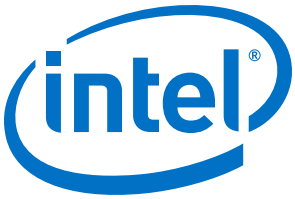


AN 845: Signal Tap Tutorial for Intel® Arria® 10 Partial Reconfiguration Design

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

AN-845 | 2018.10.08
Latest document on the web: **PDF** | **HTML**

Contents

Tutorial Overview

This document demonstrates how to debug an Intel® Arria® 10 Partial Reconfiguration design with the Signal Tap Logic Analyzer. This application note extends the Partial Reconfiguration (PR) work presented on AN 797: Partially Reconfiguring a Design on Intel Arria 10 GX FPGA Development Board to a verification environment.

The Signal Tap Logic Analyzer captures and displays real-time signal behavior in an FPGA design, allowing to examine the behavior of internal signals during normal device operation without the need for extra I/O pins or external lab equipment.

Partial Reconfiguration is an advanced design flow that allows you to reconfigure a portion of the FPGA dynamically, while the remaining FPGA design continues to function. You can define multiple personas to occupy the same design region, without impacting operation in other regions.

The PR support in the Signal Tap Logic Analyzer includes data acquisition in static and PR regions. Moreover, you can debug multiple personas present in a PR region and multiple PR regions.

Related Information

- Design Debugging with the Signal Tap Logic Analyzer
  In *Intel Quartus® Prime Pro Edition User Guide: Debug Tools*
- Creating a Partial Reconfiguration Design
  In *Intel Quartus® Prime Pro Edition User Guide: Partial Reconfiguration*
- AN 797: Partially Reconfiguring a Design on Intel Arria 10 GX FPGA Development Board

PR Debug Considerations

Debugging a PR design requires planning. Before compiling, you must decide whether you want to tap signals in the static region, which PR region you want to debug, and which personas in the PR region you want to debug.

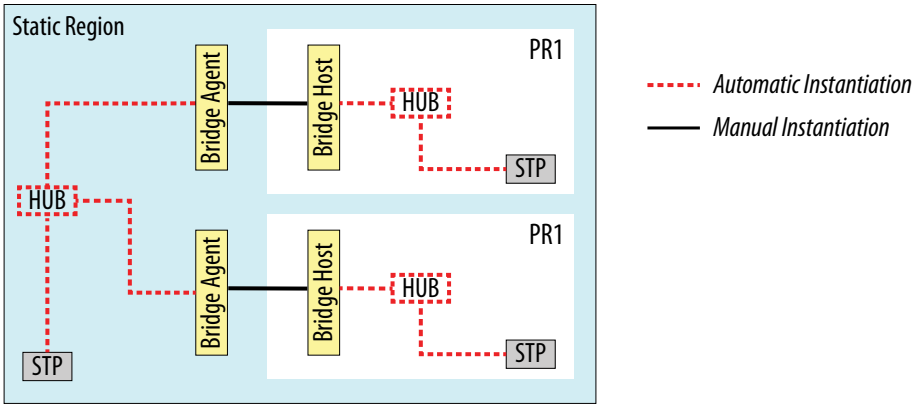
To ensure visibility, the debugging fabric must extend to all the regions that you want to tap. The Intel Quartus® Prime software performs this extension with debug bridge components: the SLD JTAG Bridge Agent Intel FPGA IP and the SLD JTAG Bridge Host Intel FPGA IP.

To incorporate these components to the design, for each PR region in the design that you want to debug:

1. Instantiate the SLD JTAG Bridge Agent in the static region.
2. Instantiate the SLD JTAG Bridge Host in the PR region of the default persona.
3. Instantiate the SLD JTAG Bridge Host on the implementation revisions that you want to debug.

