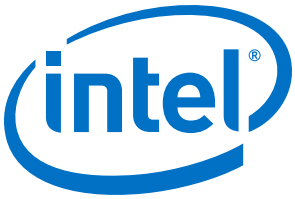


# H-tile Hard IP for Ethernet Intel® Stratix® 10 FPGA IP Design Example User Guide

Updated for Intel® Quartus® Prime Design Suite: **20.2**

IP Version: **19.3.0**


Subscribe


Send Feedback

**UG-20122 | 2020.06.22**

Latest document on the web: **PDF** | **HTML**

# Contents

**1. Quick Start Guide....3**
1.1. Directory Structure....4
1.2. Generating the Design....5
1.3. Simulating the H-Tile Hard IP for Ethernet Intel FPGA Design Example Testbench....7
1.4. Compiling the Compilation-Only Project....8
1.5. Compiling and Configuring the Design Example in Hardware....8
1.6. Testing the H-Tile Hard IP for Ethernet Intel FPGA IP Hardware Design Example....9
1.6.1. Testing the Hardware Design Example using Ethernet Link Inspector....14

**2. Design Example Description....15**
2.1. H-Tile Hard IP for Ethernet Intel FPGA MAC + PCS Simulation Design Example....16
2.2. H-Tile Hard IP for Ethernet Intel FPGA PCS Only Simulation Design Example....18
2.3. H-Tile Hard IP for Ethernet Intel FPGA OTN Simulation Design Example....19
2.4. H-Tile Hard IP for Ethernet Intel FPGA FlexE Simulation Design Example....21
2.5. Hardware Design Example Components....23
2.6. Design Example Interface Signals....25
2.7. H-Tile Hard IP for Ethernet Intel FPGA Design Example Registers....25

**3. H-tile Hard IP for Ethernet Intel Stratix 10 FPGA IP Design Example User Guide Document Archives....27**

**4. Document Revision History for the H-tile Hard IP for Ethernet Intel Stratix 10 FPGA IP Design Example User Guide....28**

# 1. Quick Start Guide

The H-Tile Hard IP for Ethernet Intel® FPGA IP core provides a simulation testbench and a hardware design example that supports compilation and hardware testing. When you generate the design example, the parameter editor automatically creates the files necessary to simulate, compile, and test the design in hardware.

In addition, you can download the compiled hardware design to the Intel Stratix® 10 MX FPGA Development Kit. Intel provides a compilation-only example project that you can use to quickly estimate IP core area and timing.

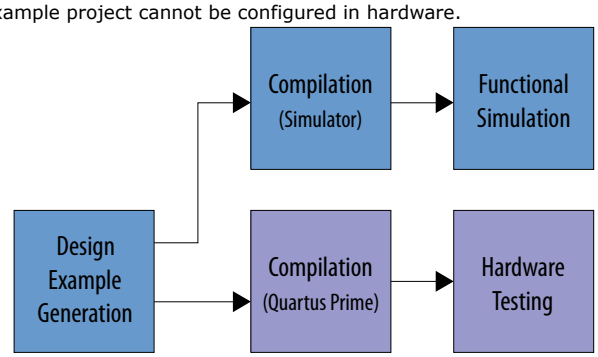
You can generate the design example H-Tile Hard IP for Ethernet Intel FPGA IP core variation:

- 100 Gbps MAC+PCS (supports simulation testbench, compilation-only example project, and hardware design example)
- 100 Gbps PCS only (supports simulation testbench and compilation-only example project, and hardware design example)
- 100 Gbps OTN (supports simulation testbench and compilation-only example project)
- 100 Gbps FlexE (supports simulation testbench and compilation-only example project)

*Note:* The H-Tile Hard IP for Ethernet Intel FPGA IP provides preliminary support for the OTN feature. For further inquiries, contact your nearest Intel sales representative or file an Intel Premier Support (IPS) case on https://www.intel.com/content/www/us/en/programmable/my-intel/mal-home.html.

**Figure 1. Development Steps for the Design Example**

Future releases of the IP core also provide a hardware design example you can compile and test in hardware. The compilation-only example project cannot be configured in hardware.



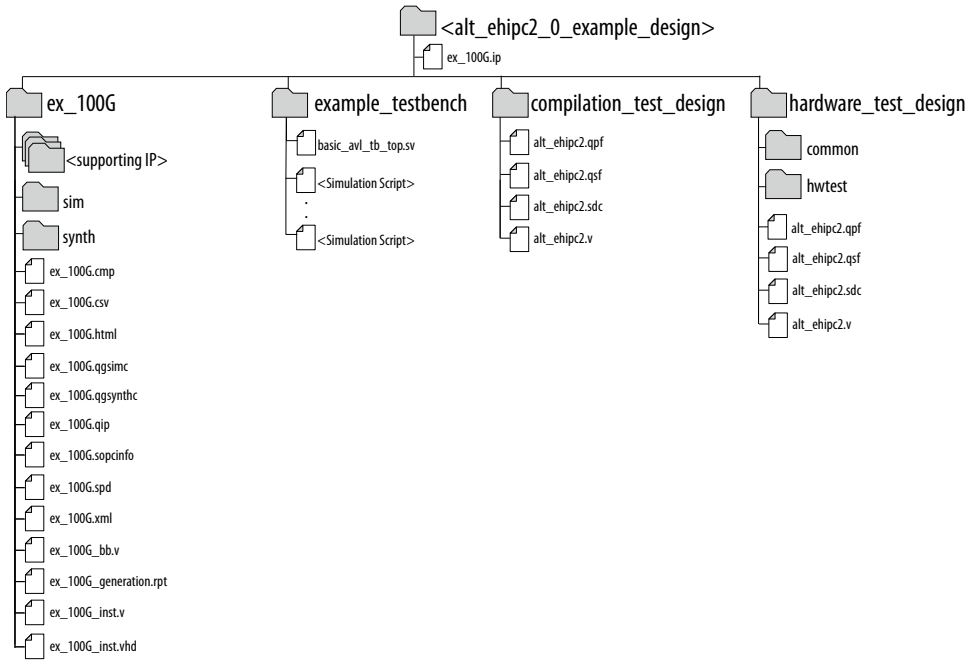

### Related Information

- H-Tile Hard IP for Ethernet Intel FPGA IP User Guide
  For detailed information on H-tile Hard IP.

- H-Tile Hard IP for Ethernet Intel Stratix 10 Release Notes
  The IP Release Notes list IP changes in a particular release.

## 1.1. Directory Structure

**Figure 2. H-Tile Hard IP for Ethernet Intel FPGA Design Example Directory Structure**




The hardware configuration and test files (the hardware design example) are located in *<design_example_dir>*/hardware_test_design. The simulation files (testbench for simulation only) are located in *<design_example_dir>*/example_testbench. The compilation-only design example is located in *<design_example_dir>*/compilation_test_design.

