

This section includes the following chapters:

- Chapter 26, Introduction to the HPS Component
- Chapter 27, Instantiating the HPS Component
- Chapter 28, HPS Component Interfaces
- Chapter 29, Simulating the HPS Component



For information about the revision history for chapters in this section, refer to “Document Revision History” in each individual chapter.

The hard processor system (HPS) component is a soft component that you can instantiate in the FPGA fabric of the Cyclone® V SoC FPGA. It enables other soft components to interface with the HPS hard logic. The HPS component itself has a small footprint in the FPGA fabric, because its only purpose is to enable soft logic to connect to the extensive hard logic in the HPS.

- 

For a description of the HPS and its integration into the system on a chip (SoC), refer to the *Cyclone V Device Datasheet*. For a description of HPS system architecture and features, refer to the *Introduction to the Hard Processor* chapter in volume 3 of the *Cyclone V Device Handbook* and the *CoreSight Debug and Trace* chapter in volume 3 of the *Cyclone V Device Handbook*.
- 

For descriptions of individual peripheral architectures and features, refer to the following chapters and sections:

 - “HPS Block Diagram and System Integration” in the *Introduction to the Hard Processor* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “Clock Manager Block Diagram and System Integration” in the *Clock Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “Reset Manager Block Diagram and System Integration” in the *Reset Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “Interconnect Block Diagram and System Integration” in the *Interconnect* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “AXI Bridges Block Diagrams and System Integration” in the *HPS-FPGA AXI Bridges* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “Cortex-A9 MPU Subsystem Block Diagram and System Integration” in the *Cortex-A9 Microprocessor Unit Subsystem* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “CoreSight Debug and Trace Block Diagram and System Integration” in the *CoreSight Debug and Trace* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “SDRAM Controller Subsystem Block Diagram and System Integration” in the *SDRAM Controller Subsystem* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “On-Chip RAM Block Diagram and System Integration” in the *On-Chip Memory* chapter in volume 3 of the *Cyclone V Device Handbook*.
 - “Boot ROM Block Diagram and System Integration” in the *On-Chip Memory* chapter in volume 3 of the *Cyclone V Device Handbook*.

- “NAND Flash Controller Block Diagram and System Integration” in the *NAND Flash Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “SD/MMC Controller Block Diagram and System Integration” in the *SD/MMC Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “Quad SPI Flash Controller Block Diagram and System Integration” in the *Quad SPI Flash Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “FPGA Manager Block Diagram and System Integration” in the *FPGA Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “System Manager Block Diagram and System Integration” in the *System Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “Scan Manager Block Diagram and System Integration” in the *Scan Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “DMA Controller Block Diagram and System Integration” in the *DMA Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “EMAC Block Diagram and System Integration” in the *Ethernet Media Access Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “USB OTG Controller Block Diagram and System Integration” in the *USB 2.0 OTG Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “SPI Block Diagram and System Integration” in the *SPI Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “I²C Controller Block Diagram and System Integration” in the *I²C Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “UART Controller Block Diagram and System Integration” in the *UART Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “General-Purpose I/O Interface Block Diagram and System Integration” in the *General-Purpose I/O Interface* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “Timer Block Diagram and System Integration” in the *Timer* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “Watchdog Timer Block Diagram and System Integration” in the *Watchdog Timer* chapter in volume 3 of the *Cyclone V Device Handbook*.
- “CAN Controller Block Diagram and System Integration” in the *CAN Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.



The address map and register definitions reside in the [hps.html](#) file that accompanies this handbook volume. Click the link to open the file.

To view the description and base address for a specific peripheral, scroll to and click the link for the peripheral’s module name.

To then view the register and field descriptions, scroll to and click the register names. The register addresses are offsets relative to the base address of each module instance.

Document Revision History

Table 26–1 shows the revision history for this document.

Table 26–1. Document Revision History

Date	Version	Changes
June 2012	1.0	Initial release.
May 2012	0.1	Preliminary draft.

You instantiate the hard processor system (HPS) component in Qsys. The HPS is available in the component library under **Embedded Processors**. This chapter describes the parameters available in the HPS component parameter editor, which opens when you add or edit an HPS component.

- The HPS requires specific device targets. For a detailed list of supported devices, refer to the *Cyclone® V Device Datasheet*.
- For general information about using Qsys, refer to the *Creating a System with Qsys* chapter in volume 1 of the *Quartus® II Handbook*.

Configuring FPGA Interfaces

This section describes parameters on the **FPGA Interfaces** tab.

- For general information about interfaces, refer to the *HPS Component Interfaces* chapter in volume 3 of the *Cyclone V Device Handbook*.

General Interfaces

This section describes parameters in the **General** group on the **FPGA Interfaces** tab. When enabled, the interfaces described in [Table 27-1](#) become visible in the HPS component.

Table 27-1. General Parameters

Parameter Name	Parameter Description	Interface Name
Enable MPU standby and event signals	Enables interfaces that perform the following functions: <ul style="list-style-type: none"> ■ Notify the FPGA fabric that the microprocessor unit (MPU) is in standby mode. ■ Wake up an MPCore processor from a wait for event (WFE) state. 	h2f_mpu_events
Enable MPU general purpose signals	Enables a pair of 32-bit unidirectional general-purpose interfaces between the FPGA fabric and the FPGA manager in the HPS portion of the SoC device.	h2f_mpu_gp
Enable FPGA-to-HPS Interrupts	Enables interface for FPGA interrupt signals to the MPU (in the HPS).	f2h_irq0 f2h_irq1
Enable Debug APB interface	Enables debug interface to the FPGA, allowing access to debug components in the HPS. ⁽¹⁾	h2f_debug_apb h2f_debug_apb_sideband h2f_debug_apb_clock
Enable System Trace Macrocell hardware events	Enables system trace macrocell (STM) hardware events, allowing logic inside the FPGA to insert messages into the trace stream. ⁽¹⁾	f2h_stm_hw_events
Enable FPGA Cross Trigger interface	Enables the cross trigger interface (CTI), which allows trigger sources and sinks to interface with the embedded cross trigger (ECT). ⁽¹⁾	h2f_cti h2f_cti_clock
Enable FPGA Trace Port Interface Unit	Enables an interface between the trace port interface unit (TPIU) and logic in the FPGA. The TPIU is a bridge between on-chip trace sources and a trace port. ⁽¹⁾	h2f_tpiu h2f_tpiu_clock_in

Note to Table 27-1:

(1) For information about this functionality, refer to the [CoreSight Debug and Trace](#) chapter in volume 3 of the Cyclone V Device Handbook.


Boot and Clock Selection Interfaces

This section describes parameters in the **Boot and Clock Selection** group in the **FPGA Interfaces** tab.

Table 27-2 lists the available parameters.

Table 27-2. Boot and Clock Selection Parameters

Parameter Name	Parameter Description
Enable boot from FPGA ready	Enables an input to the HPS indicating whether a preloader is available in on-chip RAM. If the input is asserted, a preloader image is ready at memory location 0.
Enable boot from FPGA on failure	Enables an input to the HPS indicating whether a fallback preloader is available in on-chip RAM. If the input is asserted, a fallback preloader image is ready at memory location 0. The fallback preloader is to be used only if the HPS boot ROM does not find a valid preloader image in the selected flash memory device.

 For detailed information about the HPS boot sequence, refer to the *Booting and Configuration* appendix in volume 3 of the *Cyclone V Device Handbook*.

AXI Bridges

This section describes parameters in the **AXI Bridges** group on the **FPGA Interfaces** tab.

Table 27-3. Bridge Parameters

Parameter Name	Parameter Description	Interface Name
FPGA-to-HPS interface width	Enable or disable the FPGA-to-HPS interface; if enabled, set the data width to 32, 64, or 128 bits.	f2h_axi_slave
HPS-to-FPGA interface width	Enable or disable the HPS-to-FPGA interface; if enabled, set the data width to 32, 64, or 128 bits.	h2f_axi_master
Lightweight HPS-to-FPGA interface width	Enable or disable the lightweight HPS-to-FPGA interface. When enabled, the data width is 32 bits.	h2f_lw_axi_master

To facilitate accessing these slaves from a memory-mapped master with a smaller address width, you can use the Altera® Address Span Extender. The address span extender is discussed in “*Using the Address Span Extender Component*” on page 27-9.

 For more information, refer to the *Interconnect* chapter in volume 3 of the *Cyclone V Device Handbook*.

FPGA-to-HPS SDRAM Interface

This section describes parameters in the **FPGA-to-HPS SDRAM Interface** group on the **FPGA Interfaces** tab.

You can add one or more SDRAM ports that make the HPS SDRAM subsystem accessible to the FPGA fabric.