Figure 1. Debug Fabric in PR Design with Signal Tap

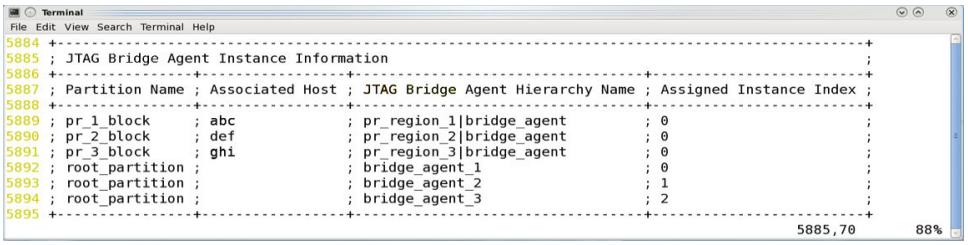
The figure shows in solid outline the entities that you instantiate manually, and in dashed outline the entities that the Compiler instantiates.



SLD JTAG Bridge Index

The index is an attribute of the SLD JTAG Bridge Agent that uniquely identifies bridge agents present in the design. You can find information regarding the bridge index in the synthesis report (`<base revision>.syn.rpt`), by looking under **JTAG Bridge Agent Instance Information**. The bridge index for the root partition is always **None**.

Figure 2. JTAG Bridge Agent Instance Information in Synthesis Report



Related Information

Debugging Partial Reconfiguration Designs Using Signal Tap Logic Analyzer
In *Intel Quartus Prime Pro Edition User Guide: Debug Tools*

Tutorial Software and Hardware Requirements

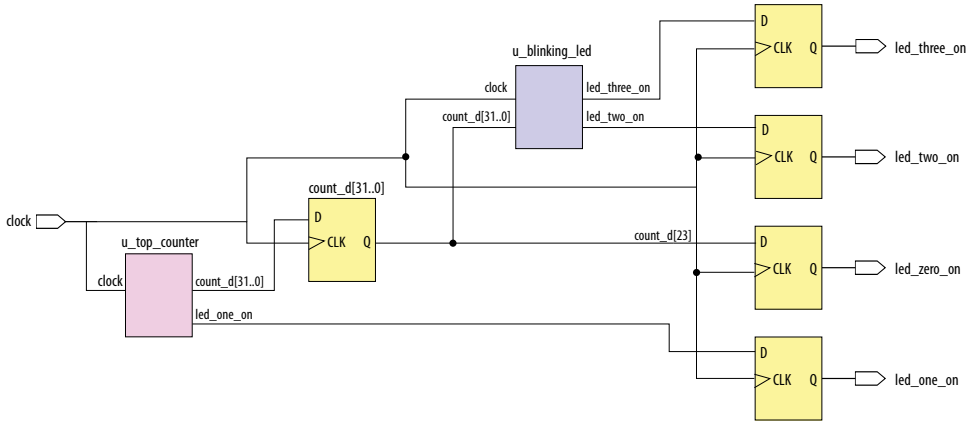
To perform this tutorial, you need the following software and hardware:

- The Intel Quartus Prime Pro Edition software version 18.1 or later. The software includes the Signal Tap Logic Analyzer and the Programmer.
- Intel Arria 10 GX FPGA development kit, or a design board with JTAG connection to the device under test.
- Intel FPGA Download Cable, for communication between the device and the Intel Quartus Prime software.

Tutorial Design Description

The design for this tutorial consists of one 32-bit counter. At the board level, the design connects the clock to a 50MHz source, and connects the output to four LEDs on the FPGA. Selecting the output from the counter bits in a specific sequence causes the LEDs to blink at a specific frequency.

Figure 3. Flat Reference Design without PR Partitioning



Downloading the Tutorial Design

The partial reconfiguration tutorial files are available in:

https://github.com/intel/fpga-partial-reconfig

To download the tutorial:

1. In the web page, click **Clone or download**, and then click **Download ZIP**.
2. Unzip the `fpga-partial-reconfig-master.zip` file.
3. Navigate to the `tutorials/a10_pcie_devkit_blinking_led_stp` sub-folder to access the design.

Tutorial Design Files

The design folder contains two subfolders: The `start` folder contains the files that you need to follow this tutorial, and the `finish` folder contains the complete set of files you create using this application note. Reference these files at any point during the walkthrough.

