XHMC v1.0

LogiCORE IP Product Guide

Vivado Design Suite

PG216 October 4, 2017

XILINX
ALL PROGRAMMABLE™

Table of Contents

IP Facts

Chapter 1: Overview

Core Architecture 6
Feature Summary 10
Applications 11

Chapter 2: Product Specification

Standards 12
Performance and Resource Utilization 12
Port Descriptions 12
Register Space 24

Chapter 3: Designing with the Core

General Design Guidelines 30
Clocking 30
Resets 33
Power State Management 35

Chapter 4: Design Flow Steps

Customizing and Generating the Core 36
Constraining the Core 43
Simulation 45
Synthesis and Implementation 45

Chapter 5: Example Design

Overview of the Example Design 46
Simulating the Example Design 51
Synthesizing and Implementing the Example Design 52

Appendix A: Upgrading

Appendix B: Debugging

Finding Help on Xilinx.com 54

Debug Tools 55

Appendix C: Additional Resources and Legal Notices

Xilinx Resources 56
Documentation Navigator and Design Hubs 56
References 56
Revision History 57
Please Read: Important Legal Notices 57

Introduction

The Xilinx® LogiCORE™ IP HMC (XHMC) Controller implements a high performance, configurable Hybrid Memory Cube (HMC) host controller that can interconnect with external HMC devices. The core provides either a Xilinx HMC Transaction Layer or an AXI4-MM interface.

Features

- Fully compliant with the Hybrid Memory Cube Specification Revision 1.x.
- Support 10 Gb/s, 12.5 Gb/s, 15 Gb/s transceiver interface.
- Support either GTH or GTY transceiver use.
- Support both HMC link modes.
- Half-width link (8-lane) supported.
- Full-width link (16-lane) supported.
- Up to 240 Gb/s full-duplex bandwidth.
- Two interface modes:
  - HMC core mode provides Xilinx HMC Transaction Layer with host controller only instantiation. Allows you to develop your user layer to interface between the user logic and controller IP.
  - HMC user mode provides standard AXI4-MM interface with Xilinx HMC user layer logic.
- AXI4-Lite slave interface for configuration access.
- Configurable internal bus width (2 – 12 flit).
- Support dynamic lane detection and reversal.
- Support user defined lane-to-lane mappings.
- Support dynamic lane polarity inversion.
- Optional user ID reorder logic.
- Optional data channel alignment bypass for area optimization.

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale™ UltraScale+™, including Zynq® UltraScale+
Supported User Interfaces	Xilinx HMC Transaction Layer, AXI4-MM
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted Verilog
Example Design	Verilog
Test Bench	Verilog
Constraints File	Xilinx Design Constraints (XDC)
Simulation Model	Third-party HMC device BFM (from Micron) included
Supported S/W Driver(2)	N/A
Tested Design Flows(2)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The XHMC host controller is a high-bandwidth HMC memory link interconnect block which allows Xilinx® device designers to access HMC memory over an the HMC link without the workload of running the HMC link protocol. The XHMC IP is designed for use with Xilinx Ultrascale™ and UltraScale+™ devices. The core instantiates the integrated GT blocks in Ultrascale and UltraScale+ devices.

Although the core is a fully verified solution, the challenge associated with implementing a complete design varies depending on the configuration and functionality of the application.

Each XHMC bridges a single HMC link between a HMC device and the user applications hosted by a Xilinx device. To connect multiple links which are usually provided by a HMC device, you need to instantiate multiple XHMC controller IPs in the system. Figure 1-1 illustrates a typical multi-link architecture.

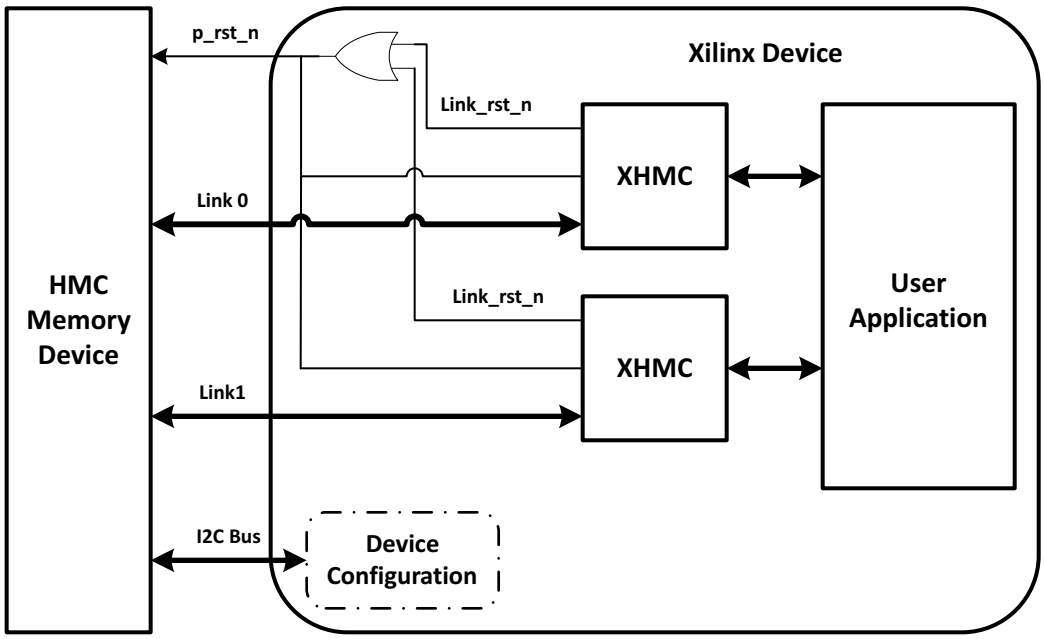

Figure 1-1: **Multi-link HMC System Architecture**

As shown in the figure, each XHMC host controller instance handles the HMC transactions from/to an individual HMC link and provides the interface independently for the user

applications. You can select the interface type when generating the IP. If you choose to include XHMC User Layer logic, the IP provides the standard AXI4-Memory-Mapped (AXI4-MM) interface. Otherwise, the IP provides the native Transaction Layer interface based on the original HMC FLIT (flow control digit) based packet format. To bring up the HMC link, a host controller requires the control over timing for the device reset and link configuration. Since the device shares the chip reset and the configuration interface among all provided links, special cares are needed. Each XHMC controller instance provides a reset output pin (`Link_rst_n`) and feedback input (`p_rst_n`). You are required to OR all the `Link_rst_n` pins towards to the same HMC device together to generated the chip reset signal (`p_rst_n`) to the HMC device. The generated device reset signal needs to be feedback to the controller as well. A controller release the link reset when it is ready to advance its reset state machine. The release of the device release flags that all the controllers are ready. An active reset feedback holds the reset state machine from advancing. You can choose to configure the device as a whole or configure each link in the system sequentially. Please refer to HMC Reset Sequence and HMC Controller Initialization Sequence in Chapter 3 for details.

Core Architecture

The XHMC IP is fully compliant with HMC Consortium specification rev1.x [Ref 2]. The entire HMC protocol and functions have been implemented exactly following the HMC specification definition. Figure 1-2 shows the block diagram of the XHMC Controller IP.

The XHMC IP consists of:

- HMC User Layer
- HMC Protocol Core
  - HMC Transaction Layer
  - HMC Link Layer
- HMC Physical Layer
- GT wrapper with up to 16 GT channels
- CSR Configuration module

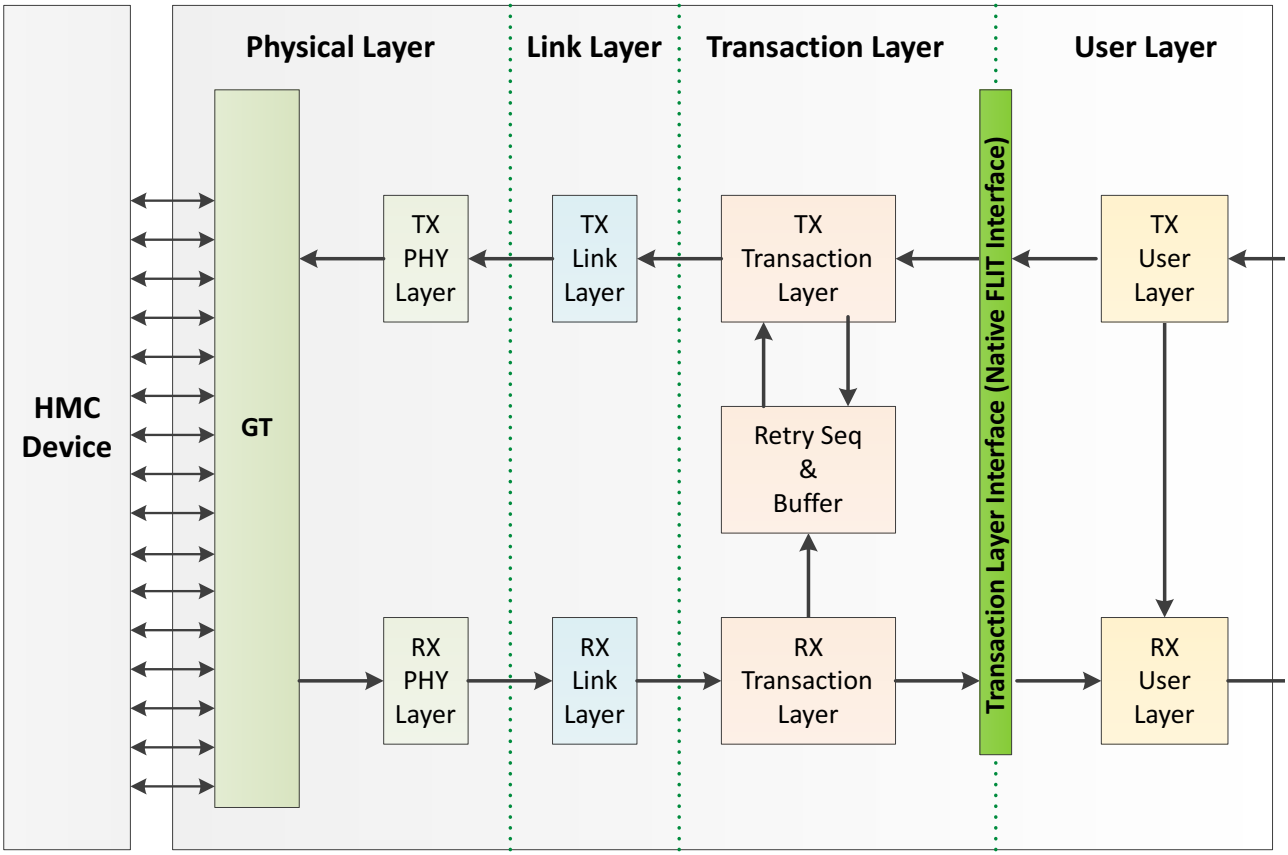

Figure 1-2: **XHMC Host Controller Layered Block Diagram**

XHMC User Layer

The XHMC User Layer provides the user with access to an HMC device through the industrial standard AMBA AXI4 Memory Mapped bus interface (AXI4-MM). XHMC User Layer handles the HMC packet framing/de-framing, as well as the HMC TAG management, which renders the HMC protocol completely transparent to the user.

In the transmission direction, the responsibilities of the XHMC User Layer includes:

- AXI4-MM user interface control
- HMC packet framing from AXI4-MM request
- HMC TAG request and AWID/ARID registration
- Transaction layer FLITs data bus scheduling and multiplexing for transmit HMC packets

For the memory transactions that received from the same AXI4-MM port, the User Layer preserves the order when delivering the corresponding HMC request packets to the downstream layer (i.e., Transaction Layer). When multiple AXI4-MM ports are configured in the system, the transaction orders among different AXI4-MM ports are not guaranteed.