You can configure the slave interface to a data width of 32, 64, 128, or 256 bits. To facilitate accessing this slave from a memory-mapped master with a smaller address width, you can use the Altera Address Span Extender. The address span extender is discussed in “Using the Address Span Extender Component” on page 27-9.

On the **FPGA Interfaces** tab, in the **FPGA to HPS SDRAM Interface** table, use + or – to add or remove FPGA-to-HPS SDRAM interfaces. The **Name** column denotes the interface name. Table 27-4 shows the parameters available for each SDRAM interface.

Table 27-4. FPGA-to-HPS SDRAM Interface Parameters

Parameter Name	Parameter Description
Name	Port name (auto assigned as shown in Table 27-5)
Type	Interface type: <ul style="list-style-type: none"> ■ AXI-3 ■ Avalon-MM Bidirectional ■ Avalon-MM Write-only ■ Avalon-MM Read-only
Width	32, 64, 128, or 256

Table 27-5. FPGA-to-HPS SDRAM Port and Interface Names

Port Name	Interface Name
f2h_sdram0	f2h_sdram0_data
f2h_sdram1	f2h_sdram1_data
f2h_sdram2	f2h_sdram2_data
f2h_sdram3	f2h_sdram3_data
f2h_sdram4	f2h_sdram4_data
f2h_sdram5	f2h_sdram5_data



For more information, refer to the *SDRAM Controller Subsystem* chapter in volume 3 of the *Cyclone V Device Handbook*.

Reset Interfaces

This section describes parameters in the **Resets** group on the **FPGA Interfaces** tab. You can enable most resets on an individual basis. Table 27–6 lists the available reset parameters.

Table 27–6. Reset Parameters

Parameter Name	Parameter Description	Interface Name
Enable HPS-to-FPGA cold reset output	Enable interface for HPS-to-FPGA cold reset output	h2f_cold_reset
Enable HPS warm reset handshake signals	Enable an additional pair of reset handshake signals allowing soft logic to notify the HPS when it is safe to initiate a warm reset in the FPGA fabric.	h2f_warm_reset_handshake
Enable FPGA-to-HPS debug reset request	Enable interface for FPGA-to-HPS debug reset request	f2h_debug_reset_req
Enable FPGA-to-HPS warm reset request	Enable interface for FPGA-to-HPS warm reset request	f2h_warm_reset_req
Enable FPGA-to-HPS cold reset request	Enable interface for FPGA-to-HPS cold reset request	f2h_cold_reset_req



For more information about the reset interfaces, refer to “Functional Description of the Reset Manager” in the *Reset Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.

DMA Peripheral Request

This section describes parameters in the **DMA Peripheral Request** group on the **FPGA Interfaces** tab.

You can enable each direct memory access (DMA) controller peripheral request ID individually. Each request ID enables an interface for FPGA soft logic to request one of eight logical DMA channels to the FPGA.



Peripheral request IDs 4–7 are shared with the controller area network (CAN) controllers.



For more information, refer to the *DMA Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.

Configuring Peripheral Pin Multiplexing

This section describes parameters on the **Peripheral Pin Multiplexing** tab.

Configuring Peripherals

The **Peripheral Pin Multiplexing** tab contains a group of parameters for each available type of peripheral. You can enable one or more instances of each peripheral type by selecting an HPS I/O pin set for each instance. When enabled, some peripherals also have a mode settings specific to their functions.

Each list in the **Peripheral Pin Multiplexing** tab has a hint describing in detail the options available in the list. The hint for each pin multiplexing list shows the I/O pins used by each available pin set. The hint for each mode list shows the signals used by each available mode.

View each hint by hovering over the corresponding list.

The tooltips for the combo boxes show useful information. The pin multiplexing parameters have tables showing the pin destinations and the mode parameters show which signals are used in which mode.

You can enable the following types of peripherals. For details of peripheral-specific settings, refer to the chapter for each peripheral:


- Ethernet Media Access Controller—*Ethernet Media Access Controller* chapter in volume 3 of the *Cyclone V Device Handbook*
- NAND Flash Controller—*NAND Flash Controller* chapter in volume 3 of the *Cyclone V Device Handbook*
- Quad serial peripheral interface (SPI) Flash Controller—*Quad SPI Flash Controller* chapter in volume 3 of the *Cyclone V Device Handbook*
- Secure Digital / MultiMediaCard (SD/MMC) Controller—*SD/MMC Controller* chapter in volume 3 of the *Cyclone V Device Handbook*
- USB 2.0 On-The-Go (OTG) Controllers—*USB 2.0 OTG Controller* chapter in volume 3 of the *Cyclone V Device Handbook*
- SPI Controllers—*SPI Controller* chapter in volume 3 of the *Cyclone V Device Handbook*
- UART Controllers—*UART Controller* chapter in volume 3 of the *Cyclone V Device Handbook*
- Inter-integrated circuit (I²C) Controllers—*I2C Controller* chapter in volume 3 of the *Cyclone V Device Handbook*
- CAN Controllers—*CAN Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.
- Trace port interface unit (TPIU)—*CoreSight Debug and Trace* chapter in volume 3 of the *Cyclone V Device Handbook*. Enabling the TPIU exposes trace signals to the device pins.

Connecting Unassigned Pins to GPIO

On the **Peripheral Pin Multiplexing** tab, the **Conflicts** table shows pins that are not assigned to any peripheral. By default, general-purpose I/O (GPIO) is disabled on these pins. You can enable a pin as GPIO by changing the **GPIO Enabled** field in the table. The table also shows the GPIO pin number assigned to that pin.


Resolving Pin Multiplexing Conflicts

Use the **Conflicts** table to view pins with invalid multiple assignments. The table shows one of two or more peripheral interfaces assigned to the same pin(s). You can use the peripherals' pin configuration to determine which other peripheral(s) are in conflict, and to resolve the conflict.

 For detailed information about available HPS pin configurations, refer to the *Cyclone V Device Family Pin Connection Guidelines*.

Configuring HPS Clocks

This section describes parameters on the **HPS Clocks** tab.

 For general information about clock signals, refer to the *Clock Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.

User Clocks

This section describes parameters in the **User Clocks** group on the **HPS Clocks** tab.


When you enable a user clock, you must manually enter its maximum frequency for timing analysis. The TimeQuest Timing Analyzer has no other information about how software running on the HPS configures the phase-locked loop (PLL) outputs. Each possible clock, including clocks that are available from peripherals, has its own parameter for describing the clock frequency.

Table 27-7 lists the user clock parameters. The frequencies that you provide are the maximum expected frequencies. The actual clock frequencies can be modified through the register interface, for example by software running on the microprocessor unit (MPU). For further details, refer to “Selecting PLL Output Frequency and Phase” on page 27-9.

Table 27-7. User Clock Parameters

Parameter Name	Parameter Description	Clock Interface Name
Enable HPS-to-FPGA user 0 clock	Enable main PLL from HPS to FPGA	h2f_user0_clock
User 0 clock frequency	Specify the maximum expected frequency for the main PLL	
Enable HPS-to-FPGA user 1 clock	Enable peripheral PLL from HPS to FPGA	h2f_user1_clock
User 1 clock frequency	Specify the maximum expected frequency for the peripheral PLL	
Enable HPS-to-FPGA user 2 clock	Enable SDRAM PLL from HPS to FPGA	h2f_user2_clock
User 2 clock frequency	Specify the maximum expected frequency for the SDRAM PLL	

The clock frequencies you provide are reported in a Synopsys Design Constraints File (.sdc) generated by Qsys.

 For details about driving these clocks, refer to the *Clock Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.

PLL Reference Clocks

This section describes parameters in the **PLL Reference Clocks** group on the **HPS Clocks** tab.

Table 27-8. PLL Reference Clock Parameters

Parameter Name	Parameter Description	Clock Interface Name
Enable FPGA-to-HPS peripheral PLL reference clock	Enable the interface for FPGA fabric to supply reference clock to HPS peripheral PLL	f2h_periph_ref_clock
Enable FPGA-to-HPS SDRAM PLL reference clock	Enable the interface for FPGA fabric to supply reference clock to HPS SDRAM PLL	f2h_sdram_ref_clock

 For more information, refer to the *Clock Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.

Configuring the External Memory Interface


This section describes parameters on the **SDRAM** tab.

The HPS supports one memory interface implementing double data rate 2 (DDR2), double data rate 3 (DDR3), and low-power double data rate 2 (LPDDR2) protocols. The interface can be up to 40 bits wide with optional error correction code (ECC).


