



AN 847: Signal Tap Tutorial with Design Block Reuse

for Intel® Arria® 10 FPGA Development Board

Updated for Intel® Quartus® Prime Design Suite: **18.0**



Subscribe



Send Feedback

AN-847 | 2018.05.07

Latest document on the web: [PDF](#) | [HTML](#)



Contents

1. Introduction.....	3
1.1. Signal Tap with Core Partition Reuse.....	3
1.2. Signal Tap with Root Partition Reuse.....	5
1.3. Tutorial Software and Hardware.....	7
1.4. Tutorial Files.....	7
2. Core Partition Reuse Debug—Developer.....	9
2.1. Step 1: Create a Core Partition.....	9
2.2. Step 2: Create Partition Boundary Ports.....	10
2.3. Step 3: Compile & Check Debug Nodes.....	12
2.4. Step 4: Export the Core Partition & Create the Black Box File.....	13
2.5. Step 5: Copy Files to Consumer Project.....	15
2.6. Step 6: Create a Signal Tap File (Optional).....	15
2.7. Step 7: Device Programming and Hardware Verification.....	17
2.8. Step 8: Hardware Verification with Signal Tap.....	18
3. Core Partition Reuse Debug—Consumer.....	20
3.1. Step 1: Add Files & Run Synthesis.....	20
3.2. Step 2: Create a Signal Tap File.....	21
3.3. Step 3: Create a Partition for blinking_led_top.....	23
3.4. Step 4: Compile the Design & Verify Debug Nodes.....	23
3.5. Step 5: Device Programming and Hardware Verification.....	24
3.6. Step 6: Hardware Verification with Signal Tap.....	24
4. Root Partition Reuse Debug—Developer.....	26
4.1. Step 1: Create a Periphery Reuse Core Partition & Define a Logic Lock Region.....	26
4.2. Step 2: Generate and Instantiate SLD JTAG Bridge Agent in the Root Partition.....	27
4.3. Step 3: Generate and Instantiate the SLD JTAG Bridge Host.....	28
4.4. Step 4: Generate HDL Instance of Signal Tap	29
4.5. Step 5: Compile, Export Root Partition and Copy Files to Consumer Project.....	29
4.6. Step 6: Device Programming and Hardware Verification.....	30
4.7. Step 7: Generate a Signal Tap File for the Root Partition.....	31
4.8. Step 8: Hardware Verification with Signal Tap.....	31
5. Root Partition Reuse Debug—Consumer.....	34
5.1. Step 1: Add Files to Customer Project.....	34
5.2. Step 2: Generate and Instantiate SLD JTAG Bridge Host in PRC Partition.....	35
5.3. Step 3: Synthesize, Create Signal Tap File and Compile	35
5.4. Step 4: Device Programming and Hardware Verification.....	36
5.5. Step 5: Hardware Verification of PRC Partition with Signal Tap.....	36
5.6. Step 6: Hardware Verification of Root Partition with Signal Tap.....	39
6. Document Revision History for AN 847: Signal Tap Tutorial with Design Block Reuse for Intel Arria 10 FPGA Development Board.....	40



1. Introduction

The Intel® Quartus® Prime Pro Edition software supports verification of block-based design flows with the Signal Tap logic analyzer. This tutorial demonstrates how to incorporate internal signal verification into design block reuse flows in the Intel Quartus Prime Pro Edition software.

A design block is the logic comprising a hierarchical design instance. Block-based design flows⁽¹⁾ enable preservation of blocks within a project via Incremental Block-Based Compilation, as well as reuse of design blocks in other projects via Design Block Reuse. To preserve or reuse a design block, you must designate the block as a *design partition*.

Setting up a project using design block reuse flow for verification requires planning to ensure communication between logic in the partition and the Signal Tap logic analyzer. The preparation steps depend on whether you are reusing a core partition or a root partition.

This tutorial assumes basic understanding of reusing design blocks. For information about designing with reusable blocks, refer to the *Block-Based Design User Guide: Intel Quartus Prime Pro Edition*. For step-by-step instructions on reusing design blocks, refer to *AN 839: Design Block Reuse Tutorial for Intel Arria® 10 FPGA Development Board*.

Related Information

- [Block-Based Design Flows](#)
In *Block-Based Design User Guide: Intel Quartus Prime Pro Edition*
- [AN 839: Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board](#)

1.1. Signal Tap with Core Partition Reuse

In a core partition reuse flow, the Developer identifies signals to use for debug, and adds those signals as ports to the partition boundary, so the signals are visible in the Consumer project.

(1) This feature has some known limitations. Refer to [Why I can't compile Intel Stratix 10 partitions exported from another project with a different top level](#) and [Internal Error: Sub-system: PTI, File: /quartus/tsm/pti/pti_tdb_builder.cpp](#).

The Developer can use two methods to add Signal Tap to the core partition:

- **Signal Tap HDL instance:** In this method, the Developer defines a Signal Tap HDL instance in the core partition. During synthesis, the Compiler creates partition boundary ports for all the nodes that connect to the Signal Tap input port. Finally, the Developer exports the .qdb file and generates a supporting black box file. This black box file defines the ports for the block that you reuse.

The Consumer must generate one Signal Tap file for each HDL instance present in the design. The parent partition and the reused core partition have separate Signal Tap files

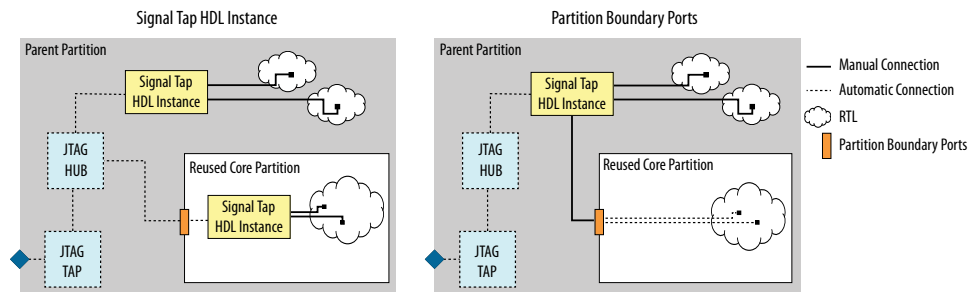
- **Create partition boundary ports:** In this method, the Developer directly assigns signals as ports to the partition boundary, through a QSF assignment or with the Assignment Editor. Assigning boundary ports simplifies the management of hierarchical blocks, by automatically creating ports and tunneling through layers of logic, without making RTL changes.

The Developer must include the user created partition boundary ports in the black box file. This action allows the Consumer to tap these ports as pre-synthesis or post-fit nodes.

The Consumer can add the Signal Tap logic analyzer to the parent partition with any of these methods:

- Signal Tap HDL instance
- Signal Tap GUI to tap pre-synthesis Nodes.
- Signal Tap GUI to tap post-fit nodes.

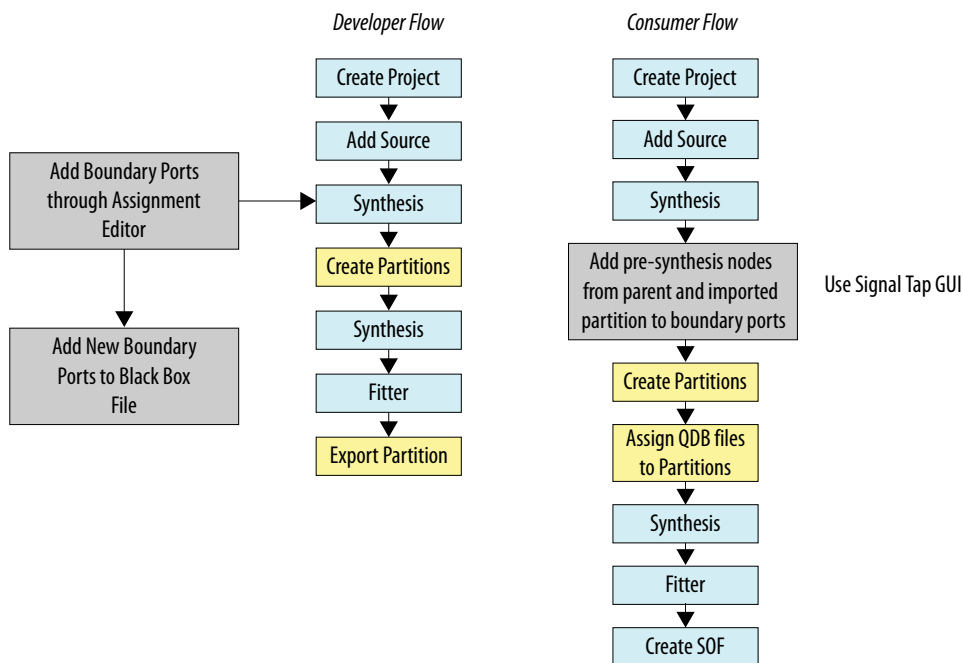
Figure 1. Debug Setup with Reused Core Partition





In this core partition reuse tutorial, the Developer creates partition boundary ports with the Assignment Editor, and the Consumer adds pre-synthesis nodes to the parent partition with the Signal Tap GUI. The following figure describes the Developer and Consumer flows:

Figure 2. Tutorial Design Flow for Core Partition Reuse



For more information, please refer to *Debugging Block-Based Designs Using Signal Tap* in the *Block-Based Design User Guide: Intel Quartus Prime Pro Edition*.

Related Information

Using HDL Signal Tap Instances

In *Block-Based Design User Guide: Intel Quartus Prime Pro Edition*

1.2. Signal Tap with Root Partition Reuse

In a root partition reuse flow, the Developer extends the debug fabric into the periphery reuse core (PRC) partition with a debug bridge. The debug bridge also allows debugging of the PRC partition in the Consumer project. The components of a debug bridge are:

- SLD JTAG Bridge Agent Intel FPGA IP—Instantiate in the higher-level partition to connect to an SLD JTAG Bridge Host in the child partition.
- SLD JTAG Bridge Host Intel FPGA IP—Instantiate in the child partition to connect to an SLD JTAG Bridge Agent in the higher-level partition.

With the bridge in place, the Developer adds the Signal Tap logic analyzer to the root partition with either of the following methods:

- Signal Tap HDL instance
- Signal Tap GUI to tap pre-synthesis or post-fit nodes

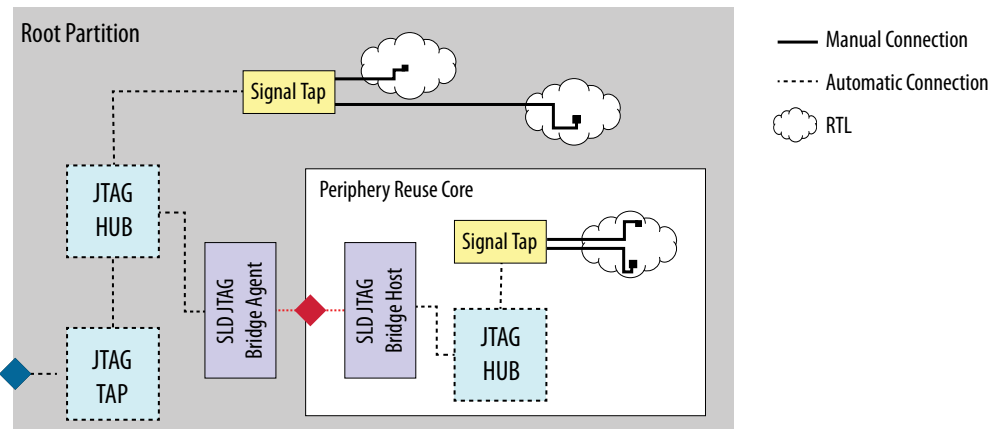
The Intel Quartus Prime Pro Edition software supports multiple instances of bridge components in partitions and their children. The Compiler assigns an index number to distinguish each instance. The bridge index for the root partition is always **None**. You can view the bridge index for child partitions in the synthesis report, under **JTAG Bridge Agent Instance Information**.

Each instance of the Signal Tap logic analyzer can only connect within the partition that the instance resides. Therefore, the root partition and PRC partition require separate Signal Tap files in this flow.