**Table 1. H-Tile Hard IP for Ethernet Intel FPGA IP Core Testbench File Descriptions**

| File Names | Description |
|---|---|
| Key Testbench and Simulation Files | |
| basic_avl_tb_top.sv | Top-level testbench file. The testbench instantiates the DUT and runs Verilog HDL tasks to generate and accept packets. |
| Testbench Scripts | |
| run_vsim.do | The Mentor Graphics ModelSim* script to run the testbench. |
| run_vcs.sh | The Synopsys VCS* script to run the testbench. |
| run_vcsmx.sh | The Synopsys VCS MX* script (combined Verilog HDL and System Verilog with VHDL) to run the testbench. |
| run_ncsim.sh | The Cadence NCSim* script to run the testbench. |
| run_xcelium.sh | The Xcelium* script to run the testbench. |

**Table 2. H-Tile Hard IP for Ethernet Intel FPGA IP Core Hardware Design Example File Descriptions**

| File Names | Description |
|---|---|
| `alt_ehip2.qpf` | Intel Quartus® Prime project file |
| `alt_ehip2.qsf` | Intel Quartus Prime project settings file |
| `alt_ehip2.sdc` | Synopsys Design Constraints files. You can copy and modify these files for your own H-Tile Hard IP for Ethernet Intel FPGA design. |
| `alt_ehip2.v` | Top-level Verilog HDL design example file |
| `common/` | Hardware design example support files |
| `hwtest/main.tcl` | Main file for accessing System Console |

## 1.2. Generating the Design

**Figure 3. Procedure**

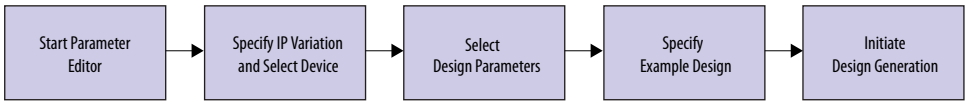
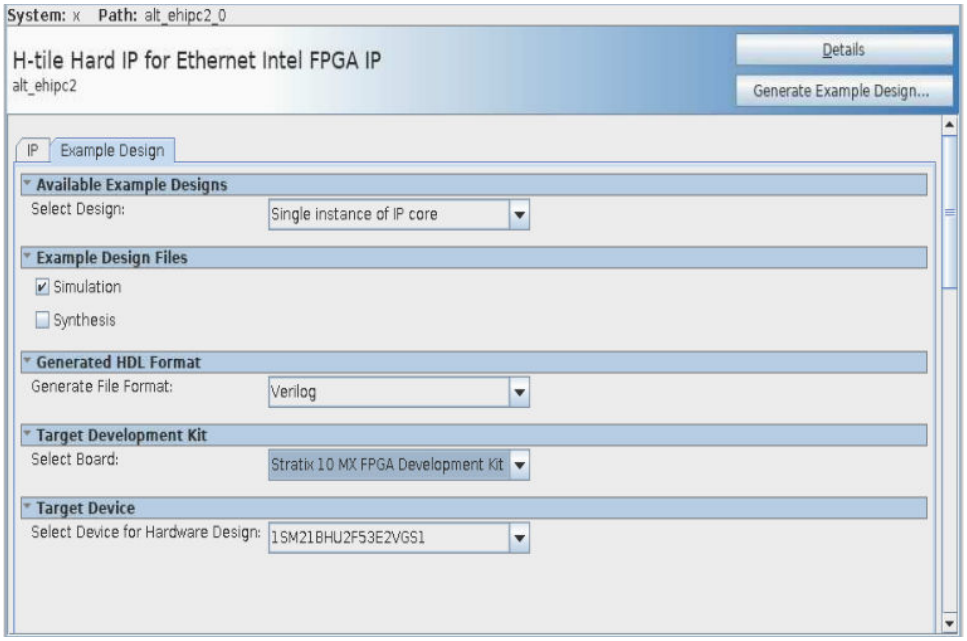



**Figure 4. Example Design Tab in the H-Tile Hard IP for Ethernet Intel FPGA IP Parameter Editor**




Follow these steps to generate the H-Tile Hard IP for Ethernet Intel FPGA IP hardware design example and testbench:

1. If you do not already have an Intel Quartus Prime Pro Edition project in which to integrate your H-Tile Hard IP for Ethernet Intel FPGA IP core, you must create one.

a. In the Intel Quartus Prime Pro Edition, click **File ➤ New Project Wizard** to create a new Quartus Prime project, or **File ➤ Open Project** to open an existing Intel Quartus Prime project. The wizard prompts you to specify a device.

b. Specify the device family **Intel Stratix 10** and select a device that meets all of these requirements:
   - Transceiver tile is H-tile
   - Transceiver speed grade is –1 or –2
   - Core speed grade is –1 or –2

c. Click **Finish**.

2. In the IP Catalog, locate and select **H-tile Hard IP for Ethernet Intel FPGA IP**. The **New IP Variation** window appears.
3. Specify a top-level name <*your_ip*> for your custom IP variation. The parameter editor saves the IP variation settings in a file named <*your_ip*>.ip.
4. Click **OK**. The parameter editor appears.
5. On the **IP** tab, specify the parameters for your IP core variation.

   *Important:* The design example testbench supports only the default IP parameter settings listed below. Any changes to these settings may cause simulation failure in the testbench.

| IP Parameter Settings | Default Value |
|---|---|
| **Ready latency** | 0 |
| **TX maximum frame size** | 1518 |
| **RX maximum frame size** | 1518 |
| **Enforce maximum frame size** | Disable |
| **Link fault generation option** | Off |
| **Stop TX traffic when link partner send pause** | No |
| **Bytes to remove from RX frames** | Remove CRC bytes |
| **Forward RX pause requests** | Disable |
| **Use source address insertion** | Disable |
| **TX VLAN detection** | Enable |
| **RX VLAN detection** | Enable |
| **PHY Reference Frequency** | 644.53125 |
| **Enable AN/LT** | Disable |
| **Enable Native PHY Debug Master endpoint (NPDME)** | Disable |
| **Enable JTAG to Avalon Master Bridge** | Disable |

6. On the **Example Design** tab, under **Example Design Files**, select the **Simulation** option to generate the testbench and the compilation-only project. Select the **Synthesis** option to generate the hardware design example.

   *Note:* You must select at least one of the **Simulation** and **Synthesis** options to generate the design example.

   *Note:* You must select the **Simulation** option to generate the testbench.

7. On the **Example Design** tab, under **Generated HDL Format**, select **Verilog** HDL or **VHDL**

   *Note:* If you select **VHDL**, you must simulate the testbench with a mixed-language simulator. The device under test in the `ex_100G` directory is a VHDL model, but the main testbench file is a System Verilog file.

8. Under **Target Development Kit** select the **Stratix 10 MX FPGA Development Kit** to generate the hardware design example. Selecting **None** generates only the simulation and compilation-only design examples.

   *Note:* The compilation-only and hardware design examples target your project device. For correct hardware design functionality out of the box, you must ensure your project device is the device on your development kit.

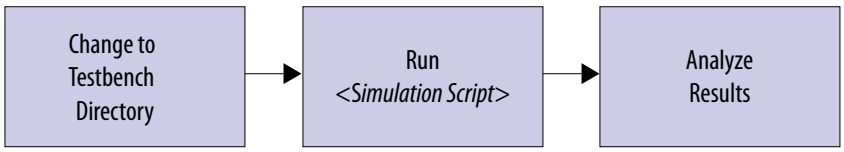
9. Under **Target Device** select **1SM21BHU2F53E1VG** or **1SM21BHU2F53E2VGS1**.
10. Click the **Generate Example Design** button. The **Select Example Design Directory** window appears.
11. If you want to modify the design example directory path or name from the defaults displayed (`alt_ehipc2_0_example_design`), browse to the new path and type the new design example directory name (<*design_example_dir*>).

**Related Information**

- H-tile Hard IP for Ethernet Intel FPGA IP Parameters
- Intel Stratix 10 MX FPGA Development Kit

## 1.3. Simulating the H-Tile Hard IP for Ethernet Intel FPGA Design Example Testbench

**Figure 5. Procedure**




Follow these steps to simulate the testbench:

1. Change to the testbench simulation directory <*design_example_dir*>/`example_testbench`.
2. Run the simulation script for the supported simulator of your choice. The script compiles and runs the testbench in the simulator. Refer to the table *Steps to Simulate the Testbench*.
3. Analyze the results. The successful testbench sends ten or fourteen packets, receives the same number of packets, and displays `"Testbench complete."`

**Table 3. Steps to Simulate the Testbench**

| Simulator | Instructions |
|---|---|
| Mentor Graphics ModelSim | In the command line, type `vsim -do run_vsim.do`<br>If you prefer to simulate without bringing up the ModelSim GUI, type `vsim -c -do run_vsim.do` |

