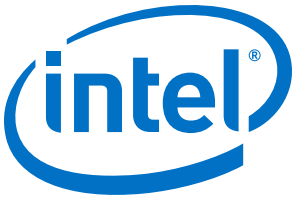


Intel® Stratix® 10Hard Processor System Component Reference Manual

Updated for Intel® Quartus® Prime Design Suite: **18.1**

S10-HPSCOMPONENT | 2018.11.30
Latest document on the web: **PDF** | **HTML**

Contents

1. Introduction to the Intel® Stratix® 10 Hard Processor System Component

The hard processor system (HPS) component is a wrapper that interfaces logic in your design to the:

- HPS hard logic
- Simulation models
- Bus functional models (BFMs)
- Software handoff files

The HPS component instantiates the HPS hard logic in your design and enables other soft components to interface with the HPS hard logic. The HPS component has a small footprint in the FPGA fabric, as the component only serves to enable soft logic to hard logic connection in the HPS.

After you connect the soft logic to the HPS, you can use Platform Designer to ensure the following features:

- Interoperability by adapting Avalon® Memory-Mapped (Avalon-MM) interfaces to AXI*
- Handling of data width mismatches and clock domain transfer crossings

You are able to integrate Intel® FPGA IP, 3rd party IP, and custom IP that you define into the HPS without creating integration logic. This reference manual details the interfaces exposed and configured by the options in the component.

For more information about the HPS system architecture and features, refer to the "Introduction to the Hard Processor" chapter in the *Intel Stratix® 10 Hard Processor System Technical Reference Manual*.

Related Information

- Introduction to the Hard Processor System
  For more information, refer to this chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*
- Intel Stratix 10 Device Datasheet
- Intel Quartus Prime Pro Edition User Guide: Platform Designer

1.1. Cortex*-A53 MPCore* Processor

The Arm* Cortex*-A53 MPCore* Processor is composed of four Armv8-A architecture central processing units (CPUs).

Related Information

Cortex-A53 MPCore Processor
For more information, refer to this chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*

1.2. CoreSight* Debug Components

The following lists the Arm CoreSight* debug components:

- Debug Access Port (DAP)
- System Trace Macrocell (STM)
- Embedded Trace FIFO (ETF)
- AMBA* Trace Bus Replicator
- Embedded Trace Router (ETR)
- Trace Port Interface Unit (TPIU)
- Embedded Cross Trigger (ECT)

Related Information

CoreSight Debug and Trace
For more information, refer to this chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*

1.3. Interconnect

The interconnect consists of the L3 interconnect, SDRAM L3 interconnect, and level 4 (L4) buses.

The L3 Interconnect provides high-bandwidth routing featuring Arm TrustZone*-compliant security firewalls with programmable Quality of Service (QoS) between masters and slaves in the HPS. The L3 Interconnect also provides a lower performance tier of L4 buses for mid to low-bandwidth slave peripherals and peripheral control and status registers. The SDRAM L3 interconnect connects the HPS to the hard memory controller located in the FPGA I/O column.

Related Information

System Interconnect
For more information, refer to this chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*

1.4. FPGA Bridges

The FPGA bridges provide a variety of communication channels between the HPS and the FPGA fabric. The HPS is highly integrated with the FPGA fabric, resulting in thousands of connecting signals. Some of the HPS-to-FPGA interfaces include:

- FPGA-to-HPS bridge
- HPS-to-FPGA bridge
- Lightweight HPS-to-FPGA bridge
- FPGA-to-HPS SDRAM bridge

Related Information

HPS-FPGA Bridges
For more information, refer to this chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*

1.5. Memory Controllers

The following lists the memory controller peripherals:

- NAND Flash Controller
- SD/MMC Controller

Related Information

- NAND Flash Controller
  For more information, refer to this chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*
- SD/MMC Controller
  For more information, refer to this chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*

1.6. Support Peripherals

The following lists the support peripherals:

- Clock Manager
- Reset Manager
- System Manager
- Timer
- Watchdog Timer
- Direct Memory Access (DMA) Controller
- Error Checking and Correction Controller

Related Information

- Clock Manager
  For more information about the support peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- Reset Manager
  For more information about the support peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- System Manager
  For more information about the support peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- Timer
  For more information about the support peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

- Watchdog Timer
  For more information about the support peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- DMA Controller
  For more information about the support peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- Error Checking and Correction Controller
  For more information about the support peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

1.6.1. Interface Peripherals

The following lists the interface peripherals:

- Ethernet Media Access Controllers (EMAC)
- USB 2.0 On-The-Go (OTG) Controllers
- I^2C Controllers
- UARTs
- SPI Master Controllers
- SPI Slave Controllers
- GPIO Interfaces

Related Information

- Ethernet Media Access Controller
  For more information about the interface peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- USB 2.0 OTG Controller
  For more information about the interface peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- SPI Controller
  For more information about the interface peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- I^2C Controller
  For more information about the interface peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.
- UART Controller
  For more information about the interface peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

- General-Purpose I/O Interface
  For more information about the interface peripherals, refer to its corresponding chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

1.6.2. On-Chip Memories

On-Chip RAM is the only on-chip memory.

Related Information

On-Chip Memory
For more information, refer to this chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*

1.7. Introduction to the HPS Component Revision History

Table 1. Document Revision History

Document Version	Changes
2018.11.30	Each of the folder tabs have been updated to reflect updates for Intel Stratix 10.
2018.10.12	Maintenance release
2017.11.06	Initial release

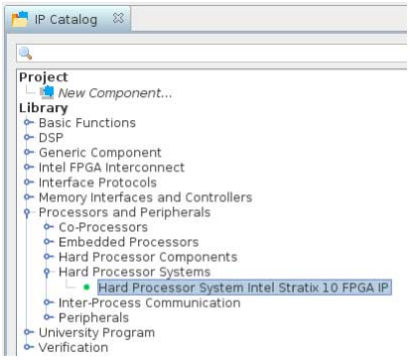
2. Configuring the Intel Stratix 10 Hard Processor System Component

This chapter describes the parameters available and the interfaces enabled by those parameters in the hard processor system (HPS) component parameter editor. The parameter editor opens when you add or edit an Intel Stratix 10 HPS component in Platform Designer.

2.1. Parameterizing the HPS Component

1. Install the current version of the Intel Quartus® Prime Pro Edition design software, along with Intel Stratix 10 device support.
2. Open the Intel Quartus Prime software.
3. Open Platform Designer by selecting **Tools ➤ Platform Designer**.
4. Select an existing Intel Quartus Prime project and Platform Designer system or create new files. Ensure that Intel Stratix 10 is selected in the **Device Family** dropdown, and a device (denoted in this format: 1SXxxx) is selected in the **Device Part** dropdown.
5. In the **IP Catalog** tab, under Library, select **Processors and Peripherals ➤ Hard Processor Systems ➤ Hard Processor System Intel Stratix 10 FPGA IP**.

