Cyclone V Hard Processor System User Guide

Subscribe

Send Feedback

UG-1120
2014.06.30

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Contents

Introduction to the HPS Component 1

2014.06.30

cv_54026 Subscribe Send Feedback

The hard processor system (HPS) component is a soft component that you can instantiate in the FPGA fabric of the Cyclone® V SoC. 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*.

The HPS supports the following peripheral architectures and features. The chapters that describe these features can be found on the Cyclone V documentation page.

- Clock manager
- Reset manager
- Interconnect
- HPS-FPGA AXI Bridge
- Cortex™-A9 Microprocessor Unit Subsystem
- CoreSight™ Debug and Trace
- SDRAM Controller Subsystem
- On-Chip RAM and boot block ROM
- NAND Flash Controller
- SD/MMC Controller
- Quad SPI Flash Controller
- FPGA Manager Block Diagram and System Integration
- System Manager
- Scan Manager
- DMA Controller
- Ethernet Media Access Controller
- USB OTG Controller
- SPI Controller
- I^{2}C Controller
- UART Controller
- General-Purpose I/O Interface
- Timer

- Watchdog Timer

Related Information

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

Document Revision History

Table 1-1: Document Revision History

Date	Version	Changes
June 2014	2014.06.30	Maintenance release
February 2014	2014.02.28	Maintenance release
December 2013	2013.12.30	Maintenance release
June 2012	1.0	Maintenance release.
May 2012	0.1	Preliminary draft.

Instantiating the HPS Component 2

2014.06.30

cv_54027 Subscribe Send Feedback

You instantiate the hard processor system (HPS) component in Qsys. The HPS is available in the Qsys IP catalog under **Processor and Peripherals** > **Hard Processor Systems**. This chapter describes the parameters available in the HPS component parameter editor, which opens when you add or edit an HPS component.

Related Information

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

FPGA Interfaces

The **FPGA Interfaces** tab is one of four tabs on the HPS Component. This tab contains several groups with the following parameters:

General Interfaces on page 2-1

AXI Bridges on page 2-3

FPGA-to-HPS SDRAM Interface on page 2-4

Reset Interfaces on page 2-5

DMA Peripheral Request on page 2-6

Interrupts on page 2-6

Related Information

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

General Interfaces

When enabled, the interfaces described in the following table become visible in the HPS component.

Table 2-1: General Parameters

Parameter Name	Parameter Description	Interface Name
Enable MPU standby and event signals	Enables interfaces that perform the following functions: • 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 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_gp`

Enable Debug APB interface	Enables debug interface to the FPGA, allowing access to debug components in the HPS. For more information, refer to the *CoreSight Debug and Trace* chapter.	`h2f_debug_apb`
		`h2f_debug_apb_sideband`
		`h2f_debug_apb_clock`
		`h2f_debug_apb_reset`

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. For more information, refer to the *CoreSight Debug and Trace* chapter.	`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). For more information, refer to the *CoreSight Debug and Trace* chapter.	`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. For more information, refer to the *CoreSight Debug and Trace* chapter.	`h2f_tpiu`
		`h2f_tpiu_clock_in`
		`h2f_tpiu_clock`

Parameter Name	Parameter Description	Interface Name
Enable boot from FPGA signals	Enable an input to the HPS indicating whether a preloader is available in on-chip memory of FPGA. This option also enables a separate input to the HPS indicating a fallback preloader is available in the FPGA memory. A fallback preloader is used when there is no valid preloader image found in flash memory. For more information, refer to *Appendix A: Booting and Configuration*.	`f2h_boot_from_fpga`
Enable HLGPI Interface	Enable a general purpose interface that is connected to the General Purpose I/O (GPIO) peripheral of HPS. This is an input-only interface with 14-bit width. This interface shares the I/O pins with the HPS DDR SDRAM controller.	`hps_io (hps_io_gpio_inst_HLGPI[0..13])`

Related Information

- **CoreSight Debug and Trace**
  Trace port interface unit (TPIU). Enabling the TPIU exposes trace signals to the device pins. Refer to the *CoreSight Debug and Trace* chapter in the *Cyclone V Device Handbook, Volume 3* for more information.
- **Booting and Configuration**
  For detailed information about the HPS boot sequence, refer to the *Booting and Configuration* appendix in the *Cyclone V Device Handbook, Volume 3.*

AXI Bridges

Table 2-2: 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` `f2h_axi_clock`
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` `h2f_axi_clock`
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` `h2f_lw_axi_clock`

Note: To facilitate accessing these slaves from a memory-mapped master with a smaller address width, you can use the Altera® Address Span Extender.

Related Information

- **Using the Address Span Extender Component** on page 2-15
  Address span extender details
- **Interconnect**
  For more information, refer to the *Interconnect* chapter in the *Cyclone V Device Handbook, Volume 3*.

FPGA-to-HPS SDRAM Interface

In the **FPGA-to-HPS SDRAM Interface** table, use + or – to add or remove FPGA-to-HPS SDRAM interfaces.

