nand_clk` cycles.

Note: The minimum reset time for the NAND flash controller is 10 `nand_clk` clock cycles.

Related Information

Reset Manager on page 161

15.4.6.1. 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.

After the Cortex-A53 MPCore 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 section: *Reset Signals and Registers* in the *Reset Manager* chapter.

You should ensure that both the NAND ECC RAM and the NAND Module resets are deasserted before beginning transactions. Program the `nandocp` bits and the `nand` bits in the `per0modrst` register of the Reset Manager to deassert reset in the NAND ECC RAM and the NAND module, respectively.

15.4.7. 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.

15.4.7.1. 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 97. 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 or 16-bit wide data path.

Related Information

Command Mapping on page 194

15.4.7.2. 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.

15.4.8. 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—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

15.4.8.1. 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.

15.4.8.1.1. MAP00 Command Format

Table 98. 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)	0
27:26	CMD_MAP	0
25:13	(reserved)	0
12:2	BUFF_ADDR	Data width-aligned buffer address on the memory device. Maximum page access is 8 KB.
1:0	(reserved)	0

15.4.8.1.2. 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) on the device; reading directly from the page buffer while executing 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 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.

15.4.8.2. 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.

15.4.8.2.1. MAP01 Command Format

Table 99. 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)	0
27:26	CMD_MAP	1
25:24	(reserved)	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

15.4.8.2.2. 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.

15.4.8.3. 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.

15.4.8.3.1. MAP10 Command Format

Table 100. 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)	0
27:26	CMD_MAP	2
25:24	(reserved)	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		

15.4.8.3.2. MAP10 Operations

Table 101. 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

15.4.8.3.3. 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.

15.4.8.4. 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.

15.4.8.4.1. MAP11 Control Format

Table 102. 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)	0
27:26	`CMD_MAP`	3
25:2	(reserved)	0
1:0	`TYPE`	Sets the control type as follows:

continued...

Address Bits	Name	Description
		• 0 = Command cycle • 1 = Address cycle • 2 = Data Read/Write Cycle

15.4.8.4.2. 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.

15.4.9. 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 multi-transaction DMA command, and the burst DMA command.

15.4.9.1. Multi-Transaction DMA Command

The NAND flash controller processes multi-transaction 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 multi-transaction 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 201

15.4.9.1.1. Command-Data Pair Formats

Table 103. Command-Data Pair 1

	31:28	27:26	25:24	23:<*M*> [25]	(<*M*> – 1):0
(<*M*> – 1):0	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 104. 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 105. Command-Data Pair 3

	31:28	27:26	25:24	23:8	7:0
Command	0x0	0x2	0x0	Memory address low[26]	0x0

	31:16	15:12	11:8	7:0
Data	0x0	0x2	0x3	0x0

Table 106. Command-Data Pair 4

	31:28	27:26	25:24	23:17	16	15:8	7:0
Command	0x0	0x2	0x0	0x0	INT	Burst length	0x0
Note: INT specifies the host interrupt that is generated at the end of the complete DMA transfer; and 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_comp` bit is set to 1.							

(25) <*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.

(26) The buffer address in host memory, which must be aligned to a 4-byte boundary.

	31:16	15:12	11:8	7:0
Data	0x0	0x2	0x4	0x0

Related Information

- Indexed Addressing on page 193
- Burst DMA Command on page 202

15.4.9.1.2. 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 multi-transaction 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 multi-transaction 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 193
- Burst DMA Command on page 202
- System Manager on page 171

15.4.9.2. 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 multi-transaction format (described in the "Multi-Transaction DMA Command on page 200").

Table 107. 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[27]	Memory address high															
Beat 2[27]	Memory address low															
Beat 3	0x0							INT	Burst length							

Note: INT specifies the host interrupt to be generated at the end of the complete DMA transfer; and 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 200
- MAP10 Command Format on page 197
- System Manager on page 171

15.4.10. 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.

(27) The buffer address in memory, which must be aligned to a 4 byte-boundary.

15.4.10.1. Correction Capability, Sector Size, and Check Bit Size

Table 108. 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

15.4.10.2. 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

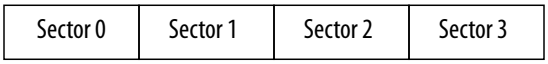
- Main Area Transfer Mode on page 204
- Spare Area Transfer Mode on page 204
- Main+Spare Area Transfer Mode on page 204

15.4.10.3. 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.

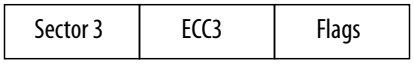
Figure 36. Main Area Transfer Mode for ECC



15.4.10.4. 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 37. Spare Area Transfer Mode for ECC

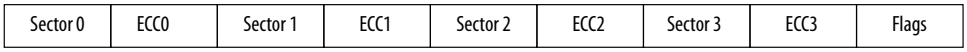


15.4.10.5. Main+Spare Area Transfer Mode

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 38. Main+Spare Area Transfer Mode for ECC



15.4.10.6. 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.

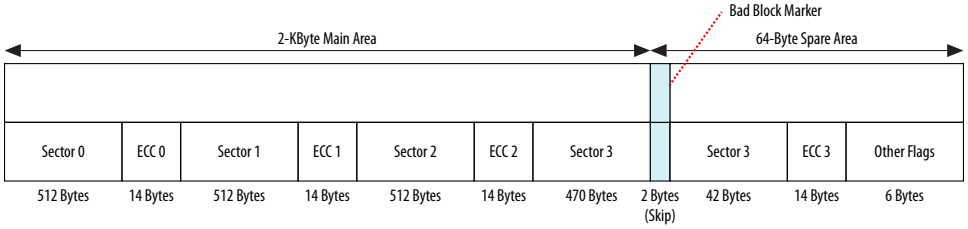
In main area transfer mode, the NAND flash controller does not skip the bad block marker. Instead, it overrides the bad block marker with the value programmed in the `spare_area_marker` register in the `config` group. This 8-bit register is used in conjunction with the `spare_area_skip_bytes` register 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 39. 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 213
For detailed information about configuring the NAND flash controller for default, spare, or main+spare area transfer mode.

15.4.10.7. 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.

15.5. 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.

15.5.1. Basic Flash Programming

This section describes the steps that must be taken by the software to access and control the NAND flash controller.

15.5.1.1. 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.

15.5.1.2. 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 188

15.5.1.3. 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.

15.5.1.4. ECC Enabling

Before you start any data operation on the flash device, you must decide whether you want 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 ECC.

Set up the appropriate correction level depending on the page size and the spare area available on the device by writing to the `ecc_correction` register in the `config` group.

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 203

15.5.1.5. 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.

15.5.1.6. 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.

15.5.1.6.1. 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.(28)
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:

(28) 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)

15.5.1.7. 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

15.5.1.8. 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.

15.5.2. 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 194

15.5.2.1. 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 197

15.5.2.1.1. 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.

15.5.2.1.2. 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.

15.5.2.2. 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.

15.5.2.2.1. 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.

15.5.2.2.2. 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.

15.5.2.2.3. 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.

15.5.2.3. 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.

15.5.2.3.1. 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 109. transfer_spare_reg and MAP10 Transfer Mode Commands

transfer_spare_reg	MAP10 Transfer Mode Commands	Resulting NAND Flash Controller Mode
0	0x42	Main[29]
0	0x41	Spare
0	0x43	Main+spare
1	0x42	Main+spare[29]
1	0x41	Spare
1	0x43	Main+spare

Related Information

MAP10 Commands on page 196

15.5.2.3.2. 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.

15.5.2.3.3. 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:

(29) 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.

15.5.2.3.4. 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.

15.5.2.4. 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.

15.5.2.4.1. 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.

15.5.2.5. 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.

15.5.2.5.1. 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.

15.5.2.5.2. 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.

15.5.2.6. 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.

15.5.2.6.1. 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.

15.5.2.6.2. 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.

15.5.2.6.3. 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.

15.5.2.6.4. 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.

15.5.2.6.5. 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.

15.6. NAND Flash Controller Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

16. 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 flash controller enables you to 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 SynopsysDesignWare Mobile Storage Host (SD/MMC controller) controller.[30]

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 Agilex Hard Processor System Technical Reference Manual Revision History on page 12
  For details on the document revision history of this chapter

16.1. Features of the SD/MMC Controller

The HPS SD/MMC controller offers the following features:

- Supports card detection and initialization
- Supports up to 50 MHz card operating frequency
- Programmable block size up to 64 KB
- Supports SD/SDIO/MMC versions 4.3 to 4.5 devices

(30) 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.

- Supports the following standards or card types:
  - SD memory (SD mem), including eSD support—versions 3.01
  - SDIO, including embedded SDIO (eSDIO) support—version 3.0
  - CE-ATA—version 1.1
- Supports various types of multimedia cards (MMC version 4.41 and eMMC versions 4.51 and 5.0)
  - 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): 1-bit, 4-bit, and 8-bit data bus
- Supports CE-ATA digital protocol commands
- Supports only Single Card
  - SDR mode only
  - Programmable card width: x1, x4, or x8
  - Programmable card type: SD, SDIO, or MMC
- Integrated descriptor-based direct memory access (DMA)

Related Information

MMC Support Matrix on page 223
For more information on what is supported, refer to the MMC Support Matrix table.

16.1.1. Device Support

The following devices have been tested and are compatible with the HPS.

Table 110. Devices Compatible with Each Device Type

Device Types	Devices
SD Mem	• SanDisk 64 MB SD, 256 MB SD, 256 MB MiniSD • Panasonic 128 MB SD • PNY 256 MB SD • Memorex 32MB SD • SimpleTech 64 MB SD
SDIO[32]	• PALM Bluetooth • Toshiba Bluetooth
CE-ATA	Hitachi Microdrive 3K8 hard drive
MMC	• San Disk 64 MB MMC • SimpleTech 128 MB MMC • Lexar 32 MB MMC

[32] Verified only Basic CMD5, CMD52, and CMD53 I/O commands.

16.1.2. SD Card Support Matrix

Table 111. SD Card Support Matrix

Device Card Type	Bus Modes Supported			Bus Speed Modes Supported			
				Default Speed	High Speed	SDR12	SDR25[33]
	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: The Intel Agilex HPS supports only 1.8V signaling. For flash devices that do not use 1.8V signaling, you must use an external level shifter.

Note: Although, all device types, except SDSC (SD), support the following bus speed modes, the HPS does not support these modes:

- SDR50—50 MBps (100 MHz)
- SDR104—104 MBps (208 MHz)
- DDR50—50 MBps (50 MHz)

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.

16.1.3. MMC Support Matrix

Table 112. MMC Support Matrix

Card Device Type	Max Clock Speed (MHz)	Max Data Rate (MBps)	Bus Modes Supported			Bus Speed Modes Supported	
			1 bit	4 bit	8 bit	Default Speed	High Speed
MMC	20	2.5	√			√	
RSMMC	20	10	√	√		√	√
MMCPlus	50	25	√	√		√	√
		50			√	√	√
MMCMobile	50	6.5	√			√	√
eMMC	50	25	√	√		√	√
		50			√	√	√

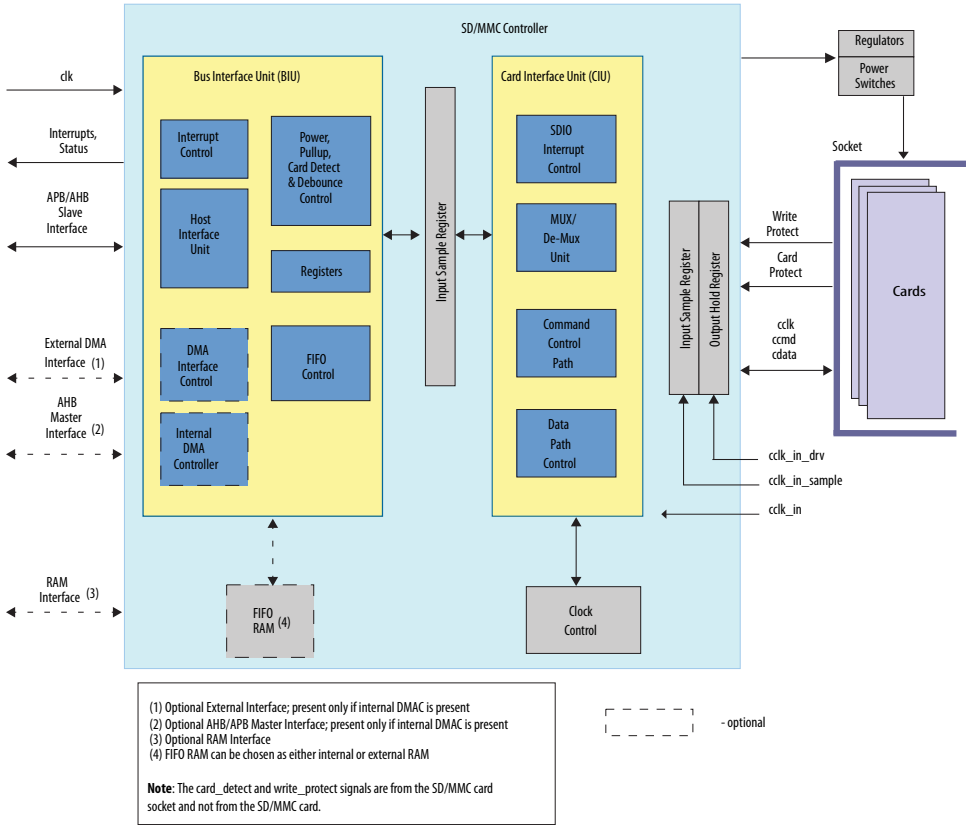
(33) SDR25 speed mode requires 1.8-V signaling. Note that even if a card supports UHS-I modes (for example SDR50, SDR104, DDR50) it can still communicate at the lower speeds (for example SDR12, SDR25).

Note: The Intel Agilex HPS supports only 1.8V signaling. For flash devices that do not use 1.8V signaling, you must use an external level shifter.

Note: Although all device types, except MMC and RSMMC, support the DDR bus speed mode; the HPS does not support this mode.

16.2. SD/MMC Controller Block Diagram

Figure 40. SD/MMC Controller Connectivity



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 MPU system complex.

16.2.1. Distributed Virtual Memory Support

The system memory management unit (SMMU) in the HPS supports distributed virtual memory transactions initiated by masters.

As part of the SMMU, a translation buffer unit (TBU) sits between the SD/MMC and the L3 interconnect. The SD/MMC shares a TBU with the USB, NAND, and ETR. An intermediate interconnect arbitrates accesses among the multiple masters before they are sent to the TBU. The TBU contains a micro translation lookaside buffer (TLB) that holds cached page table walk results from a translation control unit (TCU) in the SMMU. For every virtual memory transaction that this master initiates, the TBU compares the virtual address against the translations stored in its buffer to see if a physical translation exists. If a translation does not exist, the TCU performs a page table walk. The SMMU allows the SD/MMC driver to pass virtual addresses directly to the SD/MMC without having to perform virtual to physical address translations through the operating system.

For more information about distributed virtual memory support and the SMMU, refer to the *System Memory Management Unit* chapter.

Related Information

System Memory Management Unit on page 71

16.3. SD/MMC Controller Signal Description

The following table shows the SD/MMC controller signals that are connected to the FPGA and the HPS I/O.

Table 113. SD/MMC Controller Interface I/O Pins

Signal	Connected to FPGA	Connected to HPS I/O
sdmmc_cclk_out	Yes	Yes
sdmmc_cmd_i	Yes	Yes
sdmmc_cmd_o	Yes	Yes
sdmmc_cmd_en	Yes	Yes
sdmmc_data_i [7 :0]	Yes	Yes
sdmmc_data_o [7 :0]	Yes	Yes
sdmmc_data_en [7 :0]	Yes	Yes
sdmmc_pwr_ena_o	Yes	Yes
sdmmc_cdn_i	Yes	No
sdmmc_wp_i	Yes	No
sdmmc_vs_o	Yes	No
sdmmc_rstn_o	Yes	No
sdmmc_card_intn_i	Yes	No
sdmmc_intr	Yes	No

Note: All signals must be routed to only the HPS I/O or only the FPGA I/O.

16.4. Functional Description of the SD/MMC Controller

This section describes the SD/MMC controller components and how the controller operates.

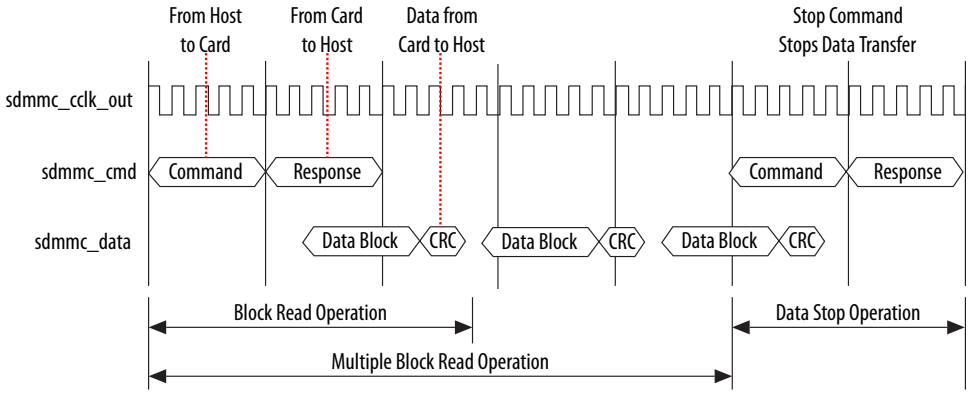
16.4.1. SD/MMC/CE-ATA Protocol

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.†

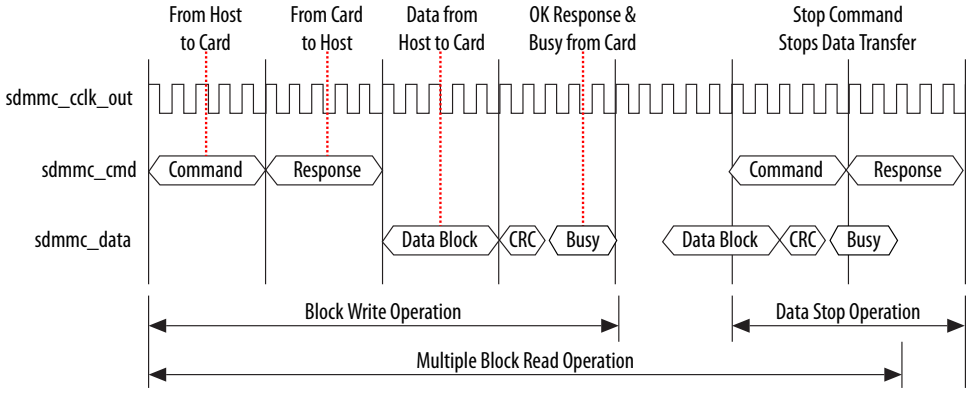
In the following figure, the clock is a representative only and does not show the exact number of clock cycles.

Figure 41. 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 42. Multiple-–Block Write Operation†



16.4.2. 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

16.4.2.1. Slave Interface

The host processor accesses the SD/MMC controller registers and data FIFO buffers through the slave interface.

16.4.2.2. 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.

16.4.2.2.1. 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.†

16.4.2.3. 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.

For more information about ECC, refer to the Error Checking and Correction Controller chapter.

Related Information

- System Manager on page 171
- Error Checking and Correction Controller on page 128

16.4.2.4. 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 115. 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 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. †

continued...

Bits	Interrupt	Description
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†`

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.†

continued...

Bits	Interrupt	Description
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.†

16.4.2.4.1. 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.†

16.4.2.5. 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.†

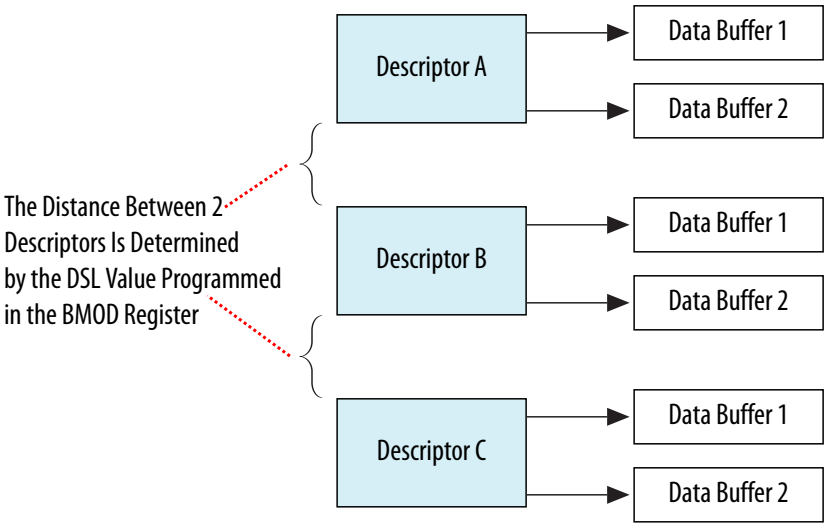
A single descriptor is used for both reception and transmission. The base address of the list is written into the descriptor list base address register (`dbaddr`). A descriptor list is forward linked. The last descriptor can point back to the first entry to create a ring structure. The descriptor list resides in the physical memory address space of the host. Each descriptor can point to a maximum of two data buffers.†

16.4.2.5.1. Internal DMA Controller Descriptors

The internal DMA controller uses these types of descriptor structures:†

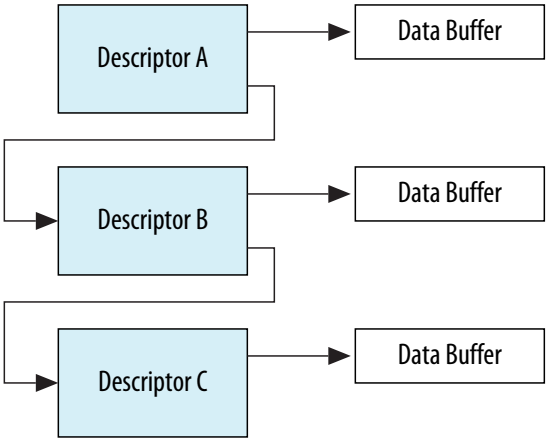
- Dual-buffer structure—The distance between two descriptors is determined by the skip length value written to the descriptor skip length field (`dsl`) of the bus mode register (`bmod`).†

Figure 43. Dual-Buffer Descriptor Structure†



- Chain structure—Each descriptor points to a unique buffer, and to the next descriptor in a linked list.†

Figure 44. Chain Descriptor Structure†



16.4.2.5.2. 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 116. 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 233
Refer to this table for information about each of the bits of the descriptor.

16.4.2.5.3. Internal DMA Controller Descriptor Fields

The DES0 field in the internal DMA controller descriptor contains control and status information.

Table 117. 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.

continued...

Bits	Name	Description
		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.
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 118. 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 119. 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 120. 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.

16.4.2.5.4. 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.†

16.4.2.5.5. Host Data Buffer Alignment

The transmit and receive data buffers in system memory must be aligned to a 32-bit boundary.

16.4.2.5.6. 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.

16.4.2.5.7. 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.†

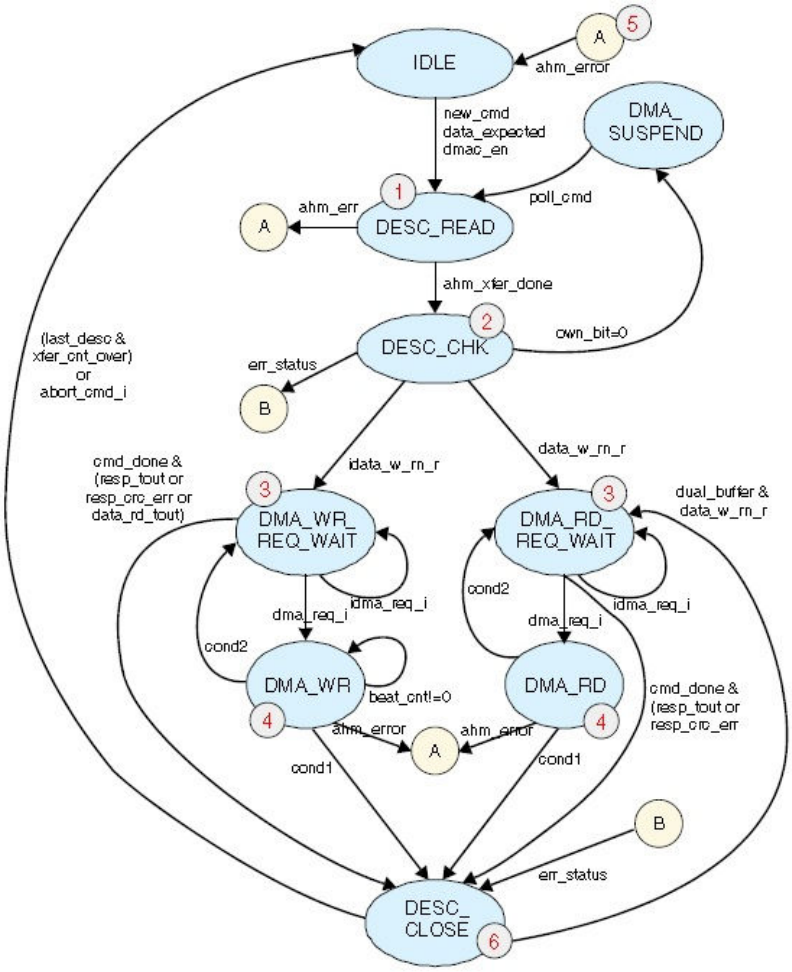
There are two summary interrupts—the normal interrupt summary bit (`nis`) and the abnormal interrupt summary bit (`ais`)—in the `idsts` register.† The `nis` bit results from a logical OR of the transmit interrupt (`ti`) and receive interrupt (`ri`) bits in the `idsts` register. The `ais` bit is a logical 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.

16.4.2.5.8. 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.†

16.4.2.6. 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.†

16.4.2.7. 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.†

16.4.2.7.1. 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.†

16.4.2.7.2. PBL and Watermark Levels

This table shows legal PBL and FIFO buffer watermark values for internal DMA controller data transfer operations.†

Table 121. 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

16.4.3. 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†

16.4.3.1. 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.†

16.4.3.1.1. 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.†

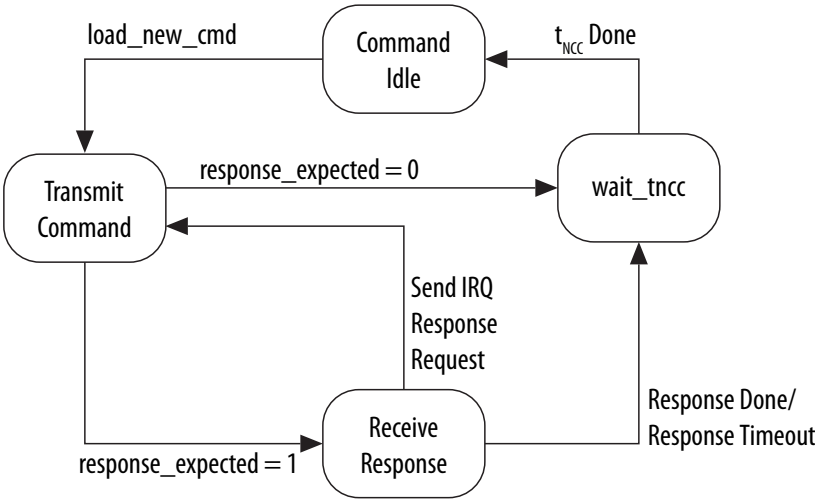
Loading a new command from the BIU in the command path depends on the following `cmd` register bit settings:†

- `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.†

16.4.3.1.2. 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 45. 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.†

16.4.3.1.3. 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 122. 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.

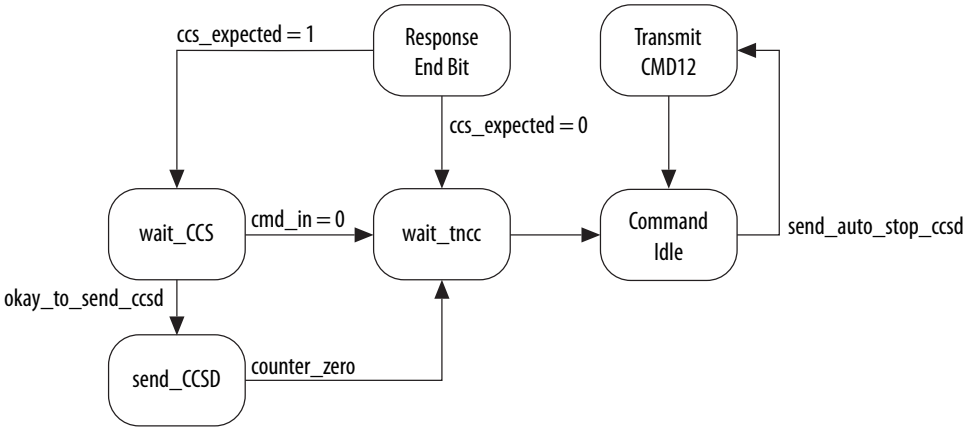
16.4.3.1.4. Driving P-bit to the CMD Pin

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.†

16.4.3.1.5. 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 46. 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.†

16.4.3.1.6. 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.†

16.4.3.1.7. 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.†

16.4.3.1.8. 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.†

16.4.3.1.9. 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.†

16.4.3.2. 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

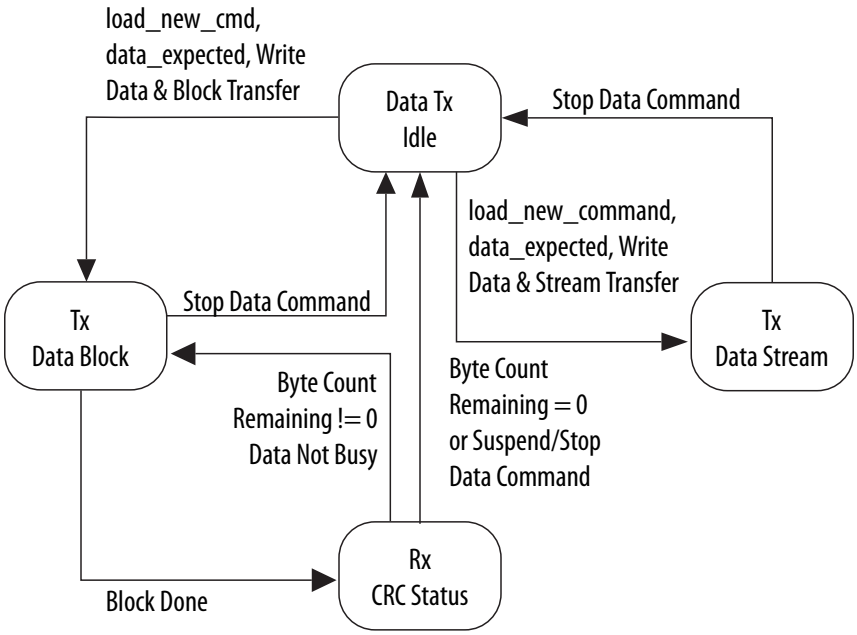
registers—whenever a data transfer command is not in progress. If the data transfer expected bit (`data_expected`) in the `cmd` register is set to 1, the new command is a data transfer command and the data path starts one of the following actions:†

- Transmits data if the read/write bit = 1†
- Receives data if read/write bit = 0†

16.4.3.2.1. 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 47. 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.†

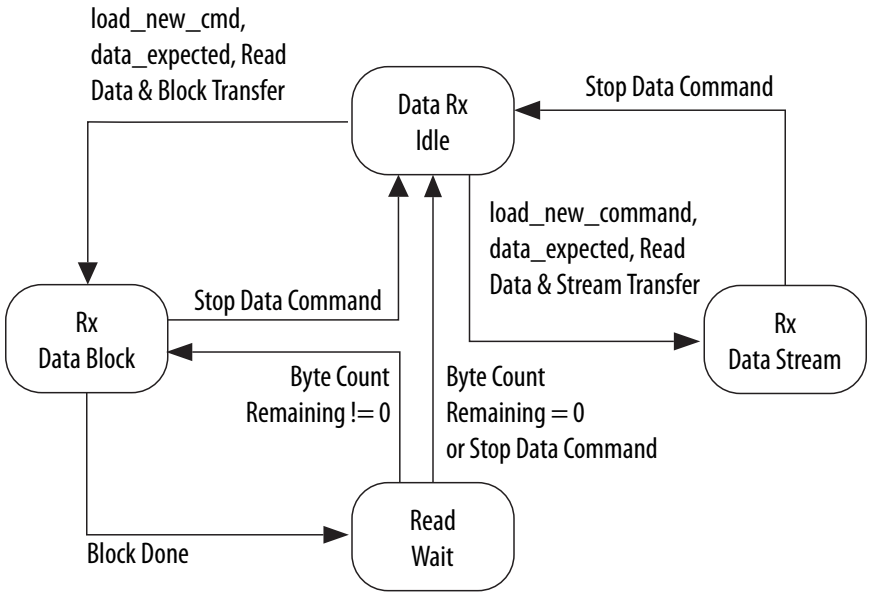
If the bytcnt is zero (the block size must be greater than zero) the transfer is an open-ended block transfer. The data transmit state machine for this type of data transfer continues the block-write data transfer until the host software issues an SD/SDIO STOP or STOP_TRANSMISSION (CMD12) command.†

16.4.3.2.2. 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 48. 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.†

16.4.3.2.3. 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 123. 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(34)†	Pre-defined multiple block
Multiple-block write	0	No	Open-ended multiple block
Multiple-block write	>0	Yes (34)†	Pre-defined multiple block

Auto-Stop Generation for SD Cards

Table 124. 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 125. 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...

(34) 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

16.4.3.2.4. 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 126. 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 127. 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 128. 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(35)	64	4	8

Table 129. 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(36)	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.

16.4.3.3. 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.†

(35) Num_bytes = Number of bytes specified as per the lock card data structure. Refer to the SD specification and the MMC specification.†

(36) 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.†

16.4.3.4. Error Detection

Errors can occur during card operations within the CIU in the following situations.