*continued...*

| Simulator | Instructions |
|---|---|
| | *Note:* The ModelSim - Intel FPGA Edition simulator does not have the capacity to simulate this IP core. You must use another supported ModelSim simulator such as ModelSim SE. |
| Cadence NCSim | In the command line, type `sh run_ncsim.sh` |
| Synopsys VCS*/VCS MX* | In the command line, type `sh run_vcs.sh` or `sh run_vcsmx.sh`<br>*Note:* `run_vcs.sh` is only available if you select **Verilog** as the **Generated HDL Format**. If you select **VHDL** as the **Generated HDL Format**, you must simulate the testbench with a mixed language simulator using `run_vcsmx.sh`. |
| Xcelium* | In the command line, type `sh run_xcelium.sh` |

The successful test run displays output confirming the following behavior:

1. Waiting for the ATX PLLs to lock.
2. Waiting for RX transceiver reset to complete.
3. Waiting for RX alignment.
4. Sending ten (MAC+PCS, OTN, and FlexE) or hundred (PCS Only) packets.
5. Receiving those packets (MAC+PCS, OTN, and FlexE) or receiving and checking those packets (PCS Only).
6. Displaying `Testbench complete.`

## 1.4. Compiling the Compilation-Only Project

To compile the compilation-only example project, follow these steps:

1. Ensure compilation design example generation is complete.
2. In the Intel Quartus Prime Pro Edition software, open the Intel Quartus Prime Pro Edition project `<design_example_dir>/compilation_test_design/alt_ehipc2.qpf`.
3. On the Processing menu, click **Start Compilation**.

After successful compilation, reports for timing and for resource utilization are available in your Intel Quartus Prime Pro Edition session.

**Related Information**

Block-Based Design Flows

## 1.5. Compiling and Configuring the Design Example in Hardware

To compile the hardware design example and configure it on your Intel Stratix 10 device, follow these steps:

1. Ensure hardware design example generation is complete.

*Note:* The hardware design example in Intel Quartus Prime version 19.1 supports only MAC+PCS and PCS Only variants.

2. In the Intel Quartus Prime Pro Edition software, open the Intel Quartus Prime project `<design_example_dir>/hardware_test_design/alt_ehip2.qpf`.
3. On the Processing menu, click **Start Compilation**.
4. After successful compilation, a `.sof` file is available in your specified directory. Follow these steps to program the hardware design example on the Intel Stratix 10 device:
   a. On the **Tools** menu, click **Programmer**.
   b. In the Programmer, click **Hardware Setup**.
   c. Select a programming device.
   d. Select the **Stratix 10 MX FPGA Development Kit** to which your Intel Quartus Prime Pro Edition session can connect.
   e. Ensure that **Mode** is set to **JTAG**.
   f. Select the Intel Stratix 10 device and click **Add Device**. The Programmer displays a block diagram of the connections between the devices on your board.
   g. In the row with your `.sof`, check the box for the `.sof`.
   h. Check the box in the **Program/Configure** column.
   i. Click **Start**.

**Related Information**

- Block-Based Design Flows
- Programming Intel FPGA Devices
- Analyzing and Debugging Designs with System Console

## 1.6. Testing the H-Tile Hard IP for Ethernet Intel FPGA IP Hardware Design Example

After you compile the H-Tile Hard IP for Ethernet Intel FPGA IP core design example and configure it on your Intel Stratix 10 device, you can use the System Console to program the IP core and its embedded Native PHY IP core registers.

You can test the H-Tile Hard IP for Ethernet Intel FPGA IP hardware design example in either MAC + PCS mode or PCS Only mode.

### Testing the H-Tile Hard IP for Ethernet Intel FPGA IP Hardware Design Example in MAC + PCS Mode

To turn on the System Console and test the hardware design example, follow these steps:

1. After the hardware design example is configured on the Intel Stratix 10 device, in the Intel Quartus Prime Pro Edition software, click **Tools ➤ System Debugging Tools ➤ System Console**.
2. In the Tcl Console pane, type `cd hwtest` to change directory to `<design_example_dir>/hardware_test_design/hwtest`.
3. Type `source main.tcl` to open a connection to the JTAG master.

You can program the IP core with the following design example commands:

- `chkphy_status`: Displays the clock frequencies and PHY lock status.
- `chkmac_stats`: Displays the values in the MAC statistics counters.
- `clear_all_stats`: Clears the IP core statistics counters.
- `start_pkt_gen`: Starts the packet generator.
- `stop_pkt_gen`: Stops the packet generator.
- `loop_on`: Turns on internal serial loopback
- `loop_off`: Turns off internal serial loopback.
- `reg_read <addr>`: Returns the IP core register value at `<addr>`.
- `reg_write <addr> <data>`: Writes `<data>` to the IP core register at address `<addr>`.

### Testing the H-Tile Hard IP for Ethernet Intel FPGA IP Hardware Design Example in PCS Only Mode

The PCS Only design has its own MII packet generator, which is different from the MAC+PCS design packet generator.

You can run this packet generator in three modes:

- incremental mode
- constant mode (value configurable via the top level)
- random packet content mode

After you compile the PCS Only design example and configure it on your Intel Stratix 10 device, you can use the System Console to program the IP core and its embedded Native PHY IP registers.

1. After the hardware design example is configured on the Intel Stratix 10 device, in the Intel Quartus Prime Pro Edition software, click **Tools ➤ System Debugging Tools ➤ System Console**.
2. In the Tcl Console pane, type `cd hwtest` to change directory to `<design_example_dir>/hardware_test_design/hwtest`.
3. Type `source main.tcl` to open a connection to the JTAG master.

4. Copy the Example Script on page 11 below to the hwtest directory and save as `pcs_only_traffic_test.tcl`.
5. Type `source pcs_only_traffic_test.tcl`.
6. Check the traffic by sending and receiving packets by the running the command below:

   `pcs_only_traffic_test 10` (this command should transmit and receive 10 packets).

**Table 4. Registers**

| Functions | Bits | Description |
|---|---|---|
| Base register | 0xf000 | The base register for the PCS packet generator command. |
| proc start_pcs_pkt_gen | 0x0<br>0x01 | Starting the packet generator. |
| proc read_pcs_pkt_gen | 0x2 | Reading the results. |
| proc pcs_gen_num_frames | 0x6 num_frames | Setting the number of frames to send |
| proc get_pcs_gen_num_frames | 0x6 | Getting the number of frames |
| proc pcs_gen_frame_size | 0x7 frame_size | Setting the size of frames to send |
| proc get_pcs_gen_frame_size | 0x7 | Getting the size of frames |
| proc get_pcs_gen_match_count | 0x8 | Getting the number of matched frames |
| proc get_pcs_gen_mismatch_count | 0x9 | Getting the number of mismatched frames |
| proc pcs_gen_set_frame_type | Frame type = Incr; 0xa 0x1<br>Frame type = constant; 0xa 0x3<br>Frame type = random; 0xa 0x4 | Changing the type of frame<br>• 0xa: Set frame type<br>• 0x1: Set to incremental mode<br>• 0x3: Set to constant mode<br>• 0x4: Set to random packet content mode |

To access the registers, use `reg_read` or `reg_write`. For example:

