AXI High Bandwidth Memory Controller v1.0

LogiCORE IP Product Guide

Vivado Design Suite

XILINX.

Table of Contents

Introduction

The AXI High Bandwidth Memory Controller (HBM) is an integrated IP core. This core provides access to a HBM stack with up to 16 AXI3 slave ports, each with its own independent clocking. The AXI HBM solution interfaces with JEDEC JESD235 HBM2 GEN2 memory devices. A small soft IP block is generated to configure and initialize the HBM IP as well as provide debug access in the Vivado® Hardware Manager. The IP has a configurable user interface available through the Vivado IP Catalog.

Features

- User access to the HBM stacks through AXI3 slave ports
  - 16 independent 256-bit ports
  - Optional 32-bit data bus extension
    - Can be used for either user parity protection or data width extension
  - 64 AXI IDs support per port
- 16 x 16 AXI crossbar switch per HBM stack
  - Full memory space access from all AXI ports
  - Up to 128 Gb (16 GB) directly addressable data storage in two stack configuration
  - Expansion to 32 AXI ports for dual stack configurations
- AMBA® APB 32-bit register bus access
  - Vivado® generated initialization with optional user access port
- Memory performance
  - Configurable access reordering to improve bandwidth utilization
    - Reordering transactions with different IDs
    - Honors ordering rules within IDs
    - Read after Write and Write after Write coherency checking for transactions generated by same master with same ID

- Refresh cycles are handled by the controller
  - Temperature controlled refresh rates
  - Optional hidden single row refresh option to minimize overhead
- Increase efficiency based on user access patterns
  - Flexible memory address mapping from AXI to the physical HBM stacks
  - Configurable reordering and memory controller behaviors to optimize latency or bandwidth
  - Grouping of Read/Write operations
  - Minimizing page opening activation overhead
- Performance monitoring and logging activity registers
  - Bandwidth measured at each HBM memory controller
  - Configurable sample duration
  - Maximum, minimum, and average Read and Write bandwidth is logged
- Reliability (RAS) support
  - Optional SECDED ECC
    - Partial word writes supported with Read Modify Write (RMW) operation
    - Background scan of memory for error scrubbing
    - Correctable ECC errors found are fixed and updated in memory
  - Optional parity with memory access retry due to data parity errors in Write operations
    - Parity data protection available in datapath between user logic and HBM
    - The external parity uses the 32-bit `WDATA_PARITY` and `RDATA_PARITY` buses with one parity bit per byte of data.
    - Uses Odd parity where a 1 is asserted for every data byte when the sum of the bits in the byte is an odd value.
  - Error logging registers
- Power management
  - Per memory channel clock gating
  - Per memory channel divided clock rate for reduced power
  - Power down mode supported
    - Optional self-refresh mode to retain contents of memory
    - Selectable idle timeout to self-refresh entry

- JEDEC JESD235 HBM2 GEN2 memory stack organization
  - 32 Gb density (4H stack), 64 Gb density (8H stack) depending on the device
  - 16 independent 64-bit channels
- Optional PHY only interface

IP Facts

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex® UltraScale+™ HBM Devices
Supported User Interfaces	AXI3, AMBA APB
Resources	N/A
Provided with Core	
Design Files	Encrypted RTL
Example Design	Verilog
Test Bench	Verilog AXI Traffic Generator
Constraints File	Provided
Simulation Model	Encrypted Verilog
Supported S/W Driver	N/A
Tested Design Flows[2]	
Design Entry	Vivado® Design Suite
Simulation [3][4]	Simulation is supported with Verilog Compliler Simulator (VCS), Incisive Enterprise Simulator (IES), and Questa Advanced Simulator. For supported simulator versions, see Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Release Notes and Known Issues	Master Answer Record: 69267
All Vivado IP Change Logs	Master Vivado IP Change Logs: 72775
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.
3. Behavioral simulations using verilog simulation models are supported. Netlist (post-synthesis and post-implementation) simulations are not supported.
4. Simulations in 32-bit environments are not recommended due to the limitation of addressable system memory.

Overview

The AXI High Bandwidth Memory Controller provides access to one or both the 1024-bit wide HBM stacks depending on the selected device; 64 Gb for 4H devices or 128 Gb for 8H devices. Each stack is split into eight independent memory channels, each of which is further divided into two 64-bit pseudo channels. Pseudo channel memory access is limited to its own section of the memory (1/16 of the stack capacity). Furthermore, each memory channel can operate at an independent clock rate that is an integer divide of a global reference clock.

The AXI HBM Controller has simplified the interface between HBM and CLB-based user logic in several ways. The AXI3 protocol is selected to provide a proven standardized interface. The 16 AXI ports provided match the total throughput of the HBM. Each port operates at a 4:1 ratio to lower the clock rate required in the user logic. This ratio requires a port width of 256-bits (4 × 64). Each AXI3 channel has 6-bit AXI ID port which helps in reordering transactions to achieve the required bandwidth. On the selected AXI3 channel, if the transactions are triggered using a different ID tag, then the transactions are reordered as per the AXI3 protocol. Conversely, if the selected AXI3 channel transactions are triggered using same ID tag, then the transactions are executed sequentially in the order they are triggered.

The ports are distributed across the general interconnect to reduce congestion and each port is based on an independent clock domain. This flexibility, along with each AXI port attached to its own registered column interface, reduces congestion and eases timing closure.

The 16 × 16 AXI crossbar switch is included in this core which allows each memory port to access the full HBM space by addressing all 16 pseudo channels. In the case of a two-stack system, this is extended to a 32 × 32 crossbar to allow direct access across both HBM stacks as shown in the following 4H device figure.

Figure 1: **HBM Two Stack Configuration**
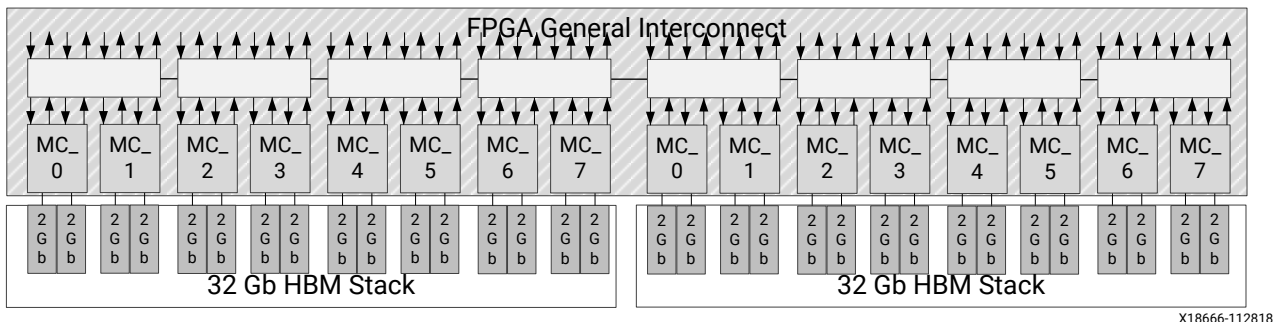
X18666-112818

Unsupported Features

The following features are not supported in the AXI3 interface or the HBM IP:

- Fixed addressing mode
- QoS signaling
- HBM GEN1 Legacy operating mode

Licensing and Ordering

This Xilinx® LogiCORE™ IP module is provided at no additional cost with the Xilinx Vivado® Design Suite under the terms of the Xilinx End User License.

Note: To verify that you need a license, check the License column of the IP Catalog. Included means that a license is included with the Vivado® Design Suite; Purchase means that you have to purchase a license to use the core.

For more information about this core, visit the HBM enabled Virtex® UltraScale+™ device product page.

Information about other Xilinx® LogiCORE™ IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

The AXI High Bandwidth Memory Controller provides user logic access to the attached HBM through the AXI3 ports distributed throughout the general interconnect. Each of the 16 AXI3 ports per stack has a throughput capacity equal to 1/16 of the total HBM bandwidth.

Each of these ports has pipeline registers to facilitate timing closure and routing of user logic. Each port can optionally address the entire HBM space (global addressing) to greatly reduce the need for any cross channel routing in general interconnect. Alternatively, non-global addressing (direct addressing) limits an AXI3 port to the associated pseudo channel with the least latency.

The eight memory controllers in a stack each have flexibility to optimize latency and utilization trade-offs for a given application by using different reordering or operating modes. Also, activity monitoring registers are provided to facilitate analysis.

Reliability is enhanced with full datapath parity checking and optional SECDED ECC protection in memory. Error logging is provided by ECC status registers.

Standards

The AXI HBM controller supports AXI3. For more information of features not included, see the Unsupported Features.

Documented registers are addressable through the AMBA APB bus protocol.

Performance

Maximum Frequencies

The following table shows the maximum frequencies for the AXI HBM controller.

Table 1: **Maximum Frequencies**

Clock	Maximum Frequency[1] (MHz)	Notes
AXI3 Port Clocks[15:0]	450	Per AXI3 port
AMBA APB Register Bus Clock	100	
Memory CLK[7:0]	900	Per HBM channel

Notes:

1. Maximum frequency supported for fastest speed grade.

Latency

The following table shows the latency for the AXI HBM controller.

Table 2: **Latency**

| Latency Component Description	Open Page – No Activate	Closed Page – Activate	Notes
	Memory Clocks	Memory Clocks	
AXI Port Latency – Global addressing disabled	90	108	Direct routing from AXI3 port to aligned Memory Channel[1].
AXI Port Latency – Global addressing enabled	110 (Minimum)	128 (Minimum)	Latency is dependent on the source and destination traffic for all the ports. Thus, a maximum latency cannot be provided.

Notes:

1. For more information, see Table 6: AXI Port Assignments in the Port Descriptions section.

Lateral AXI Switch Access Throughput Loss

Two lateral connections are provided between sets of 4 masters x 4 slaves within the switch, with one lateral connected to M0 and M1, and the other connected to M2 and M3. The shared connections limit the maximum throughput laterally to 50% of the full bandwidth, but enables global addressing from any AXI port to any portion of the HBM. For Write cycles there is a single dead cycle when switching between masters on the lateral channel. The throughput loss due to this inserted cycle depends on the block size of the writes and the extent to which throughput is switch limited, rather than HBM MC limited.

Figure 2: **HBM AXI Switch Connections**
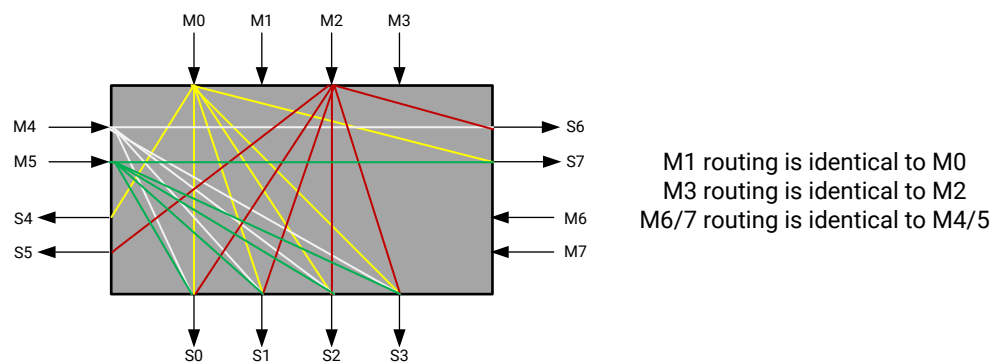
For East/West transactions within an AXI Switch instance there is no lateral performance loss. For East/West transactions which leave an AXI Switch instance there *will* be a lateral throughput performance loss.

A transaction that is stalled by the AXI Master while crossing switches will continue to reserve the lateral path, preventing its use from other channels. For example, an AXI port, reading from a memory location that requires crossing switches, is unable to receive a full data burst and de-asserts RREADY mid-transaction. This would cause the lateral path to remain claimed by this transaction and unable to process transactions from other AXI channels that require the same path.

A similar behavior may occur on a write transaction. When a write command is issued and accepted by the switch, the corresponding data path is reserved and will continue to be held until the data has been fully transmitted. The AXI protocol allows commands to be sent first and data at a later time. However, doing that can result in significant impact to overall switch performance if the data path is a lateral connection needed by another command. It is recommended that write commands are only sent when the corresponding write data is available.

Table 3: **MC Performance Impact Per Switch Instance**

| MC Transaction	Performance Impact
M0/1 going east to S2/S3	No
M0/1 going west to S4 or east to S7	Yes
M2/3 going west to S0/S1	No
M2/3 going east to S6 or west to S5	Yes

Figure 3: **Switch Lateral Connections**
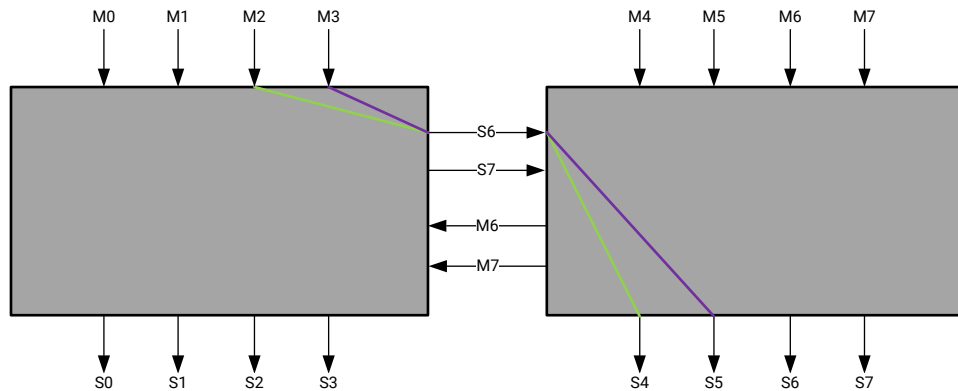
X22063-072320

A Write sequence with near 100% MC throughput results in the largest drop in throughput due to the extra switching cycle. As a result, smaller blocks have a larger percentage loss than larger blocks. The following figure demonstrates the lateral throughput loss with 32-byte and 256-byte transactions. For the 32-byte transactions two masters are each issuing three 32-byte bursts which must laterally traverse the AXI switch. For the 256-byte transactions each master is issuing two 256-byte bursts which must laterally traverse the AXI switch. In the 32-byte case there is one clock inserted between each data burst for each master which results in a total of 12 clock cycles to move 6 beats of data. In the 256-byte case there is one idle clock inserted between each 256-byte burst which results in a total of 36 clock cycles to move 32 beats of data. Because M2 and M3 only have access to one lateral slave port, the total bandwidth for these ports is split between the two masters. This results in a total efficiency for the 32-byte transactions of about 25% due to the additional clock cycles for switching between masters as well as both masters sharing a single slave port. For the 256-byte transactions these same behaviors result in approximately 44.4% efficiency.

Figure 4: **Switch Throughput Pattern**
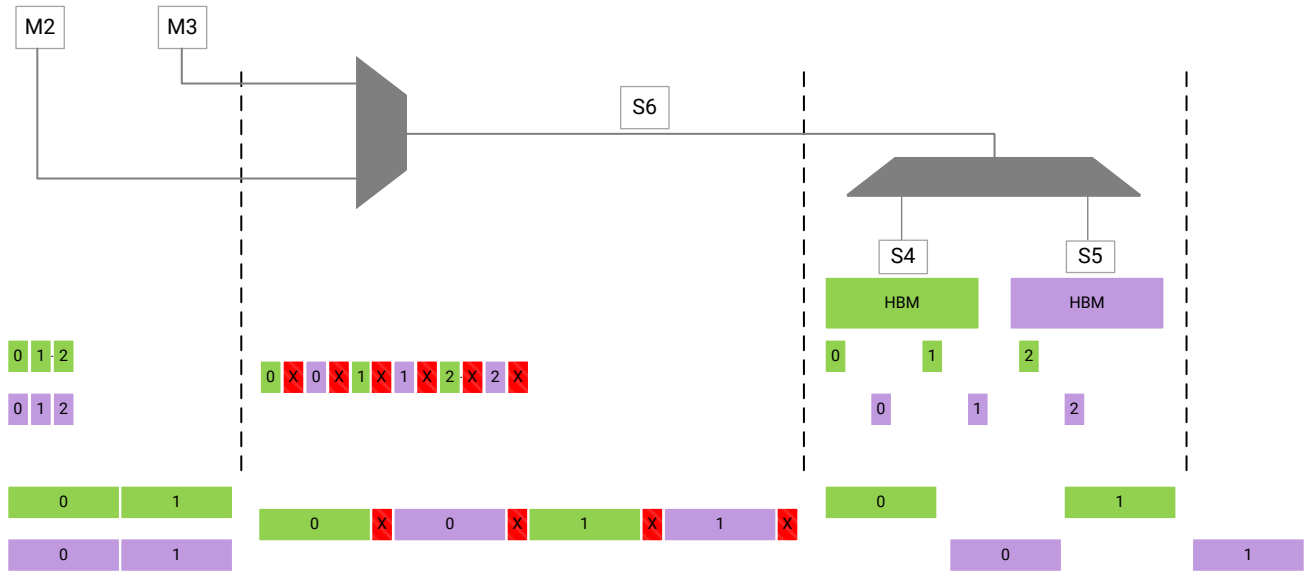
X22062-072420

Table 4: **Measured Lateral Throughput Efficiency Based on 100% Page Hit Rate Write Streams for 4H Stack**

Block Size	Switch Limited BW Pct
32 B	24.9%
64 B	33.3%
128 B	39.9%
256 B	44.4%
512 B	47%

Resource Use

Minimal CLB and block RAM resources are required to support the HBM stack controller. Primarily the logic is used for register initialization based on the values set in the Vivado® HBM wizard. Additional CLB resources are used for the clocks required for the design, the enabled AXI interface(s), and the APB register interface. The following figure shows the HBM controller initialization logic.

Figure 5: **CLB-Based Initialization Logic**

X22092-073120

Send Feedback
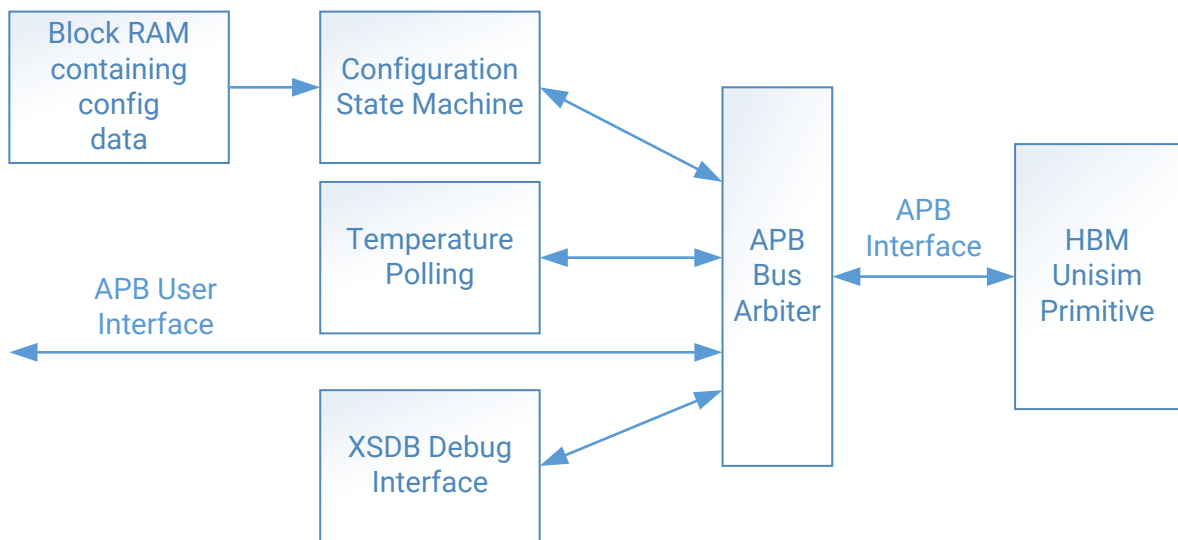
Port Descriptions

The following table shows the AXI HBM controller signals.

Table 5: **AXI HBM Controller Signals**[1]

Port Name	I/O	Description
AXI_xx_ACLK	I	Clock for the AXI Interface
AXI_xx_ARESET_N	I	Active-Low AXI Reset. This reset port should only be used prior to the start of data traffic. Using it after traffic has begun will cause the AXI interface and the memory controller to become out-of-sync. When the switch is enabled, the IP will internally connect all of the AXI_xx_ARESET_N pins together because the memory behaves as a single unit.
AXI_xx_ARADDR	I	[32:0] Read Address for 4H Stack, [33:0] Read Address for 8H Stack[2]
AXI_xx_ARBURST	I	[1:0] Address Read Burst. The fixed address burst type (2'b00) is not supported
AXI_xx_ARID	I	[5:0] Read Address ID Tag
AXI_xx_ARLEN	I	[3:0] Read Burst Length
AXI_xx_ARSIZE	I	[2:0] Read Burst Size Only 256-bit size supported (3'b101)
AXI_xx_ARVALID	I	Read Address Valid
AXI_xx_ARREADY	O	Read address Ready
AXI_xx_AWADDR	I	[32:0] Write Address for 4H Stack, [33:0] Write Address for 8H Stack [2]
AXI_xx_AWBURST	I	[1:0] Write Burst Type. The fixed address burst type (2'b00) is not supported.
AXI_xx_AWID	I	[5:0] Write Address ID
AXI_xx_AWLEN	I	[3:0] Write Burst Length
AXI_xx_AWSIZE	I	[2:0] Write Burst Only 256-bit size supported (3'b101)
AXI_xx_AWVALID	I	Write Address Valid
AXI_xx_RREADY	I	Read Ready
AXI_xx_BREADY	I	Response Ready
AXI_xx_WDATA	I	[255:0] Write Data
AXI_xx_WLAST	I	Write Last
AXI_xx_WSTRB	I	[31:0] Write Strobe
AXI_xx_WDATA_PARITY	I	[31:0] Write Data Can be used for user parity or data width expansion. User parity is calculated on a per byte basis and the calculation is Odd. For Odd parity a 1 is asserted per byte if the sum of the bits in that byte is Odd.
AXI_xx_WVALID	I	Write Valid
AXI_xx_AWREADY	O	Write Address Ready
AXI_xx_RDATA_PARITY	O	[31:0] Read Data Parity Can be used for user parity or data width expansion. User parity is calculated on a per byte basis and the calculation is Odd. For Odd parity a 1 is asserted per byte if the sum of the bits in that byte is Odd.
AXI_xx_RDATA	O	[255:0] Read Data
AXI_xx_RID	O	[5:0] Read ID Tag

Send Feedback

Table 5: **AXI HBM Controller Signals[1]** *(cont'd)*

Port Name	I/O	Description
AXI_xx_RLAST	O	Read Last
AXI_xx_RRESP	O	[1:0] Read Response
AXI_xx_RVALID	O	Read Valid
AXI_xx_WREADY	O	Write Ready
AXI_xx_BID	O	[5:0] Response ID Tag
AXI_xx_BRESP	O	[1:0] Write Response
AXI_xx_BVALID	O	Write Response Valid
APB_y_PCLK	I	APB Port Clock
APB_y_PENABLE	I	APB Enable
APB_y_PRESET_N	I	APB active-Low Reset. When this bit is asserted, it resets the entire memory interface including the controller, PHY, and memory stacks. Runs in the APB_PCLK domain and can be asserted/deasserted asynchronously with a minimum pulse width of one APB clock cycle.
APB_y_PSEL	I	APB Select
APB_y_PWRITE	I	APB Bus Direction
APB_y_PRDATA	O	[31:0] APB Read Data
APB_y_PREADY	O	APB Ready
APB_y_PSLVERR	O	APB Transfer Error
DRAM_y_STAT_CATTRIP	O	HBM Catastrophic Temperature Flag. This bit is asserted when a DRAM's temperature has exceeded 120C. When this bit is asserted, ensure to immediately disable the memory access.
DRAM_y_STAT_TEMP	O	[6:0] Temperature in Celsius. This is the worst-case scenario for the two memory stacks. When the temperature of the two memory stacks is over 5°C, the highest temperature of the two is driven out on both the DRAM_y_STAT_TEMP ports. When the temperature of the two memory stacks is below 5°C, the lowest temperature of the two stacks is driven out on both the DRAM_y_STAT_TEMP ports. The update frequency is configurable. When there are two ports in the IP for a two Stack enabled design, the data on both ports is identical because it is driven by a single source.[3]
apb_complete_0	O	Indicates initial configuration sequence for Stack-0 is complete
apb_complete_1	O	Indicates initial configuration sequence for Stack-1 is complete

Notes:

1. There are two AXI ports per memory controller enabled and one APB port per memory stack enabled.
2. In `AXI_xx_ARADDR` and `AXI_xx_AWADDR` signals:
   a. 4H Stack Bit [32], 8H Stack[33] is used to select the HBM Stack.
   b. 4H Stack Bits [31:28], 8H Stack[32:29] are used to select the AXI Port.
   c. 4H Stack Bits [27:5], 8H Stack[28:5] are used to access the actual HBM.
   d. Bits [4:0] are unused because the `AXI_xx_ARSIZE` and `AXI__xx_AWSIZE` signals are always 3'b101 (32-byte aligned).
3. This behavior change is implemented in the Vivado® Design Suite from 2018.3 release. Prior to 2018.3, lower 3-bits represented the refresh rate required for the memory based on the stack temperature.