The Consumer must instantiate the SLD JTAG Bridge Host in the PRC partition, and add Signal Tap to the PRC partition with either of the following methods:

- Signal Tap HDL instance
- Signal Tap GUI to tap pre-synthesis or post-fit nodes

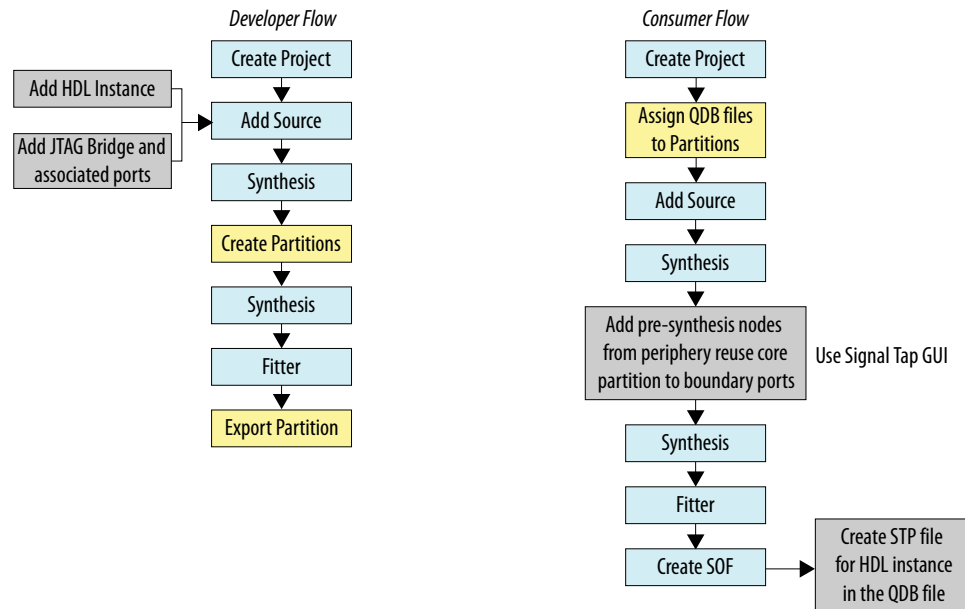
Figure 3. Debug Setup with Reused Root Partition



In this root partition reuse tutorial, the Developer adds an HDL instance of the Signal Tap logic analyzer to debug the root partition, and adds bridge components to enable debug of the PRC partition. Then, in the Consumer project, the Consumer adds the SLD JTAG Bridge Host to enable debug of the PRC partition, and uses the Signal Tap GUI to tap pre-synthesis nodes.



Figure 4. Tutorial Design Flow for Root Partition Reuse



Related Information

- [Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer](#)
In *Debug Tools User Guide: Intel Quartus Prime Pro Edition*
- [SLD JTAG Bridge Intel FPGA IP](#)
In *Debug Tools User Guide: Intel Quartus Prime Pro Edition*

1.3. Tutorial Software and Hardware

The steps in this tutorial correspond with the following Intel software and hardware:

- Linux installation of Intel Quartus Prime Pro Edition software version 18.0, with Intel Arria 10 device support.
- The Intel Arria 10 GX FPGA Development Kit.

You can adapt this tutorial for Windows and other software or hardware configurations.

Related Information

[AN 839: Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board](#)

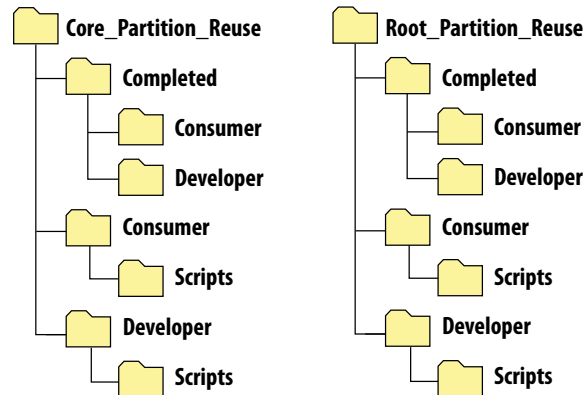
1.4. Tutorial Files

This tutorial includes a design example organized into directories that correspond with the flow (Core or Root partition reuse) and role (Developer or Consumer).

Locate and download the design example at:

https://www.altera.com/content/dam/altera-www/global/en_US/others/literature/an/a10_pcie_devkit_design_block_reuse_stp.zip.

Figure 5. Tutorial Directory Structure



The **Completed** directories contain the final versions of all the files required to complete that tutorial module. You can use the files in the **Completed** directories to bypass tutorial steps, or skip to the final step of the tutorial module. The **Scripts** directories contain bash scripts and files to restore the single project.

- To restore all of the tutorial files to the original run:

```
./restoreall.sh
```

from the `a10_devkit_design_block_reuse_stp`

- To restore a single project, run:

```
Script/restore.sh
```

from the `Consumer` or `Developer` directory.



2. Core Partition Reuse Debug—Developer

Process Description

In this tutorial module, the Developer assigns signals as ports to the partition boundary using the Assignment Editor, and then exports the core partition to a .qdb file. As a result, these user created boundary ports are available for debug as pre-synthesis nodes in the Consumer project, as a part of the reused .qdb file.

Completed Tutorial Files

In the a10_pcie_devkit_design_block_reuse_stp folder, the Core_Partition_Reuse/Completed/Developer/ directory contains the completed files for this tutorial module.

Completed Tutorial Files

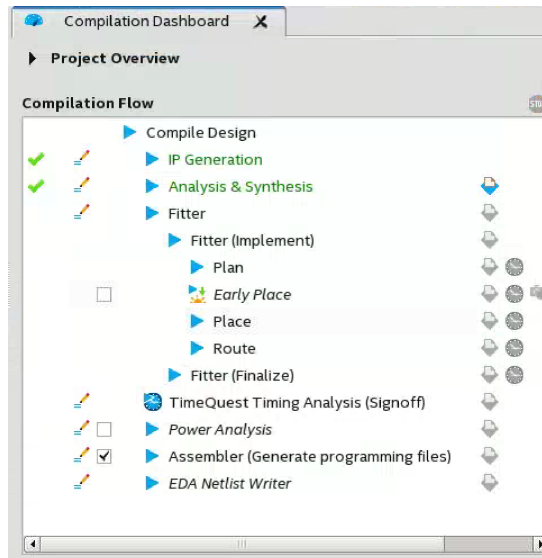
This tutorial module includes the following steps:

- [Step 1: Create a Core Partition](#) on page 9
- [Step 2: Create Partition Boundary Ports](#) on page 10
- [Step 3: Compile & Check Debug Nodes](#) on page 12
- [Step 4: Export the Core Partition & Create the Black Box File](#) on page 13
- [Step 5: Copy Files to Consumer Project](#) on page 15
- [Step 6: Create a Signal Tap File \(Optional\)](#) on page 15
- [Step 7: Device Programming and Hardware Verification](#) on page 17
- [Step 8: Hardware Verification with Signal Tap](#) on page 18

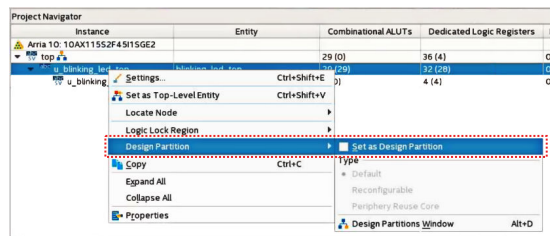
2.1. Step 1: Create a Core Partition

During this step, you open the project, run synthesis, and define a design partition for the core logic.

1. In the Intel Quartus Prime Pro Edition software, click **File ► Open Project**, and open the a10_pcie_devkit_design_block_reuse_stp/Core_Partition_Reuse/Developer/top.qpf project file.
2. On the Compilation Dashboard, click **Analysis & Synthesis** to synthesize the design. When synthesis is complete, the Compilation Dashboard displays a check mark.



- In the Project Navigator, right-click the **u_blinking_led_top** instance in the **Hierarchy** tab, and then click **Design Partition > Set as Design Partition**. A design partition icon appears next to each instance you assign.



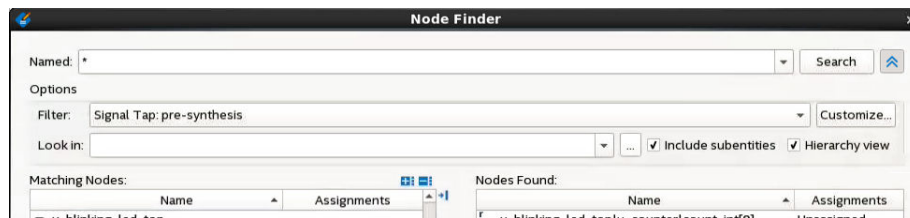
2.2. Step 2: Create Partition Boundary Ports

Partition Boundary Ports make signals visible from the parent partition.

- In the Intel Quartus Prime Software, click **Assignments > Assignment Editor**.
- In the **Assignment Editor** window, locate the row that says **<<new>>** at the bottom of the window.
- Double-click the **To** column, and then click the Node Finder button to open the Node Finder.

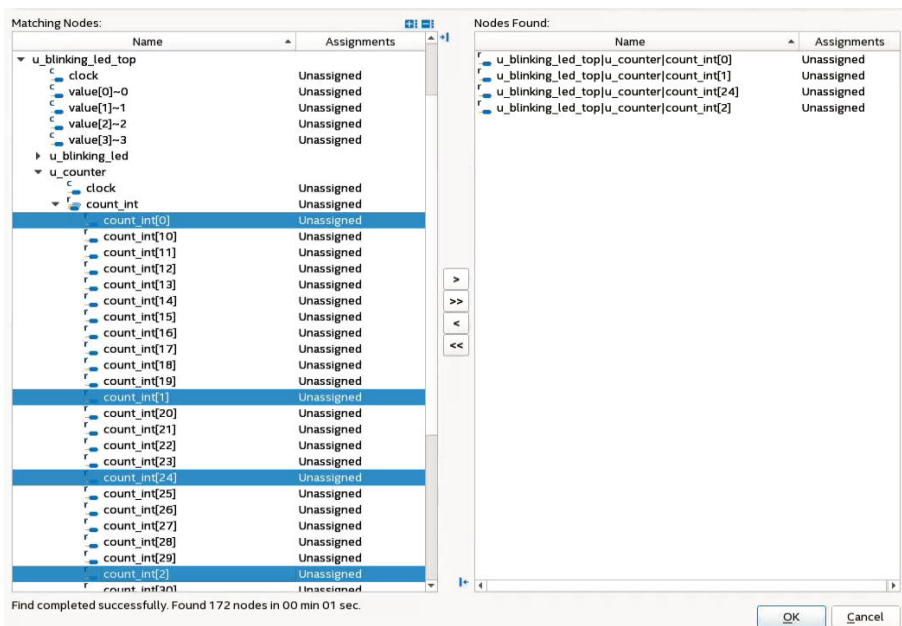
	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
34	✓ Ok		out LED[1]	I/O Standard	1.8 V	Yes	top		
35	✓ Ok		out LED[0]	I/O Standard	1.8 V	Yes	top		
36	✓ Ok		u_blinking_led_top	Partition	blink..ed_top	Yes	top		
37		<<new>>		<<new>>					

- In the **Node Finder** window, type ***** in the **Named** section, set **Filter** to **Signal Tap: pre-synthesis**, and then click **Search**.

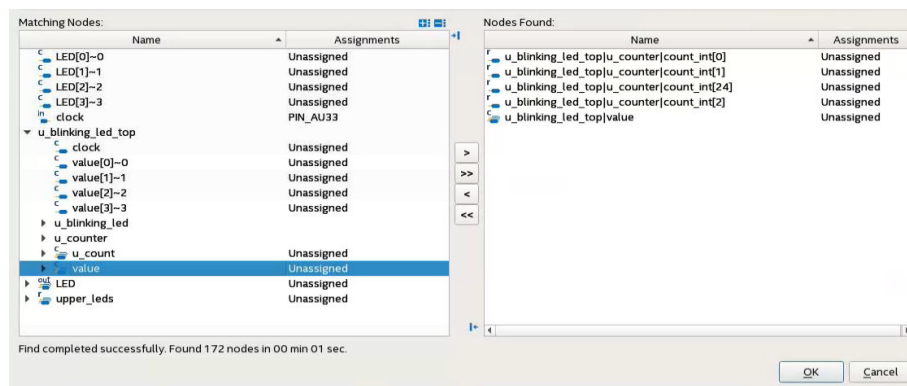


The **Matching Nodes** list shows the signals that match the search criteria.