In the receiving direction, the HMC User Layer takes HMC packets from the HMC Transaction Layer and performs following functions:

- HMC TAG lookup and free
- HMC packets dispatching and data bus unpacking
- HMC packet de-framing
- Response data reordering

If the response reordering function is enabled, the responses are returned to the user logic in the BID/RID sequence which corresponding to the request's AWID/ARID. An AWID/ARID aging timer is engaged in this configuration to avoid HMC traffic being stuck forever due to any vanished AWID/ARID. The time period for AWID/ARID timeout is configurable and should be much longer than normal HMC latency. In case a request's AWID/ARID was timeout, the HMC user layer will generate a timeout response to the user logic with the aging-out flag in BUSER/RUSER being set and you should discard all information in the response.

XHMC Transaction Layer

The transaction layer is the upmost layer in architecture defined by the HMC specification. The primary functionalities, according the specification, are:

- Input and output buffering
- Flow control
- FRP/RRP/RTC/SEQ field insertion/extraction
- Packet CRC generation
- RX packet validation - LNG/SEQ/CRC fields
- Retry buffer and retry sequence control
- Flow packets and error packet filtering
- Error Response packet handling

The XHMC defines the XHMC Transaction Layer Interface as the communication protocol between any user layer implementation and the XHMC Transaction Layer. The datapath of the XHMC Transaction Layer Interface is a FLIT based bus protocol with side band control signals. The bus width is configurable in the unit of FLIT to accommodate the bandwidth and system clock frequency requirement.

In the transmission direction, the XHMC stores the packets from the user layer in the embedded FIFO (output buffer). Besides link retry, two scenarios prevents the packet at the FIFO head to be forward to the downstream layers along the datapath. First, insufficient memory/buffer space at the HMC side to receive an incoming packet. The controller holds the packet forwarding when the available device token is less than 32 to guarantee that the

device have enough space to receive the FLIT data along the downstream datapath. Second, the full XHMC retry buffer holds any packet from dispatching. If the number of FLIT data stored in the FIFO exceeds the programmable threshold, the XHMC Transaction Layer back-pressures the user logic by deasserting the ready signal.

In the receiving direction, the XHMC Transaction Layer checks the integrity of every receiving HMC packet. If any packet error is on the link, the transaction layer logic triggers the link retry sequence and pauses the packet forwarding in the transmission datapath. The transaction layer discards all flow packets and the packets with error. Any unsolicited error response packet will be discarded as well and its error status information is written into an error response FIFO to be read through error response interface. On the XHMC Transaction Layer Interface, user logic should only receive valid HMC response packets.

XHMC Link Layer

The XHMC Link Layer acts as an intermediate stage between the transaction layer and the physical layer. In the transmission direction, the primary responsibility is to generate flow control packets and sends them to the link partner when no data packet is available for transmitting. In receiving direction, the HMC Link Layer detects the packet boundary and passes the boundary information along with packet data to the transaction layer.

XHMC Physical Layer

The XHMC Physical Layer bridges the link layer logic and the Xilinx device transceivers (GT). At the link layer to physical layer interface, the data is in the format of the HMC FILT, while the actual data communication on the HMC link is in the format of raw data stream which are distributed in multiple independent transceiver lanes. The physical layer conducts the following functions:

- Link training with TS1 sequence
- Lane polarity detection and correction
- Data scrambling/ descrambling
- De-skew among multiple GT lanes
- Lane order detection and dynamic reverse the lane order if configured
- Stripe/De-striping HMC packet from/to GT lanes

GT wrapper

The XHMC IP integrates a GT wrapper which instantiates GT channels and the associated common wrapper. The number of GT channels to be included depends on the user's selection of HMC link mode.

The GT channel is the electrical sub-block that performs serial to parallel converter and clock and data recovery from high speed HMC serial interface. The GT channel in the XHMC

IP is operated in raw mode and buffer-bypass mode to reduce datapath latency. In addition, the HMC Controller IP requires the HMC reference clock share with the same source as the reference clock of the connected the HMC device.

CSR Configuration Module

The CSR configuration module provides control and status signals that allow application processor to control and check the status of the HMC Controller during the HMC Controller operation. The CSR configuration module is assessable through AMBA AXI4-Lite interface as specified in Table 2-11. The traffic statistic counters report on the healthiness of the HMC link so that you can do re-training or warm-reset for the HMC link if a high error rate is detected. You may also be able to put the HMC link into sleep mode to reduce power consumption in the period that no HMC memory access is expected. Detail CSR registers are specified in Chapter 2.

Feature Summary

- Fully compliant with the HMC Consortium specification rev1.1
- Physical Layer
  - Data Scrambling/descrambling and alignment
  - Lane de-skew
  - Lane reversal and polarity detection/inversion
- Link Layer
  - FRP/RRP/SEQ insertion/extraction
  - Flow packets (NULL/PRET/TRET/IRTRY) generation
  - Link retry buffer and retry sequence control
- Transaction Layer
  - Xilinx Transaction Layer interface (Native FLIT interface)
  - Flow control and RTC token field insertion
  - CRC generation and checking
  - Error Response packet handling
- User Layer
  - AXI4-MM user interface
  - HMC packet framing/de-framing
  - TAG management and lookup

    - Response reordering
- Support 10Gbps, 12.5Gbps, and 15Gbps serial link interface
- Support both full-width link (16-lanes) and half-width link (8-lanes)
- Programmable 2-FLITs to 12-FLITs HMC Core datapath width for clock speed and gate count optimization.

Applications

Applications of this IP include:

- HPC/Server -VPU/GPU
- Graphics
- Networking systems
- Test equipment

Product Specification

Standards

The XHMC IP revision 1.0 is fully compliant with the industry standard Hybrid Memory Cube Specification 1.1 [Ref 2].

Performance and Resource Utilization

For full details about performance and resource utilization, visit the Performance and Resource Utilization web page.

Port Descriptions

This section provides detailed port descriptions for the following interfaces:

- Clock Reset Interface
- HMC Device Interfaces
- User Interface
  - Xilinx Transaction Layer Interface
  - AXI4-MM Core Interface
- AXI4-Lite CSR Interface
- Other Core Interfaces
- Optional Debugging Ports

Clock Reset Interface

Table 2-1 defines the clock and reset signals of the XHMC IP.

Table 2-1: **Clock Reset Interface Ports**

Port	Direction	Clock Domain	Description	Width
rst	In	Async	Hard reset for the HMC controller IP. Active-High.	1
clk_in	In	Clock	Free-running clock	1
clk_out	Out	Clock	HMC controller system clock which is generated from HMC reference clock	1
l2_rst_mmcm	Out	System Clock	The reset signal which will be asserted until clk_out is ready. Active-High.	1
l3_rst_gtdn	Out	System Clock	The reset signal which will be asserted until GT initialization is done. Active-High.	1
l4_rst_phydn	Out	System Clock	The reset signal which will be asserted until HMC initialization is done. Active-High.	1

The `clk_in` port is a free-running clock input which is used by HMC reset finite state machine, partial CSR control and partial GT reset control. The `clk_out` port is the HMC Controller system clock generated from HMC reference clock. The entire HMC Controller datapath operates in this clock domain including both AXI4-MM user interface and Xilinx Transaction Layer interface.

The `rst` port is a hard reset signal. The entire HMC Controller IP will be placed in reset state if `rst` is asserted, including the soft reset CSR register. Until both `rst` and the soft reset CSR are released, the HMC reset finite state machine starts the power-on sequence to bring up HMC link. The `l2_rst_mmcm`, `l3_rst_gtdn`, and `l4_rst_phydn` ports are the reset signals generated by reset finite state machine along the HMC power-on sequence. You should not start the HMC memory access before the assertion of `l4_rst_phydn`.

HMC Device Interfaces

The HMC device interface is the interface that HMC host controller connects to HMC device. Table 2-2 defines the ports of HMC device interfaces.

Table 2-2: **HMC Device Interfaces Port Descriptions**

Port	Direction	Clock Domain	Description	Width
refclk_p	In	Clock	Reference clock of HMC link. Differential signal	1
refclk_n	In	Clock	Reference clock of HMC link. Differential signal	1
refclk_out	Out	Clock	Single end clock generated from differential reference clock input.	1
txp	Out (Pad)	Serial Clock	Transmitting lanes to device. Differential signal	8/16

Table 2-2: **HMC Device Interfaces Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description	Width
txn	Out (Pad)	Serial Clock	Transmitting lanes to device. Differential signal	8/16
rxp	In (Pad)	Serial Clock	Receiving lanes from device. Differential signal	8/16
rxn	In (Pad)	Serial Clock	Receiving lanes from device. Differential signal	8/16
p_rst_n_link	Out	clk_in	System reset to device. If multiple HMC controllers are instantiated for the HMC links of the same device, this output neet to be AND together to generate the p_rst_n signal to the target HMC device. Active low	1
p_rst_n_device	In	Async	Feedback signal from the p_rst_n pin of the HMC device. Used to synch up the reset sequence among multiple HMC controller instances connected to the same HMC device.	1
lxrxps	In	Async	Power-reduction to device	1
lxtxps	Out	clk_in	Power-reduction from device	1

User Interface

Xilinx Transaction Layer Interface (Native FLIT Interface)

If you elect not to include the User Layer Logic during IP generation, the XHMC Controller IP provides the Transaction Layer interface for the user to access to HMC memory. The Transaction Layer interface provides a flexible and efficient HMC packet interface so you can design the application-specific interface controller to create and transmit HMC packets as needed. Table 2-3 defines the Transaction Layer interface.

Table 2-3: **Transaction Layer Interface Port Description**

Port	Direction	Clock Domain	Description	Width
tltx_valid	In	System clock	The user is driving a valid transfer. A transfer takes place when both valid and ready are asserted	1
tltx_flit_dat	In	System clock	Given the data are crescendoing to the header portion of a FLIT, the content of the CUB field must be identical to the CUBE ID of the HMC device. The fields of a FLIT tail can be any value (zeros are suggested). The controller updates the fields at the transaction layer.	128*N_FLIT
tltx_flit_vld	In	System clock	The content of the associated FLIT in tltx_flit_dat is valid HMC packet. For data flits that are marked as invalid, you must set the data to zeros.	N_FLIT
tltx_flit_sop	In	System clock	The content of the associated FLIT in tltx_flit_dat is the start FLIT of a HMC packet.	N_FLIT

Table 2-3: **Transaction Layer Interface Port Description *(Cont'd)***

Port	Direction	Clock Domain	Description	Width
tltx_flit_eop	In	System clock	The content of the associated FLIT in tltx_flit_dat is the end FLIT of a HMC packet.	N_FLIT
tltx_ready	Out	System clock	HMC is ready to accept data in current cycle.	1
tlrx_valid	Out	System clock	HMC is driving a valid transfer. A transfer takes place when both valid and ready are asserted.	1
tlrx_flit_dat	Out	System clock	The HMC data bus for core to transfer HMC packet to the user logic.	128*N_FLIT
tlrx_flit_vld	Out	System clock	The content of the associated FLIT in tlrx_flit_dat is valid HMC packet	N_FLIT
tlrx_flit_sop	Out	System clock	The content of the associated FLIT in tlrx_flit_dat is the start FLIT of a HMC packet	N_FLIT
tlrx_flit_eop	Out	System clock	The content of the associated FLIT in tlrx_flit_dat is the end FLIT of a HMC packet	N_FLIT
tlrx_ready	In	System clock	The user logic is ready to accept data in current cycle	1

The Transaction Layer interfaces provides HMC packet based access. The data transferred across the interface is in HMC packet format. The user application has to be responsible for HMC packet framing/deframing and manage and identify the traffic based on HMC TAG field.

The transfer of HMC packet across the Transaction Layer interfaces must be the consecutive FLITs and could cross multiple beats. A HMC packet can be started from any FLIT and ended at any FLIT of the bus. The SOP/EOP/VLD flags indicated the boundary information of packet streaming. No gap in middle of packet is allowed, which means VLD flag must be asserted starting from SOP FLIT all the way to EOP FLIT. Zero or more FLITs gap is possible between packets. There is no limitation for the number of packets to be transferred in the same beat.

AXI4-MM Interface

If HMC AXI4-MM Mode is selected, the Xilinx HMC Controller IP provides standard AXI4-MM user interfaces for the user logic to access the HMC memory. The AXI4-MM interface includes five independent channels:

- **Write Address Channel Interface**: The interface through which the write command/address of request from the user is delivered to the HMC device.
- **Write Data Channel Interface**: The interface through which the write data of request from the user is delivered to the HMC device.
- **Write Response Channel Interface**: The interface through which write status generated by the HMC device responses to the write request is transmitted to the user.