16.4.3.4.1. 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.†

16.4.3.4.2. 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.†

16.4.3.4.3. 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.†

16.4.4. Clocks

Clocking Architecture

The clocking architecture is composed of:

Table 130. Clocking Architecture

Clock Name	Source	IP Clock Name	Range	Description
`l4_mp_clk`	Clock Manager	`clk`	200 MHz	System, host, AHB clock
`sdmmc_clk`	Wrapper Generated	`cclk_in`	50 MHz	Card interface unit (CIU) clock
		`cclk_in_drv` (phase shifted `cclk_in`)		Phase-shifted/delayed version of `cclk_in` on which output-related registers work.
		`cclk_in_sample` (phase shifted `cclk_in`		Phase-shifted/delayed version of `cclk_in` used for sampling the data from the card.
—	Synopsys IP Generated	`cclk_out`	Synopsys IP Generated	Card clocks. Output from internal clock dividers.

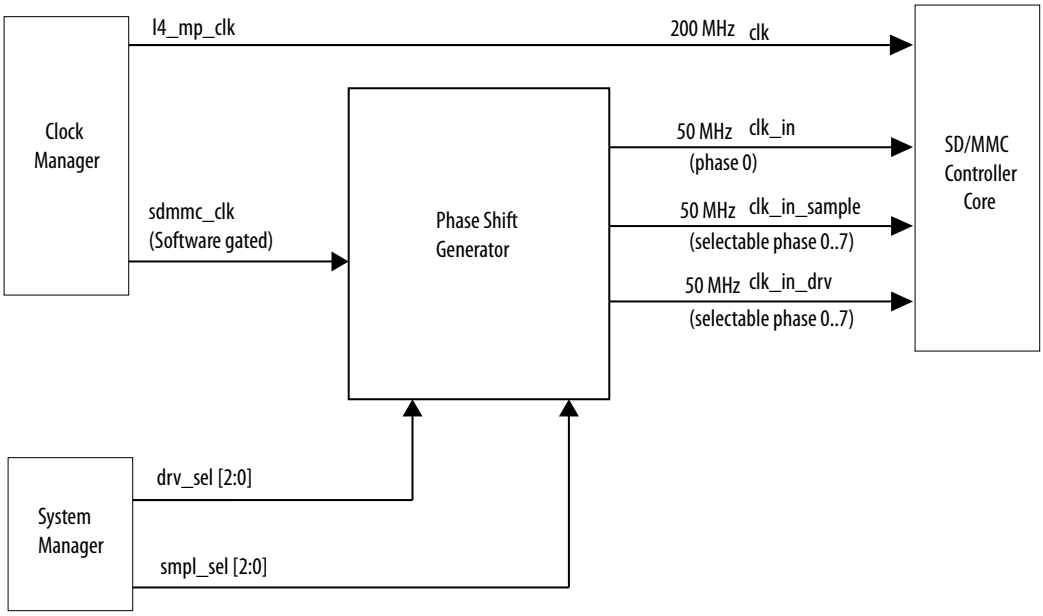
Clock Generation

The phase shift block is required for the following actions:

- To divide the 200 MHz `sdmmc_clk` input by 4 to generate a 50 MHz clock
- To generate 0, 45, 90, 135, 180, 225, 270 and 315 degree phase shifts of a 50 MHz clock

The System Manager provides software controlled selects, `drv_sel[2:0]` and `smpl_sel[2:0]`, to control the phase shifts for the `cclk_in_drv` and `cclk_in_sample`, respectively.

Figure 49. SD/MMC Controller Clock Connections - HPS



Related Information

Clock Control Block on page 253
Refer to this section for information about the generation of the `sdmmc_cclk_outclock`.

16.4.5. 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.

The single reset signal, **reset_n**, has the following attributes:

- Active-low
- Asynchronously asserted and synchronously deasserted to `clk (l4_mp_clk)`
- Kept active for at least two clocks of `clk` or `clk_in` (whichever has lower frequency)
- The Phase Shift Logic would also require an appropriate reset synchronized to its clock
- The resets to each of the two ports are synchronous to different clocks (`clk` and `cclk_in`)

Table 132. Reset Signal Definition

System Reset	Source	Reset	Assertion	Deassertion	Description
sdmmc_rst_n	External	reset_n	Asynchronous	Synchronous—clk (l4_mp_clk)	System active-low reset pin. Synchronous to clk.

Related Information

Reset Manager on page 161

16.4.5.1. 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.

After the Cortex-A53 MPCore 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 section: *Reset Signals and Registers* in the *Reset Manager* chapter.

You should ensure that both the SD/MMC ECC RAM and the SD/MMC Module resets are deasserted before beginning transactions. Program the sdmmcocp bits and the sdmmc bits in the per0modrst register of the Reset Manager to deassert reset in the SD/MMC ECC RAM and the SD/MMC module, respectively.

16.4.6. Voltage Switching

This section describes the general steps to switch voltage level.

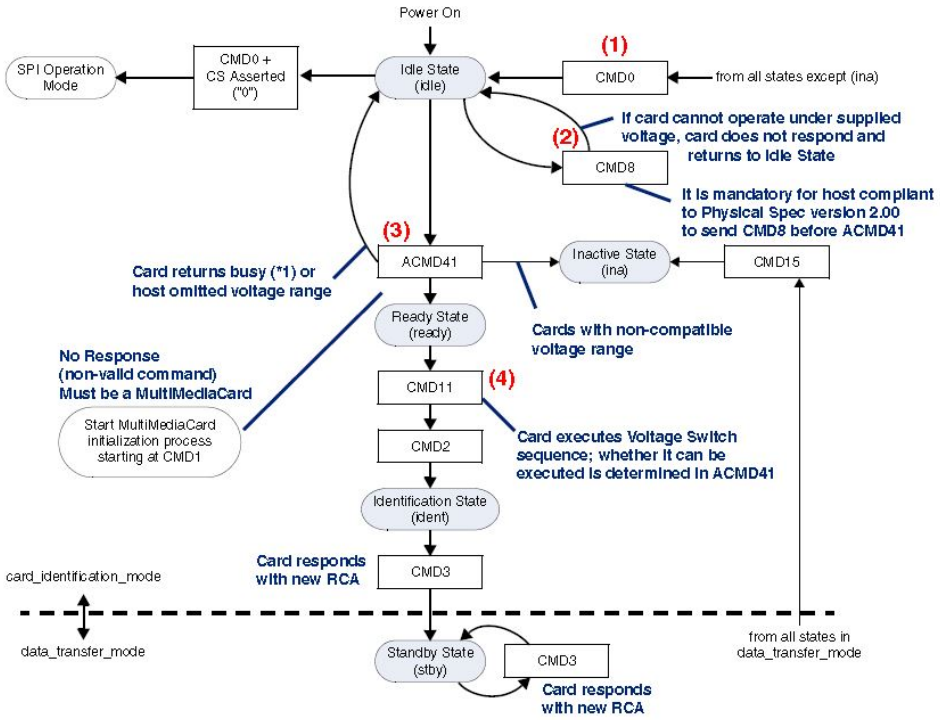
The SD/MMC cards support various operating voltages, for example 1.8V and 3.3V. If you have a card which is at 1.8V and you eject it and replace it with another card, which is 3.3V, then voltage switching is required.

In order to have the right voltage level to power the card, separate devices on the board are required: voltage translation transceiver and power regulator/supply. When the software is aware that voltage switching is needed, it notifies the power regulator that it needs to supply another voltage level to the card (switching between 1.8V and 3.3V).

Many SD cards have an option to signal at 1.8 or 3.3 V, however the initial power-up voltage requirement is 3.3V. 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 52. Voltage Switching Command Flow Diagram†



(*1) **Note:** Card returns busy when:†

- Card executes internal initializaiton process†
- Card is High or Extended capacity SD Memory Card and host does not support High†

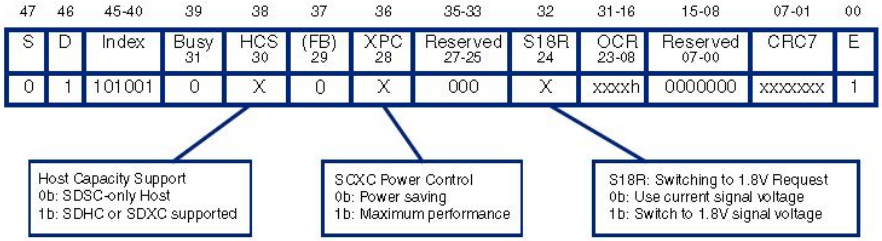
The following outlines the steps for the voltage switch programming sequence.†

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 53. ACMD41 Argument

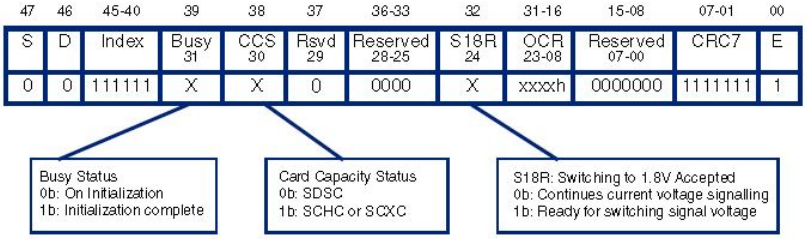


a. Bit 30 informs the card if host supports SDHC/SDXC or not; this bit should be set to 1'b1.†

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 54. 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

16.5. SD/MMC Controller Programming Model

16.5.1. 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 SynopsysDesignWare Cores Mobile Storage Host Databook.

16.5.1.1. 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, Clock Source registers, or both and send a command to the CIU in order 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.†

16.5.1.2. 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.

16.5.1.3. 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.†

16.5.1.4. Data-Transfer Requirement Between the FIFO and Host†

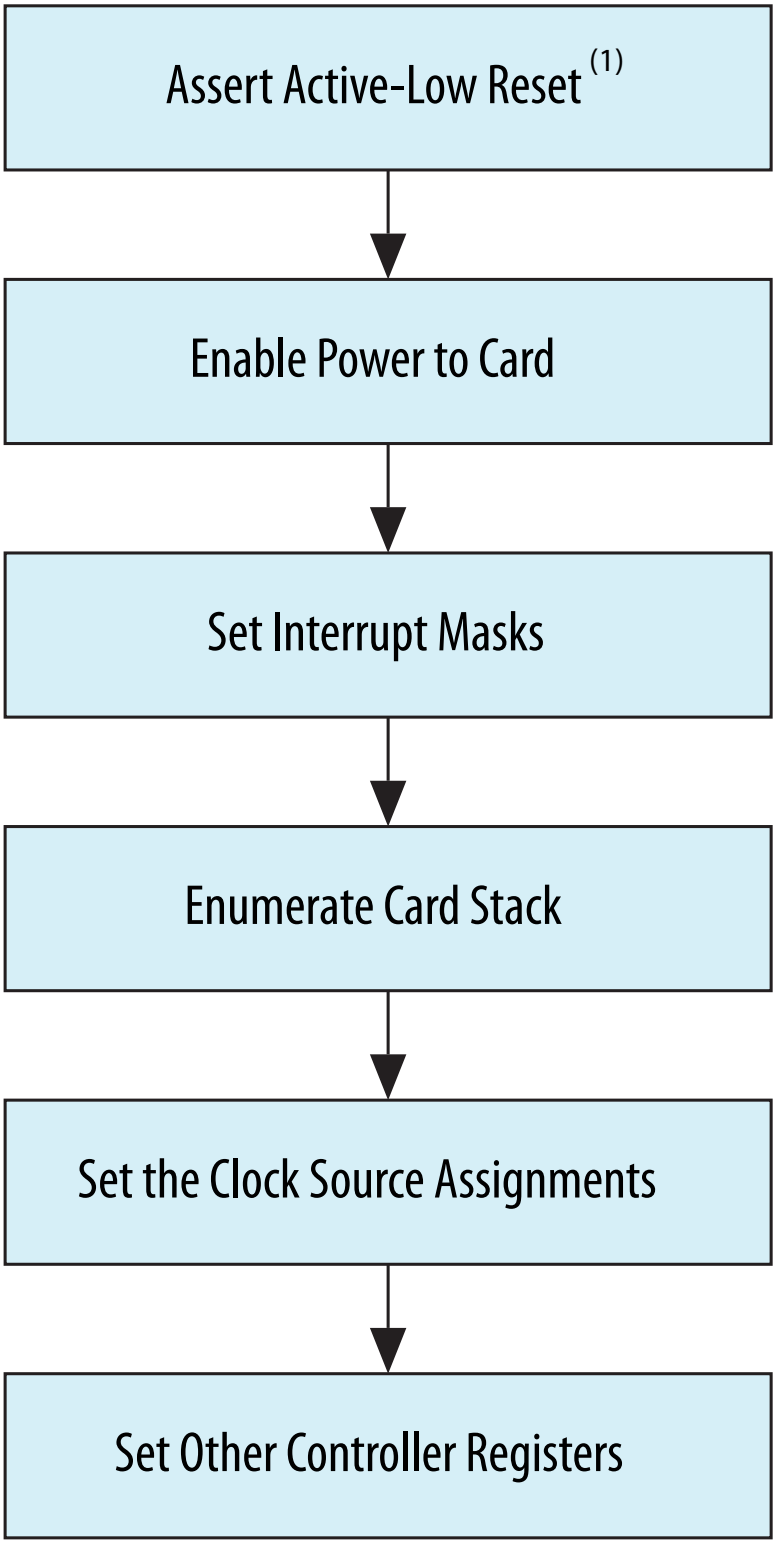
One data-transfer requirement between the FIFO and host is that the number of transfers should be a multiple of the FIFO data width (`F_DATA_WIDTH`). The software can still program the Byte Count register to only 15, at which point only 15 bytes can be transferred to the card. Similarly, when 15 bytes 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.†

16.5.2. 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 55. SD/MMC Controller Initialization Sequence†



(1) For at least two clocks of clk or cclk_in, whichever is slower.

16.5.2.1. 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 267
  Refer to this section for information on setting the clock source assignments.
- Enumerated Card Stack on page 264
  Refer to this section for information on discovering the card stack according to the card type.

16.5.2.2. 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.

16.5.2.2.1. 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 265.
4. Go to Determine if Card is a CE-ATA 1.1, CE-ATA 1.0, or MMC Device on page 266.
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 of 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 of 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 of 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.

16.5.2.3. 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.

16.5.2.3.1. 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 and update_clk_regs_only bits change 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 294
Refer to this section for information about hardware lock errors.

16.5.2.3.2. Timing Tuning

This section is pending further information.

16.5.3. 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.

16.5.4. 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 270
Refer to this table for information about Non-Data Transfer commands.

16.5.4.1. cmd Register Settings for Non-Data Transfer Command†

Table 133. 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 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 134. 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

16.5.5. 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.†

16.5.5.1. 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.†

16.5.5.2. 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.†

16.5.5.3. 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.†

16.5.5.4. Single-Block or Multiple-Block Read

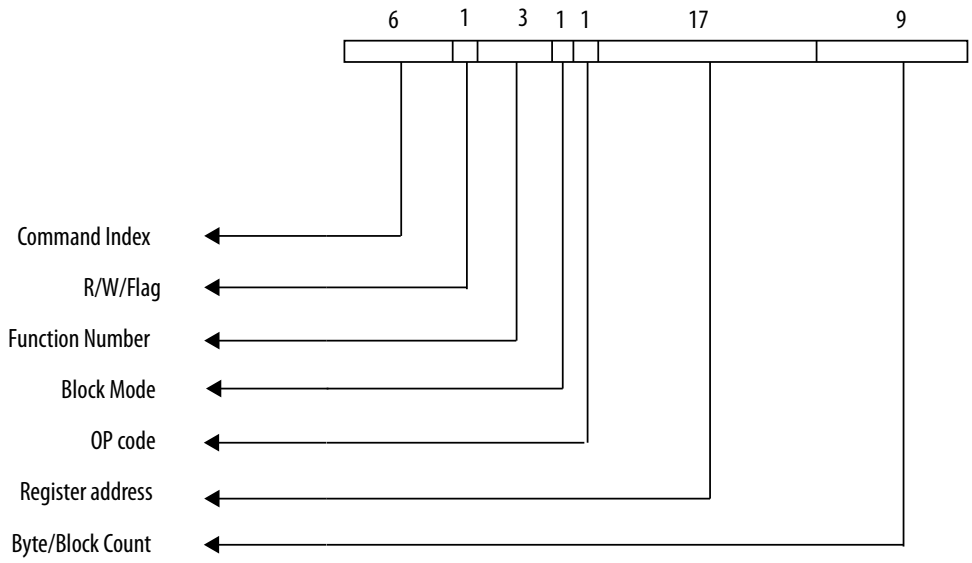
To implement a single-block or multiple-block read, the software performs the following steps:†

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 56. Command Argument for IO_RW_EXTENDED (CMD53)†



Related Information

- Card Read Threshold on page 291
  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 273
  Refer to this table for information about the settings for Single-Block and Multiple-Block Reads.

16.5.5.4.1. cmd Register Settings for Single-Block and Multiple-Block Reads†

Table 135. 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

continued...

Parameter	Value	Comment
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 136. 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 responce 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 250
Refer to this table for information about setting the send_auto_stop parameter.

16.5.5.5. 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.†

16.5.5.5.1. cmd Register Settings for Single-Block and Multiple-Block Write

Table 137. 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.

continued...

Parameter	Value	Comment
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 138. 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 250
Refer to this table for information about setting the send_auto_stop parameter.

16.5.5.6. 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 272
  Refer to this section for more information about a stream read.
- Single-Block or Multiple-Block Write on page 274
  Refer to this section for more information about a stream write.

16.5.5.7. 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 ready to be 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

www.jedec.org
For more information about packed commands, refer to JEDEC Standard No. 84-A441, available on the JEDEC website.

16.5.6. 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. †

16.5.6.1. 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 269
Refer to this section for information on the STOP_TRANSMISSION command.

16.5.6.2. 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 269
Refer to this section for information on the ABORT command.

16.5.6.2.1. 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).

16.5.6.2.2. cmdarg Register Settings for SD/SDIO ABORT Command†

Table 139. 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

16.5.7. 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.

16.5.7.1. 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 279
  Refer to this section for information about the Internal DMA Controller Transmission Sequences.
- Internal DMA Controller Reception Sequences on page 280
  Refer to this section for information about the Internal DMA Controller Reception Sequences.

16.5.7.2. 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.†

16.5.7.3. 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.†

16.5.8. Commands for SDIO Card Devices

This section describes the commands to temporarily halt the transfers between the controller and SDIO card device.

16.5.8.1. 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.†

16.5.8.1.1. 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.†

16.5.8.1.2. 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 140. 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 272
  Refer to this section for more information about writing to the `cmd` register.
- Single-Block or Multiple-Block Write on page 274
  Refer to this section for more information about writing to the `cmd` register.

16.5.8.2. 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.†

16.5.8.2.1. Signalling 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.†

16.5.9. CE-ATA Data Transfer Commands

This section describes CE-ATA data transfer commands.

Related Information

Data Transfer Commands on page 270
Refer to this section for information about the basic settings and interrupts generated for different conditions.

16.5.9.1. 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. †

16.5.9.2. 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.†

16.5.9.2.1. 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 284
Refer to this table for information on how to set these registers.

16.5.9.2.2. Register Settings for ATA Task File Transfer

Table 141. 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)

continued...

Bit	Value	Comment
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 142. 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 143. 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 144. 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.

16.5.9.2.3. 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

www.jedec.org
For information about the (MMC TRAN) state, MMC reset and initialization, refer to JEDEC Standard No. 84-A441, available on the JEDEC website.

16.5.9.3. 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. †

16.5.9.3.1. 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.†

16.5.9.3.2. Register Settings for ATA Payload Transfer

You must set the `cmdarg`, `cmd`, `blksiz`, and `bytcnt` registers according to the following tables.†

Table 145. 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 146. 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 147. 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 148. 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)

16.5.9.4. CE-ATA CCS

This section describes disabling the CCS, recovery after CCS timeout, and recovery after I/O read transmission delay (N_{ACIO}) timeout. †

16.5.9.4.1. 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.†

16.5.9.4.2. 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 277
Refer to this section for more information about sending the STOP command.

16.5.9.4.3. 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 288
For more information about what steps to take if the CCS is expected from the CE-ATA card device.

16.5.9.5. 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.†

16.5.9.5.1. 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†

16.5.9.5.2. 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.†

16.5.9.5.3. 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.†

16.5.9.5.4. 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 †

16.5.9.5.5. 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 †

16.5.10. 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 149. Card Read Threshold Guidelines†

Bus Speed Modes	Round Trip Delay (Delay_R) (37)	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

(37) 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) †

16.5.10.1. 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.†

16.5.10.2. 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 150. Legal Values of dw_dma_multiple_transaction_size and rx_wmark for Block Size = 512†

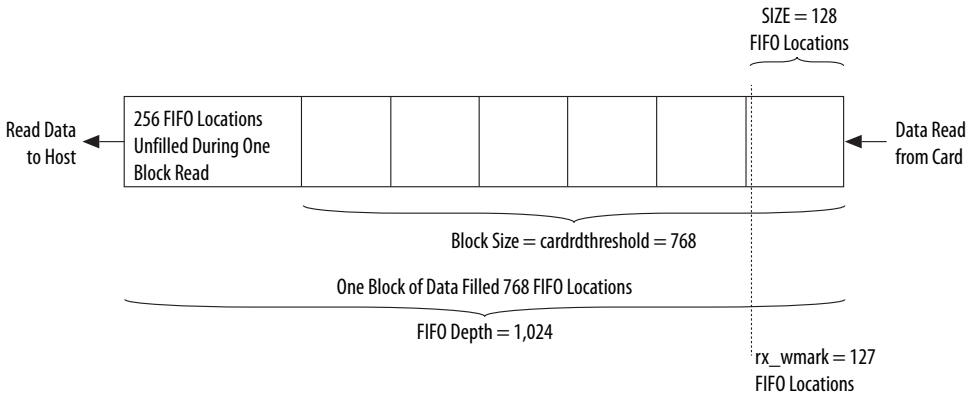
Block Size	dw_dma_multiple_transaction_size	rx_wmark
512	1	0
512	4	3
512	8	7
512	16	15
512	32	31
512	64	63
512	128	127

16.5.10.3. 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 57. FIFO Buffer content when Card Read Threshold is set to 768†



16.5.11. 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.†

16.5.12. 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.

16.5.12.1. 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

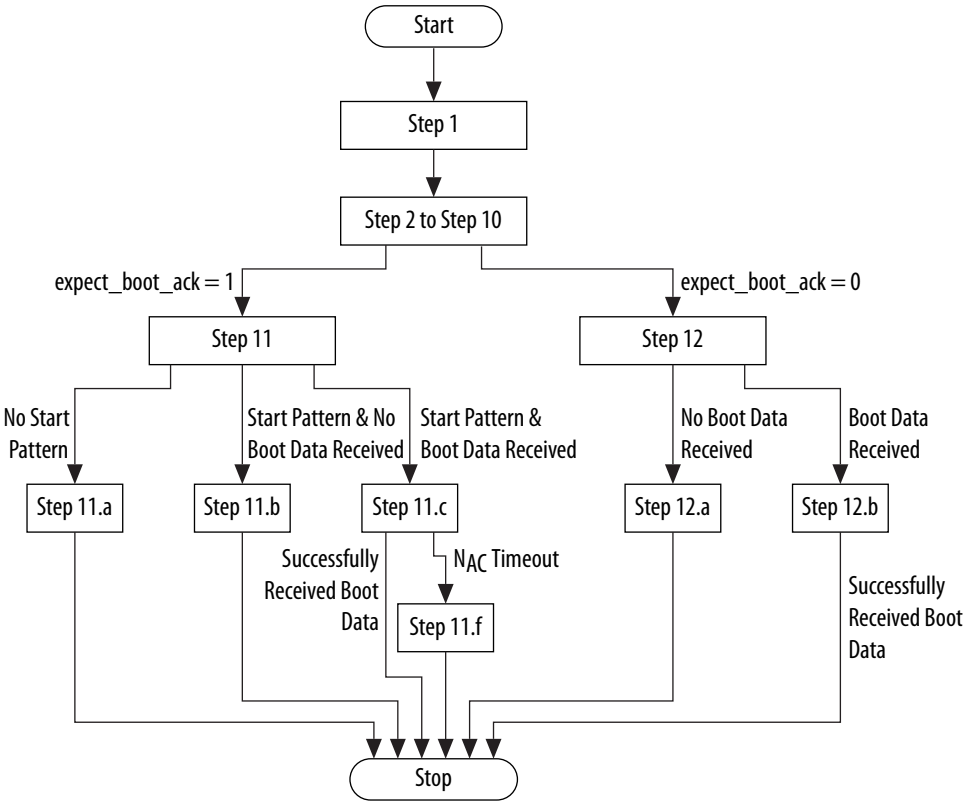
Related Information

www.jedec.org
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.

16.5.12.2. 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 58. 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 267
  Refer to this section for information on how to set the card device frequency.
- Internal DMA Controller Transmission Sequences on page 279
  Refer to this section for information about the Internal DMA Controller Transmission Sequences.
- Internal DMA Controller Reception Sequences on page 280
  Refer to this section for information about the Internal DMA Controller Reception Sequences.

16.5.12.3. Boot Operation for Removable MMC4.3, MMC4.4 and MMC4.41 Cards

16.5.12.3.1. 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†

16.5.12.3.2. 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 264
  Refer to this section for more information on discovering removable MMC cards.
- www.jedec.org
  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 301
  Refer to this section for information about alternative boot operation steps.

16.5.12.4. 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

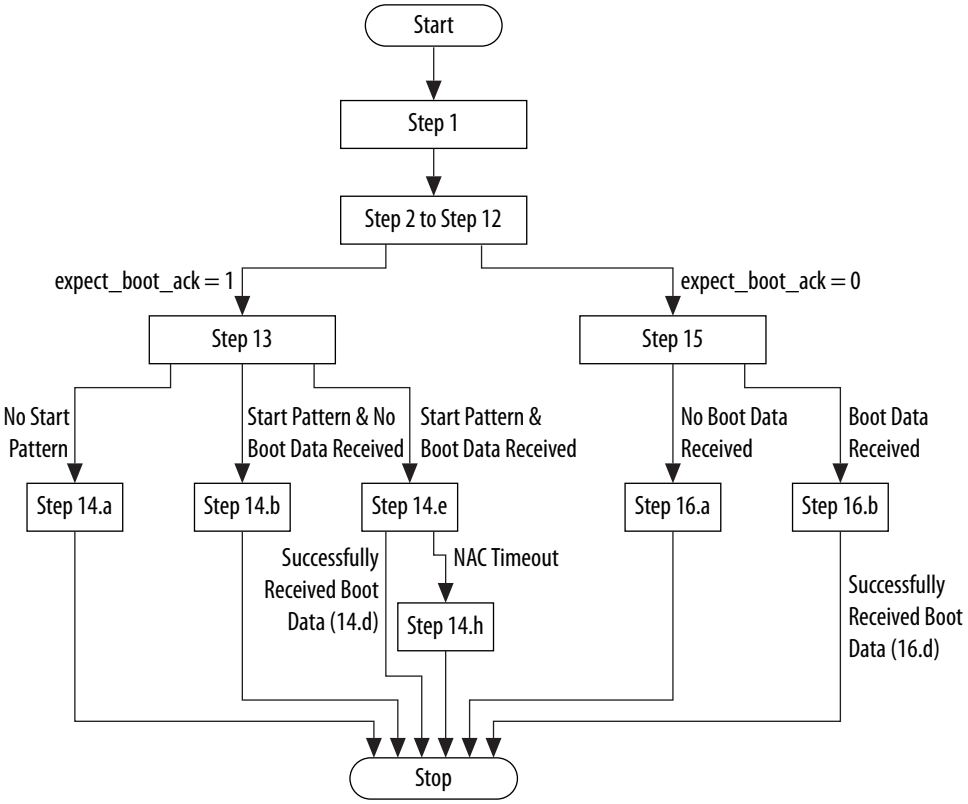
Related Information

www.jedec.org
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.

16.5.12.5. 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 59. 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 267
  Refer to this section for information on how to set the card device frequency.
- Internal DMA Controller Transmission Sequences on page 279
  Refer to this section for information about the Internal DMA Controller Transmission Sequences.
- Internal DMA Controller Reception Sequences on page 280
  Refer to this section for information about the Internal DMA Controller Reception Sequences.

16.5.12.6. Alternative Boot Operation for MMC4.3 Cards

16.5.12.6.1. 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†

16.5.12.6.2. 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 264
  Refer to this section for more information on discovering removable MMC cards.
- www.jedec.org
  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 301
  Refer to this section for information about alternative boot operation steps.

16.6. SD/MMC Controller Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

17. 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 3.74a).

The EMAC has an extensive memory-mapped control and status register (CSR) set, which can be accessed by the Arm Cortex-A53.

For an understanding of this chapter, you should be familiar with the basics of IEEE 802.3 media access control (MAC). (38)

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 Agilex Hard Processor System Technical Reference Manual Revision History on page 12
  For details on the document revision history of this chapter

(38) 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.

17.1. Features of the Ethernet MAC

17.1.1. 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
- 16 KB TX and 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
- Supports Cut-Through or Store and Forward for full Jumbo 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 9000 Bytes)
- Programmable inter-frame gap (IFG), from 40- to 96-bit times in steps of eight bits
- Preamble length of up to one byte 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
- Supports robust set of MAC counters

17.1.2. 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

17.1.3. 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
- Big endian and little endian configurable support for transmission and reception data paths

17.1.4. 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)

17.1.5. 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.
- Reduced Gigabit Media Independent Interface (RGMII) with additional required adaptor logic
- 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

17.2. EMAC Block Diagram and System Integration

Figure 60. EMAC Block Diagram

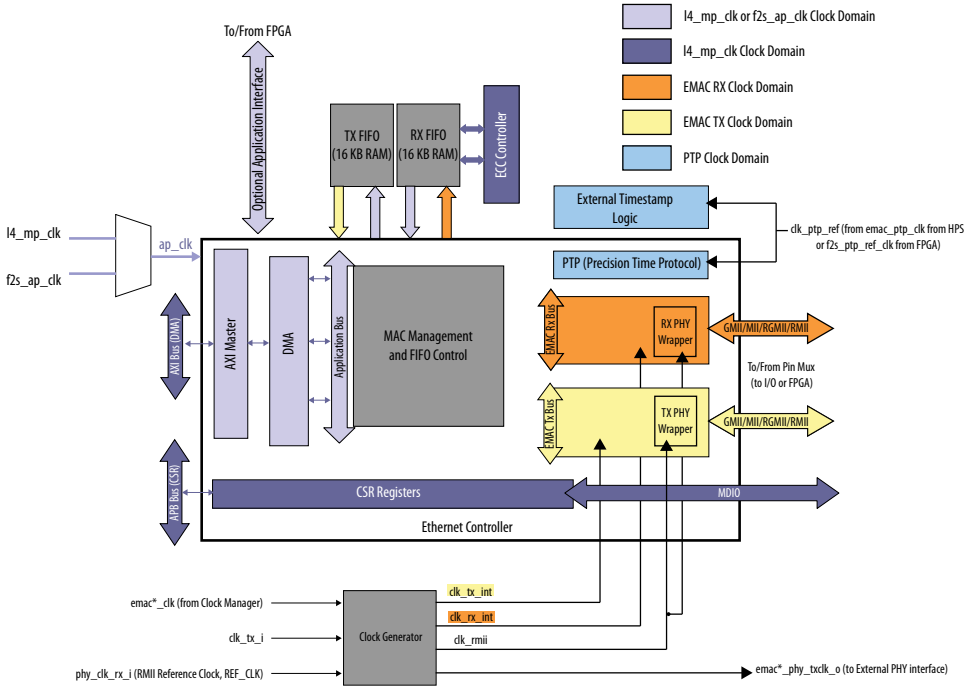
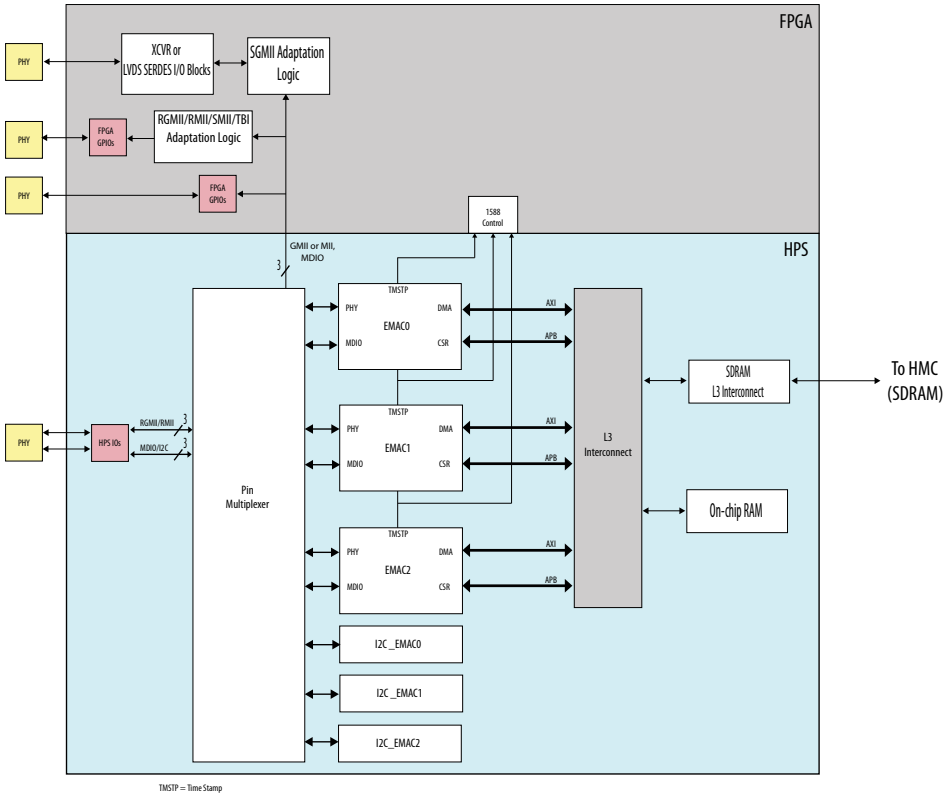


Figure 61. EMAC System Integration



EMAC Overview

Each EMAC contains a dedicated DMA controller that masters 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. This port can be implemented using the I^2C modules in the HPS or the EMAC's MDIO interface.

Each EMAC has an IEEE 1588 Timestamp interface with 10 ns resolution. The Arm Cortex-A53 MPCore processor 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.

17.3. Distributed Virtual Memory Support

The system memory management unit (SMMU) in the HPS supports distributed virtual memory transactions initiated by masters.