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

Table 2-3: FPGA-to-HPS SDRAM Port and Interface Names

Port Name	Interface Name
f2h_sdram0	`f2h_sdram0_data` `f2h_sdram0_clock`
f2h_sdram1	`f2h_sdram1_data` `f2h_sdram1_clock`
f2h_sdram2	`f2h_sdram2_data` `f2h_sdram2_clock`
f2h_sdram3	`f2h_sdram3_data` `f2h_sdram3_clock`
f2h_sdram4	`f2h_sdram4_data` `f2h_sdram4_clock`
f2h_sdram5	`f2h_sdram5_data` `f2h_sdram5_clock`

The following table shows the parameters available for each SDRAM interface where the **Name** parameter denotes the interface name.

Table 2-4: FPGA-to-HPS SDRAM Interface Parameters

Parameter Name	Parameter Description
Name	Port name (auto assigned as shown in **Table 2-3** table below)

Parameter Name	Parameter Description
Type	Interface type: • AXI-3 • Avalon-MM Bidirectional • Avalon-MM Write-only • Avalon-MM Read-only
Width	32, 64, 128, or 256

Note: 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*.

Related Information

- **Using the Address Span Extender Component** on page 2-15
  Address span extender details
- **SDRAM Controller Subsystem**
  For more information, refer to the *SDRAM Controller Subsystem* chapter in the *Cyclone V Device Handbook, Volume 3*.

Reset Interfaces

You can enable most resets on an individual basis.

Table 2-5: 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`

Related Information

Reset Manager

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

DMA Peripheral Request

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.

Related Information

DMA Controller

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

Interrupts

Table 2-6: FPGA-to-HPS Interrupts Interface

Parameter Name	Parameter Description	Interface Name
Enable FPGA-to-HPS Interrupts	Enables interface for FPGA interrupt signals to the MPU (in the HPS).	f2h_irq0 f2h_irq1

You can enable the interfaces for each individual HPS peripheral interrupt to the FPGA. Table 2-7 shows the available HPS to FPGA interrupt interfaces and the corresponding parameters to enable them.

Table 2-7: HPS-to-FPGA Interrupts Interface

Parameter Name	Parameter Description	Interface Name
Enable CAN interrupts	Enables interface for HPS CAN controllers interrupt to the FPGA	`h2f_can0_interrupt` `h2f_can1_interrupt`
Enable clock peripheral interrupts	Enables interface for HPS clock manager and MPU wake-up interrupt signals to the FPGA	`h2f_clkmgr_interrupt` `h2f_mpuwakeup_interrupt`
Enable CTI interrupts	Enables interface for HPS cross-trigger interrupt signals to the FPGA	`h2f_cti_interrupt0` `h2f_cti_interrupt1`

Parameter Name	Parameter Description	Interface Name

Enable DMA interrupts	Enables interface for HPS DMA channels interrupt and DMA abort interrupt to the FPGA	`h2f_dma_interrupt0`
		`h2f_dma_interrupt1`
		`h2f_dma_interrupt2`
		`h2f_dma_interrupt3`
		`h2f_dma_interrupt4`
		`h2f_dma_interrupt5`
		`h2f_dma_interrupt6`
		`h2f_dma_interrupt7`
		`h2f_dma_abort_interrupt`

Enable EMAC interrupts	Enables interface for HPS Ethernet MAC controller interrupt to the FPGA	`h2f_emac0_interrupt` `h2f_emac1_interrupt`
Enable FPGA manager interrupts	Enables interface for the HPS FPGA manager interrupt to the FPGA	`h2f_fpga_man_interrupt`

Enable GPIO interrupts	Enables interface for the HPS general purpose IO (GPIO) interrupt to the FPGA	`h2f_gpio0_interrupt`
		`h2f_gpio1_interrupt`
		`h2f_gpio2_interrupt`

Enable I^{2}C-EMAC interrupts (for I2C2 and I2C3)	Enables interface for the HPS I^{2}C interrupt to the FPGA (I^{2}C used for EMAC media interface control)	`h2f_i2c_emac0_interrupt` `h2f_i2c_emac1_interrupt`

Enable I^{2}C peripheral interrupts (for I2C0 and I2C1)	Enables interface for the HPS I^{2}C interrupt to the FPGA (I^{2}C used for general purpose)	`h2f_i2c0 _interrupt`
		`h2f_i2c1_interrupt`

Enable L4 timer interrupts	Enables interface for the HPS SP timer interrupt to the FPGA. For more information, refer to the Timer Clock Characteristics table in the *Timer* chapter in the *Cyclone V Device Handbook, Volume 3*.	`h2f_l4sp0_interrupt` `h2f_l4sp1_interrupt`
Enable NAND interrupts	Enables interface for the HPS NAND controller interrupt to the FPGA	`h2f_nand_interrupt`

Parameter Name	Parameter Description	Interface Name
Enable OSC timer interrupts	Enables interface for the HPS OSC timer interrupt to the FPGA. For more information, refer to the Timer Clock Characteristics table in the *Timer* chapter in the *Cyclone V Device Handbook, Volume 3*.	`h2f_osc0_interrupt` `h2f_osc1_interrupt`
Enable Quad SPI interrupts	Enables interface for the HPS QSPI controller interrupt to the FPGA	`h2f_qspi_interrupt`
Enable SD/MMC interrupts	Enables interface for the HPS SD/MMC controller interrupt to the FPGA	`h2f_sdmmc_interrupt`
Enable SPI master interrupts	Enables interface for the HPS SPI master controller interrupt to the FPGA	`h2f_spi0_interrupt` `h2f_spi1_interrupt`
Enable SPI slave interrupts	Enables interface for the HPS SPI slave controller interrupt to the FPGA	`h2f_spi2_interrupt` `h2f_spi3_interrupt`
Enable UART interrupts	Enables interface for the HPS UART controller interrupt to the FPGA	`h2f_uart0_interrupt` `h2f_uart1_interrupt`
Enable USB interrupts	Enables interface for the HPS USB controller interrupt to the FPGA	`h2f_usb0_interrupt` `h2f_usb1_interrupt`
Enable watchdog interrupts	Enables interface for the HPS watchdog interrupt to the FPGA	`h2f_wdog0_interrupt` `h2f_wdog1_interrupt`

Related Information

Timer
For more information, refer to the *Timer* chapter in the *Cyclone V Device Handbook, Volume 3*.