5. In the **Matching Nodes** list, expand **u_blinking_led_top|u_counter|count_int**.
6. Select **count_int[0]**, **count_int[1]**, **count_int[2]**, and **count_int[24]**, and click **>** to move to the **Nodes Found** list. Do not click **OK**.



7. In the **Matching Nodes** list, expand **u_blinking_led_top**, and select **value**. Click **>**.



8. Click **OK** to close the Node Finder.



- In the **Assignment Editor** window, for each of these nodes, populate information for rest of the columns.

For example, for node **u_blinking_led_top|u_blinking_led|value**, double-click **Assignment Name** and select **Create Partition Boundary Ports**. Double-click **value** to provide a port name **db_value**. Leave rest of the columns as default.

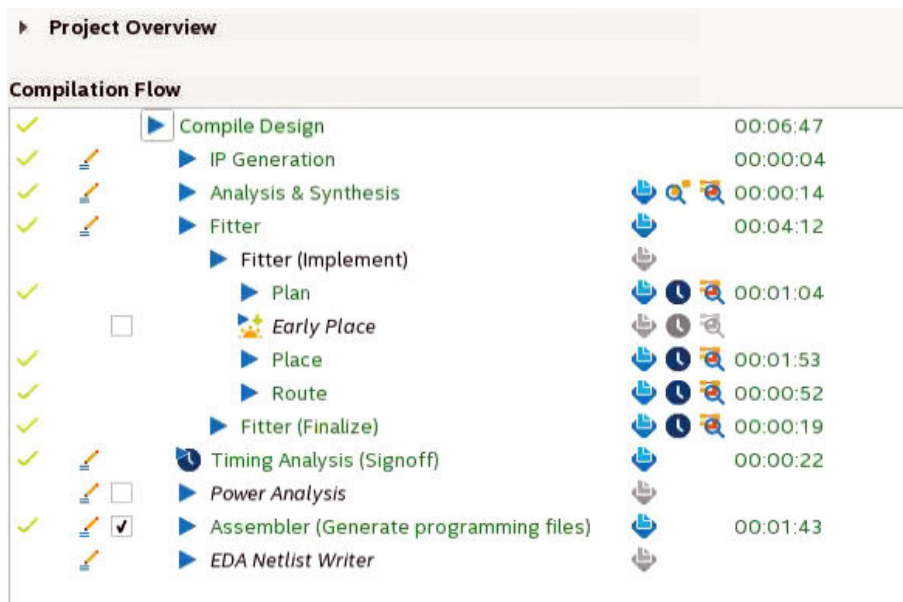
Status	From	To	Assignment Name	Value	Enabled	Er
✓ Ok		u_blinking_led_top	Partition	blink.ed_top	Yes	top
✓ Ok		u_blinking_led_top value	Create Partition Boundary Ports	db_value	Yes	
✓ Ok		u_blinking_led_top u_counter count_int[0]	Create Partition Boundary Ports	db_count_0	Yes	
✓ Ok		u_blinking_led_top u_counter count_int[1]	Create Partition Boundary Ports	db_count_1	Yes	
✓ Ok		u_blinking_led_top u_counter count_int[2]	Create Partition Boundary Ports	db_count_2	Yes	
✓ Ok		u_blinking_led_top u_counter count_int[24]	Create Partition Boundary Ports	db_count_24	Yes	
	<<new>>	<<new>>	<<new>>			

- Click **File ► Save** to save the changes.
- Optionally, you can verify the assignments in the `top.qsf` file, located in `a10_pcie_devkit_design_block_reuse_stp/Core_Partition_Reuse/Developer`. In the file, check for the following assignments:

```
set_instance_assignment -name CREATE_PARTITION_BOUNDARY_PORTS db_count_1 \
    -to u_blinking_led_top|u_counter|count_int[1]
set_instance_assignment -name CREATE_PARTITION_BOUNDARY_PORTS db_count_24 \
    -to u_blinking_led_top|u_counter|count_int[24]
set_instance_assignment -name CREATE_PARTITION_BOUNDARY_PORTS db_count_2 \
    -to u_blinking_led_top|u_counter|count_int[2]
set_instance_assignment -name CREATE_PARTITION_BOUNDARY_PORTS db_value \
    -to u_blinking_led_top|value
set_instance_assignment -name CREATE_PARTITION_BOUNDARY_PORTS db_count_0 \
    -to u_blinking_led_top|u_counter|count_int[0]
```

2.3. Step 3: Compile & Check Debug Nodes

- Click **Compile Design** on the Compilation Dashboard.
Check marks ✓ indicate when each stage of compilation is complete.



2. Open the compilation report by clicking **Processing > Compilation Report**.
3. Under **Table of Contents**, find **Synthesis > In-System Debugging > Create Partition Boundary Ports**.

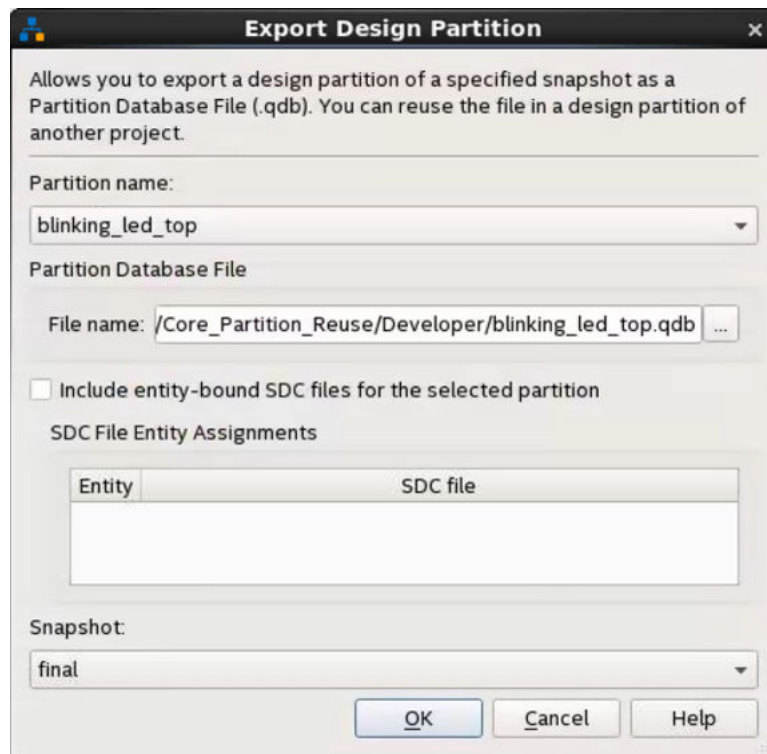
The **Create Partition Boundary Ports** table lists all the boundary ports.

Create Partition Boundary Ports		
Q <<Filter>>		
	Partition Hierarchy Name	Create Partition Boundary Port Hierarchy Name
1	u_blinking_led_top	u_blinking_led_top db_count_0
2	u_blinking_led_top	u_blinking_led_top db_count_1
3	u_blinking_led_top	u_blinking_led_top db_count_2
4	u_blinking_led_top	u_blinking_led_top db_count_24
5	u_blinking_led_top	u_blinking_led_top db_value_0
6	u_blinking_led_top	u_blinking_led_top db_value_1
7	u_blinking_led_top	u_blinking_led_top db_value_2
8	u_blinking_led_top	u_blinking_led_top db_value_3

2.4. Step 4: Export the Core Partition & Create the Black Box File

After compilation, you export the core partition and create a supporting black box port definitions file. This tutorial reuses the final snapshot.

1. Click **Project > Export Design Partition**. Select **blinking_led_top** for the **Partition name**, and the **final Snapshot** for export.



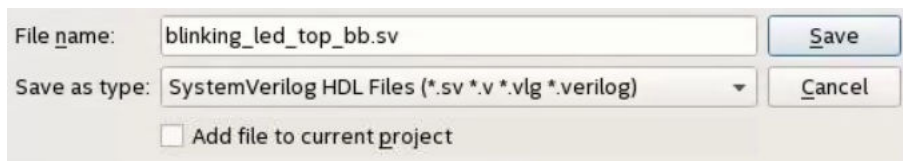
2. Confirm `blinking_led_top.qdb` as the **Partition Database File** name, and then click **OK**. The final `blinking_led.qdb` that you export preserves the placement and routing information from the Developer project reused in the Consumer project.
3. To create the black box file, click **File > New**, select **SystemVerilog HDL File** under **Design Files**, and then click **OK**.
4. Create a file that contains the port definitions for the partition you export and the partition boundary ports created in the previous step.

Make sure to include any Verilog parameters or VHDL generics in the definition. The port definitions in the black box file must look just like the original, without the logic RTL.

```
module blinking_led_top(
    output [3:0] value,
    input clock,
    output db_count_0,
    output db_count_1,
    output db_count_2,
    output db_count_24,
    output db_value_0,
    output db_value_1,
    output db_value_2,
    output db_value_3
);

endmodule
```

5. Save the black box file as `blinking_led_bb.sv`. When saving this file, turn off the option to **Add file to current project**.



2.5. Step 5: Copy Files to Consumer Project

After exporting the core partition and creating the black box file, copy the files to the Consumer project directory for subsequent use.

1. Copy the `blinking_led_top.qdb` and `blinking_led_top_bb.sv` files to the `Core_Partition_Reuse/Consumer/` directory.

Note: The `.sdc` file does not influence the logic, placement, or routing of a finalized partition in the Consumer project.

In the *Core Partition Reuse Debug-Consumer* tutorial module, the Consumer integrates the `blinking_led_top.qdb` and `blinking_led_top_bb.sv` files into the Consumer project.

Related Information

[Step 1: Add Files & Run Synthesis](#) on page 20

2.6. Step 6: Create a Signal Tap File (Optional)

In this step, you configure the Signal Tap logic analyzer, and then tap partition boundary ports and pre-synthesis nodes from the parent partition to debug in the Developer project. Configuring the Signal Tap logic analyzer includes adding a reference clock and specifying acquisition settings.


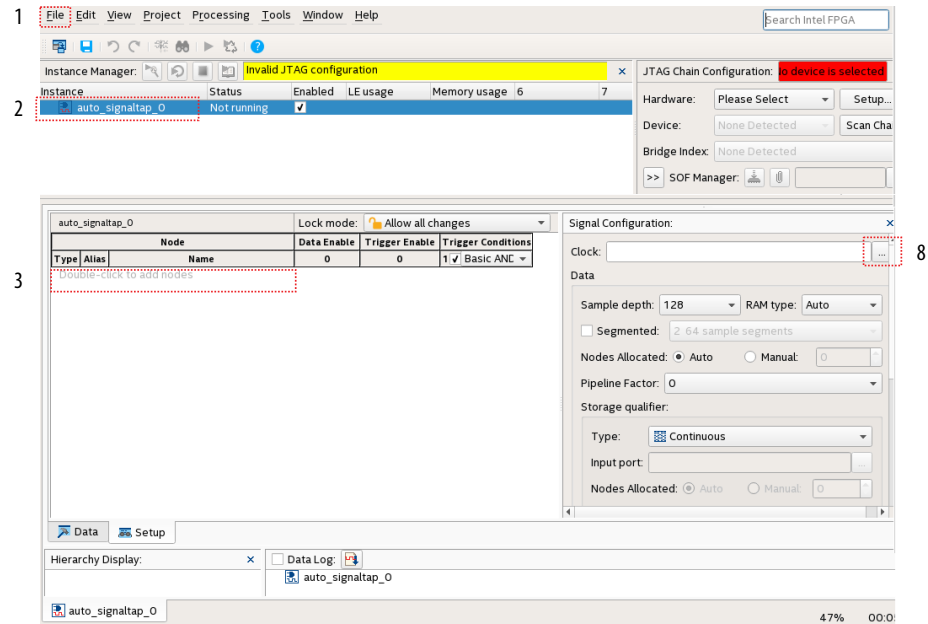
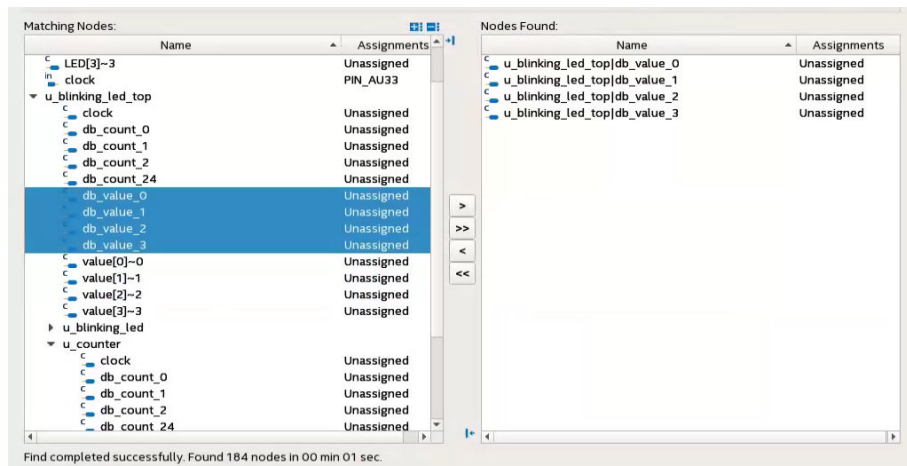
1. In the Intel Quartus Prime Pro Edition software, click **Tools** ►  **Signal Tap Logic Analyzer**.

