## Contents

1. Introduction to the Intel® Stratix® 10 Hard Processor System Component ..........4
   1.1. Cortex*-A53 MPCore* Processor .................................................................4
   1.2. CoreSight* Debug Components ..................................................................5
   1.3. Interconnect ...............................................................................................5
   1.4. FPGA Bridges ............................................................................................5
   1.5. Memory Controllers ..................................................................................6
   1.6. Support Peripherals ...................................................................................6
       1.6.1. Interface Peripherals ..............................................................7
       1.6.2. On-Chip Memories ..................................................................8
   1.7. Introduction to the HPS Component Revision History .................................8

2. Configuring the Intel Stratix 10 Hard Processor System Component ...............9
   2.1. Parameterizing the HPS Component ..........................................................9
   2.2. FPGA Interfaces .......................................................................................9
       2.2.1. General Interfaces ..................................................................10
       2.2.2. HPS-FPGA AXI Bridges .........................................................12
       2.2.3. HPS Boot Source ....................................................................14
       2.2.4. DMA Controller Interface .......................................................14
       2.2.5. Interrupts ...............................................................................15
   2.3. HPS Clocks and Resets ............................................................................16
       2.3.1. Input Clocks ..........................................................................17
       2.3.2. Internal Clocks and Output Clocks ...........................................18
       2.3.3. Resets .................................................................................20
   2.4. SDRAM ..................................................................................................20
   2.5. I/O Delays ...............................................................................................20
   2.6. Pin MUX and Peripherals .......................................................................21
       2.6.1. Auto-Place IP ........................................................................21
       2.6.2. Advanced .............................................................................22
   2.7. Generating and Compiling the HPS Component ........................................22
   2.8. Using the Address Span Extender Component ...........................................23
   2.9. Configuring the HPS Component Revision History ....................................24

3. Simulating the Intel Stratix 10 HPS Component ..............................................25
   3.1. Simulation Flows ....................................................................................25
       3.1.1. Setting Up the HPS Component for Simulation ..........................25
       3.1.2. Generating the HPS Simulation Model in Platform Designer .......27
       3.1.3. Running the Simulations ...............................................................27
   3.2. Clock and Reset Interfaces .......................................................................31
       3.2.1. Clock Interface ......................................................................31
       3.2.2. Reset Interface ...................................................................31
   3.3. FPGA-to-HPS AXI Slave Interface ..........................................................32
   3.4. HPS-to-FPGA AXI Master Interface .......................................................32
   3.5. Lightweight HPS-to-FPGA AXI Master Interface .....................................33
   3.6. HPS-to-FPGA MPU Event Interface .......................................................33
   3.7. Interrupts Interface ................................................................................33
   3.8. HPS-to-FPGA Debug APB Interface .......................................................35
   3.9. FPGA-to-HPS System Trace Macrocell Hardware Event Interface ...........36
   3.10. HPS-to-FPGA Cross-Trigger Interface ..................................................36
3.11. HPS-to-FPGA Trace Port Interface
3.12. FPGA-to-HPS DMA Handshake Interface
3.13. General Purpose Input Interface
3.14. EMIF Conduit
3.15. Simulating the HPS Component Revision History
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.


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

1.1. Cortex*-A53 MPCore* Processor

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

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.

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
1.5. Memory Controllers

The following lists the memory controller peripherals:

- NAND Flash Controller
- SD/MMC Controller

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

- **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*.

- **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*. 
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^2^C 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^2^C 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

1.7. Introduction to the HPS Component Revision History

Table 1. Document Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.11.30</td>
<td>Each of the folder tabs have been updated to reflect updates for Intel Stratix 10.</td>
</tr>
<tr>
<td>2018.10.12</td>
<td>Maintenance release</td>
</tr>
<tr>
<td>2017.11.06</td>
<td>Initial release</td>
</tr>
</tbody>
</table>
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.
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:
2. Configuring the Intel Stratix 10 Hard Processor System Component

S10-HPSCOMPONENT | 2018.11.30

- 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.