The write response interface functions only when non-posted write requests are sent by the user in write request interface.

- **Read Address Channel Interface**: The interface through which the read command/address of request from the user is delivered to the HMC device.
- **Read Response Channel Interface**: The interface through which the returned data generated by the HMC device response to the read requests are transmitted to the user.

See the AMBA AXI and ACE Protocol Specification [Ref 1] for detail AXI4 interface protocols and timing diagrams. Not all AXI4 functions are supported by HMC AXI4-MM interface. Following are the limitations while using the HMC AXI4-MM interface:

- Burst size (AWSIZE/ARSIZE) is not used. The number of bytes to transfer in each data transfer must be multiple number of FLIT (16-bytes) and can be up to the pre-defined maximum HMC payload size.
- Burst type (AWBURST/ARBURST) is fixed to "INCR" burst type.
- Lock transaction (AWLOCK/ARLOCK) is not supported.
- Memory type (AWCACHE/ARCACHE) is not supported.
- Protection type (AWPROT/ARPROT) is not supported.
- Quality of Service (AWQOS/ARQOS) is not supported.
- Region identifier (AWREGION/ARREGION) is not supported.
- Write last (WLAST) is not used. HMC AXI4-MM user interface uses AWLEN to determine the final write transfer in the burst.

Write Address Channel Interface

The Write Address Channel Interface is the interface through which the user can send WRITE request command/address to HMC device. It is also the interface for the user to send all ATOMIC requests.

Table 2-4 defines the ports in the write request interface of the core. For multiple AXI4-MM interfaces, the bus width of each signal will be multiplied by the number of AXI4-MM interfaces.

Table 2-4: **Write Address Channel Interface Port Descriptions**

Port	Direction	Clock Domain	Description	Width
axi4mm_awvalid	In	System clock	The user logic is driving a valid transfer. A transfer takes place when both AWVALID and AWREADY are asserted.	1
axi4mm_awid	In	System clock	The identifier uniquely identifies the write request in each individual WR request interface. The AWID is mapped to a HMC TAG which uniquely identifies a HMC request from HMC host controller. When HMC device responses a request by attaching the associated request TAG, the response can be directed to the corresponding originated user interface and along with the associated user-id (marked as BID) through TAG lookup. The AWID must be the sequential numbers. The user interface will be back-pressured by de-asserting the AWREADY when all HMC TAGs are used and no TAG is immediately available. For single user interface application, TAG and AWID are one-to-one mapping.	9
axi4mm_awuser	In	System clock	Sideband information that is transmitted alongside the write request address. For HMCC Rev 1.x, the AWUSER is 9-bits wide and defined as following: • awuser[5:0]: HMC write request command • awuser[8:6]: HMC device ID (CUB)	9
axi4mm_awaddr	In	System clock	Memory address of the write request. HMC address is 16-bytes aligned. The lower 4-bits in AWADDR is ignored by HMC memory.	34
axi4mm_awlen	In	System clock	WDATA burst length. The number of WDATA transfer = AWLEN + 1	8
axi4mm_awready	Out	System clock	HMC host controller can accept a request address info in current cycle	1

The Write Address Channel Interface can support up to 16 outstanding command/address before deasserting the `axi4mm_awready` signal for interface back-pressure.

Write Data Channel Interface

Table 2-5 defines the ports in the write request interface of the core. For multiple AXI4-MM interfaces, the bus width of each signal will be multiplied by the number of AXI4-MM interfaces.

The WD in width column denotes the configurable interface data bus width, in unit of FLIT, for every write data channel. WD can be any number equal to or smaller than the maximum user write data size. If WD is equal to the maximum user write data size, the logic for write data offset/packing can be eliminated to save the resources.

Table 2-5: **Write Data Channel Interface Port Descriptions**

Port	Direction	Clock Domain	Description	Width
axi4mm_wvalid	In	System clock	The user is driving a valid transfer. A transfer takes place when both WVALID and WREADY are asserted.	1
axi4mm_wid	In	System clock	Unused. The user must sends WR request data to data channel in the same sequence as the WR address channel	9
axi4mm_wdata	In	System clock	The primary interface that is used to provide the WR data which will be written into the address of HMC memory. The width of WDATA data must be an integer multiples of the FLIT size (128 bit). The multiplier must be a value of the power of 2. The IP core supports the multiplier values of 2, 4, and 8. The data also needs to be 16-bytes aligned with the associated AWADDR. WDATA is packed until the burst number specified by AWLEN is reached. 1-FLIT = 128-bits.	128*WD
axi4mm_wstrb	In	System clock	The content of the associated FLIT in WDATA bus is valid and will be transferred to HMC device. For WDATA burst transferring, the number of WSTRB in the last beat decides the number of data FLITs will be transferred. No gap is allowed between data FLITs.	WD
axi4mm_wlast	In	System clock	Boundary of a request. Unused in current implementation.	1
axi4mm_wready	Out	System clock	HMC host controller can accept a request data in current cycle	1

The `axi4mm_wready` is not asserted until one or more command/address has been received from Write Address Channel Interface.

Write Response Channel Interface

The Write Response Channel Interface returns the result of write request to the user. Every write request from the Write Address Channel Interface will produce a response in Write Response Channel Interface, for both posted and non-posted write commands. The response for non-posted write request will come from HMC device. The response for posted write request is generated by the HMC Controller internally.

Table 2-6 defines the ports of write response channel interface. For multiple AXI4-MM interfaces, the bus width of each signal will be multiplied by the number of AXI4-MM interfaces.

Table 2-6: **Write Response Channel Interface Port Descriptions**

Port	Direction	Clock Domain	Description	Width
axi4mm_bvalid	Out	System clock	HMC is driving a valid transfer. A transfer takes place when both BVALIDand BREADY are asserted.	1
axi4mm_bid	Out	System clock	The identifier indicates which AWID in WR request channel that this response is corresponding to.	9
axi4mm_buser	Out	System clock	Status information that tells the user the processing result of WR request. For HMCC Rev 1.x, the buser is 18-bits wide: • buser[5:0]: HMC response command • buser[6]: DINV field • buser[13:7]: ERRSTAT field • buser[16:14]: source Link-ID • buser[17]: request timer time-out. This bit is valid only when the UID_REORDER parameter is enabled.	18
axi4mm_bready	In	System clock	The user logic can accept a response in the current cycle.	1

Read Address Channel Interface

The Read Address Channel Interface is the interface through which the user can send READ request command/address to HMC device.

Table 2-7 defines the ports of read address channel interface. For multiple AXI4-MM interfaces, the bus width of each signal will be multiplied by the number of AXI4-MM interfaces.

Table 2-7: **Read Address Channel Interface Port Descriptions**

Port	Direction	Clock Domain	Description	Width
axi4mm_arvalid	In	System clock	The user is driving a valid transfer. A transfer takes place when both ARVALID and ARREADY are asserted.	1
axi4mm_arid	In	System clock	The identifier uniquely identifies the read request in each individual RD request interface. The ARID is mapped to a HMC TAG which uniquely identifies a HMC request from HMC host controller. When HMC device responses a request by attaching the associated request TAG, the response can be directed to the corresponding originated user interface and along with the associated user-id (marked as RID) through TAG lookup. The ARID must be the sequential numbers. The user interface will be back-pressured by de-asserting the ARREADY when all HMC TAGs are used and no TAG is immediately available. For single user interface application, TAG and ARID are one-to-one mapping.	9

Table 2-7: **Read Address Channel Interface Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description	Width
axi4mm_aruser	In	System clock	Sideband information that is transmitted alongside the write request address. For HMCC Rev 1.x, the ARUSER is 9-bits wide and defined as following: • aruser[5:0]: HMC read request command • aruser[8:6]: HMC device ID (CUB)	9
axi4mm_araddr	In	System clock	Memory address of the read request. HMC address is 16-bytes aligned. The lower 4-bits in ARADDR is ignored by HMC memory	34
axi4mm_aready	Out	System clock	The HMC host controller can accept a request address info in the current cycle	1

Read Response Channel Interface

The read response channel interface returns the data of read request to the user. Every read request from read address channel interface will have a read response from HMC device. For multiple AXI4-MM interfaces, the bus width of each signal will be multiplied by the number of AXI4-MM interfaces.

Table 2-8 defines the ports of the read response channel interface. The RD in width column denotes the configurable interface data bus width, in unit of FLIT, for every read data channel. RD can be any number either equal to or smaller than the maximum user read data size. If RD is equal to the maximum user read data size, the logic for read data offset/unpacking can be eliminated to save the resources.

Table 2-8: **Read Response Channel Interface Port Descriptions**

Port	Direction	Clock Domain	Description	Width
axi4mm_rvalid	Out	System clock	HMC is driving a valid transfer. A transfer takes place when both RVALID and RREADY are asserted.	1
axi4mm_rid	Out	System clock	The identifier indicates which ARID in RD request channel that this response is corresponding to.	9
axi4mm_ruser	Out	System clock	Status information that tells the user the processing result of RD request. For XHMC Rev 1.x, the ruser is 18-bits wide: • ruser[5:0]: HMC response command • ruser[6]: DINV fielddd • ruser[13:7]: ERRSTAT field • ruser[16:14]: source Link-ID • ruser[17]: request timer time-out. This bit exists only when the UID_REORDER parameter is enabled.	18

Table 2-8: **Read Response Channel Interface Port Descriptions *(Cont'd)***

Port	Direction	Clock Domain	Description	Width
axi4mm_rdata	Out	System clock	The primary interface that is used to provide the RD data which has been read from the address in HMC memory. The width of RDATA data must be an integer multiples of the FLIT size (128 bit). The multiplier must be a value of the power of 2. The IP core supports the multiplier values of 2, 4, and 8. The data also needs to be 16-bytes aligned with the associated ARADDR. The payload of HMC response packet is unpacked and sent to the user logic through RDATA interface. 1-FLIT = 128-bits.	128*RD
axi4mm_rlast	Out	System clock	Indicates the last transfer in a read response burst.	1
axi4mm_rready	In	System clock	The user logic can accept a response in current cycle.	1

AXI4-Lite CSR Interface

The CSR interface allows users to program management parameters and statistic counters in the HMC controller. Note that user should follow and complete the XHMC controller power-on sequences specified in HMC Controller Initialization Sequence in Chapter 3 before accessing any other CSR registers.

Table 2-9 defines the ports of AXI4-Lite CSR interface of the core. For write operation, XHMC controller does not take action until both write address and data are presented on the interface.

Note: The `s_axi_aresetn` AXI4-Lite reset signal affects only the AXI4-Lite interface control logic.

The CSR registers are not cleared to default value by asserting AXI4-Lite reset.