Configuring the HPS SDRAM controller is similar to configuring any other Altera SDRAM controller. There are several important differences:

- The HPS parameter editor supports all SDRAM protocols with one tab. When you parameterize the SDRAM controller, you must specify the memory protocol: DDR2, DDR3, or LPDDR2.

To select the memory protocol, select DDR2, DDR3, or LPDDR2 from the **SDRAM Protocol** list in the **PHY Settings** tab in the **SDRAM** tab. After you select the protocol, settings not applicable to that protocol are disabled.

 Many HPS SDRAM controller settings are the same as for Altera's dedicated DDR2, DDR3, and LPDDR2 controllers. This section only describes SDRAM parameters that are specific to the HPS component.

- Because the HPS memory controller is not configurable through the Quartus II software, the Controller and Diagnostic tabs are not present in the HPS parameter editor.
- Some settings, such as the controller settings, are not included because they can only be configured through the register interface, for example by software running on the MPU.
- Unlike the memory interface clocks in the FPGA, the memory interface clocks for the HPS are initialized by the boot-up code using values provided by the configuration process. You can accept the values provided by UniPHY, or you can use your own PLL settings, as described in *"Selecting PLL Output Frequency and Phase"*.

 The HPS does not support external memory interface (EMIF) synthesis generation, compilation, or timing analysis.

The HPS memory controller cannot be bonded with a memory controller on the FPGA portion of the device.



For detailed information about SDRAM controller parameters, refer to the following chapters:

- The *Implementing and Parameterizing Memory IP* chapter in volume 2 of the *External Memory Interface Handbook*.
- The *Functional Description—Hard Memory Interface* chapter in volume 3 of the *External Memory Interface Handbook*. “EMI-Related HPS Features in SoC Devices” describes features specific to the HPS SDRAM controller.

Selecting PLL Output Frequency and Phase

You select PLL output frequency and phase with controls in the **PHY Settings** tab in the **SDRAM** tab. In the HPS, PLL frequencies and phases are set by software at system startup. A PLL might not be able to produce the exact frequency that you specify in **Memory clock frequency**. Normally, the Quartus II software sets **Achieved memory clock frequency** to the closest achievable frequency, using an algorithm that tries to balance frequency accuracy against clock jitter. This clock frequency is used for timing analysis by the TimeQuest analyzer.

It is possible to use a different software algorithm for configuring the PLLs. You can force the **Achieved memory clock frequency** box to take on the same value as **Memory clock frequency**, by turning on **Use specified frequency instead of calculated frequency** in the **PHY Settings** tab, under **Clocks**.



If you turn on **Use specified frequency instead of calculated frequency**, the Quartus II software assumes that the value in the **Achieved memory clock frequency** box is correct. If it is not, timing analysis results are incorrect.

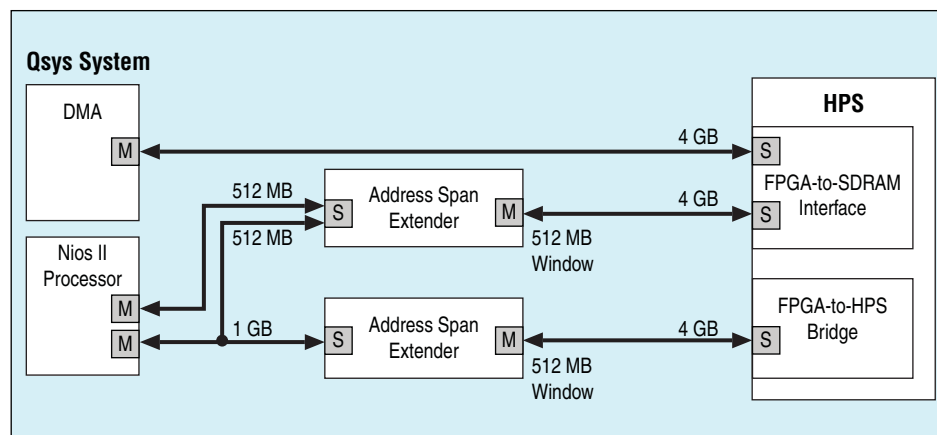
Using the Address Span Extender Component

The FPGA-to-HPS bridge and FPGA-to-HPS SDRAM memory-mapped interfaces expose their entire 4 GB address spaces to the FPGA fabric. The Address Span Extender component provides a memory-mapped window into the address space that it masters. Using the address span extender, you can expose portions of the HPS memory space without needing to expose the entire 4-GB address space.

You can use the address span extender between a soft logic master and an FPGA-to-HPS bridge or FPGA-to-HPS SDRAM interface. This component reduces the number of address bits required for a master to address a memory-mapped slave interface located in the HPS.

Figure 27-1 shows how two address span extender components might be used in a system with the HPS.

Figure 27-1. Address Span Extender



You can also use the address span extender in the HPS-to-FPGA direction, for slave interfaces in the FPGA. In this case, the HPS-to-FPGA bridge exposes a limited, variable address space in the FPGA, which can be paged in using the address span extender.

For example, suppose that the HPS-to-FPGA bridge has a 1 GB span, and the HPS needs to access three independent 1 GB memories in the FPGA portion of the device. To achieve this, the HPS programs the address span extender to access one SDRAM (1 GB) in the FPGA at a time. This technique is commonly called paging or windowing.

For more information about the address span extender, refer to “Bridges” in the *Qsys Interconnect and System Design Components* chapter in volume 1 of the *Quartus II Handbook*.

Generating and Compiling the HPS Component


The process of generating and compiling an HPS design is very similar to the process for any other Qsys project. Perform the following steps:

1. Generate the design with Qsys. The generated files include an **.sdc** file containing clock timing constraints. If simulation is enabled, simulation files are also generated.

For information about generating a Qsys project, refer to the *Creating a System with Qsys* chapter in volume 1 of the *Quartus II Handbook*. For a description of the simulation files generated, refer to “Simulation Flow” in the *Simulating the HPS Component* chapter in volume 3 of the *Cyclone V Device Handbook*.

2. Add **system.qip** to the Quartus II project. **system.qip** is the Quartus II IP File for the HPS component, generated by Qsys.


3. Perform Analysis and Elaboration with the Quartus II software.
4. Assign constraints to the SDRAM component. When Qsys generates the HPS component (step 1), it generates the pin assignment Tcl Script File (.tcl) to perform memory assignments. The script file name is `<qsys_system_name>_pin_assignments.tcl`, where `<qsys_system_name>` is the name of your Qsys system. Run this script to assign constraints to the SDRAM component.

 For information about running the pin assignment script, refer to “MegaWizard Plug-In Manager Flow” in the *Implementing and Parameterizing Memory IP* chapter in volume 2 of the *External Memory Interface Handbook*.

You do not need to specify pin assignments other than memory assignments. When you configure pin multiplexing as described in “*Configuring Peripheral Pin Multiplexing*” on page 27-5, you implicitly make pin assignments for all HPS peripherals. Each peripheral is routed exclusively to the pins you specify. HPS I/O signals are exported to the top level of the Qsys design, with information enabling the Quartus II software to make pin assignments automatically.

You can view and modify the assignments in the **Peripheral Pin Multiplexing** tab. You can also view the assignments in the Quartus fitter report.

5. Compile the design with the Quartus II software.
6. Optionally back-annotate the SDRAM pin assignments, to eliminate pin assignment warnings the next time you compile the design.

 For information about back-annotating pin assignments, refer to *About Back-Annotating Assignments* in Quartus II Help.

Document Revision History

Table 27-9 shows the revision history for this document.


Table 27-9. Document Revision History

Date	Version	Changes
November 2012	1.1	<ul style="list-style-type: none"> ■ Added debug interfaces ■ Added boot options ■ Corrected slave address width ■ Corrected SDRAM interface widths ■ Added TPIU peripheral ■ Added .sdc file generation ■ Added .tcl script for memory assignments
June 2012	1.0	Initial release.
May 2012	0.1	Preliminary draft.

This chapter describes the interfaces, including clocks and resets, implemented by the hard processor system (HPS) component.

The majority of the resets can be enabled on an individual basis. The exception is the `h2f_reset` interface, which is always enabled.

You must declare the clock frequency of each HPS-to-FPGA clock for timing purposes. Each possible clock, including ones that are available from peripherals, has its own parameter for describing the clock frequency. Declaring the clock frequency for HPS-to-FPGA clocks specifies how you plan to configure the PLLs and peripherals, to enable TimeQuest to accurately estimate system timing. It has no effect on PLL settings.