Turning on this option enables the following interfaces and signals:

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>Interface Type</th>
<th>Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_debug_apb_clock</td>
<td>Clock Input</td>
<td>h2f_dbg_apb_clk</td>
</tr>
<tr>
<td>h2f_debug_apb_reset</td>
<td>Reset Output</td>
<td>h2f_dbg_apb_rst_n</td>
</tr>
<tr>
<td>h2f_debug_apb</td>
<td>APB Master</td>
<td>h2f_dbg_apb_PADDR[14..0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_PADDR31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_PENABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_PRDATA[31..0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_PREADY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_PSEL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_PSLVERB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_PRDATA[31..0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_PWRITE</td>
</tr>
<tr>
<td>h2f_debug_apb_sideband</td>
<td>Conduit</td>
<td>h2f_dbg_apb_PCLKEN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_dbg_apb_DBG_APB_DISABLE</td>
</tr>
</tbody>
</table>

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.


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].
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]

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.

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.


**Related Information**

- Using the Address Span Extender Component on page 23
- HPS-FPGA Bridges

### 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 drop-downs 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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Description</th>
<th>Interface Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Clock Peripheral Interrupts</td>
<td>Enables interface for HPS clock manager and MPU wake-up interrupt signals to the FPGA</td>
<td><code>h2f_clkmgr_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_mpuwakeup_interrupt</code></td>
</tr>
<tr>
<td>Enable DMA Interrupts</td>
<td>Enables interface for HPS DMA channels interrupt and DMA abort interrupt to the FPGA</td>
<td><code>h2f_dma_interrupt0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_dma_interrupt1</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_dma_interrupt2</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_dma_interrupt3</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_dma_interrupt4</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_dma_interrupt5</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_dma_interrupt6</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_dma_interrupt7</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_dma_abort_interrupt</code></td>
</tr>
<tr>
<td>Enable EMAC Interrupts</td>
<td>Enables interface for HPS Ethernet MAC controller interrupt to the FPGA. EMAC must be enabled in Pin Mux Tab before enabling interrupt.</td>
<td><code>h2f_emac0_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_emac1_interrupt</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>h2f_emac2_interrupt</code></td>
</tr>
</tbody>
</table>

*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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMAC 0 (emac0_md_clk clock frequency)</td>
<td>If EMAC 0 peripheral is routed to FPGA, use the input field to specify EMAC 0 MDIO clock frequency</td>
</tr>
<tr>
<td>EMAC 0 (emac0_gtx_clk clock frequency)</td>
<td>If EMAC 0 peripheral is routed to FPGA, use the input field to specify EMAC 0 transmit clock frequency</td>
</tr>
<tr>
<td>EMAC 1 (emac1_md_clk clock frequency)</td>
<td>If EMAC 1 peripheral is routed to FPGA, use the input field to specify EMAC 1 MDIO clock frequency</td>
</tr>
<tr>
<td>EMAC 1 (emac1_gtx_clk clock frequency)</td>
<td>If EMAC 1 peripheral is routed to FPGA, use the input field to specify EMAC 1 transmit clock frequency</td>
</tr>
<tr>
<td>EMAC 2 (emac2_md_clk clock frequency)</td>
<td>If EMAC 2 peripheral is routed to FPGA, use the input field to specify EMAC 2 MDIO clock frequency</td>
</tr>
<tr>
<td>EMAC 2 (emac2_gtx_clk clock frequency)</td>
<td>If EMAC 2 peripheral is routed to FPGA, use the input field to specify EMAC 2 transmit clock frequency</td>
</tr>
<tr>
<td>SD/MMC (sdmmc_cclk)</td>
<td>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</td>
</tr>
<tr>
<td>SPIM 0 (spim0_sclk_out clock frequency)</td>
<td>If SPI master 0 peripheral is routed to FPGA, use the input field to specify SPI master 0 output clock frequency</td>
</tr>
<tr>
<td>SPIM 1 (spim1_sclk_out clock frequency)</td>
<td>If SPI master 1 peripheral is routed to FPGA, use the input field to specify SPI master 1 output clock frequency</td>
</tr>
</tbody>
</table>

continued...
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, SPI2S, 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
- I²C 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_tstamp_data`, `emac<n>_ptp_tstamp_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**