Table 1. Description of Tutorial Design Files in start Folder

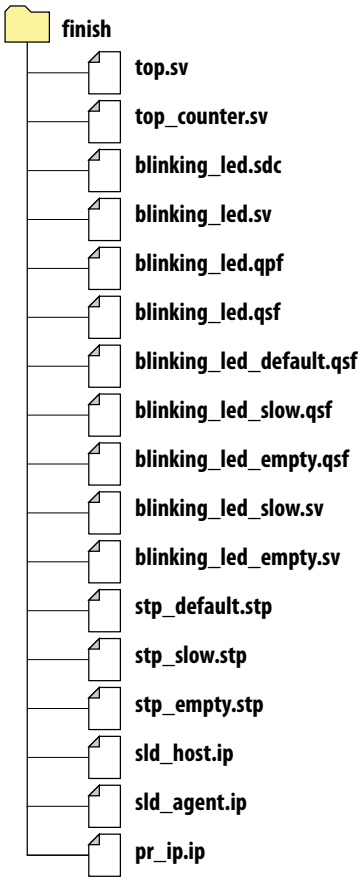
File Name	Description
`top.sv`	Top-level file. Contains the flat implementation of the design. This module instantiates the `blinking_led` sub-partition and the `top_counter` module.

continued...

File Name	Description
	This module also instantiates the SLD JTAG Bridge Agent for debugging purposes.
`top_counter.sv`	Top-level 32-bit counter that controls `LED[1]` directly. The registered output of the counter controls `LED[0]`, and powers `LED[2]` and `LED[3]` via the `blinking_led` module.
`blinking_led.sdc`	Defines the timing constraints for the project.
`blinking_led.sv`	This module acts as the PR partition. The module receives the registered output of `top_counter` module, which controls `LED[2]` and `LED[3]`.This module also instantiates the SLD JTAG Bridge Agent for debugging the default persona.
`blinking_led.qpf`	Intel Quartus Prime project file that contains a list of all the revisions in the project.
`blinking_led.qsf`	Intel Quartus Prime settings file that contains assignments and settings for the base revision of the project.
`blinking_led_default.qsf`	Contains assignments and settings for the `blinking_led_default` implementation revision of the project.
`blinking_led_slow.qsf`	Contains assignments and settings for the `blinking_led_slow` implementation revision of the project.
`blinking_led_empty.qsf`	Contains assignments and settings for the blinking led empty of the project.
`blinking_led_slow.sv`	Slower version of the PR logic. On this version, the led blinks at a slower rate than the default PR persona. The module receives the registered output of `top_counter` module, which controls `LED[2]`and `LED[3]`. This module also instantiates the SLD JTAG Bridge Agent for debugging the default persona.
`blinking_led_empty.sv`	Empty version of the PR logic. This module holds the outputs at a constant. The module receives the registered output of `top_counter` module, which controls `LED[2]` and `LED[3]`. This module also instantiates the SLD JTAG Bridge Agent for debugging the default persona.
`pr_ip.ip`	Intel Arria 10 Partial Reconfiguration Controller IP. This Intel FPGA IP enables PR over a JTAG connection.

The following Figure shows the list of files in the `finish` folder:

Figure 4. Tutorial Design Files in finish Folder



Tutorial Walkthrough

This tutorial describes preparing the blinking_led design for debug with the Signal Tap Logic Analyzer.

Note: This Application Note only covers adding Signal Tap debugging capabilities to a PR design. For information about turning a non-PR design to PR, refer to *AN 797: Partially Reconfiguring a Design on Intel Arria 10 GX FPGA Development Board*.

Process Description

To tap signals in a PR design, you extend the debug fabric to the PR regions when creating the base revision, and then define debug components for the implementation revisions.

Tutorial Steps

This tutorial includes the following steps:

Related Information

AN 797: Partially Reconfiguring a Design on Intel Arria 10 GX FPGA Development Board

Step 1: Getting Started

To copy the reference design files to your working environment and compile the initial design for this tutorial:

1. Before you begin, download the tutorial files.
2. In your working environment, create a directory named `a10_pcie_devkit_blinking_led_stp`.
3. Copy the downloaded `tutorials/a10_pcie_devkit_blinking_led_stp/start` sub-folder to your working directory.
4. In the Intel Quartus Prime Pro Edition software, click **File ➤ Open Project** and select `blinking_led.qpf`.
5. Click **Processing ➤ Start ➤ Start Analysis and Synthesis**.