As part of the SMMU, a translation buffer unit (TBU) sits between the EMAC and the L3 interconnect. The three Ethernet MACs share a TBU. An intermediate interconnect arbitrates accesses among the three EMACs before they are sent to the TBU. The TBU contains a micro translation lookaside buffer (TLB) that holds cached page table walk results from a translation control unit (TCU) in the SMMU. For every virtual memory transaction that this master initiates, the TBU compares the virtual address against the translations stored in its buffer to see if a physical translation exists. If a translation does not exist, the TCU performs a page table walk. This SMMU integration allows the EMAC driver to pass virtual addresses directly to the EMAC without having to perform virtual to physical address translations through the operating system.

For more information about distributed virtual memory support and the SMMU, refer to the *System Memory Management Unit* chapter.

Related Information

System Memory Management Unit on page 71

17.4. EMAC Signal Description

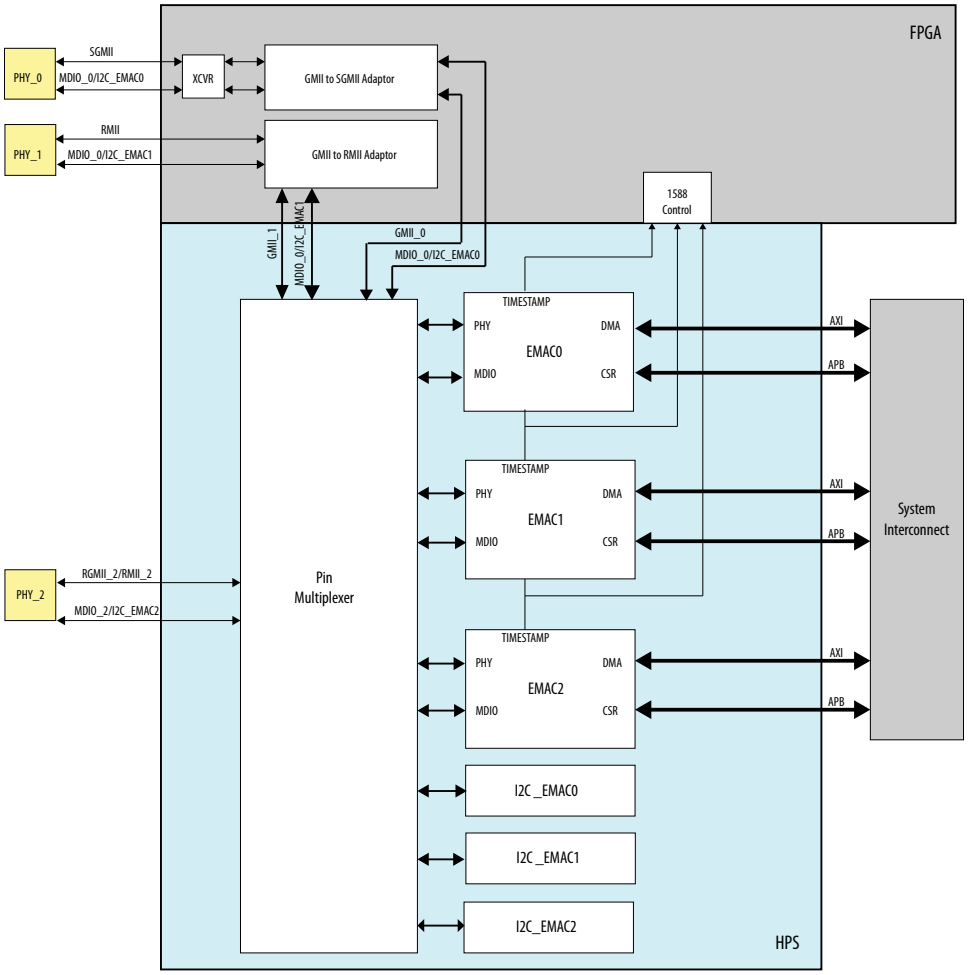
The EMAC provides a variety of PHY interfaces and control options through the HPS and the FPGA I/Os.

For designs that are pin-limited on HPS I/O, the EMAC can be configured to expose either a GMII or MII PHY interface to the FPGA fabric, which can be routed directly to FPGA I/O pins. Exposing the PHY interface to the FPGA fabric also allows adapting the GMII/MII to other PHY interface types such as SGMII, RGMII and RMII using soft logic with the appropriate general purpose or transceiver I/O resources.

The figure below depicts a design which routes the EMAC0 and EMAC1 PHY interfaces through the FPGA fabric to provide an RMII and SGMII interface using FPGA I/O. EMAC2's PHY interface has been configured to use the HPS I/O.

Refer to the "EMAC FPGA Interface Initialization" section to find out more information about configuring EMAC interfaces through FPGA.

Figure 62. EMAC to FPGA Routing Example



17.4.1. 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.

Table 151. HPS EMAC I/O Signals

EMAC HPS I/O		In/Out	Width	Description[39]
EMAC0_TX_CLK EMAC1_TX_CLK EMAC2_TX_CLK	Transmit Clock routed from one of three Platform Designer port	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 for both 10 Mbps or 100 Mbps mode). The other possible source for the common transmit and receive clock is an external clock source, in which case EMACn_TX_CLK

continued...

(39) The "n" in EMACn stands for the EMAC peripheral number.

EMAC HPS I/O		In/Out	Width	Description[39]
	signals `emac[2:0]_phy_txclk_o`			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 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 1000Mbps mode, the data bus carries transmit data at double data rate and are sampled on both the rising and falling edges of the transmit clock. In RGMII and RMII 10/100Mbps modes, the data bus is single data rate, synchronous to the rising edge of the transmit clock. Additionally in RMII 10Mbps mode, the data and control signals are held stable for 10 transmit clock cycles. 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 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`.
				Same clock to data relationships on CTL as with the data in the above row across the modes.
				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 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 port signals `emac[2:0]_phy_rxd[3:0]`	In	4	These data signals are received from the PHY. In RGMII 1000 Mbps mode, data is received at double data rate with bits[3:0] valid on the rising and falling edges of `EMACn_RX_CLK`. In RGMII 10/100Mbps modes, data is received at single data rate with bits[3:0] valid on the rising edge 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`. Additionally in RMII 10Mbps mode, the data and control signals are held stable for 10 receive clock cycles. 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 Designer port signals `emac[2:0]_phy_rxdv`.	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.

(39) The "n" in EMACn stands for the EMAC peripheral number.

17.4.1.1. HPS-to-PHY Interface Diagrams

Each EMAC module in the HPS supports one PHY interface. If you are using the HPS pins for interfacing to a PHY, the following diagrams show the interface options available depending on what PHY you choose.

Figure 63. HPS EMAC to RGMII PHY Interface

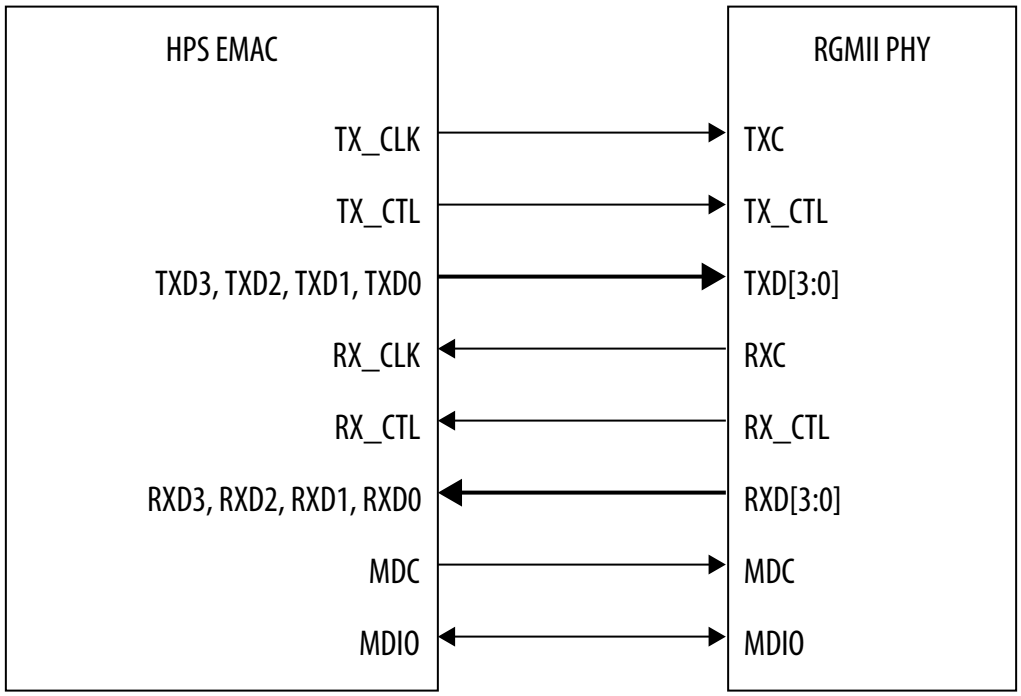


Figure 64. HPS EMAC to RMII PHY with HPS-Sourced Reference Clock

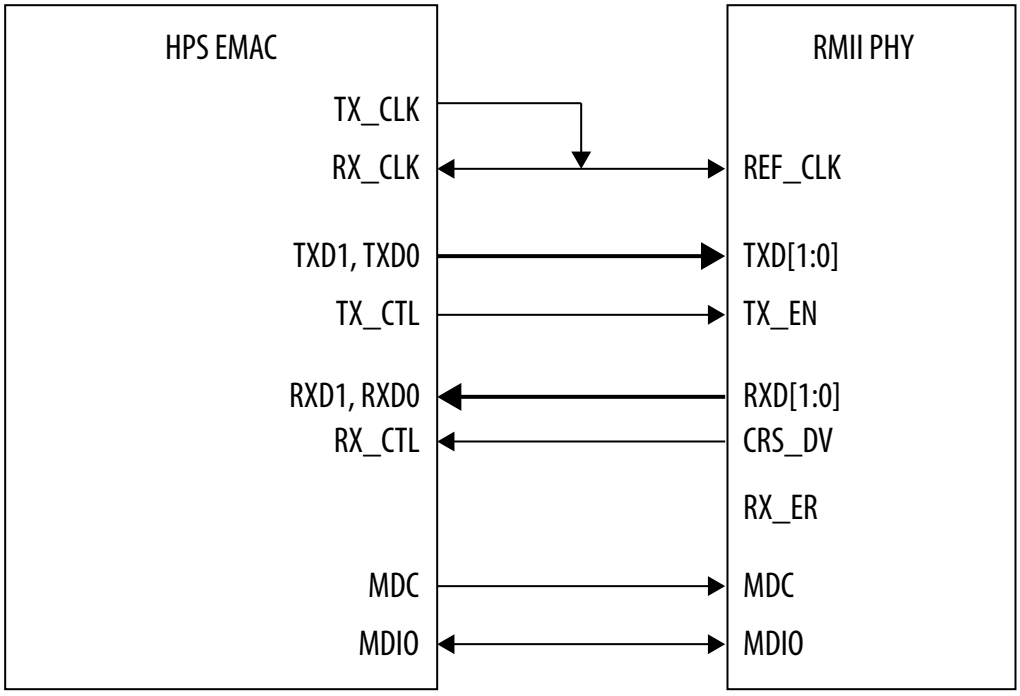
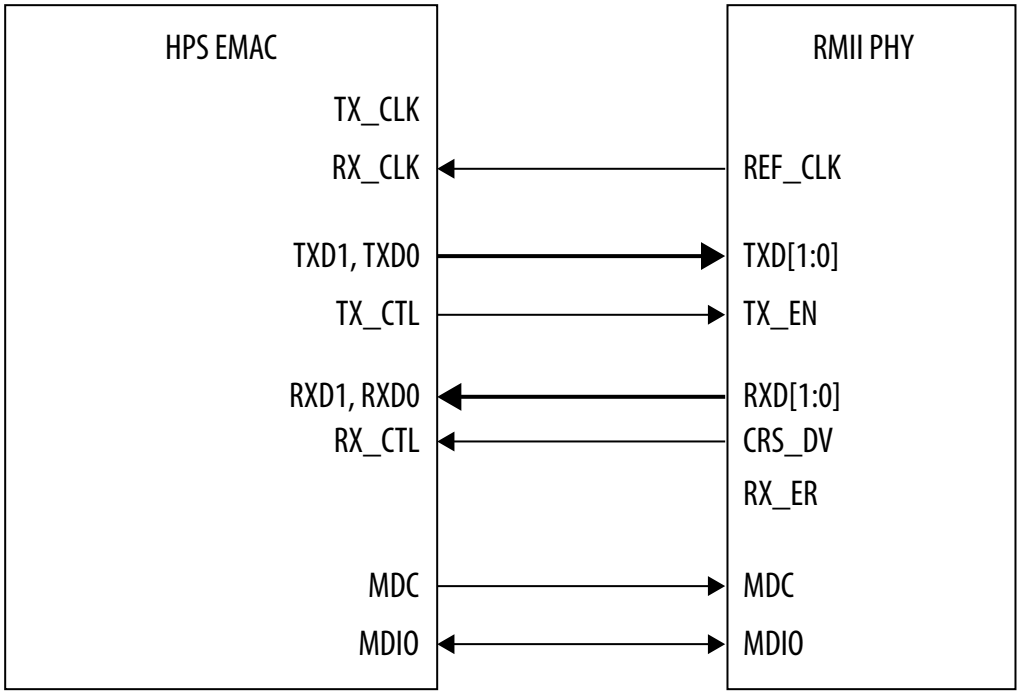


Figure 65. HPS EMAC to RMII PHY with PHY-Sourced Reference Clock



17.4.2. 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, refer to the HPS I/O table for general clock to data relationships across the modes.

Table 152. 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 PHY in 10/100 Mbps modes. 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 not used by the PHY, however the transmit clock input from the PHY in 10/100Mbps modes of operation (input on emac_clk_tx_i) is muxed onto this clock output and should be used for the synchronous clock by any adaptation logic on the transmit data and control path in the FPGA fabric for GMII and MII modes.
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 10/100 Mbps modes of 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 phy_txd_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_o 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 phy_txclk_o 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.

continued...

Signal Name		In/Out	Width	Description
				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 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_i 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 clk_rx_i.
emac_rst_clk_rx_n_o	Receive clock reset output.	Out	1	Receive clock reset output, synchronous to clk_rx_i. The reset pulse width of the rst_clk_rx_n_o signal is three transmit clock cycles.
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.

17.4.3. PHY Management Interface

The HPS can provide support for either MDIO or I^2C PHY management interfaces.

17.4.3.1. 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 153. 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 signal is asserted whenever valid data is driven on the gmii_mdo_o signal and can be used as a tri-state control for the gmii_mdo_o FPGA I/O tri-state output buffers. 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.

17.4.3.2. I²C External PHY Management Interface

Some PHY devices use the I²C 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 I²C peripherals for controlling the PHY devices:

- i2c_emac_0
- i2c_emac_1
- i2c_emac_2

17.4.4. PHY Interface Options

The table below identifies the signals used for each PHY interface selected.

Table 154. PHY Interface Options

Port Name	MII	GMII	RMII	RGMII	SGMII
emac_phy_txd_o[7:0]	Yes, [3:0]	Yes, [7:0]	Yes, [1:0]	Yes, [3:0]	Yes, [7:0]
emac_phy_mac_speed_o(40)	Yes	Yes	Yes	Yes	Yes
emac_phy_txen_o	Yes	Yes	Yes	Yes	Yes
emac_phy_txer_o(40)	No	Yes	No	No	Yes, part of transmit code
emac_phy_rxdv_i	Yes	Yes	Yes	Yes	Yes, part of receive code

continued...

(40) This signal is only available through the FPGA interface.

Port Name	MII	GMII	RMII	RGMII	SGMII
emac_phy_rxer_i[40]	Yes	Yes	No	No	Yes, part of receive code
emac_phy_rxd_i[7:0]	Yes, [3:0]	Yes, [7:0]	Yes, [1:0]	Yes, [3:0]	Yes, [7:0]
emac_phy_col_i[40]	Yes	Yes	No	No	No
emac_phy_crs_i[40]	Yes	Yes	No	No	No
emac_clk_rx_i	Yes	Yes	Yes	Yes	Yes
emac_clk_tx_i[40]	Yes	Yes	No	No	Yes
emac_phy_txclk_o	No	No	Yes	Yes	No
emac_rst_clk_tx_n_o[40]	Yes	Yes	No	No	No
emac_rst_clk_rx_n_o[40]	Yes	Yes	No	No	No
emac_gmii_mdc_o	Yes	Yes	Yes	Yes	Yes
emac_gmii_mdo_o, emac_gmii_mdo_o_e, emac_gmii_mdi_i[41]	Yes	Yes	Yes	Yes	Yes
emac_ptp_pps_o[42]	Yes	Yes	No	No	No
emac_ptp_aux_ts_trig_i[42]	Yes	Yes	No	No	No

17.5. EMAC Internal Interfaces

17.5.1. 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.

(41) These three signals make up the MDIO output signal.

(42) This is an optional signal.

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 326
  Information regarding DMA Controller functionality
- System Manager on page 171

17.5.2. 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 155. 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`.

continued...

Signal Name		In/Out	Width	Description
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 351
More information regarding IEEE 1588-2002 timestamp functionality

17.5.3. 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 156. 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.

continued...

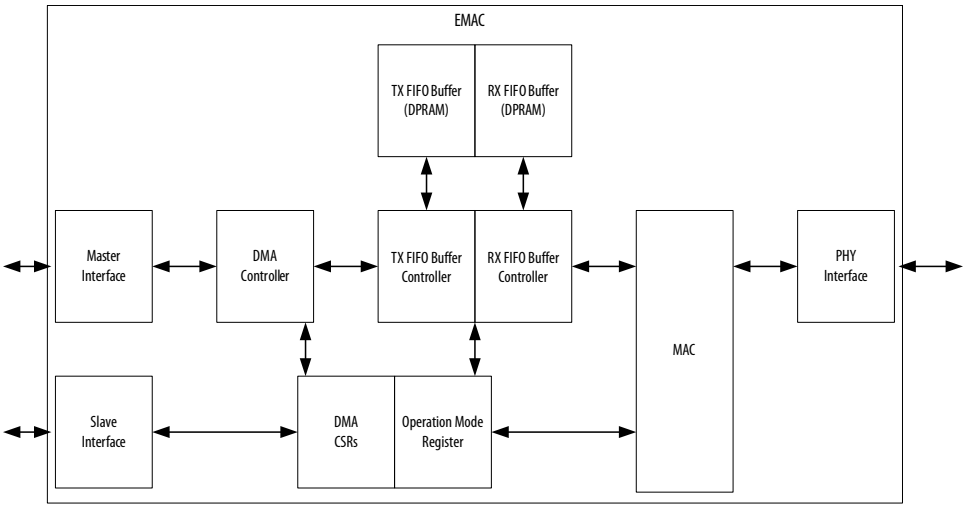
Function	Description
PTP Timestamp Clock Select	Selects the source of the PTP reference clock between `emac_ptp_clk` from the Clock Manager or `f2s_emac_ptp_ref_clk` from the FPGA Fabric. All three EMAC modules must use the same reference clock.
AXI Cache and Protection Settings	Static settings are provided 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 Manager on page 171

17.6. Functional Description of the EMAC

Figure 66. 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 16 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.

17.6.1. 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 and TX FIFO buffers are each 16KB dual ported memories. 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 Register 0 (Bus Mode Register).

  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 128

17.6.2. 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 †

17.6.2.1. 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 67. Descriptor Ring Structure

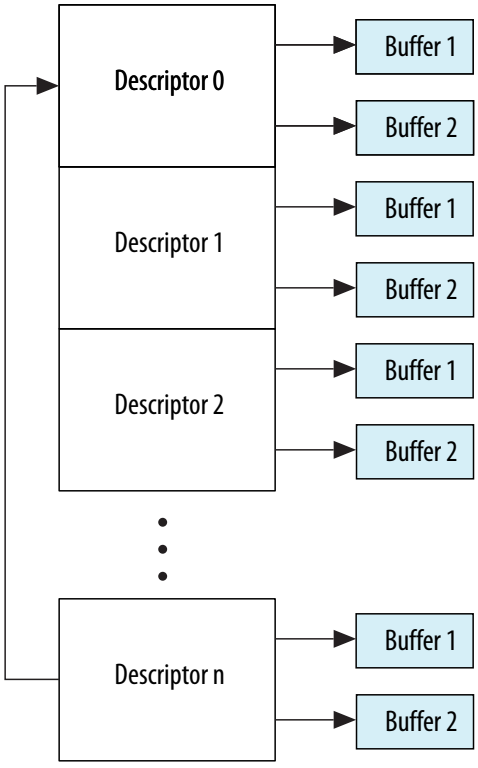
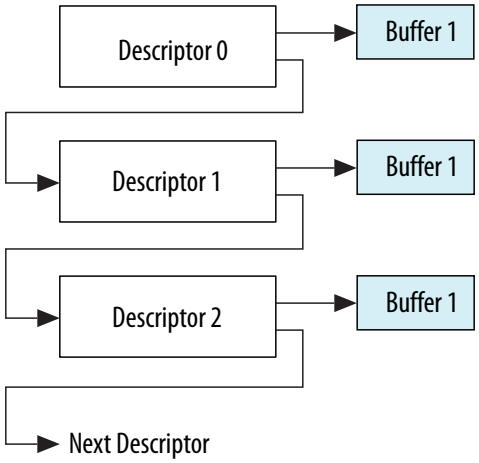


Figure 68. Descriptor Chain Structure



Note: 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.

17.6.2.2. 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.

17.6.2.3. 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.†

17.6.2.3.1. 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.

17.6.2.3.2. 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.

17.6.2.4. 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.

17.6.2.5. Transmission

The DMA can transmit with or without an optional second frame (OSF).

Related Information

Transmit Descriptor on page 340

17.6.2.5.1. TX DMA Operation: Default (Non-OSF) Mode

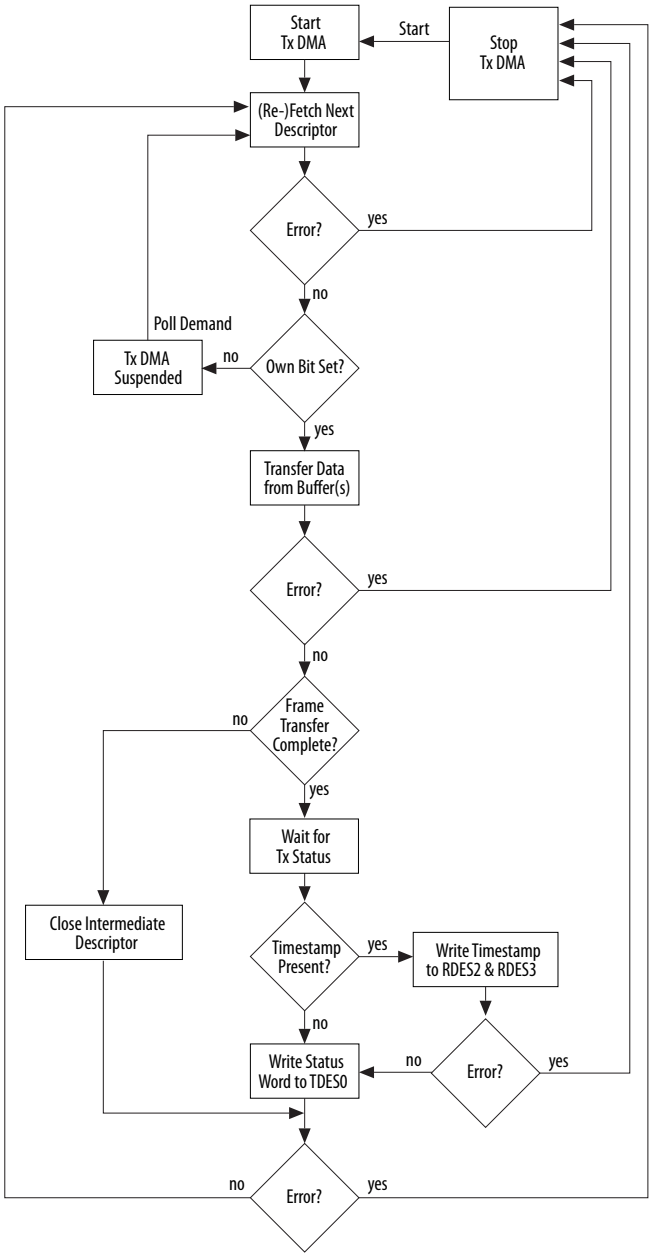
The transmit DMA engine in default mode proceeds as follows: †

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 331.
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 330, 4 on page 330, and 5 on page 330 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 330. †
9. In the Suspend state, the DMA tries to re-acquire the descriptor (and thereby return to 3 on page 330) when it receives a Transmit Poll demand and the Underflow Interrupt Status bit is cleared. †

Figure 69. TX DMA Operation in Default Mode



17.6.2.5.2. TX DMA Operation: OSF Mode

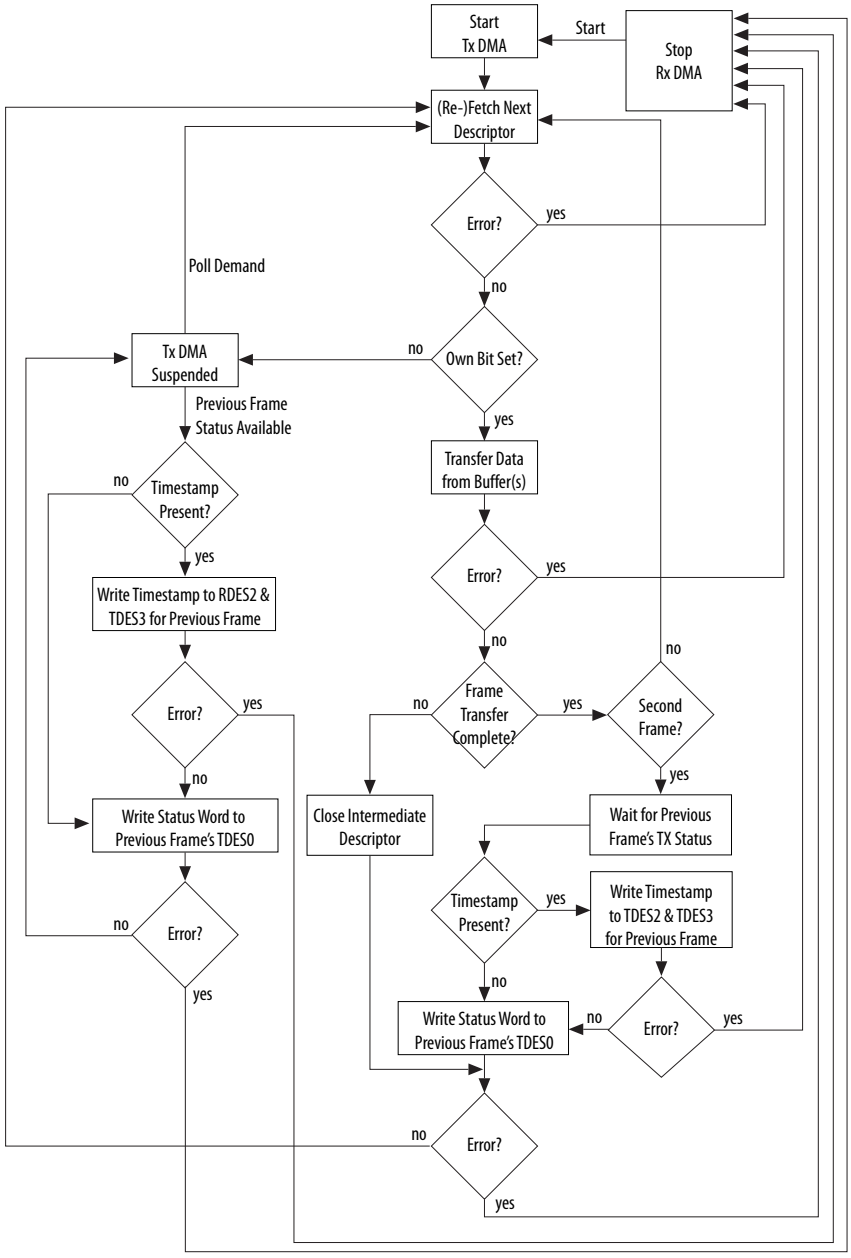
While in the Run state, the transmit process can simultaneously acquire two frames without closing the Status descriptor of the first [if Bit 2 (OSF) in Register 6 (Operation Mode Register) is set]. As the transmit process finishes transferring the first frame, it immediately polls the transmit descriptor 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" 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 332.†
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 332 (when Status is normal). If the previous transmission status shows an underflow error, the DMA goes into Suspend mode (7 on page 332). †
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 332 or 2 on page 332 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 70. TX DMA Operation in OSF Mode



Related Information

TX DMA Operation: Default (Non-OSF) Mode on page 330

17.6.2.5.3. 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. †

17.6.2.5.4. 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. †

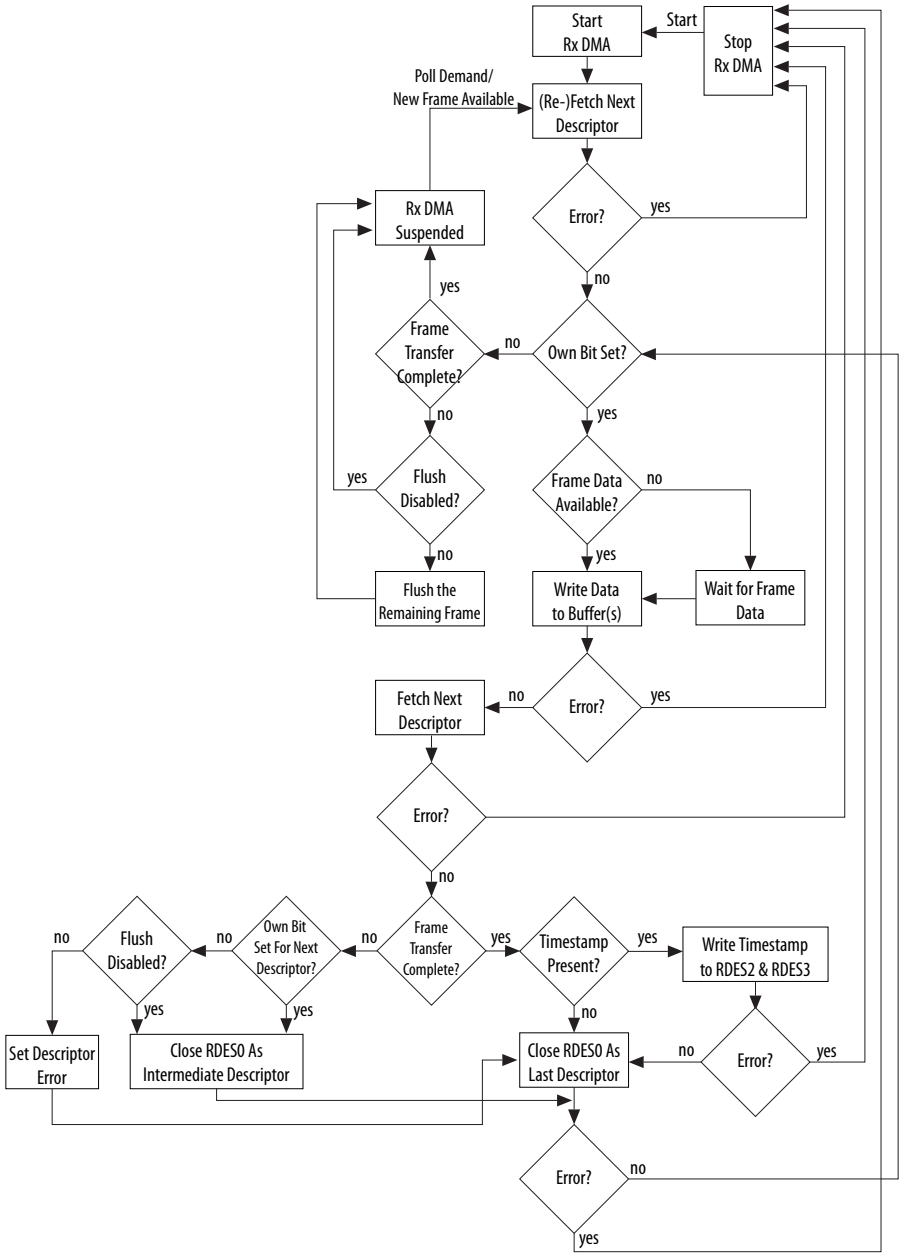
17.6.2.6. Reception

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 335.†
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 335. 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 335. 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 335.†
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 335 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 335 and refetches the next descriptor. †

Figure 71. 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.

17.6.2.6.1. 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. †

17.6.2.6.2. 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. †

17.6.2.6.3. 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. †

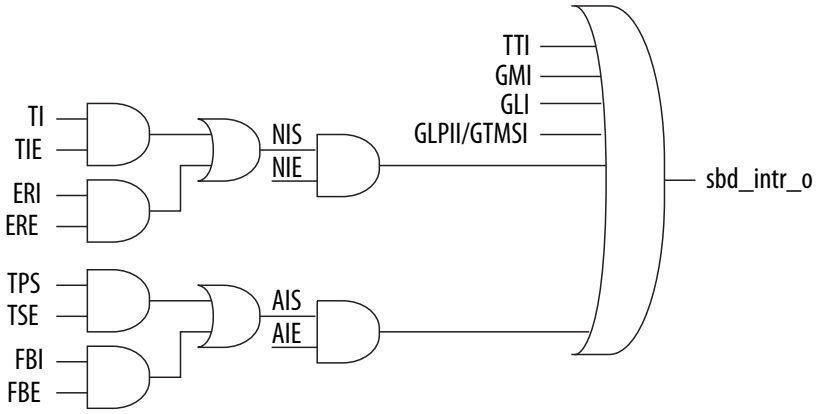
The discarding or flushing of the frame at the top of the MTL EX FIFO buffer can be avoided by disabling Flushing (Bit 24 of Register 6 (Operation Mode Register)). In such conditions, the receive process sets the Receive Buffer Unavailable status and returns to the Suspend state. †

17.6.2.7. 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 72. Summary Interrupt (sbd_intr_o) Generation(43)



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

(43) 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 345

17.6.2.8. 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.