Figure 1. Platform Designer IP Catalog



2.2. FPGA Interfaces

The **FPGA Interfaces** tab allows you to specify options for the primary interfaces between the FPGA and the HPS. The following groups in this tab are:

- General
- HPS FPGA AXI Bridges
- HPS Boot Source
- DMA Peripheral Request
- Interrupts

Figure 2. HPS Component Parameter Editor FPGA Interfaces Tab

2.2.1. General Interfaces

2.2.1.1. Enable MPU Standby and Event Interfaces

Microprocessor Unit (MPU) standby signals are notification signals to the FPGA fabric that the MPU is in standby. Event signals wake up the Cortex-A53 processors from a wait-for-event (WFE) state. Turning on the **Enable MPU Standby and Event Interfaces** option enables the **h2f_mpu_events** conduit, which is comprised of the following signals:

- `h2f_mpu_eventi`—Input for FPGA to signal events to all processors. This FPGA-to-HPS signal is used to wake up a processor that is in a WFE state. Asserting this signal has the same effect as executing the SEV instruction in the Cortex-A53. You must deassert the signal until the FPGA fabric configures. You must assert the signal high for at least two MPU clock cycles for the processor to recognize any of the Cortex-A53 cores.
- `h2f_mpu_evento`—Output from any MPU core into the FPGA fabric. This HPS-to-FPGA signal is asserted when an SEV instruction is executed by one of the Cortex-A53 processors. This signal is output as a multiple cycle pulse so logic in the FPGA should use a rising edge detector circuit to detect the occurrence of the event.
- `h2f_mpu_standbywfe[3:0]`—Output per processor that indicates if the processor is in WFE standby mode. When high, the processor is in WFE standby mode.
- `h2f_mpu_standbywfi[3:0]`—Output per processor that indicates if the processor is in the wait-for-interrupt (WFI) standby mode. When the logic level is high, the processor is in WFI standby mode.

2.2.1.2. Enable General Purpose Signals

Turning on the **Enable General Purpose Signals** option enables the **h2f_gp** conduit, which is comprised of a pair of 32-bit uni-directional general-purpose interfaces between the HPS System Manager and the FPGA fabric. These signal names are `h2f_gp_in` and `h2f_gp_out`, and are inputs to the HPS and outputs from the HPS, respectively.

2.2.1.3. Enable Debug APB Interface

The debug Advanced Peripheral Bus (APB)* interface allows debug components in the FPGA fabric to access debug components in the HPS.

For more information about the **Debug APB interface**, refer to the "CoreSight Debug and Trace" chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

Turning on this option enables the following interfaces and signals:

Table 2. APB Interfaces and Signals

Interface Name	Interface Type	Signals
h2f_debug_apb_clock	Clock Input	`h2f_dbg_apb_clk`
h2f_debug_apb_reset	Reset Output	`h2f_dbg_apb_rst_n`

h2f_debug_apb	APB Master	`h2f_dbg_apb_PADDR[14..0]`
		`h2f_dbg_apb_PADDR31`
		`h2f_dbg_apb_PENABLE`
		`h2f_dbg_apb_PRDATA[31..0]`
		`h2f_dbg_apb_PREADY`
		`h2f_dbg_apb_PSEL`
		`h2f_dbg_apb_PSLVERB`
		`h2f_dbg_apb_PWDATA[31..0]`
		`h2f_dbg_apb_PWRITE`

h2f_debug_apb_sideband	Conduit	`h2f_debug_apb_PCLKEN` `h2f_debug_apb_DBG_APB_DISABLE`

Related Information

CoreSight Debug and Trace
For more information about the interfaces described in this section, refer to the "CoreSight Debug and Trace" chapter in the Intel Stratix 10 Hard Processor System Technical Reference Manual.

2.2.1.4. Enable System Trace Macrocell (STM) Hardware Events

The system trace microcell hardware events interface allows logic in the FPGA to insert messages into the trace stream.

For more information about the **System Trace Macrocell Hardware Events** interface, refer to the "CoreSight Debug and Trace" chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

Turning on the **Enable System Trace Macrocell Hardware Events** option enables the **f2h_stm_hw_events** conduit, which is comprised of the single bus `f2h_stm_hwevents[42…0]`.

Related Information

CoreSight Debug and Trace
For more information about the interfaces described in this section, refer to the "CoreSight Debug and Trace" chapter in the Intel Stratix 10 Hard Processor System Technical Reference Manual.

2.2.1.5. Enable FPGA Cross Trigger Interface

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

For more information about the **FPGA Cross Trigger** interface, refer to the "CoreSight Debug and Trace" chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

If this interface must be connected to a Signal Tap II instance in the FPGA fabric, then it must be left disabled in Platform Designer. Turning on the **Enable FPGA Cross Trigger Interface** option enables the **h2f_cti** conduit, which is comprised of the following signals:

- `h2f_cti_trig_in [7..0]`
- `h2f_cti_trig_out_ack [7..0]`
- `h2f_cti_trig_out [7..0]`
- `h2f_cti_trig_in_ack [7..0]`

Related Information

CoreSight Debug and Trace
For more information about the interfaces described in this section, refer to the "CoreSight Debug and Trace" chapter in the Intel Stratix 10 Hard Processor System Technical Reference Manual.

2.2.1.6. Enable DDR ARM Trace Bus (ATB)

Turning on the **Enable DDR ARM Trace Bus** option enables the **ddr_atb_clock** clock input and **ddr_atb_reset** reset input interfaces.

Related Information

CoreSight Debug and Trace
For more information about the interfaces described in this section, refer to the "CoreSight Debug and Trace" chapter in the Intel Stratix 10 Hard Processor System Technical Reference Manual.

2.2.2. HPS-FPGA AXI Bridges

2.2.2.1. FPGA-to-HPS Slave Interface

The FPGA-to-HPS interface is:

- 128-bit width ACE-Lite compliant slave allowing FPGA masters to issue transactions to the HPS
- Configurable to be an AXI-4 compliant slave, using the **Interface Specification** dropdown

Since the FPGA-to-HPS interface is fixed-width:

- **Enable/Data Width** dropdown allows you to choose whether the interface is disabled (unused) or enabled and 128 bits wide.
- **Ready Latency pipeline** dropdown configures the flexible ready latency pipelining available in the FPGA fabric. This can assist with timing closure at the FPGA-to-HPS boundary and is configurable to depths of 0 (none), 1, 2, 3, or 4.
- **Bridge address width** is configurable from 20 bits to 37 bits, which allows the FPGA fabric to access the majority of the HPS address space. To facilitate masters in the FPGA logic with a smaller address width than the bridge in accessing the HPS address space, you can use the Intel Address Span Extender component.

For more information, refer to the "Using the Address Span Component Extender" chapter.

When this bridge is enabled, the interfaces **f2h_axi_slave**, **f2h_axi_clock**,and **f2h_axi_reset** are made available.

This interface allows the FPGA to access the majority of the HPS slaves. When configured as an ACE-lite slave, this interface provides a coherent memory interface. 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 Platform Designer system integration tool automatically generates adapter logic to connect AXI to Avalon-MM interfaces.

For more information, refer to *Features of the Intel Stratix 10 HPS-FPGA Bridge* in the "HPS-FPGA Bridges" chapter in the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

Related Information

- Using the Address Span Extender Component on page 23
- HPS-FPGA Bridges
  For more information, refer to the “HPS-FPGA Bridges” chapter in the Intel Stratix 10 Hard Processor System Technical Reference Manual.