 For information about instantiating the HPS component, refer to the *Instantiating the HPS Component* chapter in volume 3 of the *Cyclone® V Device Handbook*. For Avalon™ protocol timing, refer to *Avalon Interface Specifications*. For Advanced Microcontroller Bus Architecture (AMBA®) Advanced eXtensible Interface (AXI™) protocol timing, refer to the *AMBA AXI Protocol Specification v1.0*, which you can download from the ARM website (infocenter.arm.com).

Memory-Mapped Interfaces

FPGA-to-HPS Bridge

Table 28–1. FPGA-to-HPS Bridges and Clocks

Interface Name	Description	Associated Clock Interface ⁽¹⁾
<code>f2h_axi_slave</code>	FPGA-to-HPS AXI slave interface	<code>f2h_axi_clock</code>

Note to Table 28–1:

(1) Refer to “Clocks” on page 28–4 for information about clock interfaces.

The FPGA-to-HPS interface is a configurable data width AXI slave allowing FPGA masters to issue transactions to the HPS. This interface allows the FPGA fabric to access the majority of the HPS slaves. This interface also provides a coherent memory interface.


The FPGA-to-HPS interface is an AXI-3 compliant interface with the following features:

- Configurable data width: 32, 64, or 128 bits

- Accelerator Coherency Port (ACP) sideband signals
- HPS-side AXI bridge to manage clock crossing, buffering, and data width conversion.


Other interface standards in the FPGA fabric, such as connecting to Avalon® Memory-Mapped (Avalon-MM) interfaces, can be supported through the use of soft logic adaptors. The Qsys system integration tool automatically generates adaptor logic to connect AXI to Avalon-MM interfaces.

This interface has an address width of 32 bits. To access existing Avalon-MM/AXI masters, you can use the Altera® Address Span Extender.

 For more information about the FPGA-to-HPS bridge, refer to the *HPS-FPGA AXI Bridges* chapter in volume 3 of the *Cyclone V Device Handbook*. For information about the address span extender, refer to “Using the Address Span Extender Component” in the *Instantiating the HPS Component* chapter in volume 3 of the *Cyclone V Device Handbook*.

ACP Sideband Signals

For communication with the ACP on the microprocessor unit (MPU) subsystem, AXI sideband signals are used to describe the inner cacheable attributes for the transaction.

 For more information about the ACP sideband signals, refer to the *Cortex-A9 Microprocessor Unit Subsystem* chapter in volume 3 of the *Cyclone V Device Handbook*.

HPS-to-FPGA and Lightweight HPS-to-FPGA Bridges

Table 28-2. HPS-to-FPGA and Lightweight HPS-to-FPGA Bridges and Clocks

Interface Name	Description	Associated Clock Interface ⁽¹⁾
h2f_axi_master	HPS-to-FPGA AXI master interface	h2f_axi_clock
h2f_lw_axi_master	HPS-to-FPGA lightweight AXI master interface	h2f_lw_axi_clock

Note to Table 28-2:

(1) Refer to “Clocks” on page 28-4 for information about clock interfaces.

The HPS-to-FPGA interface is a configurable data width AXI master (32, 64, or 128-bit) that allows HPS masters to issue transactions to the FPGA fabric.

The lightweight HPS-to-FPGA interface is a 32-bit AXI master that allows HPS masters to issue transactions to the FPGA fabric.

Both HPS-to-FPGA interfaces are AXI-3 compliant. The HPS-side AXI bridges manage clock crossing, buffering, and data width conversion where necessary.

Other interface standards in the FPGA fabric, such as connecting to Avalon-MM interfaces, can be supported through the use of soft logic adaptors. The Qsys system integration tool automatically generates adaptor logic to connect AXI to Avalon-MM interfaces.

Each AXI bridge accepts a clock input from the FPGA fabric and performs clock domain crossing internally. The exposed AXI interface operates on the same clock domain as the clock supplied by the FPGA fabric.



For more information, refer to the *HPS-FPGA AXI Bridges* chapter in volume 3 of the *Cyclone V Device Handbook*.

FPGA-to-HPS SDRAM Interface

The FPGA-to-HPS SDRAM interface is a direct connection between the FPGA fabric and the HPS SDRAM controller. This interface is highly configurable, allowing a mix between number of ports and port width. The interface supports both AXI-3 and Avalon-MM protocols.

Table 28-3. HPGA-to-HPS SDRAM Interfaces and Clocks

Interface Name	Description	Associated Clock Interface ⁽¹⁾
f2h_sdram0_data	SDRAM AXI or Avalon-MM port 0	f2h_sdram0_clock
f2h_sdram1_data	SDRAM AXI or Avalon-MM port 1	f2h_sdram1_clock
f2h_sdram2_data	SDRAM AXI or Avalon-MM port 2	f2h_sdram2_clock
f2h_sdram3_data	SDRAM AXI or Avalon-MM port 3	f2h_sdram3_clock
f2h_sdram4_data	SDRAM AXI or Avalon-MM port 4	f2h_sdram4_clock
f2h_sdram5_data	SDRAM AXI or Avalon-MM port 5	f2h_sdram5_clock

Note to Table 28-3:

(1) Refer to “Clocks” on page 28-4 for information about clock interfaces.

The FPGA-to-HPS SDRAM interface is a configurable interface to the multi-port SDRAM controller.

The total data width of all interfaces is limited to a maximum of 256 bits in the read direction and 256 bits in the write direction. The interface is implemented as four 64-bit read ports and four 64-bit write ports. As a result, the minimum data width used by the interface is 64 bits, regardless of the number or type of interfaces.

You can configure this interface the following ways:

- AXI-3 or Avalon-MM protocol
- Number of interfaces
- Data width of interfaces

The FPGA-to-HPS SDRAM interface supports six command ports, allowing up to six Avalon-MM interfaces or three bidirectional AXI interfaces.

Each command port is available either to implement a read or write command port for AXI, or to form part of an Avalon-MM interface.

You can use a mix of Avalon-MM and AXI interfaces, limited by the number of command/data ports available. Some AXI features are not present in Avalon-MM interfaces.

This interface has an address width of 32 bits. To access existing Avalon-MM/AXI masters, you can use the Altera Address Span Extender.



For more information about available combinations of interfaces and ports, refer to the *SDRAM Controller Subsystem* chapter in volume 3 of the *Cyclone V Device Handbook*. For information about the address span extender, refer to “Using the Address Span Extender Component” in the *Instantiating the HPS Component* chapter in volume 3 of the *Cyclone V Device Handbook*.

Clocks

The HPS-to-FPGA clock interface supplies physical clocks and resets to the FPGA. These clocks and resets are generated in the HPS.

Alternative Clock Inputs to HPS PLLs

This section lists alternative clock inputs to HPS PLLs.

- `f2h_periph_ref_clock`—FPGA-to-HPS peripheral PLL reference clock. You can connect this clock input to a clock in your design that is driven by the clock network on the FPGA side.
- `f2h_sdram_ref_clock`—FPGA-to-HPS SDRAM PLL reference clock. You can connect this clock to a clock in your design that is driven by the clock network on the FPGA side.

User Clocks

A user clock is a PLL output that is connected to the FPGA fabric rather than the HPS. You can connect a user clock to logic that you instantiate in the FPGA fabric.

- `h2f_user0_clock`—HPS-to-FPGA user clock, driven from main PLL
- `h2f_user1_clock`—HPS-to-FPGA user clock, driven from peripheral PLL
- `h2f_user2_clock`—HPS-to-FPGA user clock, driven from SDRAM PLL

AXI Bridge FPGA Interface Clocks

The AXI interface has an asynchronous clock crossing in the FPGA-to-HPS bridge. The FPGA-to-HPS and HPS-to-FPGA interfaces are synchronized to clocks generated in the FPGA fabric. These interfaces can be asynchronous to one another. The SDRAM controller’s multiport front end (MPFE) transfers the data between the FPGA and HPS clock domains.

- `f2h_axi_clock`—AXI slave clock for FPGA-to-HPS bridge, generated in FPGA fabric
- `h2f_axi_clock`—AXI master clock for HPS-to-FPGA bridge, generated in FPGA fabric
- `h2f_lw_axi_clock`—AXI master clock for lightweight HPS-to-FPGA bridge, generated in FPGA fabric

SDRAM Clocks

You can configure the HPS component with up to six FPGA-to-HPS SDRAM clocks.

Each command channel to the SDRAM controller has an individual clock source from the FPGA fabric. The interface clock is always supplied by the FPGA fabric, with clock crossing occurring on the HPS side of the boundary.

The FPGA-to-HPS SDRAM clocks are driven by soft logic in the FPGA fabric.