The following table shows the AXI port assignments in the general interconnect. This includes the start address associated with each HBM channel, memory controller, and pseudo channel to which it is aligned. In non-global address mode, an AXI port can only access its associated pseudo channel. Each AXI port has a fixed address map of 2 Gb (4H Stack) or 4 Gb (8H Stack). Therefore in non-global address mode each AXI port must be mapped to the address space with its start address mentioned in the first column and ending before the start address of the next pseudo channel. In global address mode each port can access all pseudo channels but with varying performance and latency.

Table 6: **AXI Port Assignments**

Start Address (16 GB)	Start Address (8 GB)	HBM Stack	AXI Port	HBM Channel/PC	HBM Controller
0x0_0000_0000	0x0_0000_0000	Left	0	A/PC0	MC0
0x0_2000_0000	0x0_1000_0000	Left	1	A/PC1	MC0
0x0_4000_0000	0x0_2000_0000	Left	2	E/PC0	MC1
0x0_6000_0000	0x0_3000_0000	Left	3	E/PC1	MC1
0x0_8000_0000	0x0_4000_0000	Left	4	B/PC0	MC2
0x0_A000_0000	0x0_5000_0000	Left	5	B/PC1	MC2
0x0_C000_0000	0x0_6000_0000	Left	6	F/PC0	MC3
0x0_E000_0000	0x0_7000_0000	Left	7	F/PC1	MC3
0x1_0000_0000	0x0_8000_0000	Left	8	C/PC0	MC4
0x1_2000_0000	0x0_9000_0000	Left	9	C/PC1	MC4
0x1_4000_0000	0x0_A000_0000	Left	10	G/PC0	MC5
0x1_6000_0000	0x0_B000_0000	Left	11	G/PC1	MC5
0x1_8000_0000	0x0_C000_0000	Left	12	D/PC0	MC6
0x1_A000_0000	0x0_D000_0000	Left	13	D/PC1	MC6
0x1_C000_0000	0x0_E000_0000	Left	14	H/PC0	MC7
0x1_E000_0000	0x0_F000_0000	Left	15	H/PC1	MC7
0x2_0000_0000	0x1_0000_0000	Right	16	A/PC0	MC8
0x2_2000_0000	0x1_1000_0000	Right	17	A/PC1	MC8
0x2_4000_0000	0x1_2000_0000	Right	18	E/PC0	MC9
0x2_6000_0000	0x1_3000_0000	Right	19	E/PC1	MC9
0x2_8000_0000	0x1_4000_0000	Right	20	B/PC0	MC10
0x2_A000_0000	0x1_5000_0000	Right	21	B/PC1	MC10
0x2_C000_0000	0x1_6000_0000	Right	22	F/PC0	MC11
0x2_E000_0000	0x1_7000_0000	Right	23	F/PC1	MC11
0x3_0000_0000	0x1_8000_0000	Right	24	C/PC0	MC12
0x3_2000_0000	0x1_9000_0000	Right	25	C/PC1	MC12
0x3_4000_0000	0x1_A000_0000	Right	26	G/PC0	MC13
0x3_6000_0000	0x1_B000_0000	Right	27	G/PC1	MC13
0x3_8000_0000	0x1_C000_0000	Right	28	D/PC0	MC14
0x3_A000_0000	0x1_D000_0000	Right	29	D/PC1	MC14

Table 6: **AXI Port Assignments** *(cont'd)*

Start Address (16 GB)	Start Address (8 GB)	HBM Stack	AXI Port	HBM Channel/PC	HBM Controller
0x3_C000_0000	0x1_E000_0000	Right	30	H/PC0	MC15
0x3_E000_0000	0x1_F000_0000	Right	31	H/PC1	MC15

Note: The AXI Start addresses mentioned in the table are hard coded regardless of how many stacks are enabled.

AXI Port Details

For signal definitions and protocol, see the AMBA AXI Protocol Specification. When user parity is enabled, drive the correct parity value on WDATA_PARITY for Write transactions. The RDATA_PARITY bus provides the Read data parity along with the Read data. The parity calculation is done on a per-byte basis where a 1 is asserted if the sum of the bits in a byte is Odd.

Each AXI port can accept 64 read transactions and 32 write transactions.

Non-AXI Ports

A single 32-bit APB register bus with 22 bits of addressing provides access to all the documented status registers for each HBM controller stack. For naming and protocol definitions, see the AMBA AXI Protocol Specification.

Note: `apb_slv_error` is not used.

There is one port per stack to indicate the end of initial configuration sequence through the internal APB master. This port is `apb_complete_0` for Stack-0 APB interface and `apb_complete_1` for Stack-1 APB interface. You need to monitor these ports and wait until they sample high before starting any transaction on the AXI3 interface.

Data Path Error Protection

Figure 6: **Data Path Error Protection Scheme**

X22061-112918
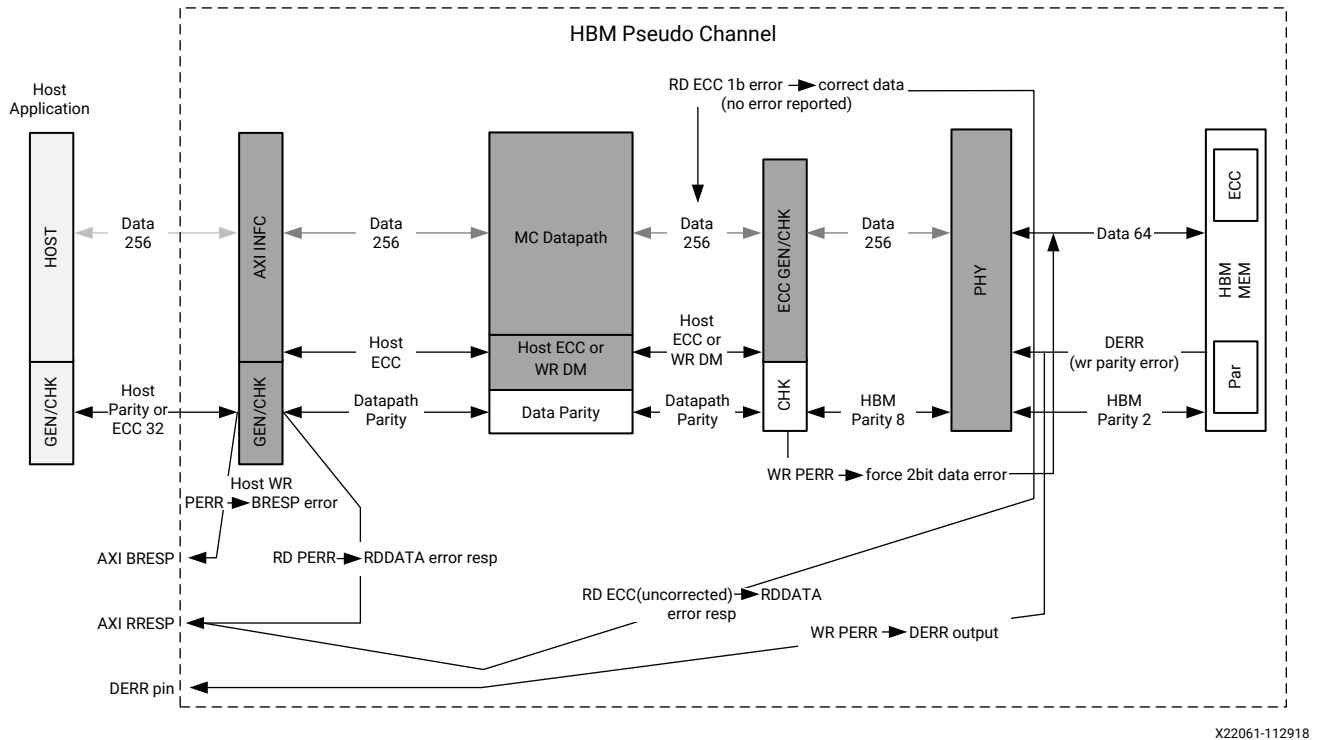
- **BRESP error:** Occurs if host parity is enabled and a parity error is detected in the AXI port for any of the associated AXI Write data. An error drives 2'b10 on the BRESP port.

- **RRESP error:** Occurs if the corresponding read data has either a parity error or an uncorrectable ECC error. The parity error check covers the path from HBM memory to AXI RDATA output port. An error drives 2'b10 on the RRESP port.

  Note: If parity retry is enabled, the error response will be asserted only if there is no read data transfer without parity errors.

- **DERR signal pulse:** Occurs if the HBM detects a parity error on Write data received. The DERR signal does not indicate exactly which AXI Write data command has caused the parity error.

- **WDATA error insertion:** Occurs if a Write data parity error is detected within the memory controller pipeline. Write data is corrupted to ensure that an ECC error will be detected at a later point when the data is read.

Send Feedback

Designing with the Core

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

Clocking

There are three clock types that must be provided to the HBM core:

- The `HBM_REF_CLK_x` drives a PLL which then generates a clock for the eight memory controllers, as well as the memory clock for the HBM stack. There is one PLL per HBM stack. This clock must be sourced from a MMCM/BUFG, or from a BUFG. The `HBM_REF_CLK_x` can be sourced from the cascading clocking sources derived from another clock. The source clock for this derived clock must come from a `GCIO` pin within the same SLR as the HBM. The clock generator driving the GCIO should have jitter less than 3 pS RMS.

- The `APB_x_PCLK` is used for the APB register port access. This can be asynchronous to the other clocks. There is one APB clock port per HBM stack. The `APB_x_PCLK` can be sourced from the cascading clocking source or `MMCM` or `GCIO` pin.

- The `AXI_xx_ACLK` is the clock for each of the AXI ports. These can be asynchronous to the other clocks. The `AXI_xx_ACLK` can be sourced from the cascading clocking source or `MMCM` or `GCIO` pins.

The global routing switch does not have a separate clock input, but rather shares one of the `AXI_xx_ACLKs`. This is automatically selected by the software to be one of the clocks in the middle of the user-selected memory controllers. This can be checked by looking at the `hbm_0.v` file. There are parameters, `CLK_SEL_xx`, and only one has the attribute TRUE.

For maximum efficiency, it is important that the AXI clock selected is the highest frequency of all of the AXI port clocks.

HBM Performance Concepts

The total performance of the HBM solution is dependent upon multiple factors:

- The HBM stack operating frequency

- The frequency of the AXI ports in the user application

- The HBM address map selection

- The global addressing mode

- The reordering options selected within the HBM IP

- The HBM stack temperature

The most important factor is how the user application logic is accessing the HBM through the AXI ports. System architects must consider how all these factors interact when evaluating the performance of the HBM memory solution in their design.

HBM Topology

The Xilinx HBM solutions are available in either 4 GB or 8 GB per stack options, with nearly all configurations containing two stacks per FPGA. This means there is a total of 8 GB or 16 GB of available memory for these dual stack devices.

The total data-bit width of an HBM stack is 1024 bits divided across eight channels of 128 bits each. Each channel is serviced by a single memory controller which accesses the HBM in pseudo channel mode, meaning two semi-independent 64-bit data channels with a shared command/address/control (CAC) bus. A 4 GB per stack device has 4 Gb per channel, and each channel has two 2 Gb or 256 MB pseudo channels. An 8 GB per stack device has 8 Gb per channel, and each channel has two 4 Gb or 512 MB pseudo channels.

Most of the HBM protocol requirements and timing can be evaluated on a pseudo channel basis, meaning that two Activate commands can be issued back to back to PC0 and PC1 without considering tRRD. If two Activate commands are issued to the same pseudo channel back to back, tRRD must first expire before the second Activate command can be issued.

The HBM always operates with a burst length of 4 in pseudo channel mode. The HBM protocol closely matches that of DDR4 memory, so many of the same performance and protocol concepts apply. The clock rate of the HBM is set in the IP configuration options in the Vivado IDE. HBM is a double data rate (DDR) memory, so the data bus toggles at twice the interface clock rate.

Raw Throughput Evaluation

It is important to understand the raw throughput of the HBM stacks and how the user logic must be designed to match this data rate. Each HBM stack has eight channels, each channel has a dedicated memory controller, each memory controller is operating in pseudo channel mode, each pseudo channel is 64 bits wide, and the data bits toggle at twice the HBM clock rate set in the IP configuration. If the HBM IP is configured with a 900 MHz clock rate, the toggle rate for a single HBM stack can be evaluated using the following formula:

(64 bits per pseudo channel) x (2 pseudo channels per memory controller) x (8 channels) x 900 MHz x 2

This results in 1,843,200 Mb per second per stack, or 230,400 MB per second per stack. Double this value to 460,800 MB per second for a dual stack device.

From the user logic perspective each AXI port for the HBM is 256 bits wide and there is one AXI port per pseudo channel in the stack. From the example above, the user logic must clock the AXI ports at 450 MHz to match the toggle rate of the HBM when it is running at 900 MHz. This is evaluated by the following formula:

(256 bits per AXI port) x (2 ports per memory controller) x (8 channels) x 450 MHz

This results in 1,843,200 Mb per second per stack, or 230,400 MB per second per stack. Double this value for 460,800 MB per second for a dual stack device with 32 AXI ports.

These are the raw HBM throughput calculations but as with all traditional volatile memories the arrays in the HBM stacks need to be refreshed to maintain data integrity. The base refresh interval (tREFI) for the HBM stacks is 3.9 µs. For 4H devices, the refresh command period (tRFC) is 260 ns and for 8H devices it is 350 ns. Adding in the refresh overhead to the raw throughput of a 4H device causes the loss of approximately 7% of peak efficiency, and 8H devices lose approximately 9%. As is the case with traditional volatile memory, tREFI decreases as the temperature increases, so more available HBM interface time is lost to refresh overhead as the stacks heat up. The base rate of 3.9 µs is used for temperatures from 0°C to 85°C. Between 85°C and 95°C tREFI is reduced to 1.95 µs.

AXI Considerations

The HBM IP requires a fixed AxSIZE of 0x5 for 32 bytes (256 bits) and it is recommended to have a minimum AxLEN of 0x1 to gain higher performance for linear accesses. Undersized AXI writes are supported by using the WSTRB port but these accesses decrease the total system performance. This is because the same amount of HBM protocol execution and interface time is used to service a full 32-byte Write as is used for an 8-byte Write. With this example the effective peak bandwidth is only 25% of the theoretical maximum.

Another AXI consideration is to make sure that the AXI Write and Read addresses are aligned to the HBM physical address space and transaction size. The lower five bits of the AWADDAR and ARADDR should be unused to ensure that accesses align to the 32-byte HBM burst boundary. This is because each individual Write or Read transaction on the HBM interface is a burst length of 4 with a DQ bus width of 64 bits, resulting in a total of 32 bytes. The AXI address space is a byte address, so 32 bytes result in 5 bits of addressing, and these are placed at the bottom of the address space. If an AXI transaction has a non-zero value in these lower address bits, the HBM controller must execute multiple commands on the HBM interface. In the case of an unaligned Read operation, two Reads are required on the HBM interface to gather the data. The controller then coalesces this to the expected payload which is delivered to the AXI Read channel. Here, the effective bandwidth is 50% or less depending on the additional protocol churn on the HBM interface. The same situation can occur with an unaligned Write, where a single Write request executes as two writes on the HBM interface while using the data mask pins to mask the unused portion of the Write payload. If ECC is enabled, an unaligned Write executes as two Read/Modify/Write (RMW) operations. When ECC is enabled the HBM controller cannot use the Data Mask pins so the first portion of the unaligned Write needs to be Read from the memory, then modified by the controller for a new data payload, and finally written back to the HBM. This process is executed twice because it requires two RMW operations to service the unaligned Write along the HBM burst boundaries.

The last AXI consideration is that the Write and Read channels operate independently while there is only a single electrical interface on the HBM stacks. This means that the HBM controller only executes Write operations and then must wait for the appropriate period of protocol time to service a bus turnaround before it can issue Read operations. For the highest possible efficiency, the user logic should minimize bus turnarounds, maximize the amount of time spent moving data in one direction before having to go the other way, or at least have a highly deterministic access pattern that minimizes protocol churn for simultaneous Write and Read streams.

HBM Address Map and Protocol Considerations

To design the most efficient system architecture and user logic to access the HBM, it is important to understand the physical address space of the HBM as well as the address map option set in the IP configuration in the Vivado IDE. Understanding of these two aspects is required to evaluate the HBM protocol execution. This is the largest factor in evaluating a system performance based on the user logic AXI access pattern. The following table defines the HBM physical address map for 4H and 8H devices.

Send Feedback

Table 7: **Physical Address Map for 4H and 8H Devices**

HBM Arrangement	4H Device (4 GB per Stack)	8H Device (8 GB per Stack)
Density per Channel	4 Gb	8 Gb
Density per Pseudo Channel	2 Gb	4 Gb
Row Address	RA[13:0]	RA[13:0]
Column Address	CA[5:1]	CA[5:1]
Bank Group Address	BA[3:0]	SID, BA[3:0]
Bank Arrangement	16 Banks 4 Bank Groups with 4 Banks	32 Banks 8 Bank Groups with 4 Banks
Total User Address Bits	23	24

The total address space of a 4H device is 32 bits and for an 8H device it is 33 bits. The following table describes the AXI addressing for these devices.

Table 8: **AXI Addressing for 4H and 8H Devices**

HBM Arrangement	4H Device (4 GB per Stack)	8H Device (8 GB per Stack)
Total Address Bits	33 total as 32:0	34 total as 33:0
Stack Select: 0 = Left 1 = Right	32	33
Destination AXI Port: 0 – 15	31:28	32:29
HBM Address Bits	27:5	28:5
Unused Address Bits	4:0	4:0

HBM operation closely follows that of traditional volatile memories and is specifically similar to DDR4. The basics of protocol operation dictate the resulting efficiency when accessing the memory array, and this must be a significant consideration along with the user AXI access pattern and how the user logic is driving the AXI channels during operation.

Like DDR4, HBM uses the concept of Banks and Bank Groups for the memory and leveraging these concepts is how to achieve a highly efficient array access pattern. 4H devices have a total of 16 Banks, arranged as 4 Bank Groups each with 4 Banks. 8H devices have 32 Banks, arranged as 8 Bank Groups with 4 Banks.

The HBM supports one active Row address per Bank. Protocol access times between Banks in different Bank Groups are lower than when accessing Banks within the same Bank Group, and the currently active Row within a Bank must be Precharged before a different Row within that Bank can be activated. When a Row is activated within a Bank, it is recommended to perform multiple Column accesses within that Row before changing the Row. Doing this is considered to result in a high page hit rate, which means higher efficiency.

By default the HBM IP is set to a Row Bank Column addressing map with the Bank Group Interleave option enabled. With these default settings, the highest order address bits are the Row address (RAx) bits, of which only one Row address per Bank can be active at a time. The middle address bits are the Bank address bits, which are displayed as BGx for Bank Groups and BAx for Bank addresses. The next lowest address range is the Column address bits which are displayed as CAx, and these are accessed by Write and Read commands.

The Bank Group Interleave option means that BG0, the least significant bit of the Bank Group addressing, is placed as the least significant user address bit of the HBM memory map (addr[5]). With the default address map, an AXI transaction with an AxLEN of 0x1 and AxADDR of 0x0 executes two discrete commands on the HBM interface. The first goes to Row 0, Bank Group 0, Bank address 0, Column 0. The second goes to Row 0, Bank Group 1, Bank address 0, and Column 0.

Having the Bank Group Interleave option with BG0 as the least significant bit is in service of sequential memory accesses. It decreases the amount of time spent waiting for protocol execution because the controller splits the accesses between two Banks in two separate Bank Groups. An AXI transaction with an AxLEN of 0x1 demonstrates this behavior, but fully leveraging these concepts for higher efficiency requires more consideration with longer transaction lengths or leveraging traffic streams mapped to Bank addresses across Bank Groups.

The default address map option is ideal for short mixed traffic because the Bank Group Interleave option alleviates some protocol exposure when the AxLEN is 0x1 or larger. This default address map option also supports the AXI reordering core which can help efficiency but might increase latency.

> ⭐ **IMPORTANT!** *For 8H HBM devices the additional Bank bit is mapped to the SID bit, which is placed at the most significant address bit within the default Row Bank Column address map.*

For 8H HBM devices the additional Bank bit is mapped to the SID bit which is placed at the most significant address bit with the default Row Bank Column address map. This is due to a limitation with the AXI Reordering core. Take notice with the default Row Bank Column address map and 8H devices to manage their traffic master addressing to use the SID bit as the most significant Bank Group bit. When a custom address map is used the SID bit can be placed as desired but the AXI Reordering Core is disabled.

When the Custom Address Map option is enabled in the HBM configuration in the Vivado IDE, more addressing options are available but the AXI Reordering Core is disabled. The Custom Address Map option makes it possible to manually assign address bits to other locations. The following sections discuss the concepts behind the Row Column Bank and Bank Row Column presets. While the presets might not be an ideal match for every application, it is still possible to manipulate any address bit to make it better suited for the use case. When the Custom Address Map option is enabled, the Bank Group Interleave setting goes to False, but once again it is possible to remap the BG0 bit to the least significant bit position of the address space to achieve the same result.

Note: When the Custom Address Map option is enabled, many of the reordering options are no longer available. Efficiency and low latency must be achieved by having well-defined traffic masters and access patterns for the physical address map.