Configuring Peripheral Pin Multiplexing

The **Peripheral Pin Multiplexing** tab is one of four tabs on the HPS Component. This tab contains several groups with the following parameters:

Configuring Peripherals on page 2-9

Connecting Unassigned Pins to GPIO on page 2-10

Using Unassigned IO as LoanIO on page 2-10
You can utilize unused HPS IOs as LoanIO, which is directly driven by the FPGA and can be used as input, output, or bi-directional.

Resolving Pin Multiplexing Conflicts on page 2-11
Use the **Peripherals MUX Table** to view pins with invalid multiple assignments.

Route Peripheral Signals to FPGA on page 2-11
You can route the peripheral signals to the FPGA fabric and assign them to the FPGA I/O pins.

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.

When you assign a peripheral to an HPS I/O pin set, the corresponding peripheral is highlighted in the **Peripherals MUX Table** at the end of **Peripheral Pin Multiplexing** tab. The other unused peripheral pin set remains un-highlighted in the table.

Each list in the **Peripheral Pin Multiplexing** tab has a hint describing in detail the options available in the list. The hint for each mode list shows the signals used by each available mode. View each hint by hovering over the corresponding mode list.

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

Related Information

- **CoreSight Debug and Trace**
  Trace port interface unit (TPIU). Enabling the TPIU exposes trace signals to the device pins. Refer to the *CoreSight Debug and Trace* chapter in the *Cyclone V Device Handbook, Volume 3* for more information.
- **NAND Flash Controller**
  For more information, refer to the *NAND Flash Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **SD/MMC Controller**
  For more information, refer to the *SD/MMC Controller Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **Quad SPI Flash Controller**
  For more information, refer to the *Quad SPI Flash Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **Ethernet Media Access Controller**
  For more information, refer to the *Ethernet Media Access Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **USB 2.0 OTG Controller**
  For more information, refer to the *USB 2.0 OTG Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **SPI Controller**
  For more information, refer to the *SPI Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.

- **I^2C Controller**
  For more information, refer to the *I^2C Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **UART Controller**
  For more information, refer to the *UART Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **CAN Controller**
  For more information refer to the *CAN Controller* chapter in the *Cyclone V Device Handbook, Volume 3*.

Connecting Unassigned Pins to GPIO

For pins that are not assigned to any peripherals, you can assign them to the GPIO peripheral by clicking on the corresponding GPIO push button in the table. Click on the selected push button to remove GPIO pin assignment. The table also shows the GPIO bit number that correlates to the I/O pins.

Using Unassigned IO as LoanIO

You can utilize unused HPS IOs as LoanIO, which is directly driven by the FPGA and can be used as input, output, or bi-directional.

Each LOANIO port has an input, output, and output enable, which directly controls the HPS IO functions. The LoanIO only operates when the HPS registers have been set up in the pre-loader to allow their operation. The LoanIO are asynchronous, thus no clocking is required.

Use the following steps to enable the LoanIO signals:

1. Select **Peripheral Pins Multiplexing** tap in the HPS Megawizard.
2. Choose the corresponding LoanIO pins from the **Peripherals Mux Table**, click the push button to select/unselect it.
3. Export the peripheral signals out of the Qsys system.
4. In the Quartus II software, connect the user logic to the LoanIO interface to drive the HPS IOs.

Table 2-8: Generated Conduit Signal Interface

Conduit Name	Direction	Declarations
._hps_io_gpio_inst_LOANIOXX	Bi-direction	User must declare as a top-level pin; pin assignment is hardcoded following the HPS IO location.
._h2f_loan_io_in(1)	Out	HPS IO data input signal, output to FPGA user logic.
._h2f_loan_io_out(1)	In	HPS IO data output signal, input from FPGA user logic.
._h2f_loan_io_oe(1)	In	HPS IO data output enable signal, input from FPGA user logic.

(1) Qsys will generate a full signal array for `h2f_loan_io_in, h2f_loan_io_out`, and `h2f_loan_io_oe`. You must assign user logic to the specific signal array. For example, you have triggered LoanIO 40, so its respective signal array is `h2f_loan_io_in[40], h2f_loan_io_out[40]`, and `h2f_loan_io_oe[40]`.

Resolving Pin Multiplexing Conflicts

Use the **Peripherals MUX Table** to view pins with invalid multiple assignments.

Pins that have multiple peripherals assigned to them are highlighted in red color with the conflicting peripherals in boldface font style. Solve the multiplexing conflicts by de-selecting peripherals that are assigned to the pins, leaving only one peripheral, which is needed.

Route Peripheral Signals to FPGA

You can route the peripheral signals to the FPGA fabric and assign them to the FPGA I/O pins.