17.6.3. 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 326

17.6.3.1. Transmit Descriptor

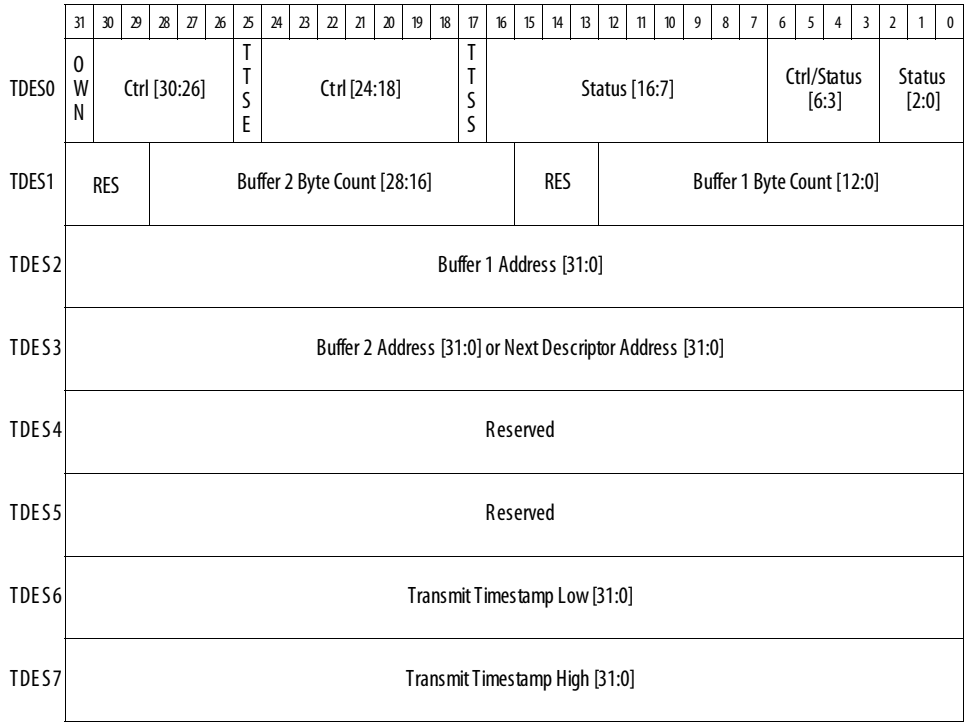
The application software must program the control bits TDES0[31:18] during the transmit descriptor initialization. When the DMA updates the descriptor, it writes back all the control bits except the OWN bit (which it clears) and updates the status bits[7:0].

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 73. 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 74. Transmit Descriptor Fetch (Read) Format

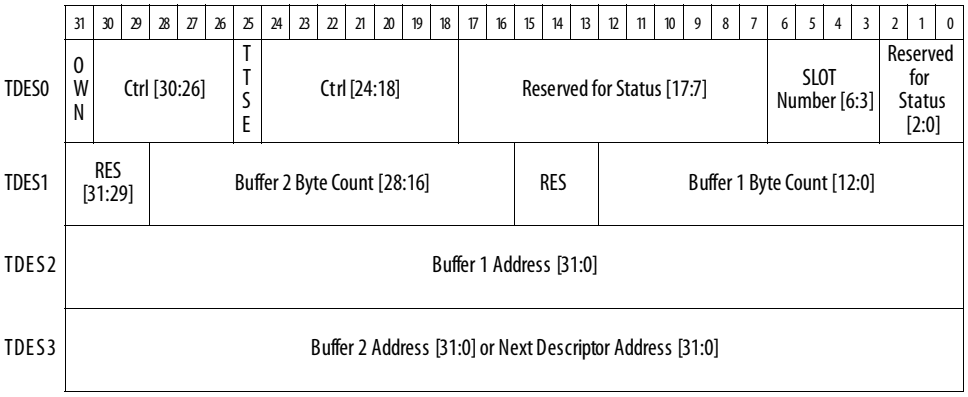


Table 157. Transmit Descriptor Word 0 (TDES0)

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 158. 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 159. 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 160. 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 161. 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 162. Transmit Descriptor 7 (TDES7)

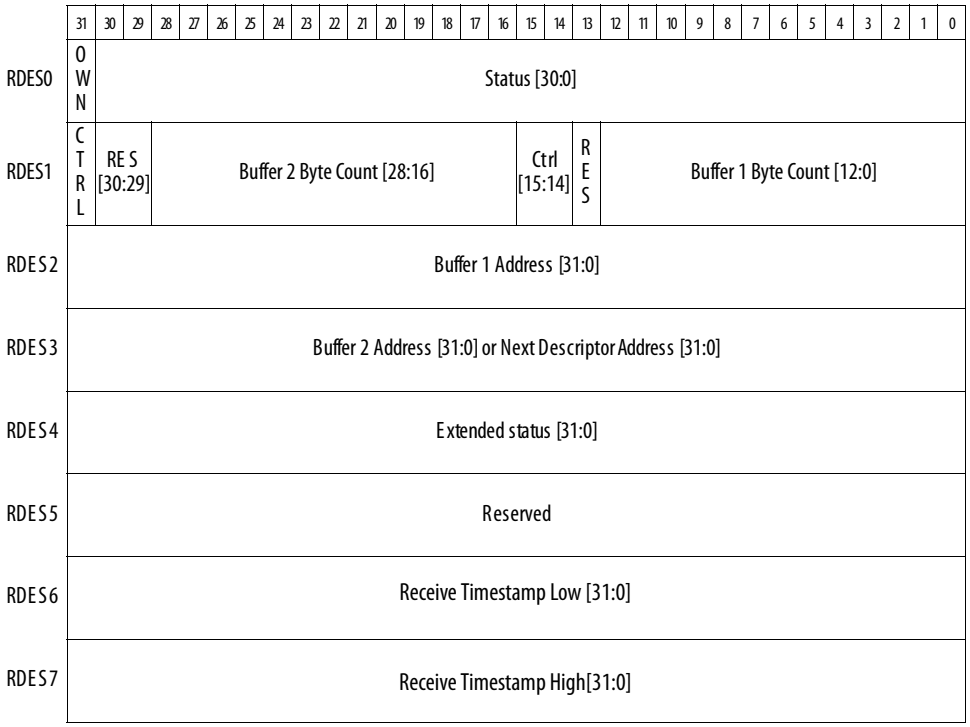
Bit	Description
31:0	TTSH: Transmit Frame Timestamp High This field is updated by DMA with the most significant 32 bits of the timestamp captured for the corresponding receive frame. This field has the timestamp only if the Last Segment bit (LS) in the descriptor is set and Timestamp status (TTSS) bit is set. †

17.6.3.2. 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 75. Receive Enhanced Descriptor Fields Format



17.6.3.2.1. Receive Descriptor Field 0 (RDES0)

Table 163. 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 349
- Receive Descriptor Field 6 (RDES6) on page 351
- Receive Descriptor Field 7 (RDES7) on page 351

17.6.3.2.2. Receive Descriptor Field 1 (RDES1)

Table 164. 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 329
- Receive Descriptor Field 2 (RDES2) on page 348
- Receive Descriptor Field 3 (RDES3) on page 349

17.6.3.2.3. Receive Descriptor Fields (RDES2) and (RDES3)

Receive Descriptor Field 2 (RDES2)

Table 165. 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 329

Receive Descriptor Field 3 (RDES3)

Table 166. 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 329
- Receive Descriptor Field 1 (RDES1) on page 348

17.6.3.2.4. 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 167. 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 345

17.6.3.2.5. 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 345

Receive Descriptor Field 6 (RDES6)

Table 168. 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 345

Receive Descriptor Field 7 (RDES7)

Table 169. 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 345

17.6.4. IEEE 1588-2002 Timestamps

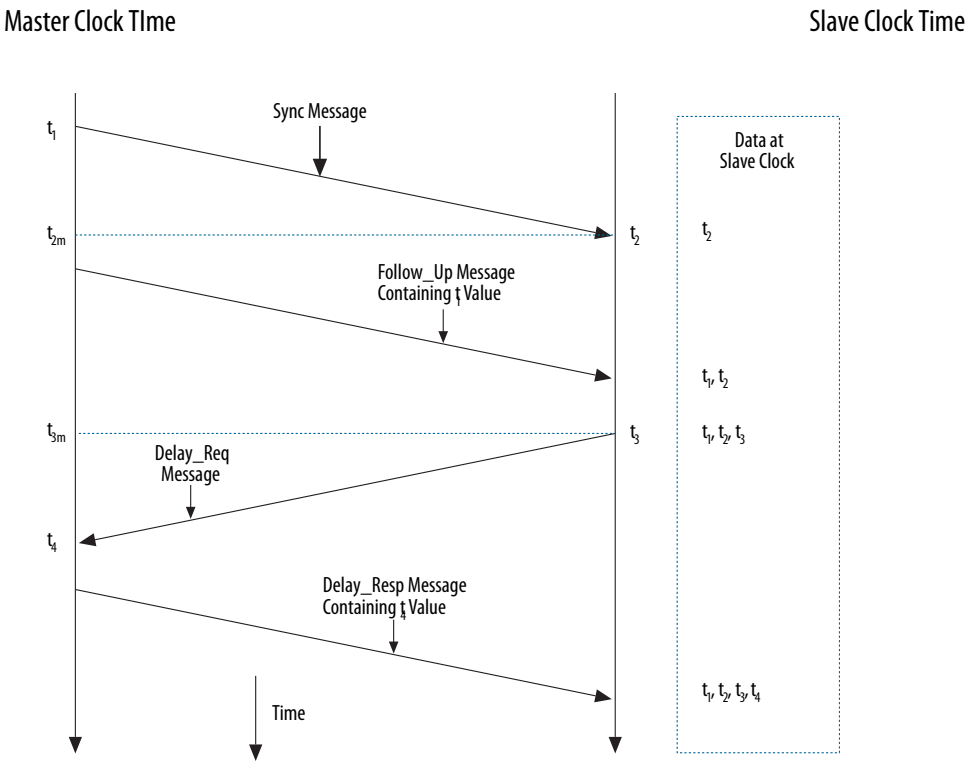
The IEEE 1588-2002 standard defines the Precision Time Protocol (PTP) that enables precise synchronization of clocks in a distributed network of devices. The PTP applies to systems communicating by local area networks supporting multicast messaging. This protocol enables heterogeneous systems that include 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 76. 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 Cortex-A53 MPCore processor 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.†

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.†

17.6.4.1. 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.†

17.6.4.2. 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.

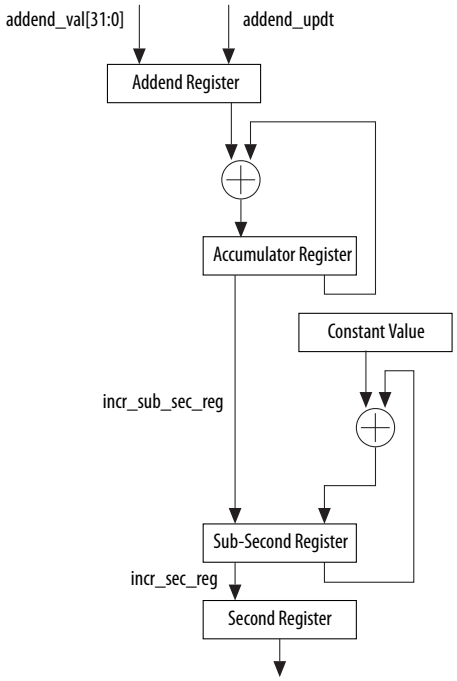
The system time counter can be initialized or corrected using the coarse correction method. In this method, the initial value or the offset value is written to the Timestamp Update register. For initialization, each EMAC's system time counter is written with the value in the Timestamp Update registers, 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 77. 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.†

17.6.4.3. 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 conrol 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 corrsponding 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.†

17.6.4.4. 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.†

17.6.4.5. 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.†

17.6.4.6. 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 170. 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.

17.6.5. 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.*

17.6.5.1. 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.†

17.6.5.2. 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†

17.6.5.2.1. 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 171. Ordinary Clock: PTP Messages for Snapshot†

Master	Slave
Delay_Req	SYNC

17.6.5.2.2. 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.†

17.6.5.2.3. 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 ingess 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 172. 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

17.6.5.2.4. 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 173. 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.†

17.6.5.3. 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

17.6.5.4. Transmit Path Functions

The advanced timestamp feature is supported through the descriptors format.

17.6.5.5. 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.

17.6.5.6. 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). †

17.6.6. 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.†

17.6.6.1. 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.†

17.6.7. 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†

17.6.8. Frame Filtering

The EMAC implements the following types of filtering for receive frames.

17.6.8.1. Source Address or Destination Address Filtering

The Address Filtering Module checks the destination and source address field of each incoming packet.†

17.6.8.1.1. 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.†

17.6.8.1.2. 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.†

17.6.8.1.3. 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.†

17.6.8.1.4. 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.†

17.6.8.1.5. 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.†

17.6.8.1.6. 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.†

17.6.8.1.7. 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 174. 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 Perfrect/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 175. Source Address Filtering†

Frame Type	PR	DAIF	DBF	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

17.6.8.2. VLAN Filtering

The EMAC supports the two kinds of VLAN filtering:

- VLAN tag-based filtering†
- VLAN hash filtering†

17.6.8.2.1. 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.†

17.6.8.2.2. 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.†

17.6.8.3. 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†

17.6.8.3.1. 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.†

17.6.8.3.2. 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.†

17.6.8.3.3. 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.†

17.6.8.3.4. 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

17.6.8.3.5. 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.†

17.6.8.3.6. 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.†

17.6.9. Clocks and Resets

17.6.9.1. Clock Structure

The Ethernet Controller has four main clock domains.

- l4_mp_clk clock
- EMAC RX clock
- EMAC TX clock
- clk_ptp_ref

Figure 78. EMAC Clock Diagram

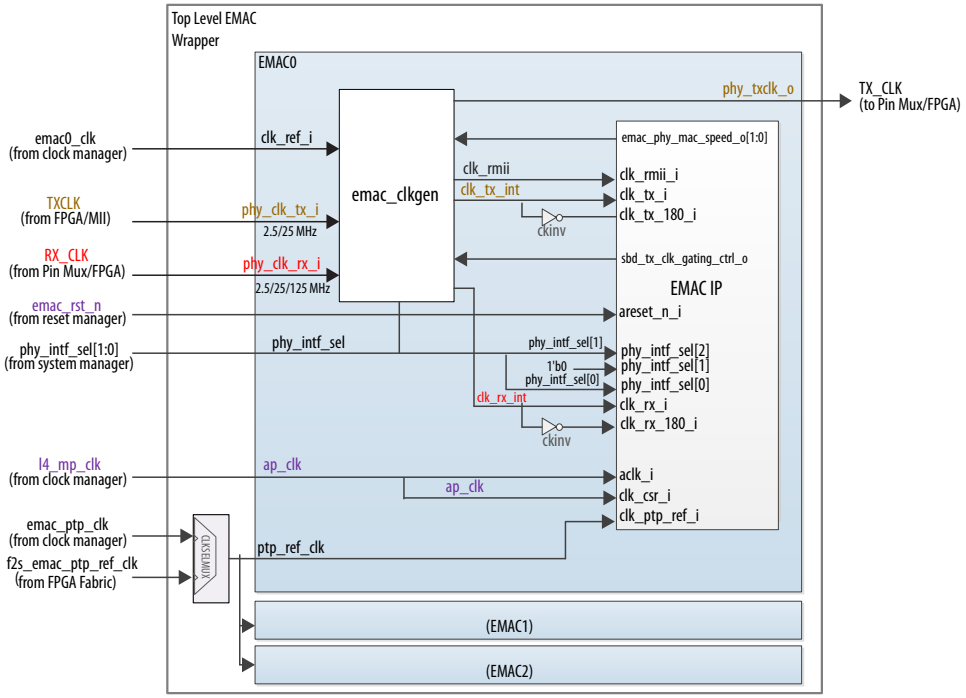
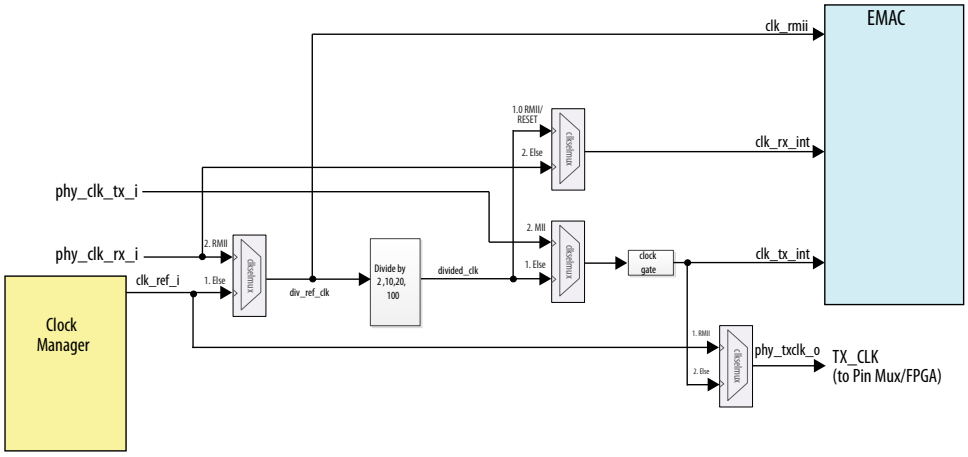


Figure 79. emac_clkgen Module



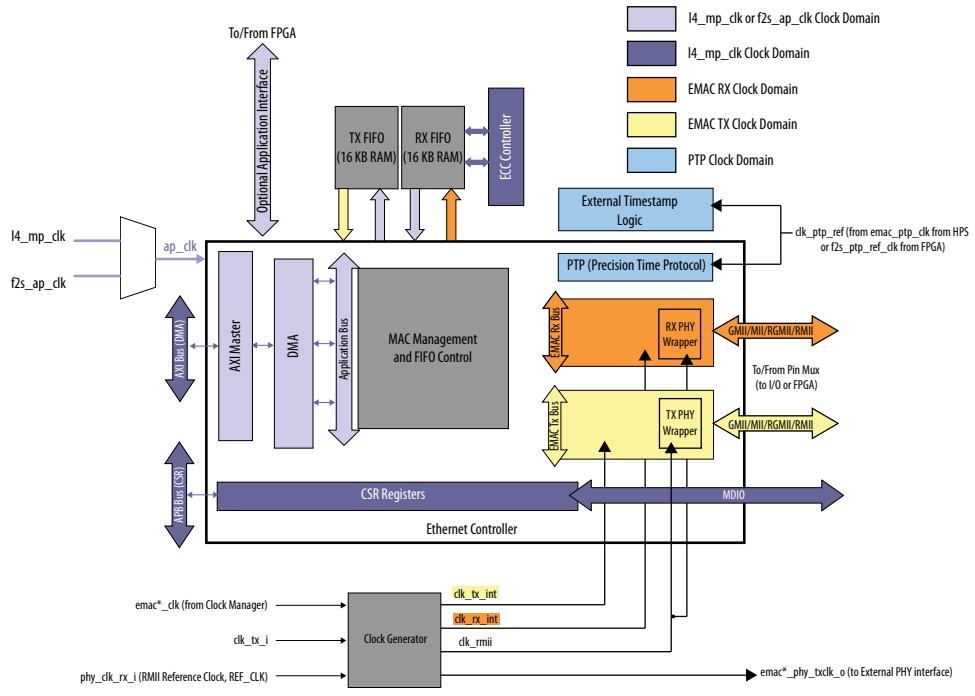
Depending on the interface, different clock domains are used:

- 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 80. EMAC Clock Domains



The following table summarizes the clock inputs and outputs to the EMAC.

Table 176. 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).

17.6.9.2. Clock Gating for EEE

For the RGMII PHY interface, you can gate the transmit clock for Energy Efficient Ethernet (EEE) applications.

17.6.9.3. 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.

17.6.9.3.1. 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.

17.6.10. 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.

17.7. 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.

17.7.1. 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 177. 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 178. 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 emac2.arprot	EMAC Master AxPROT settings. It is recommended that these bits are set while the EMAC is idle or in reset.
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 179. 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

17.7.2. 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.
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 verify that there are RX PHY clocks. For verification, you may have to coordinate with the transceiver bring up from reset.
4. If the PTP clock source is from the FPGA, 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. 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 RGMII, 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.

17.7.3. 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 verify that there are RX PHY clocks.
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.

17.7.4. 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 375) 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 375 only if you did not perform step 3 on page 374.†

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). †

17.7.5. 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.

17.7.6. 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)). †

17.7.7. 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. †

17.7.8. Programming Guidelines for Energy Efficient Ethernet

17.7.8.1. 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 377 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. †

17.7.8.2. 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.†

17.7.8.2.1. 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. †

17.7.8.2.2. 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. †

17.7.9. Programming Guidelines for Flexible Pulse-Per-Second (PPS) Output

17.7.9.1. 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.

17.7.9.2. 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 379 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 379) by programming 0110 in the PPSCMD field before the time (programmed in step 6 on page 379) 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 379) elapses. †

17.7.9.3. 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.

17.8. Ethernet MAC Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

18. 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 is fully compliant with the *On The Go and Embedded Host Supplement to the USB Revision 1.3 and 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 © 2016 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.

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

18.1. 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:

- ULPI PHY support for unidirectional or bidirectional 8-bit SDR bus interface
- 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)

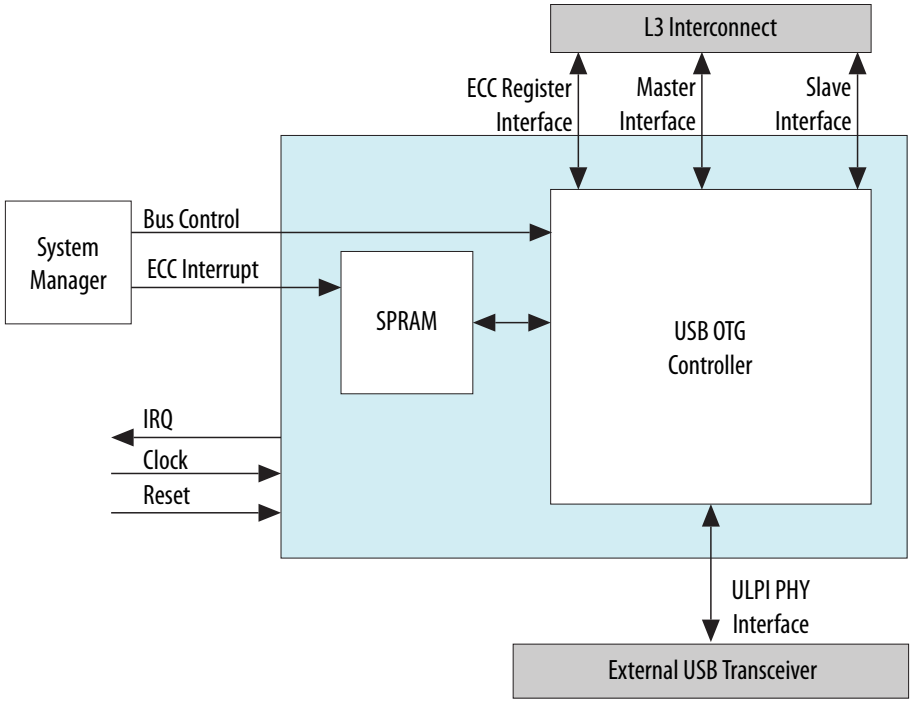
18.1.1. Supported PHYs

The USB OTG controller only supports USB 2.0 ULPI PHYs.

18.2. Block Diagram and System Integration

Figure 81. USB OTG Controller System Integration

Two subsystems are included in the HPS.



The USB OTG controller connects to the layer 3 (L3) interconnect through an L4 slave bus interface, allowing other masters to access the control and status registers (CSRs) in the controller. The controller also connects to the L3 interconnect through the I/O translation buffer unit (TBU), 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 171
  Details available in the System Manager chapter.
- General-Purpose I/O Interface on page 470

18.3. Distributed Virtual Memory Support

The system memory management unit (SMMU) in the HPS supports distributed virtual memory transactions initiated by masters.

As part of the SMMU, a translation buffer unit (TBU) sits between the USB and the L3 interconnect. The USB shares a TBU with the NAND, SD/MMC and ETR. An intermediate interconnect arbitrates accesses among the multiple masters before they are sent to the TBU. The TBU contains a micro translation lookaside buffer (TLB) that holds cached page table walk results from a translation control unit (TCU) in the SMMU. For every virtual memory transaction that this master initiates, the TBU compares the virtual address against the translations stored in its buffer to see if a physical translation exists. If a translation does not exist, the TCU performs a page table walk. This SMMU integration allows the USB driver to pass virtual addresses directly to the USB without having to perform virtual to physical address translations through the operating system.

For more information about distributed virtual memory support and the SMMU, refer to the *System Memory Management Unit* chapter.

Related Information

System Memory Management Unit on page 71

18.4. USB 2.0 ULPI PHY Signal Description

Table 180. ULPI PHY Interfaces

The ULPI PHY interface is synchronous to the `ulpi_clk` signal coming from the PHY.

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

continued...

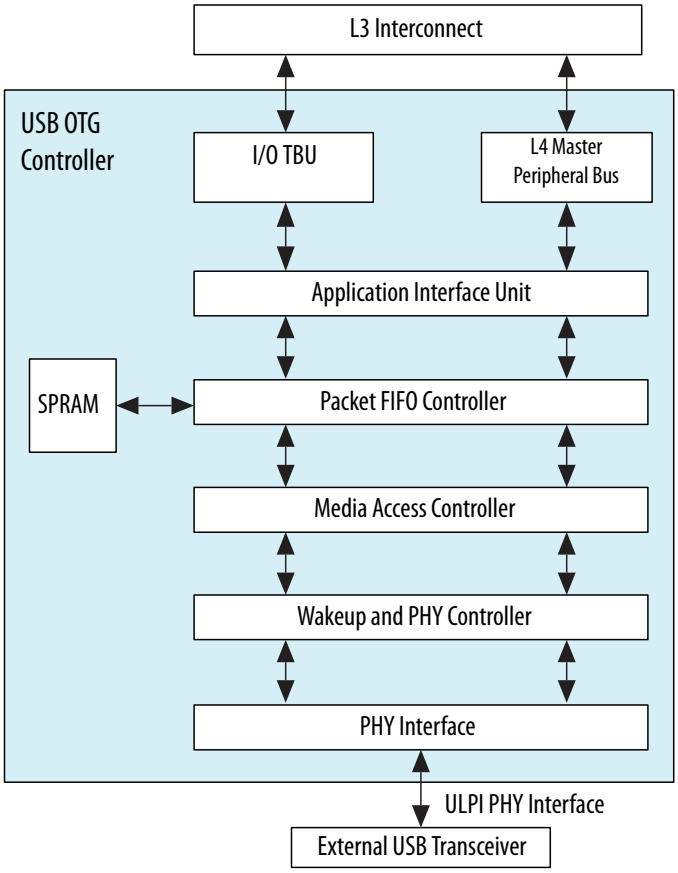
Port Name	Bit Width	Direction	Description
			Indicates that the PHY has accepted the current byte from the USB OTG controller. When the PHY is transmitting, this signal indicates that a new byte is available for the controller.
ulpi_stp	1	Output	ULPI Stop Data Control The controller drives this signal 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.

18.5. Functional Description of the USB OTG Controller

18.5.1. USB OTG Controller Components

Figure 82. USB OTG Controller Block Diagram

Details about each of the units that comprise the USB OTG controller are shown below.



18.5.1.1. 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.

18.5.1.2. 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).

18.5.1.2.1. 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.

18.5.1.3. 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.

18.5.1.4. 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.

18.5.1.5. 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.

18.5.1.6. MAC

The MAC module implements the following functionality:

- USB transaction support
- Host protocol support
- Device protocol support
- OTG protocol support

18.5.1.6.1. 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

18.5.1.6.2. 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.

18.5.1.6.3. 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

18.5.1.6.4. 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.

18.5.1.7. 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

18.5.1.8. PHY Interface Unit

The USB OTG controller supports synchronous 8-bit SDR data transmission to a ULPI PHY.

18.5.1.9. 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 functionlaity 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.

18.5.2. Local Memory Buffer

The USB OTG 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 (192 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.

18.5.3. Clocks

Table 181. 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
`l4_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

18.5.3.1. 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.

18.5.4. Resets

The USB OTG controller can be reset either through the hardware reset input or through software.

18.5.4.1. 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 `l4_mp_clk` cycles. There is no maximum assertion time.

18.5.4.2. 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 `l4_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.

18.5.4.3. 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.

18.5.4.4. 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.

After the Cortex-A53 MPCore 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 section: *Reset Signals and Registers* in the *Reset Manager* chapter.

You should ensure that both the USB ECC RAM and the USB Module resets are deasserted before beginning transactions. Program the `usb*ocp` bits and the `usb*` bits in the `per0modrst` register of the Reset Manager to deassert reset in the USB ECC RAM and the USB module, respectively.

18.5.5. Interrupts

Table 182. 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

18.6. 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

18.6.1. 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.

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

USB 2.0 OTG Controller Address Map and Register Definitions on page 396

18.6.2. Host Operation

18.6.2.1. 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.

   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.

18.6.2.2. 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.

18.6.3. Device Operation

18.6.3.1. 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.

18.6.3.2. 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 will access 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.

18.6.3.2.1. 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.

18.6.3.2.2. 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.

18.6.3.2.3. 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.

18.7. USB 2.0 OTG Controller Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

19. 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). † (44)

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

19.1. 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 33.33Mbps
- 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 frame size of 4 to 16 or 32 bits

(44) 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.

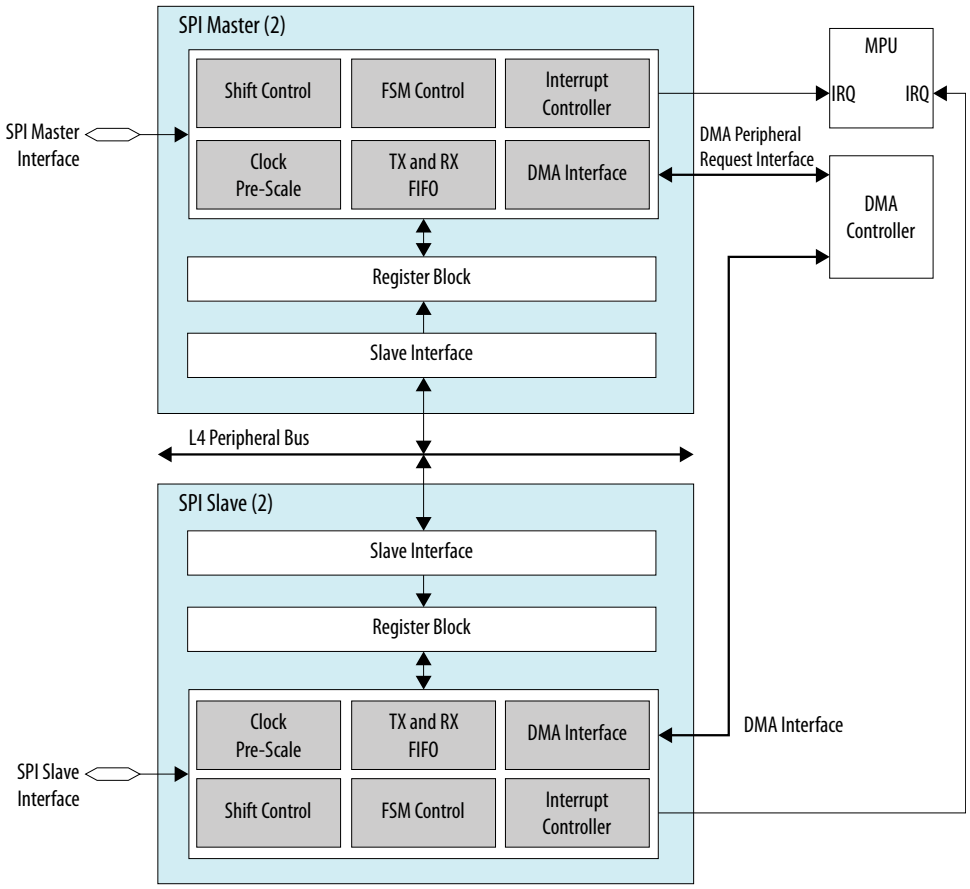
†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

19.2. SPI Block Diagram and System Integration

The SPI supports data bus widths of 32 bits. †

19.2.1. SPI Block Diagram

Figure 83. 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

19.3. 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.

19.3.1. 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.

Table 183. 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 184. 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

19.3.2. 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.

Table 185. 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 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

continued...

