High Bandwidth Memory (HBM2) Interface Intel® FPGA IP User Guide

Updated for Intel® Quartus® Prime Design Suite: 18.0
## Contents

1. Introduction to High Bandwidth Memory
   1.1. HBM in Intel Stratix 10 MX Devices
   1.2. HBM2 DRAM Structure
   1.3. Intel Stratix 10 MX HBM2 Features
   1.4. Intel Stratix 10 MX HBM2 Controller Features

2. Intel Stratix 10 MX HBM2 Architecture
   2.1. Intel Stratix 10 MX HBM2 Introduction
   2.2. Intel Stratix 10 MX HBM2 Architecture
   2.3. Intel Stratix 10 MX HBM2 Controller Architecture
      2.3.1. Intel Stratix 10 MX HBM2 Controller Details

3. Generating the High Bandwidth Memory (HBM2) Interface Intel FPGA IP
   3.1. Parameterizing the High Bandwidth Memory (HBM2) Interface Intel FPGA IP
   3.2. General Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP
   3.3. Controller Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP
   3.4. Diagnostic Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP
   3.5. Example Designs Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP
   3.6. Generating the Example Design
   3.7. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Design Example
   3.8. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Pin Planning

4. Simulating the High Bandwidth Memory (HBM2) Interface Intel FPGA IP
   4.1. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Example Design
   4.2. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with ModelSim* and Questa*
   4.3. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with Synopsys VCS*
   4.4. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with Riviera-PRO*
   4.5. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with Cadence NCSim*
   4.6. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with Cadence Xcelium*
   4.7. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP for High Efficiency

5. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Interface
   5.1. High Bandwidth Memory (HBM2) Interface Intel FPGA IP High Level Block Diagram
   5.2. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Controller Interface Signals
      5.2.1. Clock Signals
      5.2.2. Reset Signals
      5.2.3. Calibration Status Signals
      5.2.4. Memory Interface Signals
      5.2.5. AXI User-interface Signals
      5.2.6. Sideband APB Interface
   5.3. User AXI Interface Timing
      5.3.1. AXI Write Transaction
5.3.2. AXI Read Transaction.................................................................44
5.4. User APB Interface Timing...........................................................45
  5.4.1. Advanced Peripheral Bus Protocol........................................45
  5.4.2. APB Interface Timing............................................................46
5.5. User-controlled Accesses to the HBM2 Controller.........................47
  5.5.1. User-controlled Refreshes......................................................48
  5.5.2. Temperature and Calibration Status Readout........................52
  5.5.3. Power Down Status..............................................................53
  5.5.4. ECC Error Status.................................................................53
  5.5.5. User Interrupt.................................................................54
  5.5.6. ECC Error Correction and Detection....................................57

6. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Controller Performance...... 60
  6.1. High Bandwidth Memory (HBM2) DRAM Bandwidth.....................60
  6.2. High Bandwidth Memory (HBM2) Interface Intel FPGA IP HBM2 IP Efficiency...........60
  6.3. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Latency ......................62
  6.4. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Timing......................62
  6.5. High Bandwidth Memory (HBM2) Interface Intel FPGA IP DRAM Temperature Readout...62

7. Document Revision History for High Bandwidth Memory (HBM2) Interface Intel
   FPGA IP User Guide............................................................................. 63
1. Introduction to High Bandwidth Memory

High Bandwidth Memory (HBM) is a JEDEC specification (JESD-235) for a wide, high bandwidth memory device. The next generation of High Bandwidth Memory, HBM2, is defined in JEDEC specification JESD-235A. The HBM2 implementation in Intel® Stratix® 10 MX devices complies to JESD-235A.

The High Bandwidth Memory DRAM is tightly coupled to the host die with a distributed interface. The interface is divided into independent channels, each completely independent of one another. Each channel interface maintains a 128-bit data bus, operating at DDR data rates.

1.1. HBM2 in Intel Stratix 10 MX Devices

Intel Stratix 10 MX incorporates a high-performance FPGA fabric along with a HBM2 DRAM in a single package. Intel Stratix 10 MX devices support up to a maximum of two HBM2 interfaces.

Intel Stratix 10 MX incorporates Intel's Embedded Multi-Die Interconnect Bridge (EMIB) technology to implement a silicon bridge between HBM2 DRAM memory and the Universal Interface Block Subsystem (UIBSS), which contains the HBM2 controller (HBMC), physical-layer interface (PHY), and I/O ports to interface to the HBM2 stack.

As illustrated below, each Intel Stratix 10 MX device contains a single universal interface bus per HBM2 interface, supporting 8 independent channels.

The user interface to the HBM2 controller is maintained through the AIX4 protocol. Sixteen AXI interfaces are available in the user interface from each HBM2 controller, with one AXI interface available per HBM2 Pseudo Channel. HBM2 DRAM density of 4GB and 8GB are supported.

Figure 1. Intel Stratix 10 MX Device with UIB, EMIB, and HBM2 DRAM
1.2. HBM2 DRAM Structure

The HBM DRAM is optimized for high-bandwidth operation to a stack of multiple DRAM devices across several independent interfaces called channels. Each DRAM stack supports up to eight channels.

The following figure shows an example stack containing four DRAM dies, each die supporting two channels. Each die contributes additional capacity and additional channels to the stack, up to a maximum of eight channels per stack. Each channel provides access to an independent set of DRAM banks. Requests from one channel may not access data attached to a different channel.

Figure 2. High Bandwidth Memory Stack of Four DRAM Dies

1.3. Intel Stratix 10 MX HBM2 Features

Intel Stratix 10 MX FPGAs offer the following HBM2 features.

- Supports one to eight HBM2 channels per HBM2 interface in the Pseudo Channel mode.
- Each HBM2 channel supports a 128-bit DDR data bus, with optional ECC support.
- Pseudo Channel mode divides a channel into two individual 64-bit data interfaces per channel. The Pseudo Channels share the same Address and Command bus, but decodes and executes commands individually.
- Data referenced to strobes \( \text{RDQS}_{t}/\text{RDQS}_{c} \) and \( \text{WDQS}_{t}/\text{WDQS}_{c} \), one strobe pair per 32 DQs.
- Differential clock inputs \( \text{CK}_{t}/\text{CK}_{c} \). Untermminated data/address/cmd/clk interfaces.
• DDR commands entered on each positive CK_t and CK_c edge. Row Activate commands require two memory cycles; all other command are single-cycle commands.
• Supports command, write data and read data parity.
• Support for bank grouping.
• Support for data bus inversion.
• 64-bit data per Pseudo Channel. Eight additional data bits are available per Pseudo Channel, you can use these data bits for any of the following:
  — ECC. The ECC scheme implemented is single-bit error correction with double-bit error detection (SECDEC). This includes 8 bits of ECC code (also known as syndrome).
  — Data mask (DM). The data mask for masking write data per byte.
  — Can be left unused.
• I/O voltage of 1.2V and DRAM core voltage of 1.2V.

1.4. Intel Stratix 10 MX HBM2 Controller Features

Intel Stratix 10 MX FPGAs offer the following controller features.
• User applications communicate with the HBMC using the AXI4 Protocol.
• There is one AXI4 interface per HBM2 Pseudo Channel. Each HBM2 interface supports a maximum of sixteen AXI4 interfaces to the sixteen Pseudo Channels.
• The user interface can operate at a frequency lower than the HBM2 interface frequency. For information on supported clock frequencies, refer to Intel Stratix 10 MX HBM2 Supported Frequencies in Intel Stratix 10 MX HBM2 IP Controller Interface Signals.
• Each AXI interface supports a 256-bit Write Data interface and a 256-bit Read Data interface.
• The controller offers 32B and 64B access granularity supporting burst length 4 (BL 4) and pseudo-BL 8 (two back to back BL4).
• The controller offers out-of-order command scheduling and read data reordering.
• The controller supports user-initiated Refresh commands, and access to the HBM2 channel status registers, through the side band Advanced Peripheral Bus (APB) interface.
• The controller supports data mask or error correction code (ECC). When you do not use data mask or ECC, you may use those bits as additional data bits.

Related Information
Clock Signals on page 33
2. Intel Stratix 10 MX HBM2 Architecture

This chapter provides an overview of the Intel Stratix 10 MX HBM2 architecture.

2.1. Intel Stratix 10 MX HBM2 Introduction

Intel Stratix 10 MX devices use the Intel EMIB technology to interface to the HBM2 memory devices.

- The Intel Stratix 10 MX FPGAs offer up to two HBM2 interfaces.
- Each HBM2 device can have a device density of 4GB or 8GB, based on the FPGA chosen.

This system-in-package solution helps to achieve maximum bandwidth and low power consumption in a small footprint.

2.2. Intel Stratix 10 MX HBM2 Architecture

The Intel Stratix 10 MX device architecture includes the universal interface bus (UIB) subsystem (UIBSS) which contains the necessary logic to interface the FPGA core to the HBM2 DRAM.

Each UIB subsystem includes the HBM2 hardened controller and the universal interface bus, consisting of the hardened physical interface and I/O logic needed to interface to each HBM2 DRAM device. The AMBA AXI4 protocol interfaces the core logic with the universal interface bus subsystem. An optional soft logic adapter implemented in the FPGA fabric helps to efficiently interface user logic to the hardened HBM2 controller.

The following figure shows a high-level block diagram of the Intel Stratix 10 HBM2 universal interface bus subsystem. The UIB subsystem includes the following hardened logic:

- Rate-matching FIFOs that transfer logic from the user core clock to the HBM2 clock domain.
- HBM2 memory controller (HBMC).
- UIB PHY, including the UIB physical layer and I/O.
Figure 3. Block Diagram of Intel Stratix 10 MX HBM2 Implementation

The user core clock drives the logic highlighted in green, while the UIB clocks the logic highlighted in blue. The UIB clock also drives the HBM2 interface clock. User logic can run up to one-to-four times slower than the HBM2 interface.

Soft Logic AXI Adaptor

The HBM2 IP also includes a soft logic adaptor implemented in core logic. The soft logic adaptor gates the user valid signals (write address valid, write data valid, and read address valid) with the corresponding pipelined ready signals from the HBM2 controller. The soft logic adapter also temporarily stores output from the HBM2 controller (AXI write response and AXI read data channels) when the AXI ready signal is absent. You can disable the temporary storage logic if user logic is always ready to accept output from the HBM2 controller through the parameter editor when generating the HBM2 IP.

HBM2 DRAM

The HBM2 DRAM is ideal for high-bandwidth operation to multiple DRAM devices across many independent interfaces called channels. Each channel provides access to an independent set of DRAM banks. Requests from one channel cannot access data attached to another channel.

Each HBM2 channel consists of an independent command and data interface to and from the HBM2 DRAM. A channel provides access to a discrete pool of memory in the DRAM device; no channel can access the memory storage for another channel. Each channel interface provides an independent interface to a specific number of banks of DRAM of a defined page size.

The following table lists the HBM2 signals that interface to the UIB. The UIB drives the HBM2 signals and decodes the received data from the HBM2. These signals cannot be accessed through the AXI4 User Interface.
Table 1. Summary of Per-channel Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Signal Width</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>128</td>
<td>128 bit bidirectional DQ per channel</td>
</tr>
<tr>
<td>Column command/address</td>
<td>8</td>
<td>8-bit wide column address bits</td>
</tr>
<tr>
<td>Row command/address</td>
<td>6</td>
<td>6-bit wide row address bits</td>
</tr>
<tr>
<td>DBI</td>
<td>16</td>
<td>1 DBI per 8 DQs</td>
</tr>
<tr>
<td>DM_CB</td>
<td>16</td>
<td>1 DM per 8 DQs. You can use these pins for DM or ECC, but not both.</td>
</tr>
<tr>
<td>PAR</td>
<td>4</td>
<td>1 parity bit per 32 DQs</td>
</tr>
<tr>
<td>DERR</td>
<td>4</td>
<td>1 data error bit per 32 DQs</td>
</tr>
<tr>
<td>Strobes</td>
<td>16</td>
<td>Separate strobes for read and write strobes. One differential pair per 32 DQs for read and write.</td>
</tr>
<tr>
<td>Clock</td>
<td>2</td>
<td>Clocks address and command signals</td>
</tr>
<tr>
<td>CKE</td>
<td>1</td>
<td>Clock enable</td>
</tr>
<tr>
<td>AERR</td>
<td>1</td>
<td>Address error</td>
</tr>
</tbody>
</table>

The following table lists the HBM2 signals that are common to all Pseudo Channels in each HBM2 interface. The HBM2 controller interfaces with the following signals; these signals are not available at the AXI4 user interface.

Table 2. Summary of Global HBM2 Signals

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Signal Width</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>1</td>
<td>Reset input</td>
</tr>
<tr>
<td>TEMP</td>
<td>3</td>
<td>Temperature output from HBM2.</td>
</tr>
<tr>
<td>Cattrip</td>
<td>1</td>
<td>Catastrophic temperature sensor.</td>
</tr>
</tbody>
</table>

The Intel Stratix 10 MX HBM2 IP supports only the Pseudo Channel mode of the HBM2 specification. Pseudo Channel mode includes the following features:

- Pseudo Channel mode divides a single HBM2 channel into two individual subchannels of 64 bit I/O.
- Both Pseudo Channels share the channel’s row and column command bus, CK, and CKE inputs, but decode and execute commands individually.
- Pseudo Channel mode requires a burst length of 4.
- Address BA4 directs commands to either Pseudo Channel 0 (BA4 = 0) or Pseudo Channel 1 (BA4 = 1). The HBM2 controller handles the addressing requirements of the Pseudo Channels.
- Power-down and self-refresh are common to both Pseudo Channels, due to a shared CKE pin. Both Pseudo Channels also share the channel’s mode registers.