### 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.

**Related Information**


### 2.9. Configuring the HPS Component Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 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

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**
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

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Direction</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_pending_rst_req_n</td>
<td>Output</td>
<td>1</td>
</tr>
<tr>
<td>f2h_pending_rst_ack_n</td>
<td>Input</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 6.  h2f_gp

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Direction</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_gp_in</td>
<td>Input</td>
<td>32</td>
</tr>
<tr>
<td>h2f_gp_out</td>
<td>Output</td>
<td>32</td>
</tr>
</tbody>
</table>

### Table 7.  h2f_mpu_events

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Direction</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_mpu_eventi</td>
<td>Input</td>
<td>1</td>
</tr>
<tr>
<td>h2f_mpu_evento</td>
<td>Output</td>
<td>1</td>
</tr>
<tr>
<td>h2f_mpu_standbywfe</td>
<td>Output</td>
<td>4</td>
</tr>
<tr>
<td>h2f_mpu_standbywfi</td>
<td>Output</td>
<td>4</td>
</tr>
</tbody>
</table>

### Table 8.  f2h_dma0 to f2h_dma7

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Direction</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>f2h_dma_req&lt;0-7&gt;_req</td>
<td>Input</td>
<td>1</td>
</tr>
<tr>
<td>f2h_dma_req&lt;0-7&gt;_single</td>
<td>Input</td>
<td>1</td>
</tr>
<tr>
<td>f2h_dma_req&lt;0-7&gt;_ack</td>
<td>Output</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 9.  h2f_debug_apb_sideband

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Direction</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_dbg_apb_PCLKEN</td>
<td>Input</td>
<td>1</td>
</tr>
<tr>
<td>h2f_dbg_apb_DBG_APB_DISABLE</td>
<td>Input</td>
<td>1</td>
</tr>
</tbody>
</table>

### Table 10.  f2h_stm_hw_events

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Direction</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>f2h_stm_hwevents</td>
<td>Input</td>
<td>43</td>
</tr>
</tbody>
</table>

### Table 11.  h2f_cti

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Direction</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_cti_trig_in</td>
<td>Input</td>
<td>8</td>
</tr>
<tr>
<td>h2f_cti_trig_in_ack</td>
<td>Output</td>
<td>8</td>
</tr>
<tr>
<td>h2f_cti_trig_out</td>
<td>Output</td>
<td>8</td>
</tr>
<tr>
<td>h2f_cti_trig_out_ack</td>
<td>Input</td>
<td>8</td>
</tr>
<tr>
<td>h2f_cti_fpga_clk_en</td>
<td>Input</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 12. h2f_tpiu

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Direction</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_tpiu_clk_ctrl</td>
<td>Input</td>
<td>1</td>
</tr>
<tr>
<td>h2f_tpiu_data</td>
<td>Output</td>
<td>32</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Script Name</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mentor Graphics Modelsim Intel Edition</td>
<td>msim_setup.tcl</td>
<td>&lt;project directory&gt;/&lt;Platform Designer design name&gt;/simulation/mentor</td>
</tr>
<tr>
<td>Cadence® NC-Sim</td>
<td>ncsim_setup.sh</td>
<td>&lt;project directory&gt;/&lt;Platform Designer design name&gt;/simulation/cadence</td>
</tr>
</tbody>
</table>

(1) VHDL is supported for HPS simulation and requires a mix language simulator. However, the BFM s 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
  
  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>`/<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**.
3. Simulating the Intel Stratix 10 HPS Component