- f2h_sdram0_clock—SDRAM clock for port 0
- f2h_sdram1_clock—SDRAM clock for port 1
- f2h_sdram2_clock—SDRAM clock for port 2
- f2h_sdram3_clock—SDRAM clock for port 3
- f2h_sdram4_clock—SDRAM clock for port 4
- f2h_sdram5_clock—SDRAM clock for port 5

Resets

This section describes the reset interfaces to the HPS component.



For details about the HPS reset sequences, refer to “Functional Description of the Reset Manager” in the *Reset Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.

HPS-to-FPGA Reset Interfaces

The following interfaces allow the HPS to reset soft logic in the FPGA fabric:

- h2f_reset—HPS-to-FPGA cold and warm reset
- h2f_cold_reset—HPS-to-FPGA cold reset
- h2f_warm_reset_handshake—Warm reset request and acknowledge interface between HPS and FPGA

HPS External Reset Sources

The following interfaces allow soft logic in the FPGA fabric to reset the HPS:

- f2h_cold_reset_req—FPGA-to-HPS cold reset request
- f2h_warm_reset_req—FPGA-to-HPS warm reset request
- f2h_dbg_reset_req—FPGA-to-HPS debug reset request

Debug and Trace Interfaces

Trace Port Interface Unit

The TPIU is a bridge between on-chip trace sources and a trace port.

- h2f_tpiu
- h2f_tpiu_clock_in

FPGA System Trace Macrocell Events Interface

The system trace macrocell (STM) hardware events allow logic in the FPGA to insert messages into the trace stream.

- f2h_stm_hw_events

FPGA Cross Trigger Interface

The cross trigger interface (CTI) allows trigger sources and sinks to interface with the embedded cross trigger (ECT).

- h2f_cti
- h2f_cti_clock

Debug APB Interface

The debug Advanced Peripheral Bus (APB™) interface allows debug components in the FPGA fabric to debug components on the CoreSight™ debug APB.

- h2f_debug_apb
- h2f_debug_apb_sideband
- h2f_debug_apb_reset
- h2f_debug_apb_clock

Peripheral Signal Interfaces

DMA Controller Peripheral Request Interfaces

The DMA controller interface allows soft IP in the FPGA fabric to communicate with the DMA controller in the HPS. You can configure up to eight separate interface channels.

- f2h_dma_req0—FPGA DMA controller peripheral request interface 0
- f2h_dma_req1—FPGA DMA controller peripheral request interface 1
- f2h_dma_req2—FPGA DMA controller peripheral request interface 2
- f2h_dma_req3—FPGA DMA controller peripheral request interface 3
- f2h_dma_req4—FPGA DMA controller peripheral request interface 4
- f2h_dma_req5—FPGA DMA controller peripheral request interface 5
- f2h_dma_req6—FPGA DMA controller peripheral request interface 6
- f2h_dma_req7—FPGA DMA controller peripheral request interface 7



For more information, refer to the *DMA Controller* chapter in volume 3 of the *Cyclone V Device Handbook*.

Other Interfaces

MPU Standby and Event Interfaces

MPU standby and event signals are notification signals to the FPGA fabric that the MPU is in standby. Event signals are used to wake up the Cortex-A9 processors from a wait for event (WFE) state. The standby and event signals are included in the following interfaces:

- `h2f_mpu_events`—MPU standby and event interface, including the following signals:
 - `h2f_mpu_eventi`—Sends an event from logic in the FPGA fabric to the MPU. This FPGA-to-HPS signal is used to wake up a processor that is in a Wait For Event state. Asserting this signal has the same effect as executing the SEV instruction in the Cortex-A9. This signal must be de-asserted until the FPGA fabric is powered-up and configured.
 - `h2f_mpu_evento`—Sends an event from the MPU to logic in the FPGA fabric. This HPS-to-FPGA signal is asserted when an SEV instruction is executed by one of the Cortex-A9 processors.
 - `h2f_mpu_standbywfe[1:0]`—Indicates whether each Cortex-A9 processor is in the WFE state
 - `h2f_mpu_standbywfi[1:0]`—Indicates whether each Cortex-A9 processor is in the wait for interrupt (WFI) state
- `h2f_mpu_gp`—General purpose interface

The MPU provides signals to indicate when it is in a standby state. These signals are available to custom hardware designs in the FPGA fabric.



For more information, refer to the *Cortex-A9 Microprocessor Unit Subsystem* chapter in volume 3 of the *Cyclone V Device Handbook*.

FPGA-to-HPS Interrupts

You can configure the HPS component to provide 64 general-purpose FPGA-to-HPS interrupts, allowing soft IP in the FPGA fabric to trigger interrupts to the MPU's generic interrupt controller (GIC). The interrupts are implemented through the following 32-bit interfaces:

- `f2h_irq0`—FPGA-to-HPS interrupts 0 through 31
- `f2h_irq1`—FPGA-to-HPS interrupts 32 through 63

The FPGA-to-HPS interrupts are asynchronous on the FPGA interface. Inside the HPS, the interrupts are synchronized to the MPU's internal peripheral clock (`periphclk`).

General-Purpose Interfaces

You can use the FPGA manager to supply the `h2f_mpu_gp` interface, which includes the following general-purpose signals:

- 32 FPGA-to-HPS signals
- 32 HPS-to-FPGA signals



For more information, refer to the *FPGA Manager* chapter in volume 3 of the *Cyclone V Device Handbook*.

Document Revision History

Table 28-4 shows the revision history for this document.

Table 28-4. Document Revision History

Date	Version	Changes
November 2012	1.1	<ul style="list-style-type: none"> ■ Added debug interfaces. ■ Updated HPS-to-FPGA reset interface names. ■ Updated HPS external reset source interface names. ■ Removed DMA peripheral interface clocks. ■ Referred to Altera Address Span Extender.
June 2012	1.0	Initial release.
May 2012	0.1	Preliminary draft.

HPS Simulation Support

This section describes the simulation support for the hard processor system (HPS) component. The HPS simulation models support interfaces between the HPS and FPGA fabric, including:

- Bus functional models (BFMs) for most interfaces between HPS and FPGA fabric
- A simulation model for the HPS SDRAM memory

The HPS simulation support does not include modules implemented in the HPS, such as the ARM® Cortex™-A9 MPCore processor.

You specify simulation support files when you instantiate the HPS component in the Qsys system integration tool. When you enable a particular HPS-FPGA interface, Qsys provides the corresponding model during the generation process. Refer to “Simulation Flows” on page 29–10 for a description of the simulation flows.



For general information about instantiating the component, refer to the *Instantiating the HPS Component* chapter in volume 3 of the *Cyclone® V Device Handbook*.

The HPS simulation support enables you to develop and verify your own FPGA soft logic or intellectual property (IP) that interfaces to the HPS component.

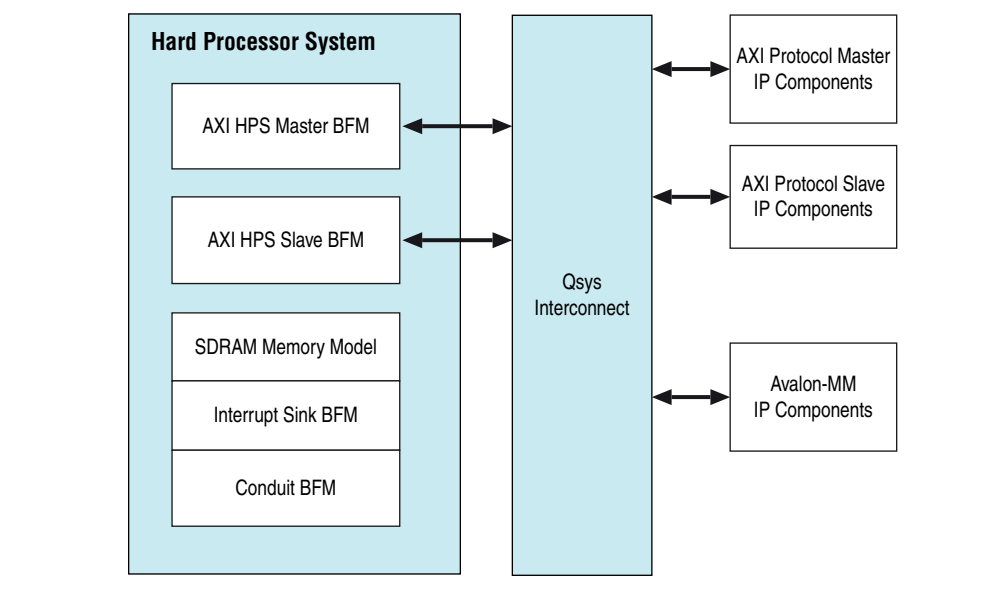
The simulation model supports the following interfaces:

- Clock and reset interfaces
- FPGA-to-HPS Advanced Microcontroller Bus Architecture (AMBA®) Advanced eXtensible Interface (AXI™) slave interface
- HPS-to-FPGA AXI master interface
- Lightweight HPS-to-FPGA AXI master interface
- FPGA-to-HPS SDRAM interface
- Microprocessor unit (MPU) general-purpose I/O interface
- MPU standby and event interface
- Interrupts interface
- Direct memory access (DMA) controller peripheral request interface
- Debug Advanced Peripheral Bus (APB™) interface
- System Trace Macrocell (STM) hardware event

- FPGA cross trigger interface
- FPGA trace port interface

Figure 29-1 on page 29-2 shows the HPS with its BFM.