To start the generator: `reg_write 0xf000 0x0 0x01`

To write to the number of frames register: `reg_write 0xf000 0x6 $num_frames`

## Example Script

This example script includes the functions listed in the Table 4 on page 11 table and a traffic test that runs in loopback mode and randomly picks a number (30 – 16,384) of frames to send, size of frames, and type of frame to send.

```
set  log_file_handle [open loopback_test_results.log w]

source main.tcl
set PCS_pkt_client_base 0xf000
set frame_type_list { "incr" "constant"  "random"}

proc hard_reset { }
{
    open_issp *
    sp_assert_reset
    sp_deassert_reset
    close_issp
}
```

```
proc random_element { list } {
    lindex $list [expr {int (rand()* [llength $list])}]
}

proc random { {min 30 } { max 100 } } \
{
    return [expr {int($min + rand() * $max)}]
}

proc start_pcs_pkt_gen { } \
{
   variable PCS_pkt_client_base
    reg_write $PCS_pkt_client_base 0x0 0x01
}

proc read_pcs_pkt_gen { } \
{
    variable PCS_pkt_client_base
    set res [reg_read $PCS_pkt_client_base 0x2]
    return $res
}

proc pcs_gen_num_frames { { num_frames 200} } \
{
    variable PCS_pkt_client_base
    reg_write $PCS_pkt_client_base 0x6 $num_frames
}

proc get_pcs_gen_num_frames { } \
{
    variable PCS_pkt_client_base
    reg_read $PCS_pkt_client_base 0x6
}

proc pcs_gen_frame_size { { frame_size 200} } \
{
    variable PCS_pkt_client_base
    reg_write $PCS_pkt_client_base 0x7  $frame_size
}

proc get_pcs_gen_frame_size {   } \
{
    variable PCS_pkt_client_base
    reg_read $PCS_pkt_client_base 0x7
}
proc get_pcs_gen_match_count {   } \
{
    variable PCS_pkt_client_base
    reg_write $PCS_pkt_client_base 0x8
}

proc get_pcs_gen_mismatch_count {   } \
{
    variable PCS_pkt_client_base
    reg_read $PCS_pkt_client_base 0x9
}

proc pcs_gen_set_frame_type { {frame_type "incr"} } \
{
    variable PCS_pkt_client_base
    if {$frame_type == "incr"} {
        reg_write $PCS_pkt_client_base 0xa 0x1
    } elseif    { $frame_type == "constant" } {
        reg_write $PCS_pkt_client_base 0xa 0x3
    } elseif    { $frame_type == "random" } {
        reg_write $PCS_pkt_client_base 0xa 0x4
    } else {
        puts "ERROR"
        return -1
    }
    return 0
```

```
}

proc pcs_only_traffic_test { iterations } {
    variable    log_file_handle
    variable    frame_type_list
    set test_name [lindex [info level [info level]] 0]
    puts "Running ${test_name} test"
    after 100
    #hard_reset
    chkphy_status
    set phy_locked [phy_locked]
    puts "Phy locked status is $phy_locked "
    # global $CLIENT_BASE
    set PCS_pkt_client_base 0xf000
    for {set i 0} {$i < $iterations} {incr i} {
        #randomize some stuff. Like number and size of Frames
        set res [random 30 16384]
        puts "Setting Number of frames to $res\n"
        pcs_gen_num_frames $res
        set res [random 30 16384]
        puts "Setting Size of frames to $res\n"
        pcs_gen_frame_size $res
        set res [random_element $frame_type_list]
        puts "Setting Type of frames to $res\n"
        pcs_gen_set_frame_type $res
        puts "PCS TRAFFIC = ${i}"
       start_pcs_pkt_gen
        after 1000
        set res [read_pcs_pkt_gen]
        if { $res == 0x53} {
            puts  "${test_name}:pass"
        } else {
            puts  "${test_name}:fail, Got $res instead of 0x53"
            puts $log_file_handle "${test_name}:fail, Got $res instead of 0x53"
            return -1
        }
    }
    puts $log_file_handle "${test_name}:pass"
    return 0
}

proc pcs_gen_result { } \
{
    variable PCS_pkt_client_base
    set pass [read_pcs_pkt_gen]
    set max_frames [reg_read $PCS_pkt_client_base 0x6]
    set frame_size [reg_read $PCS_pkt_client_base 0x7]
    set match_count [reg_read $PCS_pkt_client_base 0x8]
    set mismatch_count [reg_read $PCS_pkt_client_base 0x9]
    set frame_type [reg_read $PCS_pkt_client_base 0xa]

    if { $pass == 0x53} {
        puts  "Last test passed!\n"
    } else {
        puts  "Last test failed, Got $pass instead of 0x53"
    }
    puts "Num of frames = $max_frames"
    puts "Frame size    = $frame_size"
    puts "Match count    = $match_count"
    puts "Mismatch count    = $mismatch_count"
    if {$frame_type == 0x1} {
        puts "Frame Type = incr"
    } elseif    { $frame_type == 0x9 } {
        puts "Frame type = constant_crc"
    } elseif    { $frame_type == 0x3 } {
        puts "Frame type = constant"
    } elseif    { $frame_type == 0x4 } {
        puts "Frame type = random"
    }
}
```

**Related Information**

- Analyzing and Debugging Designs with System Console
- Hardware Design Example Components on page 23

## 1.6.1. Testing the Hardware Design Example using Ethernet Link Inspector

You can also test your design using the Ethernet Link Inspector (ELI) tool available in System console.

Design examples have built-in JTAG to AVMM bridge allowing you to use the Ethernet Link Inspector. Refer to the user guide on how to use the ELI to test your design.
The ELI tool is accessible via **System Console** in the **Tools ➤ Legacy Toolkits** in the Intel Quartus Prime Pro Edition software.

**Related Information**

Ethernet Link Inspector User Guide for Intel Stratix 10 Devices

# 2. Design Example Description

The design example demonstrates the basic functions of the H-Tile Hard IP for Ethernet Intel FPGA IP core with the following variants:

- 100 Gbps datarate:
  - MAC + PCS
  - PCS Only
  - OTN
  - FlexE

*Note:* The hardware design example in Intel Quartus Prime version 20.2 supports only MAC +PCS and PCS Only variants.

*Note:* The H-Tile Hard IP for Ethernet Intel FPGA IP provides preliminary support for the OTN feature. For further inquiries, contact your nearest Intel sales representative or file an Intel Premier Support (IPS) case on https://www.intel.com/content/www/us/en/programmable/my-intel/mal-home.html.

You can generate the design from the **Example Design** tab in the H-Tile Hard IP for Ethernet Intel FPGA parameter editor.

To generate the design example, you must first set the parameter values for the IP core variation you intend to generate in your end product. Generating the design example creates a copy of the IP core; the testbench, compilation-only, and hardware design example use this variation as the DUT. If you do not set the parameter values for the DUT to match the parameter values in your end product, the design example you generate does not exercise the IP core variation you intend.

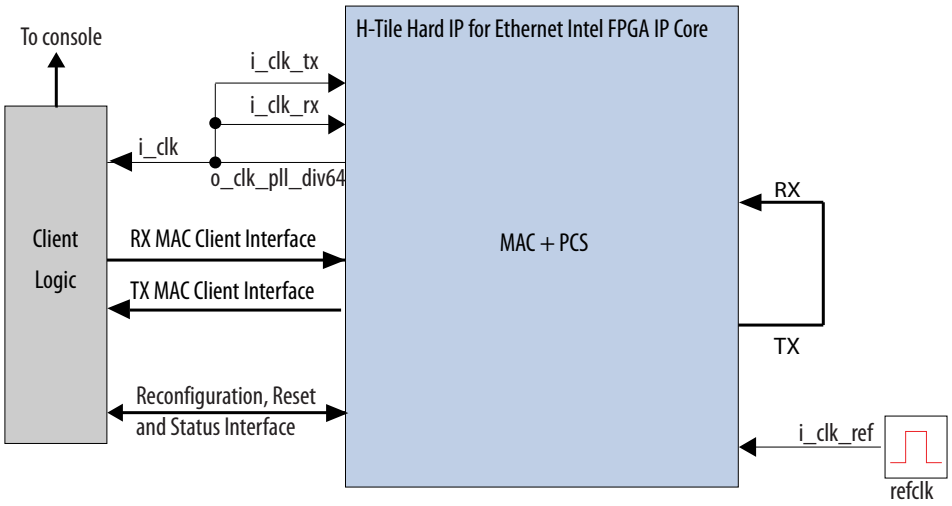
*Note:* The testbench demonstrates a basic test of the IP core. It is not intended to be a substitute for a full verification environment. You must perform more extensive verification of your own H-Tile Hard IP for Ethernet Intel FPGA design in simulation and in hardware.