S10-HPSCOMPONENT | 2018.11.30

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

<table>
<thead>
<tr>
<th>Library</th>
<th>Directory</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Verification IP Library</td>
<td>&lt;Avalon Verification IP&gt;/</td>
<td>verbosity_pkg.sv</td>
</tr>
<tr>
<td></td>
<td>lib/</td>
<td>avalon_mm_pkg.sv</td>
</tr>
<tr>
<td></td>
<td></td>
<td>avalon_utilities_pkg.sv</td>
</tr>
<tr>
<td>Avalon Clock Source BFM</td>
<td>&lt;Avalon Verification IP&gt;/</td>
<td>altera_avalon_clock_source.sv</td>
</tr>
<tr>
<td></td>
<td>altera_avalon_clock_source/</td>
<td></td>
</tr>
<tr>
<td>Avalon Reset Source BFM</td>
<td>&lt;Avalon Verification IP&gt;/</td>
<td>altera_avalon_reset_source.sv</td>
</tr>
<tr>
<td></td>
<td>altera_avalon_reset_source/</td>
<td></td>
</tr>
<tr>
<td>Avalon MM Slave BFM</td>
<td>&lt;Avalon Verification IP&gt;/</td>
<td>altera_avalon_mm_slave_bfm.sv</td>
</tr>
<tr>
<td></td>
<td>altera_avalon_mm_slave_bfm/</td>
<td></td>
</tr>
<tr>
<td>Avalon Interrupt Sink BFM</td>
<td>&lt;Avalon Verification IP&gt;/</td>
<td>altera_avalon_interrupt_sink.sv</td>
</tr>
<tr>
<td></td>
<td>altera_avalon_interrupt_sink/</td>
<td></td>
</tr>
<tr>
<td>Mentor AXI Verification IP</td>
<td>&lt;AXI Verification IP&gt;/</td>
<td>questa_mvc_svapi.svh</td>
</tr>
<tr>
<td>Library</td>
<td>common/</td>
<td></td>
</tr>
<tr>
<td>Mentor AXI3 BFM</td>
<td>&lt;AXI Verification IP&gt;/axi3/</td>
<td>mgc_common_axi.sv</td>
</tr>
<tr>
<td></td>
<td>axi3/bfm/</td>
<td>mgc_axi_master.sv</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mgc_axi_slave.sv</td>
</tr>
<tr>
<td>HPS Post-Fit Simulation</td>
<td>&lt;HPS Post-fit Sim&gt;/</td>
<td>All the files in the directory</td>
</tr>
<tr>
<td>Library</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device Simulation Library(4)</td>
<td>&lt;Device Sim Lib&gt;/</td>
<td>altera_primitives.v</td>
</tr>
<tr>
<td></td>
<td></td>
<td>220model.v</td>
</tr>
</tbody>
</table>

continued...
<table>
<thead>
<tr>
<th>Library</th>
<th>Directory</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA Netlist Writer Generated Post-Fit Simulation Model</td>
<td><code>&lt;User project directory&gt;/</code></td>
<td><code>*.vo</code>&lt;br&gt;<code>*.vho</code>&lt;br&gt;(Mixed-language simulator is needed for Verilog HDL and VHDL mixed design)</td>
</tr>
<tr>
<td>User testbench files</td>
<td><code>&lt;User project directory&gt;/</code></td>
<td><code>*.v</code>&lt;br&gt;<code>*.sv</code>&lt;br&gt;<code>*.vhd</code>&lt;br&gt;(Mixed-language simulator is needed for Verilog HDL and VHDL mixed design)</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>f2h_free_clk</td>
<td>f2h_free_clock_inst</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_user0_clock</td>
<td>h2f_user0_clock_inst</td>
</tr>
<tr>
<td>h2f_user1_clock</td>
<td>h2f_user1_clock_inst</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
<th>API Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_warm_reset_handshake</td>
<td>h2f_warm_reset_handshake_inst</td>
<td>set_h2f_pending_rst_req_n()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_pending_rst_ack_n()</td>
</tr>
</tbody>
</table>

Table 18. **HPS Reset Output Interface Simulation Model**
The Intel reset source BFM application programming interface applies to all the BFMs listed.