2.2.2.2. HPS to FPGA AXI-4 Master Interface

The HPS-to-FPGA AXI-4 Master interface allows HPS masters to issue transactions to the FPGA fabric. You can use the:

- **Enable/Data Width** dropdown to configure this master interface's data widths to 32-, 64-, or 128-bit.
- **Ready Latency pipeline** dropdown configures the flexible ready latency pipelining available in the FPGA fabric. This can assist with timing closure at the FPGA-to-HPS boundary and is configurable to depths of 0 (none), 1, 2, 3, or 4.
- **Bridge address width** is configurable from 32 bits down to 20 bits. When this bridge is enabled, the interfaces **h2f_axi_master**, **h2f_axi_clock**, and **h2f_axi_reset** are made available.

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

2.2.2.3. Lightweight HPS to FPGA Master Interface

The lightweight HPS-to-FPGA interface, a low-bandwidth control interface, allows HPS masters to issue transactions to the FPGA fabric. The **Enable/Data Width** dropdown is thus limited to a fixed 32-bit data width. The **Ready Latency pipeline** drop-down configures the flexible ready latency pipelining available in the FPGA fabric. This can assist with timing closure at the FPGA-to-HPS boundary and is configurable to depths of 0 (none), 1, 2, 3, or 4. The **Bridge address width** is configurable to either 21 bits or 20 bits. When this bridge is enabled, the interfaces **h2f_lw_axi_master**, **h2f_lw_axi_clock**, and **h2f_lw_axi_reset** are made available.

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

2.2.2.4. FPGA-to-HPS SDRAM AXI-4 Slave Interface

The FPGA-to-HPS SDRAM interface is a group of three direct connections between the FPGA fabric and the HPS SDRAM Scheduler in the L3 SDRAM Interconnect. For each of the F2SDRAM interfaces, you can select between 32-, 64- or 128-bit data widths using the corresponding **enable/data width** dropdown. The **Ready Latency pipeline** dropdowns configure the flexible ready latency pipelining available in the FPGA fabric for each corresponding interface. This can assist with timing closure at the FPGA-to-HPS boundary and is configurable to depths of 0 (none), 1, 2, 3, or 4. The **Bridge address width** is configurable from 37 bits to 21 bits. 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.

2.2.3. HPS Boot Source

The **HPS SSBL Location** dropdown allows you to choose one of three sources for the HPS Second Stage Bootloader:

- Use the boot flash as used by the SDM
- Use HPS SD/MMC flash
- Use HPS NAND flash

2.2.4. DMA Controller Interface

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 by clicking on the dropdown in the **Enabled** column for the corresponding channel row. Each DMA peripheral request interface conduit **f2h_dma<n>** contains the following three signals, where <*n*> corresponds to a specific request interface enabled in Platform Designer:

- f2h_dma<n>_req—This signal is used to request burst transfer using the DMA
- f2h_dma<n>_single—This signal is used to request single word transfer using the DMA
- f2h_dma<n>_ack—This signal indicates the DMA acknowledgment upon requests from the FPGA

Note: FPGA DMA interfaces 6 and 7 are multiplexed with the **EMAC2 I2C DMA** interface.

2.2.5. Interrupts

The Interrupts section is divided into two subsections, **FPGA-to-HPS** and **HPS-to-FPGA**.

2.2.5.1. FPGA-to-HPS

Turning on the **Enable FPGA-to-HPS Interrupts** option configures 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 (mpu_periph_clk).

2.2.5.2. HPS-to-FPGA

Table 3. HPS-to-FPGA Interrupts Interface

The following table lists the available HPS to FPGA interrupt interfaces and the corresponding interfaces available when they are enabled.

Parameter Name	Parameter Description	Interface Name
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 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. EMAC must be enabled in Pin Mux Tab before enabling interrupt.	h2f_emac0_interrupt h2f_emac1_interrupt h2f_emac2_interrupt

continued...

Parameter Name	Parameter Description	Interface Name
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 I2C-EMAC Interrupts	Enable the HPS peripheral interrupt for I2CEMAC to be driven into the FPGA fabric	h2f_i2c_emac0_interrupt h2f_i2c_emac1_interrupt h2f_i2c_emac2_interrupt
Enable I2C Peripherals Interrupts	Enable the HPS peripheral interrupt for I2C0 to be driven into the FPGA fabric. The I2C must be enabled in the Pin Mux Tab before enabling interrupt.	h2f_i2c0_interrupt h2f_i2c1_interrupt
Enable L4 Timer Interrupts	Enables the HPS peripheral interrupt for L4TIMER to be driven into the FPGA fabric.	h2f_timer_l4sp_0_interrupt h2f_timer_l4sp_1_interrupt
Enable NAND Interrupts	Enables interface for the HPS NAND controller interrupt to the FPGA. The NAND IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_nand_interrupt
Enable SYS Timer Interrupts	Enables the HPS peripheral interrupt for SYSTIMER to be driven into the FPGA fabric.	h2f_timer_sys_0_interrupt h2f_timer_sys_1_interrupt
Enable SD/MMC Interrupts	Enables interface for the HPS SD/MMC controller interrupt to the FPGA. The SD/MMC IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_sdmmc_interrupt
Enable SPI Master Interrupts	Enables interface for the HPS SPI master controller interrupt to the FPGA. The SPI Master IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_spim0_interrupt h2f_spim1_interrupt
Enable SPI Slave Interrupts	Enables interface for the HPS SPI slave controller interrupt to the FPGA. The SPI IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_spis0_interrupt h2f_spis1_interrupt
Enable ECC/Parity_L1 Interrupts	Enables the HPS peripheral interrupt for ECC single and double bit error and L1 parity error to be driven into the FPGA fabric.	h2f_ecc_serr_interrupt h2f_ecc_derr_interrupt h2f_parity_l1_interrupt
Enable UART Interrupts	Enables interface for the HPS UART controller interrupt to the FPGA. The UART IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_uart0_interrupt h2f_uart1_interrupt
Enable USB Interrupts	Enables interface for the HPS USB controller interrupt to the FPGA. The USB IP Block must be enabled in Pin Mux Tab before enabling interrupt.	h2f_usb0_interrupt s2f_usb1_interrupt
Enable Watchdog Interrupts	Enables interface for the HPS watchdog interrupt to the FPGA	h2f_wdog0_interrupt h2f_wdog1_interrupt

2.3. HPS Clocks and Resets

HPS Clocks and Reset is the second of five tabs in the HPS component and is made up of three tabs: **Input Clocks**, **Internal Clocks and Output Clocks**, and **Resets**.

2.3.1. Input Clocks

The **Input Clocks** tab is comprised of three subsections: **External Clock Source**, **FPGA-to-HPS Clocks Source**, and **Peripheral FPGA Clocks**.

2.3.1.1. External Clock Source

The **EOSC clock frequency** field is used to specify the frequency of the input clock to the `hps_osc_clk` pin that drives the main HPS PLL.

For more information about the requirements for this clock, refer to the *Intel Stratix 10 Device Datasheet*.