The Row Column Bank address map option is ideal for long sequential access patterns. This is because for long transaction lengths, for instance AxLEN of 0x8 or longer, the majority of the protocol exposure is hidden by Bank Group switching. This address map is best used when the AXI transactions are only going in one direction at a time. If the traffic must change direction, those accesses should target the same Row/Bank combinations to guarantee a page hit for the Column access. This means when a long sequence of Writes is being serviced, the user logic should not issue any Read requests because this causes a bus turnaround, resulting in idle periods on the HBM interface and lowering efficiency. If the Write and Read accesses do not target the same Row/Bank combinations, the user logic should never try to switch directions until the current access sequence is complete. In this scenario, if a Write stream is executing and a Read stream is trying to access a different Row/Bank combination, the controller issues multiple Precharge and Activate commands because the new traffic stream consists only of page misses. This causes a significant amount of idle time on the HBM interface and lowers efficiency.

The Bank Row Column address map option is ideal for designs where the user logic has segmented traffic streams into separate Bank Group and Bank address ranges. This means multiple traffic streams can operate independently in their own Bank Group/address combinations and use the remaining address space as required. In a simplified example scenario, there are two streams where one is mapped to BA0 and BA1 across all Bank Groups while another is mapped to BA2 and BA3 across all Bank Groups. The first stream is long and sequential and the second stream is completely random and short. The first stream maintains high efficiency because it has a high page hit rate. The second stream with random addressing has low efficiency, but it never targets the same Bank/Row combination as the first stream, so high efficiency is maintained for the first stream. Without these considerations, if a traffic stream is long and sequential while another is random, the random stream interferes with the sequential stream and both have low efficiency.

HBM Reordering Options

The HBM IP has many reordering and refresh options available in the Reorder, Refresh, and Power Savings options page in the Vivado IDE. When the default Row Bank Column address map option is used you can select from the different reordering options. However, when the Custom Address Map option is selected, many of the reordering options are disabled, except for the Disable Dynamic Open Page option, but the Refresh and Power Savings Options remain. This is due to dependencies within the reordering core logic and the HBM controller. To make the best use of the reordering options with the default address map it is necessary to understand the functional and performance implications for each of these.

The HBM solution has two levels of command reordering. The HBM memory controller itself has a 12-entry deep command queue where it uses look ahead logic to optimize for the current state of the HBM memory arrays and the pending commands. This is enabled by default and functions like any other look ahead logic within a DDR controller.

The other level of reordering is a 64-entry deep AXI command queue which is available with the default Row Bank Column address map. This can be used by selecting the Enable Request Reordering option in the HBM configuration options in the Vivado IDE. It interacts with the Enable Coherency in Reordering, Reorder Queue Age Limit, and Enable Close Page Reorder options. When enabled the AXI command queue operates similarly to the standard reordering logic in a DDR controller. The logic targets already open Bank/Row combinations to increase the page hit rate. Accesses to different Bank Groups are promoted over accesses to the same Bank Group. If the controller can issue an Activate immediately, these are promoted over accesses which require a Precharge before they can be activated. Read/Write accesses are coalesced when possible to reduce bus turnaround.

In a case where data coherency is not guaranteed by the user traffic masters, using the Enable Coherency in Reordering option ensures that accesses with Bank/Row locality are executed in the order they are received. If the system has latency limitations when Request Reordering is enabled, the Reorder Queue Age Limit option can be decreased from the default value of 128. This value means that up to 128 new commands can be serviced before a pending command is bumped to the top of the queue. The Enable Closed Page Reorder option turns any command in the queue to an Auto-Precharge operation. This means that every Write or Read access causes a new Bank activation because each command is marked as Auto-Precharge. This option is useful if the access pattern has short bursts with highly random addressing.

When evaluating system performance through simulation or in hardware, it is important to note the implications when the AXI reordering queue is enabled. Firstly, the traffic pattern sequence needs to be sufficiently long for the controller to enter steady state operation. If your Write sequence is too short, you might see overly optimistic performance because the controller is consuming the incoming commands without any backpressure. The opposite is true for the Read sequence. If too few commands are issued the Read performance appears to be low because most of the time is being spent on protocol overhead rather than data transfer. When evaluating latency with the reordering queue enabled it is important to model an access pattern which matches the user application and is sufficiently long, especially with mixed traffic, to see if the Reorder Queue Age Limit needs to be adjusted. A good starting point for this analysis would be a run time which covers 10 refresh cycles, or about 40 µs.

Send Feedback

Within the HBM memory controller you can change the look ahead behavior as well as the open or closed page policy of the controller. By default the Look Ahead Precharge and Activate options are enabled and this follows the same concepts seen in standard DDR controllers. When these options are enabled the controller considers the current state of the memory array as well as the pending 12 commands in the queue, and inserts Precharge or Activate commands opportunistically. The Disable Dynamic Open Page option forces the controller to keep a currently active Row/Bank open until there is a discrete command to the target Bank which requires a Row change. This is the Precharge and Activate sequence. This might be helpful for low bandwidth traffic streams with a high level of locality in the memory array.

System-Level Considerations

The Global Addressing option allows for flexibility on accessing the HBM array by routing an AXI command from any ingress port which the AXI Switch then routes to the destination address. This routing is determined by the Stack Select bit, which is the most significant bit of the AXI address. The destination AXI port is then determined by the following four address bits. The AXI Port Assignments table in Port Descriptions gives a visual representation of the Stack and AXI Port mapping. As described in the Lateral AXI Switch Access Throughput Loss section, there are latency and performance implications when AXI commands are traversing the Switch. It is important to consider these limitations along with their user logic implementation and access pattern to determine how much traversal your use case can support and if Global Addressing is a viable option. If excessive traversal causes too many performance issues, it might be necessary to rearrange the traffic masters driving into the AXI ports to be closer to their destination memory controllers.

If a user application requires low latency the Global Addressing option should be disabled alongside the 64-entry deep AXI reordering queue. When Global Addressing is disabled the AXI commands are no longer routed from any ingress port to any destination. In this scenario, the AXI commands enter the ingress port and route directly to the pseudo channel attached to this AXI port. This bypasses the AXI Switch logic and enables the lowest latency path to the memory controller. Disabling AXI reordering also decreases latency because commands are directly consumed by the controller and reordered in the local 12-entry deep queue.

If AXI reordering is enabled, commands might sit in the queue for some time before they are serviced. Additionally, if a user application requires low latency, significant analysis must be performed on the AXI access pattern by the traffic masters and the HBM memory map. HBM follows the same basic protocol principles of DDR4, so with a well-defined access pattern, the HBM controller options and memory map should be reviewed to ensure the highest efficiency operation for a given use case. Each application and workload has a different solution because the optimal results are specific to that use case.

Depending on the application, the traffic masters might only issue a single AXI ID or multiple AXI IDs. If the master only generates a single AXI ID, transactions with the same ID are blocked on a channel level and execute in the order in which they are received. If multiple AXI IDs are generated, these are reordered within the AXI Reordering core if it is enabled. If the user logic does not manage AXI IDs or accesses to ensure coherency in these scenarios, enable the **Enable Coherency in Reordering** option.

Additional care should be taken when Global Addressing is enabled. The amount of time for an AXI access to navigate through the Switch is not deterministic because it is contending with all the other accesses and routing occurring in real time. In addition to the **Enable Coherency in Reordering** option, the user logic should manage this by waiting for the AXI write response signal (`BRESP`) before issuing a subsequent access that is dependent on the first access being accepted. Within the 12-entry command queue of the memory controller, coherency is always guaranteed because, as is the case with traditional DDR controllers, it does not break Write/Read order dependency for efficiency.

For user applications with small transaction sizes, highly random addressing, or if the traffic master can only generate a single AXI ID, consider using the Xilinx® Random Access Memory Attachment IP. The RAMA IP is specifically designed to assist HBM-based designs with non-ideal traffic masters and use cases. It is capable of resizing and reordering transactions to improve efficiency and bandwidth as well as generating substitution AXI IDs to prevent ID blocking if a master only generates a single AXI ID. More information on the RAMA IP can be found in the *RAMA LogiCORE IP Product Guide* (PG310).

Non-Synthesizable Traffic Generator Considerations for Workload Simulation

Many factors contribute to the bandwidth and latency of the HBM solution, all of which must be considered when designing a system. The Non-Synthesizable Traffic Generator is used to model the traffic masters and allows for quick modifications to the traffic profiles in the `axi_tg.txt` file. You can set up one traffic generator per AXI port, multiple traffic generators targeting a single port, or multiple traffic generators targeting multiple ports using the Global Addressing option. The solution offers functionality and flexibility beyond that of the example design `axi_tg.txt` file, which is an example of a simple Write and Read access pattern.

The following section outlines the options for the command input fields and how these should be used. It also highlights the types of usage that might affect the application logic and result in low performance.

Write and Read Commands

- **`txn_count`:** The transaction count field can be set to the number of times a specific command definition is to be repeated, or a specific amount of data to be transferred. To truly model a full application load in a steady-state system, specify a transaction count number rather than a fixed amount of data. Specifying a fixed amount of data can result in a short burst of activity which the traffic generator attempts to service as quickly as possible.

  If the intent is to model a busy traffic master that is constantly requesting data, or a low-bandwidth master that periodically makes requests, always use the transaction count. The rest of the available command fields should be used to pace the behavior of the master. The `start_delay` field can be used to specify the Write/Read stream bandwidth, or the `inter_beat_delay` can be used to throttle the master to decrease bandwidth or make the accesses more periodic. A low transaction count does not model steady state performance and will give inaccurate results.

- **`start_delay`:** The start delay field can be used to delay the start of a traffic generator by a specified number of clocks, or it can be used to specify the bandwidth of the traffic master. For most applications it makes sense to specify the bandwidth of the traffic master, otherwise the traffic generator attempts to service the data payload request as fast as possible which might not represent the traffic master in application. Along with the clock delay, similar functionality can be achieved by using the `WAIT` or `START_LOOP/END_LOOP` commands.

- **`inter_beat_delay`:** Use this command to model a slow or periodic traffic master.

- `wdata_pattern`, `rdata_pattern`, `wdata_pat_value`, **and** `rdata_pat_value`: Here you can test if data coherency is being maintained by the user logic when it is *not* using the AXI reordering queue by having unique settings per traffic master. This is also a good mechanism for ensuring that the traffic masters are generating addresses correctly when global addressing is enabled.

- `data_integrity`: Enable this setting to check traffic master behavior when global addressing is enabled or if the traffic masters are responsible for data coherency.

- `addr_incr_by`: When there is a specified number of transactions in the *txn_count* field, the AXI address increments by the value specified here. For ideal performance, remember that the traffic masters should always use an aligned 32-byte address to the HBM memory space to prevent unaligned accesses. Unaligned accesses can cause a 50% or 75% performance drop depending on the use case. If you are not specifying the number of transactions, set this to `auto_incr` for linear accesses through the memory map. If a random addressing mode is used, this can be set to a PRBS seed value. If a random addressing mode is used, the bandwidth of the HBM is low while the latency is high. Additional considerations need to be made for the traffic masters or the specifics of the random addressing mode if this is the intended use case for the HBM.

- `dest_id`: This can be used for more dynamic traffic generator configurations which use variables or fixed values to target specific slaves.

- `base_addr`: The starting AXI address for a traffic generator, which can be anything from the base address of the HBM pseudo channel attached to the AXI port, or any other destination address in the HBM, if global addressing is enabled. When the `high_addr` is reached, the traffic generator starts again from the `base_addr`.

- `high_addr`: The high address or the end address range for a traffic generator. When this is reached, the traffic generator starts again from the `base_addr`.

- `axi_addr`: This is one of the most important fields to understand for adequately modeling a realistic workload with HBM. If the `txn_count` is a non-zero value, the next address is calculated by the `addr_incr_by` field with `axi_addr` being the start address. It is important to ensure that the traffic masters are making aligned accesses by aligning to the HBM 32B burst boundary. For example, if the starting address is 0x1000_E0C0, this is aligned to the 32-byte boundary. If the starting address is 0x1000_E010, this is not aligned and likely subsequent accesses will not be aligned based on the `addr_incr_by` field. This means an immediate efficiency loss of 50% to 75% depending on the remaining details of the use case and IP configuration.

  - Setting this field to `random` also results in low efficiency due to the high amount of protocol overhead because every access has a very high probability of a page miss resulting in Precharge and Activate commands. Additionally if the address is completely random there is an extremely high probability of an unaligned access resulting in more efficiency loss.

- The `random_aligned` option has the same issues as the `random` setting, except the access is aligned to the HBM burst boundary. This still results in low efficiency due to the high number of page misses but every Write/Read access is fully used, which results in higher efficiency than the `random` setting. This represents one of the better random traffic masters in a system.

- The `random_unaligned` option is similar to the `random` setting, but every operation is guaranteed to be unaligned. This results in slightly lower efficiency than the `random` option.

- The `random_uniform` option also generates a low-efficiency access pattern because this is still a random address with a high probability of unaligned accesses.

- The `random_uniform_aligned` option can be used to generate the access pattern for a well-behaved random traffic master because it moves randomly through the `base_addr` and `high_addr` ranges, but always remains aligned.

- The `random_uniform_unaligned` option behaves similarly to `random_uniform_aligned` except every access is unaligned, thus further lowering efficiency.

- **`axi_len`**: AXI transaction length. Supported values are 0x0 through 0xF.

- **`axi_size`**: The AXI transaction size is expected to be 0x5.

- **`axi_id`**: This can be set to a fixed value, which results in AXI ID blocking, or it can be set to `auto_incr`, which starts at 0x0 and increments by 1 for every transaction up to a value of 0x3F before returning to 0x0.

- **`axi_burst`**: Only INCR and WRAP transactions are allowed.

Wait Commands

- **`txn_count`**: Wait commands can be used to create conditional statements or behaviors with the traffic generators. You can set this to wait for all Read responses before continuing (`all_rd_resp`), wait for all Write responses before continuing (`all_wr_resp`), wait for both all the Write and Read responses (`all_wr_rd_resp`), or a value which can be expressed in the number of AXI clock cycles or a set period of time.

- **`start_delay`**: Postpone the start of the traffic generator by a specified number of AXI clock cycles or a set period of time.

Start and End Loop Commands

- **`txn_count`**: The `START_LOOP` and `END_LOOP` commands can be used to create sophisticated and repeating access patterns. This field is set to the number of loops to perform.

- **start_delay:** This field is used to control the loop addressing. When set to
  `use_original_addr`, it uses the original starting address for the Write/Read command
  `axi_addr` field. When set to `incr_original_addr`, it increments the address by the value
  specified in the `inter_beat_delay` field below. This can be useful for masters which move
  through large address ranges.

- **inter_beat_delay:** When the `start_delay` field for the loop command is set to
  `incr_original_addr`, the address increments by the value specified in this field when the
  loop has finished.

Design Flow Steps

This section 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 IP Subsystems using IP Integrator* (UG994)

- *Vivado Design Suite User Guide: Designing with IP* (UG896)

- *Vivado Design Suite User Guide: Getting Started* (UG910)

- *Vivado Design Suite User Guide: Logic Simulation* (UG900)

Customizing and Generating the Core

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

If you are customizing and generating the core in the Vivado IP integrator, see the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) for detailed information. IP integrator might auto-compute certain configuration values when validating or generating the design. To check whether the values do change, see the description of the parameter in this chapter. To view the parameter value, run the `validate_bd_design` command in the Tcl console.

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 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) and the *Vivado Design Suite User Guide: Getting Started* (UG910).

Figures in this chapter are illustrations of the Vivado IDE. The layout depicted here might vary from the current version.

HBM Configuration Selection Tab

The following figure shows the **Customize IP** dialog box for HBM IP with the **HBM Configuration Selection** tab.

Figure 7: **AXI HBM Customize IP – HBM Configuration Selection Tab**
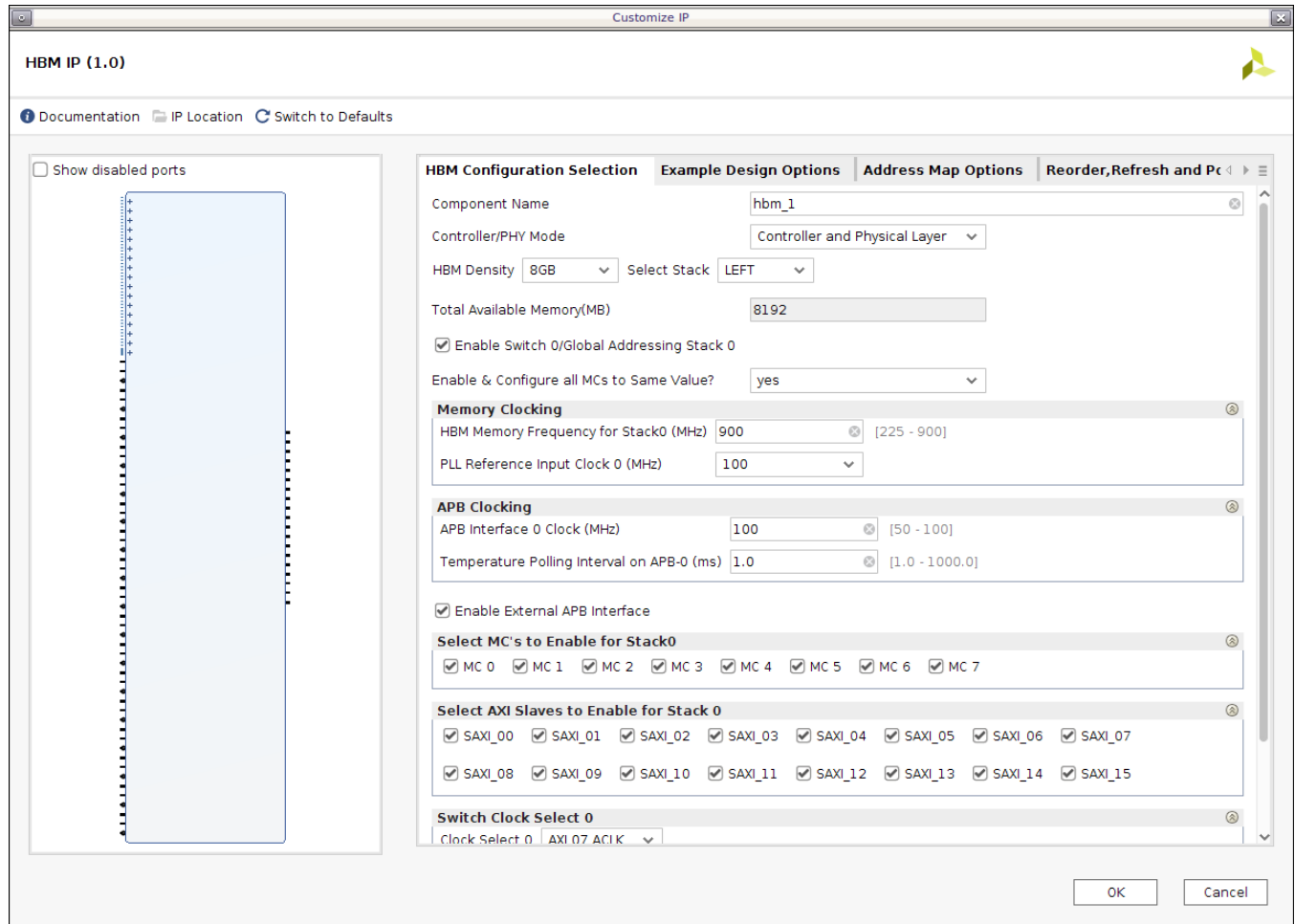
- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9, and "_".

- **Controller/PHY Mode:** Select between a complete Controller and Physical Layer option or a Physical Layer Only design.

- **HBM Density:** Selects 1 or 2 memory stacks to be configured. After selecting the memory density, the Total Available Memory field underneath updates accordingly. The settings in the MC Stack sub-tab also affects the memory value displayed.

- **Stack Select:** Selects physical location (LEFT/RIGHT) of a Single Stack configuration. After selecting the physical location of the stack, the LOC constrains are modified in the IP XDC accordingly.

- **Total Available Memory (MB):** Displays the total available memory based on the HBM Density as well as the number of memory channels enabled.

- **Enable Switch (0/1)/Global Addressing Stack (0/1):** Select this to enable Global Addressing. If the total available memory is selected for the HBM Density, then a duplicate option for Stack 1 appears. Selecting this option allows the flexibility of global addressing at the cost of latency. If this is disabled, each AXI port can only access the pseudo channel assigned to it. See Table 6: AXI Port Assignments

- **Enable and Configure all MCs to Same Value:** Selecting **Yes** sets all Memory Channels to the same value. All the options in the GUI references MC0 and all MC0 settings are applied to all enabled Memory Channels. Selecting **No** for this option displays options for each Memory Channel.

- **Enable External APB Interface:** Enable the external APB interface to Write/read the documented controller status and performance registers.

Clocking

- **HBM Frequency for Stack 0/1 (MHz):** Specifies the memory interface clock (`ck_t`) frequency. The wizard internally generates the necessary PLL settings to generate this memory clock frequency. The supported range is shown in Figure 7: AXI HBM Customize IP – HBM Configuration Selection Tab.

- **PLL Reference Input Clock (MHz):** Specifies the PLL reference clock frequency that the core uses to generate the HBM clock.

- **APB Interface 0/1 Clock (MHz):** Specifies the register interface (APB) clock frequency. This frequency should match with the exact frequency driven on `APB_PCLK_*` port. The supported range is shown in Figure 7: AXI HBM Customize IP – HBM Configuration Selection Tab.

- **Temperature Polling Interval on APB-0/1 (ms):** Specifies the interval at which the temperature sensor is read internally and output on the DRAM_x_STATE_TEMP port. The interval is specified in milliseconds.

Select MCs to Enable for Stack 0/1

The MC Stack tab lists all the Memory Channel controllers in each stack, and each might be individually enabled or disabled for power savings.

> **TIP:** *Disabling MCs also disables the memory underneath. This is not the same as not using one of the 16 available AXI interfaces.*

Select AXI Slaves to Enable for Stack 0/1

The AXI Slaves section lists all the possible AXI ports which can be used in the stack depending on the Global Address option and the enabled MCs.

Switch Clock Select 0/1

Select the clock to use for the AXI Switch. The IP automatically defaults to a clock that is roughly in the middle of the enabled Memory Channels. For each Memory Channel there are two AXI ports available. If Memory Channels 0, 2, 3, and 7 are enabled this means that AXI ports 0, 1 (for MC0), 4, 5 (for MC2), 6, 7 (for MC3), 14, and 15 (for MC7) are enabled and might be selected to use for the Switch Clock.

Select Line Rate for Each MC of Stack 0/1

Selects the memory channel operating frequency. This setting is automatically updated based on the IP configuration and is for information only.

Example Design Options Tab

The following figure shows the Example Design Options tab for the HBM IP.