Each Intel Stratix 10 MX HBM2 interface supports a maximum of eight HBM2 channels. Each HBM2 channel has two AXI4 interfaces, one per Pseudo Channel. Each AXI4 interface includes a 256-bit wide Write and Read Data interface per Pseudo Channel. The following figure shows the flow of data from user logic to the HBM2 DRAM through the UIBSS, while selecting HBM2 channels 0 and 7.
There is one AXI interface per Pseudo Channel. Each AXI interface supports a 256-bit wide Write Data interface and a 256-bit wide Read Data interface, to and from the HBM2 controller. The AXI4 protocol can handle concurrent writes and reads to the HBM2 controller. There is also a sideband user port per user channel pair, compliant to the Advanced Peripheral Bus (APB). The sideband provides access to user-controlled features such as refresh requests, ECC status, Power Down status, HBM2 temperature readout, calibration status, and User Interrupt.

Related Information
Intel Stratix 10 MX HBM2 Controller Details on page 11

2.3. Intel Stratix 10 MX HBM2 Controller Architecture

The hardened HBM2 controller provides a controller per Pseudo Channel.

Each controller consists of a write and read data path and the control logic that helps to translate user commands to the HBM2 memory. The HBM2 controller logic accounts for the HBM2 memory specification timing and schedules commands in an efficient manner. The following figure shows a block diagram of the HBM2 controller, corresponding to channel 0. The HBM2 controller’s user-logic interface follows the AXI4 protocol. You can find more information about the interface timing details in the User AXI Interface Timing section.
2.3.1. Intel Stratix 10 MX HBM2 Controller Details

This topic explains some of the high level HBM2 controller features.

HBM2 burst transactions

The HBM2 controller supports only the Pseudo Channel mode of accessing the HBM2 device; consequently, it can only support BL4 transactions to the DRAM. For improving efficiency, it supports the pseudo-BL8 mode, which helps to provide two back-to-back BL4 data using a given start address, similar to a BL8 transaction.

Each BL4 transaction corresponds to 4*64 bits or 32 bytes and a BL8 transaction corresponds to 64 bytes per Pseudo Channel. You can select the burst transaction mode (32 B vs 64B) through the parameter editor.

The user logic can interface to a maximum of 16 Pseudo Channels (16 AXI ports) per HBM2 interface. Each AXI port has a separate write and read interface, and can handle write and read requests concurrently at the same clock. Each write and read data interface per AXI port is 256 bits wide; that is, each AXI Write or Read Data transaction corresponds to data transfers corresponding to two HBM2 memory clock cycles.
User interface vs HBM2 Interface Frequency

The user interface runs at a frequency lower than the HBM2 interface; the maximum interface frequency depends on the chosen device speed grade and the FPGA core logic frequency. The rate-matching FIFOs within the UIB subsystem handle the data transfer between the two clock domains.

Command Priority

You can set command priority for a write or read command request through the AXI interface, through the qos signal in the AXI write address channel, or in the AXI read address channel. The HBM2 controller supports normal and high priority levels. The system executes commands with the same priority level in a round-robin scheme.

Starvation limit

The controller tracks how long each command waits and leaves no command unserviced in the command queue for a long period of time. The controller ensures that it serves every command efficiently.

Command scheduling

The HBM2 controller schedules the incoming commands to achieve maximum efficiency at the HBM2 interface. The HBM2 controller also follows the AXI ordering model of the AXI4 protocol specification.

Data re-ordering

The controller can reorder read data to match the order of the read requests.

Address ordering

The HBM2 controller supports different address ordering schemes that you can select for best efficiency given your use case. The chosen addressing scheme determines the order of address configurations in the AXI write and read address buses, including row address, column address, bank address, and stack ID (applicable only to the 8H devices). The HBM2 controller remaps the logical address of the command to physical memory address.
Thermal Control

The HBM2 controller uses the TEMP and CATTRIP outputs from the HBM2 device to manage temperature variations in the HBM2 interface.

- Temperature compensated refresh (TEMP): The HBM2 DRAM provides temperature compensated refresh information to the controller through the TEMP[2:0] pins, which defines the proper refresh rate that the DRAM expects to maintain data integrity. Absolute temperature values for each encoding are vendor-specific. The encoding on the TEMP[2:0] pins reflects the required refresh rate for the hottest device in the stack. The TEMP data updates when the temperature exceeds vendor-specified threshold levels appropriate for each refresh rate.

- Catastrophic temperature sensor (CATTRIP): The CATTRIP sensor detects whether the junction temperature of any die in the stack exceeds the catastrophic trip threshold value CATTEMP. The device vendor programs the CATTEMP to a value less than the temperature at which permanent damage to the HBM stack would occur.

  If a junction temperature anywhere in the stack exceeds the CATTEMP value, the HBM stack drives the external CATTRIP pin to 1, indicating that catastrophic damage may occur. When the CATTRIP pin is at 1, the controller stops all traffic to HBM and stalls indefinitely. To resolve the overheating situation and return the CATTRIP value to 0, remove power from the device and allow sufficient time for the device to cool before again applying power.

- Thermal throttling: Thermal throttling is a controller safety feature that helps control thermal runaway if the HBM2 die overheats, preventing a catastrophic failure. You can specify the HBM2 device junction temperature at which the controller begins to throttle input commands, and the throttle ratio that determines the throttle frequency. The controller deasserts the AXI ready signals (awready, wready and arready) when it is actively throttling the input commands and data.

Refresh requests

The HBM2 controller handles HBM2 memory refresh requirements and issues refresh requests at the optimal time. The controller automatically controls refresh rates based on the temperature setting of the memory through the TEMP vector that the memory provides. You can select the HBM2 controller refresh policy, based on the frequency of refresh requests. You can choose to issue refresh commands directly, through the sideband APB interface.

Precharge policy

The HBM2 controller issues precharge commands to the HBM2 memory based on the write/read transaction address. In addition, you can issue an auto-precharge command together with a write and read command, through the AXI write address port and AXI read address port.

There are two auto-precharge modes:
- HINT – You can issue the auto-precharge request. The controller then decides when to issue the precharge command.
- FORCED – You provide auto-precharge requests through the AXI interface and the precharge request executes.
Power down enable

To conserve power, the HBM2 controller can enter power-down mode when the bus is idle for a long time. You can select this option if required.

ECC

The HBM2 controller supports ECC. The ECC scheme implemented is single-bit error correction with double-bit error detection, with 64-bits of data and 8-bits of ECC code (also known as the syndrome).

HBM2 Controller features enabled by default

The HBM2 controller enables the following features by default:

- DBI – The DBI option supports both write and read DBI, and optimizes SI/power consumption by restricting signal switching on the HBM2 DQ bus.
- Parity – Supports command/address parity and DQ parity.

Related Information

- Clock Signals on page 33
- Intel Stratix 10 MX HBM2 Architecture on page 7
3. Generating the High Bandwidth Memory (HBM2) Interface Intel FPGA IP

You can parameterize and generate the High Bandwidth Memory (HBM2) Interface Intel FPGA IP using the Intel Quartus® Prime Pro Edition software.

1. Before generating the HBM2 IP, you must create a new project:
   a. Launch the Intel Quartus Prime Pro Edition software.
   b. Launch the New Project Wizard by clicking File ➤ New Project Wizard.
   c. Type a name for your project in the Directory, Name, Top-Level Entity field.
   d. In the Project Type section, select Empty Project.
   e. In the Add Files section, click Next.
   f. In the Family, Device, and Board Settings section, select Stratix 10 MX as the device family.
   g. Under Available Devices, select any MX device and your desired speed grade.
   h. Click Next and follow the Wizard's prompts to finish creating the project.

2. In the IP Catalog, open Library ➤ Memory Interfaces and Controllers.

3. Select High Bandwidth Memory (HBM2) Interface Intel FPGA IP and launch the parameter editor.

Figure 6. Selecting High Bandwidth Memory (HBM2) Interface Intel FPGA IP in the IP Catalog
3.1. Parameterizing the High Bandwidth Memory (HBM2) Interface Intel FPGA IP

You can parameterize your HBM2 IP with the HBM2 IP parameter editor. The parameter editor comprises the following tabs, on which you set the parameters for your IP:

- General
- Controller
- Diagnostics
- Example Designs

3.2. General Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP

The General tab allows you to select the channels that you want to implement, and to select the memory and fabric core clock frequency.

Figure 7. General Tab

![General Tab](image-url)
### Table 3. Group: General / FPGA

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed Grade</td>
<td>Specifies the speed grade of the Intel Stratix 10 FPGA.</td>
</tr>
<tr>
<td>HBM2 Device Type</td>
<td>Select the HBM2 Memory Device: 4GB/4H refers to HBM2 device with a total device density of 4GB in a 4-high Stack, and 8GB8H refers to a total HBM2 device density of 8GB in an 8-high Stack.</td>
</tr>
<tr>
<td>HBM2 Location</td>
<td>Selects the location of the HBM2 interface in the Intel Stratix 10 FPGA. The FPGA offers HBM2 interfaces on the top and bottom of the FPGA core.</td>
</tr>
</tbody>
</table>

### Table 4. Group: General / HBM2 Interface

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Controller for HBM Channel 0 ––– 7</td>
<td>Allows you to select the HBM2 memory channels that you want to implement. Each HBM2 channel supports a 128-bit interface to the HBM2 device, using two 64-bit Pseudo Channels. The user interface to the HBM2 Controller uses the AXI4 protocol. Each Controller has one AXI4 interface per Pseudo Channel or 2 AXI4 interfaces per channel.</td>
</tr>
</tbody>
</table>

### Table 5. Group: General / AXI Interface

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow backpressure of AXI read data and write response channels</td>
<td>Instantiates FIFOs in soft logic to buffer read data and write response on the AXI interfaces. Enable this parameter if user logic ever deasserts the AXI rready/bready signals. You can disable this parameter to reduce latency, but only if you never use rready/bready to backpressure the interface.</td>
</tr>
<tr>
<td>Threshold temperature for AXI throttling</td>
<td>This parameter defines the temperature, in degrees Celsius, above which the HBM2 controller throttles AXI interface transactions. The temperature setting applies to all the AXI4 interfaces; however, you must enable this feature on the corresponding controller tab of each HBM2 controller. When you enable throttling, the HBM2 controller reduces the amount of traffic on the DRAM channel.</td>
</tr>
<tr>
<td>AXI throttling rate</td>
<td>If you enable AXI interface throttling based on temperature, this parameter defines the throttle ratio as a percentage (0: no throttling, 100: full throttling).</td>
</tr>
</tbody>
</table>

### Table 6. Group: General / Clocks

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| Memory clock interface                                 | Specifies the clock frequency for the HBM2 interface. The maximum supported HBM2 clock frequency depends on the FPGA device speed grade:  
• -1 Speed grade : 1 GHz  
• -2 Speed grade : 800 MHz  
• -3 Speed grade : 600 MHz  
Automatically calculates the PLL reference clock frequency for best performance. You should disable this parameter if you want to select a different PLL reference clock frequency |
<p>| Use recommended PLL reference clock frequency          | Continues...                                                                                                                                                                                                |</p>
<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLL reference clock frequency</td>
<td>Enable this parameter only if you disable Use recommended PLL reference clock frequency, and want to specify a PLL reference clock frequency. You can select a PLL reference clock frequency from a displayed list. The values in the list can change when the memory interface frequency changes or the clock rate of user logic changes. You should use the fastest possible PLL reference clock frequency to achieve best jitter performance.</td>
</tr>
<tr>
<td>Core clock frequency</td>
<td>Specifies the clock frequency at which the FPGA core logic of the AXI4 interface operates. This clock must come from an I/O PLL clock output that is external to this IP core. The maximum supported core frequency depends on the device speed grade and timing closure of this clock within the FPGA. The minimum frequency of the core clock is one-fourth the HBM2 interface frequency.</td>
</tr>
<tr>
<td>Use recommended example design core clock PLL reference clock frequency</td>
<td>Automatically calculates the example design core clock PLL reference clock frequency for best performance. Disable this parameter if you want to select a different reference clock frequency.</td>
</tr>
<tr>
<td>Reference clock frequency for example design core clock PLL</td>
<td>Specify the externally provided reference clock frequency for the core clock PLL.</td>
</tr>
</tbody>
</table>

**Related Information**

Clock Signals on page 33

### 3.3. Controller Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP

The parameter editor contains one Controller tab for each memory channel that you specify on the General tab. The Controller tab allows you to select the HBM2 controller options that you want to enable.
Table 7. Group: Controller/Controller 0 Configuration

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is clone of</td>
<td>Specifies a controller from which to copy all parameter values; this parameter applies when you select more than one HBM2 controller. Set this parameter if you want one controller to have the same settings as another.</td>
</tr>
<tr>
<td>Enable Re-order buffer</td>
<td>Specifies that read data be reordered to match the order in which transactions are issued. If you disable this feature, the controller may read data in a different order than it is written; you can then maintain the order, based on the AXI read ID of the transaction. This parameter applies to cases with multiple AXI transaction IDs. By using different AXI read/write IDs, you allow the HBM2 controller to reorder transactions for better efficiency. If you use the same AXI ID for all transactions, the controller issues the commands to memory in the order in which they arrive; in this instance, you need not enable the reorder buffer.</td>
</tr>
<tr>
<td>Enable AXI interface throttling based on temperature and cattrip</td>
<td>Enables AXI thermal throttling for the specific HBM2 controller. The parameter that sets the trigger temperature for thermal throttling resides on the General tab. When the temperature of the HBM2 stack reaches the threshold temperature for HBM2 throttling as set on the General tab, the HBM2 controller begins to throttle user requests.</td>
</tr>
<tr>
<td>Address reordering</td>
<td>Specifies the pattern for mapping from the AXI interface to the HBM2 memory device. By choosing the right address reordering configuration, you help to improve the efficiency of data transfers.</td>
</tr>
</tbody>
</table>
of accesses to the HBM2 memory device, based on user traffic pattern. The HBMC supports three types of address reordering:

- **Address order (32B access: pseudo-BL8 disabled):**
  
  - SID-BG-BANK-ROW-COL[5:1]
  - SID-ROW-BANK-COL[5:1]-BG
  - ROW-SID-BANK-COL[5:1]-BG
  - COL[6] = 0

- **Address order (64B access (pseudo-BL8 enabled):**
  
  - SID-BG-BANK-ROW-COL[5:1]
  - SID-ROW-BANK-COL[5:2]-BG-COL[1]
  - ROW-SID-BANK-COL[5:2]-BG-COL[1]
  - COL[1:0] = {00}

SID applies only to the 8GB/8H HBM2 devices and is not available for 4GB/4H devices.