Figure 29-1. HPS BFM Block Diagram



The HPS BFM uses standard function calls from the Altera® BFM application programming interface (API), as detailed in the remainder of this section.

For information about the BFM API, refer to the *Avalon Verification IP Suite User Guide* and the *Mentor Verification IP Altera Edition User Guide*.

HPS simulation supports only Verilog HDL or SystemVerilog simulation environments.

Clock and Reset Interfaces

For general information about clock and reset interfaces, refer to “Memory-Mapped Interfaces” in the *HPS Component Interfaces* chapter in volume 3 of the *Cyclone V Device Handbook*.

Clock Interface

Qsys generates the clock source BFM for each clock output interface from the HPS component. For HPS-to-FPGA user clocks, specify the BFM clock rate in the **User clock frequency** field in the **HPS Clocks** page when instantiating the HPS component in Qsys.

The HPS-to-FPGA trace port interface unit generates a clock output to the FPGA, named `h2f_tpiu_clock`. In simulation, the clock source BFM also represents this clock output's behavior.

Table 29–1 lists all HPS clock output interfaces with the BFM instance name.

Table 29–1. HPS Clock Output Interface Simulation Model

Interface Name	BFM Instance Name
h2f_user0_clock	h2f_user0_clock
h2f_user1_clock	h2f_user1_clock
h2f_user2_clock	h2f_user2_clock
h2f_tpiu_clock	h2f_tpiu_clock

The Altera clock source BFM application programming interface (API) applies to all the BFM instance names listed in Table 29–1. Your Verilog interfaces use the same API, passing in different instance names.

Qsys does not generate BFM instance names for FPGA-to-HPS clock input interfaces.

Reset Interface

The HPS reset request and handshake interfaces are connected to Altera conduit BFM instance names for simulation. Table 29–2 lists the name of each interface. You can monitor the reset request interface state changes or set the interface by using the API listed in Table 29–2.

Table 29–2. HPS Reset Input Interface Simulation Model

Interface Name	BFM Instance Name	API Function Names
f2h_cold_reset_req	f2h_cold_reset_req	get_f2h_cold_rst_req_n()
f2h_debug_reset_req	f2h_debug_reset_req	get_f2h_dbg_rst_req_n()
f2h_warm_reset_req	f2h_warm_reset_req	get_f2h_warm_rst_req_n()
h2f_warm_reset_handshake	h2f_warm_reset_handshake	set_h2f_pending_rst_req_n() get_f2h_pending_rst_ack_n()

Table 29–3 lists all HPS reset output interfaces with the BFM instance name. The Altera reset source BFM application programming interface applies to all the BFM instance names listed in Table 29–3.

Table 29–3. HPS Reset Output Interface Simulation Model

Interface Name	BFM Instance Name
h2f_reset	h2f_reset
h2f_cold_reset	h2f_cold_reset
h2f_debug_apb_reset	h2f_debug_apb_reset

The HPS reset output interface is connected to a reset source BFM. Qsys configures the BFM as shown in [Table 29-4](#).

Table 29-4. Configuration of Reset Source BFM for HPS Reset Output Interface

Parameter	BFM Value ⁽¹⁾	Meaning
Assert reset high	Off	This parameter is off, specifying an active-low reset signal from the BFM.
Cycles of initial reset	0	This parameter is 0, specifying that the BFM does not assert the reset signal automatically.
Note to Table 29-4: (1) The parameter value of the instantiated BFM as configured for HPS simulation		

FPGA-to-HPS AXI Slave Interface

The FPGA-to-HPS AXI slave interface, `f2h_axi_slave`, is connected to a Mentor Graphics AXI slave BFM for simulation. Qsys configures the BFM as shown in [Table 29-5](#). The BFM clock input is connected to `f2h_axi_clock` clock.

Table 29-5. Configuration of FPGA-to-HPS AXI Slave BFM

Parameter	Value
AXI Address Width	32
AXI Read Data Width	32, 64, 128
AXI Write Data Width	32, 64, 128
AXI ID Width	8

You control and monitor the AXI slave BFM by using the BFM API.



For more information, refer to the [Mentor Verification IP Altera Edition User Guide](#). For general information about FPGA-to-HPS AXI slave interfaces, refer to “Memory-Mapped Interfaces” in the [HPS Component Interfaces](#) chapter in volume 3 of the *Cyclone V Device Handbook*.

HPS-to-FPGA AXI Master Interface

The HPS-to-FPGA AXI master interface, `h2f_axi_master`, is connected to a Mentor Graphics AXI master BFM for simulation. Qsys configures the BFM as shown in [Table 29-6](#). The BFM clock input is connected to `h2f_axi_clock` clock.

Table 29-6. Configuration of HPS-to-FPGA AXI Master BFM

Parameter	Value
AXI Address Width	30
AXI Read and Write Data Width	32, 64, 128
AXI ID Width	12

You control and monitor the AXI master BFM by using the BFM API.

- For more information, refer to the *Mentor Verification IP Altera Edition User Guide*. For general information about HPS-to-FPGA AXI master interfaces, refer to “Memory-Mapped Interfaces” in the *HPS Component Interfaces* chapter in volume 3 of the *Cyclone V Device Handbook*.

Lightweight HPS-to-FPGA AXI Master Interface

The lightweight HPS-to-FPGA AXI master interface, `h2f_lw_axi_master`, is connected to a Mentor Graphics AXI master BFM for simulation. Qsys configures the BFM as shown in Table 29-7. The BFM clock input is connected to `h2f_lw_axi_clock` clock.

Table 29-7. Configuration of Lightweight HPS-to-FPGA AXI Master BFM

Parameter	Value
AXI Address Width	21
AXI Read and Write Data Width	32
AXI ID Width	12

You control and monitor the AXI master BFM by using the BFM API.

- For more information, refer to the *Mentor Verification IP Altera Edition User Guide*. For general information about lightweight HPS-to-FPGA AXI master interfaces, refer to “Memory-Mapped Interfaces” in the *HPS Component Interfaces* chapter in volume 3 of the *Cyclone V Device Handbook*.

FPGA-to-HPS SDRAM Interface

The HPS component contains a memory interface simulation model to which all of the FPGA-to-HPS SDRAM interfaces are connected. The model is based on the HPS implementation and provides cycle-level accuracy, reflecting the true bandwidth and latency of the interface. However, the model does not have the detailed configuration provided by the HPS software, and hence does not reflect any inter-port scheduling that might occur under contention on the real hardware when different priorities or weights are used.

- For more information, refer to “EMI-Related HPS Features in SoC Devices” in the *Functional Description—Hard Memory Interface* chapter in volume 3 of the *External Memory Interface Handbook*.

HPS-to-FPGA MPU General-Purpose I/O Interface

The HPS-to-FPGA MPU general-purpose I/O interface is connected to an Altera conduit BFM for simulation. Table 29-8 lists the name of each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API listed in Table 29-8.

Table 29-8. HPS-to-FPGA MPU General-Purpose I/O Interface Simulation Model

Interface Name	BFM Instance Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
h2f_mpu_gp	h2f_mpu_gp	set_h2f_mpu_gp_out() get_h2f_mpu_gp_in()	set_gp_out() get_gp_in()

HPS-to-FPGA MPU Event Interface

The HPS-to-FPGA MPU event interface is connected to an Altera conduit BFM for simulation. Table 29-9 lists the name of each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API listed in Table 29-9.

Table 29-9. HPS-to-FPGA MPU Event Interface Simulation Model

Interface Name	BFM Instance Name	RTL Simulation API Function Names ⁽¹⁾	Post-Fit Simulation API Function Names
h2f_mpu_events	h2f_mpu_events	get_h2f_mpu_eventi() set_h2f_mpu_evento() set_h2f_mpu_standbywfe() set_h2f_mpu_standbywfi()	get_eventi() set_evento() set_standbywfe() set_standbywfi()

Note to Table 29-9:

(1) The usage of conduit get_*() and set_*() API functions is the same as with the general Avalon conduit BFM.