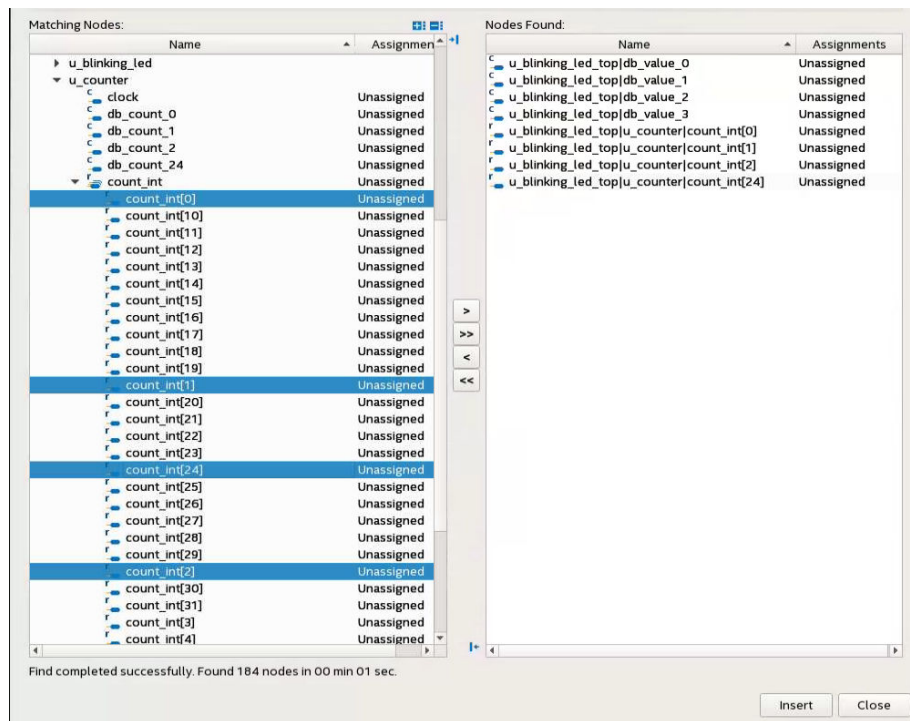
Figure 6. Signal Tap Logic Analyzer Window



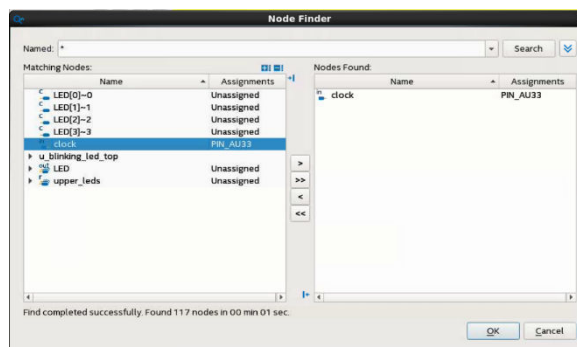
2. In the Instance Manager, click **auto_sigtap_0**.
3. In the **Setup** tab, double-click to open the Node Finder.
4. In the **Node Finder** window, type * in the **Named** section, set **Filter** to **Signal Tap: pre-synthesis**, and then click **Search**.
5. In the **Matching Nodes** list, expand **u_blinking_led_top**. Select **db_value_0**, **db_value_1**, **db_value_2**, and **db_value_3**.



6. In the **Matching Nodes** list, expand the **u_blinking_led_top|u_counter|count_int**.
7. From the node list, select **count_int[0]**, **count_int[1]**, **count_int[2]**, and **count_int[24]**, and insert the nodes by clicking **>**. Click **Insert**, and then **Close**.



8. In the Signal Tap window, under **Signal Configuration**, click (...) next to **Clock**.
9. In the Node Finder, search for *, and select the clock. Click >, and then click **OK** to close.



10. Leave all the other options as default under **Signal Configuration**. Go to **File** ➤ **Save** and save the file as `stp_core_partition_reuse.stp`.
A dialog box appears asking if you want to enable Signal Tap file for the project.
11. Click **Yes**, and close the file.
12. Click **Compile Design** on the Compilation Dashboard.
Check marks ✓ indicate finished Compiler modules.

2.7. Step 7: Device Programming and Hardware Verification

1. Program the device.



For instructions to program the device, refer to *Device Programming* in AN 839: *Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board*.

2. Verify the LEDs behavior.

After completing this tutorial module:

- LEDs D6-D3 map to the `blinking_led_top` core.
- LEDs D10-D7 map to the top-level design.

The `blinking_led_top` core flashes red in binary order, and the top-level design does not illuminate any LEDs.

Related Information

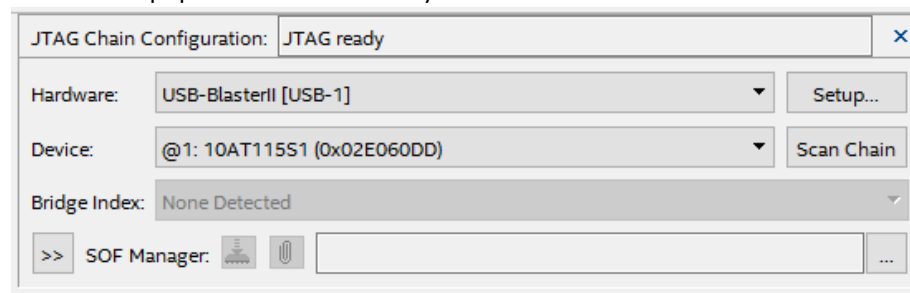
Device Programming

In AN 839: *Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board*

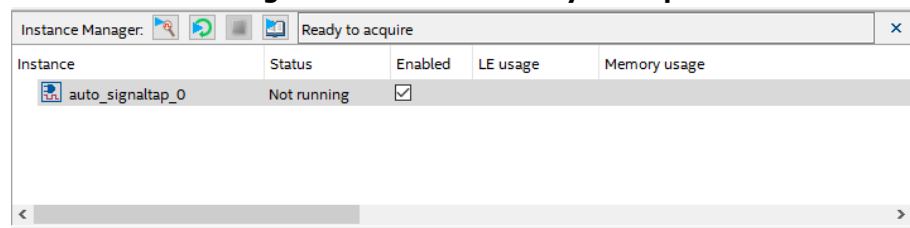
2.8. Step 8: Hardware Verification with Signal Tap

1. In the Signal Tap window, click **File ► Open**, and open `stp_core_partition_reuse.stp`.
2. Make sure that the development kit is powered on and connected to the machine from which you open the Signal Tap Logic Analyzer.
3. In the **JTAG Chain Configuration** tab, set up the JTAG connection to the board by clicking **Setup** and then selecting the **USB-BlasterII** under **Hardware**.

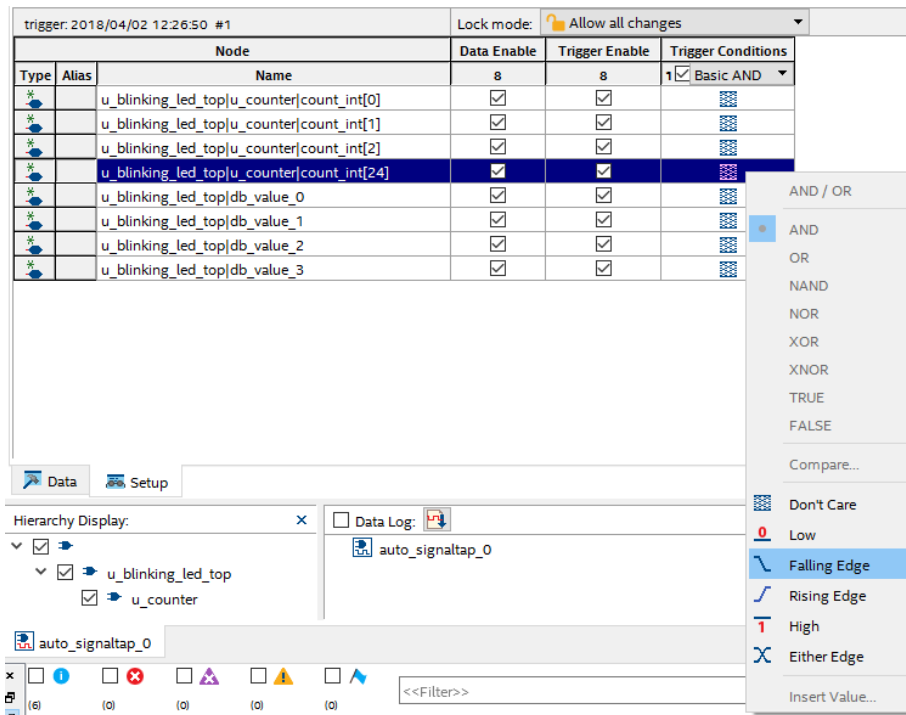
The device populates automatically.



The **Instance Manager** window shows **Ready to acquire**.

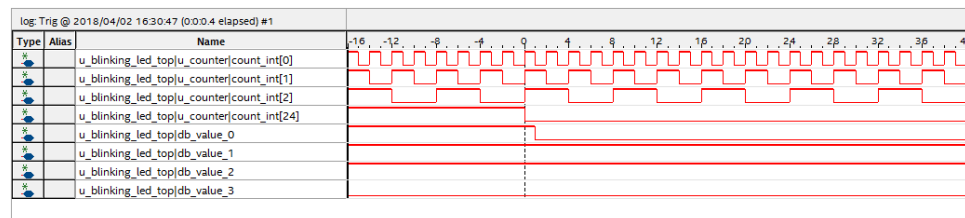


4. To set the trigger condition, select `count_int[24]`, right-click the column under **Trigger Conditions**, and set to **Falling Edge**.

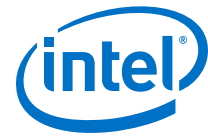


- Run analysis by clicking the  icon, next to the **Instance Manager**. When the analysis finishes, the **Waveform** tab shows the captured data.

Figure 7. Waveform after Signal Tap Analysis



- The count_int[27:24] register in u_blinking_led_top|u_counter | count_int[27:24] drives u_blinking_led_top|u_blinking_led | value[3:0].
- The partition boundary ports created for each bit of value[3:0] are db_value_3, db_value_2, db_value_1, and db_value_0.
- The value of db_value_0 changes a cycle later after count_int[24] transitions to 0. The count_int[2:0] shows the transitioning of the counter during this process



3. Core Partition Reuse Debug—Consumer

In this tutorial, the Consumer receives a final core partition with boundary ports that correspond to signals useful for debugging. The Consumer adds the black box file and assigns the .qdb in the Consumer project. Then, the Consumer debugs the parent and reused core partition with a Signal Tap HDL instance, tapping partition boundary ports as pre-synthesis nodes.