Table 2-9: **AXI4-Lite CSR Interface Port Description**

Port	Direction	Clock Domain	Description	Width
s_axi_aclk	In	Clock (AXI4L clock)	AXI4-Lite clock	1
s_axi_aresetn	In	AXI4L clock	AXI4-Lite reset, active-Low	1
s_axi_awvalid	In	AXI4L clock	Write Address Channel - Valid	1
s_axi_awaddr	In	AXI4L clock	Write Address Channel - Address	10
s_axi_awready	Out	AXI4L clock	Write Address Channel - Ready	1
s_axi_wvalid	In	AXI4L clock	Write Data Channel - Valid	1
s_axi_wdata	In	AXI4L clock	Write Data Channel - Data	32
s_axi_wstrb	In	AXI4L clock	Write Data Channel - Strobe	4
s_axi_wready	Out	AXI4L clock	Write Data Channel - Ready	1

Table 2-9: **AXI4-Lite CSR Interface Port Description *(Cont'd)***

Port	Direction	Clock Domain	Description	Width
s_axi_bvalid	Out	AXI4L clock	Write Response Channel - Valid	1
s_axi_bresp	Out	AXI4L clock	Write Response Channel - Response	2
s_axi_bready	In	AXI4L clock	Write Response Channel - Ready	1
s_axi_arvalid	In	AXI4L clock	Read Address Channel - Valid	1
s_axi_araddr	In	AXI4L clock	Read Address Channel - Address	10
s_axi_arready	Out	AXI4L clock	Read Address Channel - Ready	1
s_axi_rvalid	Out	AXI4L clock	Read Response Channel - Valid	1
s_axi_rdata	Out	AXI4L clock	Read Response Channel - Data	32
s_axi_rresp	Out	AXI4L clock	Read Response Channel - Response	2
s_axi_rready	In	AXI4L clock	Read Response Channel - Ready	1

Other Core Interface

Table 2-10: **Other Core Interfaces Port Description**

Port	Direction	Clock Domain	Description	Width
errresp_valid	Out	System clock	Error-Response data is valid for the user. A transfer take place when both valid and ready are asserted	1
errresp_data	Out	System clock	Error-Response data • [9:7]: HMC CUB who generates Error-Response packet • [6:0]: ERRSTAT vector in Error-Response packet	10
errresp_ready	In	System clock	The user logic is ready to accept data in current cycle	1
hmc_int	Out	AXI4L clock	Interrupt	1

Optional Ports for Debugging

The HMC Controller core provides options to enable additional ports for transceiver (GT) related debugging. The following four sets of debugging ports can be enabled independently through HMC IP GUI:

- GT Dynamic Reconfiguration Port for Channel
- GT Dynamic Reconfiguration Port for Common
- Additional GT Control and Status Ports
- GT Ports for In-system IBERT

1. All four ports are standard GT ports. Refer to the *UltraScale Architecture GTH Transceivers User Guide* [Ref 5] and *UltraScale Architecture GTY Transceivers User Guide* [Ref 6] for more information on the GT ports.

The GT ports for In-system IBERT is a subset of the additional GT control and status ports used to connect the HMC core to In-system IBERT. To connect HMC IP to an In-system IBERT, the following four ports 4 ports are needed:

- eyescanreset
- txdiffctrl
- txprecursor
- txpostcursor

You can also use the TX control port to tune the GT TX parameters as necessary.

Register Space

The register access types are defined as following:

- **Read-Write (RW)** - Readable and writable register.
- **Read Only (RO)** - Readable register and write has no effect.
- **Write-1-Clear (W1C)** - Readable and write 1 to clear the content.

Note: All bits and addresses not defined in the register table are reserved, RO, and set to value of 0 unless otherwise stated.

Table 2-11: **XHMC Configuration Registers**

Register	Addr	Field	R/W Type	Width	Default	Description
gt_reset_ctrl	0x0		RW	32		HMC host controller configuration register.
		csr_gt_reset_all	RW	0	1	software reset all hardware logic
		csr_sys_sleep	RW	8	0	put system into sleep mode
		reserved	RW	27-9	0	
		cfg_open_loop	RW	28	0	Host controller open loop mode. Once set, the host controller ignores the token limit on the TX direction.
		cfg_retry_disable	RW	29	0	Enable or disable retry function. • 0: retry enabled • 1: retry disabled
		cfg_scrambler_disable	RW	30	0	Enable or disable scrambler. • 0: Scrambler enabled • 1: Scrambler disabled
		csr_rst_sim_enable	RW	31	0	Speedup reset sequence for simulation.
sys_status	0x4		RO	32		System status information register.
		state_gt_reset_done	RO	0	0	GT reset done on both TX and RX directions.
		state_gt_reset_tx_dn	RO	1	0	GT TX portion reset done.
		state_gt_reset_rx_dn	RO	2	0	GT RX portion reset done.

Table 2-11: **XHMC Configuration Registers *(Cont'd)***

Register	Addr	Field	R/W Type	Width	Default	Description
		state_gt_speed	RO	7-4	0	GT Speed information: • 0: 10 G • 1: 12.5 G • 2: 15 G • others: reserved
		flit_n	RO	11-8	0	The internal data bus width in FLITs.
		full_width	RO	12	1	HMC using full width.
hmc_conf	0x10		RW	32		HMC host controller configuration register.
		warm_reset	RW	0	0	Warm reset HCM host controller.
		init_continue	RW	1	0	The user starts the initialization process by asserting this field.
		max_blk_sz	RW	3-2	2	Maximum block size configuration. • 0: 32-byte • 1: 64-byte • 2: 128-byte • 3: Reserved
		dev_cube_id	RW	6:4	0	HMC device CUBE ID, which must be set before (or at the time of) asserting the init_continue bit. The value must be identical to the Cube ID value that specified in the Request Identification Register of the corresponding HMC device.
		csr_sys_retrain	RW	8	0	Force retraining the HMC link.
hmc_status	0x14		RO	32		HMC status information register.
		stat_invert_polarity	RO	15-0	0	Per Lane polarity status information. • 0: none-inverted • 1: inverted
		stat_lane_reversal	RO	16	0	Lane reversal status.
		stat_init_done	RO	17	0	Initialization done status.
		stat_deskew_done	RO	18	0	Deskew done status.
		stat_phy_reset_done	RO	19	0	PHY reset done status.

Table 2-11: **XHMC Configuration Registers *(Cont'd)***

Register	Addr	Field	R/W Type	Width	Default	Description
hmc_imr	0x18		RW	32		HMC interrupt mask register.
		retry_tmr_expire_en	RW	0	0	Enable retry timeout interrupt.
		retry_failed_en	RW	1	0	Enable retry failed interrupt.
		inbuf_ovf_det_en	RW	2	0	Enable input buffer overflow detected interrupt.
		erresp_int_en	RW	3	0	Enable error response interrupt.
hmc_isr	0x1c			32		HMC interrupt status register.
		retry_tmr_expire	W1C	0	0	Retry timeout interrupt.
		retry_failed	W1C	1	0	Retry failed interrupt.
		inbuf_ovf_det	W1C	2	0	Input buffer overflow detected interrupt.
		errresp_int	W1C	2	0	Error response interrupt indicating an error response packet is available for processing.
retry_conf	0x24			32		HMC retry configuration register.
		cfg_retry_tmout	RW	15-0	0x100	Retry timeout threshold.
		cfg_retry_limit	RW	19-16	0x4	Retry attempt limit.
retry_thrd	0x28			32		HMC retry threshold register.
		cfg_tx_irtry_n	RW	15-0	0x20	IRTRY packet transmission number. This number has to be greater than the HMC device RX IRTRY count setting.
		cfg_rx_irtry_n	RW	31-16	0x10	IRTRY packet receiving threshold. The number of HMC device TX IRTRY count has to be greater than (this threshold + HMC core data path FLIT number + 7).
crc_err_cnt	0x2c			32		Debugging error status register (CRC).
		dbg_crc_err_cnt	W1C	31-0	0	Number of packets with CRC check error. Write all 1s to clear.
dln_err_cnt	0x30			32		Debugging error status register (DLN).
		dbg_dln_err_cnt	W1C	31-0	0	Number of packets with DLN check error. Write all 1s to clear.
seq_err_cnt	0x34			32		Debugging error status register (SEQ).

Table 2-11: **XHMC Configuration Registers *(Cont'd)***

Register	Addr	Field	R/W Type	Width	Default	Description
		dbg_seq_err_cnt	W1C	31-0	0	Number of packets with sequence check error. Write all 1s to clear.
rxbuf_full_thrd	0x38			32		RX input buffer control register.
		cfg_rxbuf_full_thrd	RW	31-0	16	If the number of available entries in the buffer is equal to, or is less than this threshold, FIFO assert full status. The value must be less than or equal to the maximum number of entries of the FIFO.
txbuf_full_thrd	0x3c			32		TX output buffer control register.
		cfg_txbuf_full_thrd	RW	31-0	16	If the number of available entries in the buffer is equal to, or is less than this threshold, FIFO assert the full status. The value must be less than or equal to the maximum number of entries of the FIFO.
csr_misc_out1	0x40			32		MISC OUT register 1.
		csr_misc_out1	RW	31-0	0	Miscellaneous output register. Used to control peripheral devices, such as, the User Layer.
csr_misc_out2	0x44			32		MISC OUT register 2.
		csr_misc_out2	RW	31-0	0	Miscellaneous output register. Used to control peripheral devices, such as, the User Layer.
phy_ctrl_state	0x50			32		
		phy_ctrl_state	RO	2-0	0	Internal PHY control state machine for link training: 0: PHY_IDLE 1: PHY_DESCRAM_INIT 2: PHY_SEND_TS1 3: PHY_DESKEW 4: PHY_SEND_NULL 5: PHY_LINK_UP
dbg_ultx_flit_cnt	0x64			32		
		dbg_ultx_flit_cnt	RO	31-0	0	Number of FLITs sent from the User Layer TX port to the Transaction Layer.
dbg_tltx_flit_cnt	0x68			32		

Table 2-11: **XHMC Configuration Registers *(Cont'd)***

Register	Addr	Field	R/W Type	Width	Default	Description
		dbg_tltx_flit_cnt	RO	31-0	0	Number of FLITs sent from the Transaction Layer TX port to the Link Layer.
dbg_lltx_flit_cnt	0x6c			32		
		dbg_lltx_flit_cnt	RO	31-0	0	Number of FLITs sent from the Link Layer TX port to the Physical Layer.
dbg_ulrx_flit_cnt	0x70			32		
		dbg_ulrx_flit_cnt	RO	31-0	0	Number of FLITs sent from the Transaction Layer to the User Layer RX port.
dbg_tlrx_flit_cnt	0x74			32		
		dbg_tlrx_flit_cnt	RO	31-0	0	Number of FLITs sent from the Link Layer to the Transaction Layer RX port.
dbg_llrx_flit_cnt	0x78			32		
		dbg_llrx_flit_cnt	RO	31-0	0	Number of FLITs sent from the Physical Layer to the Link Layer RX port.
dbg_tret_tx_cnt	0x90			32		
		dbg_tret_tx_cnt	RO	31-0	0	Number of TRET packets sent to the HMC device.
dbg_pret_tx_cnt	0x94			32		
		dbg_pret_tx_cnt	RO	31-0	0	Number of PRET packets sent to the HMC device.
dbg_irtry_tx_cnt	0x98			32		
		dbg_irtry_tx_cnt	RO	31-0		Number of IRTRY packets sent to the HMC device.
dbg_tret_rx_cnt	0x9c			32		
		dbg_tret_rx_cnt	RO	31-0		Number of TRET packets received from the HMC device.
dbg_pret_rx_cnt	0xA0			32		
		dbg_pret_rx_cnt	RO	31-0		Number of PRET packets received from the HMC device.
dbg_irtry_rx_cnt	0xA4			32		
		dbg_irtry_rx_cnt	RO	31-0		Number of IRTRY packets received from the HMC device.

Table 2-11: **XHMC Configuration Registers *(Cont'd)***

Register	Addr	Field	R/W Type	Width	Default	Description
dbg_retry_reqcnt	0xA8			32		
		dbg_retry_reqcnt	RO	31-0		Number of Retry requests sent to the HMC device.
dbg_retry_respcnt	0xAc			32		
		dbg_retry_respcnt	RO	31-0		Number of Retry responses sent to the HMC device.
cfg_token_init	0xB0			32		HMC Controller initial token count for the device TX.
		cfg_token_init	RW	9-0	0	Specify the initial RX token count at the host controller side. The maximum token count is 1023. This register must be configured before asserting init_continue after hard reset.
stt_token_cnt	0xB4			32		HMC Controller remaining TX Token count.
		stt_token_cnt	RO	9-0	0	Current available token count for sending a packet from the HMC controller to the HMC device.

Designing with the Core

This chapter includes guidelines and additional information to facilitate designing with the core.

General Design Guidelines

For more information about clock, transceiver, and I/O placement rules, see:

- *UltraScale Architecture SelectIO™ Resources User Guide (UG571)* [Ref 3]
- *UltraScale Architecture Clocking Resources User Guide (UG572)* [Ref 4]
- *UltraScale Architecture GTH/GTY Transceivers User Guide (UG576, UG578)* [Ref 5] [Ref 6].