The following steps show you how to enable the peripheral signals:

1. Select FPGA in the peripheral pin multiplexing selection drop-down box.
2. Export the peripheral signals out of Qsys system.
3. In the Quartus II software, connect the signals to the FPGA I/O pins.

Configuring HPS Clocks

The **HPS Clocks** tab is one of four tabs on the HPS Component. This tab contains several groups with the following parameters:

User Clocks on page 2-11

PLL Reference Clocks on page 2-12

Peripheral FPGA Clocks on page 2-13

User Clocks

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.

Related Information

Selecting PLL Output Frequency and Phase on page 2-14

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*.

Table 2-9: 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	

Parameter Name	Parameter Description	Clock Interface Name
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	

Related Information

Selecting PLL Output Frequency and Phase on page 2-14

Clock Frequency Usage

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

Related Information

- **Selecting PLL Output Frequency and Phase** on page 2-14
- **Clock Manager**
  For general information about clock signals, refer to the *Clock Manager* chapter in the *Cyclone V Device Handbook, Volume 3*.

PLL Reference Clocks

Table 2-10: 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`

Related Information

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

Peripheral FPGA Clocks

Table 2-11: Peripheral FPGA Clock Parameters

Parameter Name	Parameter Description
EMAC 0 (emac0_md_clk clock frequency)	If EMAC 0 peripheral is routed to FPGA, use this field to specify EMAC 0 MDIO clock frequency
EMAC 0 (emac0_gtx_clk clock frequency)	If EMAC 0 peripheral is routed to FPGA, use this field to specify EMAC 0 transmit clock frequency
EMAC 1 (emac1_md_clk clock frequency)	If EMAC 1 peripheral is routed to FPGA, use this field to specify EMAC 1 MDIO clock frequency
EMAC 1 (emac1_gtx_clk clock frequency)	If EMAC 1 peripheral is routed to FPGA, use this field to specify EMAC 1 transmit clock frequency
QSPI (qspi_sclk_out clock frequency)	If QSPI peripheral is routed to FPGA, use this field to specify QSPI serial clock frequency
SPIM 0 (spim0_sclk_out clock frequency)	If SPI master 0 peripheral is routed to FPGA, use this field to specify SPI master 0 output clock frequency
SPIM 1 (spim1_sclk_out clock frequency)	If SPI master 1 peripheral is routed to FPGA, use this field to specify SPI master 1 output clock frequency
I^2C0 (i2c0_clk clock frequency)	If I^2C 0 peripheral is routed to FPGA, use this field to specify I^2C 0 output clock frequency
I^2C1 (i2c1_clk clock frequency)	If I^2C 1 peripheral is routed to FPGA, use this field to specify I^2C 1 output clock frequency
I^2C2 (i2c2_clk clock frequency)	If I^2C 2 peripheral is routed to FPGA, use this field to specify I^2C 2 output clock frequency
I^2C3 (i2c3_clk clock frequency)	If I^2C 3 peripheral is routed to FPGA, use this field to specify I^2C 3 output clock frequency

Configuring the External Memory Interface

The **SDRAM** tab is one of four tabs on the HPS component. This tab contains the PLL output frequency and phase group.

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 **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*.

Note: 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:

Related Information

- **Selecting PLL Output Frequency and Phase** on page 2-14
- **Implementing and Parameterizing Memory IP**
  The Implementing and Parameterizing Memory IP chapter in the *External Memory Interface* Handbook.
- **Functional Description Hard Memory Interface**
  The Functional Description--Hard Memory Interface chapter in 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 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**.

Note: 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.

Related Information

- **Implementing and Parameterizing Memory IP**
  The Implementing and Parameterizing Memory IP chapter in the *External Memory Interface* Handbook.
- **Functional Description Hard Memory Interface**
  The Functional Description--Hard Memory Interface chapter in the *External Memory Interface* Handbook. "EMI-Related HPS Features in SoC Devices" describes features specific to the HPS SDRAM controller.

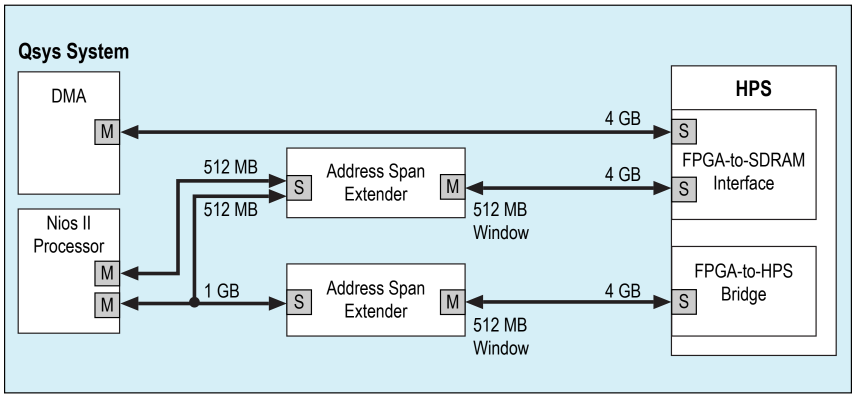
Using the Address Span Extender Component

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 2-1: Address Span Extender Components

Two address span extender components used in a system with the HPS.



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.