Related Information

Downloading the Tutorial Design on page 5

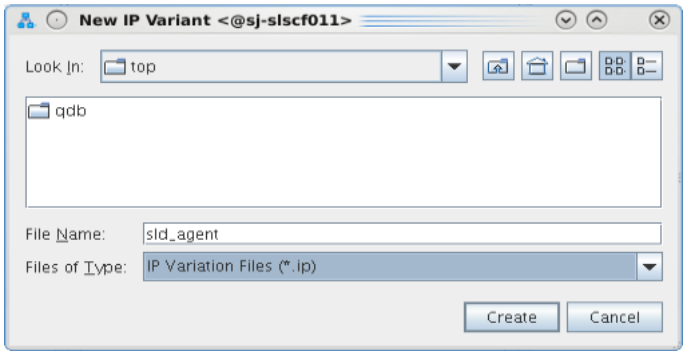
Step 2: Preparing the Base Revision

This step extends the debug fabric to the PR regions that you want to debug. To accomplish this goal, you must instantiate the SLD JTAG Bridge Agent in the static region and the SLD JTAG Bridge Host in the default persona of the PR region.

Preparing the Static Region

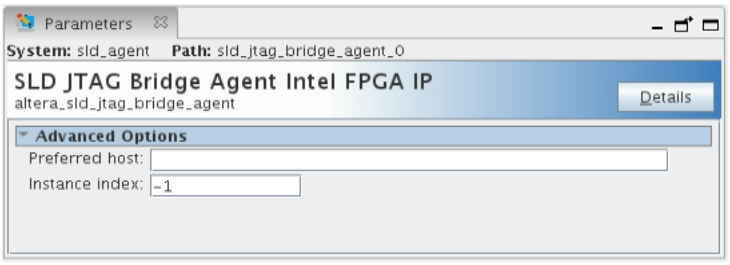
1. In the IP Catalog (**Tools ➤ IP Catalog**), type `SLD JTAG Bridge Agent`, and double-click the **SLD JTAG Bridge Agent Intel FPGA IP**.
2. In the **Create IP Variant** dialog box, type `sld_agent` as the file name, and then click **Create**.

Figure 5. Create IP Variant Dialog Box



3. In the parameter editor, use the default parameterization for `sld_agent`. Click **Generate HDL...**, and then click **Generate**.

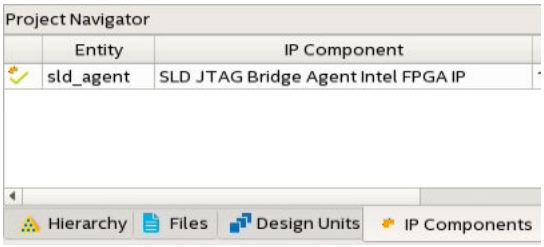
Figure 6. SLD JTAG Bridge Agent Intel FPGA IP Parameters



The parameter editor generates the `sld_agent.ip` IP variation file and adds the file to the `blinking_led` project.

4. Close the parameter editor.
5. Verify whether the `sld_agent` IP variant appears in the **IP Components** tab of the Project Navigator.

Figure 7. sld_agent IP Variant in Project Navigator



If the IP variant does not appear in the Project Navigator, click **Project ➤ Add/ Remove Files in Project**, find the `sld_agent.ip` file, and add to the project.

