

This chapter describes the hard (on-chip) memory interface components available in the Arria V and Cyclone V device families.

## Hard Memory Interface

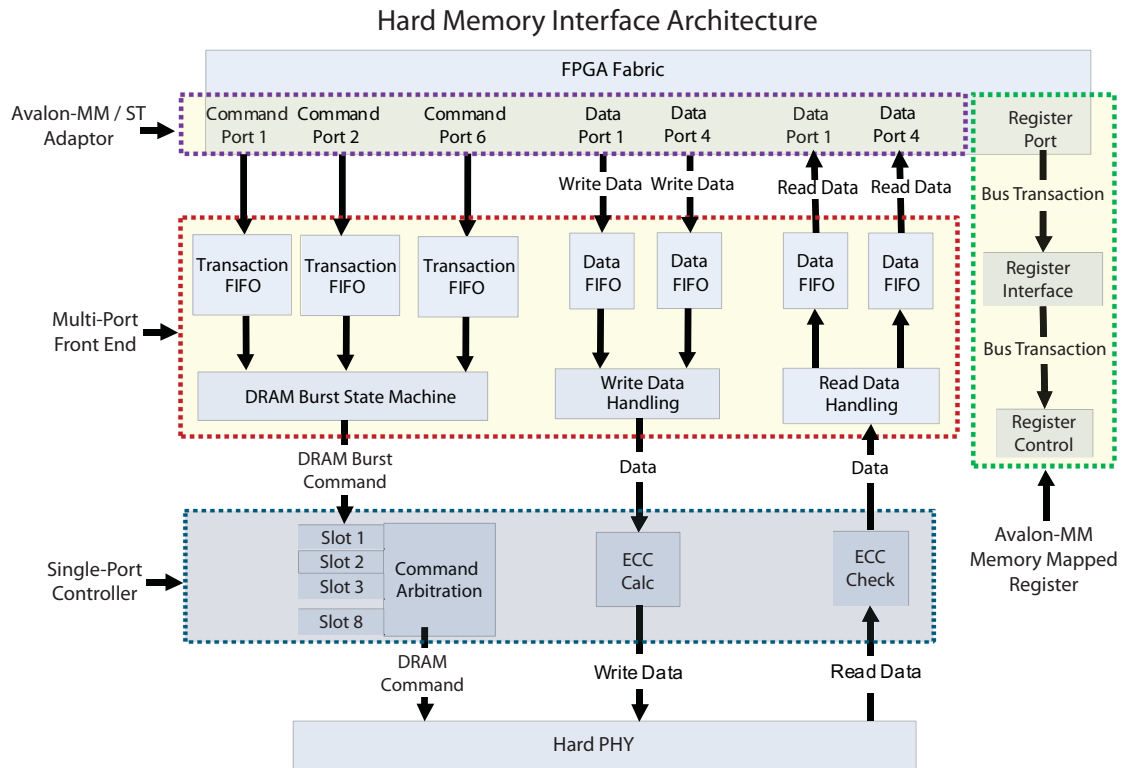
The Arria V device family includes hard memory interface components supporting DDR2 and DDR3 SDRAM, QDR II SRAM, and RLDRAM II memory protocols at speeds of up to 533 MHz. For the Quartus II software version 11.1, the Cyclone V device family supports simulation only for the hard memory interface; full soft interface support is available.

### High-Level Feature Description

Conceptually, the hard memory interface consists of three main parts: i) the multi-port front end (MPFE), which allows multiple independent accesses to the hard memory controller, ii) the hard memory controller, which initializes, refreshes, manages, and communicates with the external memory device, and iii) the hard PHY, which provides the physical layer interface to the external memory device.

Figure 3-1 shows the architecture of the Arria V hard memory interface.

Figure 3-1. Hard Memory Interface Architecture



## Multi-Port Front End (MPFE)

The multi-port front end and its associated fabric interface provide up to six command ports, four read-data ports and four write-data ports, through which user logic can access the hard memory controller. Each port can be configured as read only or write only, or read and write ports may be combined to form bidirectional data ports. Ports can be 32, 64, 128, or 256 data bits wide, depending on the number of ports used and the type (unidirectional or bidirectional) of the port.