*Figure 8:* **AXI HBM Customize IP – Example Design Options Tab**
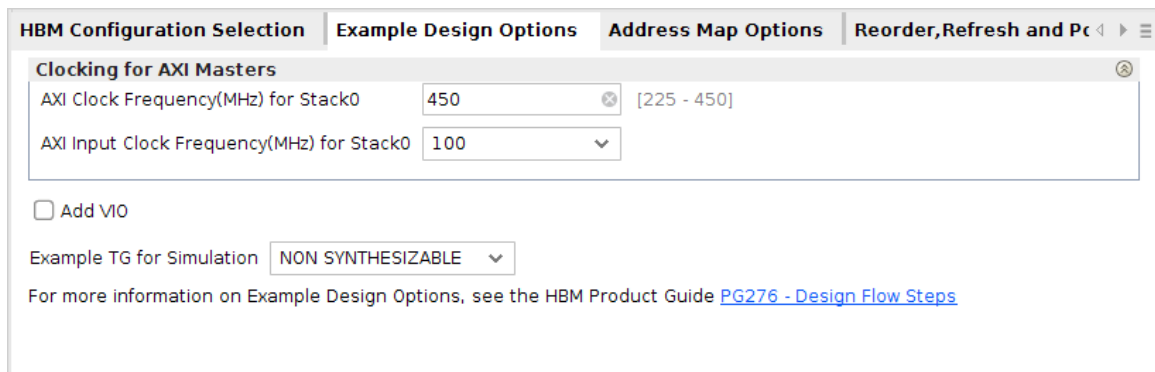
- **Clocking for AXI Masters:** Specifies the AXI clock frequency to be used for all enabled AXI interfaces in Example Design. The MMCM primitive is instantiated per stack in the example design top level file to generate the AXI clock. The 'AXI Clock Frequency (MHz)' specifies the MMCM output clock frequency that is used as AXI clock for all AXI interfaces. The 'AXI Input Clock Frequency (MHz)' specifies the MMCM input clock frequency.

- **Add VIO:** Select to add VIO instance in the example design. These VIOs can be use to start or pause traffic for the Synthesizable Traffic Generator, report any data miscompares, as well as track the number of Write and Read transactions:

  - `vio_tg_start_x` - Start traffic on AXI Port X

- `vio_tg_pause_x` - Pause traffic on AXI Port X

- `axi_x_data_msmatch_err` - Data error reported on AXI Port X

- `wr_cnt_x` - Write transaction count on AXI Port X

- `rd_cnt_x` - Read transaction count on AXI Port X

- **Example TG for Simulation:** Select a Traffic Generator module to be used in the Simulation flow. Default selection of TG for simulation flow is Non-Synthesizable TG. The Non-Synthesizable TG allows for a text based pattern entry for easy pattern generation. This is described in more detail in the Non-Synthesizable Traffic Generator section. Drop down option is provided to select 'Synthesizable TG' for a more traditional RTL stimulus which is described in more detail in the Synthesizable Traffic Generator section. The Synthesizable Traffic Generator is always present in fully implemented hardware designs.

Address Map Options Tab

The following figure shows the Address Map Options tab for the HBM IP.

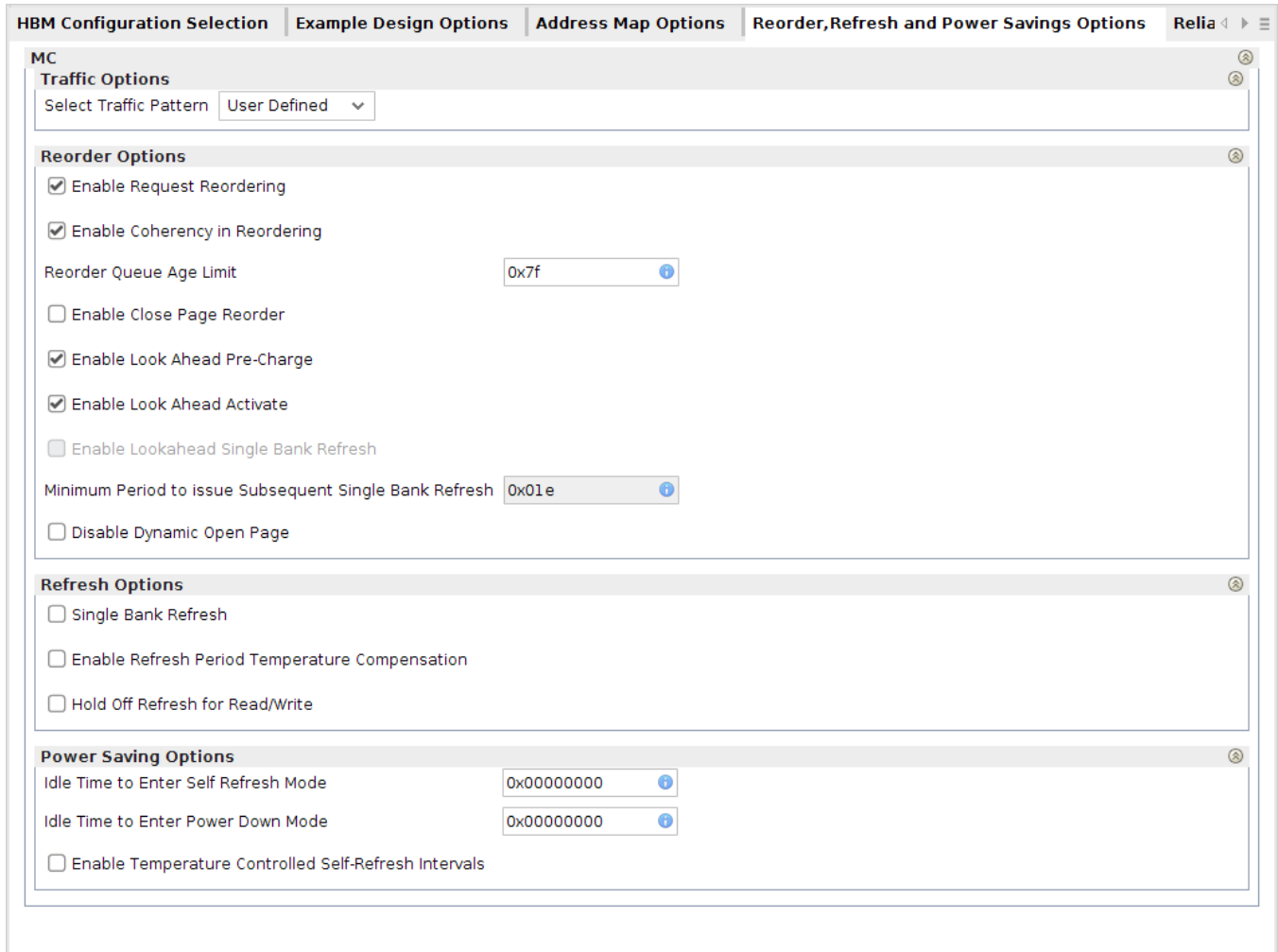*Figure 9:* **AXI HBM Customize IP – Address Map Options Tab**
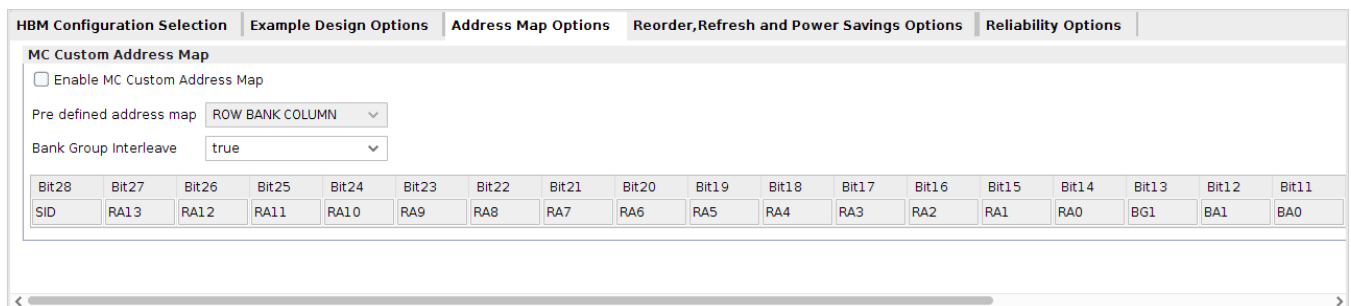
- **Custom Address Map:** Check the **Custom Address Map** box if the default settings are not desired. You can define the mapping from the AXI addresses to HBM addresses. If this is not checked, then the default settings are assigned. If the **Configure MCs to the Same Value** options is set on the **Configuration** tab, then only MC0 is displayed and all MCs are set to this value. If that option is not set, then all the enabled MCs appear. Selecting a **Custom Address Map** is not compatible with all the re-ordering options in the **Reordering** tab.

- **Bank Group Interleave:** Enables sequential address operations to alternate between even and odd bank groups to maximize memory efficiency. Random access patterns might not benefit from this setting.

Send Feedback

Reorder, Refresh, and Power Savings Options Tab

The following figure shows the Reorder, Refresh, and Power Savings Options tab of the HBM IP.

Figure 10: **AXI HBM Customize IP – Reorder, Refresh, and Power Savings Options Tab**

Traffic Options

- **Select Traffic Pattern:** Traffic pattern can be selected as Linear, Random, or User_Defined. For Linear and Random traffic pattern, the wizard configures the IP to provide optimum performance for the given traffic pattern. When User_Defined traffic pattern is selected, predefined configuration is not provided.

Reorder Options

Note: These options are only available if not using a custom address map.

- **Enable Request Reordering:** This enables the controller to re-order commands within a 64 deep window to improve efficiency. This re-ordering is different than the re-ordering happens in the AXI3 interface based on the ID tag.

- **Coherency in Reordering:** Permits the controller to re-order commands provided they are not in the same bank and row.

- **Reorder Queue Age Limit:** Sets the maximum number of newer AXI commands that can be issued ahead of a pending AXI command. This allows for adjustment of maximum latency for any AXI command versus reordering for optimum memory throughput.

- **Enable Close Page Reorder:** When enabled, this closes a page after an instruction has completed. When disabled, the page remains open until a higher priority operation is requested for another page in the same bank.

- **Enable Lookahead Pre-Charge, Activate:** These are used to allow the controller to maximize throughput by optimizing commands in the 12-command queue. If these options are disabled, all commands are processed in the order received.

- **Enable Lookahead Single Bank Refresh:** This allows the controller to insert refresh operations based on pending operations to maximize efficiency. This option isn't selectable until the "Single Bank Refresh" option is enabled.

- **Minimum Period to issue Subsequent Single Bank Refresh:** Enter a value for the minimum period for a single bank refresh.

- **Disable Dynamic Open Page:** When checked the Dynamic Open Page mode is disabled and a Pure Open Page Mode is enabled.

Refresh Options

- **Single Bank Refresh:** Banks are refreshed individually. This affects bank access order to minimize the number of refreshes required.

- **Enable Refresh Period Temperature Compensation:** The controller adjusts the rate of refreshes based on the temperature of the memory stacks.

- **Hold Off Refresh for Read/Write:** This option allows a refresh to be delayed to permit operations to complete. Refreshes might need to be done more frequently to compensate for this delay.

Power Saving Options

- **Idle Time to enter Self Refresh Mode:** After this number of idle cycles, the controller places all memory ranks in self-refresh. This feature is disabled when configured as '0'.

- **Idle Time to enter Power Down Mode:** After this number of idle cycles, the controller places all memory ranks in power down mode. This feature is disabled when configured as '0'. The controller still generates refresh commands at the expected refresh interval.

- **Enable Temperature Controlled Self-Refresh Intervals:** When in Self-Refresh, adjust the refresh rate as needed based on the memory temperature. This only applies when "Idle Time to enter Self Refresh Model" has non-zero value.

Reliability Options Tab

The following figure shows the Reliability Options tab for the HBM IP.

Figure 11: **AXI HBM Customize IP – Reliability Options Tab**
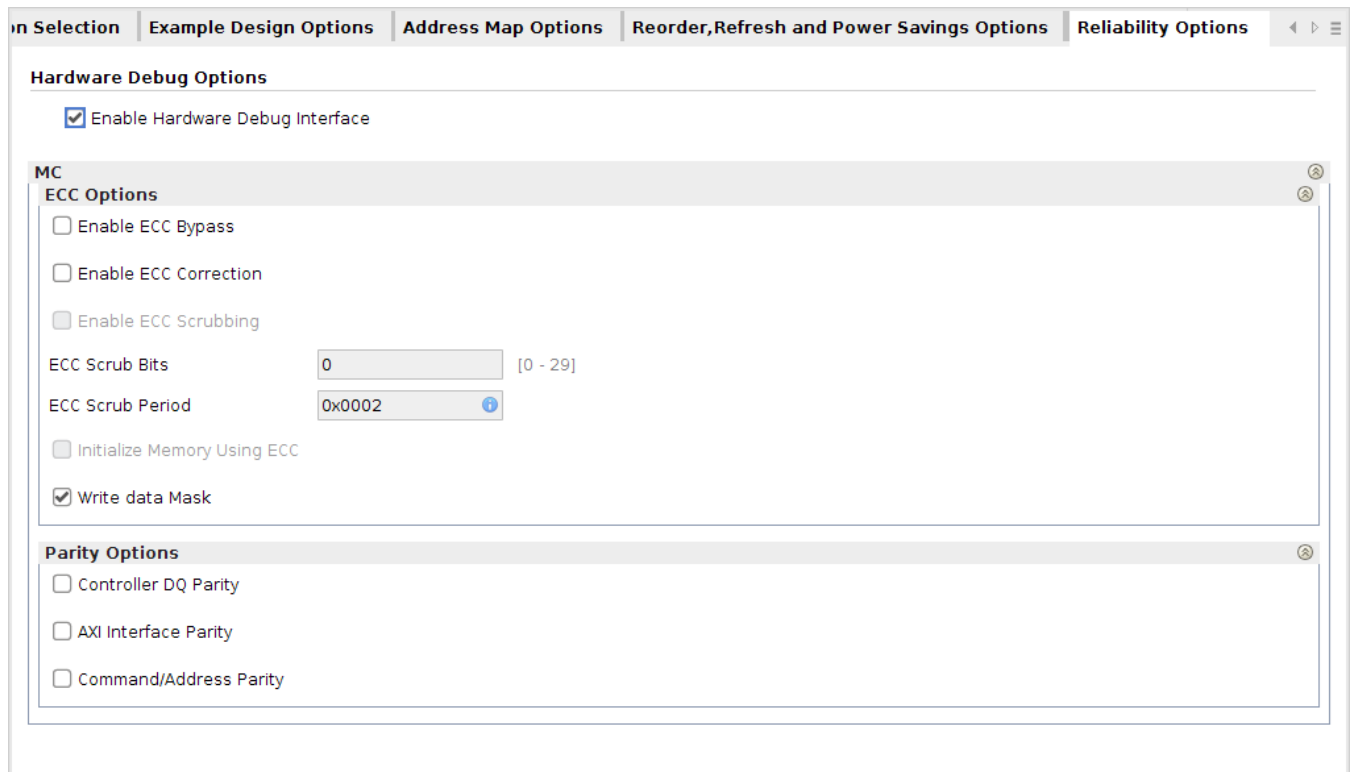
Hardware Debug Options

- **Enable Hardware Debug Interface:** Enable the XSDB hardware debugger core in the HBM IP to see the HBM properties and status in the Vivado Hardware Manager.

ECC Options

- **Enable ECC Bypass:** When enabled, the controller will not compute ECC check data or perform ECC correction on received data. In ECC bypass mode, wdata_parity input pins are repurposed to be the data that gets written to ECC memory. Likewise, rdata_parity outputs are repurposed to be the data that gets read out of ECC memory.

- **Enable ECC Correction:** When selected, single bit errors are corrected and double bit errors are only detected. In both cases these values are logged in status registers.

- **Enable ECC Scrubbing:** This option requires ECC Correction to be enabled. This enables continuous scrubbing of the memory, correcting single bit errors.

- **ECC Scrub Bits:** Number of memory address bits to use for ECC Initialization and Scrubbing. If set to 0, the default max number of bits for the currently configured memory will be used.

- **ECC Scrub Period:** Period between read operations for ECC scrubbing. This value is in units of 256 memory clock periods. For example, a value of 2 means a delay of 2x256=512 memory controller clock periods between each read.

- **Initialize Memory Using ECC:** This option requires ECC Correction to be enabled. This initializes the full memory array with valid ECC values.

- **Write Data Mask:** This option is not compatible with ECC Correction. This enables the ability to use Write Data Mask.
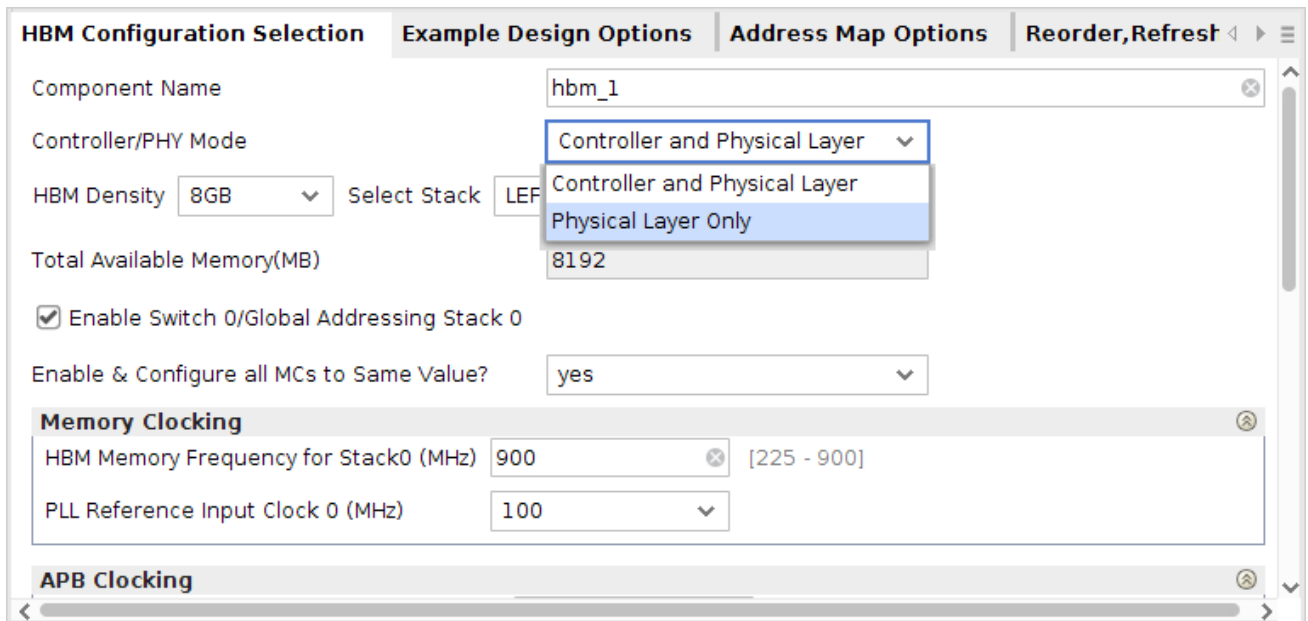
Parity Options

- **Controller DQ Parity:** Enables Write and Read Controller DQ parity. This is generated by HBM controller as per the JEDEC specification. Retry is also enabled when this option is selected.

- **AXI Interface Parity:** Enable the top level AXI Write/Read data parity ports and set the expectation that the user must provide Write parity data along with the Write payload bursts. When enabled, a BRESP SLVERR will occur if the Write DQ parity is not matched with the payload. User application must check the read DQ parity data against the response payload for errors. Odd parity is calculated per 8 bits of data. So, for 256 bits of data bus there are 32 bits of parity port.

- **Command/Address Parity:** Enables the use of the parity bits for command and address.

Send Feedback

PHY Only Mode

The AXI HBM Controller IP can be configured in PHY only mode. In this mode the AXI Switch and Memory Controllers in the HBM stacks are bypassed. You can use your own HBM controller to connect with the HBM IP in PHY only mode. A standard DFI (DDR PHY) interface, rather than an AXI interface, is given in PHY only mode. The following figure shows how to configure the IP in PHY only mode.

Figure 12: **Configuring the IP in PHY Only Mode**

In the HBM IP configuration options in the Vivado IDE a Controller and Physical Layer option is available in the Controller/PHY Mode field, which is the standard way to access the HBM IP through AXI user interfaces. The default value of this option is Controller and Physical Layer. This default option is for the full HBM IP. To configure the HBM IP in PHY only mode, select the **Physical Layer Only** value for Controller/PHY Mode field. After selecting Physical Layer Only, the Vivado IDE is updated, as shown in the following figure.

Figure 13: **PHY Only Mode Configuration Options**

Instead of memory controllers you can enable/disable PHYs in a given stack. The AXI slave interface selection is grayed out. Depending on the number of PHYs enabled, the corresponding DFI interfaces are activated as shown in the following figure.

Send Feedback

Figure 14: **DFI Interfaces**
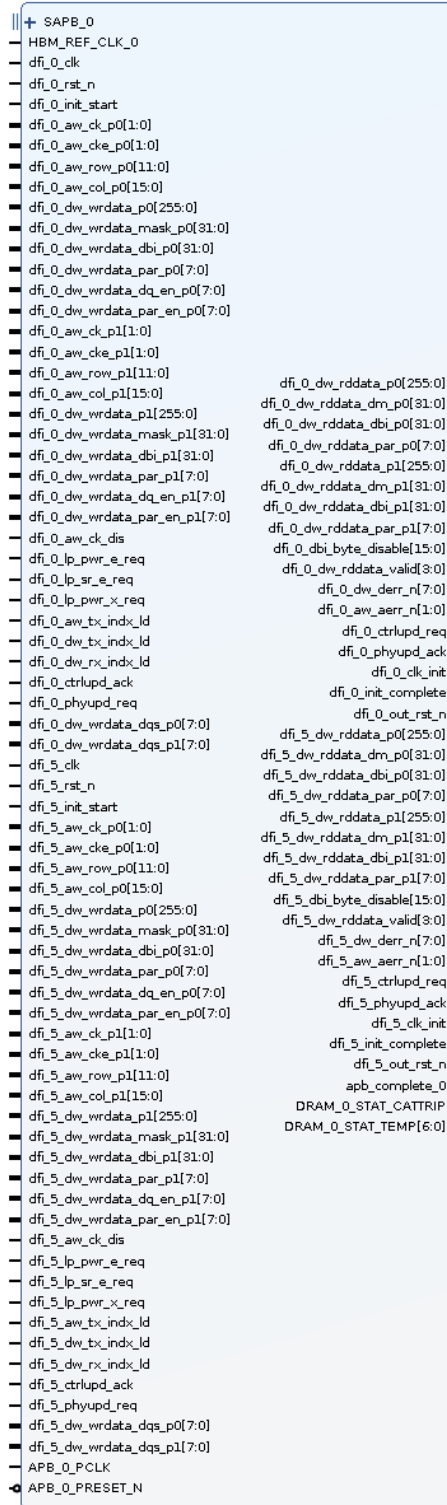
In PHY only mode, the following configuration option tabs are hidden because these tabs are only used for complete Controller and Physical Layer designs: Address Map Options, Reorder, Refresh and Power Saving Options, and Reliability Options. Only Example Design Options along with the main HBM Configuration Selection Options are available in Physical Layer Only mode.

Test Bench

Only one traffic generator is supplied in the example design for PHY only mode. This traffic generator is used for both the synthesis/implementation and simulation flows. Consequently, the option Example TG for Simulation is grayed out and the only available option is SYNTHESIZABLE.

The test bench for PHY only mode performs Write and Read operations. In simulation mode, after performing a few Write and Read operations, the simulation stops and performs a data integrity check. While running on hardware, continuous Write and Read transaction traffic is sent and data integrity checking is also performed in parallel.