Clocking

The HMC link throughput is highly sensitive to the roundtrip latency of retry pointers and return tokens. Any datapath that has clock crossing will introduce more latency. In order to reach the best performance, Xilinx® HMC (XHMC) Controller IP implements the common clock scheme so that the entire datapath in HMC. Figure 3-1 shows the common clock scheme of HMC Controller with 15G GT wrapper. The HMC core and HMC user layer is operating in the `clk_sys` clock domain. The whole 15G GT wrapper is operating in the `clk_gt` clock domain which is synchronous and aligned to `clk_sys` clock domain. The synchronous gearbox is added to adopt the data width difference between GT channel and HMC lane in physical layer.

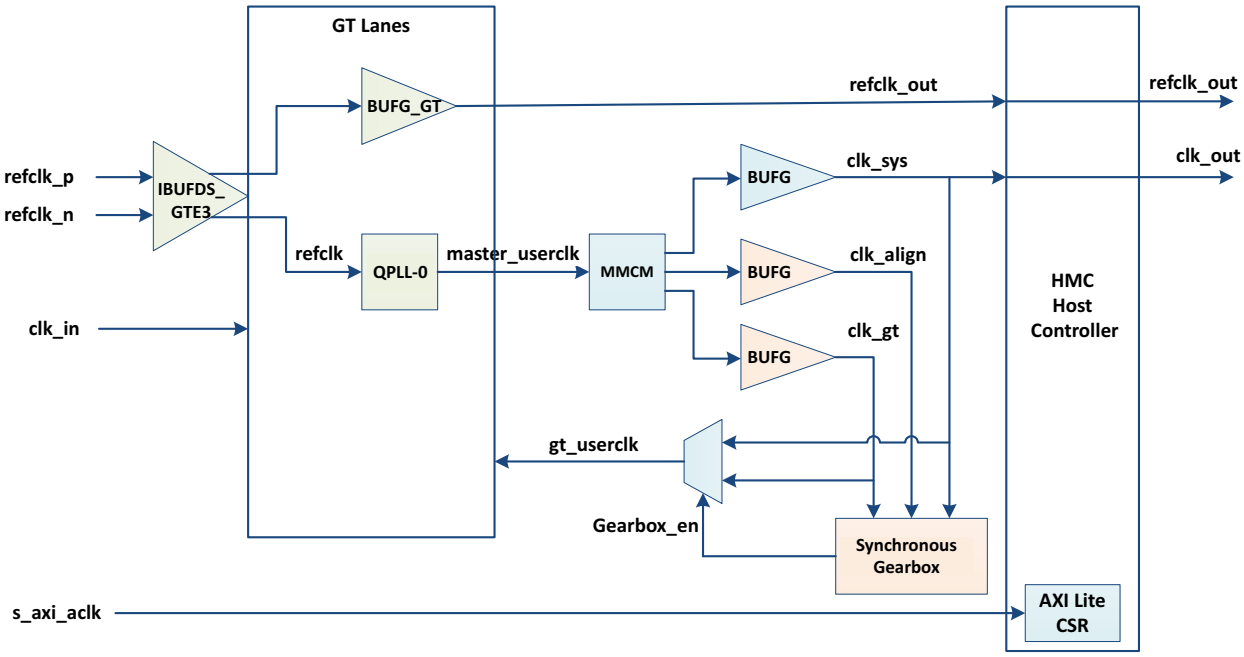

Figure 3-1: **XHMC Clock Scheme**

The XHMC host controller IP takes three input clocks, namely:

- **clk_in**: The clock serves as the free running clock for both controller logic and the GT sub-core for all reset related logic. The user must guarantee the clock is stable before release the reset of the XHMC core. For different devices, the GT sub-core sets the allowable frequency range for this clock. For details, see *UltraScale Architecture GTH/GTY Transceivers User Guide (UG576, UG578)* [Ref 5] [Ref 6].
- **refclk_p/n**: The differential clock inputs of the transceiver serves as the reference clock to the GT sub-core. XHMC requires the clock to be from the same source of the reference clock for the HMC device on the board.
- **s_axi_aclk**: The independent clock to access the internal register space of XHMC IP through AXI4-Lite interface. The CSR needs to access the registers in both `clk_in` and `clk_sys` domain. The clock domain crossing is carefully handled by the XHMC IP. The users are free to use any clock source to drive the AXI4-Lite interface. In the case that the `s_axi_aclk` is asynchronous with `clk_in` or `clk_sys`, the IP instantiates modules from the Xilinx XPM_CDC library to handle the domain crossing. The Vivado Design Suite automatic detects the modules from the XPM_CDC library and sets the corresponding timing constraints. The user logic does not need to constrain the clock relationship inside the IP.

Figure 3-1 illustrates the clocking scheme of the XHMC IP. Depends on the IP configuration, the XHMC IP generates one or three clocks for the internal logic. XHMC IP takes the user

clock output from the assigned GT master channel (TX) and uses a single MMCM to generator those clocks. The internal clocks includes:

- **clk_sys**: The major clock that drives the entire HMC transaction layer logic, as well as the optional AXI4MM user layer interface logic. You can determine the frequency of this clock with the following formula:

$$Freq_{clk\_sys} = N_{lane} \times R_{lane} \div W_{sys}$$

  Where N_{lane} stands for Number of lanes which is configurable through HMC LINK MODE user parameter. For a full-width HMC Link, N_{lane} is 16. For a half-width HMC link, N_{lane} is 8. R_{lane} is the link rate of the HMC link (i.e., 10G, 12.5G, and 15G). W_{sys} is the configured internal system data bus width in the unit of bit.

- **clk_gt**: The user clock which drives the user logic of the GT transceiver. You can calculate the frequency of the clock based on the following formula. When the frequencies of `clk_sys` and `clk_gt` are different, XHMC IP generate the GearBox logic to match the date rate between the GT boundary and the XHMC physical layer logic.

$$Freq_{clk\_gt} = R_{lane} \div W_{gt}$$

  Where R_{lane} is the HMC Lane Rate and W_{gt} is the data width configured for GT. Table 3-1 gives the typical HMC link configurations and the corresponding system clock frequency.

- **clk_align**: The internally generated clock to flag the aligned edges of `clk_sys` and `clk_gt`. This clock only exists in the design when GearBox is enabled.

When GearBox is enabled, XHMC IP sets the MMCM output clocks in the same CLOCK_DELAY_GROUP to reduce the clock skew on timing paths. By doing so, the GearBox works in a synchronous fashion.

Table 3-1: **XHMC Clock Configurations**

Link Width	Link Mode	XHMC Data Width	System Clock (MHz)	GT Data Width	GT User Clock (MHz)	GearBox
10G	Half	128*2	312.5	32	312.5	No
	Full	128*4	312.5	32	312.5	No
12.5G	Half	128*3	260.4	40	312.5	Yes
	Full	128*5	312.5	40	312.5	No
15G	Half	128*3	312.5	32	468.75	Yes
	Full	128*6	312.5	32	468.75	Yes

Resets

Xilinx HMC Controller IP provides three different types of resets:

- **Hard reset**: Hard reset is an input pin which brings the entire XHMC Controller IP to an initial state. Asserting the hard reset clears all settings, including CSR registers, controls and data flops in the controller to the default values.
- **Soft reset**: Soft reset is a CSR register which clears up whole HMC Controller datapath similar to hard reset, but all setting to CSR registers will be maintained.
- **Warm reset**: Warm reset re-initializes the HMC link state machines and tokens to allow online activity again. Pending HMC traffic in buffers will be resumed after warm reset is completed. There may be packets lost during warm reset process and causes system fail. It is recommended that the user suspends all requests and waits for all responses being flushed out in HMC Controller before asserting warm reset.

HMC Reset Sequence

XHMC provides sophisticated reset finite state machine to coordinate the reset sequence for both the Controller and device.

Figure 3-2 shows the reset FSM that XHMC implemented to bring up HMC link after hard reset or soft reset deassertion. The reset FSM sets both the controller and device in reset state at the beginning, then brings up the controller's GT transmit direction control path and followed by releasing device's reset signal. It waits until the user logic finishes the HMC device configuration and initialization, then it brings up the controller's GT receiving direction control path for clock/data recovery. After the GT receiving datapath is ready, it starts the physical layer link initialization sequence to establish the HMC link. The link layer, transaction layer, and user layer stay in reset state until HMC link is up and running. The user logic can either poll the CSR status register or monitor the `l4_rst_phydn` output pin for HMC link status. Once the link up status is asserted, the link is ready to accept memory transactions.

HMC Controller Initialization Sequence

To establish the HMC link between a XHMC controller IP and the HMC device, the user logic needs to configure both the XHMC IP and the HMC device following the correct sequence. The IP implements the initialization sequence as a hardware finite state machine. The user logic can refer the state machine as provided in the example design. To bring up the HMC link, the XHMC core must be initialized with the following sequence of steps:

1. Keep the XHMC IP in the reset state by asserting the hardware reset pin of the XHMC untile the free running clock is stable.

2. Deassert XHMC reset input. The XHMC CSR which works in the free running clock domain is now accessable. You can configure the XHMC hardware through XHMC_ad000. For example:
   - For simulation speedup, write (XHMC_ad000[31] = 1'b1)
   - To disable HMC scrambling/descrambling, write (XHMC_ad000[30] = 1'b1)
   - To set the controller in the open-loop mode (ignore the TX token limitation on the TX direction, write (XHMC_ad000[28] = 1'b1)

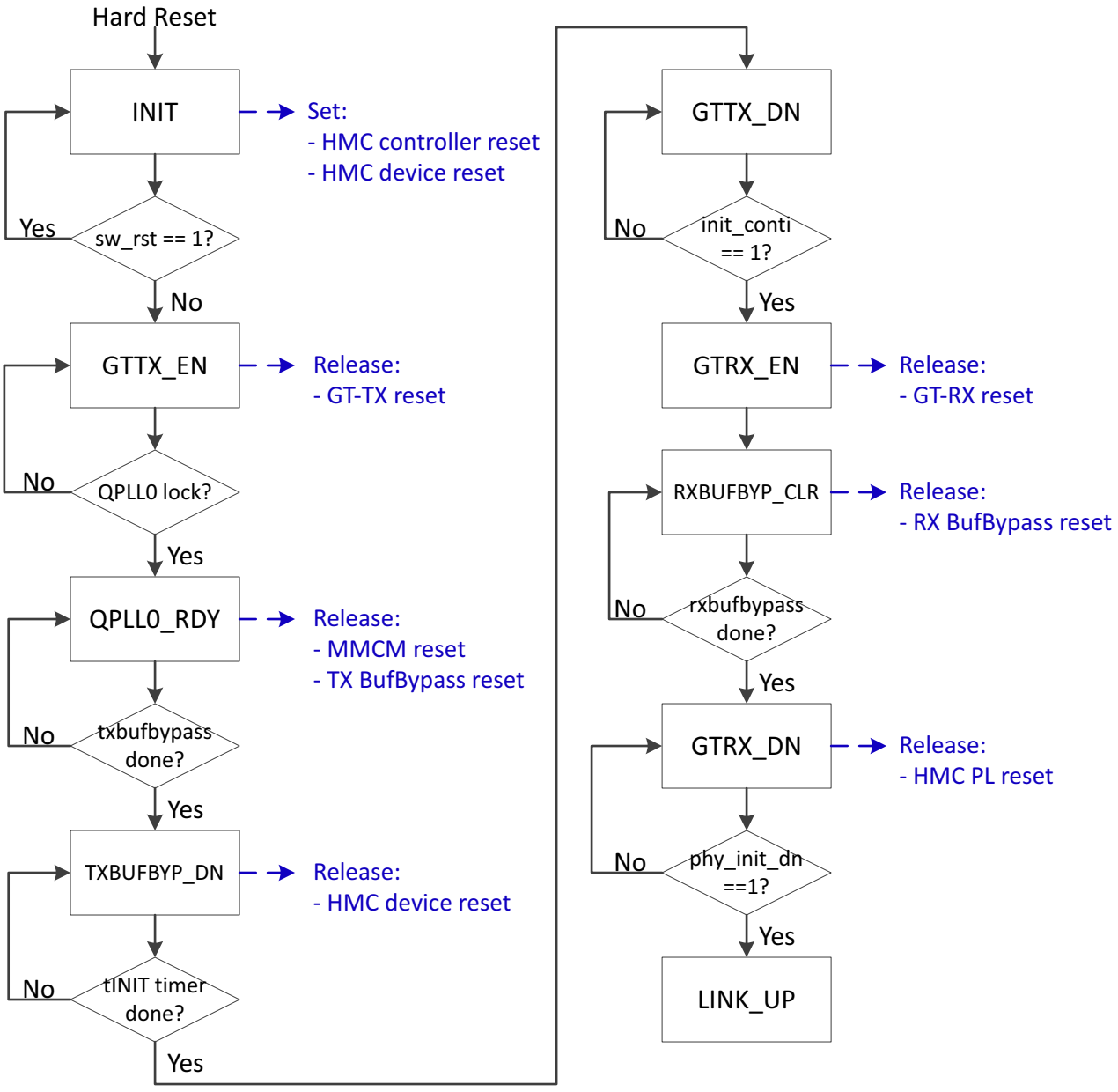

Figure 3-2: **XHMC Controller Reset Finite State Machine**

3. De-assert XHMC software reset by writing (XHMC_ad000[0] = 1'b0)
4. Polling GT TX reset status
   - Keep polling until (XHMC_ad004[1] == 1'b1)
5. Configure HMC device through I2C and enable the internal initialization of the HMC device.
6. Enable XHMC link initialization by writing (XHMC_ad010[1] = 1'b1)
7. Keep polling link status until (XHMC_ad014[17] == 1'b1)
   - HMC link is up and XHMC is ready for input requests

Power State Management

Beside the normal operation mode (active mode), the HMCC defines two low-power states for an HMC device, namely, *sleep mode*, and a minimum power state called *down mode*.