6. In the `top.sv` file, instantiate the `sld_agent` IP in the base revision by uncommenting the following lines:

```
//===============
//Enable Signal Tap
wire tck;
wire tms;
wire tdi;
wire vir_tdi;
wire ena;
wire tdo;
sld_agent u_sld_agent (
    .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
);
//===============
```

Related Information

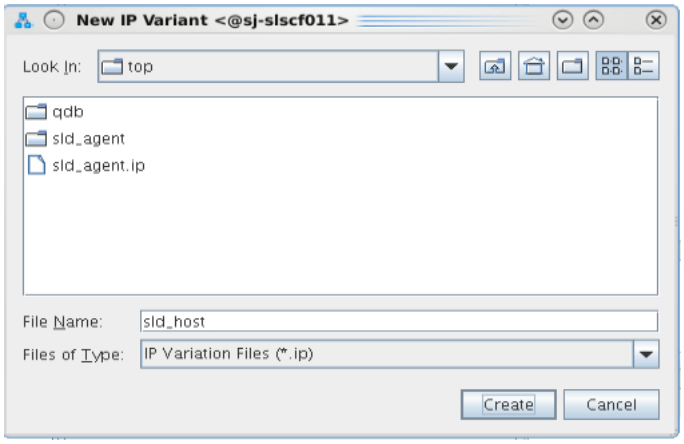
Instantiating the SLD JTAG Bridge Agent
In *Intel Quartus Prime Pro Edition User Guide: Debug Tools*

Preparing the Default PR Persona

In this phase you instantiate the SLD JTAG Bridge Host in the PR region that you want to debug.

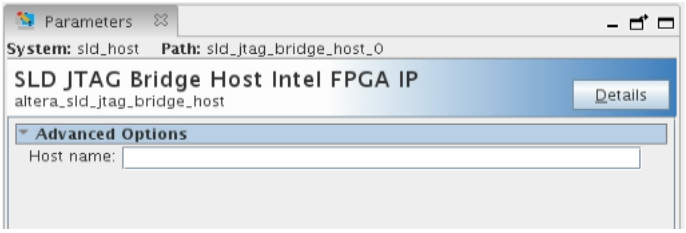
1. In the IP Catalog (**Tools ➤ IP Catalog**), type `SLD JTAG Bridge Host`, and double-click the **SLD JTAG Bridge Host Intel FPGA IP**.
2. In the **Create IP Variant** dialog box, type `sld_host` as the file name, and then click **Create**.

Figure 8. Create IP Variant Dialog Box



3. In the parameter editor, leave the default parameterization for `sld_host`. Click **Generate HDL...**, and then click **Generate**.

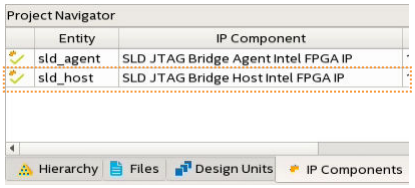
Figure 9. SLD JTAG Bridge Host Intel FPGA IP Parameters



The parameter editor generates the `sld_host.ip` IP variation file and adds the file to the `blinking_led` project.

4. Close the parameter editor.
5. Verify whether the `sld_host` IP variant appears in the **IP Components** tab of the Project Navigator.

Figure 10. sld_host IP Variant in Project Navigator



If the IP variant does not appear in the Project Navigator, click **Project ➤ Add/ Remove Files in Project**, find the `sld_host.ip` file, and add it to the project.

6. Instantiate the `sld_host` IP in the default persona by uncommenting the following blocks of code from `blinking_led.sv`:

```
//===============
//Uncomment to enable Signal Tap
wire tck;
wire tms;
wire tdi;
wire vir_tdi;
wire ena;
wire tdo;
sld_host u_sld_hostled_two_on (
   .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
);
//===============
```

7. Change the instantiation of the persona in `top.sv` to include the `sld_host` ports.

```
blinking_led u_blinking_led (
    .clock         (clock),
    .counter       (count_d),
    //===================
    //Uncomment this block to enable Signal Tap
    .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
    //====================
    .led_two_on    (pr_led_two_on),
    .led_three_on  (pr_led_three_on)
);
```

8. Update the port definition of the default PR persona to include the following ports by uncommenting this block of code in the `blinking_led.sv` file:

```
module blinking_led (
    // clock
    input wire clock,
    input wire [31:0] counter,
    //=================
    //Uncomment this block to enable Signal Tap
    input wire tck,
    input wire tms,
    input wire tdi,
    input wire vir_tdi,
    input wire ena,
    output wire tdo,
    //=================
    // Control signals for the LEDs
    output wire led_two_on,
    output wire led_three_on
);
```

Related Information

Instantiating the SLD JTAG Bridge Host
In *Intel Quartus Prime Pro Edition User Guide: Debug Tools*

Step 3: Preparing the Implementation Revisions for Debug

In this step you instantiate the SLD JTAG Bridge Host and then add a `.stp` file to the implementation revisions that you want to debug.