Because the exported .qdb includes compiled netlist information, the Consumer project must target the same FPGA device part number, and use the same Intel Quartus Prime version as the Developer project.

Completed Tutorial Files

The Core_Partition_Reuse/Completed/Consumer/ tutorial directory contains the completed files for this tutorial module.

Tutorial Module Steps

This tutorial module includes the following steps:

- [Step 1: Add Files & Run Synthesis](#) on page 20
- [Step 2: Create a Signal Tap File](#) on page 21
- [Step 3: Create a Partition for blinking_led_top](#) on page 23
- [Step 4: Compile the Design & Verify Debug Nodes](#) on page 23
- [Step 5: Device Programming and Hardware Verification](#) on page 24
- [Step 6: Hardware Verification with Signal Tap](#) on page 24

3.1. Step 1: Add Files & Run Synthesis

Incorporate the black-box file to the Consumer project.

1. In the Intel Quartus Prime Pro Edition software, click **File ► Open Project**, and open the `a10_pcie_devkit_design_block_reuse_stp/Core_Partition_Reuse/Consumer/top.qpf` project file.
2. Click **Project ► Add/Remove Files in Project**.
3. On the **Files** pane, click the browse (...) button near **File name** to locate and select the `/Core_Partition_Reuse/Consumer/bleinking_led_top_bb.sv` black box file.
4. Click **Open**, and then click **OK**.
The file is now a source file in the project.
5. On the Compilation Dashboard, click **Analysis & Synthesis** to synthesize the design. When synthesis is complete, the Compilation Dashboard displays a check mark.



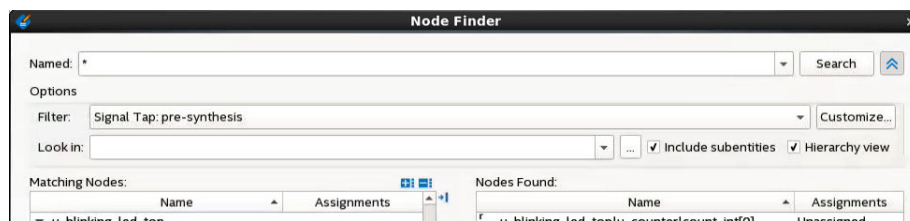
Related Information

Step 5: Copy Files to Consumer Project on page 15

3.2. Step 2: Create a Signal Tap File

Create a Signal Tap file that includes partition boundary ports from the reused core partition.

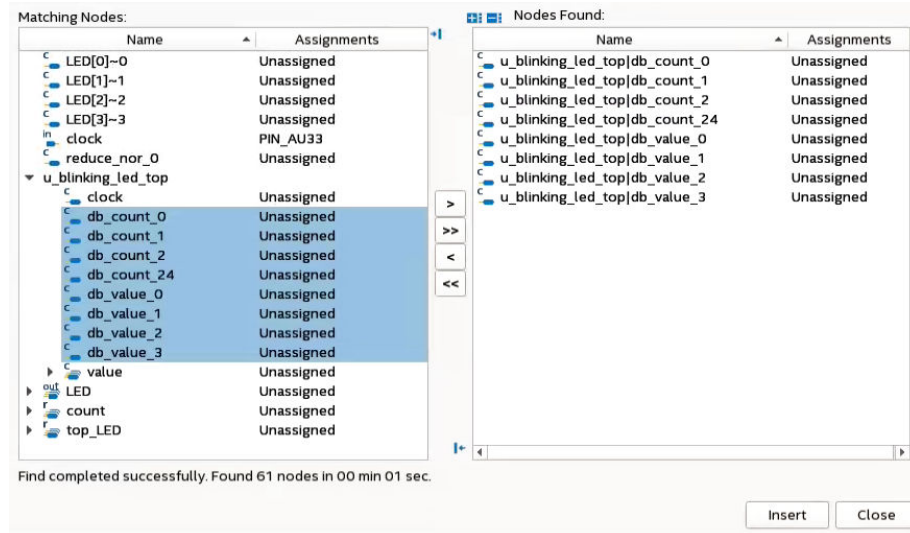
1. In the Signal Tap Logic Analyzer GUI, click **auto_sigtap_0**.
2. In the **Setup** tab, double-click to open the Node Finder.
3. In the **Node Finder** window, type * in the **Named** section, set **Filter** to **Signal Tap: pre-synthesis**, and then click **Search**.



The **Matching Nodes** list shows the signals that match the search criteria.

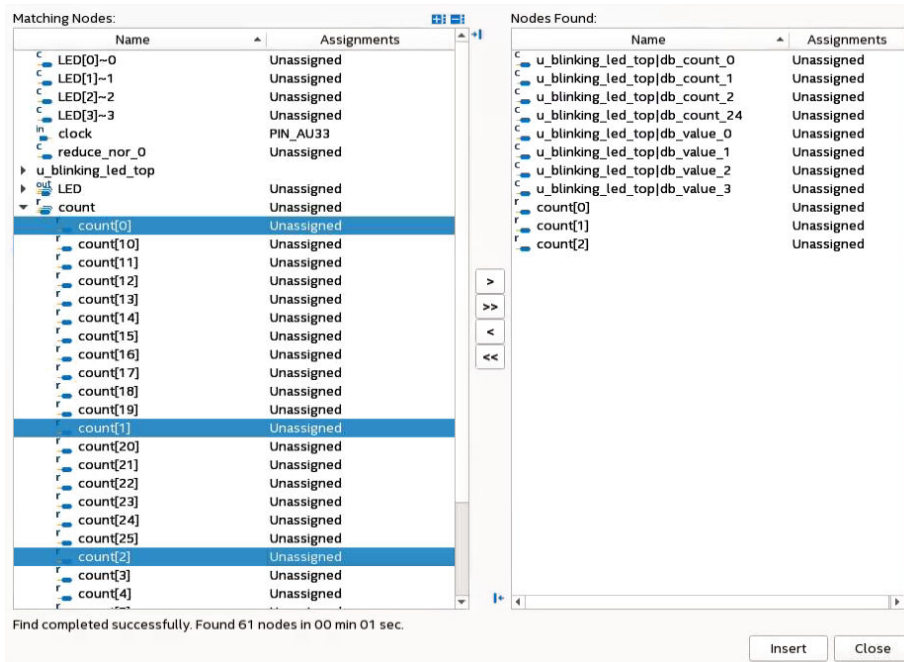
4. Add the boundary ports from the imported partition (**db_***) from **u_blinking_led_top** to the **Nodes Found** list.

The names must match what appears in the Developer project's report file.

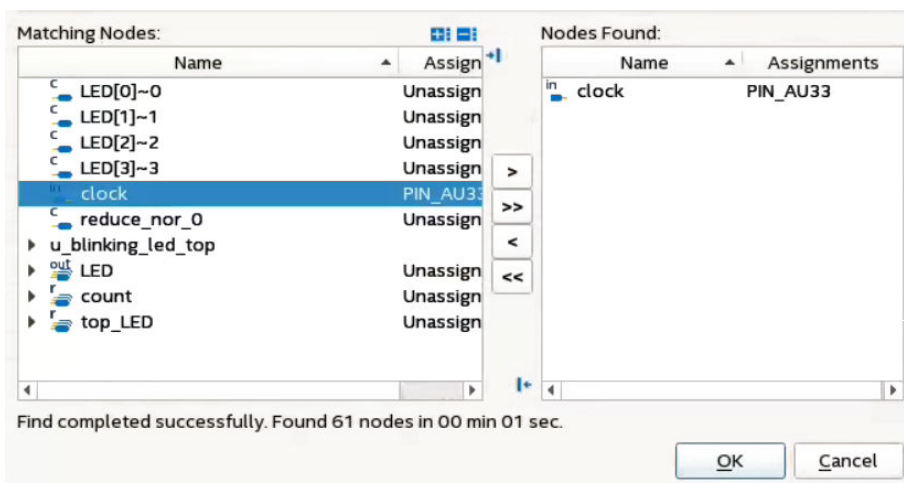


Note: If you add nodes other than the **db_*** nodes from the imported partition, the Compiler leaves them unconnected.

5. Add nodes **count[2:0]** from the **root partition** to the **Nodes Found** list, and click **OK**.



6. Configure the Signal Tap acquisition. The following figure specifies the clock source:



Details appear in the *Core Partition Reuse Debug - Developer* tutorial module, *Step 6: Create a Signal Tap File (Optional)* (1-4).

7. Click **File ► Save**, and save the file as `stp_core_partition_reuse.stp`. A dialog box appears asking if you want to enable Signal Tap file for the project.
8. Click **Yes**, and close the file.

Related Information

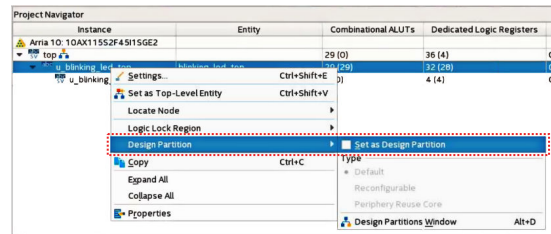
[Step 6: Create a Signal Tap File \(Optional\)](#) on page 15



3.3. Step 3: Create a Partition for blinking_led_top

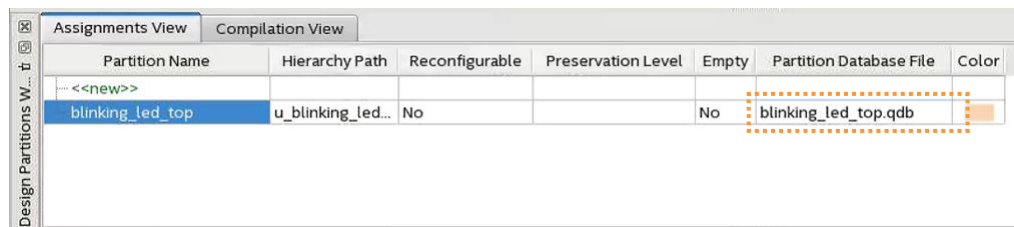
Create a new partition and assign the .qdb file from the Developer project to the partition.

1. In the Project Navigator, right-click the **u_blinking_led_top** instance in the **Hierarchy** tab, and then click **Design Partition > Set as Design Partition**. A design partition icon  appears next to each instance you assign.



2. If the **Design Partition Window** is not visible on the GUI, click **Assignments > Design Partitions Window**.
3. Double-click in the **Partition Database File** cell, and then click browse (...). Select the **blinking_led_top.qdb** file copied from the Developer project.

Figure 8. qdb Assignment in Design Partitions Window





4. Click the partition name **blinking_led_top** to deselect the **Partition Database File** column. This action confirms the .qdb file assignment.

Related Information

Step 5: Copy Files to Consumer Project on page 15

3.4. Step 4: Compile the Design & Verify Debug Nodes

After creating Signal Tap files for the parent and blinking_led_top partitions, you are ready to run a full compilation of the design. Then, verify the debug nodes connection.

1. Click  **Compile Design** on the Compilation Dashboard. Check marks  indicate finished Compiler modules.
2. In the **Compilation Report**, under **Synthesis**, view the **In-System Debugging > Connections to In-System Debugging Instance** "auto_signaltap_0" report to verify the connection of the ports.
In the report, the **Status** column shows the connected debug ports.

	Name	Type	Status	Partition Name	Actual Connection	Debug Port	Details
1	clock	Pre-Synthesis	Connected	root_partition	clock	acq_clk	N/A
2	count[0]	Pre-Synthesis	Connected	root_partition	count[0]	acq_data_in[0]	N/A
3	count[1]	Pre-Synthesis	Connected	root_partition	count[1]	acq_data_in[1]	N/A
4	count[2]	Pre-Synthesis	Connected	root_partition	count[2]	acq_data_in[2]	N/A
5	u_blinking_led_top[db_count_0]	Pre-Synthesis	Connected	root_partition	u_blinking_led_top-db_count_0	acq_data_in[3]	N/A
6	u_blinking_led_top[db_count_1]	Pre-Synthesis	Connected	root_partition	u_blinking_led_top-db_count_1	acq_data_in[4]	N/A
7	u_blinking_led_top[db_count_2]	Pre-Synthesis	Connected	root_partition	u_blinking_led_top-db_count_2	acq_data_in[5]	N/A
8	u_blinking_led_top[db_count_24]	Pre-Synthesis	Connected	root_partition	u_blinking_led_top-db_count_24	acq_data_in[6]	N/A
9	u_blinking_led_top[db_value_0]	Pre-Synthesis	Connected	root_partition	u_blinking_led_top-db_value_0	acq_data_in[7]	N/A
10	u_blinking_led_top[db_value_1]	Pre-Synthesis	Connected	root_partition	u_blinking_led_top-db_value_1	acq_data_in[8]	N/A
11	u_blinking_led_top[db_value_2]	Pre-Synthesis	Connected	root_partition	u_blinking_led_top-db_value_2	acq_data_in[9]	N/A
12	u_blinking_led_top[db_value_3]	Pre-Synthesis	Connected	root_partition	u_blinking_led_top-db_value_3	acq_data_in[10]	N/A

3.5. Step 5: Device Programming and Hardware Verification

You can now verify the results of the Core Partition Reuse—Consumer tutorial module in hardware.