The XHMC controller IP provides the CSR entry (XHMC_ad000[1]) for the users to control the state. In order to enter sleep mode from normal operation (active mode), set the `csr_sys_sleep` (XHMC_ad000[1]) in the configuration register to 1. After detecting the configuration, XHMC toggles the power state management pin, `lxtxps`, from HIGH to LOW, which brings the connect HMC link into the sleep mode. XHMC also monitors its `lxrxps` input pins. When XHMC detects that the pin has changed to LOW, it puts the internal reset finite state machine into the sleep state as well.

To bring the link back to normal operation from sleep mode, follow this sequence of steps:

1. Make sure the controller initialization is diabled (XHMC_ad010[1] = 1'b0). The setting will hold the physical layer initialization state machine in the initial state.
2. Deassert the `csr_sys_sleep` (XHMC_ad000[1] = 1'b0). XHMC now toggles the `lxtxps` output pin to the HMC device to wake up the device. The internal reset state machine then reset the GT RX datapath.
3. Wait for the GT exist from the reset state (XHMC_ad000[0] == 1'b1).
4. Enable the controller's initialization by assert (XHMC_ad010[1] = 1'b1).
5. Wait for the link status (XHMC_ad014[17] == 1'b1).

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 9]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10]

Customizing and Generating the Core

This section includes information about using Xilinx tools to customize and generate the core in the Vivado Design Suite.

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the Vivado IP catalog.
2. Double-click the selected IP or select the **Customize IP** command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 9].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). The layout depicted here might vary from the current version.

Core Customization Parameters

The available parameters for customizing this core are shown in Figure 4-1 and described below.

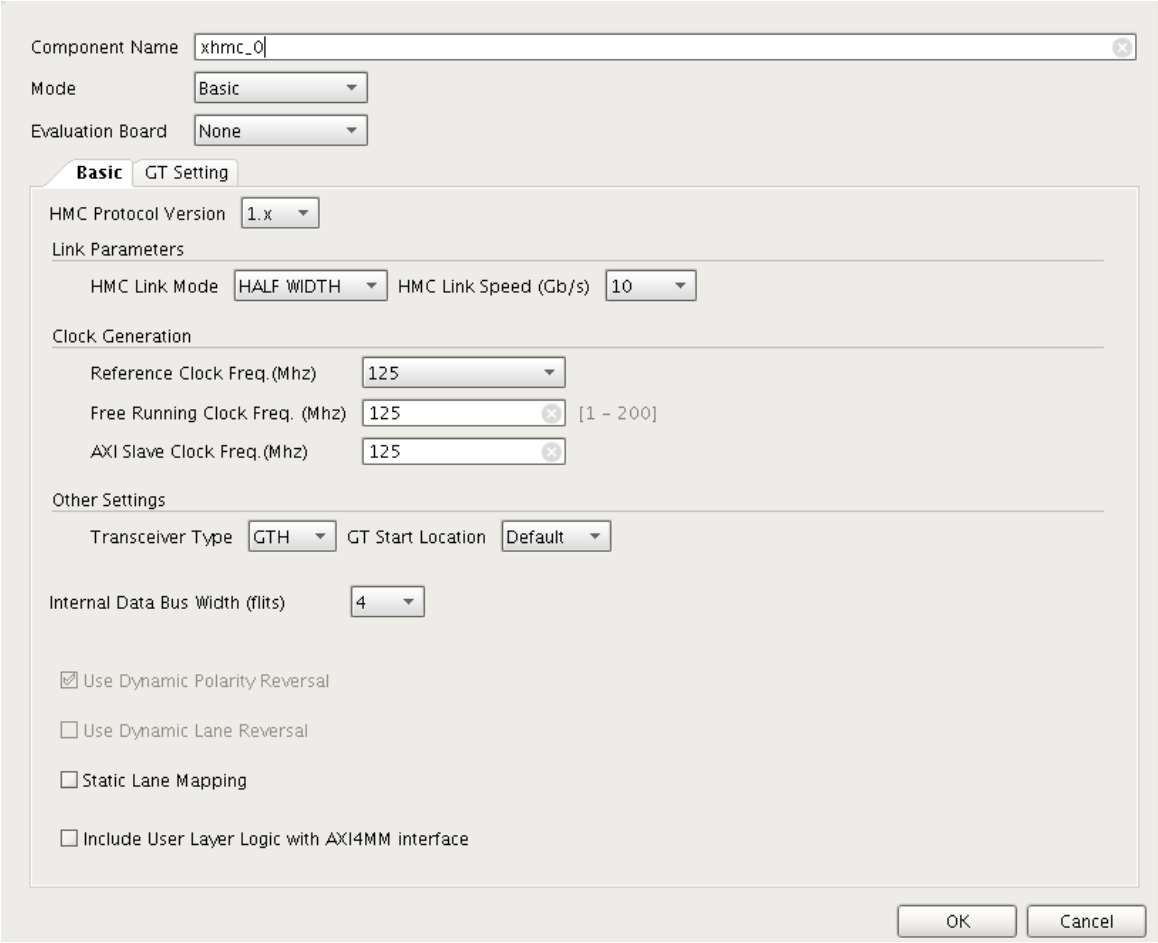

Figure 4-1: **Basic Parameters**

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). The layout depicted here might vary from the current version.

Component Name

The base name of the output files generated for the core. The name must begin with a letter and can be composed of these characters: a to z, 0 to 9, and "_."

Mode

Allows you to select the Basic or Advanced mode of the configuration of core.

Basic Tab

Click the tab to set the basic parameters for the XHMC IP.

HMC Protocol Version

Allows you to select the protocol revision of HMC deceive. The current XHMC IP supports HMCC protocol revision 1.x.

HMC Link Mode

Allows you to select the link mode of the controller. The link mode decides the number of GT lanes for the HMC link. A half-width link consists of 8 GT lanes. A full-width link uses 16 GT lanes.

HMC Link Speed

Allows you to select the supported link speed use the drop-down list. The options are 10 Gb/s, 12.5 Gb/s, and 15 Gb/s.

Reference Clock Freq

Select the reference clock frequency for the GT sub IP of the XHMC controller. The XHMC IP requires that the reference clock of the controller and the reference clock of the HMC device are from the same clock source in order to simplify the design. The XHMC only provides the valid reference clock frequencies that are supported by the HMC device in this list.

RECOMMENDED: *It is highly recommended, but not necessary, to use the same reference frequency for both the controller and the device.*

Free Running Clock Freq

Specify the clock frequency of the free running clock in the textbox. The free running clock drives the reset logic of both the controller and the GT sub core. The frequency range is 1 MHz to 200 MHz, which is set by the GT sub-core. The default value is 125 MHz. The parameter is passed to the GT Wizard IP when creating the sub-core. The XHMC controller IP uses the parameter to calculate the timing constraint in the example design. The parameter has no functional impact for the controller IP.

AXI Slave Clock Freq

Specify the clock frequency of the clock frequency of the AXI slave interface. There is no hard restriction of the frequency.

Transceiver Type

Select the target GT type. Please check the availability of the GT resource for the target device.

GT Start Location

Select the start location of the GT lanes.

The naming of the location follows the convention of Xilinx GT, which uses XY coordinate system to describe the column number and the relative position within that column. For a given device/package combination, the transceiver with the coordinates X0Y0 is located at the lowest position of the lowest available bank. The XHMC expects consecutive GT allocation within the same super logic region (SLR) and the same column, and always regard the GT lane at the lowest position as the start location.

According to the setting of Link Mode and Transceiver Type, the XHMC IP customization GUI calculates all the possible start locations for the target device/package. Refer the list in the drop-down menu for the feasibility of the GT allocation plan.

Internal Data Bus Width (flits)

Select the internal Data Bus width from the drop-down menu. The unit (flit) uses the HMCC convention, which is 128 bit.

The XHMC provides flexible data bus width configuration for you to interface with the parent design. However, the data bus width configuration has multiple folders on the system area, power consumption and performance.

CAUTION! *Be cautious when choosing a value other than the recommendation.*

As discussed in Clocking in Chapter 3, the XHMC always matches the bandwidth of the internal data bus with the link bandwidth. To achieve this, the XHMC adjust the system clock based on the value specified here and the calculated HMC link bandwidth. For the same link bandwidth, the wider the data bus, the controller uses the slower system clock and requires larger chip area. On the other hand, narrower data bus saves the area but place tighter pressure for timing closure.

See Figure 3-1 for the typical setting, which generally provides the best performance per area tradeoff.

The HMC design supports the internal data bus width from 2 to 12 filts. Starting from Vivado 2016.4, the HMC IP GUI excludes the configurations that have a high probability of causing timing failure. As a general rule, the IP GUI enables the bus width configurations which constrains the system clock to under 312.5 MHz for 20nm devices or 375 MHz for 16nm devices.

Use Dynamic Polarity Reversal

The XHMC IP automatically detects the polarity of each HMC lanes on the receiving direction. When checked, the controller reverts the polarity of a receiving lane if reversed polarity is detected during the link training. The IP generation enables the feature by default, and changing the setting is disabled in the **Basic Mode**.

Use Dynamic Lane Reversal

When checked, the XHMC includes the hardware logic to reverse the lane order on the receiving direction. For the data stream from each GT lane, the controller allocates a dedicated datapath for the lane at the Physical Layer. During the link training stage, if the controller detected the training sequence of HMC Lane 0 (0xF03x) at Lane 0 and the training sequence of HMC Lane 8/15 (0xF0Cx) at Lane 8/15, the lane order detecting logic marks the lane order as reversed. You can check the status of the lane order through the controller's CSR. If the lane reversal logic is included, the controller sets the logic mapping of the lane order dynamically.

IMPORTANT: *The lane reversal logic is in the critical path of the packet processing. Enabling the feature may have negative impact on the timing closure.*

Static Lane Mapping

When checked, the IP customization GUI displays an addition table in which you can customize the mapping between the HMC lanes and the GT lanes. The GT sub-core uses the fixed convention to order the multi-lane link; for example, Lane 0 starts at the lane at the lowest physical position. Given the HMC to Xilinx device lane mapping does not follow the same convention, you must specify the order here. The IP assigns the mapping between the GT lanes and the datapaths at the physical layer according to the table. For example, if the *N*th Lane from the HMC device is connected to the GT at the lowest location (i.e., X0Y0, GT Lane 0), the XHMC connects the *N*th datapath at the physical layer to GT Lane 0, as well.