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_reset</td>
<td>h2f_reset_inst</td>
</tr>
<tr>
<td>h2f_cold_reset</td>
<td>h2f_cold_reset_inst</td>
</tr>
<tr>
<td>h2f_debug_apb_reset</td>
<td>h2f_debug_apb_reset_inst</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>BFM Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assert reset high</td>
<td>Off</td>
<td>This parameter is off, specifying an active-low reset signal from the BFM.</td>
</tr>
<tr>
<td>Cycles of initial reset</td>
<td>0</td>
<td>This parameter is 0, specifying that the BFM does not assert the reset signal automatically.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI Address Width</td>
<td>20 - 37</td>
</tr>
<tr>
<td>AXI Read Data Width</td>
<td>128</td>
</tr>
<tr>
<td>AXI Write Data Width</td>
<td>128</td>
</tr>
<tr>
<td>AXI ID Width</td>
<td>4</td>
</tr>
</tbody>
</table>

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

Related Information
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI Address Width</td>
<td>32</td>
</tr>
<tr>
<td>AXI Read and Write Data Width</td>
<td>32, 64, or 128</td>
</tr>
<tr>
<td>AXI ID Width</td>
<td>4</td>
</tr>
</tbody>
</table>
You control and monitor the AXI master BFM by using the BFM API.

Related Information

The Mentor Verification IP User guide provides details of the API and connection guidelines for the AXI3 and AXI4 BFM.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI Address Width</td>
<td>20 - 21</td>
</tr>
<tr>
<td>AXI Read and Write Data Width</td>
<td>32</td>
</tr>
<tr>
<td>AXI ID Width</td>
<td>4</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
<th>RTL Simulation API Function Names</th>
<th>Post-Fit Simulation API Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_mpu_events</td>
<td>h2f_mpu_events_inst</td>
<td>get_h2f_mpu_eventi() set_h2f_mpu_evento() set_h2f_mpu_standbywfe() set_h2f_mpu_standbywfl()</td>
<td>get_eventi() set_evento() set_standbywfe() set_standbywfl()</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>f2h_irq0</td>
<td>f2h_irq0_inst</td>
</tr>
<tr>
<td>f2h_irq1</td>
<td>f2h_irq1_inst</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_clkmgr_interrupt</td>
<td>h2f_clkmgr_interrupt_inst</td>
</tr>
<tr>
<td>h2f_dma_interrupt0</td>
<td>h2f_dma_interrupt0_inst</td>
</tr>
<tr>
<td>h2f_dma_interrupt1</td>
<td>h2f_dma_interrupt1_inst</td>
</tr>
<tr>
<td>h2f_dma_interrupt2</td>
<td>h2f_dma_interrupt2_inst</td>
</tr>
<tr>
<td>h2f_dma_interrupt3</td>
<td>h2f_dma_interrupt3_inst</td>
</tr>
<tr>
<td>h2f_dma_interrupt4</td>
<td>h2f_dma_interrupt4_inst</td>
</tr>
<tr>
<td>h2f_dma_interrupt5</td>
<td>h2f_dma_interrupt5_inst</td>
</tr>
<tr>
<td>h2f_dma_interrupt6</td>
<td>h2f_dma_interrupt6_inst</td>
</tr>
<tr>
<td>h2f_dma_interrupt7</td>
<td>h2f_dma_interrupt7_inst</td>
</tr>
<tr>
<td>h2f_dma_abort_interrupt</td>
<td>h2f_dma_abort_interrupt_inst</td>
</tr>
<tr>
<td>h2f_emac0_interrupt</td>
<td>h2f_emac0_interrupt_inst</td>
</tr>
<tr>
<td>h2f_emac1_interrupt</td>
<td>h2f_emac1_interrupt_inst</td>
</tr>
<tr>
<td>h2f_emac2_interrupt</td>
<td>h2f_emac2_interrupt_inst</td>
</tr>
<tr>
<td>h2f_gpio0_interrupt</td>
<td>h2f_gpio0_interrupt_inst</td>
</tr>
<tr>
<td>h2f_gpio1_interrupt</td>
<td>h2f_gpio1_interrupt_inst</td>
</tr>
<tr>
<td>h2f_gpio2_interrupt</td>
<td>h2f_gpio2_interrupt_inst</td>
</tr>
<tr>
<td>h2f_i2c_emac0_interrupt</td>
<td>h2f_i2c_emac0_interrupt_inst</td>
</tr>
<tr>
<td>h2f_i2c_emac1_interrupt</td>
<td>h2f_i2c_emac1_interrupt_inst</td>
</tr>
<tr>
<td>h2f_i2c_emac2_interrupt</td>
<td>h2f_i2c_emac2_interrupt_inst</td>
</tr>
<tr>
<td>h2f_i2c0_interrupt</td>
<td>h2f_i2c0_interrupt_inst</td>
</tr>
<tr>
<td>h2f_i2c1_interrupt</td>
<td>h2f_i2c1_interrupt_inst</td>
</tr>
<tr>
<td>h2f_14sp0_interrupt</td>
<td>h2f_14sp0_interrupt_inst</td>
</tr>
<tr>
<td>h2f_nand_interrupt</td>
<td>h2f_nand_interrupt_inst</td>
</tr>
<tr>
<td>h2f_sdmmc_interrupt</td>
<td>h2f_sdmmc_interrupt_inst</td>
</tr>
<tr>
<td>h2f_spim0_interrupt</td>
<td>h2f_spim0_interrupt_inst</td>
</tr>
<tr>
<td>h2f_spim1_interrupt</td>
<td>h2f_spim1_interrupt_inst</td>
</tr>
</tbody>
</table>