1. In the Intel Quartus Prime GUI, set `blinking_led_slow` as the current revision.
2. Include `sld_host.ip` as a project file in the `blinking_led_slow` implementation revision.
3. Uncomment the following blocks of code from `blinking_led_slow.sv`:

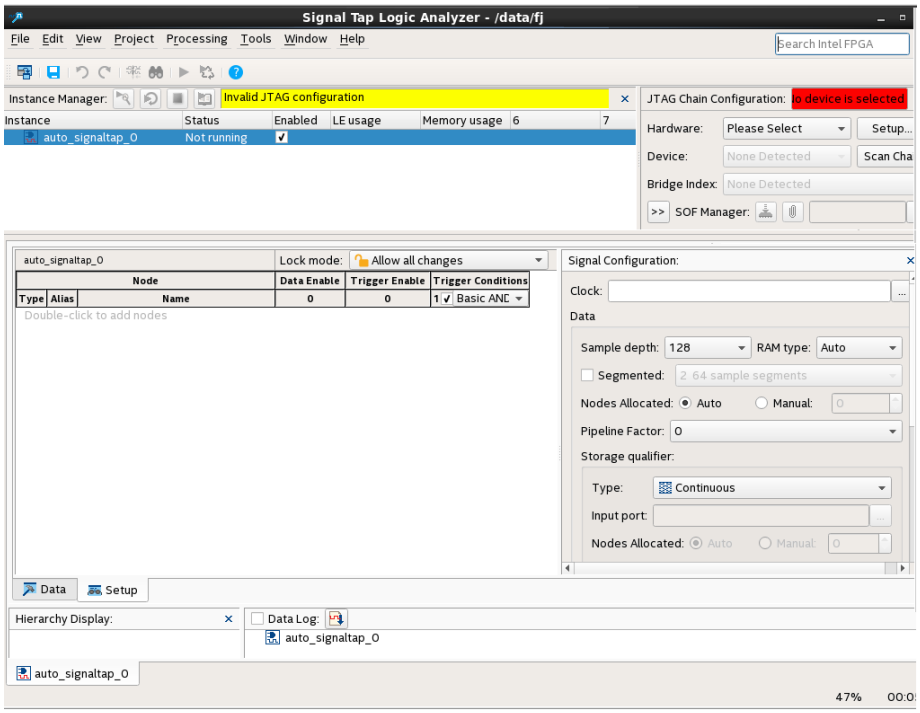
```
//==================
// Uncomment this block to enable Signal Tap
input wire tck,
input wire tms,
input wire tdi,
input wire vir_tdi,
input wire ena,
output wire tdo,
//==================
//==================
// Uncomment this block to enable Signal Tap
sld_host u_sld_hostled_two_on (
    .tck    (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
);
//==================
```

4. Update the port definition for the PR personas to include the following ports, by uncommenting this block of code in `blinking_led_slow.sv` and `blinking_led_empty.sv` files:

```
module blinking_led_slow (
    // clock
    input wire clock,
    input wire [31:0] counter,
    //=================
    //Uncomment this block to enable Signal Tap
    input wire tck,
    input wire tms,
    input wire tdi,
    input wire vir_tdi,
    input wire ena,
    output wire tdo,
    //=================
    // Control signals for the LEDs
    output wire led_two_on,
    output wire led_three_on
);
```

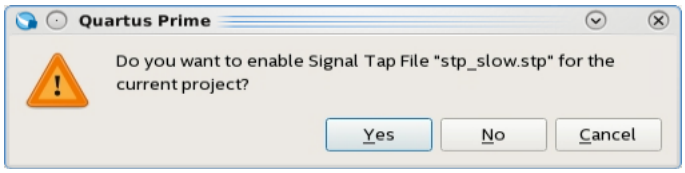
5. Click **Tools ➤ Signal Tap Logic Analyzer** to open the Signal Tap Logic Analyzer Window.