Related Information

Qsys Interconnect and System Design Components

For more information about the address span extender, refer to *Bridges* in the *Qsys* Interconnect and System Design Components chapter in 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.
2. Add <*qsys_system_name*>*.qip* to the Quartus II project. <*qsys_system_name*>*.qip* is the Quartus II IP File for the HPS component, generated by Qsys.
3. Perform analysis and synthesis 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.

   Note: For information about running the pin assignment script, refer to "MegaWizard Plug-In Manager Flow" in the *Implementing and* Parameterizing Memory IP chapter in 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*, 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.

Related Information

- **Configuring Peripheral Pin Multiplexing**
- **Creating a System with Qsys**
  For general information about using Qsys, refer to the *Creating a System with Qsys* chapter in the *Quartus® II* Handbook.
- **Implementing and Parameterizing Memory IP**
  The Implementing and Parameterizing Memory IP chapter in the *External Memory Interface* Handbook.
- **Simulating the HPS component**
  For a description of the simulation files generated, refer to "Simulation Flow" in the *Simulating the HPS Component* chapter of the *Cyclone V Device Handbook, Volume 3.*
- **About Back-Annotating Assignments**
  For information about back-annotating pin assignments, refer to *About Back-Annotating Assignments* in Quartus II Help.

Document Revision History

Table 2-12: Document Revision History

Date	Version	Changes
June 2014	2014.06.30	• Updated the Instantiating the HPS Component section. • Added the Using Unassigned IO as LoanIO section.
February 2014	2014.02.28	• Add interfaces to tables • Add parameters to General Parameters table
December 2013	1.2	Maintenance release.
November 2012	1.1	• 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.

HPS Component Interfaces 3

2014.06.30

cv_54028 Subscribe Send Feedback

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.

Related Information

- **Avalon**
  For Avalon protocol timing, refer to *Avalon Interface Specifications*.
- **Instantiating the HPS Component**
  For information about instantiating the HPS component, refer to the *Instantiating the HPS Component* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **ARM Infocenter (http://infocenter.arm.com)**
  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 info center website.

Memory-Mapped Interfaces

FPGA-to-HPS Bridge

Table 3-1: FPGA-to-HPS Bridges and Clocks

Interface Name	Description	Associated Clock Interface
`f2h_axi_slave`	FPGA-to-HPS AXI slave interface	`f2h_axi_clock`

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 adapters. The Qsys system integration tool automatically generates adapter 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.

Related Information

- Clocks on page 3-5
- HPS-FPGA Bridges
  For more information, refer to the *HPS-FPGA Bridges* chapter in the *Cyclone V Device Handbook, Volume 3.*
- Instantiating the HPS Component
  For information about the address span extender, refer to "Using the Address Span Extender Component" in the *Instantiating the HPS Component* chapter in the *Cyclone V Device Handbook, Volume 3.*

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.

Related Information

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

HPStoFPGA and Lightweight HPS-to-FPGA Bridges

Table 3-2: HPStoFPGA and Lightweight HPS-to-FPGA Bridges and Clocks

Interface Name	Description	Associated Clock Interface
`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`

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.

Related Information

- Clocks on page 3-5
- HPS-FPGA Bridges
  For more information, refer to the *HPS-FPGA Bridges* chapter in the *Cyclone V Device Handbook, Volume 3.*

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 3-3: FPGA-to-HPS SDRAM Interfaces and Clocks

Interface Name	Description	Associated Clock Interface
`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`

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.

Related Information

- Clocks on page 3-5
- **SDRAM Controller Subsystem**
  For more information about available combinations of interfaces and ports, refer to the *SDRAM Controller Subsystem* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **Instantiating the HPS Component**
  For information about the address span extender, refer to "Using the Address Span Extender Component" in the *Instantiating the HPS Component* chapter in the *Cyclone V Device Handbook, Volume 3*.

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
- `h 2f_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 clocksgenerated 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

Peripheral FPGA Clocks

The HPS peripheral clocks are exposed when the peripheral signals are routed to the FPGA.

Table 3-4: Peripheral FPGA Clocks

Clock Name	Description
`emac_md_clk`	Ethernet PHY management interface clock
`emac_gtx_clk`	Ethernet transmit clock that is used by the PHY in GMII mode
`emac_rx_clk_in`	Ethernet MAC reference clock from the PHY
`emac_tx_clk_in`	Ethernet MAC uses this clock input for TX reference
`emac_ptp_ref_clock`	Ethernet timestamp precision time protocol (PTP) reference clock
`qspi_sclk_out`	QSPI master clock output
`spim_sclk_out`	SPI master serial clock output
`spis_sclk_in`	SPI slave serial clock input
`i2c_clk`	I^2C outgoing clock (part of the SCL bidirectional pin signals)
`i2c_scl_in`	I^2C incoming clock (part of the SCL bidirectional pin signals)

Resets

This section describes the reset interfaces to the HPS component.

Related Information

Reset Manager

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

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

Peripheral Reset Interfaces

The following are Ethernet reset interfaces, that can be used when the Ethernet is routed to the FPGA:

- `emac_tx_reset`— Ethernet transmit clock reset output used to reset external PHY TX clock domain logic
- `emac_rx_reset`— Ethernet receive clock reset output used to reset external PHY RX clock domain logic

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`
- `h2f_tpiu_clock`

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

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_re q1`—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

Each of the DMA peripheral request interface contains the following three signals:

- `f2h_dma_req`—This signal is used to request burst transfer using the DMA
- `f2h_dma_single`—This signal is used to request single word transfer using the DMA
- `f2h_dma_ack`—This signal indicates the DMA acknowledgment upon requests from the FPGA

Related Information

DMA Controller

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

Other Interfaces

Related Information

Cortex-A9 MPU Subsystem

For more information, refer to the *Cortex-A9 MPU Subsystem* chapter in the *Cyclone V Device Handbook, Volume 3*.