Related Information

Intel Stratix 10 Device Datasheet

2.3.1.2. FPGA-to-HPS Clocks Source

Turning on the **Enable FPGA-to-HPS free clock** option enables the **f2h_free_clk** clock input. This is an alternative input to the main HPS PLL driven from the FPGA fabric instead of the dedicated `hps_osc_clk` pin. Turning on the **Enable FPGA-to-HPS free clock** option is subject to the same requirements as that pin.

For more information about the requirements for this clock, refer to the *Intel Stratix 10 Device Datasheet*.

Related Information

Intel Stratix 10 Device Datasheet

2.3.1.3. Peripheral FPGA Clocks

Parameter Name	Parameter Description
EMAC 0 (emac0_md_clk clock frequency)	If EMAC 0 peripheral is routed to FPGA, use the input field to specify EMAC 0 MDIO clock frequency
EMAC 0 (emac0_gtx_clk clock frequency)	If EMAC 0 peripheral is routed to FPGA, use the input field to specify EMAC 0 transmit clock frequency
EMAC 1 (emac1_md_clk clock frequency)	If EMAC 1 peripheral is routed to FPGA, use the input field to specify EMAC 1 MDIO clock frequency
EMAC 1 (emac1_gtx_clk clock frequency)	If EMAC 1 peripheral is routed to FPGA, use the input field to specify EMAC 1 transmit clock frequency
EMAC 2 (emac2_md_clk clock frequency)	If EMAC 2 peripheral is routed to FPGA, use the input field to specify EMAC 2 MDIO clock frequency
EMAC 2 (emac2_gtx_clk clock frequency)	If EMAC 2 peripheral is routed to FPGA, use the input field to specify EMAC 2 transmit clock frequency
SD/MMC (sdmmc_cclk)	If this peripheral pin multiplexing is configured to route to FPGA fabric, use the input field to specify the SD/MMC sdmmc_cclk clock frequency
SPIM 0 (spim0_sclk_out clock frequency)	If SPI master 0 peripheral is routed to FPGA, use the input 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 the input field to specify SPI master 1 output clock frequency

continued...

Parameter Name	Parameter Description
I^2C0 (i2c0_clk clock frequency)	If I^2C 0 peripheral is routed to FPGA, use the input 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 the input field to specify I^2C 1 output clock frequency
I2CEMAC0 (i2cemac0_clk)	If this peripheral pin multiplexing is configured to route to the FPGA fabric, use the input field to specify the I2CEMAC0 i2cemac0_clk clock frequency
I2CEMAC1 (i2cemac1_clk)	If this peripheral pin multiplexing is configured to route to the FPGA fabric, use the input field to specify the I2CEMAC1 i2cemac1_clk clock frequency
I2CEMAC2 (i2cemac2_clk)	If this peripheral pin multiplexing is configured to route to the FPGA fabric, use the input field to specify the I2CEMAC2 i2cemac2_clk clock frequency

2.3.2. Internal Clocks and Output Clocks

The **Internal Clocks and Output Clocks** tab is comprised of five subsections: **Main PLL Output Clocks – Desired Frequencies**, **HPS to FPGA User Clocks**, **HPS Peripheral Clocks – Desired Frequencies**, **Clock Sources**, and **PLL Report**.

2.3.2.1. Main PLL Output Clocks – Desired Frequencies

This section allows you to control the MPU clock frequency. The **Default MPU clock frequency** field displays the default maximum frequency for the MPU based on the device speed grade selected, and the input voltage selected in the **VCCL_HPS Value** dropdown. You may check the **Override default MPU clock frequency** box to manually enter a slower frequency than the default MPU clock frequency, in the **Custom MPU clock frequency** field.

For more information about the maximum MPU clock frequencies, refer to the *Intel Stratix 10 Device Datasheet*.

Related Information

Intel Stratix 10 Device Datasheet

2.3.2.2. HPS to FPGA User Clocks

Turning on the **Enable HPS-to-FPGA User0 clock** or **Enable HPS-to-FPGA User1 clock** option enables one of two available HPS PLL outputs into the FPGA. You can connect a user clock to logic that you instantiate in the FPGA. When you enable a HPS-to-FPGA user clock, you must manually enter its maximum frequency for timing analysis. The Timing Analyzer has no other information about how software running on the HPS configures the phase-locked loop (PLL) outputs. Both user clocks are driven from peripheral PLL.

2.3.2.3. HPS Peripheral Clocks – Desired Frequencies

The clock frequencies you provide in this section are reported in a Synopsys* Design Constraints File (.sdc) generated by Platform Designer. The `.sdc` file is referenced in the system `.qip` file when the system is generated.

- The **L3 clock frequency** allows you to configure the frequency of the L3 interconnect. For more information about the maximum frequency of this clock, refer to the *Intel Stratix 10 Device Datasheet*.
- The **L4 free clock frequency** dropdown displays the frequency of the free-running L4 clock.
- The **L4 main clock frequency** dropdown allows you to select the desired frequency of the L4 interconnect clock, which is input to the fast peripherals including DMA, SPIM, SPIS, and TCM.
- The **L4 peripheral slow clock frequency** dropdown allows you to select the desired frequency of the L4 interconnect input to the slow peripherals, including Timer, I2C, and UART.
- The **CoreSight clock frequency** dropdown allows you to select the desired clock frequency for the CoreSight trace and debug time stamp clock.
- The **CoreSight bus clock frequency** dropdown displays the default CoreSight bus clock frequency.
- The **CoreSight trace IO clock** dropdown allows you to select the frequency for the CoreSight trace I/Os. This is an independent clock and is configurable down to 50MHz for lower speed debuggers.
- The **Frequency for GPIO debouncer** field allows you to specify an input clock frequency to the GPIO controller to be used by the optional debounce circuitry. The external signal can be debounced to remove any spurious glitches that are less than one period of the debouncing clock. When input signals are debounced using this clock, the signals must be active for a minimum of two cycles to guarantee that they are registered.
- The **EMAC<n> clock frequency** dropdowns are enabled when the corresponding EMAC peripherals are enabled. These dropdowns allow you to select the reference clock for each EMAC, which must be either 50MHz or 250MHz

Related Information

Intel Stratix 10 Device Datasheet

2.3.2.4. Clock Sources

The drop-downs in this section control multiplexers in the HPS clock manager to select the source for the corresponding PLL or clock. Some of the drop-downs are enabled only when the corresponding peripherals are enabled. The FPGA to HPS Free clock is available as an option in these drop-downs when it is enabled on the **Input Clocks** tab.

Note: If you intend to use the FPGA to HPS free clock as the input to the `hps_osc_clk` pin, you must select that option for the **Main PLL reference clock source** and **Peripheral PLL reference clock source**.

2.3.2.5. PLL Report

This section is an informational section displaying the calculated parameters for the HPS PLLs and frequencies for the main and peripheral clocks.