## Fabric Interface

The fabric interface provides communication between the Avalon-ST-like internal protocol of the hard memory interface and the external Avalon-MM protocol. The following table summarizes the types of ports available in the fabric interface.

Table 3-1. Fabric Interface Port Types (Part 1 of 2)

Port Type	Description
Fabric command ports	Accept both read and write commands.

**Table 3-1. Fabric Interface Port Types (Part 2 of 2)**

Port Type	Description
Fabric 64-bit read data ports	Read and write data ports can be concatenated to form wider interfaces in power-of-two sizes (128-, 256-bit busses). Application which require only 32-bit interface can connect a 32-bit interface to the lower 32 bits of 64-bit port and configure that interface to be 32-bits wide. The remaining 32 nets to the hard memory controller are not usable when a port is configured to be 32-bits wide. 32-bit interfaces cannot be concatenated.
Fabric 64-bit write data ports	
Fabric register port	Provides access to address registers that control operation of the memory controller. Register values written across this interface can override values loaded during FPGA configuration.
Fabric write response port	Paired with the fabric write data ports to provide return acknowledgement of write operations being committed. A read operation received after the write acknowledgement on any controller port for the same address will see the updated data.

## Operation Ordering

Requests arriving at a given port are executed in the order in which they are received.

Requests arriving at different ports have no guaranteed order of service, except when a first transaction has completed before the second arrives.

## Multi-port Scheduling

User-configurable priority and weight settings determine the absolute and relative scheduling policy for each port.

### Port Scheduling

Multi-port scheduling is governed by two considerations: the absolute priority of a request and the weighting of a port.

The evaluation of absolute priority ensures that ports carrying higher-priority traffic are served ahead of ports carrying lower-priority traffic. The scheduler recognizes eight priority levels, with higher values representing higher priorities. Priority is absolute; for example, any transaction with priority seven will always be scheduled before transactions of priority six or lower.

When ports carry traffic of the same absolute priority, relative priority is determined based on port weighting. Port weighting is a five-bit value, and is determined by a weighted round robin (WRR) algorithm.

The scheduler can alter priority if the latency target for a transaction is exceeded. The scheduler tracks latency on a per-port basis, and counts the cycles that a transaction is pending. Each port has a priority escalation register and a pending counter engagement register. If the number of cycles in the pending counter engagement register elapse without a pending transaction being served, that transaction's priority is escalated.

To ensure that high-priority traffic is served quickly and that long and short bursts are effectively interleaved on ports, bus transactions longer than a single DRAM burst are scheduled as a series of DRAM bursts, with each burst arbitrated separately.

The scheduler uses a form of deficit round robin (DRR) scheduling algorithm which corrects for past over-servicing or under-servicing of a port. Each port has an associated weight which is updated every cycle, with a user-configured weight added to it and the amount of traffic served subtracted from it. The port with the highest weighting is considered the most eligible.

To ensure that lower priority ports do not build up large running weights while higher priority ports monopolize bandwidth, the hard memory controller's DRR weights are updated only when a port matches the scheduled priority. Hence, if three ports have traffic, two being priority 7 and one being priority 4, the weights for both ports at priority 7 are updated but the port with priority 4 remains unchanged.

The scheduler can be configured to lock onto a given port for a specified number of transactions when the scheduler schedules traffic at that priority level. The number of transactions is configurable on a per-port basis. For ports with large numbers of sequential addresses, you can use this feature to allow efficient open page accesses without risk of the open page being pushed out by other transactions.

### DRAM Burst Scheduling

DRAM burst scheduling recognizes addresses that access the same column/row combination—also known as open page accesses. Such operations are always served in the order in which they are received in the single-port controller.

Selection of DRAM operations is a two-stage process; first, each pending transaction must wait for its timers to be eligible for execution, then the transaction arbitrates against other transactions that are also eligible for execution.