1. Program the device.

For instructions to program the device, refer to *Device Programming* in AN 839: *Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board*.

2. Verify the LEDs behavior.

After completing this tutorial module:

- LEDs D6-D3 map to the `blinking_led_top` core
- LEDs D10-D7 map to the top-level (root) design

After configuring the FPGA, the `blinking_led_top` core flashes red in binary order. The top-level design shows a shifting bit in green.

Related Information

Device Programming

In AN 839: *Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board*

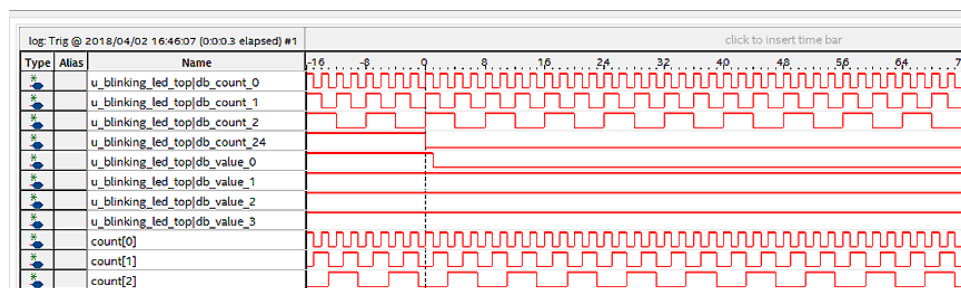
3.6. Step 6: Hardware Verification with Signal Tap

1. In the Signal Tap window, click **File ► Open**, and open `stp_core_partition_reuse.stp`.
2. Make sure that the development kit is powered on and connected to the machine from which you open the Signal Tap Logic Analyzer.
3. Set up the JTAG Chain Configuration, and ensure **Instance Manager** is **Ready to acquire**.

For detailed information, refer to *Step 8: Hardware Verification with Signal Tap* in the *Core Partition Reuse Debug - Developer* tutorial module.

4. As trigger condition, select **db_count_24**, right click the column under **Trigger Conditions**, and set to **Falling Edge**.

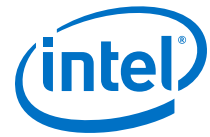
5. Run analysis by clicking the  icon, next to the **Instance Manager**. When the analysis finishes, the **Waveform** tab shows the captured data.



- db_value_0 and db_count_24 signals behaves identically to the Developer flow tutorial.
- The db_value_0 changes as per db_count_24 a cycle later.
- The db_value_* and db_count_* are the partition boundary ports from the imported partition.
- db_count_0, db_count_1 and db_count_2 signals show the transition of the counter inside the imported partition.
- The count[0], count[1], count[2] signals show the transition of another counter in the parent partition during this process.

Related Information

Step 8: Hardware Verification with Signal Tap on page 18



4. Root Partition Reuse Debug—Developer

Process Description

The Developer adds bridge components to enable debug of the PRC partition, and adds a Signal Tap HDL instance to debug the root partition. Then, the Developer compiles and exports the root partition, including logic and periphery resources, and finally, copies the root_partition .qdb and .sdc files to the Consumer project.

Completed Tutorial Files


In the a10_pcie_devkit_design_block_reuse_stp folder, the Root_Partition_Reuse/Completed/Developer/ directory contains the completed files for this tutorial module.

Steps

This tutorial module includes the following steps:

- [Step 1: Create a Periphery Reuse Core Partition & Define a Logic Lock Region](#) on page 26
- [Step 2: Generate and Instantiate SLD JTAG Bridge Agent in the Root Partition](#) on page 27
- [Step 3: Generate and Instantiate the SLD JTAG Bridge Host](#) on page 28
- [Step 4: Generate HDL Instance of Signal Tap](#) on page 29
- [Step 5: Compile, Export Root Partition and Copy Files to Consumer Project](#) on page 29
- [Step 6: Device Programming and Hardware Verification](#) on page 30
- [Step 7: Generate a Signal Tap File for the Root Partition](#) on page 31
- [Step 8: Hardware Verification with Signal Tap](#) on page 31

4.1. Step 1: Create a Periphery Reuse Core Partition & Define a Logic Lock Region

1. In the Intel Quartus Prime Pro Edition software, click **File ► Open Project**, and open the a10_pcie_devkit_design_block_reuse_stp/Root_Partition_Reuse/Developer/top.qpf project file.
2. On the Compilation Dashboard, click **Analysis & Synthesis** to synthesize the design. When synthesis is complete, the Compilation Dashboard displays a check mark.
3. In the Project Navigator, right-click the **u_blinking_led_top** instance in the **Hierarchy** tab, and then click **Design Partition ► Set as Design Partition**. A design partition icon  appears next to each instance you assign.

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

*Other names and brands may be claimed as the property of others.

ISO
9001:2008
Registered



4. If the **Design Partition Window** is not visible on the GUI, click **Assignments > Design Partitions Window**.
5. For the partition **Type**, select **Periphery Reuse Core**. Leave the other options at the default values.

Figure 9. Set Periphery Reuse Core Partition Type

Assignments View		Compilation View						
Partition Name	Hierarchy Path	Type	Preservation Level	Empty	Partition Database File	Entity Re-binding	Color	
<<new>>								
root_part6...								
blinking_le...	u_blinking_led_top	Default	Not Set	No				
		Reconfigurable						
		Periphery Reuse Core						

6. Right-click the **u_blinking_led_top** instance in the Project Navigator, and click **Logic Lock Region > Create New Logic Lock Region**.
7. To modify the region properties, click **Assignments > Logic Lock Regions Window**.
8. Change the **Width** to **123**, and the **Height** to **61**.
9. In the **Origin** column, specify **X63_Y102**.
10. Enable the **Reserved** and **Core-Only** options.
11. In the **Size/State** column, specify **Fixed/Locked**.
12. Click the **Routing Region** cell. The **Logic Lock Routing Region Settings** dialog box appears.
13. Specify **Fixed with expansion** with **Expansion Length** of **0** for the **Routing Type**. The actual size and location are arbitrary for this tutorial. However, you can view and adjust the Logic Lock Region shape in the Chip Planner.

Region Name	Members	Width	Height	Origin	Reserved	Core-Only	Size/State	Routing Region
Logic Lock Regions								
u_blinking_led_top	u_blinking_led_top	123	61	X63_Y102	On	On	Fixed/Locked	Fixed with expansion 0
<<new>>								

4.2. Step 2: Generate and Instantiate SLD JTAG Bridge Agent in the Root Partition

1. From the IP Catalog (**Tools > IP Catalog**), select and generate the **SLD JTAG Bridge Agent Intel FPGA IP**. Set the name as `debug_agent`.
For details about generating the SLD JTAG Bridge Agent Intel FPGA IP, refer to the *Debug Tools User Guide: Intel Quartus Prime Pro Edition*.
2. Open the `top.sv` file, uncomment lines 56 to 65 and 74 to 79, and save the file.
This action instantiates the SLD JTAG Bridge Agent in the root partition.
Lines 56 to 65:

```
// wire tck, tms, tdi, vir tdi, ena, tdo;
// debug_agent debug_agent_inst (
//     .tck (tck), //output, width=1, connect_to_bridge_host .tck
//     .tms (tms), //output, width=1, .tms
//     .tdi (tdi), //output, width=1, .tdi
//     .vir_tdi (vir tdi), //output, width=1, .vir_tdi
```

```
//      .ena (ena), //output, width=1,      .ena
//      .tdo (tdo) // input, width=1,      .tdo
//      );
```

Lines 74 to 79:

```
//      .tck (tck), //input, width=1, connect_to_bridge_host .tck
//      .tms (tms), //input, width=1,      .tms
//      .tdi (tdi), //input, width=1,      .tdi
//      .vir_tdi (vir tdi), //input, width=1, .vir_tdi
//      .ena (ena), //input, width=1,      .ena
//      .tdo (tdo) //output, width=1,      .tdo
```

Related Information

Instantiating the SLD JTAG Bridge Agent

In *Debug Tools User Guide: Intel Quartus Prime Pro Edition*

4.3. Step 3: Generate and Instantiate the SLD JTAG Bridge Host

1. From the IP Catalog (**Tools ► IP Catalog**), select and generate the **SLD JTAG Bridge Host Intel FPGA IP**. Set the name as `debug_host`.

For details about generating the SLD JTAG Bridge Host Intel FPGA IP, refer to the *Debug Tools User Guide: Intel Quartus Prime Pro Edition*.

2. Open the `blinking_led_top.sv` file, uncomment the lines 25 to 30 and 41 to 48, and save the file.

This action instantiates the SLD JTAG Bridge Host in the PRC partition, connecting the debug fabric to the parent partition.

Lines 25 to 30:

```
//      input wire tck (tck), //connect_to_bridge_host .tck
//      input wire tms (tms), //      .tms
//      input wire tdi (tdi), //      .tdi
//      input wire vir_tdi (vir tdi), //      .vir_tdi
//      input wire ena (ena), //      .ena
//      output wire tdo (tdo) //      .tdo
```

Lines 41 to 48:

```
//      debug_host debug_host_inst (
//      . tck (tck), //input, width=1, connect_to_bridge_host .tck
//      . tms (tms), //input, width=1,      .tms
//      . tdi (tdi), //input, width=1,      .tdi
//      .vir_tdi (vir tdi), //input, width=1, .vir_tdi
//      .ena (ena), //input, width=1,      .ena
//      .tdo () //output, width=1,      .tdo
//      );
```

Related Information

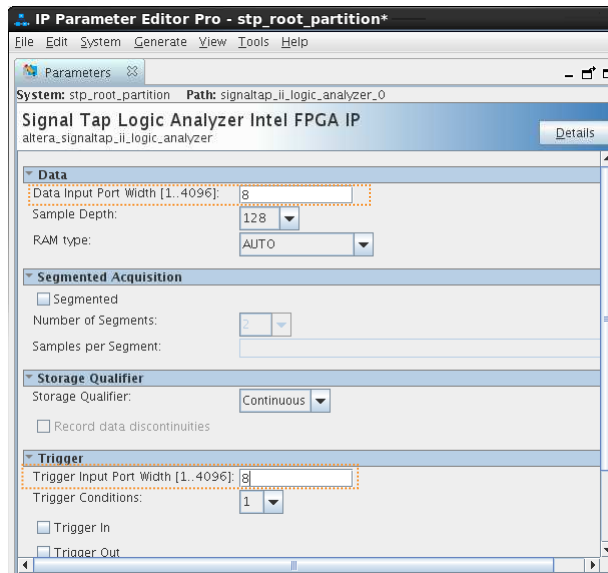
Instantiating the SLD JTAG Bridge Host

In *Debug Tools User Guide: Intel Quartus Prime Pro Edition*



4.4. Step 4: Generate HDL Instance of Signal Tap

1. From the IP Catalog (**Tools > IP Catalog**), select and double-click the **Signal Tap Logic Analyzer Intel FPGA IP**. Set the name as `stp_root_partition`.
2. In the IP Parameter Editor, change **Data Input Port Width** to **8** and **Trigger Input Port Width** to **8**.