2.3.3. Resets

- Turning on the **Enable HPS warm reset handshake signals** option enables 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. Turning on this option exposes the **h2f_warm_reset_handshake** conduit, which is comprised of the signals **h2f_pending_rst_req_n** and **f2h_pending_rst_ack_n**.
- Turning on the **Enable HPS-to-FPGA cold reset output** option exposes the **h2f_coldreset** reset output interface. This signal is asserted when the HPS undergoes a cold reset.
- Turning on the **Enable watchdog reset** option exposes the **h2f_watchdog_rst** reset output interface and is asserted when the HPS watchdog timers are triggered.
- The **How SDM handles HPS watchdog reset** dropdown provides an input to the compiled bitstream that directs the SDM to treat the HPS watchdog reset assertion as an **HPS Cold reset**, **HPS warm reset**, or **HPS Cold reset and trigger a remote update**.

2.4. SDRAM

The **SDRAM** tab is the third of five tabs in the HPS component that consists of a single option, **External Memory Interface for HPS Intel Stratix 10 FPGA IP**. This enables the HPS dedicated conduit to the Intel Stratix 10 External Memory Interface for HPS. This conduit cannot connect to any other External Memory Interface (EMIF) IP. Only the Intel Stratix 10 External Memory Interface for HPS Platform Designer IP should be used.

The HPS supports one memory interface implementing double data rate 3 (DDR3), double data rate 3 Low Voltage (DDR3L), and double data rate 4 (DDR4) protocols.

Related Information

- HPS EMIF Design Considerations
  For more information, refer to this section in the *Intel Stratix 10 SoC Device Design Guidelines*.
- External Memory Interface
  For more information, refer to this section in the *Intel FPGAs and Programmable Devices support web page.*.
- External Memory Interfaces Intel Stratix 10 FPGA IP User Guide
  For more information about External Memory Interface for HPS Intel Stratix 10 FPGA IP

2.5. I/O Delays

The **I/O Delays** tab is the fourth of five tabs in the HPS component that allows you to add an optional delay chain to the input or output of any of the 48 HPS dedicated I/O pins. Each dropdown allows you to select between the following options for the corresponding I/O pin:

- Zero_chain_dly—input or output signal bypasses the delay chain
- Chain_dly—input or output signal goes through the minimum delay chain path
- One_chain_dly to fifteen_chain_dly—input or output signal goes through between one to fifteen chain delays, in addition to the minimum delay chain path

For more information about the delay timings, refer to the *Intel Stratix 10 Device Datasheet*.

Related Information

Intel Stratix 10 Device Datasheet

2.6. Pin MUX and Peripherals

The **Pin MUX and Peripherals** tab contains two sub-tabs: **Auto-Place IP** and **Advanced**.

2.6.1. Auto-Place IP

The **Auto-Place IP** tab contains a list of HPS peripherals that can be enabled and either routed to the HPS I/Os or to the FPGA. You can enable the following types of peripherals:

- NAND Flash Controller
- SD/MMC Controller
- Ethernet Media Access Controller
- USB 2.0 OTG Controller
- I^2C Controller
- UART Controller
- SPI Master
- SPI Slave
- CoreSight Debug and Trace

For more information about each of these HPS peripherals, refer to the *Intel Stratix 10 Hard Processor System Technical Reference Manual*.

You can enable one or more instances of each peripheral type by using the dropdown menu next to each peripheral. When enabled, some peripherals also have mode settings specific to their functions. Once you have selected a peripheral, you must click the **Apply Selections** button in order to enable the selected peripherals. Clicking the **Apply Selections** button triggers the HPS component to do a best-effort automatic placement of the enabled peripheral signals to the HPS I/Os. This overrides any settings already chosen in the **Advanced** tab. The results of this placement becomes visible in the **I/O Selections** section on the right side of the **Auto-Place IP** tab. Any messages, such as failures to place a peripheral, appears in the message box in the **I/O Selections** section.

If the NAND, SD/MMC, or TRACE peripherals are enabled, there are further options to specify the desired bit width of the interface routed to the HPS I/Os. If any of the EMACs are enabled, the corresponding **Interface** and **PHY Options** dropdowns becomes available to specify the desired EMAC parameters.

The **Pin Mux Report** section details which physical pins of the device map to each HPS I/O location. In the **Emac ptp interface** section, there are options to turn on for each EMAC to enable the Precision Time Protocol (ptp) FPGA interface. These options are only applicable when an EMAC is routed to the HPS pins. When enabled, the signals `emac<n>_ptp_pps_o`, `emac<n>_ptp_aux_tx_trig_i`, `emac<n>_ptp_tstmp_data`, `emac<n>_ptp_tstmp_en`, as well as the **emac_ptp_ref_clock** clock input interface, are made available. When an EMAC is routed to the FPGA pins, these signals are automatically included in the **emac<n>** conduit.

Related Information

Intel Stratix 10 Hard Processor System Technical Reference Manual

2.6.2. Advanced

The **Advanced** tab is divided into two sub-tabs, **Advanced IP Placement** and **Advanced FPGA Placement**.

Advanced IP Placement

The **Advanced IP Placement** tab allows you to be more specific about the placement of each peripheral pin in the HPS dedicated I/O quadrant space. Each location has a pulldown selection menu where you can select which peripheral I/O to be routed to the pin location. Each pulldown menu corresponds to the inputs available to the pinmux at that location. Changes to a dropdown only become effective when the **Apply Selections** button is pressed. Changes in the **Advanced IP Placement** tab carry over to the **Auto-Place IP** tab. The **Pin Mux Report** and **EMAC ptp interface** sections are identical to those in the **Auto-Place IP** tab.

Advanced FPGA Placement

The **Advanced FPGA Placement** tab allows you to route specific peripherals to the FPGA, if those peripherals were enabled and allocated to the FPGA in the **Auto-Place IP** tab. Similar options for the SD/MMC, NAND, and TRACE bit-width allow you to specify how wide the interfaces should be when routed to the FPGA. Changes to a dropdown only become effective when the **Apply Selections** button is pressed. Changes in the **Advanced FPGA Placement** tab carry over to the **Auto-Place IP** tab. The **Pin Mux Report** and **EMAC ptp interface** sections are identical to those in the **Auto-Place IP** tab.

2.7. Generating and Compiling the HPS Component

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

1. Generate the design with Platform Designer. 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 Intel Quartus Prime project. *<qsys_system_name>.qip* is the Intel Quartus Prime IP File for the HPS component, generated by Platform Designer.

Note: Platform Designer generates pin assignments in the **.qip** file.

3. Perform analysis and synthesis with the Intel Quartus Prime software.
4. Compile the design with the Intel Quartus Prime software.
5. Optionally back-annotate the SDRAM pin assignments, to eliminate pin assignment warnings the next time you compile the design.