**Related Information**

H-tile Hard IP for Ethernet User Guide

## 2.1. H-Tile Hard IP for Ethernet Intel FPGA MAC + PCS Simulation Design Example

**Figure 6. H-Tile Hard IP for Ethernet Intel FPGA MAC + PCS Simulation Design Example Block Diagram**




The testbench sends traffic through the IP core, exercising the transmit and receive MAC client interfaces of the IP core.

The simulation design example instantiates a main ATX PLL for transceiver channel 0 and 1, and a clock buffer for channel 2 and 3.

The testbench in this design example performs the following:

1. The client logic resets the IP core.
2. Client logic waits for RX datapath to align.
3. Once alignment is complete, client logic transmits a series of packets to the IP core.
4. The client logic receives the same series of packets through RX MAC interface.
5. The client logic then checks the number of packets received and verifies that the packets have no errors.

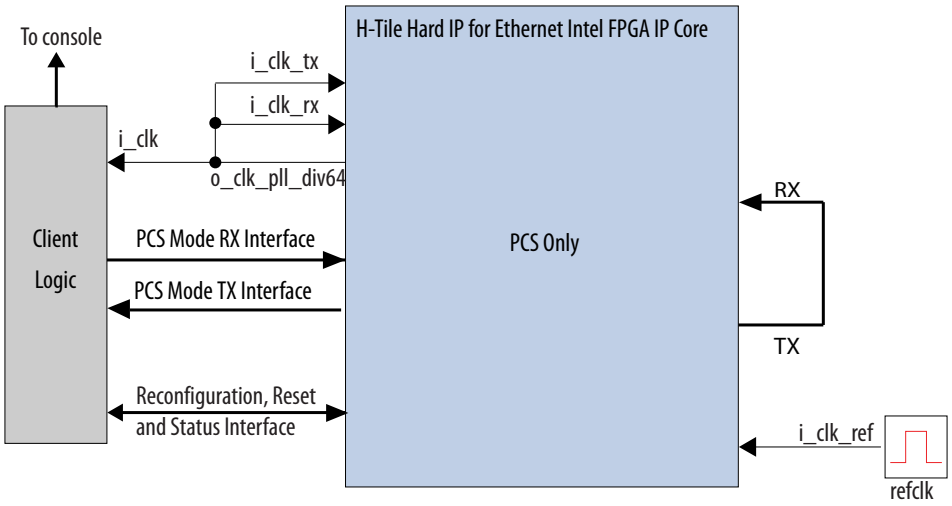
The following sample output illustrates a successful simulation test run for a 100 Gbps, MAC+PCS IP core variation. Times are in picoseconds.

```
Ref clock is 644.53125 MHz

================================================
Module ct1_hssi_cr2_ehip_pcs_interface
================================================
silicon_rev = 14nm5bcr2ea

waiting for o_tx_lanes_stable...
o_tx_lanes_stable is 1 at time                525000
waiting for tx_dll_lock....
TX DLL LOCK is 1 at time             25332363
waiting for tx_transfer_ready....
TX transfer ready is 1 at time               25652235
waiting for rx_transfer_ready....
RX transfer ready is 1 at time               31979703
```

```
EHIP PLD Ready out is 1 at time              32040000
EHIP reset out is 0 at time             32304000
EHIP reset ack is 0 at time             32612783
EHIP TX reset out is 0 at time             32928000
EHIP TX reset ack is 0 at time             82562795
waiting for EHIP Ready....
EHIP READY is 1 at time             82629435
EHIP RX reset out is 0 at time             82968000
waiting for rx reset ack....
EHIP RX reset ack is 0 at time             83029275
Waiting for RX Block Lock
EHIP RX Block Lock  is high at time              90223063
Waiting for AM lock
EHIP RX AM Lock  is high at time              91172683
Waiting for RX alignment
RX deskew locked
RX lane aligmnent locked
TX enabled
** Sending Packet           1...
** Sending Packet           2...
** Sending Packet           3...
** Sending Packet           4...
** Sending Packet           5...
** Sending Packet           6...
** Received Packet          1...
** Sending Packet           7...
** Received Packet          2...
** Sending Packet           8...
** Received Packet          3...
** Sending Packet           9...
** Received Packet          4...
** Sending Packet          10...
** Received Packet          5...
** Received Packet          6...
** Received Packet          7...
** Received Packet          8...
** Received Packet          9...
** Received Packet         10...
**
** Testbench complete.
**
****************************************
```

## 2.2. H-Tile Hard IP for Ethernet Intel FPGA PCS Only Simulation Design Example

**Figure 7. H-Tile Hard IP for Ethernet Intel FPGA PCS Only Simulation Design Example Block Diagram**




The testbench sends traffic through the IP core, exercising the transmit and receive Media Independent Interface (MII) of the IP core.

The simulation design example instantiates a main ATX PLL for transceiver channel 0 and 1, and a clock buffer for channel 2 and 3.

The testbench in this design example performs the following:

1. The client logic resets the IP core.
2. Client logic waits for RX datapath to align.
3. Once alignment is complete, client logic transmits a series of packets to the IP core through TX MII interface.
4. A counter drives `i_tx_mii_am` port with alignment marker insertion requests at the correct intervals.
5. The client logic receives the same series of packets through RX MII interface.
6. The client logic then checks the number of packets received.

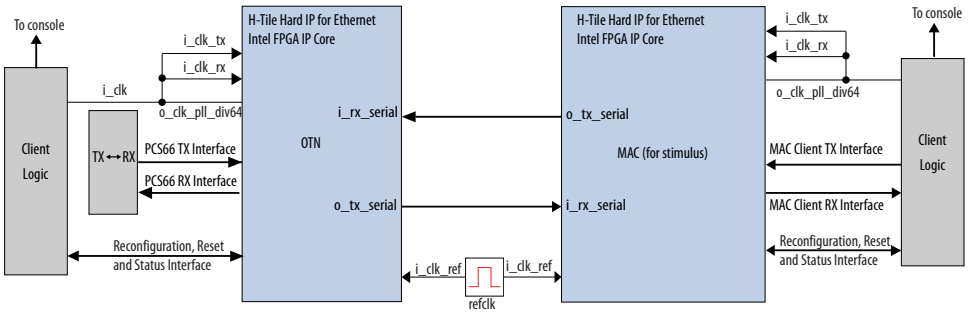
The following sample output illustrates a successful simulation test run for a 100 Gbps, PCS Only IP core variation. Times are in picoseconds.

```
waiting for o_tx_lanes_stable...
o_tx_lanes_stable is 1 at time                525000
waiting for tx_dll_lock....
TX DLL LOCK is 1 at time             19792913
waiting for tx_transfer_ready....
TX transfer ready is 1 at time              20112785
waiting for rx_transfer_ready....
RX transfer ready is 1 at time              31188353
EHIP PLD Ready out is 1 at time              31248000
EHIP reset out is 0 at time             31288000
EHIP reset ack is 0 at time             31663163
EHIP TX reset out is 0 at time              31776000
EHIP TX reset ack is 0 at time              81509883
```

```
waiting for EHIP Ready....
EHIP READY is 1 at time              81603179
EHIP RX reset out is 0 at time              81720000
waiting for rx reset ack....
EHIP RX reset ack is 0 at time              81789771
Waiting for RX Block Lock
EHIP Rx Block Lock  is high at time              87690743
Waiting for AM lock
EHIP Rx am Lock  is high at time              89748253
Waiting for RX alignment
RX deskew locked
RX lane alignment locked
Sending Packets
Receiving Packets
Address 0x0000f002 data 0x00000053
Total traffic cycle errors: 0
100G Traffic Test PASSED..!!!
**
** Testbench complete.
**
****************************************
```

## 2.3. H-Tile Hard IP for Ethernet Intel FPGA OTN Simulation Design Example

**Figure 8. H-Tile Hard IP for Ethernet Intel FPGA OTN Simulation Design Example Block Diagram**




*Note:* The H-Tile Hard IP for Ethernet Intel FPGA IP provides preliminary support for the OTN feature. For further inquiries, contact your nearest Intel sales representative or file an Intel Premier Support (IPS) case on https://www.intel.com/content/www/us/en/programmable/my-intel/mal-home.html.