Signal Name	Signal Width	Direction	Description
			Allows fourth slave to be connected to this master
spim_sclk_out	1	Out	Serial clock output

Table 186. 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

19.4. Functional Description of the SPI Controller

19.4.1. 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 †

19.4.2. 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: †

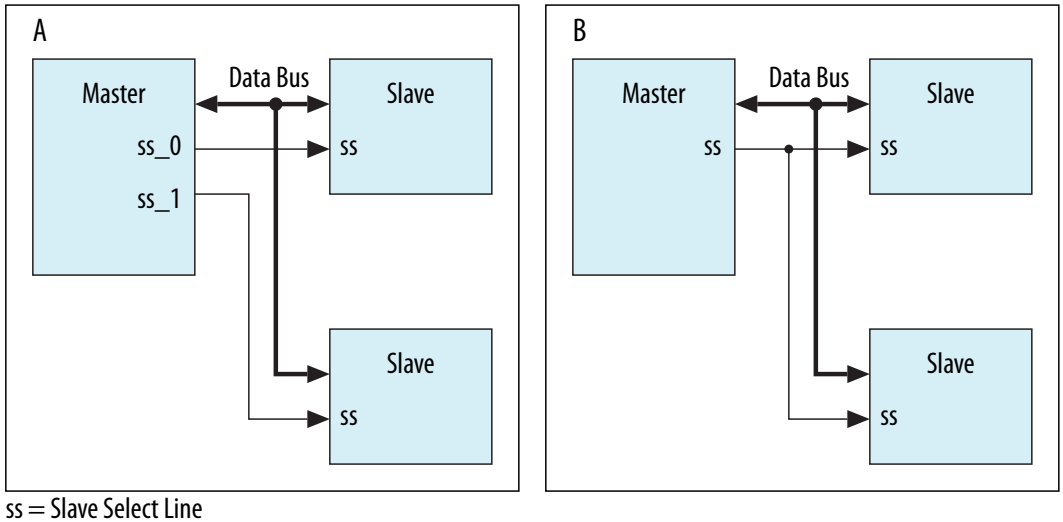
- Motorola SPI protocol – A four-wire, full-duplex serial protocol from Motorola. The slave select line is held high when the SPI controller is idle or disabled. For more information, refer to "Motorola SPI Protocol". †
- Texas Instruments Serial Protocol (SSP) – A four-wire, full-duplex serial protocol. 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 84 on page 401. †

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. †

The main program in the software domain controls selection of the target slave device; this architecture is illustrated in part B of the Figure 84 on page 401 figure below. Software would control which slave is to respond to the serial transfer request from the master device. †

Figure 84. Hardware/Software Slave Selection



Related Information

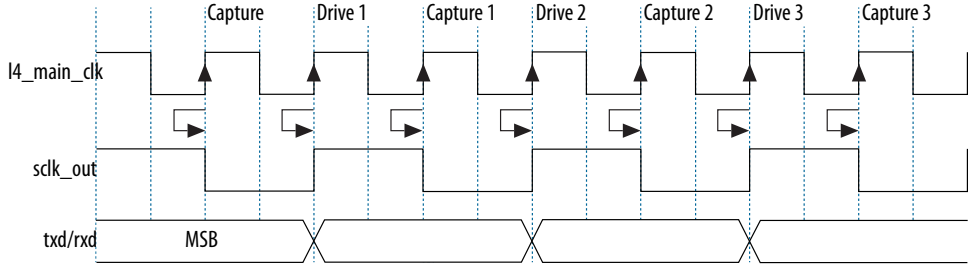
- Motorola SPI Protocol on page 412
- Texas Instruments Synchronous Serial Protocol (SSP) on page 414
- National Semiconductor Microwire Protocol on page 414

19.4.2.1. Serial Bit-Rate Clocks

19.4.2.1.1. 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 85. 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 187. SPI Master Peripheral Clock

• SPI Master Peripheral Clock
Fl4_main_clk >= 2 x (maximum F_{sclk_out}) †

19.4.2.1.2. 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 12 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: $F_{l4\_main\_clk}$ >= 12 multiply (maximum F_{sclk_in}) †

19.4.2.2. Transmit and Receive FIFO Buffers

There are two 16 or 32-bit FIFO buffers, a transmit FIFO buffer and a receive FIFO buffer, with a depth of 256. Data frames that are less than 16 or 32 bits in size must be right-justified when written into the transmit FIFO buffer. The data frame length depends on the maximum transfer size. 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 (`SPIENR=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 161
For more information, refer to the *Reset Manager* chapter.

19.4.2.3. 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). †

19.4.3. 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. †

19.4.3.1. 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. †

19.4.3.2. 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. †

19.4.3.3. 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. †

19.4.3.4. EEPROM Read

Note: This transfer mode is only valid for serial masters. †

When TMOD = 3, the transmit data is used to transmit an opcode and/or an address 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. †

19.4.4. 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 398

19.4.4.1. 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.

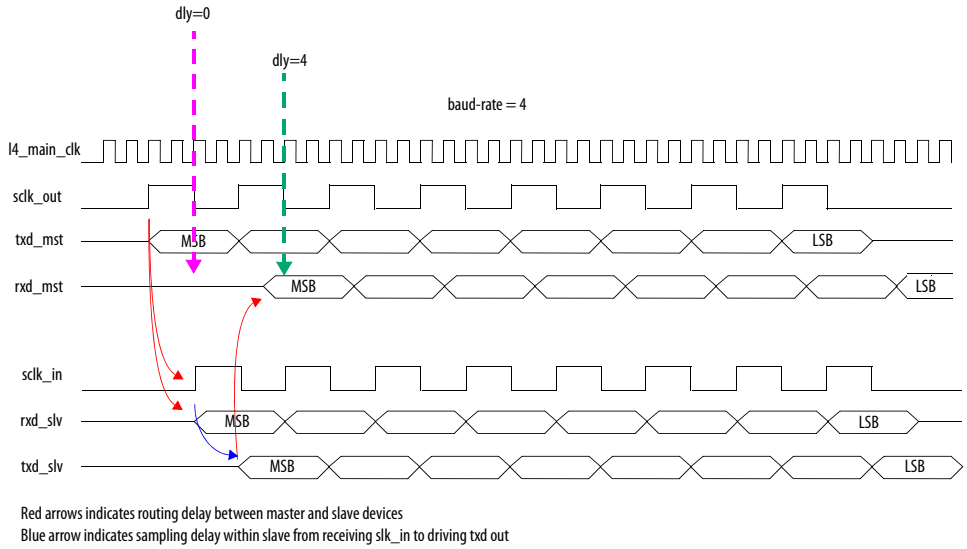
Without the RXD sample delay, you must increase the baud rate for the transfer in order to ensure that the setup times on the `rxd` signal are within range. This reduces the frequency of the serial interface.

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 86. Effects of Round Trip Routing Delays on sclk_out Signal



19.4.4.2. 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. †

19.4.4.3. 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 and/or address 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 412
- Texas Instruments Synchronous Serial Protocol (SSP) on page 414
- SPI Controller Address Map and Register Definitions on page 428

19.4.4.4. 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 414

19.4.5. SPI Slave

The SPI slave handles serial communication with transfer initiated and controlled by serial master peripheral devices.

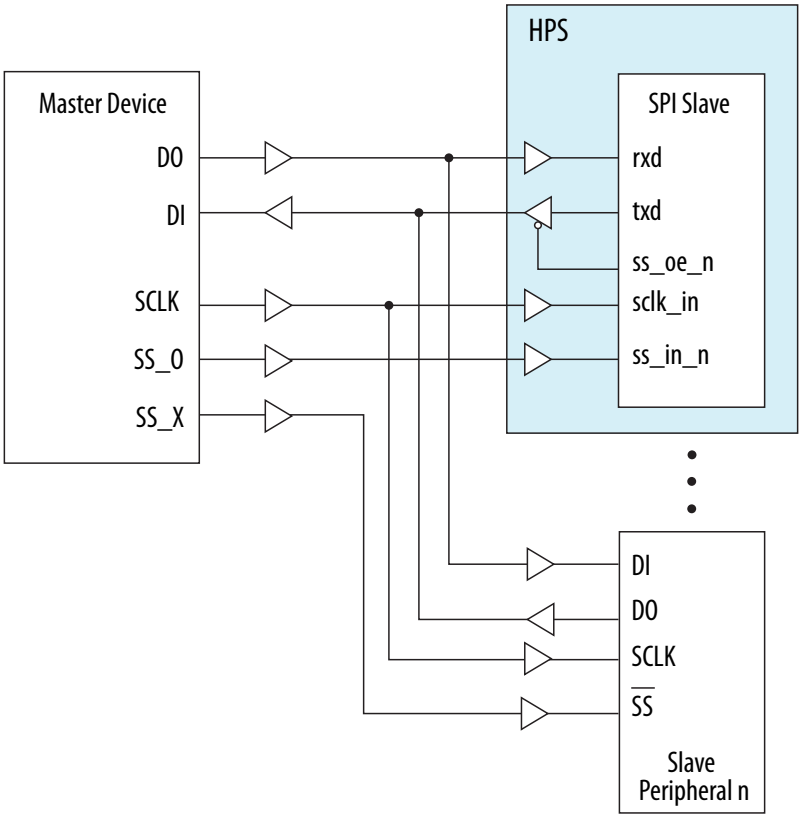
- sclk_in—serial clock to the SPI slave †
- ss_in_n—slave select input to the SPI slave †
- ss_oe_n—output enable for the SPI master or slave †
- txd—transmit data line for the SPI master or slave †
- rxd—receive data line for the SPI master or slave †

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 87 on page 410 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 87. 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.

19.4.5.1. 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 412
- Texas Instruments Synchronous Serial Protocol (SSP) on page 414

19.4.5.2. 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 414

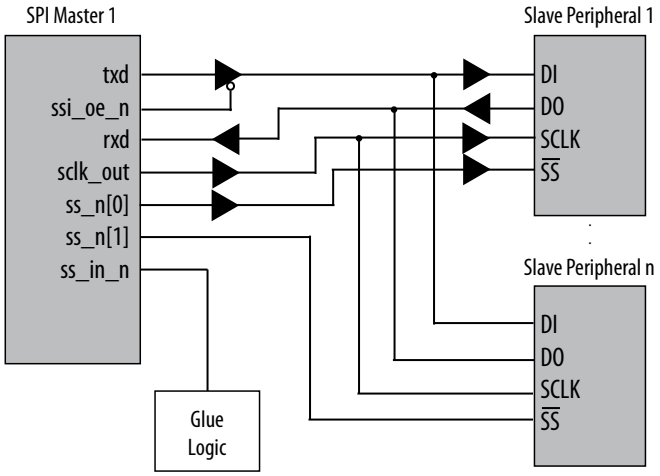
19.4.5.3. Glue Logic for Master Port ss_in_n

When configured as a master, the SPI has an input, `ssi_in_n`. 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 88. SPI Configured as Master Device



19.4.6. Partner Connection Interfaces

The SPI can connect to any serial-master or serial-slave peripheral device using one of several interfaces.

19.4.6.1. 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. †

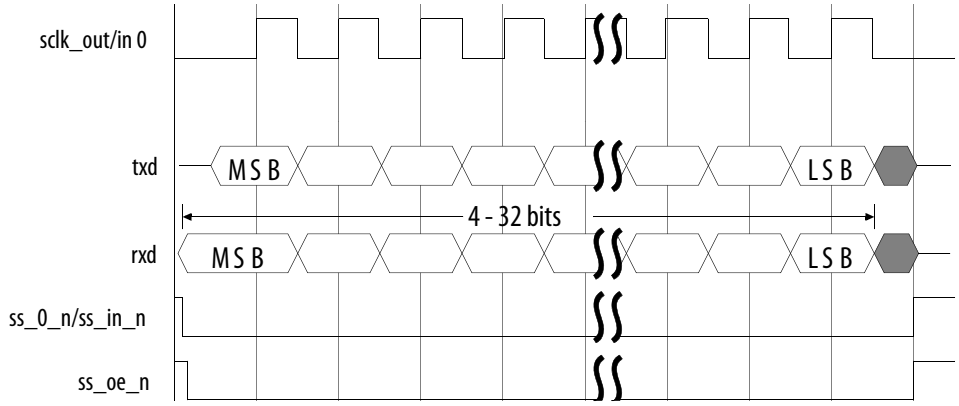
When the configuration parameter (SCPH = 0), data transmission begins on the falling edge of the slave select signal. The first data bit is captured by the master and slave peripherals on the first edge of the serial clock; therefore, valid data must be present on the `txd` and `rxd` lines prior to the 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 89. 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), opcode and/or EEPROM address 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.

Related Information

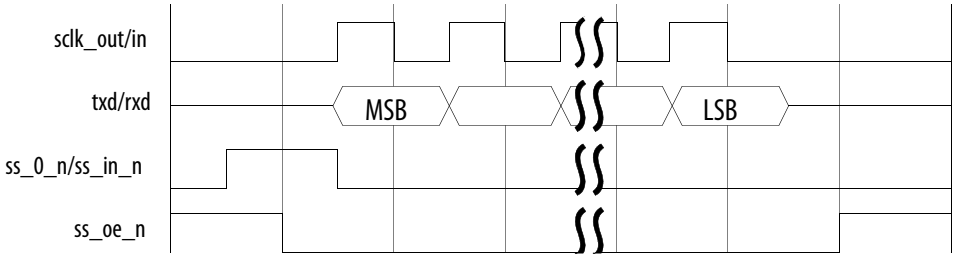
Transfer Modes on page 404

19.4.6.2. 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 or 32 bits depending on the maximum transfer size.

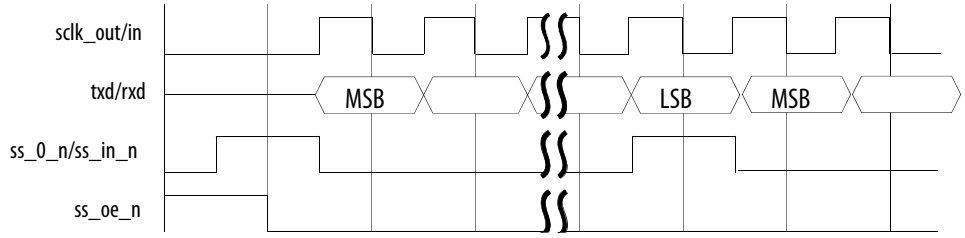
Note: The slave select signal (`ss_0_n`) takes effect only when used as slave SPI. For master SPI, the data transmission begins as soon as the output enable signal is deasserted.

Figure 90. 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 91. SSP Serial Format Continuous Transfer



19.4.6.3. 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 or 32 bits in length. The data frame length depends on the maximum transfer size. 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- or 32-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 92. Single SPI Serial Master Microwire Serial Transfer (MDD=0)

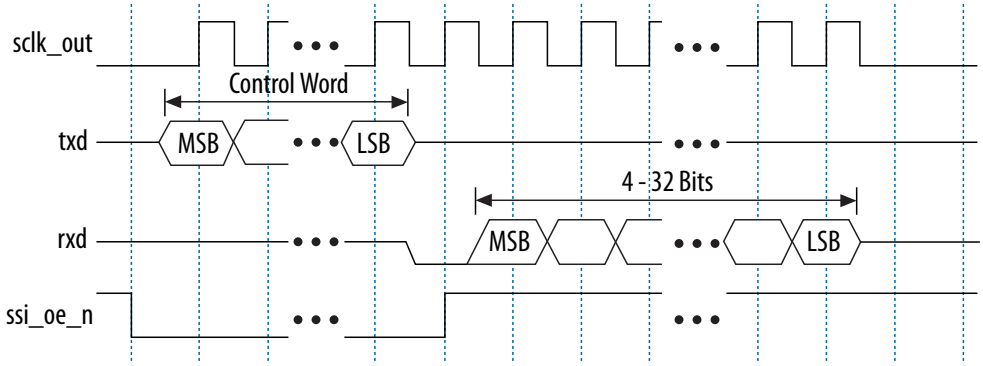
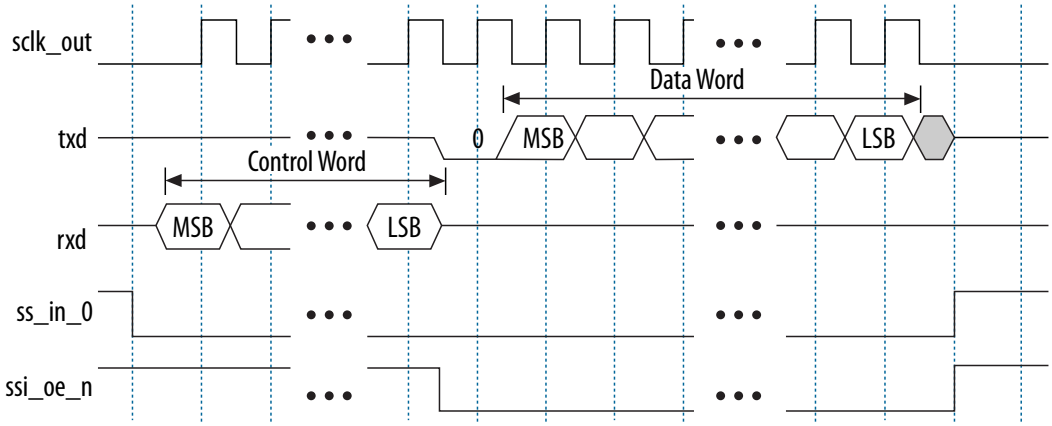


Figure 93. Single SPI Slave Microwire Serial Transfer (MDD=1)



19.4.7. 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. †

19.4.8. 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.

19.4.8.1. 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. †

19.4.8.2. Data Register Access

The data register (DR) within the SPI controller is 16 or 32 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.

19.4.9. Clocks and Resets

The SPI controller uses the clock and reset signals shown in the following table.

Table 188. 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

19.4.9.1. 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.

19.4.9.2. 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.

After the Cortex-A53 MPCore boots, it can deassert the reset signal by clearing the appropriate bits in the reset manager's corresponding reset register. For more information about reset registers, refer to the "Reset Manager" section.

Related Information

Reset Manager on page 161

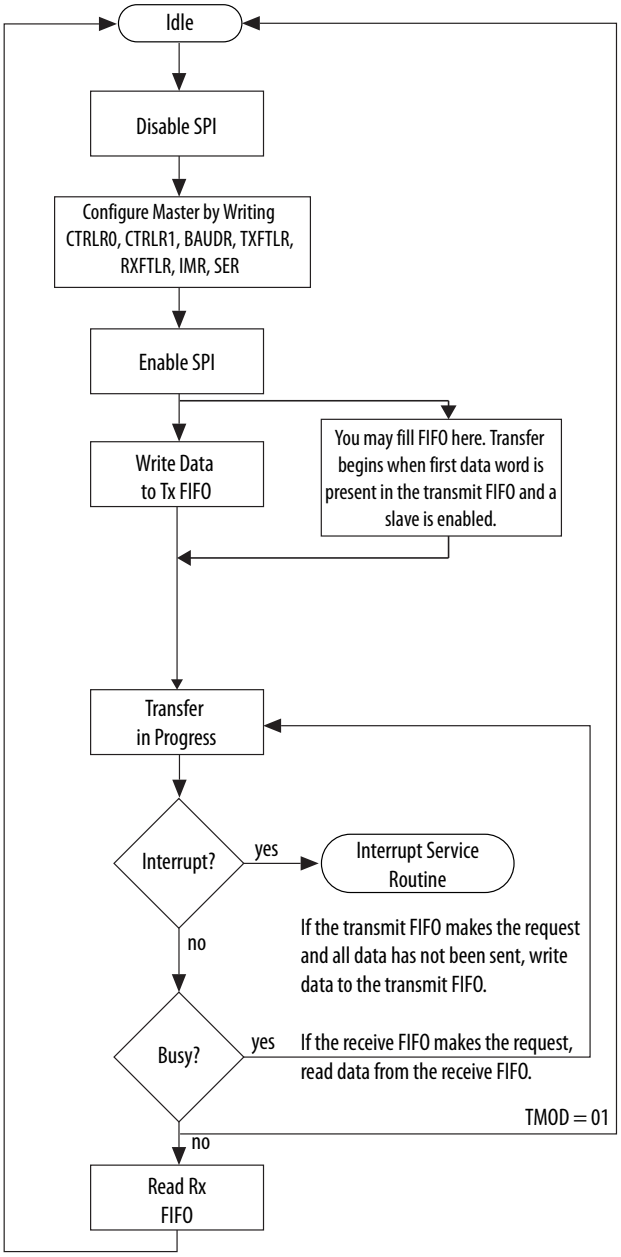
19.5. SPI Programming Model

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

19.5.1. Master SPI and SSP Serial Transfers

Figure 94. Master SPI or SSP Serial Transfer Software Flow



To complete an SPI or SSP serial transfer from the SPI master, follow these steps:

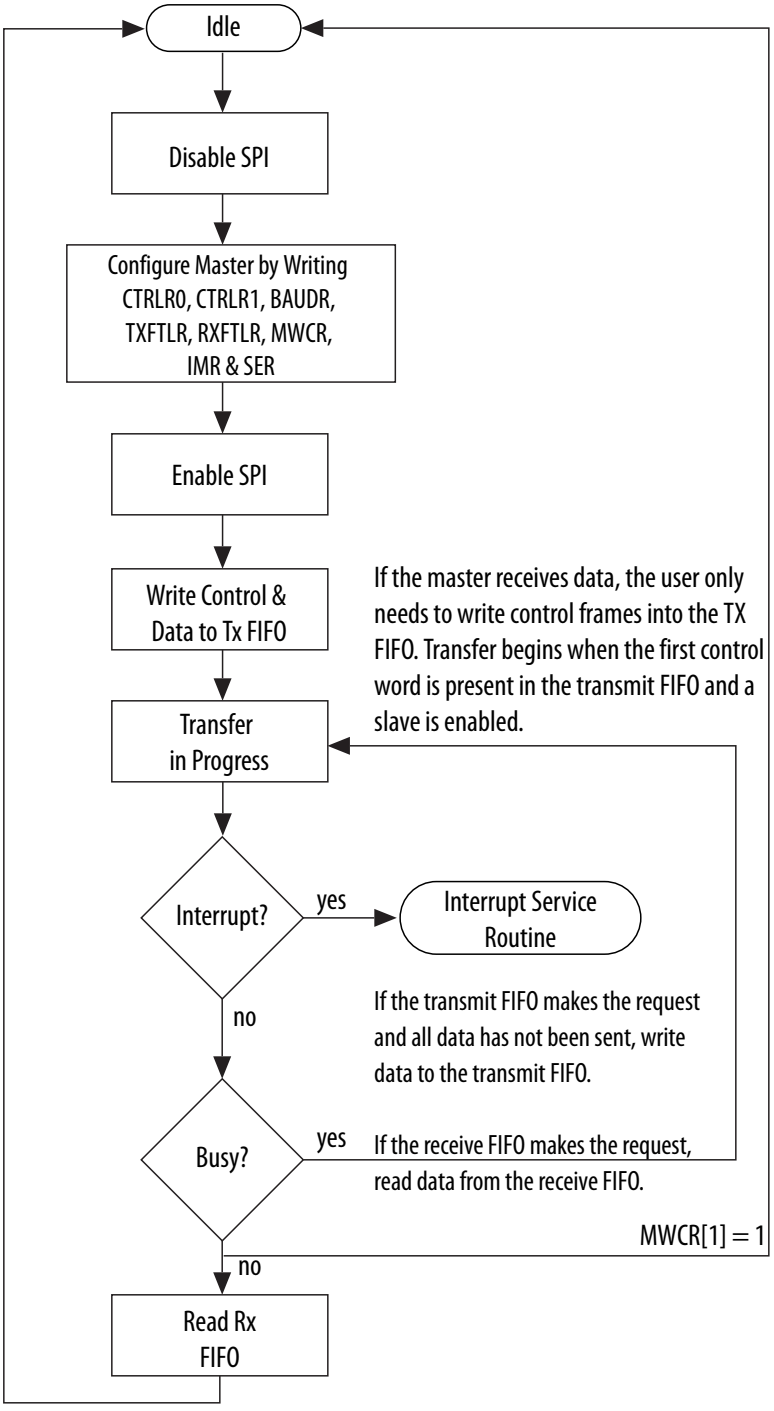
1. If the SPI master is enabled, disable it by writing 0 to the SPI Enable register (SPIENR).
2. Set up the SPI master control registers for the transfer. You can set these registers in any order.

- Write Control Register 0 (CTRLR0). 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 SPIENR 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 SPIENR.

19.5.2. Master Microwire Serial Transfers

Figure 95. 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 SPIENR.
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.

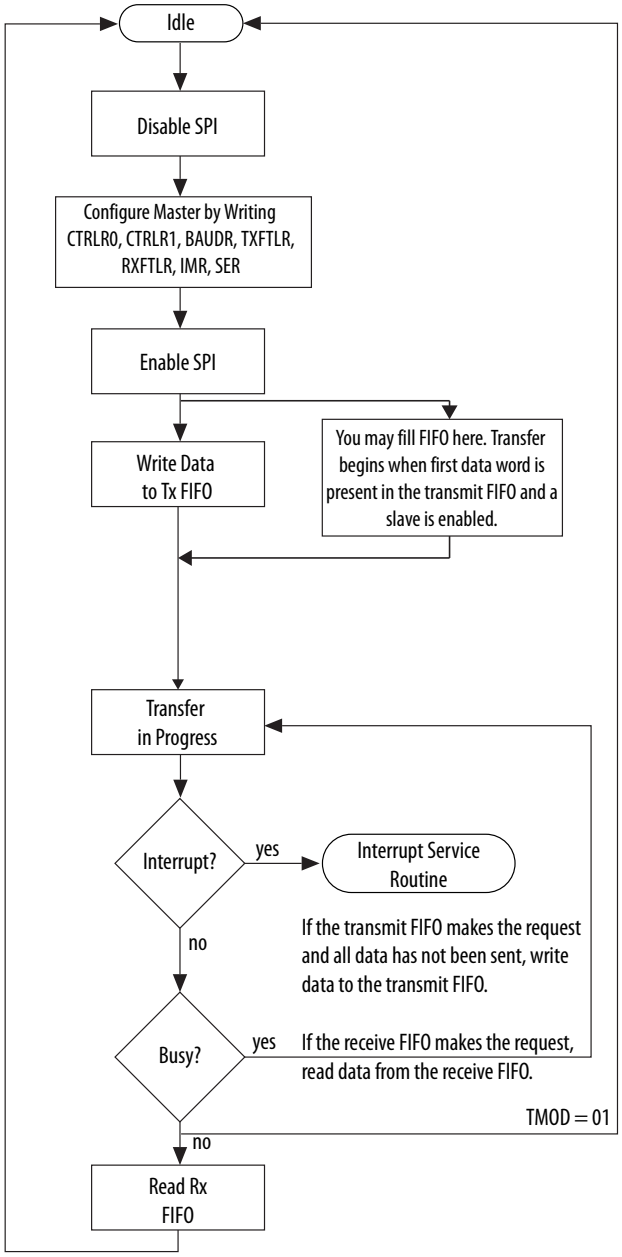
   You can write the SER register to enable the target slave for selection. If a slave is enabled here, the transfer begins as soon as one valid data entry is present in the transmit FIFO buffer. If no slaves are enabled prior to writing to the DR register, the transfer does not begin until a slave is enabled.
3. Enable the SPI master by writing 1 to the SPIENR 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 SPIENR.

Related Information

SPI Controller Address Map and Register Definitions on page 428

19.5.3. Slave SPI and SSP Serial Transfers

Figure 96. 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 SPIENR.
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 SPIENR 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 SPIENR.

19.5.4. 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.

19.5.5. 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.

19.5.5.1. Example: Slave Selection Software Flow for SPI Master

1. If the SPI master is enabled, disable it by writing 0 to SPIENR.
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 SPIENR register bit 0 to 1 to enable SPI master.

19.5.5.2. Example: Slave Selection Software Flow for SPI Slave

1. If the SPI slave is enabled, disable it by writing 0 to SPIENR.
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 SPIENR 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 (SPIENR = 0) and therefore will 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.

19.5.6. 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.†

19.5.6.1. 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 will run out of data (underflow). To prevent this condition, you must set the watermark level correctly.†

19.5.6.2. 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. †

19.5.6.2.1. Example 1: Transmit FIFO Watermark Level = 64

Consider the example where the assumption is made: †

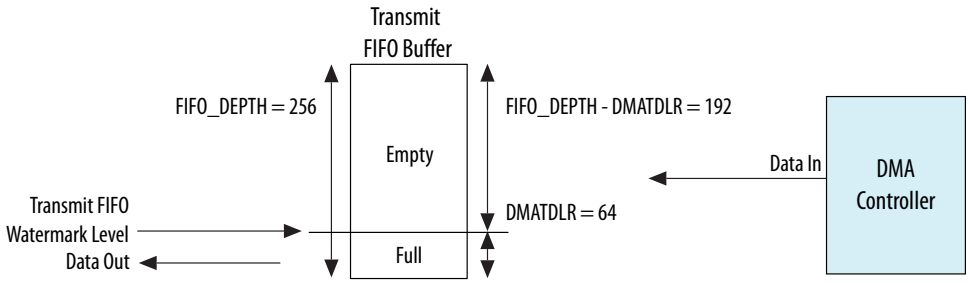
DMA burst length = FIFO_DEPTH - DMATDLR

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 97. 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 will need 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.

19.5.6.2.2. Example 2: Transmit FIFO Watermark Level = 192

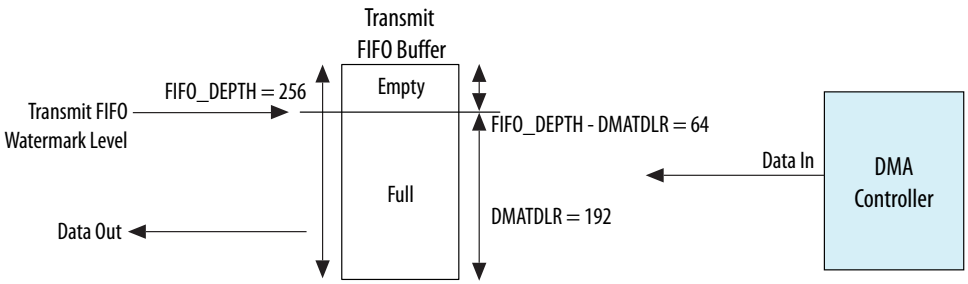
Consider the example where the assumption is made: †

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 98. 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.

19.5.6.3. 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 will 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. †

19.5.6.4. 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 DMATDLR + 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 will fill with data (overflow). To prevent this condition, the user must set the watermark level correctly. †

19.5.6.5. 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. †

19.5.6.6. 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: †

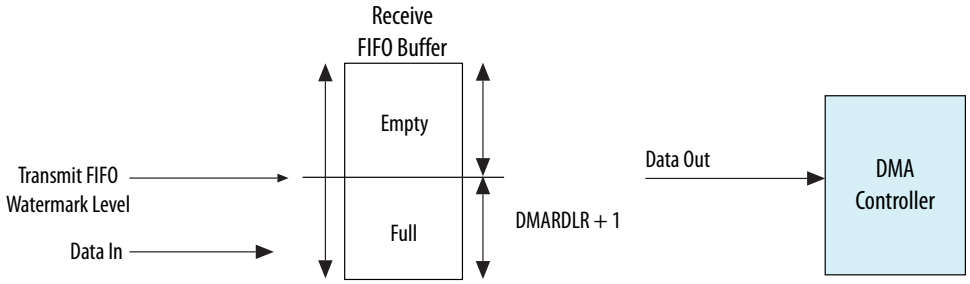
DMA burst length = DMATDLR + 1

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 will 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 99. Receive FIFO Buffer



19.6. SPI Controller Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

20. 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 SynopsysDesignWare 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. † (45)

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

20.1. 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 †

(45) 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.

- 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.

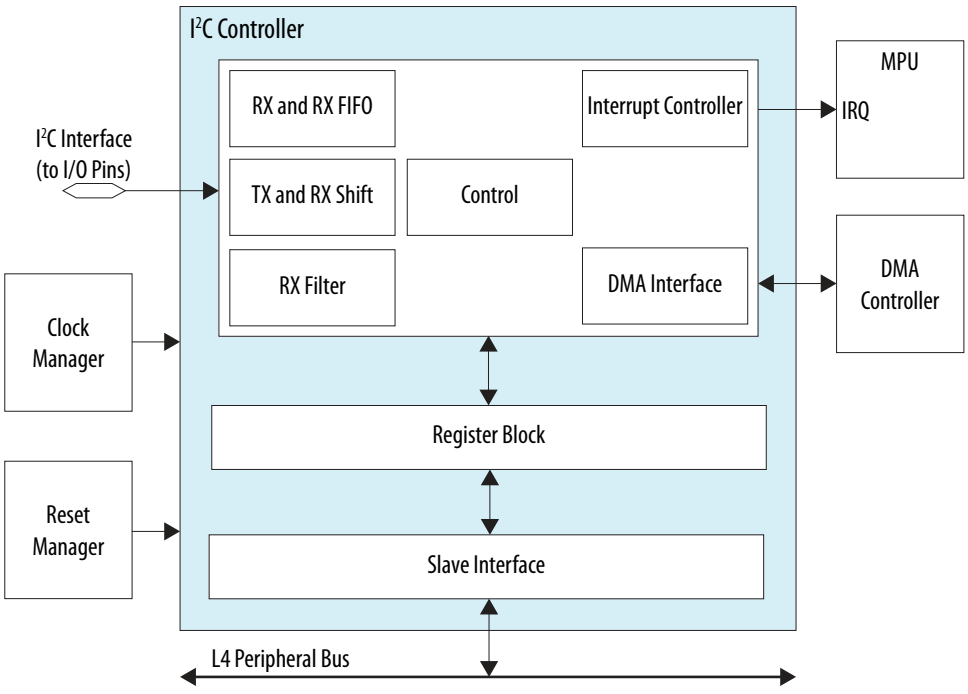
- `I2C0`
- `I2C1`

20.2. 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 100. 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. †

20.3. 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.

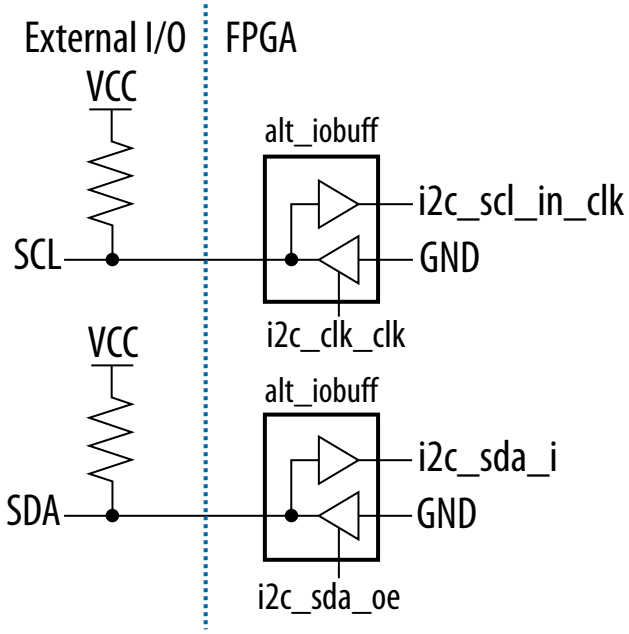
Table 189. I2C Controller Interface HPS I/O Pins

Pin Name	Signal Width	Direction	Description
SCL	1 bit	Bidirectional	Serial clock
SDA	1 bit	Bidirectional	Serial data

Table 190. HPS I2C Signals for FPGA Routing

Signal Name	Signal Width	Direction	Description
i2c<#>_scl_in_clk	1 bit	Input	Incoming I2C clock source. This is the input SCL signal
i2c<#>_clk_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_i	1 bit	Input	Incoming I2C data. This is the input SDA signal.
i2c<#>_sda_oe	1 bit	Output	Outgoing I2C data enable. Output SDA signal. This signal is logically inverted and is synchronous to the HPS peripheral clock.

Figure 101. 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<#>_sda_oe` and `i2c<#>_clk_clk` are asserted, external signal will be driven to ground.