Output Generation

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

Constraining the Core

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

Simulation

When building custom simulation scripts, there are .MEM files which contain IP settings which must be included in the simulation. There is one .MEM file for each memory stack: `xpm_internal_config_file_sim_0.mem` and `xpm_internal_config_file_sim_1.mem`. By default, these are located in the `project/project.srcs/sources_1/ip/hbm_0/hdl/rtl directory`. Not including these files will likely result in calibration failure in simulation. These files are automatically included if launching simulation runs within the Vivado environment.

HBM IP simulation is only supported with Verilog Compiler Simulator (VCS), Incisive Enterprise Simulator (IES), and Mentor Graphics Questa Advanced Simulator. Simulation is not supported natively with the Vivado Simulator.

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).

Synthesis and Implementation

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

Example Design

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

This following section contains steps for running the HBM controller simulations using the Vivado® HBM Example Design. Simulation is supported with the Verilog Compiler Simulator (VCS), Incisive Enterprise Simulator (IES), and Mentor Graphics Questa Advanced Simulator. For a list of known issues, see AR 69267 for the AXI HBM Controller.

Implementing the Example Design

1. Select the **HBM IP** from the Vivado IP catalog as shown in the following figure.
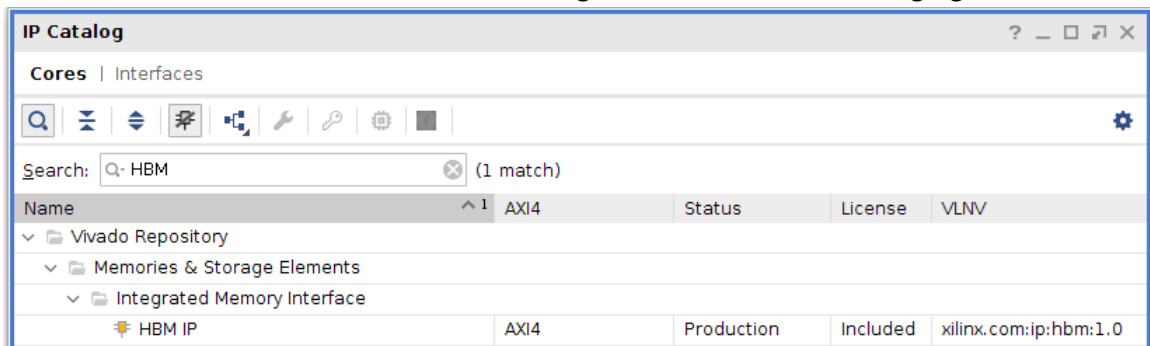
2. Choose the IP settings and click **OK**.
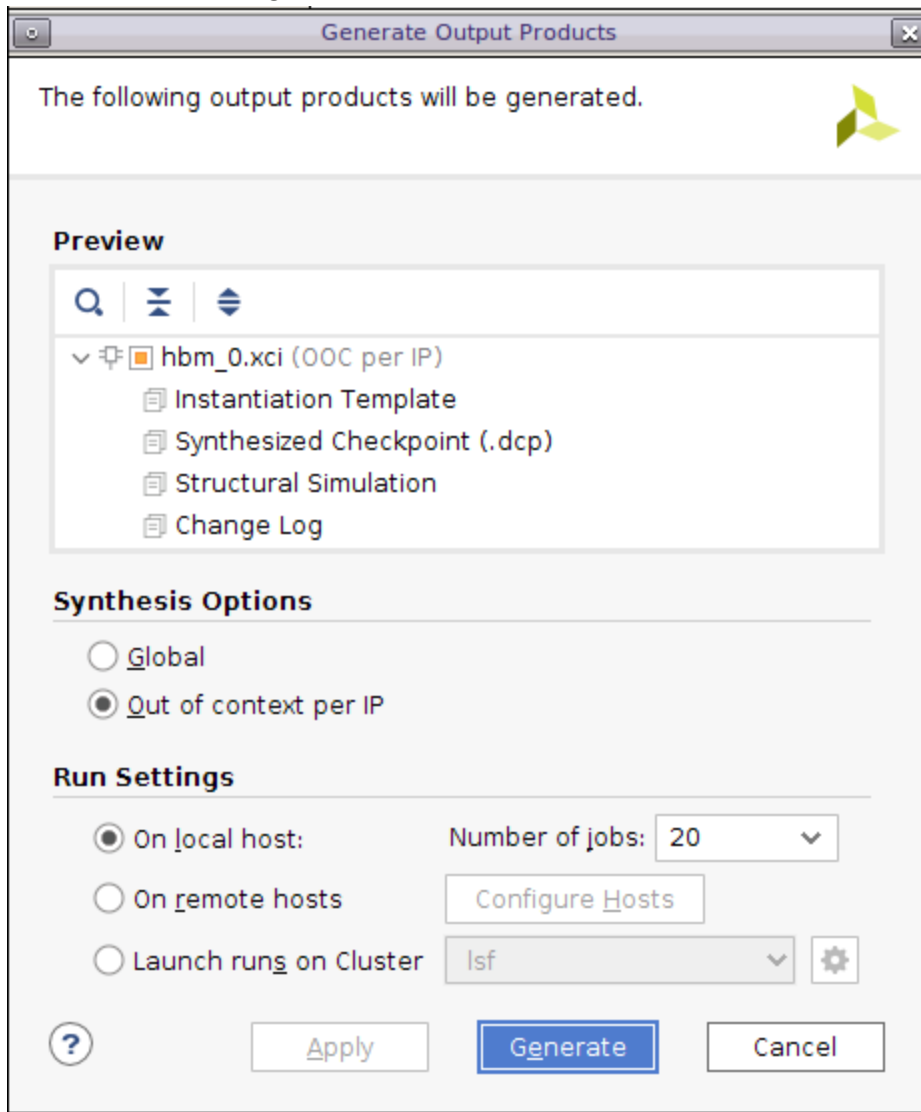
3. Under the Synthesis Options, select the **Global** option and click **Generate**.

4. Right-click the **HBM IP** and select, the following window appears **Open IP Example Design**.

5. A new project opens, under the Project Manager, select **Settings** and click **Simulation**.

6. In the Project Settings under Simulation, select **Verilog Compiler Simulator (VCS)** as the Target simulator as shown in the following figure.
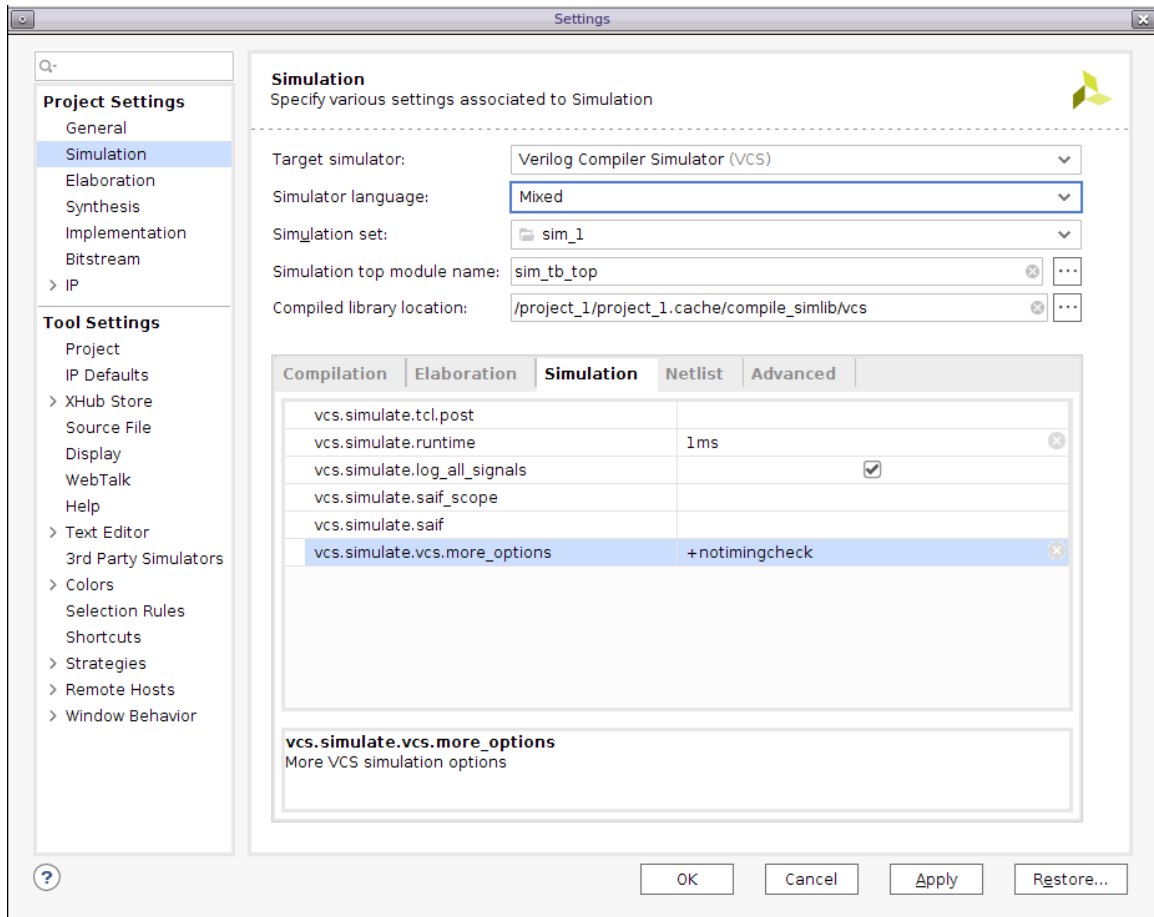
7. Change the Compiled library location to the location of the compiled library in the installation area.

8. In the Simulation tab, in the vcs.simulate.vcs.more_options enter **+notimingcheck**.

   a. For Questa simulator: In the Compilation tab questa.compile.vlog.more_options add **+notimingchecks**

   In the Simulation tab questa.simulate.vsim.more_options add **+notimingchecks** as well.

   Also in the Simulation tab questa.simulate.vsim.more_options add **-onfinish final**. This executes the final block specified by any module when simulation ends.

   b. For IES simulator, add **-notimingchecks** in the Elaboration tab ies.elaboration.ncelab.more_options.

9. In the Simulation tab, change the run-time to 1 ms as shown in the following figure.
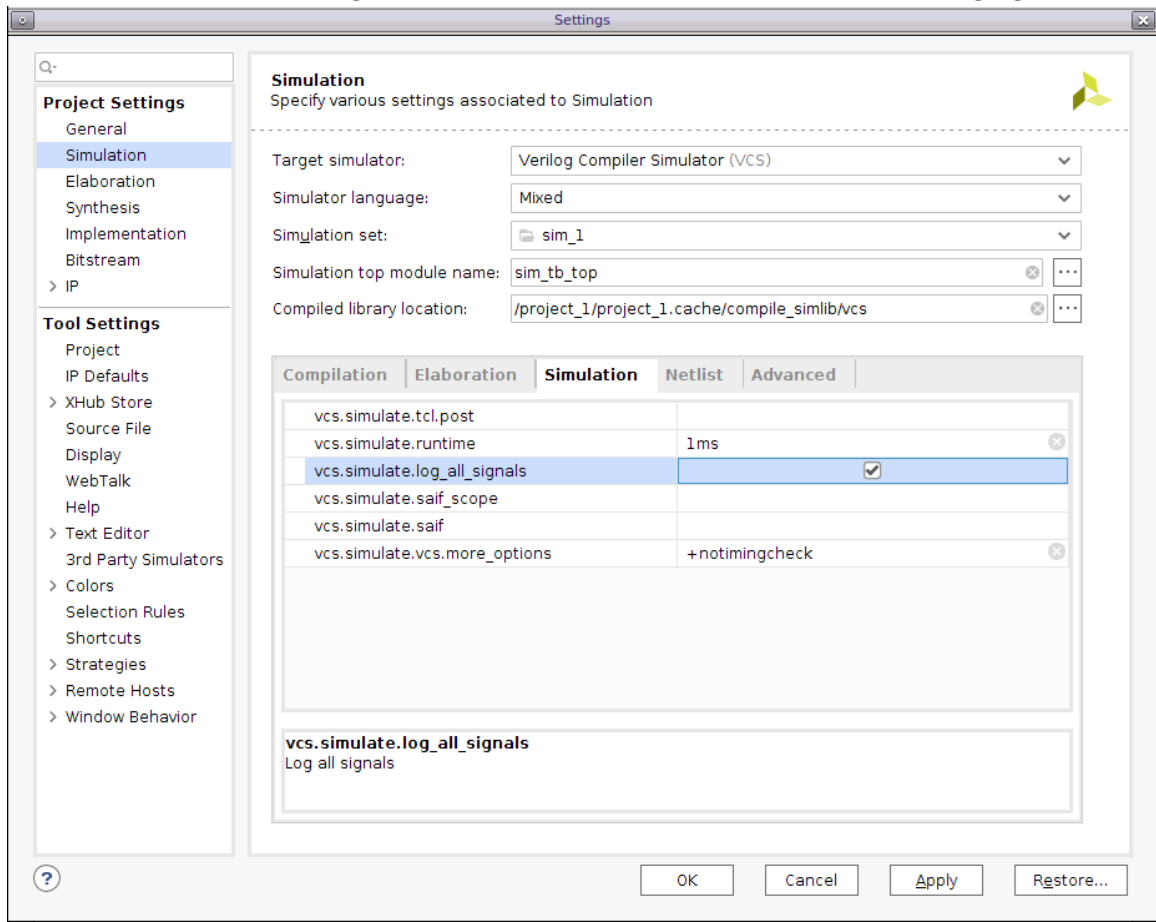
10. Run the simulation.

Simulating the Example Design

For more information on Simulation, see *Vivado Design Suite User Guide: Logic Simulation* (UG900).

Simulation Results

The simulation log files, stimulus generator file, and results can be found at:

`Example_design_directory/ip_top_name_ex/ip_top_name_ex.sim/sim_1/`
`behav/vcs directory`

The main simulation log file is `simulate.log`. This file has the transcript from the memory model. Also, towards the end of the log file the performance numbers for each master are listed. The following is an example:

```
=========================================================
>>>>>> SRC_ID 1 :: AXI_PMON :: BW ANALYSIS >>>>>>
=========================================================
AXI Clock Period = 2000 ps
Min Write Latency = 30 axi clock cycles
Max Write Latency = 180 axi clock cycles
Avg Write Latency = 163 axi clock cycles
Actual Achieved Write Bandwidth = 15724.815725 Mb/s
*****************************************************
Min Read Latency = 263 axi clock cycles
Max Read Latency = 2228 axi clock cycles
Avg Read Latency = 1369 axi clock cycles
Actual Achieved Read Bandwidth = 4894.837476 Mb/s
```

Note: The stimulus file is named as `axi_tg.txt`, it is a comma separated file (CSV). Due to tool restrictions, it is named as `.txt` instead of CSV.

To change the default stimulus, the `axi_tg.txt` file needs to be updated. For more details about the stimulus file, see the Non-Synthesizable Traffic Generator section.

The stimulus file, `axi_tg.txt`, holds the description of the behavior of all the masters that need to be simulated. The CSV files from all masters need to be concatenated in this file. Refer to `axi_tg.txt` generated in the example design project.

Non-Synthesizable Traffic Generator

The AXI traffic for each master is generated by a traffic generator that takes a CSV file as input. One traffic generator generates both Read and Write traffic. The traffic generator has a wide range of functionality and produces traffic based on user input. For the current release, stimulus to the traffic generator is supported only through the CSV file.

Non-Synthesizable Traffic Generator Commands

Each row in the CSV file specifies a traffic generator command that is executed by the AXI Traffic Generator (AXI-TG). The traffic generator commands are as follows:

- `WRITE`

- `READ`

- `WAIT`

- `START_LOOP/END_LOOP`

Send Feedback

- `SET_DEFAULT`
- `DISPLAY`

WRITE

Issues a Write command. The following table describes the WRITE command input fields and options:

Table 9: **WRITE Command Input Fields and Options**

Command Input Field	Options
txn_count	<value in dec> – Repeat the current command n times. <value in dec>KB – Amount of data to be transferred in KB. For example, 50 KB, 100 KB, etc. <value in dec>MB – Amount of data to be transferred in MB. For example, 5 MB, 10 MB, etc. <value in dec>GB – Amount of data to be transferred in GB. For example, 1 GB, 2 GB, etc.
start_delay	<value in number of clks (dec) > – Issue transaction after specified number of clks. <bandwidth in dec > Mb/s – Bandwidth for Write transactions. For example, 12,500 Mb/s.
inter_beat_delay	<value in number of clks (dec) > Add specified delay between two beats in an AXI transaction.
wdata_pattern	random – Random data is generated on all Write data beats. wdata_pat_value field gives a seed value for RNG. same_as_src – All Write data beats are generated with PARAM_SRC_ID value. same_as_addr – All Write data beats are generated with the corresponding beat start address. constant – All Write data beats are generated with the constant value is set on wdata_pat_value field. walking_1 – The value of 1 walks through (bit wise) the Write data beats where rest of bits are 0 walking_0 – The value of 0 walks through (bit wise) the Write data beats where rest of bits are 1.
wdata_pat_value	<seed_value in hex> – When wdata_pattern is selected as random, this field value is used as the seed for the RNG. <value in hex> – When wdata_pattern is selected as constant, this value is used as constant data which sent in all Write data beats.
data_integrity	enabled – Data integrity checks are enabled for the Write transaction. For example, the transaction data is stored in the memory by AXI-TG to compare later when Read occurs to this location. disabled – Data integrity checks are disabled for this Write transaction. For example, Read data comparison is not available.

Send Feedback

Table 9: **WRITE Command Input Fields and Options** *(cont'd)*

Command Input Field	Options
addr_incr_by	<value in hex> – This field is used when axi_addr is set to start_addr value. If txn_count > 0, for each AXI transaction issued, increment the address by the specified value. If add_incr_by = 0, all transactions are issues with same axi_addr. <seed_value in hex> – This field value is used as the seed for RNG to generate random addr when axi_addr field is set any one of the following options: random, random_aligned, random_unaligned, random_uniform, random_uniform_aligned, or random_uniform_unaligned. auto_incr – The first transactions start address is picked from axi_addr field and then each transactions start addresses are calculated by using this expression: start_address (from second txn) = previous_start_address + ((1<<axi_size) × (axi_len+1)). For example, txn_count 3, axi_addr 0000_0000, axi_size 6, axi_len 0. The start address of all three transactions are: First txn start address: 0000_0000, Second txn start address: 0000_0000 + ((1<<6) × (0+1)) = 0000_0040, Third txn start address: 0000_0040 + ((1<<6) × (0+1)) = 0000_0080
dest_id	<value in hex> – Targeted slave ID value can be given in this field for the transaction. <Parameters> – Targeted slave ID can be given using Parameters such SLAVE_DST_ID0, SLAVE_DST_ID1, These parameters hold the slave ID value in adv_axi_tg top.
base_addr	<value in hex> The base_addr can be specified in this field. The incremented addr for the txns are looped back to the base address when the high address boundary is reached.
high_addr	The high address can be specified here. The traffic generator increments up to this value. After the incremented addr reaches the high_addr boundary, the next transaction start address is the base_addr.
axi_addr	<value in hex> – AXI transaction start address value. If txn_count > 0, the next transaction address is calculated using addr_incr_by. random – Any random address is generated between base_addr and high_addr. addr_incr_by field value is set as the seed to the RNG. random_aligned – Any random aligned address is generated between base_addr and high_addr. addr_incr_by field value is set as the seed to the RNG. random_unaligned – Any random unaligned address is generated between base_addr and high_addr. addr_incr_by field value is set as the seed to the RNG. random_uniform – Uniformly distributed random addr is generated between base_addr and high_addr. addr_incr_by field value is set as the seed to the RNG. random_uniform_aligned – Uniformly distributed random aligned addr is generated between base_addr and high_addr. addr_incr_by field value is set a the seed to the RNG. random_uniform_unaligned – Uniformly distributed random unaligned addr is generated between base_addr and high_addr. addr_incr_by field value is set as the seed to the RNG.
axi_len	This hex value passes to the AXI_xx_AWLEN port.
axi_size	This hex value passes to the AXI_xx_AWSIZE port.
axi_id	<value in hex> – All the Write transactions are sent with this ID. auto_incr – The first Write transaction is sent out with ID of 0 and the each consecutive transactions IDs is incremented by 1.

Send Feedback

Table 9: **WRITE Command Input Fields and Options** *(cont'd)*

Command Input Field	Options
axi_burst	<value in hex> – 0x1, 0x2 INCR, WRAP
axi_lock	Set to 0, this AXI port is not used in the HBM IP.
axi_cache	Set to 0, this AXI port is not used in the HBM IP.
axi_prot	Set to 0, this AXI port is not used in the HBM IP.
axi_qos	Set to 0, this AXI port is not used in the HBM IP.
axi_region	Set to 0, this AXI port is not used in the HBM IP.
axi_user	Set to 0, thi s AXI port is not used in the HBM IP.

The following table shows the reset values for the WRITE command.

Table 10: **Reset Values for the Write Command**

Command Input Field	WRITE (Reset Value)
txn_count	100
start_delay	0
inter_beat_delay	0
wdata_pattern	Constant
wdata_pat_value	0
data_integrity	Disabled
dest_id	SLAVE_DST_ID0 (TG Parameter)
base_addr	C_AXI_WR_BASEADDR (TG Parameter)
high_addr	C_AXI_WR_HIGHADDR (TG Parameter)
addr_incr_by	auto_incr
axi_addr	0
axi_len	0
axi_size	C_AXI_WRITE_MAX_SIZE (TG Parameter)
axi_id	auto_incr
axi_burst	1
axi_lock	0

Send Feedback

Table 10: **Reset Values for the Write Command** *(cont'd)*

Command Input Field	WRITE (Reset Value)
axi_cache	2
axi_prot	0
axi_qos	0
axi_region	0
axi_user	0

READ

Issues a Read command. The following table describes the READ command input fields and options:

Table 11: **READ Command Input Fields and Options**

Command Input Field	Options
txn_count	<value in dec> – Repeat the current command n times. <value in dec>KB – Amount of data to be transferred in KB. For example, 50 KB, 100 KB, etc. <value in dec>MB – Amount of data to be transferred in MB. For example, 5 MB, 10 MB, etc. <value in dec>GB – Amount of data to be transferred in GB. For example, 1 GB, 2 GB, etc.
start_delay	<value in no of clks (dec)> – Issue transaction after specified number of clks. <bandwidth in dec>Mb/s –Bandwidth for Read transactions. For example, 12,500 Mb/s.
data_integrity	enabled – Data integrity checks are enabled for the Write txn. For example, this particslar txn details are stored in memory by the AXI-TG for data checks later when Read occurs to this location. disabled – Data integrity checks are disabled.
addr_incr_by	<value in hex> – Field value is used when axi_addr is set to start_addr value. If txn_count>0, for each axi_transaction issued, increment the addr by the specified value. If add_incr_by = 0, all transactions are issues with same axi_addr. <seed_value in hex> – Field value is used as the seed for the RNG to generate random addresses when axi_addr field is set any one of the following options: random, random_aligned, random_unaligned, random_uniform, random_uniform_aligned, or random_uniform_unaligned. auto_incr – First transactions start address is picked from axi_addr field. And then each subsequent transaction start addresses are calculated by using this expression: start_address (from second txn) = previous_start_address + ((1<<axi_size) × (axi_len+1)). For example, txn_count 3, axi_addr 0000_0000, axi_size 6, and axi_len 0. The start address of all three transactions are: First txn start address: 0000_0000, Second txn start address: 0000_0000 + ((1<<6) × (0+1)) = 0000_0040, Third txn start address: 0000_0040 + ((1<<6) × (0+1)) = 0000_0080.