3. Generate the IP.
4. In the `top.sv` file, uncomment lines 45 to 49, and save the file.

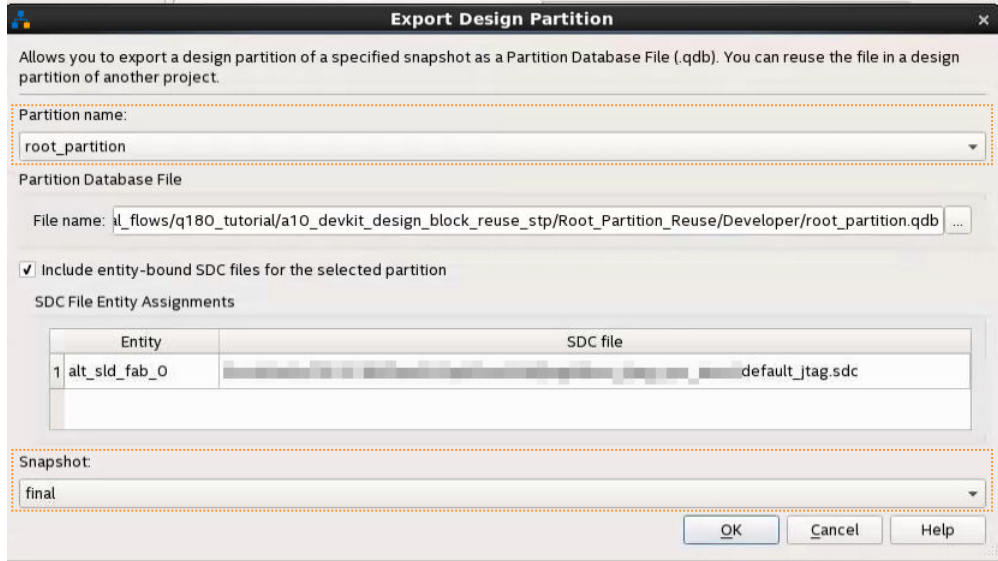
This action instantiates the HDL Signal Tap Logic Analyzer in the root partition.
Lines 45 to 49:

```
// stp_root_partition stp_root_partition inst (
//   .acq_clk (clock), // input, width=1, acq_clk.clk
//   .acq_data_in \
//     ({top_LED, count[3:0]}), // input, width=4, tap.acq_data_in
//   .acq_trigger_in \
//     ({top_LED, count[3:0]}) // input, width=4, tap.acq_trigger_in
// );
```

4.5. Step 5: Compile, Export Root Partition and Copy Files to Consumer Project

When you export the root partition, you include all resources outside of the PRC partition. The logic inside the PRC, including the SLD JTAG Bridge Host, are not exported.

1. Click **Compile Design** on the Compilation Dashboard.
Check marks indicate finished Compiler modules.
2. To export the root partition to a `.qdb` file, click **Project > Export Design Partition**. Select **root_partition** for the **Partition name**, **final** for **Snapshot** and turn on **Include entity-bound SDC files for the selected partition**:



3. Copy the `root_partition.qdb` and `top.sdc` files to the `Root_Partition_Reuse/Consumer/` directory.

When you include entity bound `.sdc` files with the partition export, you need to only copy the top-level `.sdc` file, which is not bound to an entity. The top-level design uses constraints for analysis only, and does not drive any logic or routing.

When reusing the root partition, the Consumer integrates the `root_partition.qdb` and `top.sdc` files into the Consumer project. The Consumer can also include a separate `.sdc` file to constrain the logic that they use in the PRC partition.

The Logic Lock (Standard) boundary is visible in the Chip Planner in the Consumer project for reference only. The Consumer cannot modify this region.

Related Information

[Step 1: Add Files to Customer Project](#) on page 34

4.6. Step 6: Device Programming and Hardware Verification

1. Program the device.

For instructions to program the device, refer to *Device Programming* in *AN 839: Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board*.

2. Verify the LEDs behavior.

After completing this tutorial module:

- LEDs D6-D3 map to the `blinking_led_top` core
- LEDs D10-D7 map to the top-level (root) design.

When you create and load the `.sof`, the `blinking_led_top` core does not illuminate any LEDs. The top-level design shows a shifting bit in green. The behavior of the periphery LED driver carries into the Consumer project via the final `.qdb` file.



Related Information

Device Programming

In AN 839: Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board

4.7. Step 7: Generate a Signal Tap File for the Root Partition

1. Go to the shell from where you opened the Intel Quartus Prime software.
2. In the shell, go to directory `a10_pcie_devkit_design_block_reuse_stp/Root_Partition_Reuse/Developer`, and then run the following command:

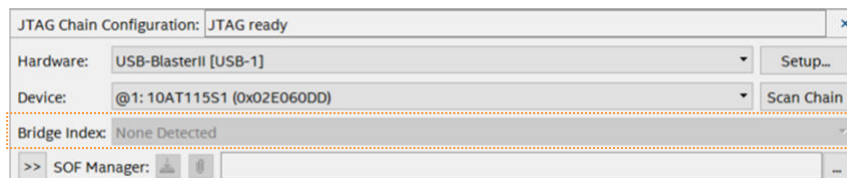
```
quartus_stp top --create_signaltap_hdl_file --stp_file \
    stp_root_partition.stp
```

4.8. Step 8: Hardware Verification with Signal Tap

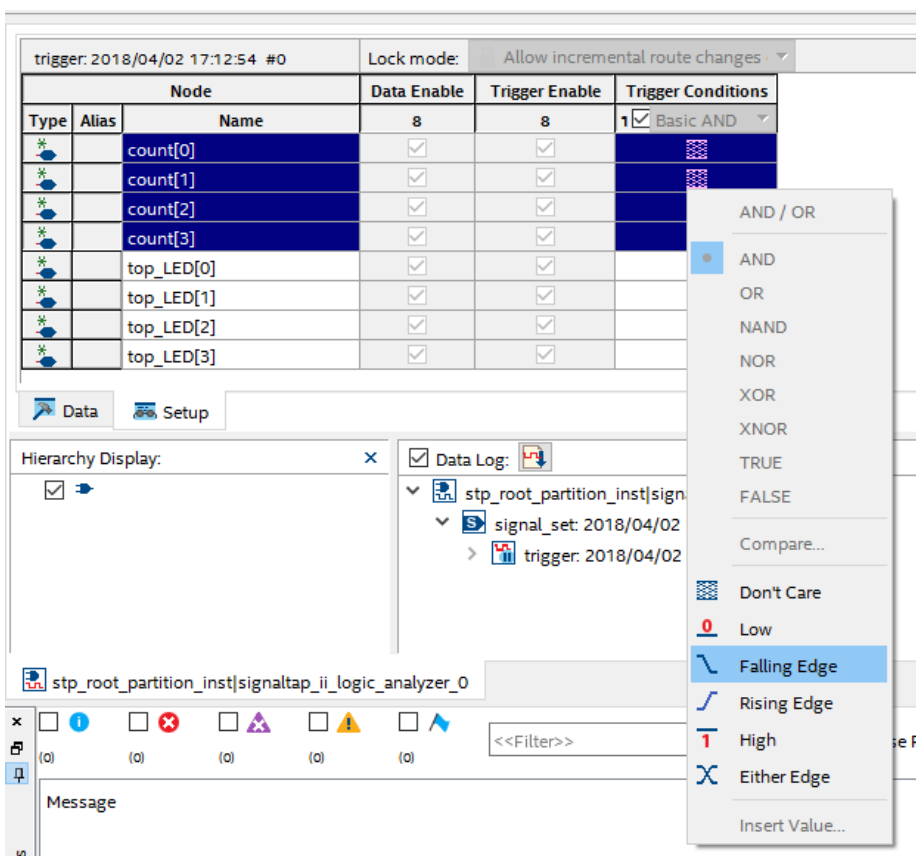
1. In the Signal Tap window, click **File ► Open**, and open the `stp_root_partition.stp` file, which you created in the previous step.
2. Make sure that the development kit is powered on and connected to the machine from which you open the Signal Tap Logic Analyzer.
3. Set up the JTAG Chain Configuration, and ensure **Instance Manager** is **Ready to acquire**.

For detailed information, refer to *Step 8: Hardware Verification with Signal Tap* in the *Core Partition Reuse Debug - Developer* tutorial module.

4. Verify that **Bridge Index** is set to **None Detected** in the **JTAG Chain Configuration** window.



5. To set the trigger condition, select the **count[0]**, **count[1]**, **count[2]**, and **count[3]** signals, right-click the column under **Trigger Conditions**, and select **Falling Edge**.




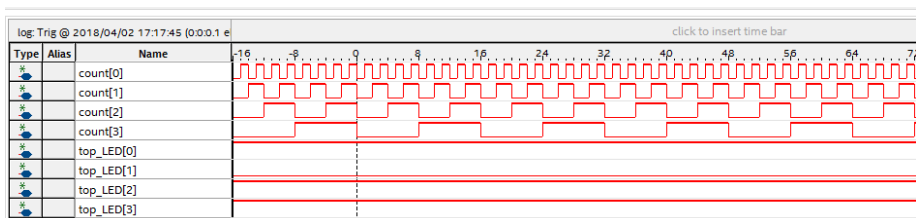
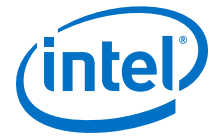
6. Run analysis by clicking the  icon, next to the **Instance Manager**. When the analysis finishes, the **Waveform** tab shows the captured data.
7. Verify the transition of the nodes in the root partition.

Figure 10. Waveforms for Root Partition Nodes in Developer Project



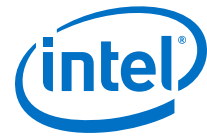
In this tutorial design, the count[3:0] signals represent the counter in the root partition, and the top_LED signals represent the green LEDs on the board, which also map to the top-level (root) design. After the trigger activates, only one of the top_LED bits is low, at any time.

If the root partition reuse succeeds, the Consumer project must present an identical behavior to the Developer project, since the Consumer imports the root partition .qdb file from this Developer project.



Related Information

Step 8: Hardware Verification with [Signal Tap](#) on page 18



5. Root Partition Reuse Debug—Consumer

Process Description

The root partition Consumer receives from the Developer the final top-level, placed, and routed root partition, and optionally a `.sdc` file. Then, the Consumer enables the PRC partition for debug by instantiating the SLD JTAG Bridge Host, which communicates with the SLD JTAG Bridge Agent instantiated in the root partition. Finally, the Consumer taps pre-synthesis nodes in the Signal Tap GUI to debug the PRC partition.

Completed Tutorial Files

The `Root_Partition_Reuse/Completed/Consumer/` tutorial directory contains the completed files for this tutorial module.

Tutorial Module Steps

This tutorial module includes the following steps:

- [Step 1: Add Files to Customer Project](#) on page 34
- [Step 2: Generate and Instantiate SLD JTAG Bridge Host in PRC Partition](#) on page 35
- [Step 3: Synthesize, Create Signal Tap File and Compile](#) on page 35
- [Step 4: Device Programming and Hardware Verification](#) on page 36
- [Step 5: Hardware Verification of PRC Partition with Signal Tap](#) on page 36
- [Step 6: Hardware Verification of Root Partition with Signal Tap](#) on page 39

5.1. Step 1: Add Files to Customer Project

You import the timing constraints and the root partition database from the Developer project.

1. In the Intel Quartus Prime Pro Edition software, click **File > Open Project**, and open the `a10_pcie_devkit_design_block_reuse_stp/Root_Partition_Reuse/Consumer/top.qpf` project file.
2. Click **Project > Add/Remove Files in Project**.
3. On the **Files** pane, click the browse (...) button near **File name** to locate and select the `top.sdc` file, and click **Add**.
4. Click **Apply**, and then click **OK**.
5. If the **Design Partitions Window** is not visible, click **Assignments > Design Partitions Window**.



6. In the **Design Partitions Window**, locate the **root partition** row, double-click the **Partition Database File** field, and then click browse (...).
7. Select the **root_partition.qdb** file copied over from the Developer project.
8. Click the partition name to confirm the .qdb assignment.

Related Information

Step 5: Compile, Export Root Partition and Copy Files to Consumer Project on page 29

5.2. Step 2: Generate and Instantiate SLD JTAG Bridge Host in PRC Partition


Exporting the root partition in the Developer project does not include logic inside the PRC or the SLD JTAG Bridge Host. The Consumer must add the SLD JTAG Bridge Host to the PRC in the Consumer project.