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.

Related Information

Cortex-A9 MPU Subsystem

For more information, refer to the *Cortex-A9 MPU Subsystem* chapter in the *Cyclone V Device Handbook, Volume 3*.

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`).

Related Information

Instantiating the HPS Component

There are various HPS peripherals to FPGA interrupt interfaces that you can configure. To learn the complete list of interfaces, refer to the *Instantiating the HPS Component* chapter in the *Cyclone V Device Handbook, Volume 3*.

Boot from FPGA Interface

You can enable the boot from FPGA interface to indicate the availability of preloader software in the FPGA memory. This interface is used to indicate the availability of a fallback preloader software in FPGA memory as well. The fallback preloader is used when there is no valid preloader image found in HPS flash memories.

Input-only General Purpose Interface

You can enable a general purpose interface that is connected to the general purpose I/O (GPIO) peripheral of the HPS. This is an input-only interface with 14-bit wide width. The interface share the same I/O pins with the HPS DDR SDRAM controller.

Document Revision History

Table 3-5: Document Revision History

Date	Version	Changes
June 2014	2014.06.30	Added address map and register descriptions

Date	Version	Changes
February 2014	2014.02.28	Added in new sections • Peripheral FPGA Clocks • Peripheral Reset Interfaces • Boot from FPGA Interface • Input-only General Purpose Interfaces Removed section • General Purpose Interfaces Updated sections • Trace Port Interface Unit • Peripheral Signal Interfaces • FPGA-to-HPS Interrupts
December 2013	2013.12.30	Minor formatting issues
November 2012	1.1	• 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 4

2014.06.30

cv_54030 Subscribe Send Feedback

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.

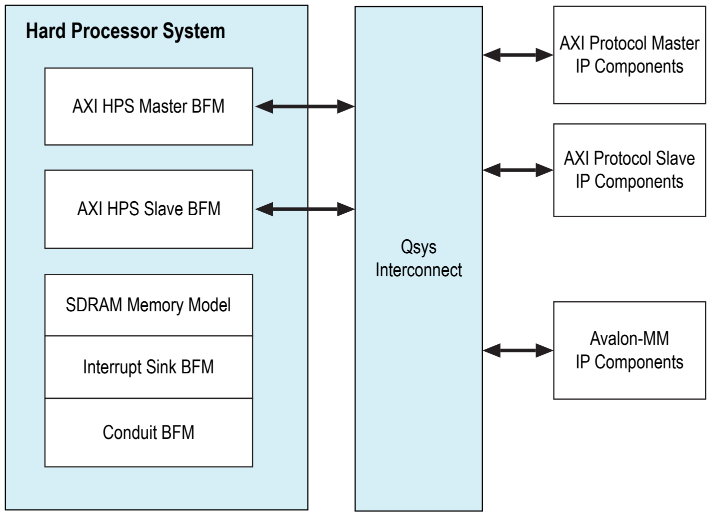
You specify simulation support files when you instantiate the HPS component in the Qsys system integration tool. When you enable a particular HPS-FPGA interface, Qsys provides the corresponding model during the generation process.

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
- Boot from FPGA interface
- Input only general purpose interface

Figure 4-1: HPS BFM Block Diagram



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

HPS simulation supports only Verilog HDL or SystemVerilog simulation environments.

Related Information

- **Simulation Flows** on page 4-10
- **Instantiating the HPS Component**
  For more information, refer to the *Instantiating the HPS Component* chapter in the *Cyclone V Device Handbook, Volume 3.*
- **Avalon Verification IP Suite User Guide**
  For information about the BFM API.
- **Mentor Verification IP Altera Edition User Guide**
  For information about the BFM API.

Clock and Reset Interfaces

Related Information

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

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 4-1: HPS Clock Output Interface Simulation Model

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

Interface Name	BFM Instance Name
`h2f_user0_clock`	`h2f_user0_clock_inst`
`h2f_user1_clock`	`h2f_user1_clock_inst`
`h2f_user2_clock`	`h2f_user2_clock_inst`
`h2f_tpiu_clock`	`h2f_tpiu_clock_inst`

Qsys does not generate BFMs for FPGA-to-HPS clock input interfaces.

Reset Interface

The HPS reset request and handshake interfaces are connected to Altera conduit BFMs for simulation.

Table 4-2: HPS Reset Input Interface Simulation Model

You can monitor the reset request interface state changes or set the interface by using the API listed.

Interface Name	BFM Instance Name	API Function Names
`f2h_cold_reset_req`	`f2h_cold_reset_req_inst`	`get_f2h_cold_rst_req_n()`
`f2h_debug_reset_req`	`f2h_debug_reset_req_inst`	`get_f2h_dbg_rst_req_n()`
`f2h_warm_reset_req`	`f2h_warm_reset_req_inst`	`get_f2h_warm_rst_req_n()`

`h2f_warm_reset_handshake`	`h2f_warm_reset_handshake_inst`	`set_h2f_pending_rst_req_n()`
		`get_f2h_pending_rst_ack_n()`

Table 4-3: HPS Reset Output Interface Simulation Model

The Altera reset source BFM application programming interface applies to all the BFMs listed.