2.8. Using the Address Span Extender Component

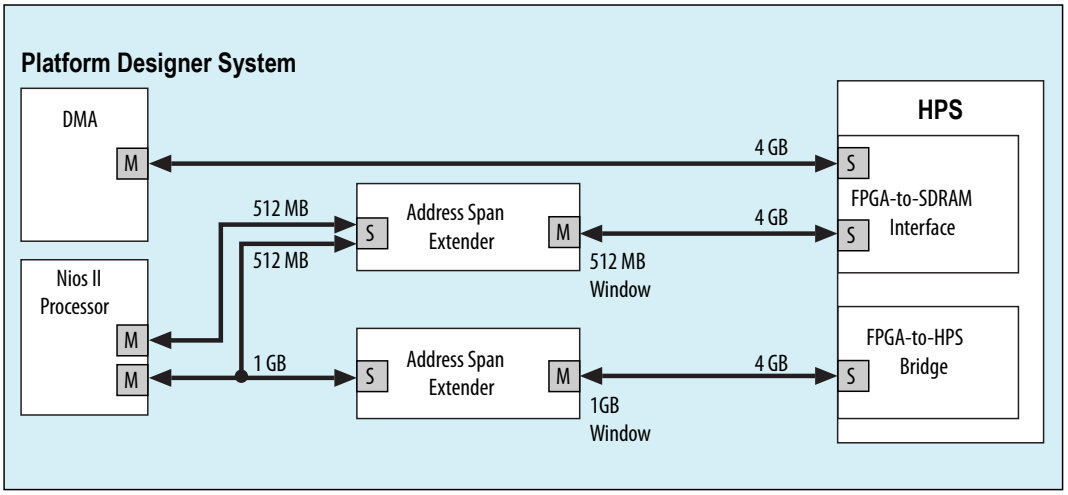
The FPGA-to-HPS bridge and FPGA-to-HPS SDRAM bridge memory-mapped interfaces can be configured to expose their entire address spaces to the FPGA fabric, 132GB and 128GB, respectively. The address span extender component provides a memory-mapped window into the address space that it masters. Using the address span extender, an FPGA master with a smaller address span can access the entire address space exposed by the FPGA bridge.

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.

In the example shown in the figure below, the bridges in the HPS component are configured for 32-bit wide addresses (4GB address span).

Figure 3. 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.

For more information about the Intel Span Extender, refer to the *Address Span Extender* section in the *Intel Quartus Prime Pro Edition User Guide: Platform Designer*.

Related Information

Intel Quartus Prime Pro Edition User Guide: Platform Designer

2.9. Configuring the HPS Component Revision History

Table 4. Document Revision History

Document Version	Changes
2018.11.30	• Merged the "Instantiating the Intel Stratix 10 HPS Component" and the " HPS Component Interfaces" sections into one section named "Configuring the HPS Component". • Updated figures in the following two sections to account for branding changes and Intel Quartus Prime release updates: — "Using the HPS Parameter Editor" — "FPGA Interfaces"
2017.11.06	Initial release

3. Simulating the Intel Stratix 10 HPS Component

Only Mentor Graphics* Bus Functional Models (BFM) are provided for the AXI interfaces; and the FPGA-to-SDRAM does not have BFM model support.

Related Information

- Simulation Flows on page 25
- Avalon Verification IP Suite User Guide
- Mentor Graphics Verification IP Altera Edition AMBA AXI3/AXI4 User Guide
  The Mentor Verification IP User guide provides details of the API and connection guidelines for the AXI3 and AXI4 BFMs.

3.1. Simulation Flows

Intel provides a functional register transfer level (RTL) simulation and a post–fitter gate–level simulation flow. Both simulation flows involve the following major steps, which is defined in the following sections:

1. Setting up the HPS component for simulation.
2. Generating the HPS simulation model in Platform Designer.
3. Running the simulation.

Related Information

Intel Quartus Prime Pro Edition User Guide: Third-party Simulation
Refer to this user guide for more information about simulation.

3.1.1. Setting Up the HPS Component for Simulation

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

1. Add the HPS component from the Platform Designer 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 or Avalon-MM master interface in another component in the system.

When you create your component, make sure the conduit interfaces have the correct role names and widths. Also make sure the conduit interfaces are opposite in direction to what is shown in the HPS Conduit Interfaces table.

3.1.1.1. HPS Conduit Interfaces Connecting to the FPGA

The following tables define the HPS Conduit interfaces that connect to the FPGA.

Table 5. h2f_warm_reset_handshake

Role Name	Direction	Width
h2f_pending_rst_req_n	Output	1
f2h_pending_rst_ack_n	Input	1

Table 6. h2f_gp

Role Name	Direction	Width
h2f_gp_in	Input	32
h2f_gp_out	Output	32

Table 7. h2f_mpu_events

Role Name	Direction	Width
h2f_mpu_eventi	Input	1
h2f_mpu_evento	Output	1
h2f_mpu_standbywfe	Output	4
h2f_mpu_standbywfi	Output	4

Table 8. f2h_dma0 to f2h_dma7

Role Name	Direction	Width
f2h_dma_req<0-7>_req	Input	1
f2h_dma_req<0-7>_single	Input	1
f2h_dma_req<0-7>_ack	Output	1

Table 9. h2f_debug_apb_sideband

Role Name	Direction	Width
h2f_dbg_apb_PCLKEN	Input	1
h2f_dbg_apb_DBG_APB_DISABLE	Input	1

Table 10. f2h_stm_hw_events

Role Name	Direction	Width
f2h_stm_hwevents	Input	43

Table 11. h2f_cti

Role Name	Direction	Width
h2f_cti_trig_in	Input	8
h2f_cti_trig_in_ack	Output	8
h2f_cti_trig_out	Output	8
h2f_cti_trig_out_ack	Input	8
h2f_cti_fpga_clk_en	Input	1

Table 12. h2f_tpiu

Role Name	Direction	Width
h2f_tpiu_clk_ctrl	Input	1
h2f_tpiu_data	Output	32

3.1.2. Generating the HPS Simulation Model in Platform Designer

The following steps outline how to generate the simulation model:

1. In Platform Designer, click **Generate HDL** under the Generate menu.
2. Choose between RTL and post–fit simulation

   For RTL simulation, perform the following steps:

   a. Set **Create simulation model** to Verilog.

   b. Click **Generate**.(1)

   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.(2)(3)
3. Click **Generate**.

Related Information

Intel Quartus Prime Pro Edition User Guide: Platform Designer
Refer to this user guide for more information about Platform Designer simulation.

3.1.3. Running the Simulations

The steps to run a simulation depend on whether you are running an RTL simulation or a post-fit simulation.

3.1.3.1. Running HPS RTL Simulation

Platform Designer generates scripts for several simulators that you can use to complete the simulation process, as listed in the following table.

Table 13. Platform Designer-Generated Scripts for Supported Simulators

Simulator	Script Name	Directory
Mentor Graphics Modelsim Intel Edition	**msim_setup.tcl**	*<project directory>/<Platform Designer design name>*/**simulation/mentor**
Cadence® NC-Sim	**ncsim_setup.sh**	*<project directory>/<Platform Designer design name>*/**simulation/cadence**

continued...

(1) VHDL is supported for HPS simulation and requires a mix language simulator. However, the BFMs always need to be in verilog. Custom components can be in VHDL.

(2) A **.bsf** file is only needed for schematic entry.

(3) This is not a requirement for simulation or implementation unless a schematic is used.

Simulator	Script Name	Directory
Synopsys VCS	**vcs_setup.sh**	*<project directory>/<Platform Designer design name>*/**simulation/synopsys/vcs**
Synopsys VCS-MX	**vcsmx_setup.sh**	*<project directory>/<Platform Designer design name>*/**simulation/synopsys/vcsmx**
Aldec® RivieraPro™	**rivierapro_setup.tcl**	*<project directory>/<Platform Designer design name>*/**simulation/aldec**

Related Information

- Avalon Verification IP Suite User Guide
- Mentor Graphics Verification IP Altera Edition AMBA AXI3/AXI4 User Guide
  The Mentor Verification IP User guide provides details of the API and connection guidelines for the AXI3 and AXI4 BFMs.