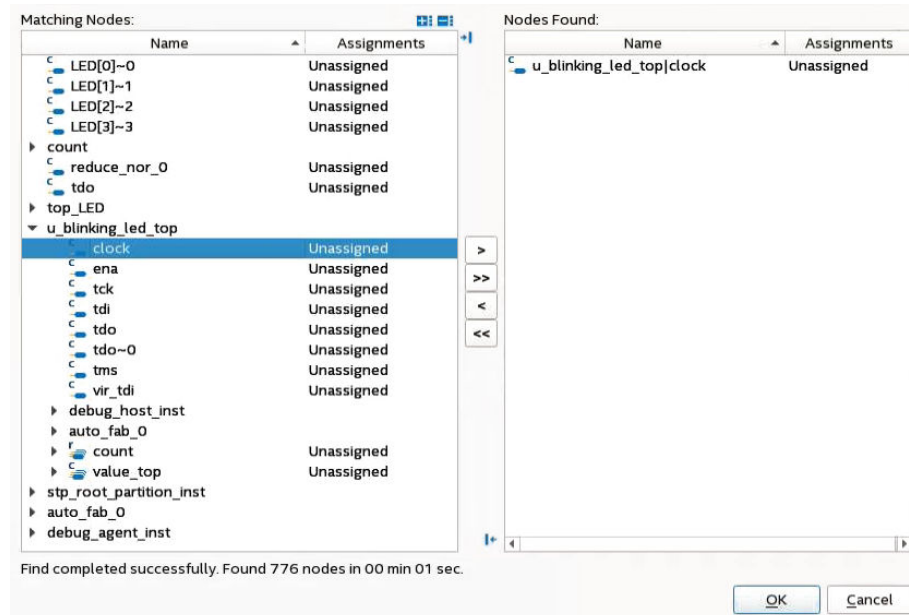
1. From the IP Catalog (**Tools > IP Catalog**), select and generate the **SLD JTAG Bridge Host Intel FPGA IP**. Set the name as `debug_host`.
2. Open the `blinking_led_top.sv` file, uncomment lines 5 to 10 and 27 to 34, and save the file.

Related Information

Step 3: Generate and Instantiate the SLD JTAG Bridge Host on page 28

5.3. Step 3: Synthesize, Create Signal Tap File and Compile

1. On the Compilation Dashboard, click **Analysis & Synthesis** to synthesize the design. When synthesis is complete, the Compilation Dashboard displays a check mark.
2. In the Intel Quartus Prime Pro Edition software, click **Tools >  Signal Tap Logic Analyzer**.
3. In the Instance Manager, click **auto_signaltap_0**.
4. In the **Setup** tab, double-click to open the Node Finder.
5. In the **Node Finder** window, type * in the **Named** section, set **Filter** to **Signal Tap: pre-synthesis**, and then click **Search**.
6. In the **Matching Nodes** list, expand the **u_blinking_led_top|count**.
7. Select **count[0]**, **count[1]**, **count[2]**, and **count[24]**. Insert the nodes by clicking **>**.
8. Select **value_top** under **u_blinking_led_top**. Click **>**, then click **Insert**, and then click **Close**.
9. In the Signal Tap window, under **Signal Configuration**, click (...) next to **Clock**.
10. In the Node Finder, search for *, and select the clock node in the periphery reuse core partition **u_blinking_led_top**. Click **>**, and then click **OK** to close.



11. Leave all the other options as default under **Signal Configuration**. Go to **File** ► **Save** and save the file as `stp_periphery_reuse_core.stp`.
A dialog box appears asking if you want to enable Signal Tap file for the project.
12. Click **Yes**, and close the file.
13. Click ► **Compile Design** on the Compilation Dashboard.
Check marks ✓ indicate finished Compiler modules.

5.4. Step 4: Device Programming and Hardware Verification

1. Program the device.
For instructions to program the device, refer to *Device Programming* in AN 839: *Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board*.
2. Verify the LEDs behavior.

After completing this tutorial module:

- LEDs D6-D3 map to the `blinking_led_top` core
- LEDs D10-D7 map to the top-level (root) design

After configuring the FPGA, the `blinking_led_top` core flashes red LEDs in a binary counting order. The top-level design shows a single bit shifting in green.

Related Information

Device Programming

In AN 839: *Design Block Reuse Tutorial for Intel Arria 10 FPGA Development Board*

5.5. Step 5: Hardware Verification of PRC Partition with Signal Tap

To use Signal Tap GUI for the PRC:



1. Determine the bridge index according to the number in the synthesis report file (Root_Partition_Reuse/Developer/output_files/top.syn.rpt), under *JTAG Bridge Agent Instance Information* in the Developer project.

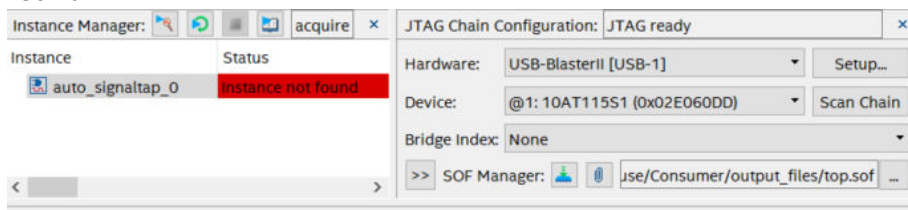
Partition Name	Associated Host	JTAG Bridge Agent Hierarchy Name	Assigned Instance Index
root_partition		debug_agent_inst sld_jtag_bridge_agent_0 sld_jtag_bridge_agent_inst	0

2. In the Signal Tap window, click **File ► Open**, and open `stp_periphery_reuse_core.stp`.
3. Make sure that the development kit is powered on and connected to the machine from which you open the Signal Tap Logic Analyzer.
4. Set up the JTAG Chain Configuration, and ensure **Instance Manager** is **Ready to acquire**.

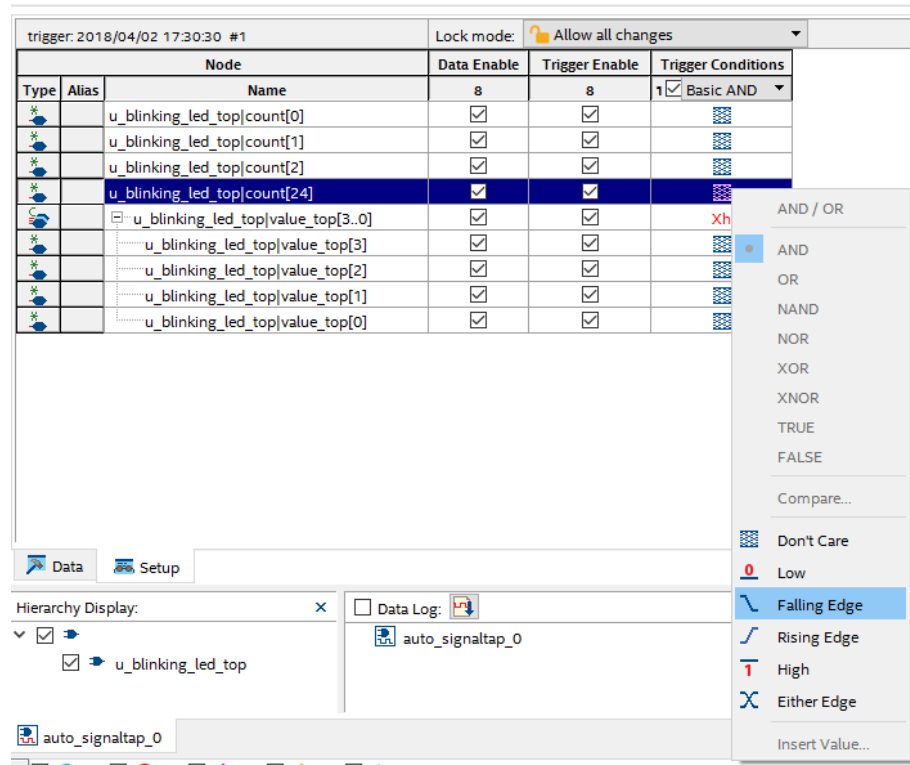
For detailed information, refer to *Step 8: Hardware Verification with Signal Tap* in the *Core Partition Reuse Debug - Developer* tutorial module.

5. Set the **Bridge Index** as found in the synthesis report (Root_Partition_Reuse/Developer/output_files/top.syn.rpt in the Developer Project

If the values for **Bridge Index** are different, Signal Tap reports **Instance not found**.

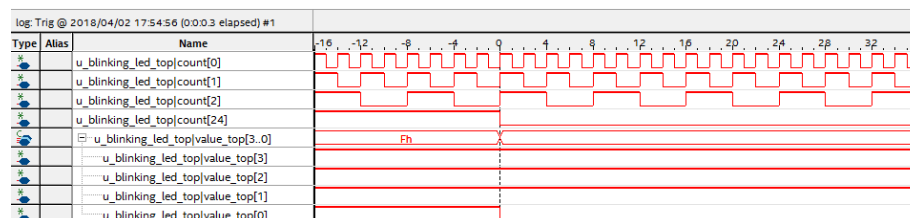


6. To set the trigger condition, select **count[24]**, right click the column under **Trigger Conditions** and select **Falling Edge**.



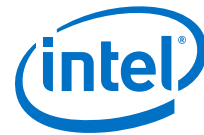
7. Run analysis by clicking the icon, next to the **Instance Manager**. When the analysis finishes, the **Waveform** tab shows the captured data.
8. Verify the transition of PRC nodes in Signal Tap GUI. The expected behavior is:
 - value_top[0] transitions along with count[24].
 - count[0], count[1], and count[2] show the transition of other counter bits in the PRC partition during this process.

Figure 11. Waveforms for PRC Partition Nodes in Consumer Project



Related Information

Step 8: Hardware Verification with Signal Tap on page 18



5.6. Step 6: Hardware Verification of Root Partition with Signal Tap

1. Go to the shell from where you opened the Intel Quartus Prime software.
2. In the shell, go to directory `al0_pcie_devkit_design_block_reuse_stp/Root_Partition_Reuse/Consumer`, and then run the following command:


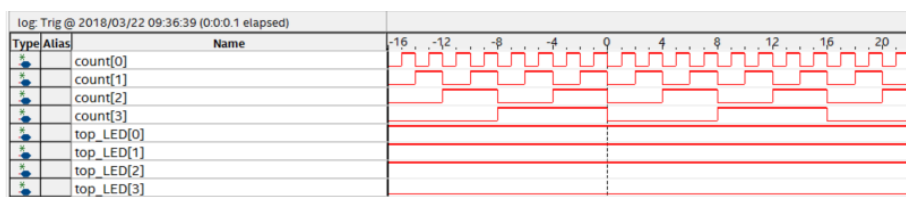

```
quartus_stp top --create_signaltap_hdl_file --stp_file \
    stp_root_partition.stp
```
3. In the Signal Tap window, click **File ► Open**, and open the `stp_root_partition.stp` file, which you created in the previous step.
4. Make sure that the development kit is powered on and connected to the machine from which you open the Signal Tap Logic Analyzer.
5. Verify that **Bridge Index** is set to **None** in the **JTAG Chain Configuration** window.
6. To set the trigger condition, select the **count[0]**, **count[1]**, **count[2]**, and **count[3]** signals, right-click the column under **Trigger Conditions**, and select **Falling Edge**.
7. Run analysis by clicking the  icon, next to the **Instance Manager**. When the analysis finishes, the **Waveform** tab shows the captured data.
8. Verify the transition of the nodes in the root partition.

Figure 12. Waveforms for Root Partition Nodes in Consumer Project



In this tutorial design, the `count[3:0]` signals represent the counter in the root partition, and the `top_LED` signals represent the green LEDs on the board, which also map to the top-level (root) design. After the trigger activates, only one of the `top_LED` bits is low, at any time.

If the implementation succeeds, the Consumer project behaves identically to the Developer project.



6. Document Revision History for AN 847: Signal Tap Tutorial with Design Block Reuse for Intel Arria 10 FPGA Development Board

Document Version	Intel Quartus Prime Version	Changes
2018.05.07	18.0.0	Initial release.

Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

*Other names and brands may be claimed as the property of others.

ISO
9001:2008
Registered