20.4. Functional Description of the I²C Controller

20.4.1. 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. †

20.4.2. Behavior

You can control the I2C controller via software to be in either mode:

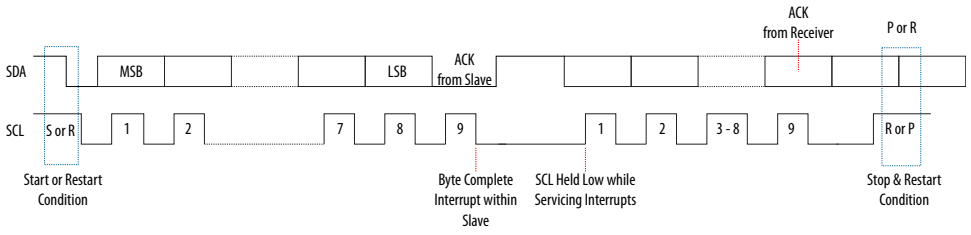
- An I2C master only, communicating with other I2C slaves.
- An I2C slave only, communicating with one or more I2C masters.

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 102. 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. †

20.4.2.1. 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. †

20.4.2.2. 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 will be 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 will be generated.†

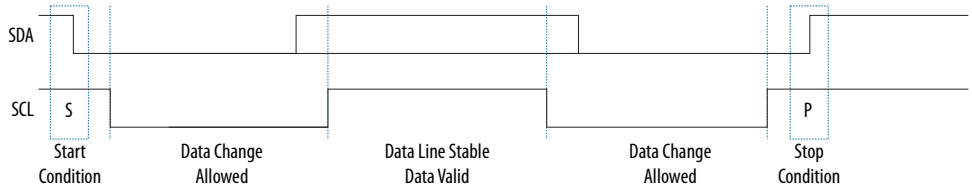
20.4.3. Protocol Details

20.4.3.1. 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 103. Timing Diagram for START and STOP Conditions



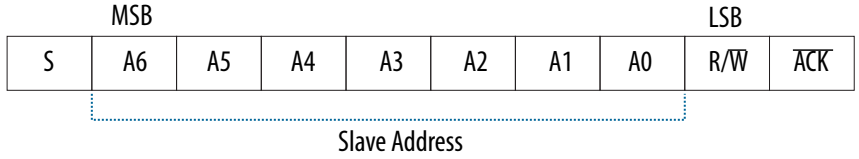
The signal transitions for the START or STOP condition, as shown in the figure, reflect those observed at the output signals of the master driving the I²C bus. Care should be taken when observing the SDA or SCL signals at the input signals of the slave(s), because unequal line delays may result in an incorrect SDA or SCL timing relationship. †

20.4.3.2. Addressing Slave Protocol

20.4.3.2.1. 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 104. 7- Bit Address Format

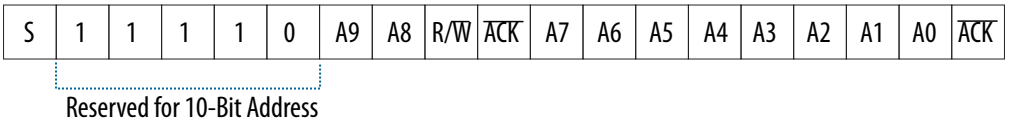


S: Start Condition
R/W: Read/Write Pulse
ACK: Acknowledge (Sent by Slave)

20.4.3.2.2. 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 105. 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 191. 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

continued...

Slave Address	R/W Bit	Description
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 438

20.4.3.3. Transmitting and Receiving Protocol

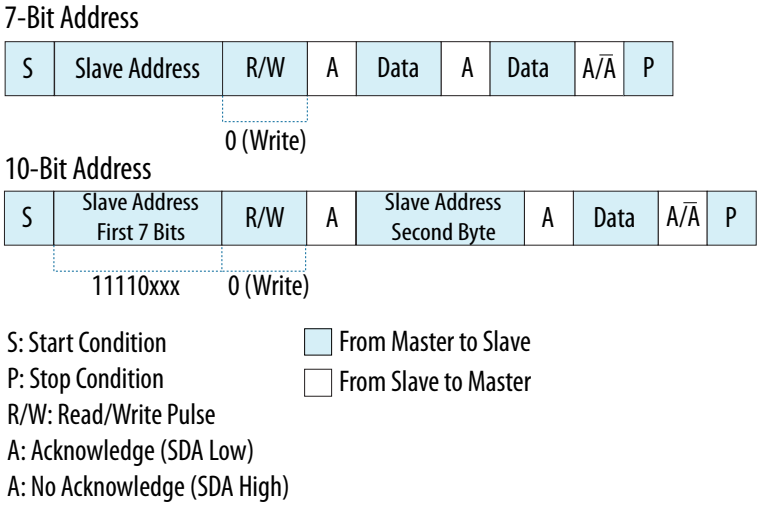
The master can initiate data transmission and reception to or from the bus, acting as either a master-transmitter or master-receiver. A slave responds to requests from the master to either transmit data or receive data to or from the bus, acting as either a slave-transmitter or slave-receiver, respectively. †

20.4.3.3.1. 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 106. Master-Transmitter Protocol †



20.4.3.3.2. Master-Receiver and Slave-Transmitter

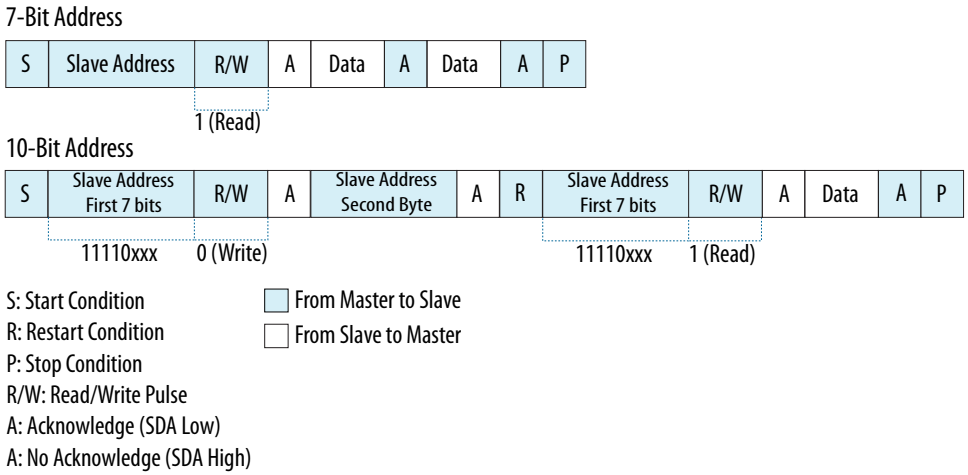
If the master is receiving data as shown in the following figure, then the master responds to the slave-transmitter with an ACK pulse after a byte of data has been received, except for the last byte. This is the way the master-receiver notifies the

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 107. Master-Receiver Protocol †



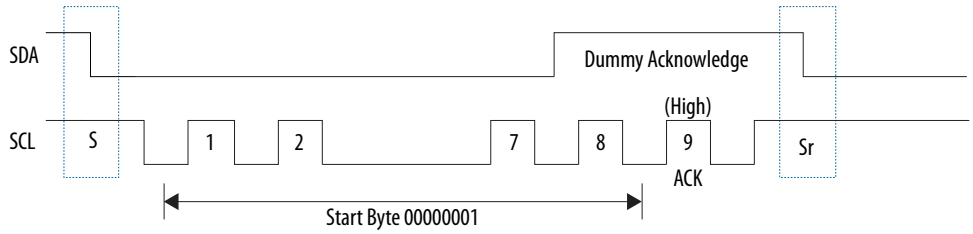
Related Information

Combined Formats on page 435

20.4.3.4. 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 108. 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. †

20.4.4. Multiple Master Arbitration

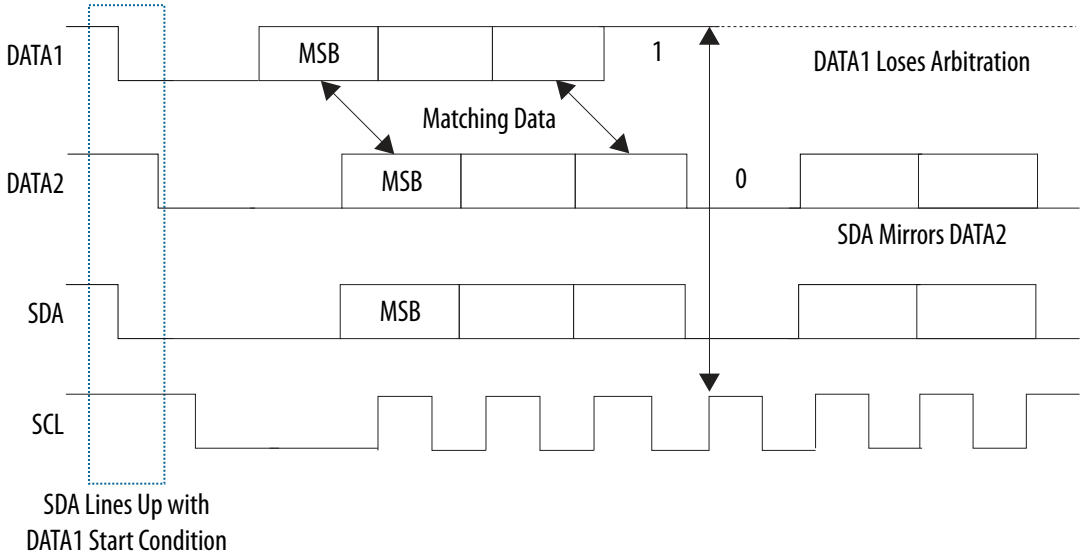
The I²C controller bus protocol allows multiple masters to reside on the same bus. If there are two masters on the same I²C-bus, there is an arbitration procedure if both try to take control of the bus at the same time by simultaneously generating a START condition. Once a master (for example, a microcontroller) 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 109. 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. †

20.4.4.1. 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 110. Multiple Master Clock Synchronization †

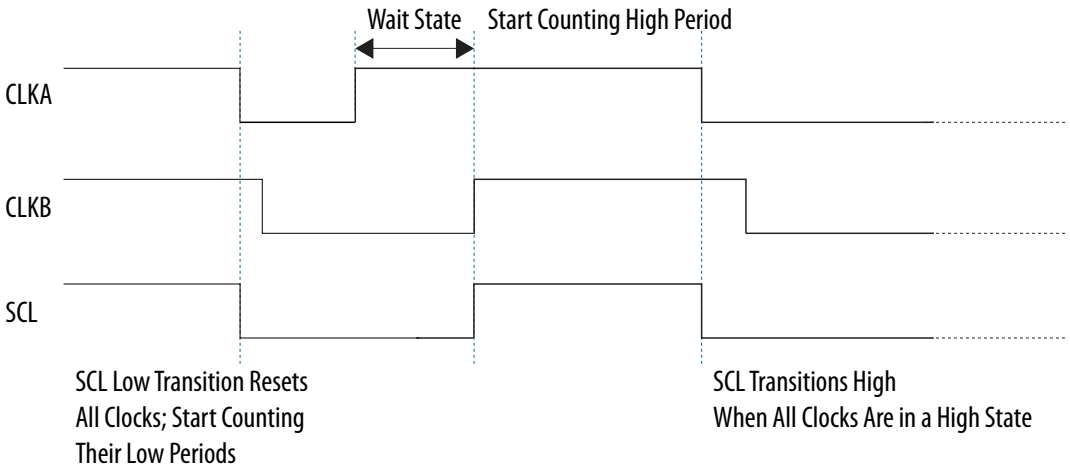
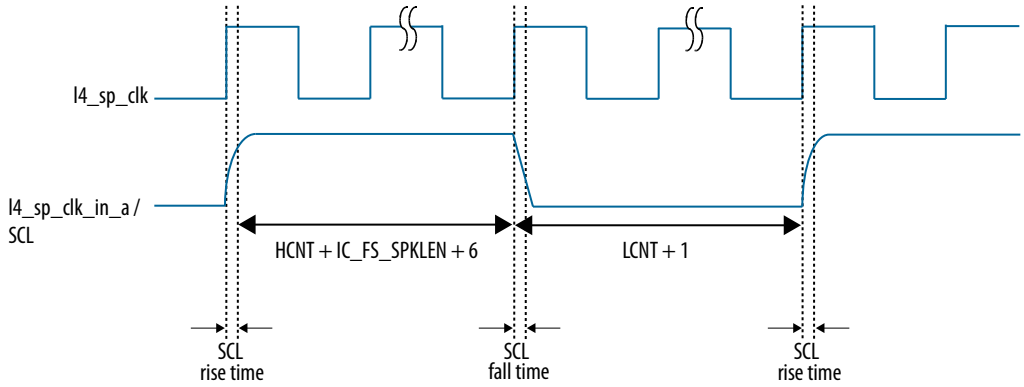


Figure 111. Impact of SCL Rise Time and Fall Time on Generated SCL



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
```

20.4.5. 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. †

20.4.5.1. 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 will be filtered out by the spike suppression logic.†

20.4.5.1.1. 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 192. l4_sp_clk Clock Pulse 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)

Example 1. 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 †
```

20.4.6. 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.

20.4.7. 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 116
For details about the DMA burst length microcode setup, refer to the *DMA controller* chapter.

20.4.8. 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 149
For more information, refer to *Clock Manager* chapter.

20.4.9. 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 161
For more information, refer to *Reset Manager* chapter.

20.4.9.1. 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.

After the Cortex-A53 MPCore 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 section: *Reset Signals and Registers* in the *Reset Manager* chapter.

20.5. 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. †

20.5.1. Slave Mode Operation

20.5.1.1. 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.†

20.5.1.2. 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 and/or TX_ABRT interrupts have been masked, then clearing of the IC_RAW_INTR_STAT register will have 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. †

20.5.1.3. 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. †

20.5.1.4. 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.†

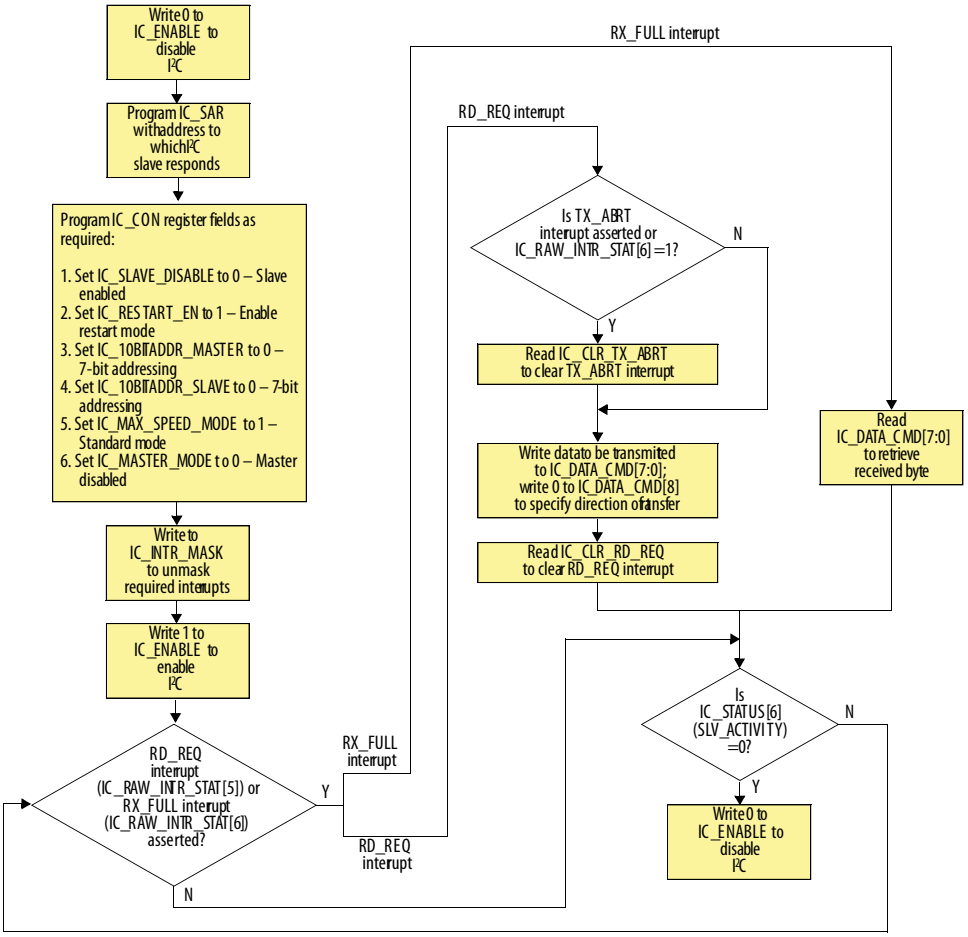
The RD_REQ interrupt is raised upon a read request, and like interrupts, must be cleared when exiting the interrupt service handling routine (ISR). The ISR allows you to either write 1 byte or more than 1 byte into the TX FIFO. During the transmission of these bytes to the master, if the master acknowledges 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. †

20.5.1.5. Slave Programming Model



20.5.2. Master Mode Operation

20.5.2.1. 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 will complete 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. †

20.5.2.2. Dynamic IC_TAR or IC_10BITADDR_MASTER Update

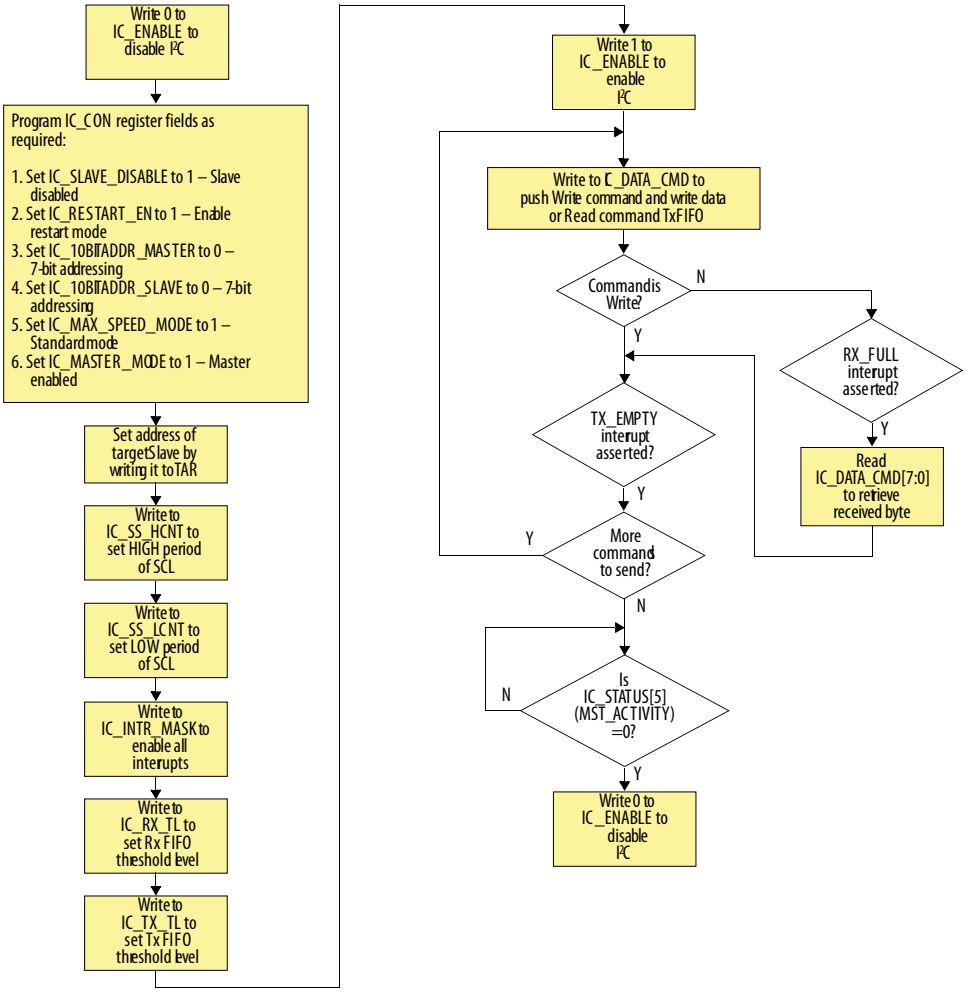
The I²C controller supports dynamic updating of the IC_TAR (bits 9:0) and IC_10BITADDR_MASTER (bit 12) bit fields of the IC_TAR register. You can dynamically write to the IC_TAR register provided the following conditions are met: †

- 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) †

20.5.2.3. 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.†

20.5.2.4. Master Programming Model



20.5.3. 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. †

20.5.4. 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.†

20.5.5. 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 116
For details about the DMA burst length microcode setup, refer to the *DMA controller* chapter.

20.5.5.1. 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 will run 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 116
For details about the DMA burst length microcode setup, refer to the *DMA controller* chapter.

20.5.5.2. Transmit Watermark Level

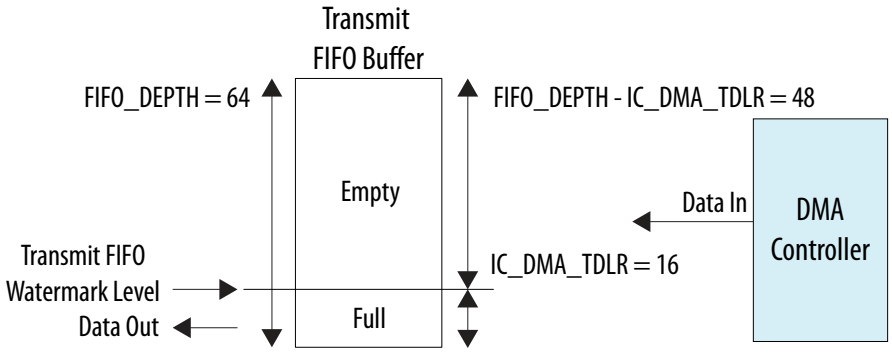
Consider the example where the assumption is made: †

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 112. Transmit FIFO Watermark Level = 16



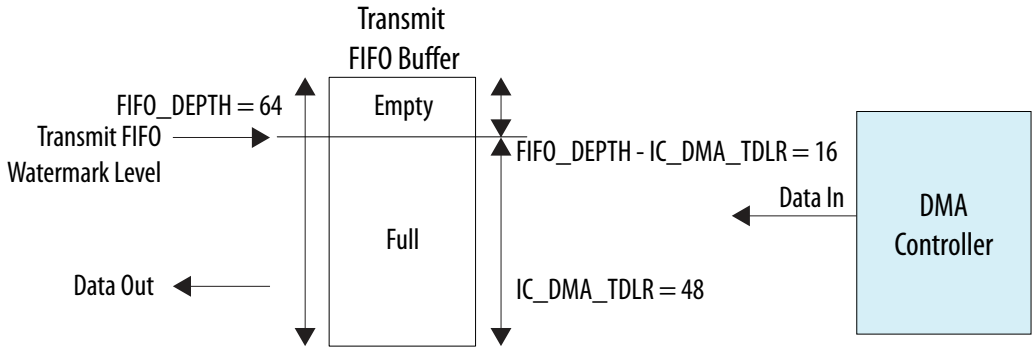
The number of burst transactions needed equals the block size divided by the number of data items per burst:

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 113. 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. †

20.5.5.3. 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 will 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. †

20.5.5.4. 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 will fill with data (overflow). To prevent this condition, the user must set the watermark level correctly. †

20.5.5.5. 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. †

20.5.5.6. 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: †

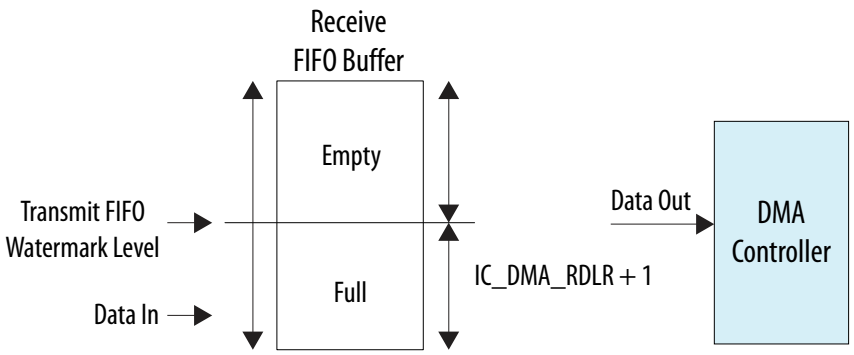
DMA burst length = IC_DMA_RDLR + 1

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 will 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 114. Receive FIFO Buffer



20.6. I²C Controller Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

21. 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 SynopsysDesignWare APB Universal Asynchronous Receiver/Transmitter (DW_apb_uart) peripheral. (46)

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

21.1. 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 †

(46) 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.

- 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

21.2. UART Controller Block Diagram and System Integration

Figure 115. UART Block Diagram

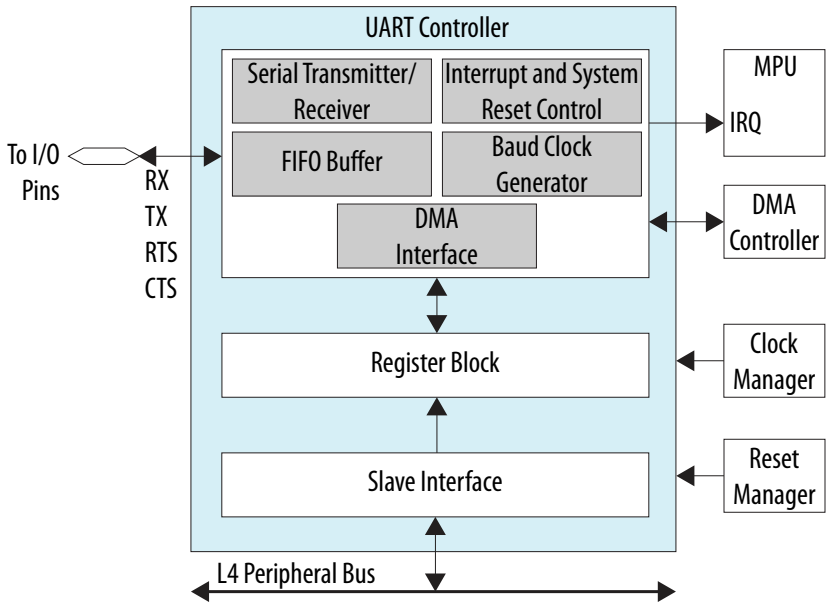


Table 193. UART Controller Block Descriptions

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 116
For more information, refer to the DMA Controller chapter.

21.3. UART Controller Signal Description

21.3.1. HPS I/O Pins

Table 194. 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

21.3.2. FPGA Routing

Table 195. 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

21.4. 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. †

21.4.1. 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. †

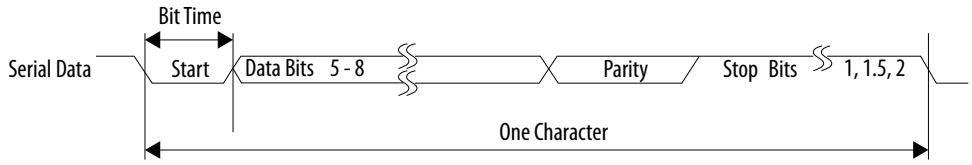
When FIFO buffer access mode is enabled, you can write data to the transmit FIFO buffer as normal; however, no serial transmission occurs in this mode and no data leaves the FIFO buffer. You can read back the data that is written to the transmit FIFO buffer with the transmit FIFO read (TFR) register. 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. †

21.4.2. 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 116. 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.†

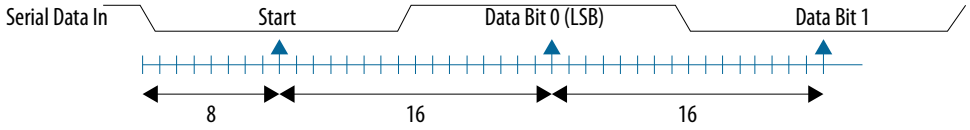
The Control Register is used to control the serial character characteristics. The individual bits of the data word are sent after the start bit, starting with the least-significant bit (LSB). These are followed by the optional parity bit, followed by the stop bit(s), which can be 1, 1.5 or 2.†

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 117. 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).†

21.4.3. 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). †

21.4.3.1. 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.

21.4.3.2. 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)

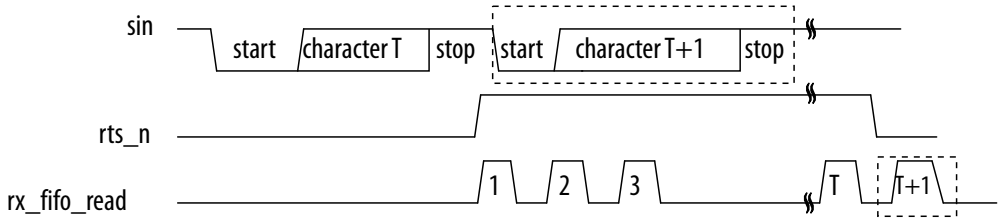
With automatic RTS enabled, the rts_n output pin is forced inactive (high) when the FIFO is almost full; where "almost full" refers to two available slots in the FIFO. When rts_n is connected to the cts_n input pin of another UART device, the other UART stops sending serial data until the receive FIFO 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 118. Automatic RTS Timing



21.4.3.3. 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. †

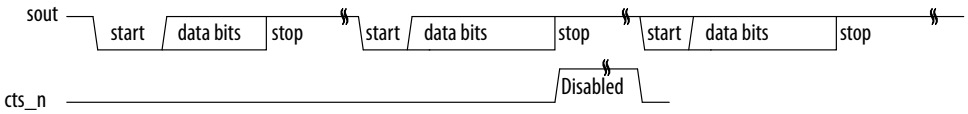
If the cts_n input is not deactivated before the middle of the last stop bit, another character is transmitted before the transmitter is disabled. While the transmitter is disabled, you can continue to write and even overflow to the transmit FIFO buffer. †

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 119. Automatic CTS Timing



21.4.4. 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 149
For more information, refer to the *Clock Manager* chapter.

21.4.5. 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.

21.4.5.1. 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.

After the Cortex-A53 MPCore 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 section: *Reset Signals and Registers* in the *Reset Manager* chapter.

21.4.6. Interrupts

The assertion of the UART interrupt output signal occurs when one of the following interrupt types are enabled and active: †

Table 196. 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

continued...

Interrupt Type	Priority	Source	Interrupt Reset Control
			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).

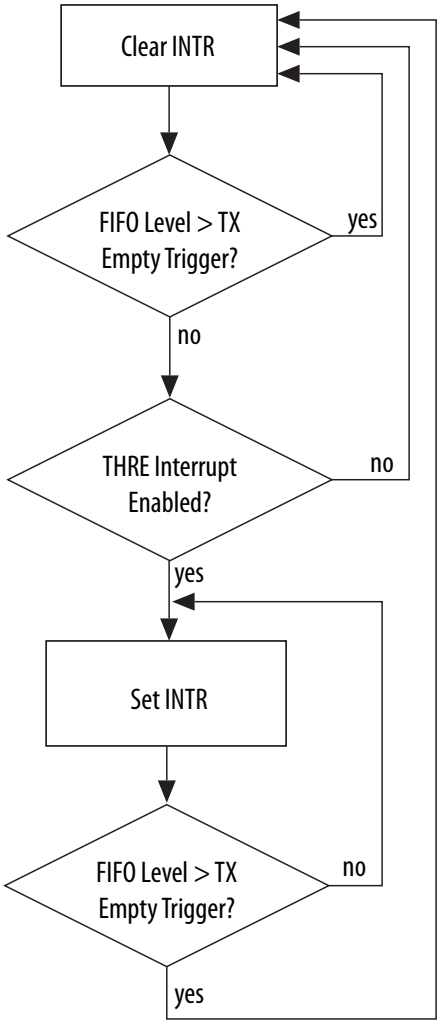
Note: Received Data Available" and "Character Timeout Indication" are enabled by a single bit in the IER_DLH register, because they have the same priority.

Once an interrupt is signaled, you can determine the interrupt source by reading the Interrupt Identity Register (IIR).

21.4.6.1. 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 120. Programmable THRE Interrupt

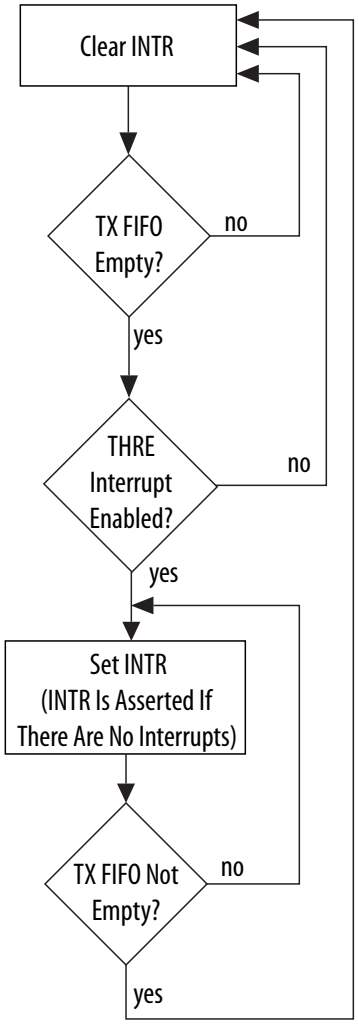


The threshold level is programmed into FCR.TET. The available empty thresholds are empty, 2, ¼, and ½. The optimum threshold value depends on the system's ability to begin a new transmission sequence in a timely manner. However, one of these thresholds should prove optimum in increasing system performance 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 121. Interrupt Generation without Programmable THRE Interrupt Mode



21.5. 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 116
For more information, refer to the DMA Controller chapter.

21.5.1. 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 will run 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 116
For more information, refer to the DMA Controller chapter.