3.1.3.2. Running HPS Post-Fit Simulation

To run HPS post-fit simulation after successful Platform Designer generation, perform the following steps:

1. Add the generated synthesis file set to your Intel Quartus Prime project by performing the following steps:
   a. In the Intel Quartus Prime software, click **Settings** in the Assignments menu.
   b. In the **Settings** *<your Platform Designer system name>* dialog box, on the **Files** tab, browse to *<your project directory>/<your Platform Designer system name>***/synthesis/** and select *<your Platform Designer system name>***.qip**.
   c. Click **Open**. The **Select File** dialog box closes.
   d. Click **OK**. The **Settings** dialog box closes.
2. Optionally instantiate your HPS system as the top-level entity in your Intel Quartus Prime project.
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 Intel Quartus Prime EDA Netlist Writer, perform the following steps:
   a. Click **Start** in the Processing menu.
   b. Click **Start EDA Netlist Writer**.

Related Information

Intel Quartus Prime Pro Edition Help version 18.1

3.1.3.2.1. Post-Fit Simulation Files

Post-fit simulation is the simulation of the netlist generated from the original RTL design after it has been mapped, synthesized, and fit. The netlist represents the actual hardware and its connections as they appear in the FPGA. Intel Quartus Prime generates the netlist and can generate a Standard Delay Format (**.sdf**) file with the timing information for all connections. The simulation can be functional only (without the timing information) where all wires and gates take zero time, or it can be a timing simulation where the time for all transitions is based on the SDF information.

Post-fit simulation can serve a number of different purposes:

- Used to perform a dynamic verification of the timing of the design.
- Used to verify the functional correctness of either the design, the compilation flow (in particular, the fitter), or both.

This table uses the following symbols:

- *<ACDS install>* = Intel 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**

Table 14. Post-Fit Simulation Files

Library	Directory	File
Intel 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(4)	*<Device Sim Lib>/*	**altera_primitives.v** **220model.v**

continued...

Library	Directory	File
		sgate.v **altera_mf.v** **altera_lnsim.sv** **twentynm_atoms.v** **fourteennm_atoms.sv** **mentor/twentynm_atoms_ncrypt.sv** **mentor/fourteennm_atoms_ncrypt.sv**
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)

3.1.3.2.2. 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 . The design under test is the HPS component.
- *<HPS>* is the HPS component instance name that you use in your Platform Designer system.
- *<interface>* is the instance name of 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>***/***<Platform Designer design name>***/synthesis/submodules**.
- *<space>*—You must insert one space character after the interface instance name.
- *<BFM>* is the BFM instance name. To identify the BFM instance name, in *<ACDS install>***/ip/altera/hps/postfitter_simulation**, find the SystemVerilog file corresponding to the interface type that you are using. This SystemVerilog file contains the BFM instance name.

For example, a path for the Lightweight HPS-to-FPGA master interface hierarchy can 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 can 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 Intel Quartus Prime software.

(4) The device simulation library is not needed with Modelsim-Intel.

3.2. Clock and Reset Interfaces

3.2.1. Clock Interface

Platform Designer generates the clock source BFM for the FPGA-to-HPS alternate clock source.

Table 15. HPS Clock Input Interface Simulation Model

The Intel clock source BFM application programming interface (API) applies to the BFM listed in this table. Your Verilog interfaces use the same API.

Interface Name	BFM Instance Name
f2h_free_clk	f2h_free_clock_inst

Platform Designer 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 Platform Designer.

The HPS-to-FPGA debug APB interface generates a clock output to the FPGA, named `h2f_debug_apb_clock`. In simulation, the clock source BFM also represents this clock output's behavior.

Table 16. HPS Clock Output Interface Simulation Model

The Intel clock source BFM application programming interface (API) applies to all the BFMs listed in this table. Your Verilog interfaces use the same API.

Interface Name	BFM Instance Name
h2f_user0_clock	h2f_user0_clock_inst
h2f_user1_clock	h2f_user1_clock_inst

3.2.2. Reset Interface

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

Table 17. 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
h2f_warm_reset_handshake	h2f_warm_reset_handshake_inst	set_h2f_pending_rst_req_n() get_f2h_pending_rst_ack_n()

Table 18. HPS Reset Output Interface Simulation Model

The Intel 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 19. Configuration of Reset Source BFM for HPS Reset Output Interface

The HPS reset output interface is connected to a reset source BFM. Platform Designer configures the BFM as shown in the following table. The parameter value of the instantiated BFM is 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.

Related Information

Avalon Verification IP Suite User Guide

3.3. 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`. Platform Designer configures the BFM as shown in the following table. The BFM clock input is connected to `f2h_axi_clock` clock.

Table 20. Configuration of FPGA-to-HPS AXI Slave BFM

Parameter	Value
AXI Address Width	20 - 37
AXI Read Data Width	128
AXI Write Data Width	128
AXI ID Width	4

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

Related Information

Mentor Graphics Verification IP Altera Edition AMBA AXI3/AXI4 User Guide
The Mentor Verification IP User guide provides details of the API and connection guidelines for the AXI3 and AXI4 BFMs.

3.4. 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`. In Platform Designer, you can configure the HPS-to-FPGA interface with the following address, data, and ID widths. The BFM clock input is connected to `h2f_axi_clock` clock.

Table 21. Configuration of HPS-to-FPGA AXI Master BFM

Parameter	Value
AXI Address Width	32
AXI Read and Write Data Width	32, 64, or 128
AXI ID Width	4

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

Related Information

Mentor Graphics Verification IP Altera Edition AMBA AXI3/AXI4 User Guide
The Mentor Verification IP User guide provides details of the API and connection guidelines for the AXI3 and AXI4 BFMs.

3.5. 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`. Platform Designer configures the BFM as shown in the following table. The BFM clock input is connected to `h2f_lw_axi_clock` clock.

Table 22. Configuration of Lightweight HPS-to-FPGA AXI Master BFM

Parameter	Value
AXI Address Width	20 - 21
AXI Read and Write Data Width	32
AXI ID Width	4

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

3.6. HPS-to-FPGA MPU Event Interface

The HPS-to-FPGA MPU event interface is connected to an Intel 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 23. 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()`

Related Information

Avalon Verification IP Suite User Guide

3.7. Interrupts Interface

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

Table 24. FPGA-to-HPS Interrupts Interface Simulation Model

Interface Name	BFM Instance Name
f2h_irq0	f2h_irq0_inst
f2h_irq1	f2h_irq1_inst

The HPS-to-FPGA peripheral interfaces are connected to Intel conduit BFMs for simulation. When you enable the peripheral interrupt, the corresponding peripheral signal to the FPGA is exposed.