The following rules govern transaction arbitration:

- High priority operations take precedence over lower priority operations
- If multiple operations are in arbitration, read operations will have precedence over write operations
- If multiple operations still exist, the oldest is served first

A high-priority transaction in the DRAM burst scheduler will win arbitration for that bank immediately if the bank is idle and the high-priority transaction's chip select/row/column address does not match an address already in the single-port controller. If the bank is not idle, other operations to that bank yield until the high-priority operation is finished. If the address matches another chip select/row/column, the high-priority transaction yields until the earlier transaction is completed.

You can force the DRAM burst scheduler to serve transactions in the order that they are received, by setting a bit in the register set.

### DRAM Power Saving Modes

The hard memory controller supports two DRAM power-saving modes: self-refresh, and fast/slow all-bank precharge powerdown exit.

Engagement of a DRAM power saving mode can occur due to inactivity, or in response to a user command.

The user command to enter power-down mode forces the DRAM burst-scheduling bank-management logic to close all banks and issue the power-down command. You can program the controller to power down when the DRAM burst-scheduling queue is empty for a specified number of cycles; the DRAM is reactivated when an active DRAM command is received.

## Hard Memory Controller

The following sections describe the memory controller portion of the hard memory interface.



The hard memory controller is functionally very similar to the High Performance Controller II (HPC II). For information on signals, refer to the [Functional Description—HPC II Controller](#) chapter.

### Clocking

The ports on the MPFE can be clocked at different frequencies, and synchronization is maintained by cross-domain clocking logic in the MPFE. Command ports can connect to different clocks, but the data ports associated with a given command port must be attached to the same clock as that command port. For example, a bidirectional command port that performs a 64-bit read/write function will have its read port and write port connected to the same clock as the command port.

### DRAM Interface

The DRAM interface is 40 bits wide, and can accommodate 8-bit, 16-bit, 16-bit plus ECC, 32-bit, or 32-bit plus ECC configurations. Any unused I/Os in the DRAM interface can be reused as user I/Os. The DRAM interface supports DDR2 and DDR3 memory protocols. Fast and medium speed grade devices are supported to 533 MHz for Arria V and 400 MHz for Cyclone V.

### ECC

The hard controller supports both error-correcting code (ECC) calculated by the controller and by the user.



User ECC is not available in 11.1, but will be available in a future release.

Controller ECC code employs standard Hamming logic which can detect and correct single-bit errors and detect double-bit errors. The controller ECC is available for 16-bit and 32-bit widths, each requiring an additional 8 bits of memory, resulting in an actual memory width of 24-bits and 40-bits, respectively.

In user ECC mode, all bits are treated as data bits, and are written to and read from memory. User ECC can implement nonstandard memory widths such as 24-bit or 40-bit, where ECC is not required.

## Controller ECC

Controller ECC provides the following features:

**Byte Writes**—The memory controller performs a read/modify/write operation to keep ECC valid when a subset of the bits of a word is being written. If an entire word is being written (but less than a full burst) and the DM pins are connected, no read is necessary and only that word is updated. If controller ECC is disabled, byte-writes have no performance impact.

**ECC Write Backs**—When a read operation detects a correctable error, the memory location is scheduled for a read/modify/write operation to correct the single-bit error.

**User ECC**—User ECC is 24-bits or 40-bits wide; with user ECC, the controller performs no ECC checking. The controller employs memory word addressing with byte enables, and can handle arbitrary memory widths. User ECC does not disable byte writes; hence, you must ensure that any byte writes do not result in corrupted ECC.

## Bonding of Memory Controllers

Bonding is a feature that allows data to be split between two memory controllers, providing the ability to service bandwidth streams similar to a single 64-bit controller. Bonding works by dividing data buses in proportion to the memory widths, and always sending a transaction to both controllers. When signals are returned, bonding ensures that both sets of signals are returned identically.

Bonding can be applied to asymmetric controllers, and allows controllers to have different memory clocks. Bonding does not attempt to synchronize the controllers, and different ports can be served in different order when two controllers are bonded.

The following signals require bonding circuitry:

**Read data return**—This bonding allows read data from the two controllers to return with effectively one ready signal to the bus master that initiated the bus transaction.

**Write ready**—For Avalon-MM, this is effectively bonding on the `waitrequest` signal.

**Write acknowledge**—Synchronization on returning the write completed signal.

For each of the above implementations, data is returned in order, hence the circuitry must match up for each valid cycle.