### User Read Auto-Precharge Policy

You can issue the request to precharge together with the read command, through the `axi_x_y_aruser` input, where `x` denotes the HBM2 channel number (0-7) and `y` denotes the HBM2 Pseudo Channel number (0/1).

You can choose between two values for this parameter:
- **RDAP_FORCED** mode, in which the HBM2 controller implements a user-requested auto-precharge command.
- **RDAP_HINT** mode, in which the controller determines when to issue an auto-precharge command, based on user-issued auto-precharge input and the address specified.

### User Write Auto-Precharge Policy

You can issue a precharge request together with the write command, through the `axi_x_y_awuser` input, where `x` denotes the HBM2 channel number (0-7) and `y` denotes the HBM2 Pseudo Channel number (0/1).

You can choose between two values for this parameter:
- **WRAP_FORCED** mode, in which the HBM2 controller implements a user-requested auto-precharge command.
- **WRAP_HINT** mode, in which the controller determines when to issue an auto-precharge command, based on user-issued auto-precharge input and the address specified.

### Power Down Enable

Allows the controller to power down when there are no commands in the queue for a long period of time.

### Refresh mode

Specifies how the HBM2 controller receives refresh requests:
- The default value is **Controller refresh all**, which allows the controller to decide when to issue refresh requests.
- Alternatively, you can issue refresh requests through the APB sideband interface, to all or specific banks.

### Refresh policy

Specifies how the controller issues refresh commands, when you set **Refresh mode to Controller refresh all**:
- The default **Flexible** setting allows the controller to determine when to issue refresh requests.
- The **Pre-pay** setting allows the controller to issue refresh commands earlier when the controller is idle.
- The **Post-pay** setting allows the controller to postpone refresh commands until there are no pending requests, or when it is time to issue a refresh command. Select this setting in bandwidth-sensitive applications.

---

**continued...**
### Display Name | Description
--- | ---
Enable 64B access for performance | Enable this parameter for 64 bit (burst length 8) data transfer through the Pseudo Channel between the UIB and the HBM2 device. For 32-bit (burst length 4) data transfer, disable this parameter.

Width of User Data | Specifies the AXI Write/Read Data width. The default setting is 256 bits for each HBM2 Pseudo Channel. Optionally, if you are not using the ECC or DM pins, you can specify the entire 288 bits for data.

Memory channel ECC generation and checking/correction | The HBM2 controller supports single-bit error correction and double-bit error detection. The controller does not support write data mask in ECC generation mode.

Write data mask enable | Enables the write data mask (DM) input to the HBM2 DRAM. When you use the DM pins, you cannot use ECC.

---

### 3.4. Diagnostic Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP

The **Diagnostics** tab allows you to select traffic options and to enable the efficiency monitor that measures HBM2 controller efficiency during functional simulation.

**Figure 9. Diagnostics Tab**

![Diagnostics Tab](image)

**Table 8. Group: Diagnostics / Traffic Generator**

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run the default traffic pattern</td>
<td>Runs the default traffic pattern after reset. The default traffic pattern consists of sequential transactions.</td>
</tr>
<tr>
<td>Enable mixed traffic</td>
<td>Configures the traffic generator to send out a variety of traffic patterns, including single and block reads/writes, using a mix of sequential and random addressing. If this parameter is not enabled, the traffic generator sends block reads/writes with sequential addressing. This parameter can help you understand the HBM2 interface performance over different traffic patterns.</td>
</tr>
</tbody>
</table>

*continued...*
### Table 9. Group: Diagnostics / Performance

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Efficiency Monitor</td>
<td>Adds an efficiency monitor component to the AXI interface of the memory controller. The efficiency monitor reports statistics about the efficiency of the interface during simulation.</td>
</tr>
<tr>
<td>Enable Efficiency Test Mode</td>
<td>Configures the traffic generator to send concurrent reads and writes. This parameter increases command efficiency, however, data mismatches may occur.</td>
</tr>
</tbody>
</table>

#### 3.5. Example Designs Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP

The **Example Designs** tab allows you to configure example design files for simulation and synthesis.
Figure 10. **Example Designs tab of HBM2 IP Parameters**

![Example Designs tab of HBM2 IP Parameters](image)

Table 10. **Group: Example Designs / Example Design Files**

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>Specifies that the system generate all necessary file sets for simulation when you click <strong>Generate Example Design</strong>. Expect an additional 1-2 minute delay when generating the simulation fileset. If you do not enable this parameter, the system does not generate simulation file sets. Instead, the output directory contains the <code>ed_sim.qsys</code> file which contains details of the simulation example design for Platform Designer, and a <code>make_sim_design.tcl</code> file with other corresponding tcl files. You can run the <code>make_sim_design.tcl</code> file from a command line to generate a simulation example design. The generated example designs for various simulators reside in the <code>/sim</code> subdirectory.</td>
</tr>
<tr>
<td>Synthesis</td>
<td>Specifies that the system generate all necessary file sets for synthesis when you click <strong>Generate Example Design</strong>. Expect an additional 1-2 minute delay when generating the synthesis file set. If you do not enable this parameter, the system does not generate synthesis file sets. Instead, the output directory contains the <code>ed_synth.qsys</code> file which contains details of the synthesis example design for Platform Designer, and a <code>make_qii_design.tcl</code> file with other corresponding tcl files. You can run the <code>make_qii_design.tcl</code> file from a command line to generate a synthesis example design. The generated example design resides in the <code>/qii</code> subdirectory.</td>
</tr>
</tbody>
</table>

**Tip:** The example design supports generation, simulation, and Intel Quartus Prime compilation flows for any selected device. To use the example design for simulation, enable the **Simulation** parameter. To use the example design for compilation and hardware, enable the **Synthesis** parameter.

Table 11. **Group: Example Designs / Generated HDL Format**

<table>
<thead>
<tr>
<th>Display Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation HDL format</td>
<td>Specifies the HDL format of the example design file set that you want to generate.</td>
</tr>
</tbody>
</table>
3.6. Generating the Example Design

After you finish parameterizing your IP, you can generate the HBM2 example design.

1. On the **Example Designs** tab, select **Simulation/Synthesis** in the **Example Design Files** group box.
2. On the **Example Designs** tab, select **Verilog** in the **Generated HDL Format** group box.
3. To generate the example design, press the **Generate Example Design** button, at the top-right of the parameter editor.
4. When prompted, specify a location to save the generated example design file set.
5. Press **OK** to begin generating the example design file set.

3.7. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Design Example

When you enable **Generate Example Design**, the HBM2 parameter editor generates the HBM2 design example, and the user IP that you can use to interface to the design.

![Generated Design Example and User Design Directories](image)

The generated design example is a complete IP package, including all the IP blocks necessary to model an HBM2 IP interface, both in simulation and in hardware. The design example includes the following blocks:

- A traffic generator that models actual user traffic, both for simulation and hardware validation.
- The PLLs required to generate the necessary core clock and UIB PHY clock for the HBM2 IP interface. (The user design does not instantiate the I/O PLL that generates the core clock.)
- The HBM2 UIB sub-system (both synthesizable IP and abstract PHY for simulation).
- The HBM2 DRAM simulation model for simulation.
The design example directory includes the following subdirectories:

- **ip** – includes the HBM2 IP settings.
- **qii** – includes all the files required to compile and synthesize the HBM2 IP and generate a bit file.
- **sim** – includes all the files required to complete the HBM2 IP simulation.

The top-level design example for synthesis is available under `<Design Directory>/hbm_0_example_design/qii/ed_synth/synth/ed_synth.v`. The `ed_synth_hbm_0_example_design` module is the top-level design module for the HBM2 IP.

### Table 12. Top-Level Signals of the HBM2 Design Example

<table>
<thead>
<tr>
<th>Signal Group</th>
<th>Signal Name</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLL Ref clk Inputs</td>
<td>core_clk_iopll_ref_clk_clk</td>
<td>Input</td>
<td>1</td>
<td>LVDS differential reference clock used by the I/O PLL to generate the fabric core clock. The design example automatically instantiates the I/O PLL that generates the core clock.</td>
</tr>
<tr>
<td></td>
<td>hbm_0_example_design_pll_ref_clk_clk</td>
<td>Input</td>
<td>1</td>
<td>LVDS differential reference clock used by the UIB PLL. The design example automatically instantiates the UIB PLL that generates the clock for the UIB subsystem.</td>
</tr>
<tr>
<td>Resets</td>
<td>core_clk_iopll_reset_r reset</td>
<td>Input</td>
<td>1</td>
<td>Reset input for the core clock I/O PLL. The reset polarity is active high. Refer to the Intel Stratix 10 device datasheet for I/O PLL specifications.</td>
</tr>
<tr>
<td></td>
<td>hbm_0_example_design_wmcrst_n_in_reset_n</td>
<td>Input</td>
<td>1</td>
<td>General core logic reset; active low.</td>
</tr>
<tr>
<td></td>
<td>hbm_only_reset_in_reset</td>
<td>Input</td>
<td>1</td>
<td>HBM-only reset; active high. Not currently supported; you can connect this to LOW.</td>
</tr>
<tr>
<td>Boundary Scan Signals</td>
<td>m2u_bridge_cattrip</td>
<td>Input</td>
<td>1</td>
<td>HBM2 boundary signals that are not driven by the traffic generator. These signals must be exposed at the design example top level to enable successful compilation. These signals should not be actively driven. You can let the Intel</td>
</tr>
</tbody>
</table>
### 3.8. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Pin Planning

The High Bandwidth Memory (HBM2) Interface Intel FPGA IP requires the following clock inputs:

- **UIB PLL reference clock** – Reference clock input for the UIB PLL. There is one UIB PLL reference clock per HBM2 interface.
- **Core clock input** – Fabric core clock, generated through an I/O PLL.

#### Table 13. Placement Requirements for PLL Reference Input Clocks

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
<th>Pin Placement Guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>hbm_0_example_design_pll_ref_clk_clk</td>
<td>LVDS differential input clock used by the hardened UIB-HBM2 subsystem.</td>
<td>Place this reference clock input on the UIB_PLL_REF_CLK_00 pins while using the HBM2 device on the bottom of the FPGA, or the UIB_PLL_REF_CLK_01 pins while using the HBM2 on the top of the FPGA.</td>
</tr>
<tr>
<td>core_clk_iopll_ref_clk_clk</td>
<td>LVDS differential input clock used to generate the fabric core clock. Instantiate the I/O PLL that will generate the core clock for the HBM2 IP.</td>
<td>Place the reference clock input on CLK_ pins to access the I/O PLL. You should select pins that are close to the I/O PLL REF_CLK input. The I/O PLL must be instantiated in the user design flow. The output of the I/O PLL serves as the EXT_CORE_CLK.</td>
</tr>
</tbody>
</table>
Jitter Specifications for the Input Reference Clocks

Both the reference clock inputs should meet and not exceed the following jitter requirements: 20ps peak-to-peak, or 1.42ps RMS at 1e-12 BER, 1.22ps at 1e-16 BER.
4. Simulating the High Bandwidth Memory (HBM2) Interface Intel FPGA IP

This section describes how to simulate the generated HBM2 IP.

**Simulation Assumptions**

The parameter settings that you make on the Controller tab affect efficiency during simulation. In the default configuration, with the default parameter settings, the traffic generator issues sequential transactions.