Interface Name	BFM Instance Name
`h2f_reset`	`h2f_reset_inst`
`h2f_cold_reset`	`h2f_cold_reset_inst`
`h2f_debug_apb_reset`	`h2f_debug_apb_reset_inst`

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

The HPS reset output interface is connected to a reset source BFM. Qsys configures the BFM as shown. The parameter value of the instantiated BFM as configured for HPS simulation.

Parameter	BFM Value	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.

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 with an instance name of `f2h_axi_slave_inst`. Qsys configures the BFM as shown in the following table. The BFM clock input is connected to `f2h_axi_clock` clock.

Table 4-5: 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.

Related Information

- **HPS Component Interfaces**
  For general information about interfaces, refer to "Memory-Mapped Interfaces" in the *HPS Component Interfaces* chapter of the *Cyclone V Device Handbook, Volume 3*
- **Mentor Verification IP Altera Edition User Guide**

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 with an instance name of `h2f_axi_master_inst`. Qsys configures the BFM as shown in the following table. The BFM clock input is connected to `h2f_axi_clock` clock.

Table 4-6: Configuration of HPS-to-FPGA AXI Master BFM

Parameter	Value
AXI Address Width	30

Parameter	Value
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.

Related Information

- **HPS Component Interfaces**
  For general information about interfaces, refer to "Memory-Mapped Interfaces" in the *HPS Component Interfaces* chapter of the *Cyclone V Device Handbook, Volume 3*
- **Mentor Verification IP Altera Edition User Guide**

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 with an instance name of `h2f_lw_axi_master_inst`. Qsys configures the BFM as shown in the following table. The BFM clock input is connected to `h2f_lw_axi_clock` clock.

Table 4-7: 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.

Related Information

- **HPS Component Interfaces**
  For general information about interfaces, refer to "Memory-Mapped Interfaces" in the *HPS Component Interfaces* chapter of the *Cyclone V Device Handbook, Volume 3*
- **Mentor Verification IP Altera Edition User Guide**

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.

Related Information

Functional Description—Hard Memory Interface

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. The following table 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.

Table 4-8: 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_inst`	`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. The following table 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.

Table 4-9: HPS-to-FPGA MPU Event Interface Simulation Model

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

Interface Name	BFM Instance Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names
`h2f_mpu_events`	`h2f_mpu_events_inst`	`get_h2f_mpu_eventi()` `set_h2f_mpu_evento()` `set_h2f_mpu_standbywfe()` `set_h2f_mpu_standbywfi()`	`get_eventi()` `set_evento()` `set_standbywfe()` `set_standbywfi()`

FPGA-to-HPS Interrupts Interface

The FPGA-to-HPS interrupts interface is connected to an Altera Avalon interrupt sink BFM for simulation.

Table 4-10: FPGA-to-HPS Interrupts Interface Simulation Model

Interface Name	BFM Instance Name
`f2h_irq0`	`f2h_irq0_inst`

Interface Name	BFM Instance Name
`f2h_irq1`	`f2h_irq1_inst`

Note: The simulation support for HPS peripherals to FPGA interfaces are not supported at the moment.

Related Information

Reset Interface on page 4-3

The Altera Avalon interrupt sink BFM API applies to all the BFMs listed in the HPS Reset Output Interface Simulation Model.

HPS-to-FPGA Debug APB Interface

The HPS-to-FPGA debug APB interface simulation is not supported at the moment. Updates on this subject will be provided in upcoming versions of this handbook.

Related Information

Product Support

For more information, refer to the KDB solution on the Altera product support solution page.

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. The following table 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.

Table 4-12: 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_inst`	`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. The following table 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.

Table 4-13: 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_inst	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. The following table 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.

Table 4-14: 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_inst	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. The following table 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.

Table 4-15: FPGA-to-HPS DMA Handshake Interface Simulation Model

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

Interface Name	BFM Instance Name	RTL Simulation API Function Names	Post-Fit Simulation API Function Names

f2h_dma_req0	f2h_dma_req0_inst	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_inst	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_inst	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_inst	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_inst	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_inst	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_inst	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_inst	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()

Boot from FPGA Interface

The boot from FPGA interface is connected to an Altera conduit BFM for simulation. You can monitor the interface state changes or set the interface by using the API functions in the table below.

Table 4-16: Boot from FPGA Interface Simulation Model

Interface Name	BFM Name	RTL simulation API function names	Post-fit simulation API function names

`f2h_boot_from_fpga`	`f2h_boot_from_fpga_inst`	`get_f2h_boot_from_fpga_ready()`
		`get_f2h_boot_from_fpga_on_failure()`
			`get_boot_fpga_ready()`
			`get_boot_from_fpga_on_failure()`

General Purpose Input (GPI) Interface

The general purpose input interface is connected to an Altera conduit BFM for simulation. You can monitor the interface state changes or set the interface by using the API functions in the table below.