Table 11: **READ Command Input Fields and Options** *(cont'd)*

Command Input Field	Options
dest_id	<value in hex> – Targeted slave ID value can be given in this field for the transaction. <Parameters> – Targeted slave ID can be given as parameters such SLAVE_DST_ID0, SLAVE_DST_ID1,.. These parameters hold the slave ID value in adv_axi_tg top.
base_addr	<value in hex> The base_addr can be specified in this field. The incremented address for the transactions are looped back to the base address when the high_addr boundary is reached.
high_addr	<value in hex> The high addr can be specified here. The TG increments up to this value. After the incremented address reaches the high_addr boundary, the next transaction start address is the base_addr.
axi_addr	<value in hex> – AXI transaction start address value. If txn_count > 0, the next transaction address is calculated based on addr_incr_by. random – Any random address is generated between base_addr and high_addr. addr_incr_by value is set as the seed to the RNG. random_aligned – Any random aligned address is generated between base_addr and high_addr. addr_incr_by field value is set as the seed to the RNG. random_unaligned – Any random unaligned address is generated between base_addr and high_addr. addr_incr_by field value is set as the seed to the RNG. random_uniform – A uniformly distributed random address is generated between base_addr and high_addr. addr_incr_by field value is set as the seed to the RNG. random_uniform_aligned – A uniformly distributed random aligned address is generated between base_addr and high_addr. addr_incr_by value is set as the seed to the RNG. random_uniform_unaligned – A uniformly distributed random unaligned addr is generated between base_addr and high_addr. addr_incr_by value is set as the seed to the RNG.
axi_len	This value passes to the AXI_xx_ARLEN port.
axi_size	This value passes to the AXI_xx_ARSIZE port.
axi_id	<value in hex> – All the Read transactions are sent with this ID. auto_incr – The first Read transaction is sent out with ID of 0 and the each consecutive transaction ID is incremented by 1.
axi_burst	<value in hex> – 0x1, 0x2. INCR, WRAP.
axi_lock	Set to 0, this AXI port is not used in the HBM IP.
axi_cache	Set to 0, this AXI port is not used in the HBM IP.

Send Feedback

Table 11: **READ Command Input Fields and Options** *(cont'd)*

Command Input Field	Options
axi_prot	Set to 0, this AXI port is not used in the HBM IP.
axi_qos	Set to 0, this AXI port is not used in the HBM IP.
axi_region	Set to 0, this AXI port is not used in the HBM IP.
axi_user	Set to 0, this AXI port is not used in the HBM IP.

The following table shows the reset values for the Read command.

Table 12: **Reset Values for the Read Command**

Command Input Field	READ (Reset Value)
txn_count	100
start_delay	0
inter_beat_delay	N/A
wdata_pattern	N/A
wdata_pat_value	N/A
data_integrity	Disabled
dest_id	SLAVE_DST_ID0 (TG Parameter)
base_addr	C_AXI_RD_BASEADDR (TG Parameter)
high_addr	C_AXI_RD_HIGHADDR (TG Parameter)
addr_incr_by	auto_incr
axi_addr	0
axi_len	0
axi_size	C_AXI_READ_MAX_SIZE (TG Parameter)
axi_id	auto_incr
axi_burst	1
axi_lock	0
axi_cache	2
axi_prot	0
axi_qos	0
axi_region	0
axi_user	0

Send Feedback

WAIT

Waits for responses to be received at the `axi_tg`. This blocks the `axi_tg` execution until the responses are received. The following table describes the `WAIT` command input fields and options:

Table 13: **WAIT Command Input Fields and Options**

Command Input Field	Options
txn_count (response options),	all_rd_resp – Wait until all Read responses are received. all_wr_resp – Wait until all Write responses are received. all_wr_rd_resp – Wait until all Write and Read responses are received. <value in dec> – Wait for a specified time (for example, 10 μs, 2 ms, etc.,) or specified number of clock cycles (for example, 10 clk, 50 clk). wait_by_time or wait_by_clk is decided based on the unit given in column C.
Start_delay (Addr_control),	<none> <none> <none> clk, ps, ns, μs, ms

The following table shows the reset value for the WAIT command.

Table 14: **Reset Values for the Wait Command**

Command Input Field	WAIT (Reset Value)
wait_option	all_wr_rd_resp
wait_unit	N/A

START_LOOP/END_LOOP

Repeat the following commands till `END_LOOP` is seen. The following table describes the `START_LOOP/END_LOOP` command input fields and options:

Table 15: **START_LOOP/END_LOOP Command Input Fields and Options**

Command Input Field	Options
txn_count (loop_count)	<value in dec> Repeat the loop by the specified number of time.
Start_delay (loop_operation)	incr_original_addr – The axi_addr is incremented by the mentioned value in column 4. use_original_addr – The axi_addr is reset to the original value.

Table 15: **START_LOOP/END_LOOP Command Input Fields and Options** *(cont'd)*

Command Input Field	Options
Inter_beat_delay (loop_addr_incr_by)	<value in hex> The axi_addr is incremented by this value when the column 3 option is set to "incr_original_addr" while executing the loop each time.

The following table shows the reset values for the `START LOOP` command.

Table 16: **Reset Values for the Start Loop Command**

Command Input Field	START_LOOP (Reset Value)
loop_count	10
loop_operation	use_original_addr
loop_addr_incr_by	N/A

SET_DEFAULT

Sets a default value to the attribute specified. This is overridden if a direct value is specified in the command. The following table describes the `SET_DEFAULT` command input fields and options:

Table 17: **SET_DEFAULT Command Input Fields and Options**

Command Input Field	Options
txn_count (command)	READ – Default values are applied to all READ commands. WRITE – Default values are applied to all WRITE commands.

Send Feedback

Table 17: **SET_DEFAULT Command Input Fields and Options** *(cont'd)*

Command Input Field	Options
Start_delay (Input fields)	bandwidth start_delay base_addr high_addr data_integrity addr_incr_by dest_id axi_addr axi_id axi_size axi_len axi_burst axi_cache axi_prot axi_qos axi_region axi_user txn_count Set the default values to all of the input fields of the WRITE/READ command, even if it is not listed above.
Inter_beat_delay (values)	bandwidth – <value in Mb/s> start_delay – <value in no of clocks> base_addr – <value in hex> high_addr – <value in hex> data_integrity – <enabled/disabled> addr_incr_by – <value in hex> dest_id – <value in hex> axi_addr – <value in hex> axi_id – <value in hex> axi_size – <value in hex> axi_len – <value in hex> axi_burst – <value in hex> axi_cache – <value in hex> axi_prot – <value in hex> axi_qos – <value in hex> axi_region – <value in hex> axi_user – <value in hex> txn_count – <value in dec>
wdata_pattern (random distribution option)	bandwidth – uniform, normal, <none>

Table 17: **SET_DEFAULT Command Input Fields and Options** *(cont'd)*

Command Input Field	Options
wdata_pat_value (random variance)	bandwidth – <% value in dec>, <none> The random variance value is mentioned here as a percentage of Bandwidth Example:1 – If the bandwidth value in column D is 2,500 and `Uniform` in column E, 10 in column F means the mean bandwidth of 2500 is achieved by uniformly random distribution with ±10% of 2,500 Mb/s variance. Example:2 – If the bandwidth value in column D is 2,500 and `normal` in column E, 10 (standard_deviation) in column 6 means the bandwidth of 2,500 Mb/s (mean) is achieved with standard deviation of 10%of 2,500. For normal distribution, the standard deviation value should be set to more than 30%.

DISPLAY

Displays the message. The following table describes the `DISPLAY` command input field and option:

Table 18: **DISPLAY Command Input Fields and Options**

Command Input Field	Options
txn_count (message)	The message given in the second column is displayed when this command executes.

CSV Examples

Example 1: User-Defined Pattern with Write and Read Command (WRITE, READ)

The following example describes a simple pattern, 100 writes followed by 100 reads with data integrity checks enabled. The first row specifies a WRITE command to be repeated 100 times. The AXI transactions are sent to one after the other with zero delay between them. The first AXI transaction is sent to address 0. The AXI address of consecutive transactions is incremented by `'h40`. Therefore, the final transaction is sent to address (100 × `'d64 =`' `d6400`). After sending the Write commands, the TG waits for all the Write responses to arrive at the AXI-TG and the DISPLAY command prints out this information on the console.

Next, a READ command is issued to the AXI-TG. Similar to the WRITE command, 100 Read AXI transactions are issued by the AXI-TG. The AXI address is incremented by `'d64` for each consecutive AXI transaction. Because the data-integrity checks are enabled, the Read data is compared against the sent Write data. In the case of any mismatches, the `tg_mismatch_error` output signal of AXI_TG is triggered. After all the Read commands are issued, the TG waits for all the Read data to arrive at the AXI-TG and the DISPLAY command prints out the information on the console. When all the commands in the CSV file are completed, `axi_tg_done` signal is triggered by the AXI-TG.

Writes Followed by Reads with Data-Integrity Checks Enabled

TG_NUM	CMD	txn_count	start_delay	inter-beat-delay	wdata_pattern	wdata_pat_value	data-integrity	dest_id	base_addr	high_addr	addr_incr_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
	WRITE	100	0	0	random	0	enabled	0	0000_0000	0000_FFFF	0000_0040	0000_0000	F	5	0	1	0	0	0	0	0	0
	WAIT	all_wr_resp	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	100 axi wr transactions sent	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	READ	100	0	-	–	–	enabled	0	0000_0000	0000_FFFF	0000_0040	0000_0000	F	5	0	1	0	0	0	0	0	0
	WAIT	all_rd_resp	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	100 axi rd transactions sent	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	End of Test	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Related Information

Non-Synthesizable Traffic Generator Commands

Example 2: User-Defined Pattern with Loop Command (START_LOOP, END_LOOP)

The `txn_count` field allows you to easily repeat the command multiple times. If you need to loop more than one command, the START_LOOP/END_LOOP pair is used. In the following example, you need to send the AXI Read transactions to an address (`'h0`) and the next Read to address+offset (`'h1000_0000`), then wait for the Read data to arrive and repeat this step 100 (`loop_count`) times.

With the command START_LOOP, 100, incr_addr_by, 40 during each loop, you increment the address by `'h40`. Hence the Read address (araddr) for consecutive transactions would be `'h0`, `'h1000_0000`, `'h40`, `'h1000_0040` and at the end of test 200 Read transactions are issued. You also have an option to send the reads to the same address (that is, `'h0, 'h1000_000, 'h0,'h1000_0000`) by using the command START_LOOP, <loop_count>, `use_original_addr`. This command resets the address to the original address specified in the `axi_addr` column.

Send Feedback

Looping in the Command Table of AXI-TG

TG_NUM	CMD	txn_count	start_delay	inter-beat-delay	wdata_pattern	wdata_pat_value	data-integrity	dest_id	base_addr	high_addr	addr_incr_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
	START_LOOP	100	use_original_addr	0000_0040	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	READ	1	0	–	–	–	disabled	0	0000_0000	0000_FFFF	0	0000_0000	F	5	0	1	0	0	0	0	0	0
	READ	1	0	–	–	–	disabled	0	1000_0000	1000_FFFF	0	0000_0000	F	5	0	1	0	0	0	0	0	0
	WAIT	all_rd_resp	–	-	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	END_LOOP		–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	200 axi rd transactions sent	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	End of Test	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Related Information

Non-Synthesizable Traffic Generator Commands

Example 3: User-Defined Pattern with SET_DEFAULT Command

If you have a large set of commands most of which have the same value in the columns, you can set default values to these and specify the keyword DEFAULT or leave it blank in the respective columns. In the following example, you should always send the same address 0 except in the third command where it is to a different address. Hence, specify the default address as 0 and override it to `'hA5A5_A5A5` in the third command.

You can also set the default Write and Read bandwidths. The units are specified in Mb/s. When the default bandwidth is specified, the AXI-TG internally calculates the delay to be applied between consecutive AXI transactions to achieve the specified bandwidth.

The calculation assumes an ideal AXI slave which does not apply any back pressure (All READY signals of AXI3 are always asserted). In case the slave applies back pressure, the AXI-TG does not push traffic at the specified bandwidth and the actual bandwidth is lower. You can specify either the default bandwidth or the default `start_delay`+`intra_byte_delay` (as both commands indirectly control the delay in the AXI transactions). The latest command overrides the previous command.

SET_DEFAULT Command Usage

TG_NUM	CMD	txn_count	start_delay	inter-beat-delay	wdata_pattern	wdata_pat_value	data-integrity	dest_id	base_addr	high_addr	addr_incr_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
	SET_DEFAULT	READ	bandwidth	10000	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	SET_DEFAULT	READ	base_addr	0000_0000	–	–	–	–	–	–	–		–	–	–	–	–	–	–	–	–	–
	SET_DEFAULT	READ	high_addr	0000_FFFF	–	–	–	–	–	–	–		–	–	–	–	–	–	–	–	–	–
	SET_DEFAULT	READ	data-integrity	disabled	–	–	–	–	–	–	–		–	–	–	–	–	–	–	–	–	–
	SET_DEFAULT	READ	addr_incr_by	0000_0040	–	–	–	–	–	–	–		–	–	–	–	–	–	–	–	–	–
	SET_DEFAULT	READ	axi_addr	0000_0000	–	–	–	–	–	–	–		–	–	–	–	–	–	–	–	–	–
	READ	100	–	–	–	–	–	–	–	–	–	DEFAULT	F	5	0	1	0	0	0	0	0	0
	READ	20	–	–	–	–	–	–	–	–	–	DEFAULT	F	5	0	1	0	0	0	0	0	0
	READ	100	–	–	–	–	–	–	–	–	–	A5A5_A5A5	F	5	0	1	0	0	0	0	0	0
	READ	500	–	–	–	–	–	–	–	–	–	DEFAULT	F	5	0	1	0	0	0	0	0	0
	WAIT	all_rd_resp	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	All axi rd transactions sent	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	End of Test	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Related Information

Non-Synthesizable Traffic Generator Commands

Example 4: User-Defined Pattern with # (commented out lines)

The following list describes the TG execution:

1. WRITE – 100 Write requests are sent to the AXI master interface back-to-back because the `start_delay` is 0.

2. WAIT – After sending out all 100 requests, the TG waits for all 100 responses received back.

3. DISPLAY – After all responses received, TG prints out the message "100 axi wr transactions sent."

4. #READ – This command skips from execution because it is commented out.

5. #WAIT – This command skips from execution because it is commented out.

6. #DISPLAY – This command skips from execution because it is commented out.

7. DISPLAY – TG prints out the message "End of Test."

The following example shows the # usage.

Usage

TG_NUM	CMD	txn_count	start_delay	inter-beat-delay	wdata_pattern	wdata_pat_value	data-integrity	dest_id	base_addr	high_addr	addr_incr_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
	WRITE	100	0	0	random	0	enabled	0	0000_0000	0000_FFFF	0000_0040	0000_0000	F	5	0	1	0	0	0	0	0	0
	WAIT	all_wr_resp	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	100 axi wr transactions sent	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
#	READ	100	0	–	–	–	enabled	0	0000_0000	0000_FFFF	0000_0040	0000_0000	F	5	0	1	0	0	0	0	0	0
#	WAIT	all_rd_resp	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
#	DISPLAY	100 axi rd transactions sent	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	End of Test	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Related Information

Non-Synthesizable Traffic Generator Commands

Example 5: User-Defined Pattern with Amount of Data to be Transferred, Wait by Time, Wait by Clock and Random Bandwidth Distribution

The following example shows the usage of data amount in `txn_count`, WAIT by time, clock, and random bandwidth. This also includes the reset values (column can be left empty for READ/ WRITE commands, reset values are picked for these fields).

Usage of Data Amount

TG_NUM	CMD	txn_count	start_delay	inter-beat-delay	wdata_pattern	wdata_pat_value	data-integrity	dest_id	base_addr	high_addr	addr_incr_by	axi_addr	axi_len	axi_size	axi_id	axi_burst	axi_lock	axi_cache	axi_prot	axi_qos	axi_region	axi_user
#	All empty fields take the reset value for the READ/WRITE commands.																					
	SET_DEFAULT	WRITE	bandwidth	2000	uniform	10	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	SET_DEFAULT	READ	bandwidth	4000	uniform	20	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	START_LOOP	5	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	sending 50 KB data	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	WRITE	50 KB	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	WAIT	1	ms	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	Reading 5 KB data	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	READ	5 KB	bandwidth	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	WAIT	10000	clk	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	END_LOOP	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	DISPLAY	End of Test	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Related Information

Synthesizable Traffic Generator

This section describes how to use the synthesizable traffic generator included in the HBM Example Design. The purpose of this traffic generator is to quickly generate different types of traffic patterns to demonstrate the capabilities of the HBM IP and it is not meant to be a sophisticated modeling tool for application workloads. This traffic generator can be used in simulation when the "Example Traffic Generator for Simulation" is set to "SYNTHESIZABLE" under the Example Design Options tab of the IP configuration GUI. Additionally this traffic generator is always present in the HBM Example Design when running in hardware. There is one traffic generator instantiated for every AXI port enabled in the Example Design. The synthesizable traffic generator behavior depends on the operating mode set in the top level of the Example Design which then references an operating mode table entry in the traffic generator instance hierarchy. There are a set of pre-defined operating modes for the HBM IP which will exercise the entire pseudo channel memory range when running in hardware or will test a smaller section of the memory when running simulation. When the "Add VIO" feature is enabled in the Example Design Options tab of the IP configuration GUI a set of VIOs are instantiated at the top level of the IP which allow the user to monitor the traffic generator transaction counts, data errors, as well as pause and start controls.

Default Behavior of the Synthesizable Traffic Generator

The synthesizable traffic generator behavior is controlled by the value set in the DEFAULT_MODE parameter for each traffic generator instance in the top level of the Example Design. This is set to "HBM" by default as seen below:

Figure 15: **Synthesizable Traffic Generator Default Mode**

```
    .DEFAULT_MODE                        ("HBM")
} u_atg_axi_0 (
    .i_TG_PATTERN_MODE_PRBS_ADDR_SEED    (24'h65_4321),
    .i_clk                               (AXI_ACLK0_st0_buf),
    .i_rst                               (~axi_rst0_st0_n),
    .i_init_calib_complete               (apb_seq_complete_0_st0_r2),
    .compare_error                       (axi_00_data_msmatch_err),
    .vio_tg_rst                          (vio_tg_rst_0),
    .vio_tg_start                        (vio_tg_start_0),
```

Send Feedback

This DEFAULT_MODE of "HBM" is one of the possible use cases that have been defined for the HBM IP. To change the DEFAULT_MODE simply modify the value in the `example_top_syn.sv` file for each traffic generator instance in the Example Design. When running in hardware for this mode the traffic generator run in a loop where it will linearly write through the entire pseudo channel address range with a PRBS data pattern and then read back the entire range while checking for data errors. The address range is automatically determined by the stack height of the HBM device in the design. When the "Add VIO" option is not selected the traffic generator will start once the `apb_complete_x` signal has been asserted for that stack. When the "Add VIO" option has been enabled the traffic generator will not start until the `vio_tg_start_x` signal has been manually set to 1'b1 through the VIO dashboard in the Vivado Hardware Manager. There will be one `vio_tg_start_x` signal for every AXI port that has been enabled in the design. In simulation this traffic mode will perform one linear write sequence of 256 32-byte bursts with PRBS data and then one linear read sequence of 256 32-byte bursts with error checking.

Additional Traffic Modes for the Synthesizable Traffic Generator

The synthesizable traffic generator mode is defined by the value set in the DEFAULT_MODE parameter for each traffic generator instance at the top level of the Example Design. By default this is set to "HBM" but there are additional modes which can be set by the user. The traffic generator will only access the pseudo channel address range associated with the AXI port.

- **HBM:** This is a high bandwidth Write and read access pattern. Linear aligned addressing sequence where the writes are performed first and then reads back the entire range while checking for data errors with a PRBS pattern. In hardware this mode will cover the entire pseudo channel address range depending on the stack height of the HBM in the device and will loop indefinitely. In simulation this will perform 256 32-byte bursts for the Write sequence, 256 32-byte bursts for the read sequence, and then stop.

- **HBM_WRITE:** This is a high bandwidth Write only access pattern. A PRBS data pattern is used for the Write only linear aligned addressing sequence. In hardware this mode will cover the entire pseudo channel address range depending on the stack height of the HBM in the device and will loop indefinitely. In simulation this will only Write 256 32-byte bursts.

- **HBM_READ:** This is a high bandwidth read only access pattern. This is executed as a read only linear aligned addressing sequence. In hardware this mode will cover the entire pseudo channel address range depending on the stack height of the HBM in the device and will loop indefinitely. In simulation this will only read 256 32-byte bursts.

- **HBM_W_R:** This is a low bandwidth access pattern since only one Write is executed and then the same address is read back. Linear aligned addressing sequence where a single Write is followed by a single read. The Write and read will alternate while linearly progressing through the address space with error checking on a PRBS data pattern. In hardware this mode will cover the entire pseudo channel address range depending on the stack height of the HBM in the device and will loop indefinitely. In simulation this will perform a total of 256 32-byte bursts for the Write sequence, a total of 256 32-byte bursts for the read sequence, and then stop.

- **HBM_RANDOM:** This is an even lower bandwidth access pattern since only one Write is executed and then the same address is read back but the traffic generator moves randomly through the address space which means a high probability of a page miss from the previous random address. This mode uses an aligned PRBS addressing sequence where one Write is followed by one read with a PRBS data pattern and error checking. In hardware this mode will cover the entire pseudo channel address range depending on the stack height of the HBM in the device and will loop indefinitely. In simulation this will perform a total of 256 32-byte bursts for the Write sequence, a total of 256 32-byte bursts for the read sequence, and then stop.

Using the VIO Controls

When the "Add VIO" option is selected in the Example Design Options tab of the IP configuration GUI a set of VIOs will be instantiated for every traffic generator enabled in the design. These can be seen in the VIO dashboard in the Vivado Hardware Manager when the design is running on hardware. For these configurations the `vio_tg_start_x` signal must be set to 1'b1 to start each traffic generator in the design. To pause the traffic generator set the `vio_tg_pause_x` signal to 1'b1. To resume traffic set `vio_tg_pause_x` to 1'b0 while `vio_tg_start_x` is still 1'b1. If a data error was detected then the compare error status is latched to the `axi_x_data_msmatch_err` signal. This signal will remain asserted until the entire design is reset. During operation the `wr_cnt_x and rd_cnt_x` will increment for every Write or Read transaction. These counters will not reset unless the entire design is reset.

Upgrading

This appendix is not applicable for the first release of the core.

Send Feedback

Debugging

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

Hardware Manager - HBM Debug Interface

The Xilinx® Vivado® hardware manager has a customized interface for UltraScale+™ devices utilizing the integrated HBM. This interface has a configuration view for verifying the IP settings as well as a dynamic graph for tracking performance through the activity monitor registers present in the memory controller. This interface requires the 'Enable Hardware Debug Interface' option to be set in the HBM IP. If the option is not set, the HBM will not appear in the Debug core list.