continued...
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

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Name</th>
<th>RTL Simulation API Function Names</th>
<th>Post-Fit Simulation API Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_debug_apb</td>
<td>h2f_debug_apb</td>
<td>set_h2f_dbg_apb_PADDR()</td>
<td>set_PADDR()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_h2f_dbg_apb_PADDR_31()</td>
<td>set_PADDR_31()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_h2f_dbg_apb_PENABLE()</td>
<td>set_PENABLE()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_h2f_dbg_apb_PRDATA()</td>
<td>get_PRDATA()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_h2f_dbg_apb_PREADY()</td>
<td>get_PREADY()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_h2f_dbg_apb_PSEL()</td>
<td>set_PSEL()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_h2f_dbg_apb_PSLVERR()</td>
<td>get_PSLVERR()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_h2f_dbg_apb_PWDATA()</td>
<td>set_PWDATA()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_h2f_dbg_apb_PWRITE()</td>
<td>set_PWRITE()</td>
</tr>
<tr>
<td>h2f_debug_apb_sideband</td>
<td>h2f_debug_apb_sideband</td>
<td>get_h2f_dbg_apb_PCLKEN()</td>
<td>get_PCLKEN()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_h2f_dbg_apb_DBG_APB_DISABLE()</td>
<td>get_DBG_APB_DISABLE()</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Name</th>
<th>RTL Simulation API Function Name</th>
<th>Post-Fit Simulation API Function Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>f2h_stm_hw_events</td>
<td>f2h_stm_hw_events_inst</td>
<td>get_f2h_stm_hwevents()</td>
<td>get_stm_events()</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Name</th>
<th>RTL Simulation API Function Names</th>
<th>Post-Fit Simulation API Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_cti</td>
<td>h2f_cti_inst</td>
<td>get_h2f_cti_trig_in()</td>
<td>get_trig_in()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_h2f_cti_trig_in_ack()</td>
<td>set_trig_inack()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_h2f_cti_trig_out()</td>
<td>set_trig_out()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_h2f_cti_trig_out_ack()</td>
<td>get_trig_outack()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_h2f_cti_fpga_clk_en()</td>
<td>get_clk_en()</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Name</th>
<th>RTL Simulation API Function Names</th>
<th>Post-Fit Simulation API Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2f_tpiu</td>
<td>h2f_tpiu_inst</td>
<td>get_h2f_tpiu_clk_ctl()</td>
<td>get_traceclk_ctl()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_h2f_tpiu_data()</td>
<td>set_trace_data()</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
<th>RTL Simulation API Function Names</th>
<th>Post-Fit Simulation API Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>f2h_dma0</td>
<td>f2h_dma0_inst</td>
<td>get_f2h_dma0_req()</td>
<td>get_channel0_req()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_dma0_single()</td>
<td>get_channel0_single()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_f2h_dma0_ack()</td>