FPGA-to-HPS Interrupts Interface

The FPGA-to-HPS interrupts interface is connected to an Altera Avalon interrupt sink BFM for simulation. Table 29-10 lists the name of each interface.

Table 29-10. FPGA-to-HPS Interrupts Interface Simulation Model

Interface Name	BFM Instance Name
f2h_irq0	f2h_irq0
f2h_irq1	f2h_irq1

The Altera Avalon interrupt sink BFM API applies to all the BFMs listed in Table 29-3.

HPS-to-FPGA Debug APB Interface

The HPS-to-FPGA debug APB interface is connected to an Altera conduit BFM for simulation. Table 29–11 lists the name of each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API functions listed in Table 29–11.

Table 29–11. HPS-to-FPGA Debug APB Interface Simulation Model

Interface Name	BFM Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
h2f_debug_apb	h2f_debug_apb	set_h2f_dbg_apb_PADDR() set_h2f_dbg_apb_PADDR_31() set_h2f_dbg_apb_PENABLE() get_h2f_dbg_apb_PRDATA() get_h2f_dbg_apb_PREADY() set_h2f_dbg_apb_PSEL() get_h2f_dbg_apb_PSLVERR() set_h2f_dbg_apb_PWDATA() set_h2f_dbg_apb_PWRITE()	set_PADDR() set_PADDR_31() set_PENABLE() get_PRDATA() get_PREADY() set_PSEL() get_PSLVERR() set_PWDATA() set_PWRITE()
h2f_debug_apb_side band	h2f_debug_apb_side band	get_h2f_dbg_apb_PCLKEN() get_h2f_dbg_apb_DBG_APB_DISABLE()	get_PCLKEN() get_DBG_APB_DISABLE()

FPGA-to-HPS System Trace Macrocell (STM) Hardware Event Interface

The FPGA-to-HPS STM hardware event interface is connected to an Altera conduit BFM for simulation. Table 29–12 lists the name of each interface, along with API function name for each type of simulation. You can monitor the interface state changes or set the interface by using the API functions listed in Table 29–12.

Table 29–12. FPGA-to-HPS STM Hardware Event Interface Simulation Model

Interface Name	BFM Name	RTL Simulation API Function Name	Post-Fit Simulation API Function Name
f2h_stm_hw_events	f2h_stm_hw_events	get_f2h_stm_hwevents()	get_stm_events()

HPS-to-FPGA Cross-Trigger Interface

The HPS-to-FPGA cross-trigger interface is connected to an Altera conduit BFM for simulation. Table 29-13 lists the name of each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API functions listed in Table 29-13.

Table 29-13. HPS-to-FPGA Cross-Trigger Interface Simulation Model

Interface Name	BFM Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
h2f_cti	h2f_cti	get_h2f_cti_trig_in() set_h2f_cti_trig_in_ack() set_h2f_cti_trig_out() get_h2f_cti_trig_out_ack() set_h2f_cti_asicctl() get_h2f_cti_fpga_clk_en()	get_trig_in() set_trig_inack() set_trig_out() get_trig_outack() set_asicctl() get_clk_en()

HPS-to-FPGA Trace Port Interface

The HPS-to-FPGA trace port interface is connected to an Altera conduit BFM for simulation. Table 29-14 lists the name of each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API functions listed in Table 29-14.

Table 29-14. HPS-to-FPGA Trace Port Interface Simulation Model

Interface Name	BFM Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
h2f_tpiu	h2f_tpiu	get_h2f_tpiu_clk_ctl() set_h2f_tpiu_data()	get_traceclk_ctl() set_trace_data()

FPGA-to-HPS DMA Handshake Interface

The FPGA-to-HPS DMA handshake interface is connected to an Altera conduit BFM for simulation. Table 29-15 lists the name for each interface, along with API function names for each type of simulation. You can monitor the interface state changes or set the interface by using the API listed in Table 29-15.

Table 29-15. FPGA-to-HPS DMA Handshake Interface Simulation Model

Interface Name	BFM Instance Name	RTL Simulation API Function Names ⁽¹⁾	Post-Fit Simulation API Function Names
f2h_dma_req0	f2h_dma_req0	get_f2h_dma_req0_req() get_f2h_dma_req0_single() set_f2h_dma_req0_ack()	get_channel0_req() get_channel0_single() set_channel0_xx_ack()
f2h_dma_req1	f2h_dma_req1	get_f2h_dma_req1_req() get_f2h_dma_req1_single() set_f2h_dma_req1_ack()	get_channel1_req() get_channel1_single() set_channel1_xx_ack()
f2h_dma_req2	f2h_dma_req2	get_f2h_dma_req2_req() get_f2h_dma_req2_single() set_f2h_dma_req2_ack()	get_channel2_req() get_channel2_single() set_channel2_xx_ack()
f2h_dma_req3	f2h_dma_req3	get_f2h_dma_req3_req() get_f2h_dma_req3_single() set_f2h_dma_req3_ack()	get_channel3_req() get_channel3_single() set_channel3_xx_ack()
f2h_dma_req4	f2h_dma_req4	get_f2h_dma_req4_req() get_f2h_dma_req4_single() set_f2h_dma_req4_ack()	get_channel4_req() get_channel4_single() set_channel4_xx_ack()
f2h_dma_req5	f2h_dma_req5	get_f2h_dma_req5_req() get_f2h_dma_req5_single() set_f2h_dma_req5_ack()	get_channel5_req() get_channel5_single() set_channel5_xx_ack()
f2h_dma_req6	f2h_dma_req6	get_f2h_dma_req6_req() get_f2h_dma_req6_single() set_f2h_dma_req6_ack()	get_channel6_req() get_channel6_single() set_channel6_xx_ack()
f2h_dma_req7	f2h_dma_req7	get_f2h_dma_req7_req() get_f2h_dma_req7_single() set_f2h_dma_req7_ack()	get_channel7_req() get_channel7_single() set_channel7_xx_ack()
Note to Table 29-15: (1) The usage of conduit get_*() and set_*() API functions is the same as with the general Avalon conduit BFM.			

Simulation Flows

This section describes the simulation flows for an HPS-based design.

Altera provides both functional register transfer level (RTL) simulation and post-fitter gate-level simulation flows. The simulation flows involve the following major steps:

1. Instantiate the HPS component—Refer to [“Specifying HPS Simulation Model in Qsys”](#) on page 29–10.
2. Generate the system in Qsys, including the simulation model—Refer to [“Generating HPS Simulation Model in Qsys”](#) on page 29–13.
3. Run the simulation—Refer to one of the following sections:
 - [“Running HPS RTL Simulation”](#) on page 29–13
 - [“Running HPS Post-Fit Simulation”](#) on page 29–14



For general information about simulation, refer to the [Simulating Altera Designs](#) chapter in volume 3 of the *Quartus II Handbook*.

Specifying HPS Simulation Model in Qsys

The following steps outline how to set up the HPS component for simulation.

1. Add the HPS component from the Qsys Component Library.
2. Configure the component based on your application needs by selecting or deselecting the HPS-FPGA interfaces.
3. Connect the appropriate HPS interfaces to other components in the system. For example, connect the FPGA-to-HPS AXI slave interface to an AXI master interface in another component in the system.

When you create your component, make sure the conduit interfaces have the correct role names, directions, and widths. [Table 29–16](#) lists the role names, directions, and widths for all the HPS conduit interfaces.



For general information about adding the HPS component to your design, refer to the [Instantiating the HPS Component](#) chapter in volume 3 of the *Cyclone V Device Handbook*.