Note: When any IP containing a debug core is used, the Vivado tools will automatically select a clock for the master debug hub. To manually specify a clock, use the `connect_debug_port` constraint. The following is the constraint that is used in the HBM example design.

```
connect_debug_port dbg_hub/clk [get_nets */APB_0_PCLK]
set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
```

Configuration View

After programming the device, a list of IPs with a debug interface is displayed underneath the FPGA device in a tree. There will be one icon for each stack of HBM. If the IP is configured to utilize only one HBM stack, then only one icon will appear in the debug tree. The status of calibration is displayed to the right of the HBM instance as shown in the following figure:

Figure 16: **HBM Stack Property Window**
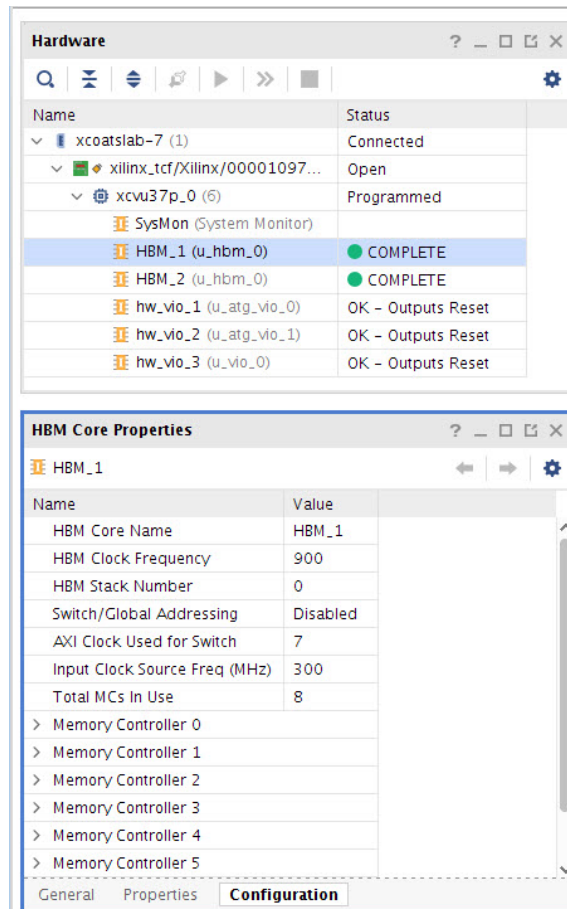
Selecting an HBM stack will change the Properties window with HBM specific information. There are three tabs in the HBM Core Properties view:

- **General tab:** The General tab lists the debug instance name (HBM_1) as shown in the following figure and the instance name. It also displays the calibration result.

- **Properties tab:** The Properties tab provides internal configuration register information. These registers are initialized when the hardware manager is opened. To load more current values, right click the HBM instance and select Refresh. To force an update to all register values, right click the HBM instance in the debug tree and select Refresh HBM Core. The register contents in the IEEE, MC, and PHY elements are in hex. Most of these registers are for internal use and may not be defined in this product guide.

- **Configuration tab:** The Configuration tab shows the options that are set for the HBM stack. The values mentioned here match with the values selected during the IP generation. A single HBM stack is partitioned and driven by 8 memory controllers, and each may be configured independently. To see the configuration options of an individual controller, double click the Memory Controller text to expand it.

Figure 17: **HBM Stack Configuration tab**
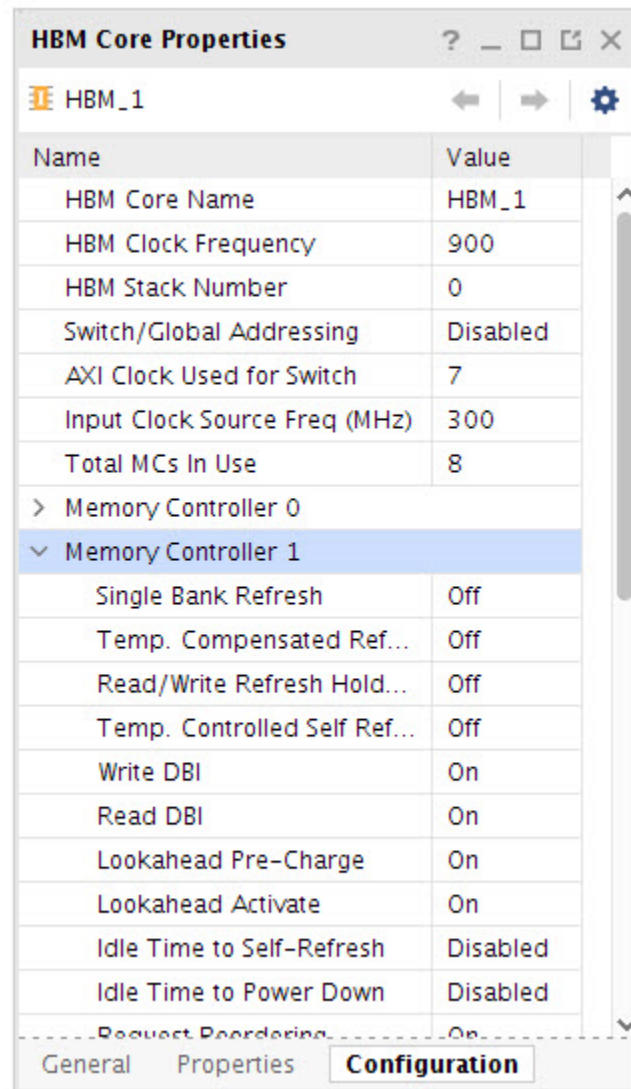
General Debugging Checks

Simulation Issues

If you experience issues while running the HBM simulation, perform the following checks:

- Only QuestaSIM, IES, and VCS are supported simulators for the AXI HBM Controller IP.

- Only behavioral simulations are supported by the AXI HBM Controller IP.

- Review the *Vivado Design Suite User Guide: Release Notes, Installation, and Licensing* (UG973) to ensure you are using a valid combination of operating system and simulator for the target Vivado® release.

- Verify that the proper definitions are added to the simulation test bench, as in AR 71097.

- Verify that the HBM test bench messages match those in AR 73372.

- If `apb_complete_x` never asserts, or if there is no activity in the simulation:

  ○ Verify that the HBM simulation MEM files, `xpm_internal_config_file_sim_x.mem`, are in the `project/project.srcs/sources_1/ip/hbm_0/hdl/rtl` directory. These MEM files contain the AXI HBM Controller IP configuration data.

  ○ Check the clocks and resets in the design.

Hardware Issues

If there are issues with the AXI HBM Controller IP in hardware, such as the HBM status in the Vivado Hardware Manager not being Complete or `apb_complete_x` never asserts, review the following:

- **Hardware Manager Status Not Enabled:** If the HBM status in the Vivado Hardware Manager shows as Not Enabled, it is likely that there is a connection problem between the Vivado Hardware Manager and the AXI HBM Controller IP.

  Ensure that the **Enable Hardware Debug Interface** option is selected in the Reliability Options tab of the AXI HBM Controller IP configuration options in the Vivado IDE, as shown in the following figure.

Figure 18: **Enable Hardware Debug Interface**
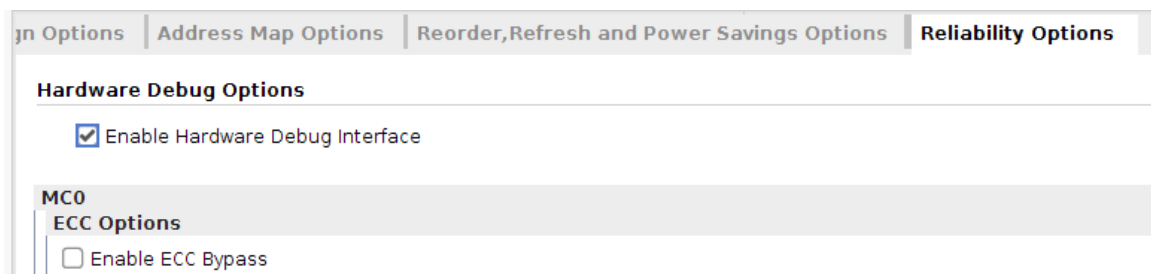
Ensure that there is a valid debug hub clock connected to the debug hub. The debug hub is a core generated by the Vivado place and route tools whenever any debug cores are detected in the design; see AR 72607 for more information.

Check the resets and reset polarities for the clock generators and APB ports in the design.

- **Hardware Manager Status Config Error:** If the HBM status in the Vivado Hardware Manager is Config Error, this might be caused by any of the following issues:

  - A clocking issue.

  - A timing issue.

  - The IP is using the incorrect MEM files for the hardware configuration.

- There is inadequate power for the HBM stacks.

- Error Correction with ECC Initialization enabled.

A Config Error usually means that the Vivado Hardware Manager can read data from the HBM, but something is preventing the IP from working correctly or the debug hub cannot reliably access the APB registers.

Check the IP configuration in the tools and verify the clock sources on the board. Ensure that the APB clocks are connected in the design and that the connected clock frequency matches the IP setting. Review the timing report for the design to make sure it is clean and timing has closed. Use a local low frequency clock for the HBM debug hub. If both HBM stacks are being used, set a false path from the APB clocks:

```
set_false_path -from [get_clocks *APB_0_PCLK] -to [get_clocks *APB_1_PCLK]
set_false_path -from [get_clocks *APB_1_PCLK] -to [get_clocks *APB_0_PCLK]
```

Check the individual memory controller statuses in the Vivado Hardware Manager by expanding the **HBM Core Properties** for each MC. Verify which ones are reporting `CTRLR_READY` and `CTRLR_INIT_DONE` are asserted. If decreasing the frequency, disabling a stack, or disabling more memory controllers causes more of them to pass, it is likely that there is a power issue on the board.

If the results are inconsistent from power cycle to power cycle, and the power supply on the board is adequate, it is likely that there is a timing issue on the debug hub or APB paths.

Review the HBM sections of the UltraScale+™ Xilinx® Power Estimator (XPE) and verify the power solution on the board can deliver adequate power for the targeted workloads. See the XPE web page for more information.

Review the *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics* (DS923) to make sure the HBM power rails are within specification.

If ECC is enabled with the ECC Initialization option, this takes a few additional seconds for the memory to perform the memory initialization after the interface has completed calibration. However the Hardware Manager is already querying the HBM for configuration status during this time thus a Configuration Error may be seen. Simply right-click on the **HBM** and select '**Refresh**' and if this is the issue, the calibration error will be replaced with 'Complete'.

- **The `apb_complete_x` signal never asserts:** If `apb_complete_x` never asserts, check the status in the Vivado Hardware Manager. This could be a timing, clocking, power, or configuration issue depending on the status and behavior.

- **The `apb_complete_x` signal asserts, but Vivado Hardware Manager reports Config Error or Not Enabled:** If the `apb_complete_x` signals are asserting in the design when monitored by some other logic or ILA, this means that the HBM stacks have completed calibration and are ready for traffic, but there is an issue with the debug hub connection. This could either be a clocking issue or a timing constraint issue. In most circumstances it is appropriate to use the `APB0` clock if routing is not a challenge. If routing is already congested, try generating another clock locally only for the debug hub.

  Note: This error might also be caused by a timing issue with the user logic in the design. It cannot be guaranteed that the user logic connected to the AXI ports is operating correctly while the HBM stacks report a ready status.

Activity Monitor

The Activity Monitor is a graphical display of the activity counters within the HBM controllers. The Hardware Manager can control and read these values to display the performance at the pseudo channel level. This can be used to quickly spot any performance variations between channels in real-time with live traffic without the need for any modification to your design. The activity counter values are displayed on a chart as well as in the summary information table. Information collected is also stored in a comma separated value (CSV) file, this file is saved in the directory that Vivado is launched from. The process of data collection is as follows:

1. You can select a pseudo channel (PC) to monitor.

2. HWM initializes the corresponding memory controllers to collect data for 450M samples. This value is configurable in the hardware but not adjustable in the HWM GUI.

3. HWM signals all enabled MCs to start the data collection process.

4. HWM polls the first MC waiting for the collection counter to be done.

5. HWM reads the activity data back from the first MC and restarts the counter to collect the next set of data.

6. HWM then checks the subsequent MCs and retrieves the activity data for all the selected MCs.

7. HWM displays the results and then restarts the polling process.

Note: The data collection process is not continuous. The data within any given sample window (450M memory clock cycles) is contiguous data. However, there is a gap in the data collection process from one sample to the next. The graphed result is an overall performance estimate based on time slices of measured data.

The summary table rows display information such as stack Temperature, Read, Write, and Overall throughput. Stack temperature is the temperature of the stack you selected. Read, Write, and Overall throughput are the sum of channels selected by you (not all the channels unless you select them). The summary table columns display information such as Latest, Min, Max, and Average. The Latest value is the most recent value read by the device. The Min and Max are the largest and smallest values sampled during the current Activity Monitor session. The Average is the average of all the data points during the current Activity Monitor session.

Figure 19: **Activity Monitor**
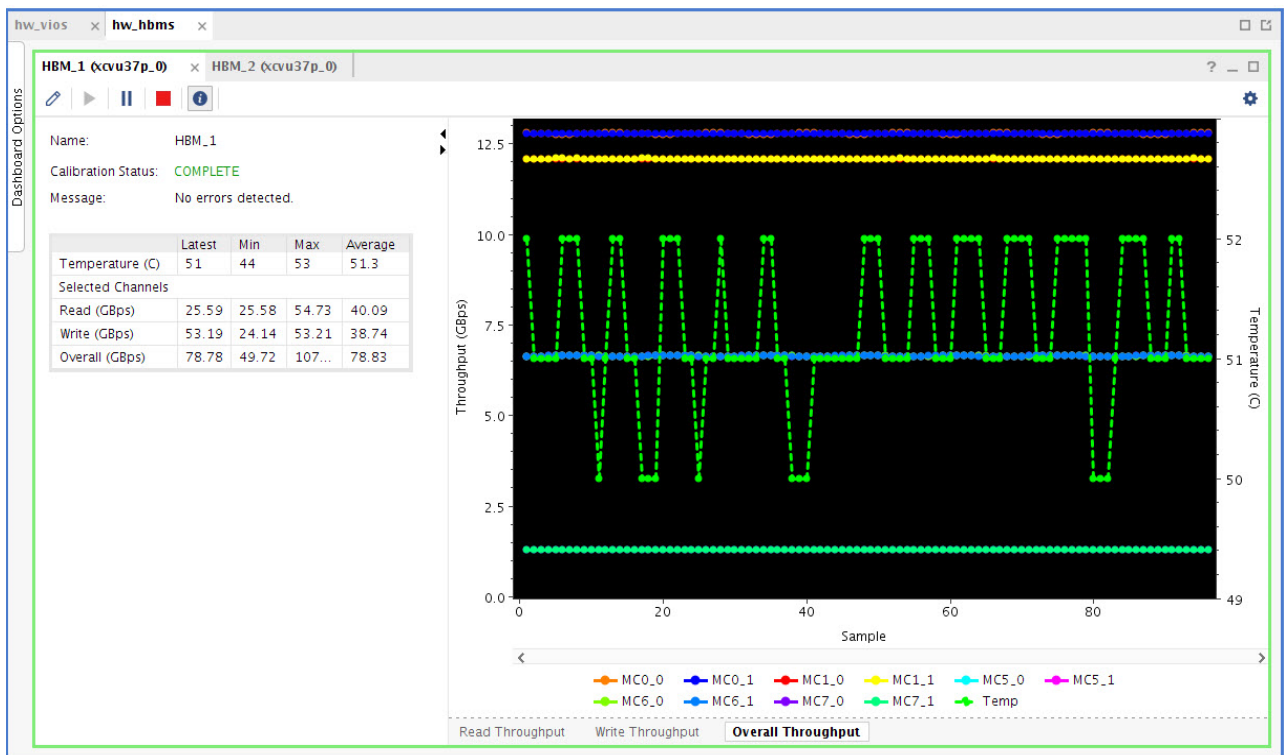
Using Activity Monitor

You can use the Activity Monitor to select channels and monitor activity on the channels you selected. Perform the following steps to use the Activity Monitor:

1. Click the ✏ icon to set up the Activity Monitor.

   Note: Alternatively, you can click **Set up Activity Monitor**.
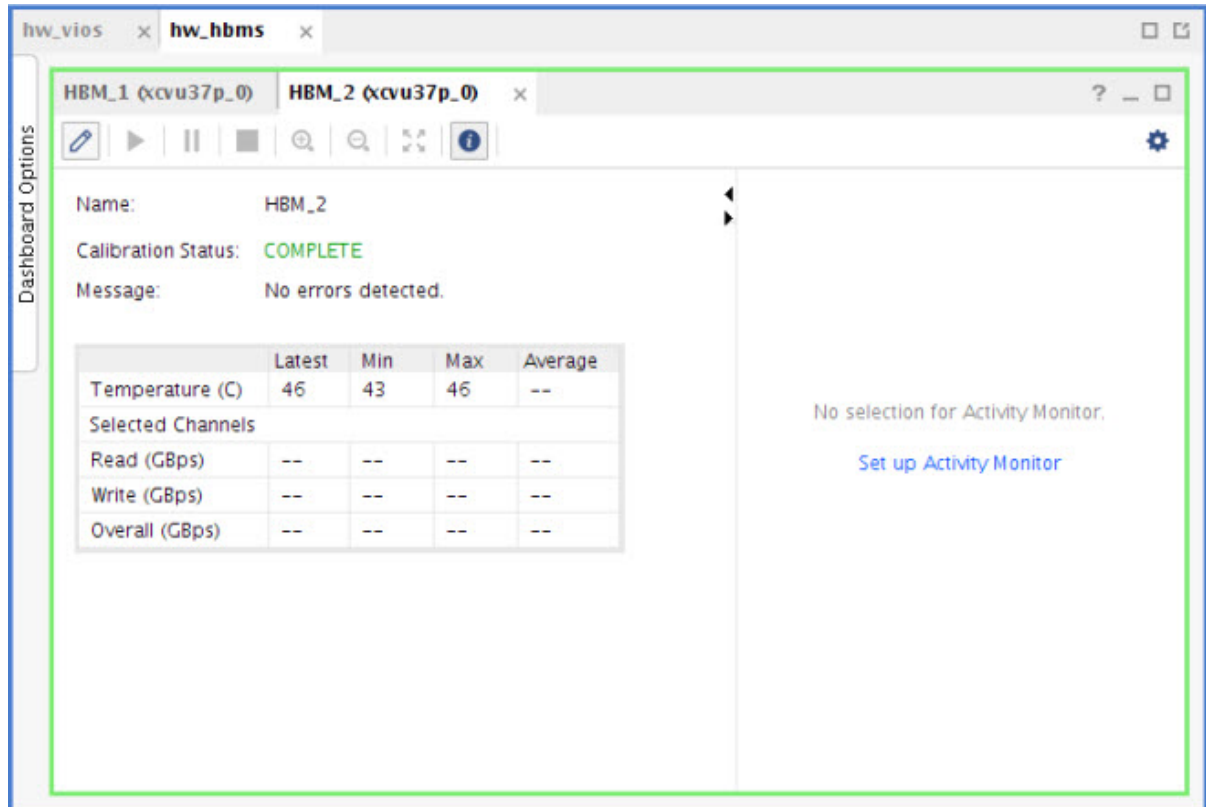
2. In the Set up Activity Monitor dialog box, select the channels you want to monitor.

   Note: Only the MCs enabled in the IP are displayed. Selecting the Channel Selector enables all channels. Selecting the MC checkbox enables both the pseudo channels PC_0 and PC_1 associated with that MC. Selecting the Show Selected Average check box adds an additional column that displays the average value of all the selected channels. Selecting the Show Temperature checkbox adds an additional row that displays the stack temperature in Celsius on the right side of the graph.

3. Click **OK**.
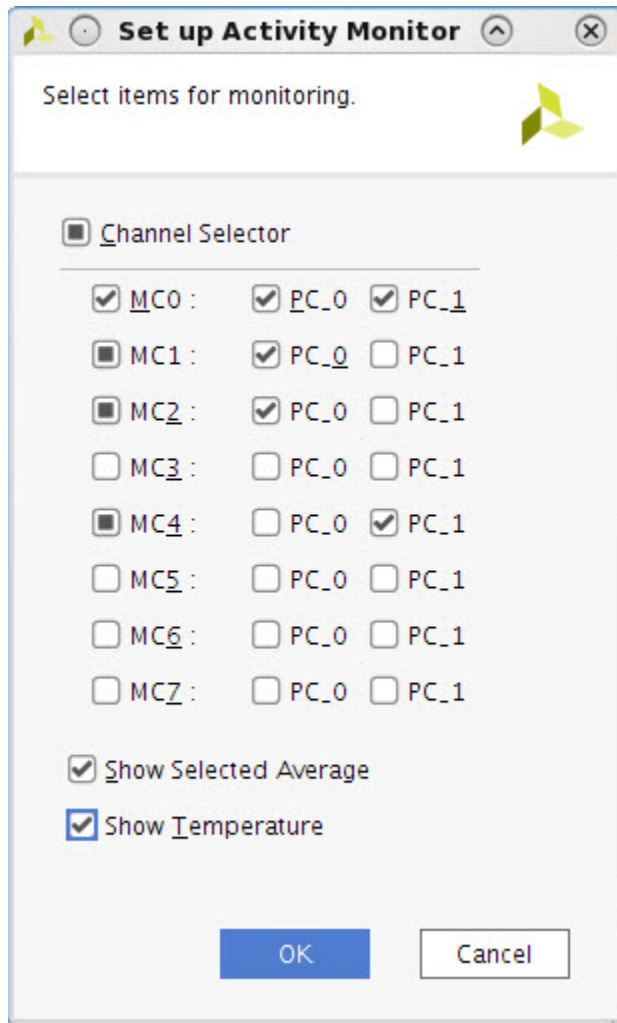
4. Click the ▶ icon to start the Activity Monitor.

    Note: Alternatively, you can click **Start Activity Monitor**.
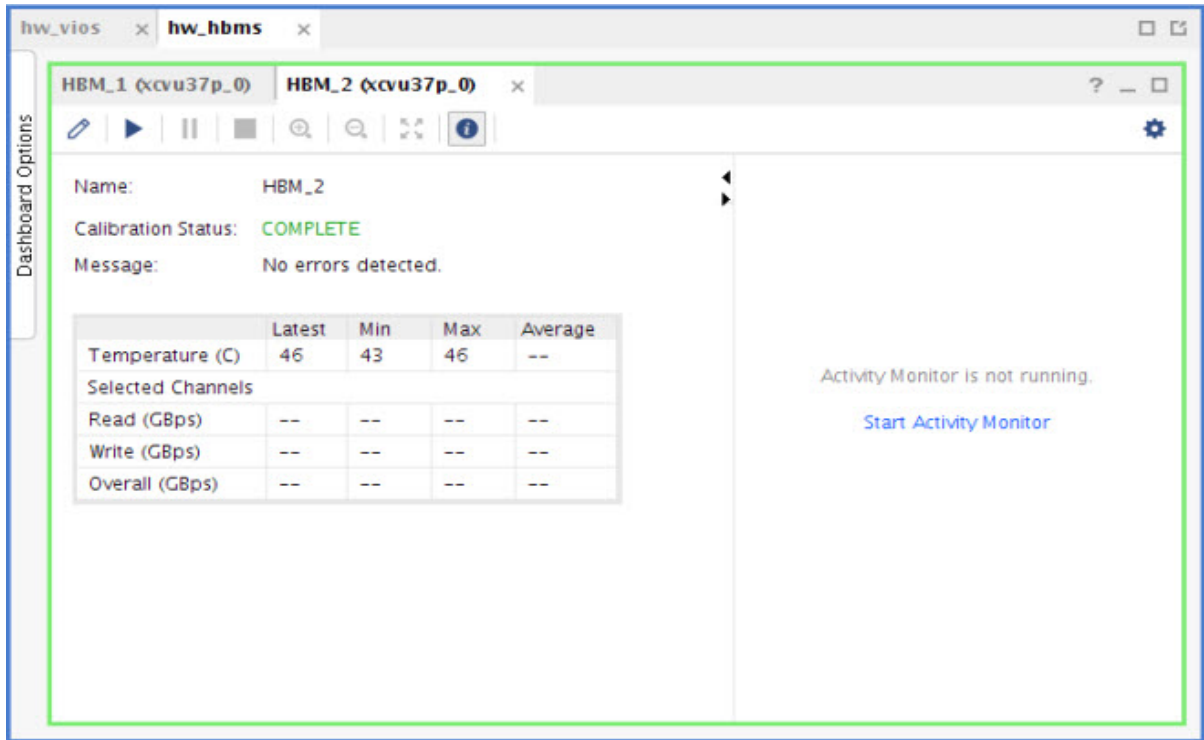
Note: The graph begins to plot. Read, Write, and Overall throughput are the three tabs available in the graph view. You can toggle between tabs to view the stored data.