**Supported Simulators**

The HBM2 IP supports the following simulators:

- ModelSim*- Intel FPGA Edition
- ModelSim SE
- Questa* Advanced Simulator
- NCSim*
- Aldec Riviera-PRO*
- Synopsys* VCS
- NCSim
- Xcelium

4.1. High Bandwidth Memory (HBM2) Interface Intel FPGA IP

**Example Design**

The following illustration shows a high-level block diagram of the HBM2 example design that provides the simulation environment for the High Bandwidth Memory (HBM2) Interface Intel FPGA IP MX HBM2 IP when generated for simulation.

---

*Other names and brands may be claimed as the property of others.
The Traffic Generator emulates a real-world application that writes to, and reads back from memory and validates the read data. You can modify the traffic generator logic to fit your traffic pattern or drive the transactions to the HBM2 memory with your own logic.

Simulation incorporates an abstract model of the hardened HBM2 controller and the universal interface block (UIB). The HBM2 controller performs data reordering and enhancement functions, and allows communication between the AXI4 user interface and the UIB PHY. The universal interface block PHY (UIB PHY) is the physical-layer interface that carries low-level signaling.

The HBM2 Model is an abstract generic model representative of the HBM2 DRAM for simulation. This is not a vendor-specific model.

4.2. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with ModelSim* and Questa*

1. Launch the ModelSim simulator.
2. Select File ➤ Change Directory and navigate to: project_directory/sim/ed_sim/sim/mentor
3. Verify that the Transcript window is visible; if it is not, display it by selecting View ➤ Transcript.
4. In the Transcript window, run source msim_setup.tcl at the bottom of the ModelSim tool screen.
5. After the Tcl script finishes running, run ld_debug in the Transcript window. This command compiles the design files and elaborates the top-level design.
6. After ld_debug finishes running, the Objects window appears. In the Objects window, select the signals to simulate by right-clicking and selecting Add Wave from the context menu. For example, if you want to see the HBM2 interface signals, select the module mem0_0 from the Instance window. With mem0_0 selected, go to the Objects window and select the signals that you want to see. (If the Objects window is not visible, you can display it by selecting View ➤ Objects.
7. To run the HBM2 simulation, type run -all.
If the simulation is not visible, select **View ➤ Wave**. With the **Wave** window open, select **File ➤ Save Format**. Click **OK** to capture your selected waveforms in a wave.do file. To display the waveforms, type `do wave.do`, and then type `run -all`.

Whenever you make changes to the design or to the wave.do, you must repeat step 7 of this procedure. Alternatively, you can combine the instructions into a script and run that script instead. The following example illustrates a run.do script containing the necessary commands:

```bash
if {[file exists msim_setup.tcl]} {
    source msim_setup.tcl
    ld_debug
    do wave.do
    run -all
} else {
    error "The msim_setup.tcl script does not exist. Please generate the example design RTL and simulation scripts. See ../../README.txt for help."
}
```

Save the run.do script in the same directory as the msim_setup.tcl file. Type `do run.do` to run this script from the Transcript window.

8. Upon completion of the simulation, the **Transcript** window displays efficiency data and other useful information.

### 4.3. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with Synopsys VCS*

1. Navigate to the `project_directory/hbm_0_example_design/sim/ed_sim/sim/synopsys/vcs` directory.

2. To run the simulation, type `sh vcs_setup.sh`. To view the simulation results, write the output to a log file. The simulation log provides efficiency data and other useful information.

3. To view the waveform, add `+vcs+dumpvars+test.vcd` to the `vcs` command.

4. To view the waveform, type `dve &` to launch the waveform viewer. Add the necessary signals or module to the waveform view to view the required signals.

### 4.4. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with Riviera-PRO*

1. Navigate to: `project_directory>/sim/ed_sim/aldec`.

2. Type `rungui` to launch the Riviera-PRO simulator.

3. Type `source rivierapro_setup.tcl`.

4. Type `ld_debug` to compile the design files and elaborate the top-level design.

5. Type `run -all` to run the HBM2 simulation.

### 4.5. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with Cadence NCSim*

1. Navigate to: `project_directory>/sim/ed_sim/aldec`.

2. Type `rungui` to launch the Riviera-PRO simulator.

3. Type `source rivierapro_setup.tcl`.

4. Type `ld_debug` to compile the design files and elaborate the top-level design.

5. Type `run -all` to run the HBM2 simulation.
1. Navigate to: `project_directory>/sim/ed_sim/cadence`

2. Type `sh ncsim_setup.sh` to launch the NCSim simulator.

3. To view the simulation results, write the output to a log file. The simulation log provides efficiency data and other useful information.

### 4.6. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP with Cadence Xcelium*

1. Navigate to: `project_directory>/sim/ed_sim/xcelium`

2. Type `sh xcelium_setup.sh` to launch the Xcelium simulator.

3. To view the simulation results, write the output to a log file. The simulation log provides efficiency data and other useful information.

### 4.7. Simulating High Bandwidth Memory (HBM2) Interface Intel FPGA IP for High Efficiency

The default traffic pattern can achieve high efficiency by efficiently utilizing the HBM2 memory bandwidth and providing an efficient flow of traffic between the HBM2 controller and AXI user interface.

The main steps to deriving higher efficiency are:

- **Turn off** Enable Reorder Buffer on the Controller tab. The Reorder Buffer rearranges the read data in the order of the issued requests.

- **Turn on** Force traffic generator to issue different AXI Read/Write IDs and Enable Efficiency Test Mode on the Diagnostics tab. In this configuration, the traffic generator issues concurrent read and write transactions; consequently, you may receive data mismatch warnings, which you can ignore.

The following sections explain the General, Controller, and Diagnostic tab parameters required to perform high efficiency HBM2 simulation. The following figures illustrate parameter settings for a high-efficiency simulation for a single-channel HBM2 controller.

For more information on improving controller efficiency, refer to High Bandwidth Memory (HBM2) Interface Intel FPGA IP Controller Performance.
Figure 13. **Controller Tab Settings for High Efficiency Simulation**

![Controller Tab Settings](image1.png)

Figure 14. **Diagnostics Tab Settings for High Efficiency Simulation**

![Diagnostics Tab Settings](image2.png)
5. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Interface

This chapter provides an overview of the signals that interface to the HBM2 IP.

5.1. High Bandwidth Memory (HBM2) Interface Intel FPGA IP High Level Block Diagram

The following figure shows a high-level block diagram of the High Bandwidth Memory (HBM2) Interface Intel FPGA IP per Pseudo Channel. The IP communicates with user logic through the AXI protocol.

Figure 15. High Level Block Diagram of HBM2 Implementation

5.2. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Controller Interface Signals

This section lists the signals that connect core logic to the High Bandwidth Memory (HBM2) Interface Intel FPGA IP.

5.2.1. Clock Signals
Table 16. Clock Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ext_core_clk</td>
<td>Input</td>
<td>Core clock. Output of user-instantiated I/O PLL. User design does not instantiate the I/O PLL required to generate the ext_core_clk.</td>
</tr>
<tr>
<td>Ext_core_clkLocked</td>
<td>Input</td>
<td>LOCKED status of core clk I/O PLL, indicating the ext_core_clk is stable.</td>
</tr>
<tr>
<td>PLL_ref_clk</td>
<td>Input</td>
<td>LVDS reference clock input for UIB PLL.</td>
</tr>
<tr>
<td>Wmc_clk_0_clk</td>
<td>Output</td>
<td>Core clock output feedback from UIB. Leave unconnected.</td>
</tr>
<tr>
<td>Phy_clk_0_clk</td>
<td>Output</td>
<td>UIB PHY clk output. Not supported. Leave unconnected.</td>
</tr>
</tbody>
</table>

Related Information
- Intel Stratix 10 MX HBM2 Controller Features on page 6
- Intel Stratix 10 MX HBM2 Controller Details on page 11
- General Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP on page 16
- High Bandwidth Memory (HBM2) Interface Intel FPGA IP Design Example on page 24

5.2.2. Reset Signals

Table 18. Reset Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wmcrst_n_in</td>
<td>Input</td>
<td>Core logic external input reset, active LOW.</td>
</tr>
<tr>
<td>hbm_only_reset_in</td>
<td>Input</td>
<td>Reset for HBM2 Controller and UIBSS. Not currently supported.</td>
</tr>
<tr>
<td>wmcrst_n_x_reset_n</td>
<td>Output</td>
<td>Core input reset synchronized to the AXI interface clock domain. One per AXI Channel (represented by x).</td>
</tr>
</tbody>
</table>

Related Information
High Bandwidth Memory (HBM2) Interface Intel FPGA IP Design Example on page 24

5.2.3. Calibration Status Signals

Table 19. Calibration Status Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_cal_success</td>
<td>Output</td>
<td>Indicates calibration success.</td>
</tr>
<tr>
<td>local_cal_fail</td>
<td>Output</td>
<td>Indicates calibration failure.</td>
</tr>
<tr>
<td>cal_lat</td>
<td>Output</td>
<td>Calibrated latency output. Not supported.</td>
</tr>
</tbody>
</table>
5.2.4. Memory Interface Signals

The following HBM2 memory signals are driven by the HBM2 controller through the UIBSS. These signals are provided at the top level, for successful compilation. You do not need to drive these signals.

Table 20. **HBM2 Memory Interface Signals**

<table>
<thead>
<tr>
<th>Signal</th>
<th>Direction</th>
<th>Width</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cattrip</td>
<td>Input</td>
<td>1</td>
<td>HBM2 signals common to each HBM2 interface. You do not need to drive these</td>
</tr>
<tr>
<td>Temp</td>
<td>Input</td>
<td>3</td>
<td>HBM2 signals common to each HBM2 interface. You do not need to drive these</td>
</tr>
<tr>
<td>Wso</td>
<td>Input</td>
<td>8</td>
<td>signals and can leave them unconnected.</td>
</tr>
<tr>
<td>Reset_n</td>
<td>Output</td>
<td>1</td>
<td>HBM2 signals per HBM2 channel. You do not need to drive these signals and</td>
</tr>
<tr>
<td>Wrst_n</td>
<td>Output</td>
<td>1</td>
<td>can leave them unconnected.</td>
</tr>
<tr>
<td>Wrck</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Shiftwr</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Capturewr</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Selectwr</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Wsi</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Ck_t</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Ck_c</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Cke</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Output</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>Output</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Dq</td>
<td>Inout</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>Dm</td>
<td>Inout</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Dbi</td>
<td>Inout</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Par</td>
<td>Inout</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Derr</td>
<td>Input</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Rdqs_t</td>
<td>Input</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Rdqs_c</td>
<td>Input</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Wdqs_t</td>
<td>Output</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Wdqs_c</td>
<td>Output</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Rd</td>
<td>Inout</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Rr</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Rc</td>
<td>Output</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Aerr</td>
<td>Input</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
5.2.5. AXI User-interface Signals

The user interface to the HBM2 controller follows the Amba AXI4 protocol specification. Each AXI port serves the read and write operations for one Pseudo Channel. Each HBM2 channel consists of two Pseudo Channels, therefore each controller has two AXI ports.

**AXI Interface Signals**

Each AXI port consists of five subchannels:

- **Write Address Channel** – AXI Write Address that maps to the HBM2 DRAM Write Address.
- **Write Data Channel** – AXI Write data provided by the core logic corresponding to the Write Address.
- **Write Response Channel** – Response from the HBM2 Controller on the status of the Writes.
- **Read Address Channel** – AXI Read Address that maps to the HBM2 DRAM Read Address.
- **Read Data Channel** – AXI Read Data provided from the corresponding HBM2 DRAM Read Address.

**AXI Address Definition**

The HBM2 DRAM addressing consists of the following:

- 14-bit wide Row Address.
- 6-bit wide Column Address.
- 4-bit wide Bank Address. Bank Group corresponds to the higher order 2 bits of the Bank Address.
- 1-bit wide Stack ID (SID) available only in the 8GB configuration.

The following figure illustrates the mapping of the AXI Address bus (28-bit wide for 4GB configurations and 29-bit wide for 8GB configurations) for the various address ordering schemes to address the HBM2 DRAM.

### Figure 16. AXI Address Definition

<table>
<thead>
<tr>
<th>AXI Address</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GB, AXI4/AXI4DR (28 bits wide)</td>
<td>04</td>
<td>03</td>
<td>02</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>4GB, AXI4/AXI4DR (29 bits wide)</td>
<td>04</td>
<td>03</td>
<td>02</td>
<td>01</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td></td>
</tr>
</tbody>
</table>

*AXI ADDRESS[0] used to zero by user.*
**AXI Subchannels Descriptions**

The syntax for referencing AXI port signal names is `axi_x_y_portname` where `x` is the channel number and `y` is the Pseudo Channel number. For example, `axi_0_1_awid` refers to the write address ID of the AXI port corresponding to channel 0 and Pseudo Channel 1.

The signals in the following tables refer to the signal names corresponding to a single AXI port: Channel 0, Pseudo Channel 0.