Table 4-17: General Purpose Input Interface Simulation Model

Interface Name	BFM Name	RTL simulation API function names

hps_io	hps_io_inst	get_hps_io_gpio_inst_HLGPI0()
		get_hps_io_gpio_inst_HLGPI1()
		get_hps_io_gpio_inst_HLGPI2()
		get_hps_io_gpio_inst_HLGPI3()
		get_hps_io_gpio_inst_HLGPI4()
		get_hps_io_gpio_inst_HLGPI5()
		get_hps_io_gpio_inst_HLGPI6()
		get_hps_io_gpio_inst_HLGPI7()
		get_hps_io_gpio_inst_HLGPI8()
		get_hps_io_gpio_inst_HLGPI9()
		get_hps_io_gpio_inst_HLGPI10()
		get_hps_io_gpio_inst_HLGPI11()
		get_hps_io_gpio_inst_HLGPI12()
		get_hps_io_gpio_inst_HLGPI13()

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, which are based on Altera Complete Design Suite (ACDS 14.0):

1. **Specifying HPS Simulation Model in Qsys** on page 4-11

2. **Generating HPS Simulation Model in Qsys** on page 4-12
3. **Running the Simulation** on page 4-13

Related Information

Simulating Altera Designs
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, widths and are opposite in direction to what is shown in the HPS conduit Interfaces table.

HPS Conduit Interfaces on page 4-11

Related Information

Instantiating the HPS Component
For more information, refer to the *Instantiating the HPS Component* chapter in the *Cyclone V Device Handbook, Volume 3*.

HPS Conduit Interfaces

Table 4-18: HPS Conduit Interfaces

Role Name	Direction	Width
h2f_warm_reset_handshake		
`h2f_pending_rst_req_n`	Output	1
`f2h_pending_rst_ack_n`	Input	1
h2f_gp		
`gp_in`	Input	32
`gp_out`	Output	32
h2f_mpu_events		
`eventi`	Input	1
`evento`	Output	1
`standbywfe`	Output	2

Role Name	Direction	Width
`standbywfi`	Output	2
f2h_dma_req0 to f2h_dma_req7		
`dma_req`	Input	1
`dma_single`	Input	1
`dma_ack`	Output	1
h2f_debug_apb_sideband		
`pclken`	Input	1
`dbg_apb_disable`	Input	1
f2h_stm_hw_events		
`stm_hwevents`	Input	28
h2f_cti		
`trig_in`	Input	8
`trig_in_ack`	Output	8
`trig_out`	Output	8
`trig_out_ack`	Input	8
`asicctl`	Output	8
`fpga_clk_en`	Input	1
h2f_tpiu		
`clk_ctl`	Input	1
`data`	Output	32
f2h_boot_from_fpga		
`boot_from_fpga_ready`	Input	1
`boot_from_fpga_on_failure`	Input	1

Generating HPS Simulation Model in Qsys

The following steps outline how to generate the simulation model.

1. In Qsys, click **Generate HDL** under the Generate menu.
2. For RTL simulation, perform the following steps:
    - Set **Create simulation model** to **Verilog**.

- Click **Generate**.

  Note: HPS simulation does not support the VHDL simulation environment.

For post-fit simulation, perform the following steps:

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

Related Information

- **Instantiating the HPS Component**
  For more information, refer to the *Instantiating the HPS Component* chapter in the *Cyclone V Device Handbook, Volume 3*.
- **Creating a System with Qsys**
  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 the Simulation

Refer to one of the following.

Running HPS RTL Simulation on page 4-13

Running HPS Post-Fit Simulation on page 4-14

Running HPS RTL Simulation

Qsys generates scripts for various simulators that you use to complete the simulation process.

Table 4-19: 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**
Aldec RivieraPro	**rivierapro_ setup.tcl**	*<project directory>*/*<Qsys design name>***/simulation/aldec**

Related Information

- **Avalon Verification IP Suite User Guide**
  For information about the BFM API.
- **Mentor Verification IP Altera Edition User Guide**
  For information about the BFM API.

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**.
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.
7. Start simulation.

Related Information

Simulation Page (Settings Dialog Box)

Post-Fit Simulation Files

Table 4-20: Post-Fit Simulation Files

This table uses the following symbols:

- *<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**

Library	Directory	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/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 (The device simulation library is not needed with Modelsim-Altera.)	*<Device Sim Lib>*/	**altera_primitives.v** **220model.v** **sgate.v** **altera_mf.v** **altera_lnsim.sv** **cyclonev_atoms.v** **arriav_atoms.v** **mentor/cyclonev_atoms_ncrypt.v** **mentor/arriav_atoms_ncrypt.v**

Library	Directory	File
EDA Netlist Writer Generated Post-Fit Simulation Model	*<User project directory>/*	***.vo** ***.vho** (Mixed-language simulator is needed for Verilog HDL and VHDL mixed design)
User testbench files	*<User project directory>/*	***.v** ***.sv** ***.vhd** (Mixed-language simulator is needed for Verilog HDL and VHDL mixed design)

BFM API hierarchy Format

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 4-21: Document Revision History

Date	Version	Changes
June 2014	2014.06.30	• Updated the Simulation Flows section. • Updated the Generating HPS Simulation Model in Qsys section.
February 2014	2014.02.28	Added new sections: • Boot from FPGA Interface • General Purpose Input (GPI) Interface Updated content in sections: • Specifying HPS Simulation Model in Qsys • Running HPS RTL Simulation
December 2013	2013.12.30	Maintenance release
November 2012	1.1	• 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.