21.5.2. Transmit Watermark Level

Consider the example where the following assumption is made: †

DMA burst length = FIFO_DEPTH - decoded watermark level of IIR_FCR.TET †

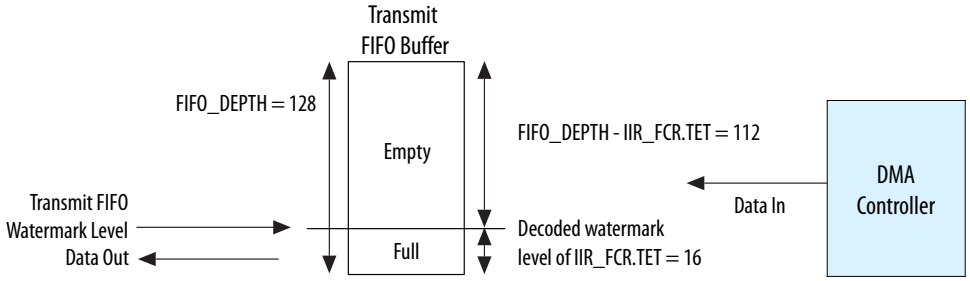
Here the number of data items to be transferred in a DMA burst is equal to the empty space in the transmit FIFO. Consider the following two different watermark level settings: †

21.5.2.1. 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 122. 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

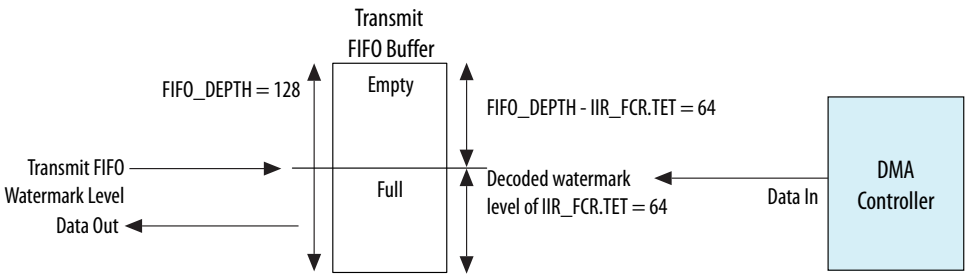
The number of burst transactions in the DMA block transfer is 4. But the watermark level, decoded level of IIR_FCR.TET, is quite low. Therefore, the probability of transmit underflow is high where the UART serial transmit line needs to transmit data, but there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the FIFO becomes empty.

21.5.2.2. 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 123. 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. †

21.5.3. 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 will 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. †

21.5.4. 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 will fill with data (overflow). To prevent this condition, the user must set the watermark level correctly. †

21.5.5. 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. †

21.5.6. 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: †

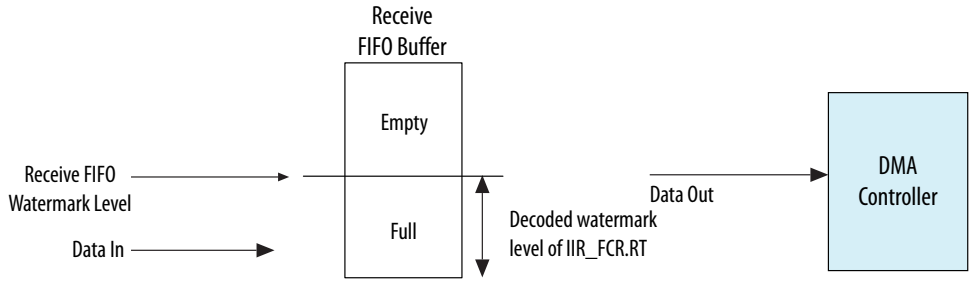
DMA burst length = decoded watermark level of IIR_FCR.RT + 1

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 will 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 124. Receive FIFO Buffer



21.6. UART Controller Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

22. General-Purpose I/O Interface

The hard processor system (HPS) provides two general-purpose I/O (GPIO) interface modules. The GPIO modules are instances of the SynopsysDesignWare APB General Purpose Programming I/O (DW_apb_gpio) peripheral.† (47)

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

22.1. Features of the GPIO Interface

The GPIO interface offers the following features:

- Supports digital debounce
- Configurable interrupt mode
- Supports up to 48 dedicated I/O pins

(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.

22.2. 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 125. Intel Agilex SoC GPIO

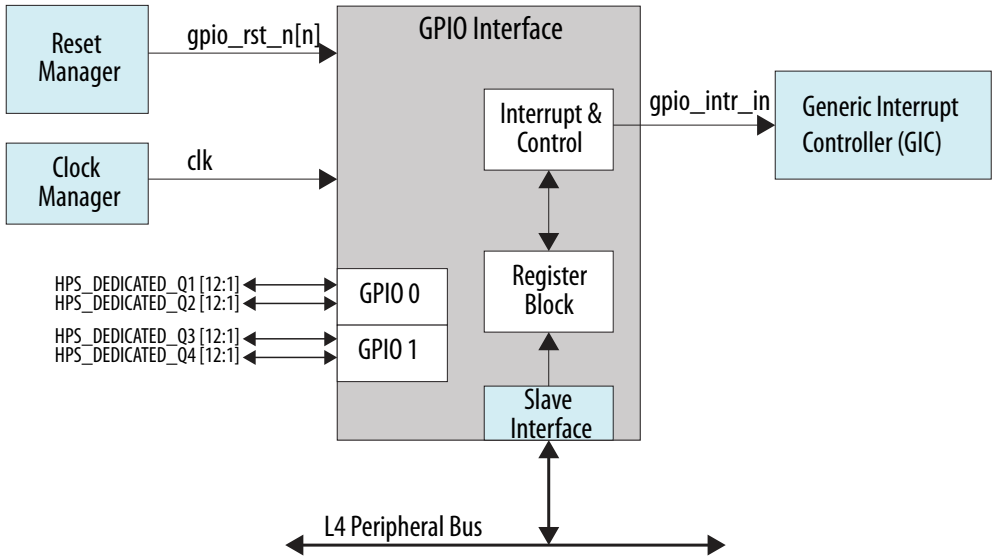


Table 197. GPIO Interface pin table

Pin Name	Mapped to GPIO Signal Name	Comments
HPS_DEDICATED_Q1 [12:1]	GPIO 0 [11:0]	Input / Output
HPS_DEDICATED_Q2 [12:1]	GPIO 0 [23:12]	Input / Output
HPS_DEDICATED_Q3 [12:1]	GPIO 1 [11:0]	Input / Output
HPS_DEDICATED_Q4 [12:1]	GPIO 1 [23:12]	Input / Output

22.3. Functional Description of the GPIO Interface

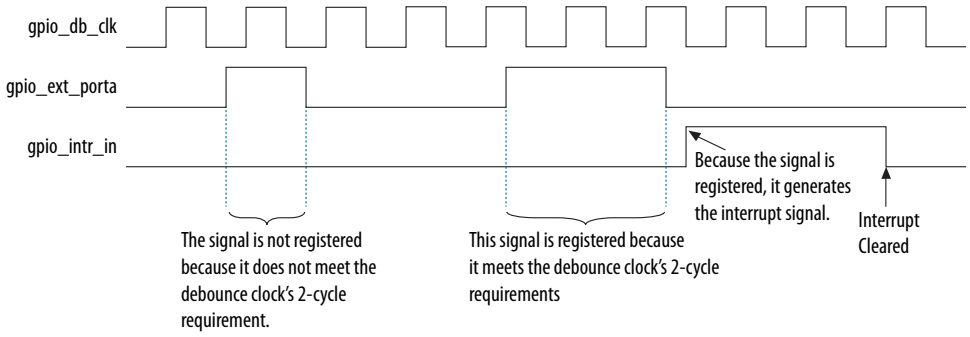
22.3.1. Debounce Operation

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 126. Debounce Timing With Asychronous Reset Flip-Flops



Note: Enabling the debounce circuitry increases interrupt latency by two clock cycles of the debounce clock.

22.3.2. Pin Directions

All GPIO pins can be configured to be either input or output signals.

22.3.3. 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.

After the Cortex-A53 MPCore 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 section: *Reset Signals and Registers* in the *Reset Manager* chapter.

22.4. 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. †

22.5. General-Purpose I/O Interface Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

23. Timers

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 2.09a) peripheral. (48)

Related Information

- Cortex-A53 MPCore Processor on page 32
  The Cortex-A53 MPCore processor provides additional timers. For more information about these timers, refer to the Cortex-A53 MPCore Processor chapter.
- Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
  For details on the document revision history of this chapter

23.1. Features of the Timers

- Supports interrupt generation
- Supports free-running mode
- Supports user-defined count mode

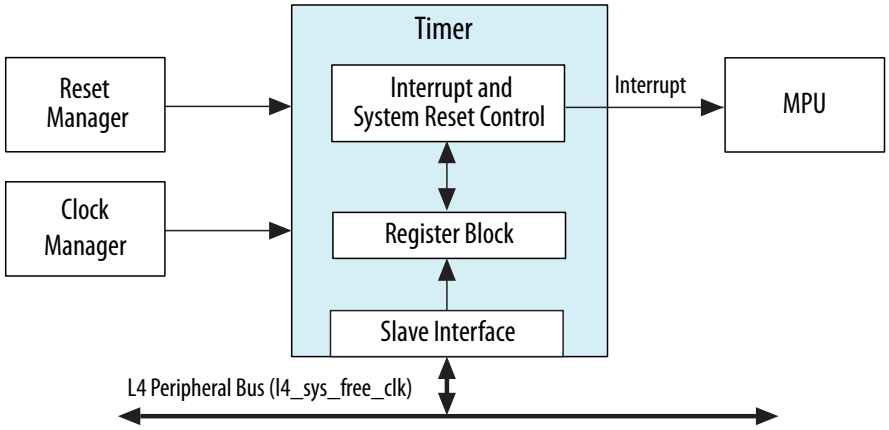
23.2. Timers Block Diagram and System Integration

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.

(48) 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.

Figure 127. Timers Block Diagram



23.3. Functional Description of the Timers

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: †

- Timers is enabled after being reset or disabled
- Timers counts down to 0

23.3.1. Clocks

Table 198. Timers Clock Characteristics

Timers		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`	Timers 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 149
For more information about clock performance, refer to the *Clock Manager* chapter.

23.3.2. 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.

23.3.3. 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.

23.4. Timers Programming Model

23.4.1. 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. †

23.4.2. Enabling the Timers

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.

23.4.3. Disabling the Timers

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. †

23.4.4. Loading the Timers 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. †

23.4.5. Servicing Interrupts

23.4.5.1. 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. †

23.4.5.2. 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. †

23.4.5.3. 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. †

23.5. Timers Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

24. Watchdog Timers

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 four programmable watchdog timers, which are connected to the level 4 (L4) peripheral bus. The watchdog timers are instances of the Synopsys DesignWare APB Watchdog Timers (DW_apb_wdt 1.08a) peripheral. (49)

The Cortex-A53 MPCore processor provides two additional watchdog timers.

Related Information

- Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
  For details on the document revision history of this chapter
- Cortex-A53 MPCore Processor on page 32
  For more information about the watchdog timers in the MPU, refer to *Quad-Core Cortex-A53 MPCore* chapter.

24.1. Features of the Watchdog Timers

The following list describes the features of the watchdog timer:

- Programmable 32-bit timeout range
- Timers 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

(49) 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.

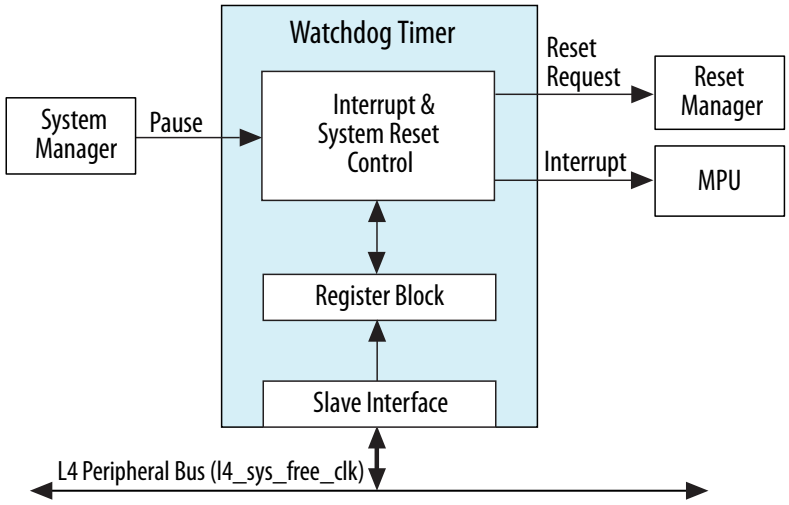
†Paragraphs marked with the dagger (†) symbol are Synopsys Proprietary. Used with permission.

24.2. Watchdog Timers 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 128. Watchdog Timers Block Diagram



Related Information

- Reset Manager on page 161
  For more information, refer to the *Reset Manager* chapter.
- Cortex-A53 MPCore Processor on page 32
  For more information about the watchdog timers in the MPU, refer to *Quad-Core Cortex-A53 MPCore* chapter.

24.3. Functional Description of the Watchdog Timers

24.3.1. Watchdog Timers 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 of 255) 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 Timers Clocks on page 480
- Setting the Timeout Period Values on page 481
- Selecting the Output Response Mode on page 481
- Reloading a Watchdog Counter on page 482

24.3.2. Watchdog Timers 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 when any CPU is in debug
- Pause the timer while only CPU0 is in debug mode
- Pause the timer while only CPU1 is in debug mode
- Pause the timer while only CPU2 is in debug mode
- Pause the timer while only CPU3 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 Timers on page 482

24.3.3. Watchdog Timers 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 199. Watchdog Timers Clocks

Timers	System Clock
watchdog0	l4_sys_free_clk
watchdog1	l4_sys_free_clk
watchdog2	l4_sys_free_clk
watchdog3	l4_sys_free_clk

Related Information

Clock Manager on page 149
For more information, refer to the *Clock Manager* chapter.

24.3.4. Watchdog Timers 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 161
For more information, refer to the *Reset Manager* chapter.

24.4. Watchdog Timers Programming Model

24.4.1. 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

24.4.2. 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 Timers Counter on page 479

24.4.3. Enabling and Initially Starting a Watchdog Timers

To enable and start a watchdog timer, write the value 1 to the watchdog timer enable bit (`wdt_en`) of the `wdt_cr` register.

24.4.4. 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.

24.4.5. Pausing a Watchdog Timers

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 171
For more information, refer to the *System Manager* chapter.

24.4.6. Disabling and Stopping a Watchdog Timers

The watchdog timers are disabled and stopped by resetting them from the reset manager.

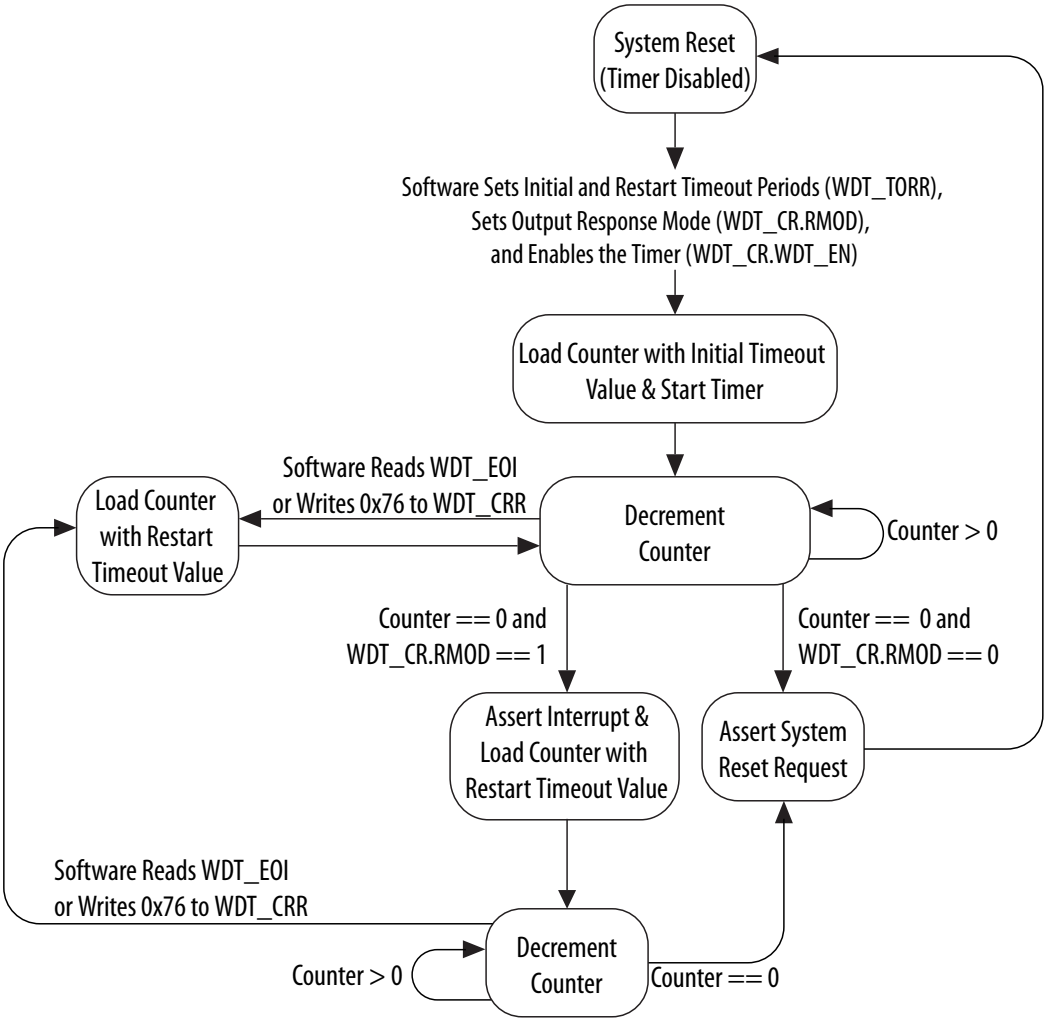
Related Information

Reset Manager on page 161
For more information, refer to the *Reset Manager* chapter.

24.4.7. Watchdog Timers 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 129. Watchdog Timers 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.

24.5. Watchdog Timers Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

25. 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 multi-core 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) infrastructure provides visibility and control of the HPS modules, the Arm Cortex-A53 MPCore processor, 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 489
- System Trace Macrocell on page 491
- Trace Funnel on page 492
- Embedded Trace FIFO on page 493
- AMBA Trace Bus Replicator on page 494
- Embedded Trace Router on page 493
- Trace Port Interface Unit on page 494
- Embedded Cross Trigger System on page 495
- Embedded Trace Macrocell on page 496
- Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
  For details on the document revision history of this chapter

25.1. Features of CoreSight Debug and Trace

The CoreSight debug and trace system offers the following features:

- One Debug APB interface slave for debug access
- Contains the following components for the Arm Cortex-A53 MPCore interface:
  - One Embedded Trace Macrocell (ETM) source per CPU with the ATB slave interface
  - One Cross Trigger Interface (CTI) per CPU
  - One Cross Trigger Matrix (CTM) for four CPU Triggers
- Supports four Trace input streams from CPUs through ATB buses
- Supports Trace input stream through AXI slave from the L3 interconnect
- Supports Trace Replicator for output interfaces
- Supports two authentication replicators:
  - CoreSight system
  - HPS MPU
- Supports Trace output bus through I/O pins
- Supports Trace output bus to the FPGA fabric
- Supports two trace outputs of NoC ports
  - MPFE NoC trace port—Disabled by default and all of the signals are in an invalid state
  - HPS NoC trace port—Connected to port 5 of the ATB
- 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:
  - Arm Cortex-A53 MPCore
  - FPGA
  - Cross Trigger Interface (CTI)
  - FPGA-CTI
  - Cross Trigger Matrix (CTM)
- Supports Debug Access Port (DAP) to allow the host to connect to the debugger through the JTAG
- Supports System Debug access port (DAPB) through the APB Slave
- Allows debug access to system resources through the DAP AXI Master Interface to the L4 Main Switch
- DAP supports the ROM table for the debugger to locate CoreSight components
- Supports the Debug access master APB output port to the MPU
- Supports the Timestamp generator for a consistent time value to multiple processors
- Supports ETR AXI Master from Fixed to Incrementing (Incr)
- Supports Timestamp replicator, encoder/decoder to CS IPs

- Supports Trace clock from Clock Manager or FPGA Fabric
- Supports CS clocks and clock enables (`cs_at_clk`/`cs_pdbg_clk`) inputs
- Supports JTAG TAP controller reset without nTRST pin and software reset bit[50]

Related Information

Cross Trigger Interface on page 495

25.2. Arm CoreSight Documentation

For more information about the Arm CoreSight components in the HPS debug system, refer to the following Arm CoreSight specifications and documentation located on the Arm Infocenter website:

- *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 0480G)
- *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)

For more information and where you can download these documents, refer to the Arm Infocenter website.

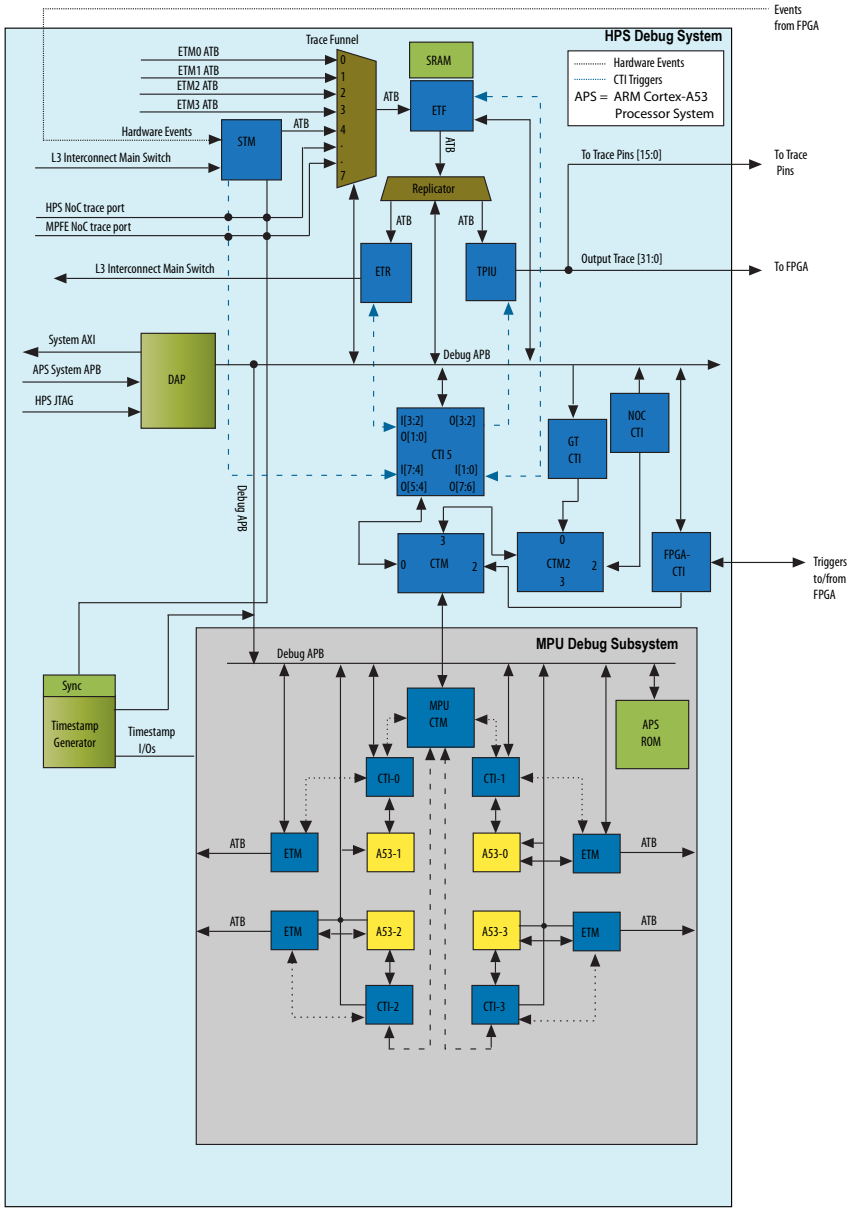
Related Information

Arm Infocenter

(50) The reset occurs by connecting the JTAG system reset (SRST pin) to the HPS reset pin.

25.3. CoreSight Debug and Trace Block Diagram

Figure 130. HPS CoreSight Debug and Trace Block Diagram



25.4. Functional Description of CoreSight Debug and Trace

25.4.1. Debug Access Port

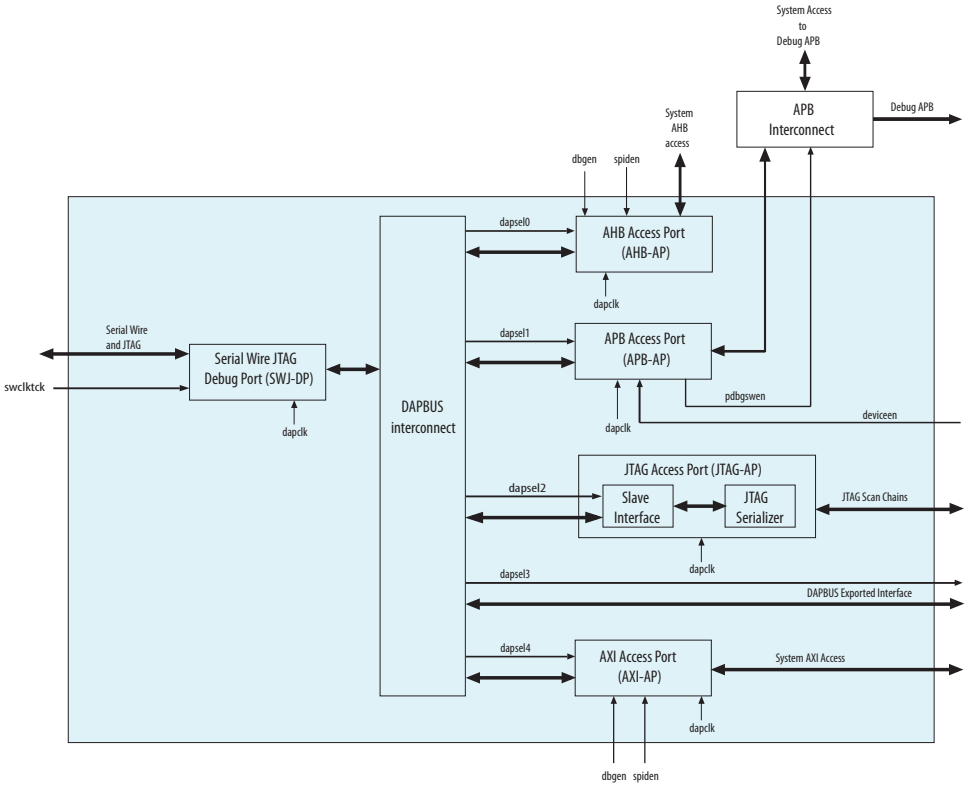
The Debug Access Port (DAP) provides the necessary ports for a host debugger, like the Arm Development Studio 5* (DS-5*) Intel SoC FPGA Edition running on a workstation, to connect to and communicate with the HPS through a JTAG interface. DAP is connected to the host using JTAG. Once this connection has been established, the debugger can access various modules inside the HPS.

There are several sub-components:

- Serial Wire JTAG Debug Port (SWJ-DP)
- DAPBUS interconnect
- DAPBUS async bridge
- DAPBUS sync bridge
- JTAG Access Port (JTAG-AP)
- AXI Access Port (AXI-AP)
- AHB Access Port (AHB-AP)
- APB Access Port (APB-AP)

The following figure shows how they are all integrated.

Figure 131. DAP Integration



Once connected, the host can access the DAPB port of the CoreSight components by using the APB-AP.

The debugger can access the system resources with the DAP which supports an AXI-AP port. The AXI-AP supports an AXI master that allows the debugger to access several memory mapped in HPS resources. The AXI-AP master port is connected to the L4 Main Switch.

This connection can be used by the debugger to access the state of an IP block that exists in FPGA.

Note: It is important to remember that all methods of access made by the debugger over an AXI port show up as normal access. There is no way for an IP block to know that an access is being made by the debugger.

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.

For more information, refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

Related Information

- CoreSight Debug and Trace Block Diagram on page 488
  Shows CoreSight components connected to the debug APB
- Arm Infocenter

25.4.1.1. JTAG Interface Options

The JTAG can be interfaced in two ways: through the HPS shared I/O or dedicated JTAG pins that are part of the device configuration pins. You can choose the method you want to use to connect to the DAP through Intel Quartus Prime Pro Edition. The HPS JTAG signals are multiplexed with HPS GPIO. The table below details which GPIO1 pin is multiplexed with each HPS JTAG pin.

Table 200. HPS Shared I/O JTAG Interface

JTAG Pins	Corresponding Multiplexed GPIO Pin
JTAG_TCK	GPIO1[8]
JTAG_TMS	GPIO1[9]
JTAG_TDO	GPIO1[10]
JTAG_TDI	GPIO1[11]

Note: The HPS JTAG interface does not support boundary scan tests (BST). To perform boundary scan testing on HPS I/Os, you must first chain the FPGA JTAG and HPS JTAG internally, and issue the boundary scan from the FPGA JTAG. To chain the FPGA and HPS JTAG internally, go to Quartus **Device and Pins Options** and select the **Configuration** category. Under the **HPS debug access port (DAP)** settings, choose **SDM Pins** from the drop down option. If boundary scan is not being used, the FPGA JTAG and HPS JTAG interfaces can be used independently. To select HPS Dedicated I/O as the interface for HPS JTAG, select HPS Pins from the drop down option instead.

For more information, refer to the "Hard Processor System I/O Pin Multiplexing" chapter.

Related Information

Hard Processor System I/O Pin Multiplexing on page 178

25.4.2. CoreSight SoC-400 Timestamp Generator

CoreSight Debug and Trace supports the following timestamp components for consistent time value to multiple processors:

- Timestamp generator—The timestamp generator is 64 bits wide and generates a timestamp value that provides a consistent view of time for multiple blocks in the HPS. The timestamp generator can be used to generate CoreSight timestamps or processor generic time.
- Timestamp encoder—The timestamp encoder converts the 64-bit timestamp value from the timestamp generator to a 7-bit encoded value. This is called a narrow timestamp. It also encodes and sends the timestamp value over a 2-bit synchronization channel.
- Timestamp decoder—The timestamp decoder converts the narrow timestamp interface and synchronization data back to a 64-bit value. This is the format in which the CoreSight SoC-400 trace components require their timestamp. It decodes the narrow timestamp interface to a 64-bit wide timestamp signal.

The timestamp generator generates the timestamp value that is distributed over the rest of the timestamp interfaces. It is used:

- To provide a system counter to the Arm Cortex-A53 MPCore processor generic timers. Only Secure software can change the timestamp value and Non-secure software can only read the timestamp value.
- To generate the time used to align traces and other debug information in the CoreSight system. The timestamp generator is controlled by debug software and connected to the debug APB interconnect.

For more information about the CoreSight SoC-400 Timestamp Generator, refer to the *Arm CoreSight SoC-400 Technical Reference Manual.*

Related Information

Arm CoreSight SoC-400 Technical Reference Manual

25.4.3. 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 supports an EVENT interface that can be used to post additional messages into a trace stream. In addition to the channels, 44 of the 64 EVENT signals are attached to the FPGA, which allows the FPGA to send additional messages using the STM.

The STM has access to one 16 MB region starting at `0xFC000000`. The hardware has fixed master IDs for the following masters that have access to this address range::

- DMA — 0x20
- FPGA ACE — 0x04
- CPU3 — 0x03
- CPU2 — 0x02
- CPU1 — 0x01
- CPU0 — 0x00

For more information, refer to the *CoreSight System Trace Macrocell Technical Reference Manual* on the Arm Infocenter website.

Related Information

- HPS Bridges on page 108
- CTI on page 501
- Arm Infocenter

25.4.4. Trace Funnel

The CoreSight Trace Funnel is used to combine multiple trace streams into one trace stream. There are multiple trace streams that use the funnel ports listed below:

Table 201. Trace Stream Connections

Funnel Port	Description
0	Used by the trace stream coming from ETM connected to instance t0 of the Cortex-A53 processor.
1	Used by ETM connected to instance 1 of the Cortex-A53 processor.
2	Used by ETM connected to instance 2 of the Cortex-A53 processor.
3	Used by ETM connected to instance 3 of the Cortex-A53 processor.
4	Connected to the STM ATB.
5	PSS NoC ATB is connected to port 5 of the ATB.
6	MPFE NoC ATB is connected to port 6 of the ATB.
7	Not connected.

For more information, refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

Related Information

- Arm Infocenter
- Embedded Trace Macrocell on page 496

25.4.5. CoreSight Trace Memory Controller

The CoreSight Trace Memory Controller (TMC) has three possible configurations:

- Embedded Trace FIFO (ETF)
- Embedded Trace Router (ETR)

ETB is not used in this device.

For more information, refer to the *CoreSight System Trace Memory Controller Technical Reference Manual* on the Arm Infocenter website.

Related Information

Arm Infocenter

25.4.5.1. Embedded Trace FIFO

The Trace Funnel output is sent to the ETF. The ETF is used as an elastic buffer between trace generators (STM, ETM) and trace destinations. The ETF stores up to 32 KB of trace data in the on-chip trace RAM.

For more information, refer to the *CoreSight System Trace Memory Controller Technical Reference Manual* on the Arm Infocenter website.

Related Information

Arm Infocenter

25.4.5.2. 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 CoreSight 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.

For more information, refer to the *CoreSight System Trace Memory Controller Technical Reference Manual* on the Arm Infocenter website.

Related Information

- HPS Bridges on page 108
- CoreSight Debug and Trace Programming Model on page 500
- Arm Infocenter

25.4.5.2.1. Distributed Virtual Memory Support

The system memory management unit (SMMU) in the HPS supports distributed virtual memory transactions initiated by masters.

As part of the SMMU, a translation buffer unit (TBU) sits between the ETR and the L3 interconnect. The ETR shares a TBU with the USB, NAND, and SD/MMC. An intermediate interconnect arbitrates accesses among the multiple masters before they are sent to the TBU. The TBU contains a micro translation lookaside buffer (TLB) that holds cached page table walk results from a translation control unit (TCU) in the SMMU. For every virtual memory transaction that this master initiates, the TBU compares the virtual address against the translations stored in its buffer to see if a physical translation exists. If a translation does not exist, the TCU performs a page table walk. This SMMU integration allows the ETR driver to pass virtual addresses directly to the ETR without having to perform virtual to physical address translations through the operating system.

For more information about distributed virtual memory support and the SMMU, refer to the *System Memory Management Unit* chapter.

Related Information

System Memory Management Unit on page 71

25.4.6. AMBA Trace Bus Replicator

The AMBA Trace Bus Replicator broadcasts trace data from the ETF to the ETR and TPIU.