**Table 21. User Port 0’s AXI4 Write Address (Command) Channel**

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axi_0_0_awid</td>
<td>9</td>
<td>Input</td>
<td>Write address ID. This signal is the ID tag for the write address group of signals.</td>
</tr>
<tr>
<td>axi_0_0_awaddr</td>
<td>28/29</td>
<td>Input</td>
<td>Write address. The write address gives the address of the first transfer in a write burst transaction. This address bus is 28 bits wide for a 4 GB device and 29 bits for an 8 GB HBM2 device.</td>
</tr>
</tbody>
</table>
| axi_0_0_awlen   | 8     | Input     | Burst Length. The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.  
  - 0b00000000 = Burst length 1  
  The AXI interface supports only one burst transfer at a time, based on burst length 4 or 8. |
| axi_0_0_awsize  | 3     | Input     | Burst Size. This signal indicates the size of each transfer in the burst.  
  - 0b101 = 32 Bytes  
  - 0b110 = 64 Bytes  
  The 32B and 64B access refers to data corresponding to 64 bits (one Pseudo Channel) for 4 burst cycles (32B) or 8 burst cycles (64B). The 64B access granularity is the default for better efficiency. |
| axi_0_0_awburst | 2     | Input     | Burst Type. The burst type and the size information, determine how the address for each transfer within the burst is calculated. This signal is not supported as multiple bursts are not supported and only 1 burst is supported at a time. |
| axi_0_0_awprot  | 3     | Input     | Protection Type. [Reserved for Future Use]  
  This signal indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access.  
  - 3'b000 = No protection |
| axi_0_0_awqos   | 4     | Input     | Quality of Service. The Quality of Service identifier sent for each write transaction.  
  - 4'b1111 = High priority  
  - 4'b0000 = Normal priority |
| axi_0_0_awuser  | 1     | Input     | User Signal for auto-precharge.  
  - 1'b0 = No auto-precharge  
  - 1'b1 = Auto-precharge |
<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axi_0_0_awvalid</td>
<td>1</td>
<td>Input</td>
<td>Write Address Valid. Indicates that the channel is signaling valid write address and control information.</td>
</tr>
<tr>
<td>axi_0_0_awready</td>
<td>1</td>
<td>Output</td>
<td>Write Address Ready. Indicates that the slave is ready to accept an address and associated control signals.</td>
</tr>
</tbody>
</table>

Table 22. User Port 0’s AXI4 Write Data Channel

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axi_0_0_wdata</td>
<td>256</td>
<td>Input</td>
<td>Write Data.</td>
</tr>
<tr>
<td>axi_0_0_wstrb</td>
<td>16</td>
<td>Input</td>
<td>Write Strobes (Byte Enables). Indicates which byte lanes (for u0_wdata) hold valid data. There is one write strobe bit for every eight bits of write data.</td>
</tr>
<tr>
<td>axi_0_0_wuser_data</td>
<td>16</td>
<td>Input</td>
<td>Extra Write Data (AXI WUSER port). Carries additional data going to CB bits on HBM2 interface.</td>
</tr>
<tr>
<td>axi_0_0_wuser_strb</td>
<td>2</td>
<td>Input</td>
<td>Extra Write Strobes (AXI WUSER port). Indicates which byte lanes (for u0_wuser_data) hold valid data, signal is aligned to u0_wstrb.</td>
</tr>
<tr>
<td>axi_0_0_wlast</td>
<td>1</td>
<td>Input</td>
<td>Write Last. Indicates the last transfer in a write burst.</td>
</tr>
<tr>
<td>axi_0_0_wvalid</td>
<td>1</td>
<td>Input</td>
<td>Write Valid. Indicates that valid write data and strobes are available</td>
</tr>
<tr>
<td>axi_0_0_wready</td>
<td>1</td>
<td>Output</td>
<td>Write Ready. Indicates that the slave (HBM2 controller) can accept write data.</td>
</tr>
</tbody>
</table>

Table 23. User Port 0’s Write Response Channel

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axi_0_0_bid</td>
<td>9</td>
<td>Output</td>
<td>Response ID Tag. The ID tag of the write response.</td>
</tr>
<tr>
<td>axi_0_0_bresp</td>
<td>2</td>
<td>Output</td>
<td>Write response. Indicates the status of the write transaction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2'b00 = OKAY; indicates that normal access is successful.</td>
</tr>
<tr>
<td>axi_0_0_bvalid</td>
<td>1</td>
<td>Output</td>
<td>Write response valid. Indicates that the channel is signaling a valid write response.</td>
</tr>
<tr>
<td>axi_0_0_bready</td>
<td>1</td>
<td>Input</td>
<td>Response ready. Indicates that the master can accept a write response.</td>
</tr>
</tbody>
</table>
Table 24. User Port 0’s AXI4 Read Address (Command) Channel

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axi_0_0_arid</td>
<td>9</td>
<td>Input</td>
<td>Read address ID. The ID tag for the read address group of signals.</td>
</tr>
<tr>
<td>axi_0_0_araddr</td>
<td>28/29</td>
<td>Input</td>
<td>Read address. The address of the first transfer in a read burst transaction. This address bus is 28-bits wide for a 4 GB device and 29-bits wide for an 8 GB device.</td>
</tr>
<tr>
<td>axi_0_0_arlen</td>
<td>8</td>
<td>Input</td>
<td>Burst Length. The burst length gives the exact number of transfers in a burst. The HBMC supports only one BL4 or BL8 transaction. • 0b00000000 = Burst length of 1.</td>
</tr>
<tr>
<td>axi_0_0_arsize</td>
<td>3</td>
<td>Input</td>
<td>Burst Size. This signal indicates the size of each transfer in the burst.    • 0b101 = 32 Bytes • 0b110 = 64 Bytes</td>
</tr>
<tr>
<td>axi_0_0_arburst</td>
<td>2</td>
<td>Input</td>
<td>Burst Type. The burst type and the size information determine how the address for each transfer within the burst is calculated. The HBMC does not support more than one burst at a time.</td>
</tr>
<tr>
<td>axi_0_0_arprot</td>
<td>3</td>
<td>Input</td>
<td>Protection Type. [Reserved for Future Use] Indicates the privilege and security level of the transaction, and whether the transaction is a data access or an instruction access. • 3’b000 = No protection</td>
</tr>
<tr>
<td>axi_0_0_arqos</td>
<td>4</td>
<td>Input</td>
<td>Quality of Service. The Quality of Service identifier sent for each write transaction. • 4’b1111 = High priority • 4’b0000 = Normal priority</td>
</tr>
<tr>
<td>axi_0_0_aruser</td>
<td>1</td>
<td>Input</td>
<td>User Signal for auto-precharge. • 1’b0 = No auto-precharge • 1’b1 = Auto-precharge</td>
</tr>
<tr>
<td>axi_0_0_arvalid</td>
<td>1</td>
<td>Input</td>
<td>Read address valid. Indicates that the channel signals valid read address and control information.</td>
</tr>
<tr>
<td>axi_0_0_arready</td>
<td>1</td>
<td>Output</td>
<td>Read address ready. Indicates that the slave is ready to accept an address and associated control signals.</td>
</tr>
</tbody>
</table>

Table 25. User Port 0’s Read Data Channel

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axi_0_0_rid</td>
<td>ARID_WIDTH</td>
<td>Output</td>
<td>Read ID tag. The ID tag for the read data group of signals generated by the slave.</td>
</tr>
<tr>
<td>axi_0_0_rdata</td>
<td>256</td>
<td>Output</td>
<td>Read data.</td>
</tr>
<tr>
<td>axi_0_0_ruser_data</td>
<td>16</td>
<td>Output</td>
<td>Extra Read Data (AXI RUSER port). Carries additional data coming from CB bits on HBM2 interface.</td>
</tr>
<tr>
<td>axi_0_0_ruser_err_dbe</td>
<td>1</td>
<td>Output</td>
<td>Double-Bit-Error (AXI RUSER port). Carries DBE information, aligned to u0_rvalid. 1’b1 indicates error.</td>
</tr>
<tr>
<td>Port Name</td>
<td>Width</td>
<td>Direction</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>axi_0_0_rresp</td>
<td>2</td>
<td>Output</td>
<td>Read response. Indicates the status of the read transfer:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 2'b00 = OKAY</td>
</tr>
<tr>
<td>axi_0_0_riast</td>
<td>1</td>
<td>Output</td>
<td>Read last. Indicates the last transfer in a read burst.</td>
</tr>
<tr>
<td>axi_0_0_rvalid</td>
<td>1</td>
<td>Output</td>
<td>Read valid. Indicates that the channel is signaling the required read data.</td>
</tr>
<tr>
<td>axi_0_0_ready</td>
<td>1</td>
<td>Input</td>
<td>Read ready. Indicates that the master can accept the read data and response</td>
</tr>
</tbody>
</table>

**Related Information**

High Bandwidth Memory (HBM2) Interface Intel FPGA IP Design Example on page 24

### 5.2.6. Sideband APB Interface

The sideband APB interface allows user logic to issue refresh commands and also provides access to the controller status signals. Each HBM2 channel has one APB Interface.

**Table 26. APB Interface Signals**

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ur_paddr</td>
<td>16</td>
<td>Input</td>
<td>APB address bus. You can use the APB address bus to access the MMR register space, to issue specific user-requested commands.</td>
</tr>
<tr>
<td>ur_psel</td>
<td>1</td>
<td>Input</td>
<td>Select. The user interface generates this signal to indicate that the channel APB interface is selected and that a data transfer is required. There is a PSEL signal for each HBM2 channel APB interface and this signal can be tied HIGH.</td>
</tr>
<tr>
<td>ur_penable</td>
<td>1</td>
<td>Input</td>
<td>Enable. This signal indicates the start of an APB transfer.</td>
</tr>
<tr>
<td>ur_pwrite</td>
<td>1</td>
<td>Input</td>
<td>Write/Read access. When this signal is HIGH, it indicates an APB write access. When this signal is LOW, it indicates an APB read access.</td>
</tr>
<tr>
<td>ur_pwdata</td>
<td>16</td>
<td>Input</td>
<td>Write data. This signal is driven by user logic during write cycles when PWRITE is HIGH.</td>
</tr>
<tr>
<td>ur_pstrb</td>
<td>2</td>
<td>Input</td>
<td>Write strobes (byte enables). This signal indicates which byte lanes to update during a write transfer. There is one write...</td>
</tr>
<tr>
<td>Port Name</td>
<td>Width</td>
<td>Direction</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>strobe for each eight bits of the write data bus. Write transfers to the HBM2 controller require both the byte enables to be active and hence must be driven to 2'b11. Write strobes must not be active during a read transfer.</td>
</tr>
<tr>
<td>ur_prready</td>
<td>1</td>
<td>Output</td>
<td>Ready. This signal indicates the completion of a write or read transaction.</td>
</tr>
<tr>
<td>ur_prdata</td>
<td>16</td>
<td>Output</td>
<td>Read data. The read data bus provides useful HBM2 controller information and status signals.</td>
</tr>
</tbody>
</table>

5.3. User AXI Interface Timing

This section explains the interface timing details between user logic and the HBM2 controller. User interface signals follow the AXI4 protocol specification while passing data to and from the HBM2 controller.

The AXI interface consists of the following channels:

- Write Address channel – Master (user logic) provides relevant signals to issue a write command to the slave (HBM2 controller).
- Write data channel – Master provides the write data signals corresponding to the write address.
- Write response channel – Slave provides response on the status of the issued write command to the master.
- Read Address channel – Master provides relevant signals to issue a read command to the slave.
- Read data channel – Slave provides read data and read data valid signals corresponding to the read command to the master.

All transactions in the five channels use a handshake mechanism for the master and slave to communicate and transfer information.

**Handshake Protocol**

All five transaction channels use the same VALID/READY handshake process to transfer address, data, and control information. Both the master and slave can control the rate at which information moves between master and slave. The source generates the VALID signal to indicate availability of the address, data, or control information. The destination generates the READY signal to indicate that it can accept the information. Transfer occurs only when both the VALID and READY signals are HIGH.
Figure 17. AXI Protocol Handshake Process

In the figure above, the source presents the address, data or control information after T1 and asserts the VALID signal. The destination asserts the READY signal after T2, and the source must keep its information stable until the transfer occurs at T3, when this assertion occurs. In this example, the source asserts the VALID signal prior to the destination asserting the READY signal. Once the source asserts the VALID signal, it must remain asserted until the handshake occurs, at a rising clock edge at which VALID and READY are both high. Once the destination asserts READY, it can deassert READY before the source asserts VALID. The destination can assert READY whenever it is ready to accept data, or in response to a VALID assertion from the source.

AXI IDs

In an AXI system with multiple masters, the AXI IDs used for the ordering model include the infrastructure IDs that identify each master uniquely. The ordering model applies independently to each master in the system.

The AXI ordering model also requires that all transactions with the same ID in the same direction must provide their responses in the order in which they are issued. Because the read and write address channels are independent, if an ordering relationship is required between two transactions with the same ID that are in different directions, then a master must wait to receive a response to the first transaction before issuing the second transaction. If a master issues a transaction in one direction before it has received a response to an earlier transaction in the opposite direction, there is no ordering guarantee between the two transactions.