Include User Layer Logic with AXI4MM Interface

When checked, the Vivado Integrated Design Environment (IDE) includes the User Layer Logic of the design. Use the AXI4-MM interface to access the XHMC link. If the option is not checked, the XHMC provides the native Transaction Layer Interface. When the option is checked, the Vivado IDE displays additional options for the User Layer setting for customizing the core.

User Layer setting

The User Layer setting is visible when **Include User Layer Logic with AXI4MM Interface** is checked. The settings affects to User Layer only.

Number of AXI4MM User Ports

Specify the number of AXI4MM User Ports in the text box. The current revision of the controller IP supports only a single AXI4-Memory Mapped interface port.

User-layer TX Memory Depth

Specify the depth of the packet buffer at the transmitting direction. The buffer provides temporary storage of the HMC requests that parsed from the AXI4MM transactions. The XHMC Transaction Layer takes the requests from the buffer whenever it is able to process it. The XHMC always allocates the physical memory resource to a buffer in depth 512. Reducing the value in the text box does not save the area but provides earlier back-pressure feedback when congestions happen downstream.

AXI4MM Write Data Width (flits)

Specify the data width of AXI4MM write channel. The XHMC requires the data width is an integer multiple of a FLIT size (128 bit). The multiplier must be a value of the power of 2. The IP core supports the multiplier values of 2, 4, and 8.

Maximum Write Data Width (flits)

Specify the maximum size of the memory write transaction. The size should not exceed the maximum payload size of HMC request packet. According to the actual maximum write transaction size, user can reduce the Maximum Write Data Width here to reduce the area for the User Layer logic.

AXI4MM Read Data Width (flits)

Specify the data width of AXI4MM read channel. The XHMC requires the data width is an integer multiples of a FLIT size (128 bit). The multiplier must be a value of the power of 2. The IP core supports the multiplier values of 2, 4, and 8.

Maximum Read Data Width (flits)

Specify the maximum size of the memory write transaction. The size should not exceed the maximum payload size of HMC request packet. According to the actual maximum write transaction size, you can reduce the Maximum Write Data Width here to reduce the area for the User Layer logic.

If Write Address Aligned

When checked, the address write address must aligned to the write data width. The XHMC bypasses the alignment hardware to save area.

If Read Address Aligned

When checked, the address write address must align to the write data width. The XHMC bypasses the alignment hardware to save area.

Enable User ID Re-ordering

When checked, the User Layer reorders the response order according to the user ID. The response at the AXI4-MM interface preserves the order of the requests.

GT Setting Tab

The GT Setting tab allows you to customize a limited number of parameters of the GT sub-core.

GT Data Width

Select the GT Data Width. The XHMC uses the value selected here to configure both the User Data width and the internal data width of the GT sub-core. Combined with the GT line rate, the setting determines the user clock speed of the GT sub-core.

Enable GT Dynamic Reconfiguration Port for Channel

When checked, the generated IP expose the Dynamic Reconfiguration Port (DRP) of GT Channels. You can use the DRP to change the GT reconfiguration during the run time.

Enable GT Dynamic Reconfiguration Port for Common

When checked, the generated IP exposes the Dynamic Reconfiguration Port (DRP) of GT Commons. You can use the DRP to change the GT reconfiguration during the run time.

Enable Additional GT Control and Status Ports

When checked, the generated IP exposes the GT Control and Status Ports which are defined by Xilinx GT Wizard IP. For details, please refer to the GT Wizard product guide [Ref 7].

Output Generation

Figure 4-2 shows the directory structure of a generated core. See Chapter 5, Example Design for descriptions of the contents of each directory.

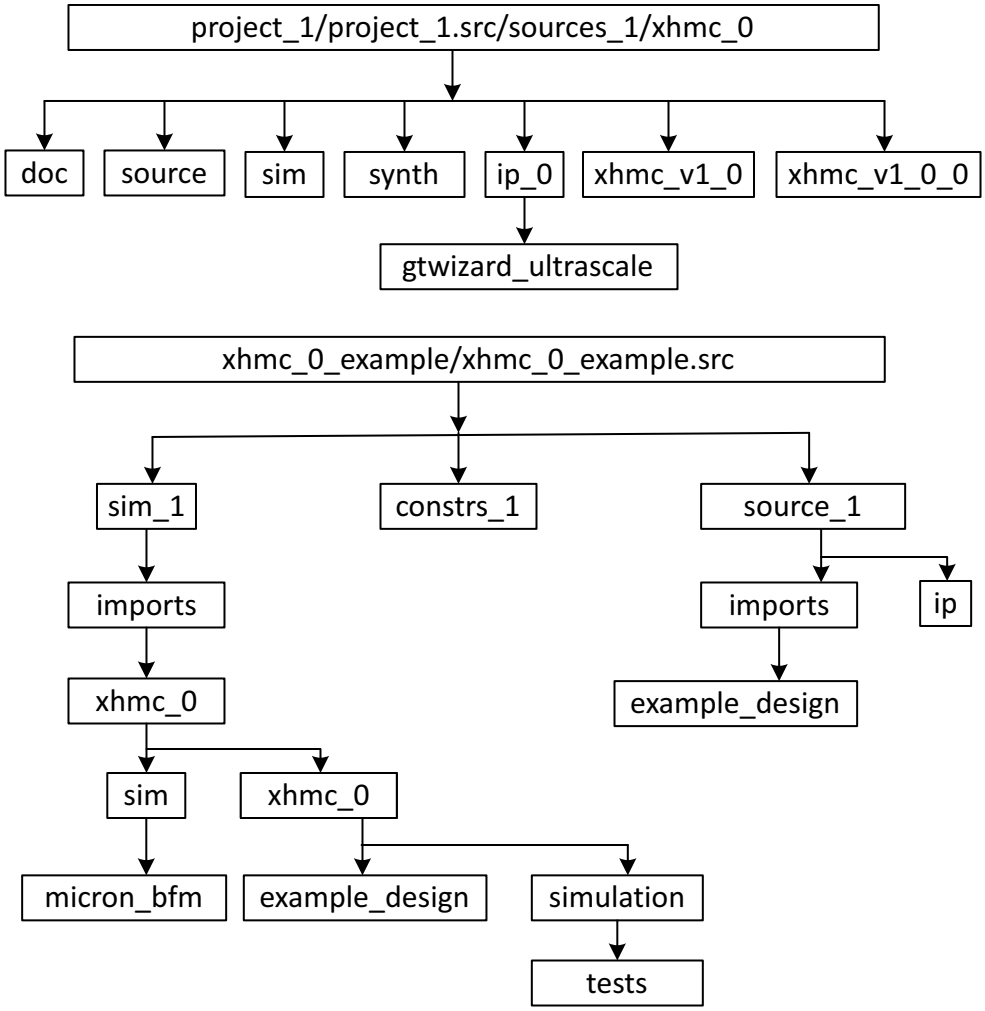

Figure 4-2: **XHMC IP Directory Structure**

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

Constraining the Core

This section contains information about constraining the core in the Vivado Design Suite.

Required Constraints

The XHMC controller IP requires minimum amount of user efforts to constraint the design.

One special GT constraint and several physical implementation contraints are provided with the example design in a Xilinx Device Constraints (XDC) file. Pinouts and hierarchy names in the generated XDC correspond to the provided example design.

To achieve consistent implementation results, an XDC containing these original, unmodified constraints must be used when a design is run through the Xilinx tools. For additional details on the definition and use of an XDC or specific constraints, see Vivado Design Suite User Guide: Using Constraints (UG903) [Ref 11].

Constraints provided with the integrated block solution have been tested in hardware and provide consistent results. Constraints can be modified, but modifications should only be made with a thorough understanding of the effect of each constraint. Additionally, support is not provided for designs that deviate from the provided constraints.

Device, Package, and Speed Grade Selections

The device selection portion of the XDC informs the implementation tools which part, package, and speed grade to target for the design. The device selection section always contains a part selection line, but can also contain part or package-specific options. An example part selection line follows:

```
CONFIG PART = XCUV190-FFGC2104-2
```

Clock Frequencies

See Chapter 3, Designing with the Core, for detailed information about clock requirements.

Clock Management

See Chapter 3, Designing with the Core, for detailed information about clock requirements.

Clock Placement

See Chapter 3, Designing with the Core, for detailed information about clock requirements.

Banking

This section is not applicable for this IP core.

Transceiver Placement

The XHMC IP takes a single reference clock input from the IO pad to drive all the GT Quads used to form the HMC link. Since the reference clock for a Quad (Q(n)) can be sourced from its own external reference clock pin paires, or a Quad that is up to two Quads below or above. Moreover, for UltraSclde devices that is using stacked silicon interconnect (SSI)

technology, the reference clock sharing is limited within its own super logic region (SLR). This generally places the requirement that the transceivers belong to be consecutive Quads in the same SLR.

I/O Standard and Placement

See Chapter 3, Designing with the Core, for detailed information about clock requirements.

Simulation

For comprehensive information about Vivado simulation components, as well as information about using supported third-party tools, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 10].

For information regarding simulating the example design, see Simulating the Example Design in Chapter 5.

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 8].

Example Design

This chapter contains information about the example design provided in the Vivado® Design Suite.

Overview of the Example Design

The XHMC IP provides a synthesizable example design along with the IP repository. To create the associated example design for a given IP configuration, right-click the IP in the **Source Window**, and select **Open IP Example Design**.

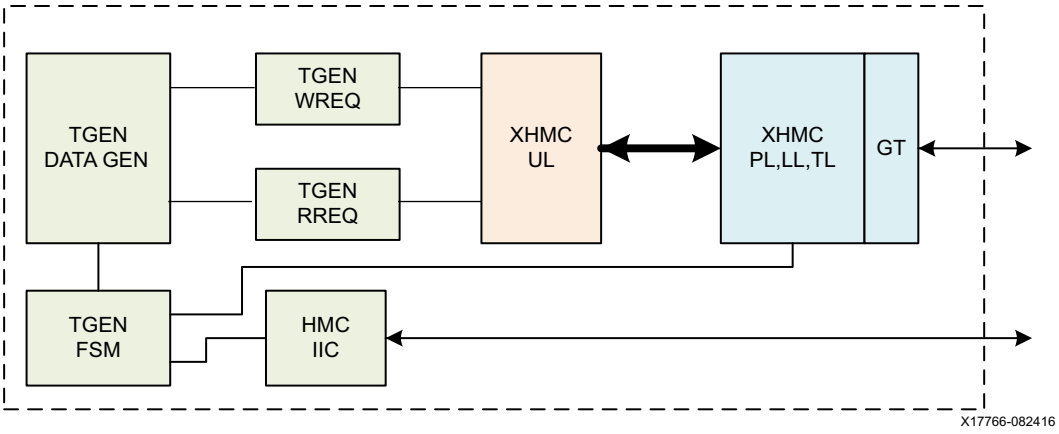

Figure 5-1: **XHMC Example Design Block Diagram**

As illustrated in Figure 5-1, a generated XHMC core consists of the GT sub-core, XHMC Physical Layer (PL), Link Layer (LL), and Transaction Layer (TL). According to the IP interface configuration, the user layer logic can be either embedded inside the XHMC core or instantiated at the example design with the source codes encrypted. Besides the XHMC core modules, the example design includes a system control state machine (TGEN FSM), a Traffic data generation (TGEN DATA GEN), AXI4MM write channel control (TGEN WREQ), AXI4MM read channel control (TGEN RREQ), a hardware state machine, and IIC interface logic for device initialization.

The example design provides the reference for XHMC structural integration and the system control flow for bringing up a HMC link. The example uses a centralized state machine to

illustrate the necessary steps to bring up the HMC link. As illustrated in Figure 5-2, the entire flow can be roughly divided into three phases:

- Reset phase
- Initialization phase
- Traffic test phase

The reset phase consists of three sub-state. For a real system, the state machine puts the system in the WT_CLKDRV state after the system reset. The WT_CLKDRV state waits for the stable on-board clock generation, i.e., the free running clock, the reference for both the device GT and the HMC device. The state is bypassed if the simulation flag is set to save the simulation time. Once the on-board clocks are stable, the state machine enters HMC_INIT states. The HMC_INIT state does nothing except release the software reset as defined in the XHMC CSR register space, which brings the XHMC IP out of its reset state. The state machine then releases the reset of the GT TX portion. The system waits for the GT to output a stable TX user clock (WT_GTTX). The XHMC core use the GT TX user clock to generate the system clock. After the user clock is stable, the system is ready for initialization phase.