5. Click the [pause icon] icon to pause the data collection.

Note: Click the pause icon to pause the HWM data collection after the current set is completed. Click it again to resume the data collection. When you pause the data collection, it inserts a gap between the data points and adds a white triangle at the top of the chart to indicate a break in time between the X and X+1 samples.
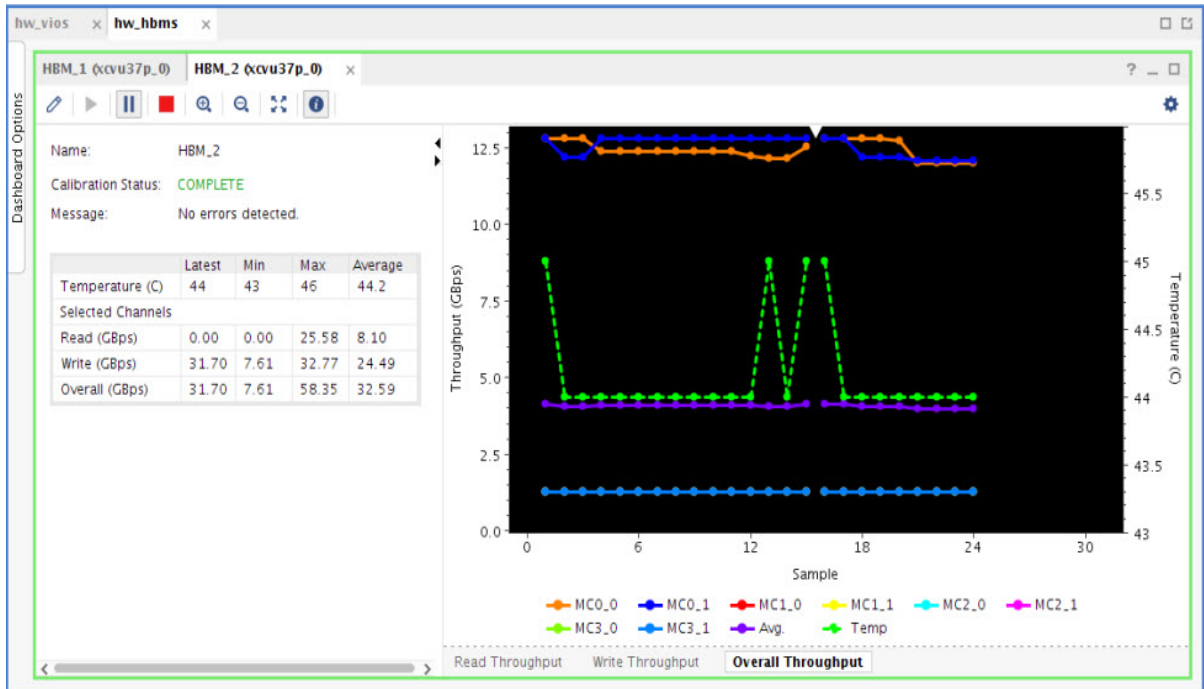
Note: Channels should not be removed or added while data collection is in progress including the paused state.

6. Click the ⬛ icon to stop the data collection.

7. Click the ⚙ icon to change the total number of samples to display on the chart in a given time.

   Note: You can adjust the total number of samples only when the activity monitor is not running or before the sample count reaches 50.
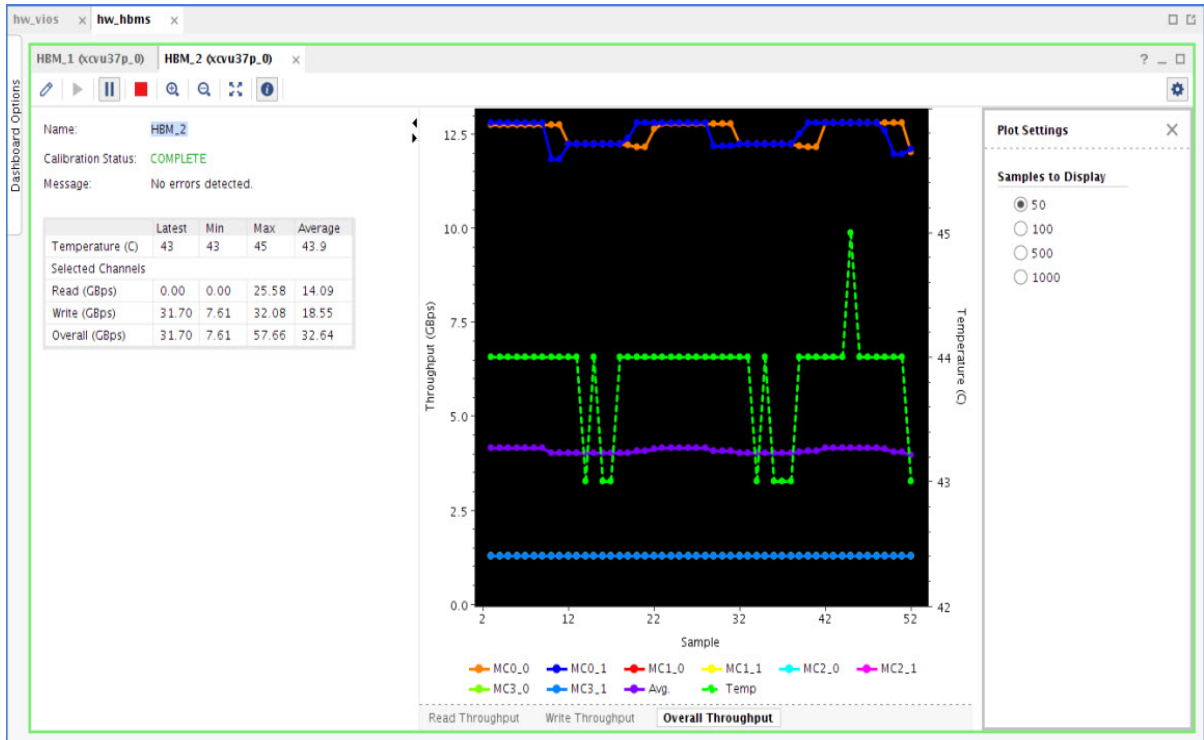
Data Log

The data collected to generate the graphs is stored in a comma separated value (CSV) format. The file name is `devicename_x_hbm_y_sz.csv` and it is written in the directory that the Vivado® software is launched from. If this directory is not writeable, the file is not generated, however, the activity monitor still runs. The `devicename_x` is the part name and its position in the JTAG chain, and `y` is the instance number of the HBM stack in the JTAG debug tree. `z` is the stack number and the value either 1 or 2. The location of the file is displayed in the TCL console when the activity monitor starts, and it overwrites the file upon starting the next session. If the information is important, you should rename the file or move it to a different location.

Note: Do not have the text file open in a separate program while the activity monitor is active. This results the activity monitor to run improperly.

Finding Help on Xilinx.com

To help in the design and debug process when using the core, 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. The Xilinx Community Forums are also available where members can learn, participate, share, and ask questions about Xilinx solutions.

Documentation

This product guide is the main document associated with the core. 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 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.

Master Answer Record for the AXI HBM Controller

AR 69267.

Technical Support

Xilinx provides technical support on the Xilinx Community Forums 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 ask questions, navigate to the Xilinx Community Forums.

Debug Tools

There are many tools available to address AXI HBM Controller design issues. It is important to know which tools are useful for debugging various situations.

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 to interact with the logic debug LogiCORE 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).

Memory Controller Register Map

The HBM memory controller configuration and status registers are accessible using an APB port. There is one APB port per stack and the 22-bit address space is not shared between stacks. The address space is segmented into individual regions which are addressed by the most significant five bits. The remaining 17 bits address the local space within the region. The _PS0 and _PS1 suffix in the register name denote the pseudo channel within the selected memory controller.

Table 19: **Region Addressing**

Upper 5 bit address	Region Name
01000	Memory Controller 0
01100	Memory Controller 1
01001	Memory Controller 2
01101	Memory Controller 3
01010	Memory Controller 4
01110	Memory Controller 5
01011	Memory Controller 6
01111	Memory Controller 7

Table 20: **ECC and Status Registers**

Register Name	Lower 17 bit Address	Definition
CFG_ECC_CORRECTION_EN	0x05800	D0 : Set this bit to correct 1-bit errors and detect 2-bit errors. Reset value is 1'b1.
INIT_ECC_SCRUB_EN	0x05804	D0 : If this bit is set, and if CFG_ECC_CORRECTION_EN is also set, then ECC scrubbing is enabled for all addresses in this memory controller. Single bit errors will be detected and corrected. Double bit errors will be detected.
CFG_ECC_SCRUB_PERIOD	0x05810	D[12..0] : Period between read operations for ECC scrubbing. This value is in units of 256 memory clock periods. A value of 0x02 means 512 memory clock periods between each read. Reset value is 13'h02.

Table 20: **ECC and Status Registers** *(cont'd)*

Register Name	Lower 17 bit Address	Definition
INIT_WRITE_DATA_1B_ECC_ERROR_GEN_PS0	0x0584c	D[3..0] : Setting one of these bits will instruct the Memory Controller to insert a single 1-bit ECC error on the next cycle of write data. The enabled bit selects which write of the BL4 has the error. For additional error generation, the bit must be reset then set again. Reset value is 4'h0.
INIT_WRITE_DATA_2B_ECC_ERROR_GEN_PS0	0x05850	D[3..0] : Setting one of these bits will instruct the Memory Controller to insert a single 2-bit ECC error on the next cycle of write data. The enabled bit selects which write of the BL4 has the error. For additional error generation, the bit must be reset then set again. Reset value is 4'h0.
INIT_WRITE_DATA_1B_ECC_ERROR_GEN_PS1	0x05854	D[3..0] : Setting one of these bits will instruct the Memory Controller to insert a single 1-bit ECC error on the next cycle of write data. The enabled bit selects which write of the BL4 has the error. For additional error generation, the bit must be reset then set again. Reset value is 4'h0.
INIT_WRITE_DATA_2B_ECC_ERROR_GEN_PS1	0x05858	D[3..0] : Setting one of these bits will instruct the Memory Controller to insert a single 2-bit ECC error on the next cycle of write data. The enabled bit selects which write of the BL4 has the error. For additional error generation, the bit must be reset then set again. Reset value is 4'h0.
STAT_ECC_ERROR_1BIT_CNT_PS0	0x05828	D[7..0] : A counter that increments whenever 1-bit ECC errors have been detected. Holds the value when maximum count has been reached (255) or until reset by INIT_ECC_ERROR_CLR. Reset value 8'b0.
STAT_ECC_ERROR_1BIT_CNT_PS1	0x05834	D[7..0] : A counter that increments whenever 1-bit ECC errors have been detected. Holds the value when maximum count has been reached (255) or until reset by INIT_ECC_ERROR_CLR. Reset value 8'b0
STAT_ECC_ERROR_2BIT_CNT_PS0	0x0582C	D[7..0] : A counter that increments whenever 2-bit ECC errors have been detected. Holds the value when maximum count has been reached (255) or until reset by INIT_ECC_ERROR_CLR. Reset value 8'b0.
STAT_ECC_ERROR_2BIT_CNT_PS1	0x05838	D[7..0] : A counter that increments whenever 2-bit ECC errors have been detected. Holds the value when maximum count has been reached (255) or until reset by INIT_ECC_ERROR_CLR. Reset value 8'b0

Table 20: **ECC and Status Registers** *(cont'd)*

Register Name	Lower 17 bit Address	Definition
INIT_ECC_ERROR_CLR	0x05818	D[0] : When set to 1 this will reset the STAT_ECC_ERR_1BIT_CNT_PSx registers. When set to 0 the counters will resume. Reset value 1'b0.
CFG_ECC_1BIT_INT_THRESH	0x0585C	D[7..0] : This register configures a count threshold that must be exceeded before STAT_INT_ECC_1BIT_THRESH is asserted and STAT_ECC_ERROR_1BIT_CNT_PSx begin to count. Reset value 8'b0.
STAT_INT_ECC_1BIT_THRESH_PS0	0x05864	D[0] : This bit is set when the number of 1-bit ECC errors exceeds the threshold defined in CFG_ECC_1BIT_INT_THRESH. Reading this register automatically clears it. Reset value 1'b0
STAT_INT_ECC_1BIT_THRESH_PS1	0x05868	D[0] : This bit is set when the number of 1-bit ECC errors exceeds the threshold defined in CFG_ECC_1BIT_INT_THRESH. Reading this register automatically clears it. Reset value 1'b0.
STAT_DFI_INIT_COMPLETE	0x10034	D[0] : This value is set to '1' when PHY initialization has completed. Reset value 1'b0.
STAT_DFI_CATTRIP	0x1004C	D[0] : This register will be set if the temperature ever exceeds the catastrophic value per HBM2 Jedec specification. Reset value 1'b0

Address decode example:

To read the STAT_ECC_ERROR_1BIT_CNT for Memory Controller 4, Pseudo Channel 1, look for the STAT_ECC_ERROR_1BIT_CNT_PS1 register. The lower 17-bit address is 0x05834. Memory Controller 4 has an upper 5-bit address value of 5'b01010. Prepending these 5 bits yields the address 22'h145834.

Send Feedback

Activity Monitor Control/Status Registers

Each Memory Controller contains counters for tracking usage statistics. This information may be useful for monitoring performance under varying real-time traffic conditions. Registers are 32 bits unless specified otherwise in the register definition.

The Activity Monitor can be used in one of two modes defined by INIT_AM_REPEAT_EN:

1. Repeating interval – Usage statistics are collected over a configurable time interval and saved to registers after completion of the interval. This process then automatically repeats.

2. Single interval – Usage statistics are collected over a configurable time interval and saved to registers after completion of the interval. This process only occurs once.

Typical repeating usage would be as follows:

1. Configure the capture interval (CFG_AM_INTERVAL) to the desired value.

2. Set the INIT_AM_REPEAT_EN to both configure and initiate the repeating data collection.

3. Poll on STAT_AM_COMPLETE to know when the collection has completed and the tracking registers have updated data. Note that STAT_AM_COMPLETE is self-clearing. Thus, after a 1 is read back there is no need to reset the register for the next cycle.

4. Read back the desired tracking registers for analysis.

5. Return to step 3.

It is important to set the CFG_AM_INTERVAL to a value long enough such that you have enough time to detect completion and read back all of the desired tracking registers before the next cycle completes and the tracking registers are overwritten with new data.

A single interval use would be as follows:

1. Configure the capture interval (CFG_AM_INTERVAL) to the desired value.

2. Set the INIT_AM_REPEAT_EN to 0.

3. Assert the INIT_AM_SINGLE_EN register to initiate the single interval data collection.

4. Poll on STAT_AM_COMPLETE to know when the collection has completed and the tracking registers have updated data. Note that STAT_AM_COMPLETE is self clearing. Thus, after a 1 is read back there is no need to reset the register for the next cycle.

5. Read back the desired tracking registers for analysis.

Table 21: **Activity Monitor Control/Status Registers**

Register Name	Local Address	Definition
INIT_AM_REPEAT	0x13800	D[0] Set to 1 to initiate the repeating interval data collection
INIT_AM_SINGLE_EN	0x13804	D[0] Set to 1 to initiate a single interval data collection

Send Feedback

Table 21: **Activity Monitor Control/Status Registers** *(cont'd)*

Register Name	Local Address	Definition
CFG_AM_INTERVAL	0x13808	D[31..0] Set the activity monitor interval, in memory clocks
STAT_AM_COMPLETE	0x1380c	D[0] This is set to 1 when the interval has completed. This register is cleared on Auto-Precharge.

Table 22: **Activity Monitor Tracking Registers**

Register Name	Local Address	Definition
AM_WR_CMD_PS0	0x13814	Number of cmd=Write commands captured in the last monitoring interval. Note that this counts writes without Auto-Precharge, since writes with Auto-Precharge are a different command. For total Write commands, sum the two counts.
AM_WR_CMD_PS1	0x13818	Number of cmd=Write commands captured in the last monitoring interval. Note that this counts writes without Auto-Precharge, since writes with Auto-Precharge are a different command. For total Write commands, sum the two counts.
AM_WR_AP_CMD_PS0	0x13820	Number of cmd=Write-with-Auto-Precharge commands captured in the last monitoring interval.
AM_WR_AP_CMD_PS1	0x13824	Number of cmd=Write-with-Auto-Precharge commands captured in the last monitoring interval.
AM_RD_CMD_PS0	0x1382c	Number of cmd=Read commands captured in the last monitoring interval. Note that this counts reads without Auto-Precharge, since reads with Auto-Precharge are a different command. For total Read commands, sum the two counts.
AM_RD_CMD_PS1	0x13830	Number of cmd=Read commands captured in the last monitoring interval. Note that this counts reads without Auto-Precharge, since reads with Auto-Precharge are a different command. For total Read commands, sum the two counts.
AM_RD_AP_CMD_PS0	0x13838	Number of Read with Auto-Precharge commands captured in the last monitoring interval.
AM_RD_AP_CMD_PS1	0x1383c	Number of Read with Auto-Precharge commands captured in the last monitoring interval.
AM_REFRESH_CMD_PS0	0x13844	Number of Refresh commands captured in the last monitoring interval.
AM_REFRESH_CMD_PS1	0x13848	Number of Refresh commands captured in the last monitoring interval.
AM_ACT_CMD_PS0	0x13850	Number of Activate commands captured in the last monitoring interval.
AM_ACT_CMD_PS1	0x13854	Number of Activate commands captured in the last monitoring interval.
AM_PRECHARGE_CMD_PS0	0x1385c	Number of Precharge (single-bank) commands captured in the last monitoring interval.
AM_PRECHARGE_CMD_PS1	0x13860	Number of Precharge (single-bank) commands captured in the last monitoring interval.

Table 22: **Activity Monitor Tracking Registers** *(cont'd)*

Register Name	Local Address	Definition
AM_PRECHARGE_ALL_CMD_PS0	0x13868	Number of times any Read command (Read or Read with Auto-Precharge) is followed by any Write command (Write or Write with Auto-Precharge) in the last monitoring interval.
AM_PRECHARGE_ALL_CMD_PS1	0x1386c	Number of times any Read command (Read or Read with Auto-Precharge) is followed by any Write command (Write or Write with Auto-Precharge) in the last monitoring interval.
AM_POWER_DOWN	0x13870	Number of clock cycles the memory is in power-down in the last monitoring interval.
AM_SELF_REFRESH	0x13874	Number of clock cycles the memory is in self-refresh in the last monitoring interval.
AM_RD_TO_WR_SWITCH_PS0	0x1387c	Number of times any Read command (Read or Read with Auto-Precharge) is followed by any Write command (Write or Write with Auto-Precharge) in the last monitoring interval.
AM_RD_TO_WR_SWITCH_PS1	0x13880	Number of times any Read command (Read or Read with Auto-Precharge) is followed by any Write command (Write or Write with Auto-Precharge) in the last monitoring interval.
AM_RO_AGE_LIMIT_PS0	0x13888	Number of times the reorder queue entry reaches its age limit in the last monitoring interval.
AM_RO_AGE_LIMIT_PS1	0x1388c	Number of times the reorder queue entry reaches its age limit in the last monitoring interval.
AM_RMW_CYCLE_PS0	0x13894	Number of Read Modify Write cycles in the last monitoring interval.
AM_RMW_CYCLE_PS1	0x13898	Number of Read Modify Write cycles in the last monitoring interval.

Send Feedback

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 (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open 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 DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

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

References

These documents provide supplemental material useful with this product guide:

Send Feedback

1. AMBA AXI Protocol Specification
2. Arm AMBA APB Protocol v2.0 Specification (ARM IHI 0024C)
3. Arm AMBA AXI Protocol v2.0 Specification (ARM IHI 0022C)
4. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide: Designing with IP* (UG896)
6. *Vivado Design Suite User Guide: Getting Started* (UG910)
7. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
8. *ISE to Vivado Design Suite Migration Guide* (UG911)
9. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
10. *Vivado Design Suite User Guide: Implementation* (UG904)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
08/06/2021 Version 1.0	
System Level Considerations	Added the use of BRESP necessary to ensure coherency
Lateral AXI Switch Access Throughput	Added data adding conditions in order to prevent switch blocking
General Debugging	If ECC Initialization is used, refresh of Hardware Manager after programming may be required
01/21/2021 Version 1.0	
Appendix C: Memory Controller Register Map	Added information about new ECC and Status Registers.
Data Path Error Protection	Added more information about BRESP error and RRESP error.
AXI Port Details	Added more information about read, and write transactions.
Lateral AXI Switch Access Throughput Loss	Added information about transactions.
07/31/2020 Version 1.0	
HBM Performance Concepts	Added information about HBM topology, raw throughput evaluation, AXI considerations, HBM address map and protocol considerations, and system-level considerations.
Non-Synthesizable Traffic Generator Considerations for Workload Simulation	Added details of Write, Read, Wait, and Start and End Loop commands.
PHY Only Mode	Explanation of PHY only mode.
General Debugging Checks	Added details of checks to perform for simulation and hardware issues.
10/30/2019 Version 1.0	
Chapter 6: Example Design	Added new section Synthesizable Traffic Generator

Section	Revision Summary
08/07/2019 Version 1.0	
General updates	Editorial updates only. No technical content updates
03/28/2019 Version 1.0	
General updates	Updated Table 5 and added a table note. Updated Clocking section.
12/05/2018 Version 1.0	
General updates	Updated figures 3, 4, 5, 6, 7, 8, 9, 10, and 11. Updated AR link.
Design Flow Steps	Added ECC bypass feature. Added pre-defined settings for different traffic.
Product Specification	Added Lateral AXI Switch Access Throughput section. Added Data Path Error Protection section.
Debugging	Added Hardware Manager - HBM Debug Interface section.
04/04/2018 Version 1.0	
Initial Xilinx release.	N/A

Please Read: Important Legal Notices

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.

Copyright