AXI Ordering

The AXI system imposes no ordering restrictions between read and write transactions. Read and write can complete in any order, even if the read address AXI ID (ARID) of a read transaction is the same as the write address AXI ID (AWID) of a write transaction. If a master requires a given relationship between a read transaction and a write transaction, it must ensure that the earlier transaction is completed before it issues a subsequent transaction. A master can consider the earlier transaction complete only when one of the following is true:

- For a read transaction, it receives the last of the read data.
- For a write transaction, it receives the write response.
5.3.1. AXI Write Transaction

AXI Write Address

You can initiate an AXI write transaction by issuing a valid Write Address signal on the AXI Write Address Bus. Your user logic should provide the valid write address in the AWADDR bus and assert the AWVALID to indicate that the address is valid. The master can assert the AWVALID signal only when it drives valid address and control information.

When the HBM2 controller is ready to accept a Write command transaction, it asserts the AWREADY signal. Address transfer happens when both AWVALID and AWREADY are asserted.

AXI Write Data

During a write burst, the master can assert the WVALID signal only when it drives valid write data. Once asserted, WVALID must remain asserted until the rising clock edge after the slave asserts WREADY. The master must assert the WLAST signal while it is driving the final write transfer in the burst. User logic must issue the write data in the same order in which the write addresses are issued.

The following diagram illustrates a BL8 Write transaction. The master asserts the Write address (WA0) in T1 using transaction ID AWID0, the HBM2 controller asserts the AWREADY in T2 when the Write command is accepted. The master asserts the Write data in clock cycle T3. Because the controller WREADY is already asserted, the write data is accepted starting cycle T3. The last piece of the burst 8 transaction is asserted in clock cycle T6.

Figure 18. AXI Write Transaction
Write Response Channel

The HBM2 controller uses the Write Response channel to respond on successful Write transactions. The slave can assert the BVALID signal only when it drives a valid write response. When asserted, BVALID must remain asserted until the rising clock edge after the master asserts BREADY. The default state of BREADY can be high, but only if the master can always accept a write response in a single cycle.

5.3.2. AXI Read Transaction

Read Address

The user logic asserts the ARVALID signal only when it drives valid Read address information. Once asserted, ARVALID must remain asserted until the rising clock edge after the HBM2 controller asserts the ARREADY signal. If ARREADY is high, the HBM2 controller accepts a valid address that is presented to it. Once calibration is completed and the HBM2 Controller is ready to accept commands, the ARREADY is asserted high.

Read Data Channel

The HBM2 controller asserts the RVALID signal when it drives valid read data to the user logic. The master interface uses the RREADY signal to indicate that it accepts the data. The state of RREADY can be always held high, if the master is always able to accept read data from the HBM2 Controller. The soft logic first in, first out (FIFO) buffers can be instantiated through the HBM2 parameter editor if the HBM2 controller expects to ever deassert the RREADY signal. The HBM2 controller asserts the RLAST signal when it is driving the final read transfer in the burst.

Figure below describes a BL8 Read transaction. The user logic asserts the Read address (RA0) in T3 using transaction ID ARID0, the HBM2 controller ARREADY is already asserted, the READ command is accepted. The controller provides the Read Data back to the user interface after issuing the READ command to the HBM2 DRAM. The HBM2 controller asserts the Read data in clock cycle TB. The Read transaction ID (RID) provided by the HBM2 controller corresponds to the Read Address ID (ARID). The last piece of the burst 8 transaction (RLAST) is asserted in clock cycle TE.
5.4. User APB Interface Timing

The Advanced Peripheral Bus (APB) interface lets you issue user-controlled refresh signals and access the HBM2 controller’s Control and Status registers.

Unlike the AXI4 interface which is used for high-bandwidth, main band operations and where each HBM pseudo-channel is mapped to its own dedicated AXI4 port, the APB interface is intended for relatively low-bandwidth sideband operations. (For example, to provide an alternative means of controlling refreshes without colliding with the main traffic stream, and for issuing custom commands useful for debugging.) Another difference is that the APB bus interface commands can target one or both Pseudo Channels at the same time, as there is one APB interface per HBM2 channel or two HBM2 Pseudo Channels.

5.4.1. Advanced Peripheral Bus Protocol

The Advanced Peripheral Bus (APB) is part of the Advanced Microcontroller Bus Architecture (AMBA) protocol family. It defines a low-cost interface that is optimized for minimal power consumption and reduced interface complexity.
5.4.2. APB Interface Timing

**Write Access**

Referring to the following diagram, write transactions to the APB interface follow these steps:

1. At T1, a write transfer starts with address ADDR1, write data DATA1, write signal PWRITE and select signal PSEL registered at the rising edge of the core clock. This is the setup phase of the write transfer.

2. At T2, PENABLE is registered at the rising edge of the core clock and held HIGH until the HBM2 controller asserts PREADY HIGH. The values of PADDR, PSEL, PENABLE, PWDATA, PSTRB and PWRITE must remain unchanged while PREADY remains LOW.

3. At Tx, when PREADY goes HIGH, the write transaction will complete at the next rising edge of the core clock. This indicates the end of the Write Access Phase. PREADY stays HIGH only for one clock cycle.

4. PENABLE is deasserted at the end of the transfer. The select signal PSEL, is also deasserted unless the transfer is to be followed immediately by another transfer.

![APB Write Transaction Diagram](image)

Figure 20. **APB Write Transaction**
Read Access

Referring to the following diagram, read transactions to the APB interface follow these steps:

1. At T1, a read transfer starts with address ADDR1, PWRITE asserted LOW and select signal PSEL registered at the rising edge of the core clock. This is the setup phase of the write transfer.

2. At T2, PENABLE is registered at the rising edge of the core clock and held HIGH until the HBM2 controller asserts PREADY HIGH. The values of PADDR, PSEL, PENABLE and PWRITE must remain unchanged while PREADY remains LOW.

3. At Tx, when PREADY goes HIGH, Read Data is available on the PRDATA bus. PREADY stays HIGH only for one clock cycle.

4. PENABLE is deasserted at the end of the transfer. The select signal PSEL, is also deasserted unless the transfer is to be followed immediately by another transfer.

Figure 21. APB Read Transaction

5.5. User-controlled Accesses to the HBM2 Controller

You can use the APB interface in applications when you need to directly control the HBM2 Refresh commands and access HBM2 Controller Status registers.
Each physical HBM2 channel is mapped to its own sideband register space. The sideband register map is shared between the two HBM2 Pseudo Channels and the allocation of addresses is organized as follows:

- Registers common to both Pseudo-Channels:
  - Address map – 16’h0000-16’h00FF.
  - Includes Refresh (per-bank, all banks), Self-Refresh, Temperature Readout and Power down status.
- Register Map for individual Pseudo Channels:
  - Address map – Pseudo Channel 0 (16’h0100-16’h01FF) and Pseudo Channel 1 (0x200-0x2FF).
  - This map is used to access ECC and Interrupt Status Registers for each Pseudo Channel.

5.5.1. User-controlled Refreshes

By default, the HBM2 parameter editor settings allow the HBM2 controller to schedule Refresh commands to the HBM2 DRAM. However, user logic can also issue Refresh commands to memory.

If you want to allow user logic to issue Refresh requests, you must specify so in the Controller Settings of the parameter editor, as follows:

Change the **Controller Settings ➤ REFRESH MODE** value from **Controller Refresh All** to **User Refresh All** or **User Refresh Per-bank**.

Each APB interface corresponds to one HBM2 channel, therefore requests must be addressed to either of the two Pseudo Channels, one Pseudo Channel at a time.

The timing of the refresh interval is specific to the HBM2 DRAM timing. The required refresh rate is provided by the HBM2 DRAM through the TEMP[2:0] vector, which can be read through the APB interface.

<table>
<thead>
<tr>
<th>TEMP[2:0]</th>
<th>Refresh Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>4x tREFI</td>
</tr>
<tr>
<td>001</td>
<td>2x tREFI</td>
</tr>
<tr>
<td>011</td>
<td>1x tREFI</td>
</tr>
<tr>
<td>010</td>
<td>0.5x tREFI</td>
</tr>
<tr>
<td>110</td>
<td>0.25x tREFI</td>
</tr>
<tr>
<td>111</td>
<td>TBD</td>
</tr>
<tr>
<td>101</td>
<td>TBD</td>
</tr>
<tr>
<td>100</td>
<td>TBD</td>
</tr>
</tbody>
</table>
Refresh command options available from user logic are as follows:

- User refresh commands issued on a per bank basis. This corresponds to banks corresponding to the HBM2 Pseudo Channel.
- User refresh commands to all banks. This corresponds to all banks corresponding to the HBM2 Pseudo Channel.
- Self-refresh command.

All the user refresh requests follow the APB interface protocol. User logic issues the Refresh request and waits for the acknowledgement signal from the HBM2 controller before issuing another Refresh request.

**User Refresh Per Bank**

User refresh requests per-bank consist of the following commands issued to the APB Interface.

### Table 29. APB Write Data Definition for User Refresh Request Per Bank

<table>
<thead>
<tr>
<th>Write Data Definition for Refresh Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>Pseudo Channel Number. Set the following values to indicate the Pseudo channel number:</td>
</tr>
<tr>
<td></td>
<td>0 - Pseudo Channel 0</td>
</tr>
<tr>
<td></td>
<td>1 - Pseudo Channel 1</td>
</tr>
<tr>
<td>[6]</td>
<td>Select Per bank or all bank refresh. Set to 1'b0 for per bank refresh.</td>
</tr>
<tr>
<td>[7]</td>
<td>Reserved. Set to 1'b0.</td>
</tr>
<tr>
<td>[8]</td>
<td>User initiated refresh request. Set this bit to 1'b1 to initiate refresh command.</td>
</tr>
<tr>
<td>[12:10]</td>
<td>Number of bursts for per bank refresh bursting feature. Burst = value[1:0] + 1. The per-bank refresh bursting is a feature supported by the HBM2 controller where the user interface can request refresh commands to be issued to more than one HBM2 DRAM bank.</td>
</tr>
<tr>
<td>[15:13]</td>
<td>Increment value for per bank refresh bursting feature; rolls over once it reached the max bank value. Increment = value[2:0] + 1, where value refers to the current bank address when the write command is issued. This becomes the starting bank address. Note: The ratio of maximum increment value to total number of banks per Pseudo Channel divided by 4. (For example, a 4H device has a maximum value of 16/4=4.)</td>
</tr>
</tbody>
</table>

To issue user refresh commands on a per bank basis, follow these steps:

1. Write to address 16'h0000 with corresponding Write Data.
For example, write data of 16'b0000_0001_0000_0010 (Pseudo-channel 0; bank address is located in Bits[5:1] ; Bit[6] set to 0 for per-bank refresh; Bit [8] set to User initiated Refresh Request).


2. Read from address 16'h0000 for Refresh Request Acknowledge.

3. Wait till PRDATA[9] returns 1'b1 to indicate refresh request is accepted by controller. You can ignore the values of all the other bits of PRDATA. The following figure explains the timing of per-bank refresh request to the APB Interface.

**Figure 22. Timing of APB Refresh Request Per Bank**

![Timing of APB Refresh Request Per Bank](image)

Individual requests to different Pseudo Channels must be issued serially. You must wait for the HBM2 controller to issue the Acknowledge signal before issuing another Refresh request.

**User Refresh to All Banks**

**Table 30. APB Write Data Definition for User Refresh Request to All Banks**

<table>
<thead>
<tr>
<th>Write Data Bit Definition for Refresh Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>Pseudo Channel Number. Set a value of 0 for Pseudo Channel 0, or a value of 1 for Pseudo Channel 1.</td>
</tr>
<tr>
<td>[6]</td>
<td>Select Per bank or all bank refresh. Set to 1'b1 - all banks refresh.</td>
</tr>
<tr>
<td>[7]</td>
<td>Reserved. Set to 1'b0.</td>
</tr>
<tr>
<td>[8]</td>
<td>User initiated refresh request. Set this bit to 1'b1 to initiate refresh command.</td>
</tr>
<tr>
<td>[15:9]</td>
<td>Ignored.</td>
</tr>
</tbody>
</table>
Table 31. APB Read Data Definition for User Refresh Request

<table>
<thead>
<tr>
<th>Read Data Bit Definition for Refresh Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8:0]</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
[9'b1 – Read data of 1'b1 indicates that the HBM2 controller Acknowledge of Refresh command.  
Once a refresh request has been issued, you must wait for the Acknowledge bit to be asserted before issuing further requests. |
| [15:10]                                     | Reserved.   |

To issue user refresh commands to all banks, follow these steps:

1. Write to address 16'h0000 with corresponding Write Data.  
   - For example, Write data of 16'b0000_0001_0100_0010 (Pseudo Channel 0; bank address is located in Bits[5:1]; Bit[6] set to 1 for all-banks refresh; Bit [8] set to User initiated Refresh Request).  
2. Read to Address 16'h0000 for Refresh Request Acknowledge.  
3. Wait till PRDATA[9] returns a value of 1'b1 to indicate User Refresh request is accepted by the controller. The following figure explains the APB interface timing for refresh request to all banks.

**Figure 23. Timing of APB Refresh Request to All Banks**

**Self Refresh**

To enter a Self Refresh state, issue these commands to the APB interface:

1. Write to address 16'h0004 with Write Data(PWDATA) of 16'h0001.  
2. Read from address 16'h004. Wait until Read Data (PRDATA [1]) returns a value of 1'b1 to indicate that the HBM2 controller is in a Self Refresh state.
To exit a Self Refresh state, issue this command to the APB interface:

- Write to address 16'h0004 with Write Data(PWDATA) of 16'h0000 to exit Self-Refresh. Wait for the Self Refresh Acknowledge to be 1'b0 prior to issuing another Self Refresh request.