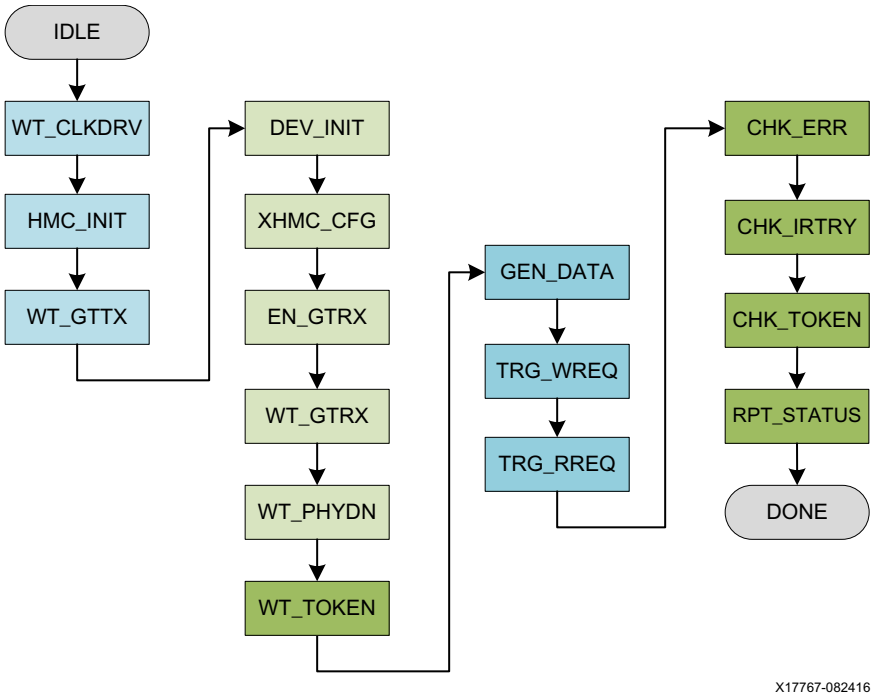

Figure 5-2: **Example Design Control State Machine**

The first step during the initialization phase is to configure both the HMC device and the XHMC IP. The example waits for the device configuration (at DEV_INIT state), and sets the necessary XHMC IP configurations in the XHMC_CFG state. There is no mandatory order between the device and controller configuration. For device configuration, you can refer to the HMC_IIC module for the detailed initialization sequence. Once the HMC device has the init_continue register set, the device starts to send PRBS pattern on the link. Only after this

point, the GT RX portion can be enabled (EN_GTRX). The state machine keeps polling the GT RX reset status in WT_GTRX. Once the GT RX is ready, the state machine assert the init_continue field in the CSR. The both sides of the HMC link are now ready to launch the handshaking protocol to establish the link. The state machine keeps polling the link status in WT_PHYDN state. Given the status shows the controller enters into the link up state, the HMC link is ready to accept memory transactions. The example design adds one extra state after the controller sees link up. Namely, the WT_TOKEN state waits until the TX token count in the CSR space matches the expected value, which is an indication that the link partners finish the initialization at the link layer.

The example design generates a simple traffic stimulus to excite the HMC link. In the GEN_DATA state, the example design randomly generate several (configurable) AXI4-MM write transactions and AXI4MM read transactions. The generated transactions are stored in the block memory. In the TRG_WREG state, the TGEN _WREQ module send all the generated WRITE requests to the controller and waits until receiving the all response packets. The state machine then enters the TGEN_RREQ state, in which the TGEN_RREQ module sends all the pre-generated read transactions to the controller. Once all the requests are received, the example design checks the status from both the traffic test and the statistic counters from the CSR space. According to the information gathered in this phase, the test ends in a different final state. Refer to Table 5-1 for the state description.

Table 5-1: **HMC Example Design Finite State Machine**

State Name	State Encoding	State Usage	State Type	Debugging Hints
FSM_IDLE	0	Wait for free running clock and releasing from reset	Initial state	Waiting for system reset or free running clock is not presenting.
FSM_WT_CLKDRV	1	Wait for external clocks ready (refclk)	System bring up	Check external reference clock configuration. If no clock configuration necessary, tie the input pin clkdrv_init_done of tgen to 1'b1.
FSM_HMC_INIT	2	Release the reset of the HMC controller through the AXI4-Lite CSR interface. The control releases the reset to the GT TX part accordingly.	Controller bring up	Check the AXI4-Lite clock, reset and connections.
FSM_WT_GTTX	3	Wait for GT finish TX reset sequence.	Controller bring up	Check the GT configuration such as free running clock frequency, reference clock configuration, GT location and GT type.

Table 5-1: **HMC Example Design Finite State Machine *(Cont'd)***

State Name	State Encoding	State Usage	State Type	Debugging Hints
FSM_DEV_INIT	4	Assert device_init_en to enable the external HMC device configuration. Wait for the completion of the configuration sequence flagged by device_init_done.	HMC device configuration	Check the IIC access to the HMC device.
FSM_HMCC_CFG0	5	Assert init_continue at the controller side. Set CUBE_ID field in the CSR (address 0x10 bit[6:4])	Controller bring up	Check the AXI4-Lite clock, reset and connections.
FSM_EN_GTRX	6	Release the GT RX portion	Controller bring up	Check the AXI4-Lite clock, reset and connections.
FSM_WT_GTRX	7	Wait for GT RX reset done	Controller bring up	Check the AXI4-Lite clock, reset and connections.
FSM_WT_PHYDN	8	Wait for controller finishing link training	Link training	Check GT configuration, i.e. Lane order. Using Chioscpoe to proble phy_ctrl_state and read device link state register can further help to determine the root cause. The encoding of the phy_ctrl _state is as following: 0: IDLE. 1: Decrambler locking. 2: Link training with TS1. 3: Lane deskew. 4: Send NULL and wait for the reception of NULL frame from all GT lanes. 5: Link up states.
FSM_WT_TOKEN	9	Wait for controller to receive all expected token from device	Sanity check	This a sanity check for the transaction layer initialization. Given stuck at this state, please check: 1. The configuration mismatch between the traffic generator and HMC device. 2. Link quality.
FSM_GEN_DATA	10	Generate test packest	Traffic test	
FSM_TRG_WREQ	11	Send Write request to HMC	Traffic test	Check the link status at the HMC device side. Cube ID mismatch can be a cause that hangs the state.
FSM_TRG_RREQ	12	Send Read Request to HMC	Traffic test	Check link status, i.e. errors

Table 5-1: **HMC Example Design Finite State Machine *(Cont'd)***

State Name	State Encoding	State Usage	State Type	Debugging Hints
FSM_CHK_CRC	13	Read CRC error count from CSR	Sanity check	
FSM_CHK_DLN	14	Read Duplicate length error count from CSR	Sanity check	
FSM_CHK_SEQ	15	Read Sequence error count from CSR	Sanity check	
FSM_CHK_IRTRY_RX	16	Read number of IRTRY received	Sanity check	
FSM_CHK_IRTRY_TX	17	Read number of IRTRY Transmitted	Sanity check	
FSM_CHK_IRTRY_REQ	18	Read number of IRTRY request transmitted	Sanity check	
FSM_CHK_IRTRY_RSQ	19	Read number of IRTRY response transmitted	Sanity check	
FSM_CHK_TOKEN	20	Read read remaining TX token count from CSR	Sanity check	
FSM_DONE	21	Dispatch to final state according to the status	Intermediate state	
FSM_FAIL	22		Test status	Traffic test failed finished with in the response packets.
FSM_ERR_DECT	23		Test status	Traffic test passed with ERROR (CRC, DLN, SEQ).
FSM_IRTRY_DECT	24		Test status	Traffic test passed with IRTRY detected.
FSM_TOKEN_MISS	25		Test status	Traffic test passed but missing tokens.
FSM_PASS	31		Test status	Traffic test passed without any error.

All the open sourced example design files are located at `prj_dir/prj_name.src/source_1/imports/example_design`. You can modify the example design and reuse the code for the XHMC integration.

Simulating the Example Design

The example design provides a quick way to simulate and observe the behavior of the XHMC controller. To enable the simulation, the XHMC IP repository includes the Micron's HMC device BFM models.

The currently supported simulators are:

- Vivado® simulator
- Cadence Incisive Enterprise Simulator (IES)
- Synopsys Verilog Compiler Simulator (VCS)
- Mentor Graphics Questa Advanced Simulator

The simulator uses the example design test bench to run the simulation. The simulation can be run as follows:

1. In the Sources Window, right-click the example project file (.xci), and select **Open IP Example Design**.

   The example project is created.

2. In the Flow Navigator, under Simulation, select **Simulation Settings**.
3. Set the Target simulator to **Incisive Enterprise Simulator (IES)** or **Verilog Compiler Simulator**.
5. In the simulator tab, select **Run Simulation > Run behavioral simulation**.
6. When prompted, click **Yes** to change and then run the simulator.

IMPORTANT: *The post-synthesis and post-implementation upon the example design is not supported directly for the current IP revision. To enable the netlist-based simulation, change the simulation structure in the example design test bench to remove any hierarchical references. In addition, the time consuming state like the device configuration through IIC bus renders extremely long simulation time and could possibly cause the simulation timeout before the real end of the simulation.*

Synthesizing and Implementing the Example Design

To run synthesis and implementation on the example design in the Vivado Design Suite environment:

1. Go to the XCI file, right-click, and select **Open IP Example Design**.

   A new Vivado tool window opens with the project name "example_project" within the project directory.

2. In the Flow Navigator, click **Run Synthesis** and **Run Implementation**.

TIP: *Click **Run Implementation** first to run both synthesis and implementation. Click **Generate Bitstream** to run synthesis, implementation, and then bitstream.*

Upgrading

This appendix is not application for this release of the core.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

Finding Help on Xilinx.com

To help in the design and debug process when using the XHMC, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the XHMC. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Technical Support

Xilinx provides technical support at the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change any section of the design labeled DO NOT MODIFY.

To contact Xilinx Technical Support, navigate to the Xilinx Support web page.

Debug Tools

Vivado Design Suite Debug Feature

The Vivado® Design Suite debug feature inserts logic analyzer and virtual I/O cores directly into your design. The debug feature also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx devices.

The Vivado logic analyzer is used with the logic debug IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

See the *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 13].

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this product guide:

1. AMBA AXI and ACE Protocol Specification
2. *Hybrid Memory Cube Specification 1.1* (HMCC)

3. *UltraScale Architecture SelectIO Resources User Guide* (UG571)
4. *UltraScale Architecture Clocking Resources User Guide* (UG572)
5. *UltraScale Architecture GTH Transceivers User Guide* (UG576)
6. *UltraScale Architecture GTY Transceivers User Guide* (UG578)
7. *UltraScale FPGAs Transceivers Wizard LogiCORE IP Product Guide* (PG182)
8. *Vivado Design Suite User Guide: Designing with IP* (UG896)
9. *Vivado Design Suite User Guide: Getting Started* (UG910)
10. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
11. Vivado Design Suite User Guide: Using Constraints (UG903)
12. *ISE to Vivado Design Suite Migration Guide* (UG911)
13. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
14. *Vivado Design Suite User Guide: Implementation* (UG904)
15. *LogiCORE IP AXI Interconnect Product Guide* (PG059)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/04/2017	1.0	Removed HMC Controller support from AXI4-MM user interface and updated port descriptions.
04/05/2017	1.0	Updated multiplier values for AXI4MM Write Data Width (flits) and AXI4MM Read Data Width (flits).
11/30/2016	1.0	Added Option Ports for Debugging section and updated Internal Data Bus Width (flits) section.
10/05/2016	1.0	Initial Xilinx Release.

Please Read: Important Legal Notices

support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.