The testbench sends traffic through the IP core with OTN mode, exercising the transmit and receive PCS66 interfaces using a separate H-Tile Hard IP for Ethernet Intel FPGA MAC as a stimulus generator.

The simulation design example instantiates a main ATX PLL for transceiver channel 0 and 1, and a clock buffer for channel 2 and 3.

The testbench in this design example performs the following:

1. The client logic resets both the IP cores.
2. The stimulus client logic waits for the stimulus RX datapath and OTN RX datapath to align.
3. Once alignment is complete, the stimulus client logic transmits a series of packets to the OTN IP core.
4. The OTN IP core receives the series of packets and transmits back to the stimulus MAC IP core.
5. The stimulus client logic then checks the number of packets received and verifies that the packets have no errors.

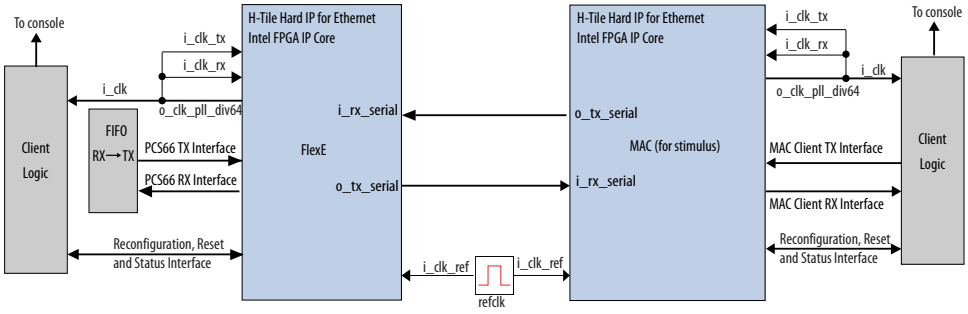
The following sample output illustrates a successful simulation test run for a 100 Gbps OTN IP core variation. Times are in picoseconds.

```
# Ref clock is 644.53125 MHz
# Ref clock is 644.53125 MHz
# iatpg_pipeline_global_en is set
.
.
.
# iatpg_pipeline_global_en is set
# test_dut:waiting for o_tx_lanes_stable...
# dut:waiting for o_tx_lanes_stable...
# test_dut:o_tx_lanes_stable is 1 at time                525000
# test_dut:waiting for tx_dll_lock....
# dut:o_tx_lanes_stable is 1 at time                525000
# dut:waiting for tx_dll_lock....
# dut:TX DLL LOCK is 1 at time              47806703
# dut:waiting for tx_transfer_ready....
# dut:TX transfer ready is 1 at time              48126575
# dut:waiting for rx_transfer_ready....
# dut:RX transfer ready is 1 at time              59518683
# dut:EHIP PLD Ready out is 1 at time             59576000
# dut:EHIP reset out is 0 at time             59840000
# dut:EHIP reset ack is 0 at time             60151763
# dut:EHIP TX reset out is 0 at time             60488000
# dut:EHIP TX reset ack is 0 at time            110085115
# dut:waiting for EHIP Ready....
# dut:EHIP READY is 1 at time            110178411
# dut:EHIP RX reset out is 0 at time             110520000
# dut:waiting for rx reset ack....
# dut:EHIP RX reset ack is 0 at time             110578251
# dut:Waiting for RX Block Lock
# test_dut:TX DLL LOCK is 1 at time             124725923
# test_dut:waiting for tx_transfer_ready....
# test_dut:TX transfer ready is 1 at time             125045795
# test_dut:waiting for rx_transfer_ready....
# test_dut:RX transfer ready is 1 at time             136437903
# test_dut:EHIP PLD Ready out is 1 at time             136496000
# test_dut:EHIP reset out is 0 at time            136760000
# test_dut:EHIP reset ack is 0 at time            137070983
# test_dut:EHIP TX reset out is 0 at time             137408000
# test_dut:EHIP TX reset ack is 0 at time             187001003
# test_dut:waiting for EHIP Ready....
# test_dut:EHIP READY is 1 at time            187107627
# test_dut:EHIP RX reset out is 0 at time             187448000
# test_dut:waiting for rx reset ack....
# test_dut:EHIP RX reset ack is 0 at time             187507467
# test_dut:Waiting for RX Block Lock
# dut:EHIP RX Block Lock  is high at time             189300083
# dut:Waiting for AM lock
# dut:EHIP RX AM Lock  is high at time            190566243
# dut:Waiting for RX alignment
# dut:RX deskew locked
```

```
# dut:RX lane aligmnent locked
# dut:**
# dut:** Testbench complete.
# dut:**
# dut:****************************************
# test_dut:EHIP RX Block Lock  is high at time            194839533
# test_dut:Waiting for AM lock
# test_dut:EHIP RX AM Lock  is high at time           196580503
# test_dut:Waiting for RX alignment
# test_dut:RX deskew locked
# test_dut:RX lane aligmnent locked
# dut:RX deskew locked
# dut:RX lane aligmnent locked
# test_dut:TX enabled
# test_dut: ** Sending Packet          1...
# test_dut: ** Sending Packet          2...
# test_dut: ** Sending Packet          3...
# test_dut: ** Sending Packet          4...
# test_dut: ** Sending Packet          5...
# test_dut: ** Sending Packet          6...
# test_dut: ** Sending Packet          7...
# test_dut: ** Sending Packet          8...
# test_dut: ** Sending Packet          9...
# test_dut: ** Received Packet         1...
# test_dut: ** Sending Packet         10...
# test_dut: ** Received Packet         2...
# test_dut: ** Received Packet         3...
# test_dut: ** Received Packet         4...
# test_dut: ** Received Packet         5...
# test_dut: ** Received Packet         6...
# test_dut: ** Received Packet         7...
# test_dut: ** Received Packet         8...
# test_dut: ** Received Packet         9...
# test_dut: ** Received Packet        10...
# test_dut:**
# test_dut:** Testbench complete.
# test_dut:**
# test_dut:****************************************
```

## 2.4. H-Tile Hard IP for Ethernet Intel FPGA FlexE Simulation Design Example

**Figure 9. H-Tile Hard IP for Ethernet Intel FPGA 100 Gbps FlexE Simulation Design Example Block Diagram**




The testbench sends traffic through the IP core with FlexE mode, exercising the transmit and receive PCS66 interfaces using a separate H-Tile Hard IP for Ethernet Intel FPGA MAC as a stimulus generator. The PCS66 interface of the FlexE core is connected to a synchronous FIFO that receives PCS66 data and alignment marker valid signals. The FIFO then writes back to the PCS66 transmit interface,

The simulation design example instantiates a main ATX PLL for transceiver channel 0 and 1, and a clock buffer for channel 2 and 3.

The testbench in this design example performs the following:

1. The client logic resets both the IP cores.
2. The stimulus client logic waits for the stimulus RX datapath and FlexE RX datapath to align.
3. Once alignment is complete, the stimulus client logic transmits a series of packets to the FlexE IP core.
4. The FlexE IP core receives the series of packets and transmits back to the stimulus MAC IP core.
5. The stimulus client logic then checks the number of packets received and verifies that the packets have no errors.

The following sample output illustrates a successful simulation test run for a 100 Gbps FlexE IP core variation. Times are in picoseconds.