The write data bus is used to request entry or exit from self refresh. The Read data bus is used to read the self-refresh status of the HBM2 channel. The following tables provide the Write and Read data definitions for accessing self refresh.

**Table 32.** APB Write Data Bit Definition for Self Refresh

<table>
<thead>
<tr>
<th>Write Data Bit Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>User-initiated Self-Refresh Request. To enter Self Refresh: • 1'b1: to enter Self Refresh. • When asserted, indicates Self Refresh request. Set this bit to 1'b1 to initiate entry to Self Refresh. To exit Self Refresh: • 1'b0: to exit Self Refresh. • Clear this bit to 1'b0 to exit Self Refresh.</td>
</tr>
<tr>
<td>[15:1]</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**Table 33.** APB Read Data Bit Definition for Self Refresh

<table>
<thead>
<tr>
<th>Read Data Bit Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>Reserved.</td>
</tr>
<tr>
<td>[1]</td>
<td>Self refresh status: • 1'b1 – HBM Channel is in Self Refresh. • 1'b1 – HBM Channel is in Self Refresh.</td>
</tr>
</tbody>
</table>

### 5.5.2. Temperature and Calibration Status Readout

User logic can read the HBM2 TEMP[2:0] signal output value and the CATTRIP signal value from APB address 16'h000C.

**Table 34.** APB Read Data Temperature and Calibration Status Information

<table>
<thead>
<tr>
<th>Read Data Bit Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3]</td>
<td>CATTRIP value from HBM2 device.</td>
</tr>
<tr>
<td>[8]</td>
<td>• 1 = calibration in progress. • 0 = calibration is done.</td>
</tr>
<tr>
<td>[9]</td>
<td>• 1'b1: Calibration/training successful. • 1'b0: Calibration/training failed.</td>
</tr>
<tr>
<td>[15:10]</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**Related Information**

High Bandwidth Memory (HBM2) Interface Intel FPGA IP DRAM Temperature Readout on page 62
5.5.3. Power Down Status

To invoke the Power Down mode in the HBM2 controller, enable the Power Down Enable Mode option in the parameter editor when generating the HBM2 IP.

To read the Power Down Enable status, issue a Read command to APB Address 16’h0008. The following table shows the HBM2 Power Down status information provided by APB Read data.

Table 35. APB Power Down Status Information

<table>
<thead>
<tr>
<th>Read Data Bit Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>Power Down status.</td>
</tr>
<tr>
<td></td>
<td>• 1'b1 – HBM Channel is in Power Down mode.</td>
</tr>
<tr>
<td></td>
<td>• 1'b0 – HBM Channel is not in Power Down mode.</td>
</tr>
<tr>
<td>[15:1]</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

5.5.4. ECC Error Status

You can read the status of various registers used by the ECC feature, using the APB interface.

ECC registers provide the following information – and corresponding APB addresses – for Pseudo Channel 0 and Pseudo Channel 1:

- Single-bit error (SBE) counter
- Double-bit error (DBE) counter
- Logical address (AXI address) of the first single-bit error
- Logical address (AXI address) of the first double-bit error
- Read transaction ID (AXI) of the first single-bit error
- Read transaction ID (AXI) of the first double-bit error
- Read Data Parity error counter

The values of the ECC registers are read when the PRREADY output from the HBM2 controller is asserted. When the HBM2 controller is RESET, these register values are reset to zero.

Table 36. ECC Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Read Data Bit Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16’h0120 – PC0 16’h0220 – PC1</td>
<td>[7:0]</td>
<td>Single-Bit-Error Counter. The counter does not overflow when value reaches maximum. Write 0 to clear the counter. Clearing the counter does not clear this valid bit or vice-versa. You must issue a separate write command to valid and counter to clear both registers.</td>
</tr>
<tr>
<td></td>
<td>[15:8]</td>
<td>Reserved.</td>
</tr>
<tr>
<td>16’h0122 – PC0 16’h0222 – PC1</td>
<td>[7:0]</td>
<td>Double-Bit-Error Counter. The counter does not overflow when value reaches maximum.</td>
</tr>
</tbody>
</table>

continued...
### Address | Read Data Bit Location | Description
--- | --- | ---
 |  | Write 0 to clear the counter. Clearing the counter does not clear this valid bit or vice-versa. You must issue a separate write command to valid and counter to clear both registers. | 
[15:8] |  | Reserved. |  
| 16'h0124 – PC0 | 16'h0224 | [15:0] | First SBE logical address (lower order). This is the logical address of the first single-bit error, corresponding to the lower 16 bits of the AXI Address bus. |  
| 16'h0126 – PC0 | 16'h0226 – PC1 | [15:0] | First SBE logical address (higher order). This is the logical address of the first single-bit error, corresponding to the higher 16 bits of the AXI address bus. The higher order 4 bits in 4GB configuration and higher-order 3 bits in 8GB configuration are padded with zeros. |  
| 16'h0128 – PC0 | 16'h0228 – PC1 | [15:0] | First DBE logical address (lower order). This is the logical address of the first double-bit error, corresponding to the lower 16 bits of the AXI address bus. |  
| 16'h012A – PC0 | 16'h0228 – PC1 | [15:0] | First DBE logical address (higher order). This is the logical address of the first double-bit error, corresponding to the higher 16 bits of the AXI address bus. The higher order 4 bits in 4GB configuration and higher order 3 bits in 8GB configuration are padded with zeros. |  
| 16'h012C – PC0 | 16'h022C – PC1 | [8:0] | Read Transaction ID of the first single-bit error. |  
| 16'h012E – PC0 | 16'h022E – PC1 | [8:0] | Read Transaction ID of the first double-bit error. |  
| 16'h0130 – PC0 | 16'h0230 – PC1 | [7:0] | Read Data Parity Error Count. The counter does not overflow when value reaches maximum. Write 0 to clear the counter. Clearing the counter does not clear this valid bit or vice-versa. You must issue a separate write command to valid and counter to clear both registers. |  
|  | [15:8] | Reserved. |  

### 5.5.5. User Interrupt

An interrupt signal is generated when one or more of the Error Status signals are TRUE. You can configure the conditions that will help to trigger the interrupt signal.
The various status signals that you can use to generate the interrupt include the following:
- Single-bit error (SBE) or double-bit error (DBE) of AXI Read Data or internal RAM used for Write and Read data storage
- Read Data parity error (RDPE) or Write Data parity error (WDPE) of AXI Read Data
- Address Command parity error
- CATTRIP
- Calibration

### 5.5.5.1. Interrupt Enable and Conditions for Interrupt Generation

If you want interrupts to occur based on certain conditions, you must set the conditions to enable interrupt generation.

To enable interrupt generation, issue a Write command to address location 16'h0100 (for Pseudo Channel 0) and 16'h0200 (for Pseudo Channel 1), with the corresponding Write Data (PWDATA).
- PWDATA[0] – Interrupt enable.
- PWDATA[11:1] - Lists the various status signals that you can use, alone or in combination, to trigger the Interrupt signal.
  - Set the Mask value to 1'b0 to use the corresponding error condition to generate the Interrupt signal.
  - If you set the Mask value to 1'b1, the Interrupt generator will ignore that specific error condition. For example, to use the double-bit error condition to generate the Interrupt signal, set PWDATA[2] to 1'b0.

The following table describes the 16-bit Write Data (PWDATA) used to set the Interrupt Enable, and the conditions of the interrupt.

<table>
<thead>
<tr>
<th>Write Data Bit Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>Interrupt Enable: Enables interrupt to the HBM2 controller when the conditions set in PWDATA[11:1] are TRUE. 1 – Enable Interrupt 0 – Disable Interrupt</td>
</tr>
</tbody>
</table>

*continued...*
**5.5.5.2. Interrupt Status**

You can read the interrupt status from address 16’h0102 for Pseudo Channel 0 and 16’h0202 for Pseudo Channel 1. The interrupt signal is cleared when the individual error status signals that contribute to the interrupt are cleared and the corresponding error counters are cleared.

**Table 38. Read Data Definition for Interrupt Status**

<table>
<thead>
<tr>
<th>Read Data Bit Definition</th>
<th>Description</th>
</tr>
</thead>
</table>
| [0]                      | Interrupt asserted when one of the following events occurs:
|                          |   • Single-bit error
|                          |   • Double-bit error
|                          |   • Read Data parity error
|                          |   • Write Data parity error
|                          |   • Address Command parity error
|                          |   • CATTRIP assertion
|                          |   • Calibration status failure |

| [15:1]                   | Reserved. |

**5.5.5.3. Error Valid Status**

You can read individual status signals from address 16’h0104 for Pseudo Channel 0 and 16’h0204 for Pseudo Channel 1. You can also write to these addresses to clear the status signals.

The above addresses provide the actual status signal for the following error conditions:

- Single bit/Double bit error value from the HBM2 memory
- Read data/Write Data parity error value from the HBM2 memory.
- Address/Command parity value.
- CATTRIP signal.
- Calibration error signal.
- Single-bit error and double-bit error from internal RAM used for Write and Read data storage.

To clear the individual Error Valid Signals, write 1'b1 to clear the Valid signal and the corresponding Error Counter.

**Table 39. Read Data Definition for Error Valid Status**

<table>
<thead>
<tr>
<th>Read Data Bit Definition</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>DRAM Single-Bit-Error Valid. Asserted when single-bit error occurs and stays high until cleared. Write 1'b1 to clear.</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Write Data Bit Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Data Bit Definition</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>You can retrieve the number of single-bit errors from the single-bit error counter. Clearing the counter does not clear this valid bit or vice-versa. You must issue a separate write command to valid and counter to clear both registers.</td>
<td></td>
</tr>
<tr>
<td>DRAM Double-Bit-Error Valid. Asserted when double-bit error occurs and stays high until cleared. Write 1'b1 to clear. You can retrieve the number of double-bit errors from the double-bit error counter. Clearing the counter does not clear this valid bit or vice-versa. You must issue a separate write command to valid and counter to clear both registers.</td>
<td></td>
</tr>
<tr>
<td>Read Data Parity Error Valid. Asserted when a Read Data parity error occurs and stays high until cleared. Write 1'b1 to clear. You can retrieve the number of Read Data parity errors from the Read Data parity error counter. Clearing the counter does not clear this valid bit or vice-versa. You must issue a separate write command to valid and counter to clear both registers.</td>
<td></td>
</tr>
<tr>
<td>Write Data Parity Error Valid. Asserted when a Write Data parity error occurs and stays high until cleared. Write 1'b1 to clear.</td>
<td></td>
</tr>
<tr>
<td>Address Command Parity Error Valid. Asserted when an Address Command parity error occurs and stays high until cleared. Write 1'b1 to clear.</td>
<td></td>
</tr>
<tr>
<td>Cat Trip Error Valid. Asserted when a Cat Trip error occurs and stays high until cleared. Write 1'b1 to clear.</td>
<td></td>
</tr>
<tr>
<td>Calibration Error Valid. Asserted when a calibration error occurs and stays high until cleared. Write 1'b1 to clear.</td>
<td></td>
</tr>
<tr>
<td>Write SRAM Single-Bit Error Valid. Asserted when a Write SRAM Single-Bit error occurs and stays high until cleared. Write 1'b1 to clear.</td>
<td></td>
</tr>
<tr>
<td>Write SRAM Double-Bit Error Valid. Asserted when a Write SRAM Double-Bit error occurs and stays high until cleared. Write 1'b1 to clear. You can retrieve the number of Write SRAM Double-Bit errors from the Write SRAM Double-Bit error counter. Clearing the counter does not clear this valid bit or vice-versa. You must issue a separate write command to valid and counter to clear both registers.</td>
<td></td>
</tr>
<tr>
<td>Read SRAM Single-Bit Error Valid. Asserted when a Read SRAM Single-Bit error occurs and stays high until cleared. Write 1'b1 to clear.</td>
<td></td>
</tr>
<tr>
<td>Read SRAM Double-Bit Error Valid. Asserted when a Read SRAM Double-Bit error occurs and stays high until cleared. Write 1'b1 to clear. You can retrieve the number of Read SRAM Double-Bit errors from the Read SRAM Double-Bit error counter. Clearing the counter does not clear this valid bit or vice-versa. You must issue a separate write command to valid and counter to clear both registers.</td>
<td></td>
</tr>
<tr>
<td>Reserved.</td>
<td></td>
</tr>
</tbody>
</table>

### 5.5.6. ECC Error Correction and Detection

The HBM2 controller supports single-bit error correction and double-bit error detection. It does not support correction or detection of more than two error bits.
The ECC encoder and decoder blocks reside inside the UIB subsystem and help to efficiently perform the ECC logic without additional latency. The ECC logic performs the following operations:

- When a single-bit error is detected, the error is corrected and passed to the AXI Interface.
- When a double-bit error is detected, a signal is asserted to indicate the error and is passed through the axi_ruser_err_dbe signal in the AXI Read Data Channel Interface.
- Error information is available through the APB interface.