Bonded FIFO buffers must have identical FIFO numbers; that is, read FIFO 1 on controller 1 must be paired with Read FIFO 1 on controller 2.

### Data Return Bonding

Long loop times can lead to communications problems when using bonded controllers. The following effects are possible when using bonded controllers:

- If one memory controller completes its transaction and receives new data before the other controller, then the second controller can send data as soon as it arrives, and before the first controller acknowledges that the second controller has data.
- If the first controller has a single word in its FIFO buffer and the second controller receives single-word transactions, the second controller must determine whether the second word is a valid signal or not.

To accommodate the above effects, the hard controller maintains two counters for each bonded pair of FIFO buffers and implements logic that monitors those counters to ensure that the bonded controllers receive the same data on the same cycle, and that they send the data out on the same cycle.

### **FIFO Ready**

FIFO ready bonding is used for write command and write data buses. The implementation is similar to the data return bonding.

### **Bonding Latency Impact**

Bonding has no latency impact on ports that are not bonded.

### **Bonding Controller Usage**

Arria V devices employ three shared bonding controllers to manage the read data return bonding, write acknowledge bonding, and command/write data ready bonding.

The three bonding controllers require three pairs of bonding I/Os, each based on a six port count; this means that a bonded hard memory controller requires 21 input signals and 21 output signals for its connection to the fabric, and another 21 input signals and 21 output signals to the paired hard memory controller.

## **Hard PHY**

A physical layer interface (PHY) is embedded in the periphery of the Arria V device, and can run at the same high speed as the hard controller and hard sequencer. The hard PHY is located next to the hard controller. Differing device configurations have different numbers and sizes of hard controller and hard PHY pairs.

The hard PHY implements logic that connects the hard controller to the I/O ports. Because the hard controller and AFI interface support high frequencies, a portion of the sequencer is implemented as hard logic. The Nios II processor, the instruction/data RAM, and the Avalon fabric of the sequencer are implemented as core soft logic. The read/write manager and PHY manager components of the sequencer, which must operate at full rate, are implemented as hard logic in the hard PHY.

## **Interconnections**

The hard PHY resides on the device between the hard controller and the I/O register blocks. The hard PHY is instantiated or bypassed entirely, depending on the parameterization that you specify.

The hard PHY connects to the hard memory controller and the core, enabling the use of either the hard memory controller or a software-based controller. (In 11.1, you can have the hard controller and hard PHY, or the soft controller and soft PHY; however, the combination of soft controller with hard PHY is not supported.) The hard PHY also connects to the I/O register blocks and the DQS logic. The path between the hard PHY and the I/O register blocks can be bypassed, but not reconfigured—in other words, if you use the hard PHY datapath, the pins to which it connects are predefined and specified by the device pin table.

## Clock Domains

The hard PHY contains circuitry that uses the following clock domains:

**AFI clock domain (pll\_afi\_clk)**—The main full-rate clock signal that synchronizes most of the circuit logic.

**Avalon clock domain (pll\_avl\_clk)**—Synchronizes data on the internal Avalon bus, namely the Read/Write Manager, PHY Manager, and Data Manager data. The data is then transferred to the AFI clock domain. To ensure reliable data transfer between clock domains, the Avalon clock period must be an integer multiple of the AFI clock period, and the phases of the two clocks must be aligned.

**Address and Command clock domain (pll\_addr\_cmd\_clk)**—Synchronizes the global asynchronous reset signal, used by the I/Os in this clock domain.

## Hard Sequencer

The sequencer initializes the memory device and calibrates the I/Os, with the objective of maximizing timing margins and achieving the highest possible performance. When the hard memory controller is in use, a portion of the sequencer must run at full rate; for this reason, the Read/Write Manager, PHY Manager, and Data Manager are implemented as hard components within the hard PHY. The hard sequencer communicates with the soft-logic sequencer components (including the Nios II processor) via an Avalon bus.

## Document Revision History

Table 3-2 lists the revision history for this document.

**Table 3-2. Document Revision History**

Date	Version	Changes
November 2011	1.0	Initial release.