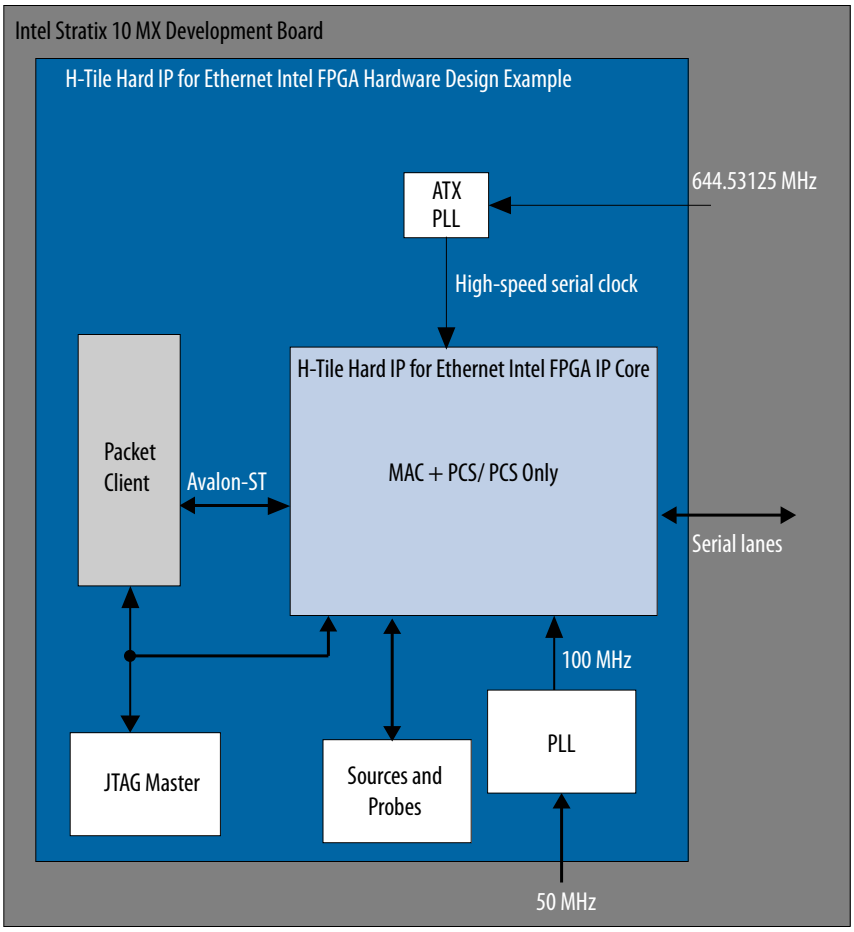
```
# Ref clock is 644.53125 MHz
# Ref clock is 644.53125 MHz
# iatpg_pipeline_global_en is set
.
.
.
# iatpg_pipeline_global_en is set
# test_dut:waiting for o_tx_lanes_stable...
# dut:waiting for o_tx_lanes_stable...
# test_dut:o_tx_lanes_stable is 1 at time               525000
# test_dut:waiting for tx_dll_lock....
# dut:o_tx_lanes_stable is 1 at time               525000
# dut:waiting for tx_dll_lock....
# dut:TX DLL LOCK is 1 at time             47806703
# dut:waiting for tx_transfer_ready....
# dut:TX transfer ready is 1 at time             48126575
# dut:waiting for rx_transfer_ready....
# dut:RX transfer ready is 1 at time             59518683
# dut:EHIP PLD Ready out is 1 at time             59576000
# dut:EHIP reset out is 0 at time            59840000
# dut:EHIP reset ack is 0 at time            60151763
# dut:EHIP TX reset out is 0 at time             60488000
# dut:EHIP TX reset ack is 0 at time            110085115
# dut:waiting for EHIP Ready....
# dut:EHIP READY is 1 at time            110178411
# dut:EHIP RX reset out is 0 at time            110520000
# dut:waiting for rx reset ack....
# dut:EHIP RX reset ack is 0 at time            110578251
# dut:Waiting for RX Block Lock
# test_dut:TX DLL LOCK is 1 at time            124725923
# test_dut:waiting for tx_transfer_ready....
# test_dut:TX transfer ready is 1 at time             125045795
# test_dut:waiting for rx_transfer_ready....
# test_dut:RX transfer ready is 1 at time             136437903
# test_dut:EHIP PLD Ready out is 1 at time             136496000
# test_dut:EHIP reset out is 0 at time            136760000
# test_dut:EHIP reset ack is 0 at time            137070983
# test_dut:EHIP TX reset out is 0 at time             137408000
# test_dut:EHIP TX reset ack is 0 at time             187001003
# test_dut:waiting for EHIP Ready....
# test_dut:EHIP READY is 1 at time            187107627
# test_dut:EHIP RX reset out is 0 at time             187448000
# test_dut:waiting for rx reset ack....
# test_dut:EHIP RX reset ack is 0 at time             187507467
# test_dut:Waiting for RX Block Lock
# dut:EHIP RX Block Lock  is high at time             189300083
# dut:Waiting for AM lock
```

```
# dut:EHIP RX AM Lock  is high at time             190566243
# dut:Waiting for RX alignment
# dut:RX deskew locked
# dut:RX lane aligmnent locked
# dut:**
# dut:** Testbench complete.
# dut:**
# dut:****************************************
# test_dut:EHIP RX Block Lock  is high at time             195472613
# test_dut:Waiting for AM lock
# test_dut:EHIP RX AM Lock  is high at time             196580503
# test_dut:Waiting for RX alignment
# test_dut:RX deskew locked
# test_dut:RX lane aligmnent locked
# dut:RX deskew locked
# dut:RX lane aligmnent locked
# test_dut:TX enabled
# test_dut: ** Sending Packet           1...
# test_dut: ** Sending Packet           2...
# test_dut: ** Sending Packet           3...
# test_dut: ** Sending Packet           4...
# test_dut: ** Sending Packet           5...
# test_dut: ** Sending Packet           6...
# test_dut: ** Sending Packet           7...
# test_dut: ** Sending Packet           8...
# test_dut: ** Sending Packet           9...
# test_dut: ** Sending Packet          10...
# test_dut: ** Received Packet          1...
# test_dut: ** Received Packet          2...
# test_dut: ** Received Packet          3...
# test_dut: ** Received Packet          4...
# test_dut: ** Received Packet          5...
# test_dut: ** Received Packet          6...
# test_dut: ** Received Packet          7...
# test_dut: ** Received Packet          8...
# test_dut: ** Received Packet          9...
# test_dut: ** Received Packet         10...
# test_dut:**
# test_dut:** Testbench complete.
# test_dut:**
# test_dut:****************************************
```

## 2.5. Hardware Design Example Components

The H-Tile Hard IP for Ethernet Intel FPGA hardware design example supports only the MAC + PCS and PCS Only variants.

**Figure 10. Intel Stratix 10 MX Development Kit Hardware Design Example High Level Block Diagram**




The H-Tile Hard IP for Ethernet Intel FPGA hardware design example includes the following components:

- H-Tile Hard IP for Ethernet Intel FPGA IP core.
- Client logic that coordinates the programming of the IP core and packet generation.
- One ATX PLL to generate the high speed serial clock to drive the device transceiver channels.
- An I/O PLL to generate a 100 MHz clock from a 50 MHz input clock to the hardware design example.
- JTAG controller that communicates with the System Console. You communicate with the client logic through the System Console.

The hardware design example uses `run_test` command to initiate packet transmission from packet generator to the IP core. The IP core receives the packets and transmit to the packet generator through the serial loopback. The client logic reads and print out the MAC statistic registers when the packet transmissions are complete.

### Related Information

Intel Stratix 10 MX FPGA Development Kit

## 2.6. Design Example Interface Signals

The H-Tile Hard IP for Ethernet Intel FPGA testbench is self-contained and does not require you to drive any input signals.

### Related Information

Interfaces and Signal Descriptions
Provides detailed descriptions of the H-Tile Hard IP for Ethernet Intel FPGA IP core signals and the interfaces to which they belong.