<td>set_channel0_xx_ack()</td>
</tr>
<tr>
<td>f2h_dma1</td>
<td>f2h_dma1_inst</td>
<td>get_f2h_dma1_req()</td>
<td>get_channel1_req()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_dma1_single()</td>
<td>get_channel1_single()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_f2h_dma1_ack()</td>
<td>set_channel1_xx_ack()</td>
</tr>
<tr>
<td>f2h_dma2</td>
<td>f2h_dma2_inst</td>
<td>get_f2h_dma2_req()</td>
<td>get_channel2_req()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_dma2_single()</td>
<td>get_channel2_single()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_f2h_dma2_ack()</td>
<td>set_channel2_xx_ack()</td>
</tr>
<tr>
<td>f2h_dma3</td>
<td>f2h_dma3_inst</td>
<td>get_f2h_dma3_req()</td>
<td>get_channel3_req()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_dma3_single()</td>
<td>get_channel3_single()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_f2h_dma3_ack()</td>
<td>set_channel3_xx_ack()</td>
</tr>
<tr>
<td>f2h_dma4</td>
<td>f2h_dma4_inst</td>
<td>get_f2h_dma4_req()</td>
<td>get_channel4_req()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_dma4_single()</td>
<td>get_channel4_single()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_f2h_dma4_ack()</td>
<td>set_channel4_xx_ack()</td>
</tr>
<tr>
<td>f2h_dma5</td>
<td>f2h_dma5_inst</td>
<td>get_f2h_dma5_req()</td>
<td>get_channel5_req()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_dma5_single()</td>
<td>get_channel5_single()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_f2h_dma5_ack()</td>
<td>set_channel5_xx_ack()</td>
</tr>
<tr>
<td>f2h_dma6</td>
<td>f2h_dma6_inst</td>
<td>get_f2h_dma6_req()</td>
<td>get_channel6_req()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_dma6_single()</td>
<td>get_channel6_single()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_f2h_dma6_ack()</td>
<td>set_channel6_xx_ack()</td>
</tr>
<tr>
<td>f2h_dma7</td>
<td>f2h_dma7_inst</td>
<td>get_f2h_dma7_req()</td>
<td>get_channel7_req()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>get_f2h_dma7_single()</td>
<td>get_channel7_single()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set_f2h_dma7_ack()</td>
<td>set_channel7_xx_ack()</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Name</th>
<th>RTL simulation API function names</th>
</tr>
</thead>
<tbody>
<tr>
<td>hps_io</td>
<td>hps_io_inst</td>
<td>get_hps_io_gpio_inst_HLP0() get_hps_io_gpio_inst_HLP1() get_hps_io_gpio_inst_HLP2() get_hps_io_gpio_inst_HLP3() get_hps_io_gpio_inst_HLP4() get_hps_io_gpio_inst_HLP5() get_hps_io_gpio_inst_HLP6()</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>BFM Instance Name</th>
<th>RTL Simulation API Function Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>emif</td>
<td>emif_inst</td>
<td>emif_emif_to_hps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>emif_hps_to_emif</td>
</tr>
</tbody>
</table>

3.15. Simulating the HPS Component Revision History

Table 33. Document Revision History

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018.11.30</td>
<td>Updated the content for Intel Stratix 10</td>
</tr>
<tr>
<td>2017.11.06</td>
<td>Initial release</td>
</tr>
</tbody>
</table>