For more information, refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

Related Information

Arm Infocenter

25.4.7. 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 ATB bus slave 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.

Table 202. Trace Pins

Signal	Description
`h2f_tpiu_clk`	TPIU trace clock output. TPIU generates this clock by dividing `cs_atclk` by 2. Supported frequency: 200/100/50/25/12.5 MHz
`h2f_tpiu_data[15:0]`	16 least significant bits of trace data output of TPIU. Data on this bus is synchronous to `h2f_tpiu_clk` and changes on both rising and falling edge of this clock. Supported data rate: 400/200/100/50/25 Mb/sec

For more information, refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

Related Information

Arm Infocenter

25.4.8. NoC Trace Ports

NoC trace ports are ports where the probes have been added and statistics collected and driven through the ATB port. Each NoC trace probe can be triggered with software writes to TP registers or by triggering CTI channels. Each NoC trace probe supports two CTI channels. Each of these channels are connected to the CTI-NoC. Each CTI supports up to eight channels: six probe channels connected to the six CTI channels and the remaining two CTI channels are open. Both PSS NoC and MPFE NoC ports are connected to one CTI.

For more information, refer to the "System Interconnect" chapter.

Related Information

System Interconnect on page 82

25.4.9. 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)

Related Information

- Cross Trigger Interface on page 495
- Features of CoreSight Debug and Trace on page 486
- Cross Trigger Matrix on page 495

25.4.9.1. Cross Trigger Interface

The HPS CTI is connected to authentication signals.

Trigger outputs can be masked when the invasive debug enable signal is LOW, to avoid debug tools changing the behavior of the system. If the corresponding bit of `todbgensel` bit is set to LOW, then the trigger outputs are masked by the `dbgen` signal. Otherwise, if the corresponding bit of `todbgensel` bit is set to HIGH, then the trigger outputs ignore the `dbgen` signal.

Trigger inputs can be masked when the non-invasive debug enable signal is LOW, to avoid debug tools being able to observe the state of the system. If the `tinidensel` bit is set to LOW, then the trigger outputs are masked by the `niden` signal. Otherwise, if the `tinidensel` bit is set to HIGH, then the trigger outputs ignore the `niden` signal.

In the HPS Cortex-A53 cluster, there are additional CTM available to communicate with other CTIs to control the halt mode of the generic timer.

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 ETM.

25.4.9.2. 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, FPGA-CTI, and MPU-CTM; the other connects CTI-NOC. The two CTMs are connected together, allowing triggers to be transmitted between the MPU debug subsystem, the debug system, and the FPGA fabric.

The following describes the inputs and outputs for the CTM.

Table 203. Cross Trigger Matrix Connections for CS CTM

Name	Source/Destination	Description
CTM channel input – Port 0	CTI	This data is in the CTI clock domain and can be synchronized.
CTM channel output – Port 0	CTI	Its output is synchronized to the CTM clock domain. The CTI can enable the clock domain synchronizers.
CTM channel input – Port 1	MPU-CTM	This data is in the MPU-CTM clock domain and can be synchronized.
CTM channel output – Port 1	MPU-CTM	Its output is synchronized to the MPU-CTM clock domain. The MPU-CTM can enable clock domain synchronizers.
CTM channel input – Port 2	FPGA-CTI	This data is in the FPGA-CTI clock domain and can be synchronized.
CTM channel output – Port 2	FPGA-CTI	Its output is synchronized to the FPGA-CTI clock domain. The FPGA-CTI can enable clock domain synchronizers.
CTM channel input – Port 3	CS CTM	This data is in the MPU-CTM clock domain and can be synchronized.
CTM channel output – Port 3	CS CTM	Its output is synchronized to the MPU-CTM clock domain. The CS CTM can enable clock domain synchronizers.

For more information, refer to the *CoreSight Components Technical Reference Manual* on the Arm Infocenter website.

Related Information

Arm Infocenter

25.4.10. Embedded Trace Macrocell

The ETM 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.

Each CPU core is paired with an ETM and a CTI. Trace data generated from the ETM 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.

25.4.11. 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 496

25.4.12. FPGA Interface

The following components connect to the FPGA fabric. This section lists the signals from the debug system to the FPGA.

25.4.12.1. DAP

The DAP uses the system APB port to connect to the FPGA.

Table 204. 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

25.4.12.2. STM

The STM has 44 event pins for FPGA to trigger events to STM and tracing event tracing.

25.4.12.3. FPGA-CTI

The FPGA-CTI allows the FPGA to send and receive triggers from the debug system.

The FPGA Cross-trigger interface directly drives a CoreSight Cross Trigger Interface (CTI). The CTI provides clock crossing capabilities.

Table 205. 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

For more information about the cross-trigger interface, refer to the Arm Infocenter website.

Related Information

Arm Infocenter

25.4.12.4. TPIU

TPIU is designed to transport trace data off the chip. Trace data enters the TPIU on the ATB bus slave port and leaves through the 32-bit wide TRACEDATA port.(51) Trace data coming out of the TPIU can be sent to the FPGA.

Table 206. 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[32]	32-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

25.4.13. Debug Clocks

The CoreSight system monitors four clocks from clock manager.

Table 207. 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	There are multiple CTIs, each with a different clock. The FPGA–CTI clock comes from the CS subsystem.
CTMCLK	Clock manager	cs_pdbg_clk	Cross trigger matrix clock. 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.

continued...

(51) Software can control the width of TRACEDATA port by programming the Current Port Size register.

Port Name	Clock Source	Signal Name	Description
PCLKSYS	Clock manager	cs_pdbg_clk	Used by the APB slave port inside the DAP. It is asynchronous to DAPCLK; and is the same as the tck signal from JTAG.
SWCLKTCK	JTAG interface	dap_tck	This is the SWJ-DP clock driven by the external debugger and is synchronous to DAPCLK. This clock is the same as the tck signal from JTAG.
TRACECLKIN (from the SoC)	Clock manager	cs_trace_clk	TPIU trace clock input. It is asynchronous to ATCLK. In the HPS, this clock comes from the clock manager.
	FPGA fabric	tpiu_traceclkin	TPIU trace clock input. It is asynchronous to ATCLK. In the HPS, this clock comes from the FPGA fabric.

For more information about the CoreSight port names, refer to the *CoreSight Technology System Design Guide* on the Arm Infocenter website.

Related Information

Arm Infocenter

25.4.14. Debug Resets

Table 208. CoreSight Resets

ARM Reset Name	Clock Source	HPS Reset Signal Name	Description
ARESETn	Reset manager	dbg_rst_n	STM AXI slave and DMA peripheral request block reset.
STMRESETn	Reset manager	dbg_rst_n	Resets the rest of the STM including the APB, HW EVT block, and the TGU block.
ATRESETn	Reset manager	dbg_rst_n	Trace bus reset. It resets all registers in the ATCLK domain.
CTIRESETn (from the CTI)	Reset manager	dbg_rst_n	CTI reset signal. It resets all registers in the CTICLK domain.
CTIRESETn (from the FPGA-CTI)	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.
PRESETDBGn	Reset manager	dbg_rst_n	Connected to the A53 CPUs. Connected to DAP AXI Reset
PRESETDBGn	Reset manager	cs_dap_rst_n	Debug APB reset. Connected to DAP AXI Reset.
RESETn	Reset manager	—	Debug APB reset. Resets all registers clocked by PCLKDBG.
PRESETSYSn	Reset manager	dbg_rst_n	Resets system APB slave port of DAP.

continued...

ARM Reset Name	Clock Source	HPS Reset Signal Name	Description
nCTMRESET	Reset manager	dbg_rst_n	CTM reset signal. It resets all signals clocked by CTMCLK.
nPOTRST	Reset manager		True power on reset signal to the DAP SWJ-DP. It must only reset at power-on.
TRESETn	Reset manager	dbg_rst_n	Reset signal for TPIU. Resets all registers in the TRACECLKIN domain.
timestamp	timestamp reset	—	GEN CPU TS. APB reset- sys_dbg_rst_n. Trace Timestamp: • APB reset (resetn) - dbg_rst_n • timestamp reset - dbg_rst_n

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.

For more information about the CoreSight port names, refer to the *CoreSight Technology System Design Guide* on the Arm Infocenter website.

Related Information

- Reset Manager on page 161
- Watchdog Timers on page 478
- Arm Infocenter

25.5. CoreSight Debug and Trace Programming Model

This section describes programming model details specific to Intel's 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.

For more information about the programming interface of each CoreSight component, refer to the Arm Infocenter website.

Related Information

Arm Infocenter

25.5.1. 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 209. 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	0x00008	NOC-CTI
0x8	0x00008	ATBREPLICATOR
0x9	0x0000A	TS
0xA	0x0000B	GT-CTI
0xB	0x00080	FPGA ROM
0xC	0x00400	A53 ROM
0xD	0x00000	End of ROM

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.

25.5.2. 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

25.5.2.1. 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 210. 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

continued...

Number	Signal	Source
4	TRIGOUTSPTE	STM
3	ACQCOMP	ETR
2	FULL	ETR
1	ACQCOMP	ETF
0	FULL	ETF

Table 211. 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 212. 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	—

25.5.2.2. 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 503
For more information about the triggers, refer to the "Configuring Embedded Cross-Trigger Connections" chapter.

25.5.2.3. L3-CTI

L3-CTI has all of its triggers available to the L3 interconnect.

25.5.3. 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.

There are soft logic signals in the FPGA that are individually connected to trigger inputs in the FPGA–CTI that are configurable.

- Connected to a trigger input—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.
- Connected to trigger input T2—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.
- Connected to trigger input T1—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.

For more information about the cross-trigger interface, refer to the Arm Infocenter website.

Related Information

- CoreSight Component Address on page 500
- Arm Infocenter

25.5.3.1. 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-to-multiple, multiple-to-single, and multiple-to-multiple.

25.5.3.2. 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.

25.5.3.3. 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.

25.5.3.4. 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.

25.6. CoreSight Debug and Trace Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)

A. Booting and Configuration

The Secure Device Manager (SDM) in the FPGA manages the hard processor system (HPS) boot and FPGA configuration of the Intel Agilex SoC device. Both the HPS boot and FPGA configuration comprise a series of stages that always begins with SDM initialization.

After the Intel Agilex SoC device is released from power-on-reset (POR), the SDM manages the initial configuration of the device. All configuration and boot source interfaces are connected to the SDM. The SDM determines and enforces the security level on the device, ensuring the bitstream, which includes the HPS first stage bootloader (FSBL), originate from a trusted source.

You can program the Intel Agilex SoC device to configure the FPGA first and then boot the HPS. Alternatively, you can boot the HPS first and then configure the FPGA core as part of the second-stage boot loader (SSBL) or after the Operating System (OS) boots.

The following documents provide a comprehensive guidance on managing the HPS boot, FPGA configuration and security:

- Intel Agilex Configuration User Guide
- Intel Agilex Boot User Guide (coming soon)

Intel describes configuration schemes from the point-of-view of the FPGA. Intel Agilex devices support active and passive configuration schemes. In active configuration schemes the FPGA acts as the master and the external memory acts as a slave device. In passive configuration schemes an external host acts as the master and controls configuration. The FPGA acts as the slave device. All Intel Agilex configuration schemes support design security, remote system update, and partial reconfiguration. To implement remote system update in passive configuration schemes, an external controller must store and drive the configuration bitstream.

Intel Agilex devices support the following configuration schemes:

- Avalon® Streaming (Avalon-ST)
- JTAG
- Configuration via Protocol (CvP)
- Active Serial (AS) normal and fast modes
- Secure Digital and Multi Media Card (SD/MMC)

Avalon-ST

The Avalon-ST configuration scheme is a passive configuration scheme. Avalon-ST is the fastest configuration scheme for Intel Agilex devices. Avalon-ST configuration supports x8, x16, and x32 modes. The x16 and x32 bit modes use general-purpose I/Os (GPIOs) for configuration. The x8 bit mode uses dedicated SDM I/O pins.

Note: The AVST_data[15:0], AVST_data[31:0], AVST_clk, and AVST_valid use dual-purpose GPIOs which operate at 1.2 V. . You can use these pins as regular I/Os after the device enters user mode.

Avalon-ST supports backpressure using the AVST_READY and AVST_VALID pins. Because the time to decompress the incoming bitstream varies, backpressure support is necessary to transfer data to the Intel Agilex device. For more information about the Avalon-ST refer to the *Avalon Interface Specifications*.

JTAG

You can configure the Intel Agilex device using the dedicated JTAG pins. The JTAG port provides seamless access to many useful tools and functions. In addition to configuring the Intel Agilex, the JTAG port is used for debugging with Signal Tap or the System Console tools.

The JTAG port has the highest priority and overrides the MSEL pin settings. Consequently, you can configure the Intel Agilex device over JTAG even if the MSEL pin specify a different configuration scheme unless you disabled JTAG for security reasons.

CvP

CvP uses an external PCIe* host device as a Root Port to configure the Intel Agilex device over the PCIe link. You can specify up to a x16 PCIe link. Typically, the bitstream compression ratio and the SDM input buffer data rate, not the PCIe link, limit the configuration data rate Intel Agilex devices support two CvP modes, CvP init and CvP update.

CvP initialization process includes the following two steps:

1. CvP configures the FPGA periphery image which includes I/O information and hard IP blocks, including the PCIe IP. Because the PCIe IP is in the periphery image, PCIe link training establishes the PCIe link of the CvP PCIe IP before the core fabric configures.
2. The host device uses the CvP PCIe link to configure your design in the core fabric.

CvP update mode updates the FPGA core image using the PCIe link already established from a previous full chip configuration or CvP init configuration. After the Intel Agilex enters user mode, you can use the CvP update mode to reconfigure the FPGA fabric. This mode has the following advantages:

- Allows reprogramming of the core to run different algorithms.
- Provides a mechanism for standard updates as a part of a release process.
- Customizes core processing for different components that are part of a complex system.

  For both CvP Init and CvP Update modes, the maximum data rate depends on the PCIe generation and number of lanes.

For Intel Agilex SoC devices, CvP is only supported in FPGA configuration first mode.

AS Normal Mode

Active Serial x4 or AS x4 or Quad SPI is an active configuration scheme that supports flash memories capable of three- and four-byte addressing. Upon power up, the SDM boots from a boot ROM which uses three-byte addressing to load the configuration firmware from the Quad SPI flash. After the configuration firmware loads, the Quad SPI flash operates using four-byte addressing for the rest of the configuration process.

AS Fast Mode

The only difference between AS normal mode and fast mode is speed. Use AS fast mode when configuration timing is a concern. Use this mode to meet the 100 ms of power up requirement for PCIe or for other systems with strict timing requirements.

In AS fast mode, the SDM first powers the external AS x4 flash. The power supply must be able to provide an equally fast ramp up for the Intel Agilex device and the external AS x4 flash devices. Failing to meet this requirement causes the SDM to assume the memory is missing . Consequently, configuration fails.

SD/MMC

SD/MMC is an active configuration scheme. The Intel Agilex SDM can initiate configuration from SD, Secure Digital High Capacity (SDHC*). Secure Digital Extended Capacity (SDXC*), MMC cards, and eMMC devices. The SD/MMC mode is almost identical to AS x4. The advantages of this mode are cost, capacity, availability, portability, and compatibility. Because Intel Agilex devices operate at 1.8 volt an intermediate voltage level shifter may be required to interface with the higher voltage I/Os in SD/MMC devices.

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

A.1. FPGA Configuration First Mode Overview

You can program the Intel Agilex SoC device to configure the FPGA first and then boot the HPS. The available configuration data sources configure the FPGA core and periphery first in this mode. After completion, you may optionally boot the HPS. All of the I/O, including the HPS-allocated I/O, are configured and brought out of tri-state. If the HPS is not booted:

- The HPS is held in reset.
- HPS-dedicated I/O are held in reset.
- HPS-allocated I/O are driven with reset values from the HPS.

The boot flow for the FPGA configuration first mode is presented in the figure below. The flow includes the time from power-on-reset (T_{POR}) to boot completion ($T_{Boot\_Complete}$).

Figure 132. Typical FPGA Configuration First Boot Flow

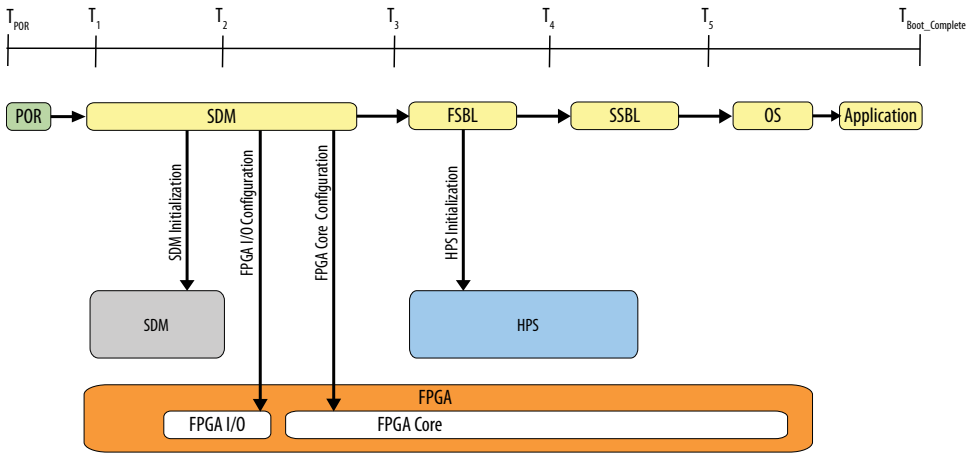


Table 213. FPGA Configuration First Stages

The sections following this table describe each stage in more detail.

Time	Boot Stage	Device State
T_{POR} to T_1	POR	Power-on reset
T_1 to T_2	SDM: Boot ROM	1. SDM samples the MSEL pins to determine the configuration scheme and boot source. 2. SDM establishes the device security level based on eFuse values. 3. SDM initializes the device by reading the configuration firmware (initial part of the bitstream) from the boot source. 4. SDM authenticates and decrypts the configuration firmware (this process occurs as necessary throughout the configuration). 5. SDM starts executing the configuration firmware.
T_2 to T_3	SDM: Configuration Firmware	1. SDM I/O are enabled. 2. SDM configures the FPGA I/O and core (full configuration) and enables the rest of your configured SDM I/O. 3. SDM loads the FSBL from the bitstream into HPS on-chip RAM. 4. SDM enables HPS SDRAM I/O and optionally enables HPS debug. 5. FPGA is in user mode. 6. HPS is released from reset. CPU1-CPU3 are in a wait-for-interrupt (WFI) state.
T_3 to T_4	First-Stage Boot Loader (FSBL)	1. HPS verifies the FPGA is in user mode. 2. The FSBL initializes the HPS, including the SDRAM. 3. HPS loads SSBL into SDRAM. 4. HPS peripheral I/O pin mux and buffers are configured. Clocks, resets, and bridges are also configured. 5. HPS I/O peripherals are available.
T_4 to T_5	Second-Stage Boot Loader (SSBL)	1. HPS bootstrap completes. 2. OS is loaded into SDRAM.
T_5 to $T_{Boot\_Complete}$	Operating System (OS)	The OS boots and applications are scheduled for runtime launch.

A.2. HPS Boot First Mode Overview

You can boot the HPS and HPS EMIF I/O first before configuring the FPGA core and periphery. The MSEL[2:0] settings determine the source for booting the HPS. In this mode, any of the I/O allocated to the FPGA remain tri-stated while the HPS is booting.

The HPS subsequently configures the FPGA core and periphery excluding the HPS EMIF I/O. Software determines the configuration source for the FPGA core and periphery. In HPS boot first mode, you have the option of configuring the FPGA core during the SSBL stage or when the OS boots.

In the context of HPS Boot First mode, the initial configuration of HPS EMIF I/O and loading of HPS FSBL is called "Phase 1 configuration". The subsequent configuration of FPGA core and periphery by HPS is called "Phase 2 configuration".

A typical HPS Boot First flow may look like the following figure. You can use U-Boot, Unified Extensible Firmware Interface (UEFI) or a custom boot loader for your FSBL or SSBL. An example of an OS is Linux or an RTOS. The flow includes the time from power-on-reset (T_{POR}) to boot completion ($T_{Boot\_Complete}$).

Figure 133. Typical HPS Boot First Flow

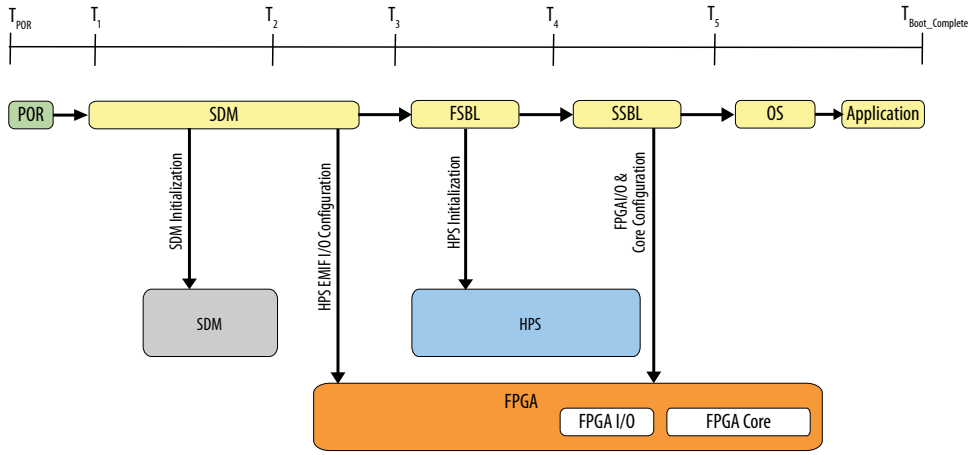


Table 214. HPS Boot First Stages

Time	Boot Stage	Device State
T_{POR}	POR	Power-on reset
T_1 to T_2	SDM: Boot ROM	1. SDM samples the MSEL pins to determine the configuration scheme and boot source. 2. SDM establishes the device security level based on eFuse values. 3. SDM initializes the device by reading the configuration firmware (initial part of the bitstream) from the boot source. 4. SDM authenticates and decrypts the configuration firmware (this process occurs as necessary throughout the configuration). 5. SDM starts executing the configuration firmware.
T_2 to T_3	SDM: Configuration Firmware	1. SDM configures the HPS EMIF I/O and the rest of the user-configured SDM I/O. 2. SDM loads the FSBL from the bitstream into HPS on-chip RAM. 3. SDM enables HPS SDRAM I/O and optionally enables HPS debug. 4. HPS is released from reset.

continued...

Time	Boot Stage	Device State
T_3 to T_4	First-Stage Boot Loader (FSBL)	1. The FSBL initializes the HPS, including the SDRAM. 2. FSBL obtains the SSBL from HPS flash or by requesting flash access from the SDM. 3. FSBL loads the SSBL into SDRAM. 4. HPS peripheral I/O pin mux and buffers are configured. Clocks, resets and bridges are also configured. 5. HPS I/O peripherals are available.

T_4 to T_5	Second-Stage Boot Loader (SSBL)	• HPS bootstrap completes.
		After bootstrap completes, any of the following steps may occur:
		1. The FPGA core configuration loads into SDRAM from one of the following sources:
		  • SDM flash
		  • HPS alternate flash
		  • EMAC interface
		2. HPS requests SDM to configure the FPGA core.(52)
		3. FPGA enters user mode
		4. OS is loaded into SDRAM.

T_5 to $T_{Boot\_Complete}$	Operating System (OS)	1. OS boot occurs and applications are scheduled for runtime launch 2. (Optional step) OS initiates FPGA configuration through a secure monitor call (SMC) to the SSBL, which then initiates the request to the SDM.

Note: The location of the source files for configuration, FSBL, SSBL and OS can vary.

(52) FPGA I/O and FPGA core configuration can occur at the SSBL or OS stage, but is typically configured during the SSBL stage.

B. Accessing the Secure Device Manager Quad SPI Flash Controller through HPS

The HPS has the capability to access serial NOR flash connected to the SDM quad serial peripheral interface (SPI). The quad SPI flash controller supports standard SPI flash devices as well as high-performance dual and quad SPI flash devices.

Note: The data transfer speed when HPS is accessing the quad SPI controller is about an order of magnitude slower than when the SDM accesses it. You must consider whether this transfer speed meets your end application requirements.

Related Information

Intel Agilex Hard Processor System Technical Reference Manual Revision History on page 12
For details on the document revision history of this chapter

B.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(53)
- Direct access and indirect access modes
- Single I/O, dual I/O, or quad I/O instructions
- Up to four chip selects(54)
- Configurable clock polarity and phase
- Programmable write-protected regions
- Programmable delays between transactions
- Programmable device sizes
- Read data capture tuning
- Local buffering for indirect transfers
- Support eXecute-In-Place (XIP) mode

B.2. Taking Ownership of Quad SPI Controller

On power up, the SDM owns the quad SPI controller. In order for the HPS to use the quad SPI Controller, it has to request ownership from the SDM.

(53) Supported flash devices limit the speed.

(54) When using multiple chip selects, the maximum device clock frequency is lower.

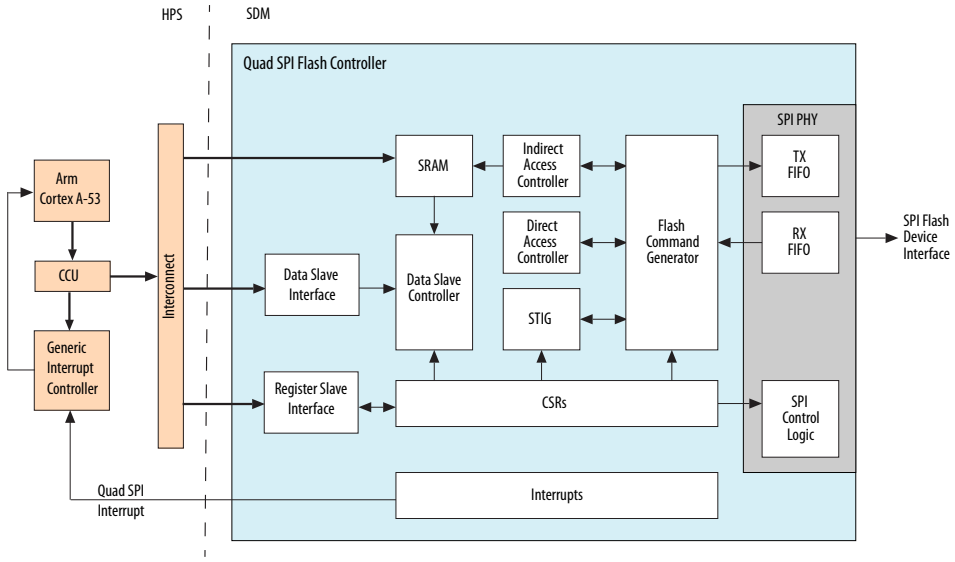
The following details the typical flow for the HPS to use the quad SPI:

1. The Bootloader (either U-Boot or UEFI) is configured to use quad SPI, and takes control of the quad SPI from the SDM. The SDM resets the quad SPI controller and reports back to the bootloader the value of the quad SPI reference clock.
2. The bootloader passes the value of the quad SPI Controller reference clock to the end application or operating system.
3. The end application uses the quad SPI controller.

For the Linux* use case, the U-Boot passes the value of the quad SPI reference clock into the Linux device tree. The HPS cannot reset the quad SPI controller, gate its clocks, nor use the DMA for quad SPI transfers; it can only obtain ownership of the quad SPI controller when the MSEL pins are configured to select quad SPI for SDM configuration. Quad SPI pin-muxing is configured in the Intel Quartus Prime project, and cannot be changed by the HPS.

B.3. Quad SPI Flash Controller Block Diagram and System Integration

Figure 134. Quad SPI Flash Controller Block Diagram and System Integration



The quad SPI controller consists of the following blocks and interfaces:

- Data slave controller - Interface and controller that provides the following functionality:
  - Performs data transfers to and from the interconnect
  - Validates incoming accesses
  - Performs byte or half-word reordering
  - Performs write protection
  - Forwards transfer requests to direct and indirect controller
- Indirect access controller - provides higher-performance access to the flash memory through local buffering and software transfer requests
- Direct access controller - provides memory-mapped slaves direct access to the flash memory
- 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
- Register slave interface - Provides access to the control and status registers (CSRs)
- SPI PHY - serially transfers data and commands to the external SPI flash devices

B.4. 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.

Table 215. Interface Pins

Pin	Mode	Direction	Function
IO0	Single	Output	Data output 0
	Dual or quad	Bidirectional	Data I/O 0
IO1	Single	Input	Data input 0
	Dual or quad	Bidirectional	Data I/O 1
IO2_WPN	Single or dual	Output	Active low write protect
	Quad	Bidirectional	Data I/O 2
IO3_HOLD	Single, dual, or quad	Bidirectional	Data I/O 3
SS0	Single, dual, or quad	Output	Active low slave select 0
SS1			Active low slave select 1

continued...

Pin	Mode	Direction	Function
SS2			Active low slave select 2
SS3			Active low slave select 3
CLK	Single	Output	quad SPI serial clock output

B.5. Functional Description of the Quad SPI Flash Controller

B.5.1. 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.

B.5.2. 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.

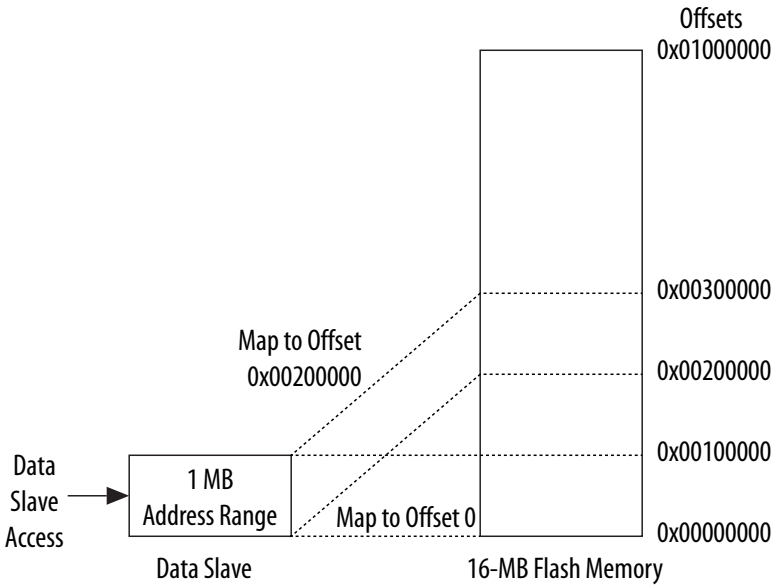
B.5.2.1. Direct Access Mode

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.

B.5.2.1.1. Data Slave Remapping Example

Figure 135. 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.

B.5.2.1.2. 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.

Software, using the documented programming interface, triggers FLASH erase operations, which may be required before a page write.

Once a page program cycle has been started, the QSPI Flash Controller automatically polls 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.

B.5.2.2. 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.

B.5.2.2.1. 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 following registers control the indirect read operations:

- 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 will pad 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" section.

Related Information

Indirect Read Operation on page 524

B.5.2.2.2. Indirect Write Operation

An indirect write operation programs data from the SRAM to the flash memory. The following registers control the indirect write operations:

- 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" section.

Related Information

Indirect Write Operation on page 524

B.5.2.2.3. 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.

B.5.3. 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`).

B.5.4. Register Slave Interface

The quad SPI flash controller uses the register slave interface, a mapped 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.

B.5.4.1. 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.

The STIG issues commands that have a higher priority than all other read accesses and interrupts any read commands that the direct or indirect controllers request. 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.

B.5.5. Local Memory Buffer

The SRAM local memory buffer is a 1024 by 32-bit (4096 total bytes) memory.

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 256 (0x100) to partition the 1024-entry SRAM to 256 entries (25%) for read usage and 768 entries (75%) for write usage.

B.5.6. 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.

B.5.7. 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.

B.5.8. 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.

B.5.9. 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.

B.5.10. 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.

B.5.11. Clocks

The quad SPI controller uses an input clock called `qspi_ref_clk`. The `qspi_clk output` clock to the flash device is derived by dividing down the `qspi_ref_clk` clock by the baud rate divisor field (`bauddiv`) of the `cfg` register.

The value of the `qspi_ref_clk` is determined by the SDM based on the desired Active Serial (AS) configuration clock value you selected in Intel Quartus Prime.

You can set this value by following these steps:

1. Open project in Intel Quartus Prime Pro Edition.
2. Navigate to **Assignments ➤ Device...**.
3. From the **Device Assignments** window, click on the **Device and Pin Options** button.
4. Under "Category", select *General*; and in the *General* sub-window, select the **Configuration clock source** from the available options.
5. Under "Category", select *Configuration*; and in the *Configuration* sub-window, select the **Active serial clock source** from the available options. These options depend on what was selected in the previous step.

The value of the `qspi_ref_clk` is obtained by the bootloader when it takes ownership of the quad SPI Flash controller. The bootloader then typically passes this information to the end application or operating system.

B.5.12. Resets

The quad SPI controller reset is controlled by the SDM. The SDM always resets the quad SPI controller just before handing ownership to the HPS. The HPS cannot initiate a quad SPI controller reset.

B.5.13. 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 216 on page 522 lists the interrupt sources in the `irqstat` register.

Table 216. 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.

continued...

Interrupt Source	Description
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.
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

B.6. Quad SPI Flash Controller Programming Model

B.6.1. 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.

B.6.2. Indirect Read Operation

The following steps describe the general software flow to set up the quad SPI controller for indirect read operation:

1. Perform the steps described in the Setting Up the Quad SPI Flash Controller on page 523 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.

B.6.3. Indirect Write Operation

The following steps describe the general software flow to set up the quad SPI controller for indirect write operation:

1. Perform the steps described in the Setting Up the Quad SPI Flash Controller on page 523 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.

B.6.4. 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.

B.6.4.1. Entering XIP Mode

B.6.4.1.1. 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.

B.6.4.1.2. 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 will be 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.

B.6.4.1.3. 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.

B.6.4.1.4. 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.

B.6.4.2. 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.

B.6.4.3. 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 known the device will 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 will 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.

B.7. Accessing the SDM Quad SPI Flash Controller Through HPS Address Map and Register Definitions

You can access the complete HPS address map and register definitions through the following:

- Intel Agilex Hard Processor System Address Map and Register Descriptions (ZIP)