You can use the HBM2 controller's Read-Modify-Write feature to correct a single or double-bit error detected in the HBM2 DRAM. A single-bit error or a double-bit error corresponds to an error detected on every 64-bit HBM2 DQ bus; consequently, multiple errors could be detected:

- Multiple single-bit errors are treated as a single single-bit error, and the single-bit error count increases by 1.
- Multiple double-bit errors are treated as a single double-bit error, and the double-bit error count increases by 1.
- A single-bit error and a double-bit error are treated as a double-bit error, because the double-bit error has higher priority. The double-bit error count increases by 1 and the single-bit error count does not change.

**Read-Modify-Write**

The Read-Modify-Write feature reads from the HBM2 DRAM, modifies the data, and writes back to the HBM2 memory. The HBM2 controller supports the following functions as part of the Read-Modify-Write process:

- Dummy Writes – Corrects HBM2 DRAM data detected to have a single-bit error.
- Partial Writes – Issues partial writes to HBM2 DRAM where not all bytes are enabled.

**Dummy Writes**

When the ECC decoder logic detects and corrects a single-bit error on the Read data, user logic may correct the corresponding bit in the HBM2 DRAM, using the Dummy Write process. A Dummy Write issues a Read from the memory location and writes the corrected Read data back to the corresponding memory location.

To request a Dummy Write, issue a regular AXI Write transaction with all byte enables deasserted, to the corresponding address.
The HBM2 controller handles the Read-Modify-Write operation internally and corrects the DRAM data without additional user intervention. The Read-Modify-Write operation follows this process:

- The AXI Adaptor within the UIB subsystem decodes all the byte enables deasserted and identifies the Read-Modify-Write request.
- The HBM2 controller then issues a Read to the corresponding address in the HBM2 DRAM.
- The ECC Decoded Read data is used as Write Data – this is the corrected Read data if a single-bit error was detected.
- The HBM2 Controller issues an HBM2 Write transaction and later writes the decoded Read data into the memory.

**Partial Writes**

The HBM2 controller’s Partial Write capability allows the user logic to issue a partial write to the HBM2 DRAM, when not all the byte enables are asserted and only selected DRAM bytes are written to. The Partial Write feature first issues a Read from the memory location, and then merges correct Read data with the correct Write data, to be written back to the memory location. (This process is necessary because Data Mask signals are not available to user logic when ECC is enabled.) The HBM2 controller intelligently supports Partial Writes using the AXI4 interface.

To request a Partial Write, you issue a regular AXI Write transaction, with not all byte enables asserted (that is, not a full Write), and with corresponding Write data to be written to the HBM2 DRAM.

The HBM2 controller handles the issuance of the corresponding memory transactions necessary to complete the Partial Write:

- The AXI Adaptor within the UIB Subsystem decodes the byte enables and identifies the Partial Write request.
- The HBM2 controller then issues a Read to the corresponding address in the HBM2 DRAM.
- The ECC Decoded Read data is merged with the requested Write data based on the byte enables.
- The HBM2 controller issues an HBM2 Write transaction and later writes the merged data into the memory with the updated ECC code.
6. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Controller Performance

This section discusses key aspects of the HBM2 IP controller performance: efficiency, latency, and timing.

6.1. High Bandwidth Memory (HBM2) DRAM Bandwidth

For each HBM2 DRAM in an Intel Stratix 10 MX device, there are eight channels of 128-bits each. The following example illustrates the calculation of bandwidth offered by one HBM2 interface.

Assuming an interface running at 1 GHz:

\[ 128 \text{ DQ} \times 1 \text{ GHz} = 128 \text{ Gbps} \]

- The interface operates in double data-rate mode, so the total bandwidth per HBM2 is: \( 128 \text{ Gbps} \times 2 = 256 \text{ Gbps} \)
- The total bandwidth for the HBM2 interface is: \( 256 \text{ Gbps} \times 8 = 256 \text{ GBytes/sec} \)
- If the HBM2 controller operates at 90% efficiency, the effective bandwidth is: \( 256 \text{ Gbps} \times 0.9 \approx 230 \text{ GByte/sec} \)

6.2. High Bandwidth Memory (HBM2) Interface Intel FPGA IP HBM2 IP Efficiency

The efficiency of the High Bandwidth Memory (HBM2) Interface Intel FPGA IP estimates data bus utilization at the AXI interface. The AXI4 protocol supports independent write and read address and data channel and accepts concurrent write and read transactions. Calculated efficiency values take into consideration that the core clock frequency and the memory clock frequency are different. The HBM2 IP design example includes an efficiency monitor that reports the efficiency and the minimum latency observed in the transactions that are simulated. You can enable the efficiency monitor on the Diagnostics tab of the HBM2 IP parameter editor.

The following equation represents the HBM2 controller efficiency:

\[
\text{Efficiency} = \frac{(\text{Write transactions} + \text{Read transactions accepted by HBM2 controller})}{\text{total valid transaction count}} \times \frac{(\text{core clk frequency} \times 2)}{\text{HBM2 interface frequency}} \times 100
\]
• Write transactions – Refers to user write data transactions that the HBM2 controller accepts (user-asserted AXI WVALID and corresponding controller-asserted AXI WREADY).

• Read transactions – Refers to user read data transactions that the HBM2 controller has processed (controller-asserted AXI RVALID and corresponding user-asserted AXI RREADY).

• Total valid transaction count – Total transaction time after first valid transaction has been issued.

• Core frequency (MHz) – The frequency at which user logic operates. The core operates at a lower frequency than the HBM2 interface.

• HBM2 interface frequency (MHz) - The frequency at which the HBM2 interface operates.

Example: Consider a case where user logic operates at 400 MHz and the HBM2 interface operates at 800 MHz. Assume 1600 write and read transactions respectively. Considering BL8 transactions, this corresponds to a total of 3200 write and read data transactions respectively. Assuming a total transaction time of 7500 AXI cycles, the efficiency can be calculated as follows:

\[
\text{Efficiency} = \left(\frac{3200 + 3200}{7500}\right) \times \left(\frac{800}{800}\right) \times 100 = 85.33\%
\]

The HBM2 controller provides high efficiency for any given address pattern from the user interface. The controller efficiently schedules incoming commands, avoiding frequent precharge and activate commands as well as frequent bus turn-around when possible.

Factors Affecting Controller Efficiency

Several factors can affect controller efficiency. For best efficiency, you should consider these factors in your design:

• User-interface frequency vs HBM2 interface frequency - The frequency of user logic in the FPGA fabric plays an important role in determining HBM2 memory efficiency, as shown in the example above.

• Traffic Patterns - Traffic patterns play an important role in determining controller efficiency.
  — Sequential vs random DRAM addresses: Sequential addresses enable the controller to issue consecutive write requests to an open page and help to achieve high controller efficiency. Random addresses require constant PRECHARGE/ACTIVATE commands and can reduce controller efficiency. Set the User Auto Precharge Policy to FORCED and set the awuser/aruser signal on the AXI interface to HIGH to enable Auto Precharge.
  — Sequential Read only or Write Only transactions: Sequential read-only or write-only transactions see higher efficiency as they avoid bus turnaround times of the DRAM bi-directional data bus.

• Burst length - The pseudo-BL8 mode helps to ensure shorter memory access timing between successive BL4 transactions, to improve controller efficiency.

• AXI Transaction IDs - Efficient use of AXI transaction IDs helps the HBM2 controller schedule the transactions for high efficiency. Use of the same AXI transaction ID preserves command order.
6.3. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Latency

Read latency measures the number of clock cycles from the time the HBM2 controller receives a valid read address command, to the time that valid read data is available at the user interface. (In other words, from the instant the master asserts the ARVALID signal and the slave asserts the ARREADY signal, until the slave asserts the RVALID signal and the master asserts the RREADY signal.)

Read latency includes the controller command path latency to issue the read command to the HBM2 memory, memory read latency, and the delay in the read data path through the HBM2 memory controller. Simulation reports the minimum latency in AXI core clock cycles seen during the simulation time.

6.4. High Bandwidth Memory (HBM2) Interface Intel FPGA IP Timing

The maximum HBM2 memory interface frequency is based on the Intel Stratix 10 MX device speed grade. The maximum core interface frequency is limited by the frequency at which the core logic can meet timing.

For the best HBM2 efficiency, ensure that your user logic follows best design practices. Take care to avoid combinatorial paths between the AXI master and slave input and output signals. Add pipeline registers as necessary and reduce logic levels in timing-critical paths to successfully meet core timing requirements.

The following documents provide detailed information on the Intel Stratix 10 device architecture and design techniques that you can adopt to achieve the best core performance:


Related Information

- Intel Stratix 10 High-Performance Design Handbook

6.5. High Bandwidth Memory (HBM2) Interface Intel FPGA IP DRAM Temperature Readout

You can read the temperature of the HBM2 DRAM through the APB interface. Refer to the Temperature and Calibration Status Readout topic for more information.

You can also read the temperature using the Intel Stratix 10 Analog to Digital Converter. Refer to the Intel Stratix 10 Analog to Digital Converter User Guide for more information.

Related Information

- Temperature and Calibration Status Readout on page 52
- Intel Stratix 10 Analog to Digital Converter User Guide
### 7. Document Revision History for High Bandwidth Memory (HBM2) Interface Intel FPGA IP User Guide

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Intel Quartus Prime Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 2018.05.07       | 18.0                        | • Changed document title from Intel Stratix 10 MX HBM2 IP User Guide to High Bandwidth Memory (HBM2) Interface Intel FPGA IP User Guide.  
• Chapter 1, Introduction to High Bandwidth Memory:  
  — Added ECC support to Intel Stratix 10 MX HBM2 Features topic.  
  — Modified fourth and sixth bullets in Intel Stratix 10 MX HBM2 Controller Features topic.  
• Chapter 2, Intel Stratix 10 MX HBM2 Architecture:  
  — Modified both figures in the Intel Stratix 10 MX HBM2 Architecture topic.  
  — Added sentence to the paragraph immediately before Figure 4, in the Intel Stratix 10 MX HBM2 Architecture topic.  
  — Expanded the last paragraph in the Intel Stratix 10 MX HBM2 Architecture topic.  
  — Changed the specified width of write and read data interfaces per AXI port from 128-bits to 256-bits, in the HBM2 burst transactions description in the Intel Stratix 10 MX HBM2 Controller Details topic.  
  — Removed the last sentence from the User interface vs HBM2 Interface Frequency description, in the Intel Stratix 10 MX HBM2 Controller Details topic.  
  — Added the ECC description near the end of the Intel Stratix 10 MX HBM2 Controller Details topic.  
• Chapter 3, Generating the High Bandwidth Memory (HBM2) Interface Intel FPGA IP:  
  — Changed chapter title from Generating the Intel Stratix 10 MX HBM2 IP to Generating the High Bandwidth Memory (HBM2) Interface Intel FPGA IP.  
  — Changed name of the menu selection in step 3 of the procedure and replaced the figure.  
  — Changed the IP name in topic titles throughout the chapter.  
  — Changed the description of the Core clock frequency parameter in the Group: General / Clocks section of the General Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP topic.  
  — In the Diagnostic Parameters for High Bandwidth Memory (HBM2) Interface Intel FPGA IP topic, removed two existing graphics and added one new one. Added Enable mixed traffic parameter to the Group: Diagnostics / Traffic Generator section.  
  — Revised the procedure in the Generating the Example Design topic.  
  — Removed the Intel Stratix 10 MX HBM2 IP Example Design for Synthesis topic.  
  — Added the High Bandwidth Memory (HBM2) Interface Intel FPGA IP Design Example and High Bandwidth Memory (HBM2) Interface Intel FPGA IP Pin Planning topics.  
  — In the High Bandwidth Memory (HBM2) Interface Intel FPGA IP Pin Planning topic, changed Jitter Specifications for the Input Reference Clocks from 10ps peak-to-peak to 20ps peak-to-peak.  |
• Chapter 4, Simulating the High Bandwidth Memory (HBM2) Interface Intel FPGA IP:
  — Changed product name from Intel Stratix 10 MX HBM2 IP to High Bandwidth Memory (HBM2) Interface Intel FPGA IP in topic titles throughout the chapter.

• Chapter 5, High Bandwidth Memory (HBM2) Interface Intel FPGA IP Interface:
  — Changed product name from Intel Stratix 10 MX HBM2 IP to High Bandwidth Memory (HBM2) Interface Intel FPGA IP in topic titles throughout the chapter.
  — Revised content of Clock Signals and Reset Signals topics.
  — Added Calibration Status Signals and Memory Interface Signals topics.
  — Added AXI Interface Signals and AXI Address Definition sections to the AXI User-interface Signals topic. Removed content from the descriptions of the axi_0_0_awaddr and axi_0_0_araddr ports, in tables 15 and 18, respectively.
  — Added User APB Interface Timing and User-controlled Access to the HBM2 Controller topics.
  — Added figures to the User Refresh Per Bank and User Refresh to All Banks sections in the User-controlled Refreshes topic.

• Chapter 6, High Bandwidth Memory (HBM2) Interface Intel FPGA IP Controller Performance:
  — Changed chapter title from Intel Stratix 10 MX HBM2 IP Controller Performance to High Bandwidth Memory (HBM2) Interface Intel FPGA IP Controller Performance.
  — Changed the IP name in topic titles throughout the chapter.
  — Added High Bandwidth Memory (HBM2) Interface Intel FPGA IP DRAM Temperature Readout topic.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
</table>