Table 25. HPS-to-FPGA Peripherals Interrupt Interface Simulation Model

Interface Name	BFM Instance Name
h2f_clkmgr_interrupt	h2f_clkmgr_interrupt_inst
h2f_dma_interrupt0	h2f_dma_interrupt0_inst
h2f_dma_interrupt1	h2f_dma_interrupt1_inst
h2f_dma_interrupt2	h2f_dma_interrupt2_inst
h2f_dma_interrupt3	h2f_dma_interrupt3_inst
h2f_dma_interrupt4	h2f_dma_interrupt4_inst
h2f_dma_interrupt5	h2f_dma_interrupt5_inst
h2f_dma_interrupt6	h2f_dma_interrupt6_inst
h2f_dma_interrupt7	h2f_dma_interrupt7_inst
h2f_dma_abort_interrupt	h2f_dma_abort_interrupt_inst
h2f_emac0_interrupt	h2f_emac0_interrupt_inst
h2f_emac1_interrupt	h2f_emac1_interrupt_inst
h2f_emac2_interrupt	h2f_emac2_interrupt_inst
h2f_gpio0_interrupt	h2f_gpio0_interrupt_inst
h2f_gpio1_interrupt	h2f_gpio1_interrupt_inst
h2f_gpio2_interrupt	h2f_gpio2_interrupt_inst
h2f_i2c_emac0_interrupt	h2f_i2c_emac0_interrupt_inst
h2f_i2c_emac1_interrupt	h2f_i2c_emac1_interrupt_inst
h2f_i2c_emac2_interrupt	h2f_i2c_emac2_interrupt_inst
h2f_i2c0_interrupt	h2f_i2c0_interrupt_inst
h2f_i2c1_interrupt	h2f_i2c1_interrupt_inst
h2f_l4sp0_interrupt	h2f_l4sp0_interrupt_inst
h2f_nand_interrupt	h2f_nand_interrupt_inst
h2f_sdmmc_interrupt	h2f_sdmmc_interrupt_inst
h2f_spim0_interrupt	h2f_spim0_interrupt_inst
h2f_spim1_interrupt	h2f_spim1_interrupt_inst

continued...

Interface Name	BFM Instance Name
h2f_spis0_interrupt	h2f_spis0_interrupt_inst
h2f_spis1_interrupt	h2f_spis1_interrupt_inst
h2f_usb0_interrupt	h2f_usb0_interrupt_inst
h2f_usb1_interrupt	h2f_usb1_interrupt_inst
h2f_wdog0_interrupt	h2f_wdog0_interrupt_inst
h2f_wdog1_interrupt	h2f_wdog1_interrupt_inst
h2f_wdog2_interrupt	h2f_wdog2_interrupt_inst
h2f_wdog3_interrupt	h2f_wdog3_interrupt_inst
h2f_uart0_interrupt	h2f_uart0_interrupt_inst
h2f_uart1_interrupt	h2f_uart1_interrupt_inst
h2f_ecc_serr_interrupt	h2f_ecc_serr_interrupt_inst
h2f_ecc_derr_interrupt	h2f_ecc_derr_interrupt_inst
h2f_timer_l4sp_0_interrupt	h2f_timer_l4sp_0_interrupt_inst
h2f_timer_l4sp_1_interrupt	h2f_timer_l4sp_1_interrupt_inst
h2f_timer_sys_0_interrupt	h2f_timer_sys_0_interrupt_inst
h2f_timer_sys_1_interrupt	h2f_timer_sys_1_interrupt_inst

3.8. HPS-to-FPGA Debug APB Interface

The HPS-to-FPGA debug APB interface is connected to an Intel 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 26. 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_sideband	h2f_debug_apb_sideband	get_h2f_dbg_apb_PCLKEN() get_h2f_dbg_apb_DBG_APB_DISABLE()	get_PCLKEN() get_DBG_APB_DISABLE()

3.9. FPGA-to-HPS System Trace Macrocell Hardware Event Interface

The FPGA-to-HPS STM hardware event interface is connected to an Intel conduit BFM for simulation. The following table lists the name of each interface, along with the 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 27. 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()

3.10. HPS-to-FPGA Cross-Trigger Interface

The HPS-to-FPGA cross-trigger interface is connected to an Intel 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 28. 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()	get_trig_in()
		set_h2f_cti_trig_in_ack()	set_trig_inack()
		set_h2f_cti_trig_out()	set_trig_out()
		get_h2f_cti_trig_out_ack()	get_trig_outack()
		get_h2f_cti_fpga_clk_en()	get_clk_en()

3.11. HPS-to-FPGA Trace Port Interface

The HPS-to-FPGA trace port interface is connected to an Intel 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 29. 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()	get_traceclk_ctl()
		set_h2f_tpiu_data()	set_trace_data()

3.12. FPGA-to-HPS DMA Handshake Interface

The FPGA-to-HPS DMA handshake interface is connected to an Intel 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 30. 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_dma0	f2h_dma0_inst	get_f2h_dma0_req() get_f2h_dma0_single() set_f2h_dma0_ack()	get_channel0_req() get_channel0_single() set_channel0_xx_ack()
f2h_dma1	f2h_dma1_inst	get_f2h_dma1_req() get_f2h_dma1_single() set_f2h_dma1_ack()	get_channel1_req() get_channel1_single() set_channel1_xx_ack()
f2h_dma2	f2h_dma2_inst	get_f2h_dma2_req() get_f2h_dma2_single() set_f2h_dma2_ack()	get_channel2_req() get_channel2_single() set_channel2_xx_ack()
f2h_dma3	f2h_dma3_inst	get_f2h_dma3_req() get_f2h_dma3_single() set_f2h_dma3_ack()	get_channel3_req() get_channel3_single() set_channel3_xx_ack()
f2h_dma4	f2h_dma4_inst	get_f2h_dma4_req() get_f2h_dma4_single() set_f2h_dma4_ack()	get_channel4_req() get_channel4_single() set_channel4_xx_ack()
f2h_dma5	f2h_dma5_inst	get_f2h_dma5_req() get_f2h_dma5_single() set_f2h_dma5_ack()	get_channel5_req() get_channel5_single() set_channel5_xx_ack()
f2h_dma6	f2h_dma6_inst	get_f2h_dma6_req() get_f2h_dma6_single() set_f2h_dma6_ack()	get_channel6_req() get_channel6_single() set_channel6_xx_ack()
f2h_dma7	f2h_dma7_inst	get_f2h_dma7_req() get_f2h_dma7_single() set_f2h_dma7_ack()	get_channel7_req() get_channel7_single() set_channel7_xx_ack()

Related Information

Avalon Verification IP Suite User Guide

3.13. General Purpose Input Interface

The general purpose input (GPI) interface is connected to an Intel 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 31. 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()

Interface Name	BFM Name	RTL simulation API function names
		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()

3.14. EMIF Conduit

Enables the HPS dedicated conduit to the Intel Stratix 10 External memory Interface for HPS. This conduit cannot connect to any other External memory Interface (EMIF). Only IP generated by the Intel Stratix 10 External memory Interface for HPS Platform Designer library should be used.

Table 32. EMIF Conduit Interface Simulation Model

Interface Name	BFM Instance Name	RTL Simulation API Function Names
`emif`	`emif_inst`	`emif_emif_to_hps` `emif_hps_to_emif`

3.15. Simulating the HPS Component Revision History

Table 33. Document Revision History

Document Version	Changes
2018.11.30	Updated the content for Intel Stratix 10
2017.11.06	Initial release