Table 29–16. HPS Conduit Interfaces (Part 1 of 3)

Role Name	Direction	Width
h2f_warm_reset_handshake		
h2f_pending_rst_req_n	Output	1
f2h_pending_rst_ack_n	Input	1
h2f_mpu_gp		
gp_in	Input	32
gp_out	Output	32

Table 29-16. HPS Conduit Interfaces (Part 2 of 3)

Role Name	Direction	Width
h2f_mpu_events		
eventi	Input	1
evento	Output	1
standbywfe	Output	2
standbywfi	Output	2
f2h_dma_req0		
req0_req	Input	1
req0_single	Input	1
req0_ack	Output	1
f2h_dma_req1		
req1_req	Input	1
req1_single	Input	1
req1_ack	Output	1
f2h_dma_req2		
req2_req	Input	1
req2_single	Input	1
req2_ack	Output	1
f2h_dma_req3		
req3_req	Input	1
req3_single	Input	1
req3_ack	Output	1
f2h_dma_req4		
req4_req	Input	1
req4_single	Input	1
req4_ack	Output	1
f2h_dma_req5		
req5_req	Input	1
req5_single	Input	1
req5_ack	Output	1
f2h_dma_req6		
req6_req	Input	1
req6_single	Input	1
req6_ack	Output	1
f2h_dma_req7		
req7_req	Input	1
req7_single	Input	1
req7_ack	Output	1

Table 29-16. HPS Conduit Interfaces (Part 3 of 3)

Role Name	Direction	Width
h2f_debug_apb		
paddr	Input	18
paddr_31	Input	1
penable	Input	1
prdata	Output	32
pready	Output	1
psel	Input	1
pslverr	Output	1
pwdata	Input	32
pwrite	Input	1
h2f_debug_apb_sideband		
pclken	Output	1
dbg_apb_disable	Output	1
f2h_stm_hw_events		
stm_hwevents	Output	28
h2f_cti		
trig_in	Output	8
trig_in_ack	Input	8
trig_out	Input	8
trig_out_ack	Output	8
asicctl	Input	8
fpga_clk_en	Output	1
h2f_tpiu		
clk_ctl	Output	1
data	Input	32

Generating HPS Simulation Model in Qsys

The following steps outline how to generate the simulation model.

1. Go to the **Generation** page in Qsys.
2. For RTL simulation, perform the following steps:
 - a. Set **Create simulation model to Verilog**.
 - b. Click **Generate**.



HPS simulation does not support the VHDL simulation environment.

For post-fit simulation, perform the following steps:

- a. Turn on the **Create HDL design files for synthesis** option.
- b. Turn on the **Create block symbol file (.bsf)** option.
- c. Click **Generate**.



For general information about generating the HPS component, refer to the *Instantiating the HPS Component* chapter in volume 3 of the *Cyclone V Device Handbook*. For more information about Qsys simulation, refer to “Simulating a Qsys System” in the *Creating a System with Qsys* chapter in volume 1 of the *Quartus II Handbook*.

Running HPS RTL Simulation

Qsys generates scripts for various simulators that you use to complete the simulation process. Table 29-17 lists simulation tools, script names and the script directories for various vendors.

Table 29-17. Qsys-Generated Scripts for Various Simulators

Simulator	Script Name	Directory
Mentor Graphics Modelsim® Altera Edition	msim_setup.tcl	<project directory>/<Qsys design name>/simulation/mentor
Cadence NC-Sim	ncsim_setup.sh	<project directory>/<Qsys design name>/simulation/cadence
Synopsys VCS	vcs_setup.sh	<project directory>/<Qsys design name>/simulation/synopsys/vcs
Synopsys VCS-MX	vcsmx_setup.sh	<project directory>/<Qsys design name>/simulation/synopsys/vcsmx



For detailed simulation steps, refer to the *Mentor Verification IP Altera Edition User Guide*, and to the *Qsys Tutorial* chapter of the *Avalon Verification IP Suite User Guide*.

Running HPS Post-Fit Simulation

The section describes how you run HPS post-fit simulation. After successful Qsys generation, perform the following steps:

1. Add the Qsys-generated synthesis file set to your Quartus II project by performing the following steps:
 - a. In the Quartus II software, click **Settings** in the Assignments menu.
 - b. In the **Settings - <your Qsys system name>** dialog box, on the **Files** tab, browse to <your project directory>/<your Qsys system name>/**synthesis/** and select <your Qsys system name>.**qip**.
 - c. Click **OK**.
 2. You can instantiate the Qsys system that contains HPS component as your Quartus II project top-level entity, if necessary.
 3. Compile the design by clicking **Start Compilation** in the Processing menu.
 4. Change the EDA Netlist Writer settings, if necessary, by performing the following steps:
 - a. Click **Settings** in the Assignment menu.
 - b. On the **Simulation** tab, under the **EDA Tool Settings** tab, you can specify the following EDA Netlist Writer settings:
 - **Tool name**—The name of the simulation tool
 - **Format for output netlist**
 - **Output directory**
 - c. Click **OK**.
- ❓ For more information about EDA Netlist Writer settings, refer to *Simulation Page (Settings Dialog Box)* in Quartus II Help.
5. To create the post-fitter simulation model with Quartus II EDA Netlist Writer, in the Start menu, point to **Processing** and click **Start EDA Netlist Writer**.
 6. Compile the necessary simulation files in your simulation tool. [Table 29-18](#) lists the required libraries and files.
 7. Start simulation.

Table 29-18. Post-Fit Simulation Files

Library	Directory ⁽¹⁾	File
Altera Verification IP Library	<Avalon Verification IP>/lib/	verbosity_pkg.sv avalon_mm_pkg.sv avalon_utilities_pkg.sv
Avalon Clock Source BFM	<Avalon Verification IP>/altera_avalon_clock_source/	altera_avalon_clock_source.sv
Avalon Reset Source BFM	<Avalon Verification IP>/altera_avalon_reset_source/	altera_avalon_reset_source.sv
Avalon MM Slave BFM	<Avalon Verification IP>/altera_avalon_mm_slave_bfm/	altera_avalon_mm_slave_bfm.sv
Avalon Interrupt Sink BFM	<Avalon Verification IP>/altera_avalon_interrupt_sink/	altera_avalon_interrupt_sink.sv
Mentor AXI Verification IP Library	<AXI Verification IP>/common/	questa_mvc_svapi.svh
Mentor AXI3 BFM	<AXI Verification IP>/axi3/bfm/	mgc_common_axi.sv mgc_axi_master.sv mgc_axi_slave.sv
HPS Post-Fit Simulation Library	<HPS Post-fit Sim>/	All the files in the directory
Device Simulation Library ⁽²⁾	<Device Sim Lib>/	altera_primitives.v 220model.v sgate.v altera_mf.v altera_Insim.sv cyclonev_atoms.v arriav_atoms.v mentor/cyclonev_atoms_ncrypt.v mentor/arriav_atoms_ncrypt.v
EDA Netlist Writer Generated Post-Fit Simulation Model	<User project directory>/	*.vo *.vho ⁽³⁾
User testbench files	<User project directory>/	*.v *.sv *.vhd ⁽³⁾

Notes to Table 29-18:

- (1) <ACDS install> = Altera Complete Design Suite installation path
 <Avalon Verification IP> = <ACDS install>/ip/altera/sopc_builder_ip/verification
 <AXI Verification IP> = <ACDS install>/ip/altera/mentor_vip_ae
 <HPS Post-fit Sim> = <ACDS install>/ip/altera/hps/postfitter_simulation
 <Device Sim Lib> = <ACDS install>/quartus/eda/sim_lib
- (2) The device simulation library is not needed with Modelsim-Altera.
- (3) Mixed-language simulator is needed for Verilog HDL and VHDL mixed design

For post-fit simulation, you must call the BFM API in your test program with a specific hierarchy. The hierarchy format is:

```
<DUT>.\<HPS>|fpga_interfaces|<interface><space>.<BFM>.<API function>
```

Where:

- <DUT> is the instance name of the design under test that you instantiated in your test bench that consists of the HPS component.
- <HPS> is the HPS component instance name that you use in your Qsys system.
- <interface> is the instance name for a specific FPGA-to-HPS or HPS-to-FPGA interface. This name can be found in the **fpga_interfaces.sv** file located in <project directory>/<Qsys design name>/synthesis/submodules.
- <space>—You must insert one space character after the interface instance name.
- <BFM> is the BFM instance name. In <ACDS install>/ip/altera/hps/postfitter_simulation, identify the SystemVerilog file corresponding to the interface type that you are using. The SystemVerilog file contains the BFM instance name.

For example, a path for the Lightweight HPS-to-FPGA master interface hierarchy could be formed as follows:

```
top.dut.\my_hps_component|fpga_interface|hps2fpga_light_weight .h2f_lw_axi_master
```

Notice the space after “hps2fpga_light_weight”. Omitting this space would cause simulation failure because the instance name “hps2fpga_light_weight ”, including the space, is the name used in the post-fit simulation model generated by the Quartus® II software.

Document Revision History

Table 29-19 lists the revision history for this document.

Table 29-19. Document Revision History

Date	Version	Changes
November 2012	1.1	<ul style="list-style-type: none"> ■ Added debug APB, STM hardware event, FPGA cross trigger, FPGA trace port interfaces. ■ Added support for post-fit simulation. ■ Updated some API function names. ■ Removed DMA peripheral clock.
June 2012	1.0	Initial release.
May 2012	0.1	Preliminary draft.