## 2.7. H-Tile Hard IP for Ethernet Intel FPGA Design Example Registers

### Table 5. H-Tile Hard IP for Ethernet Intel FPGA Hardware Design Example Register Map

Lists the memory mapped register ranges for the hardware design example. You access these registers with the `reg_read` and `reg_write` functions in the System Console.

| Word Offset | Register Type |
|---|---|
| 0x000000 | KR4 registers |
| 0x000300 | RX PHY registers |
| 0x000400 | TX MAC registers |
| 0x000500 | RX MAC registers |
| 0x000800 | TX Statistics Counter registers |
| 0x000900 | RX Statistics Counter registers |
| 0x001000 | Packet Client registers |

### Table 6. Packet Client Registers

You can customize the H-Tile Hard IP for Ethernet Intel FPGA hardware design example by programming the packet client registers.

| Addr | Name | Bit | Description | HW Reset Value | Access |
|---|---|---|---|---|---|
| 0x1000 | PKT_CL_SCRATCH | [31:0] | Scratch register available for testing. | | RW |
| 0x1001 | PKT_CL_CLNT | [31:0] | Four characters of IP block identification string "CLNT" | | RO |
| 0x1008 | Packet Size Configure | [29:0] | Specify the transmit packet size in bytes. These bits have dependencies to PKT_GEN_TX_CTRL register. | 0x25800040 | RW |

*continued...*

| Addr | Name | Bit | Description | HW Reset Value | Access |
|---|---|---|---|---|---|
| | | | • Bit [29:16]: Specify the upper limit of the packet size in bytes. This is only applicable to incremental mode.<br>• Bit [13:0]:<br>— For fixed mode, these bits specify the transmit packet size in bytes.<br>— For incremental mode, these bits specify the incremental bytes for a packet. | | |
| 0x1009 | Packet Number Control | [31:0] | Specify the number of packets to transmit from the packet generator. | 0xA | RW |
| 0x1010 | PKT_GEN_TX_CTRL | [7:0] | • Bit [0]: Reserved.<br>• Bit [1]: Packet generator disable bit. Set this bit to the value of 1 to turn off the packet generator, and reset it to the value of 0 to turn on the packet generator.<br>• Bit [2]: Reserved.<br>• Bit [3]: Has the value of 1 if the IP core is in MAC loopback mode; has the value of 0 if the packet client uses the packet generator.<br>• Bit [5:4]:<br>— 00: Random mode<br>— 01: Fixed mode<br>— 10: Incremental mode<br>• Bit [6]: Set this bit to 1 to use 0x1009 register to turn off packet generator based on a fixed number of packets to transmit. Otherwise, bit [1] of PKT_GEN_TX_CTRL register is used to turn off the packet generator.<br>• Bit [7]:<br>— 1: For transmission without gap in between packets.<br>— 0: For transmission with random gap in between packets. | 0x6 | RW |
| 0x1011 | Destination address lower 32 bits | [31:0] | Destination address (lower 32 bits) | 0x56780ADD | RW |
| 0x1012 | Destination address upper 16 bits | [15:0] | Destination address (upper 16 bits) | 0x1234 | RW |
| 0x1013 | Source address lower 32 bits | [31:0] | Source address (lower 32 bits) | 0x43210ADD | RW |
| 0x1014 | Source address upper 16 bits | [15:0] | Source address (upper 16 bits) | 0x8765 | RW |
| 0x1016 | PKT_CL_LOOPBACK_RESET | [0] | MAC loopback reset. Set to the value of 1 to reset the design example MAC loopback. | 1'b0 | RW |

### Related Information

H-Tile Hard IP for Ethernet Intel FPGA IP core register descriptions

# 3. H-tile Hard IP for Ethernet Intel Stratix 10 FPGA IP Design Example User Guide Document Archives

IP versions are the same as the Intel Quartus Prime Design Suite software versions up to v19.1. From Intel Quartus Prime Design Suite software version 19.2 or later, IP cores have a new IP versioning scheme.

If an IP core version is not listed, the user guide for the previous IP core version applies.

| Intel Quartus Prime Version | IP Core Version | User Guide |
|---|---|---|
| 19.3 | 19.2.0 | H-tile Hard IP for Ethernet Intel Stratix 10 FPGA IP Design Example User Guide |
| 19.1 | 19.1 | H-tile Hard IP for Ethernet Intel Stratix 10 FPGA IP Design Example User Guide |
| 18.1 | 18.1 | H-tile Hard IP for Ethernet Intel Stratix 10 FPGA IP Design Example User Guide |
| 18.0 | 18.0 | H-tile Hard IP for Ethernet Intel Stratix 10 FPGA IP Design Example User Guide |
| 17.1 | 17.1 | Intel Stratix 10 H-tile Hard IP for Ethernet Design Example User Guide |

# 4. Document Revision History for the H-tile Hard IP for Ethernet Intel Stratix 10 FPGA IP Design Example User Guide

| Document Version | Intel Quartus Prime Version | IP Version | Changes |
|---|---|---|---|
| 2020.06.22 | 20.2 | 19.3.0 | • Updated the *Quick Start Guide* section:<br>— Added User Guide link<br>— Added Release Notes link<br>• Removed 50-Gbps Ethernet variant support. |
| 2019.10.31 | 19.3 | 19.2.0 | • Changed 0x104 register's name from `Source address lower 16 bits` to `Source address upper 16 bits` in the *Packet Client Registers* table.<br>• Added Xcelium and Synopsys VCS MX simulators support.<br>• Updated *Generating the Design* section:<br>— Updated the *Example Design Tab in the H-tile Hard IP for Ethernet Intel FPGA Parameter Editor* figure<br>— Added parameter: **PHY Reference Frequency**<br>— Replaced **Enable Altera Debug Master endpoint** parameter with **Enable Altera Debug Master endpoint**<br>— Added parameter: **Enable JTAG to Avalon Master Bridge**<br>— Updated target development kit to **Stratix 10 MX FPGA Development Kit**<br>— Added step to select the Target Device<br>• Updated the *Steps to Simulate the Testbench* table:<br>— Added instruction for Xcelium and Synopsys VCS MX simulator<br>— Added note to clarify `run_vcs.sh` and `run_vcsmx.sh` usage<br>• Updated the *Intel Stratix 10 MX FPGA Development Kit Hardware Design Example High Level Block Diagram* figure in the *Hardware Design Example Components*.<br>• Updated target development kit to **Stratix 10 MX FPGA Development Kit** in the *Hardware Design Example Components* section.<br>• Added the *Testing the Hardware Design Example using Ethernet Link Inspector*. |
| 2019.04.01 | 19.1 | 19.1 | Removed information about Intel Stratix 10 GXT Transceiver SoC development kit in the *Generating the Design* and *Hardware Design Example Components* sections. The H-tile Hard IP for Ethernet Intel FPGA IP version 19.1 does not support Intel Stratix 10 GXT Transceiver SoC development kit. |

*continued...*

| Document Version | Intel Quartus Prime Version | IP Version | Changes |
|---|---|---|---|
| 2019.01.21 | 18.1 | 18.1 | • Changed the image of the H-tile Hard IP for Ethernet FPGA IP design example parameter editor to reflect the latest version.<br>• Updated the *Compiling and Configuring the Design Example in Hardware*, *Design Example Description*, and *Hardware Design Example Components* sections to include that the hardware design example now supports PCS Only variants.<br>• Edited the successful test behavior description for PCS Only variant in the *Simulating the H-tile Hard IP for Ethernet Intel FPGA Design Example Testbench* section. A successful test run sends hundred packets for PCS Only variants.<br>• Added the *Testing the Hardware Design Example in PCS Only Mode* section.<br>• Edited the Intel Stratix 10 GXT Transceiver Signal Integrity and Intel Stratix 10 GXT Transceiver SoC development kit hardware design example diagrams to add PCS Only variant and remove RS-FEC. The current version does not support the Reed Solomon Forward Error Correction (RS-FEC) feature. |
| 2018.08.10 | 18.0 | 18.0 | • Added design example testbench components and test behavior for OTN and FlexE variations.<br>• Added hardware design example components and test behavior.<br>• Added hardware design example register description.<br>• Rebranded the IP core name from Intel Stratix 10 H-Tile Hard IP for Ethernet IP core to H-Tile Hard IP for Ethernet Intel FPGA per Intel rebranding. |
| 2017.11.29 | 17.1 | 17.1 | Initial release. |