Figure 11. Signal Tap Logic Analyzer Window



6. Click **File ➤ Save As**, and save the file as `stp_slow.stp`.
   A dialog box appears prompting you to enable Signal Tap file `stp_slow.stp` for the current project.

Figure 12. Enable stp_slow.stp for the Current Project



7. Click **Yes**.

Repeat these steps for the `blinking_led_default` and the `blinking_led_empty` personas. Use `stp_default.stp` and `stp_empty.stp` for the Signal Tap files.

You can disable Signal Tap in the project by clicking **Assignments ➤ Settings**. In the **Category** pane select Signal Tap Logic Analyzer. Then, turn off **Enable Signal Tap Logic Analyzer**.

Related Information

Tutorial Design Description on page 5

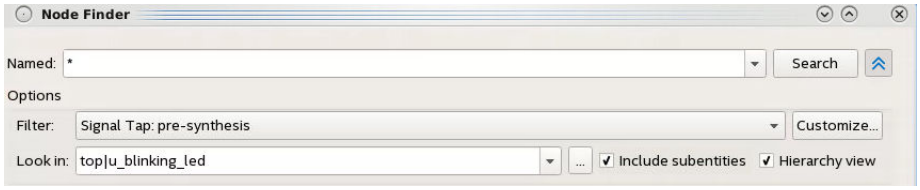
Step 4: Tapping Signals in the Implementation Persona

To add signals from the implementation persona to the Signal Tap logic analyzer:

1. Set blinking_led_slow as the current revision in the Intel Quartus Prime GUI.
2. Open the `stp_slow.stp` file.
3. Double-click the **Setup** tab to open the Node Finder.
4. Set the following search fields, and then click **Search**

Field	Value
Named	*
Filter	Signal Tap: pre-synthesis
Look in	`top|u_blinking_led`

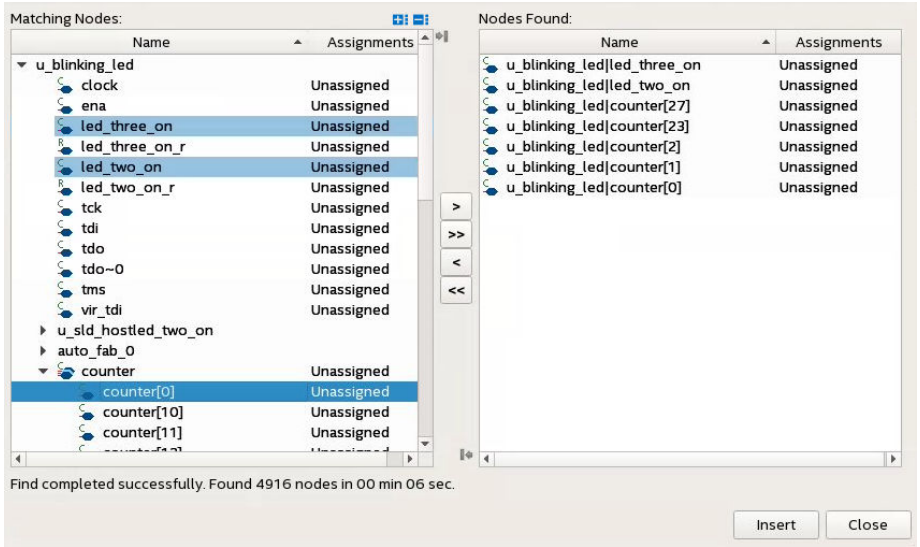
Figure 13. Search Parameters to Find Signals



This action displays all the nodes that you can probe in this revision.

5. From the **Matching Nodes** list, select `led_three_on`, `led_two_on`, and `counter[2:0]`, `counter[27]`, and `counter[23]`, and then click **>**. This action adds the signals to the **Nodes Found** list.

Figure 14. Signals in Nodes Found List



6. Click **Insert**.

The signals now appear in the **Instance Manager** pane of the Signal Tap GUI.

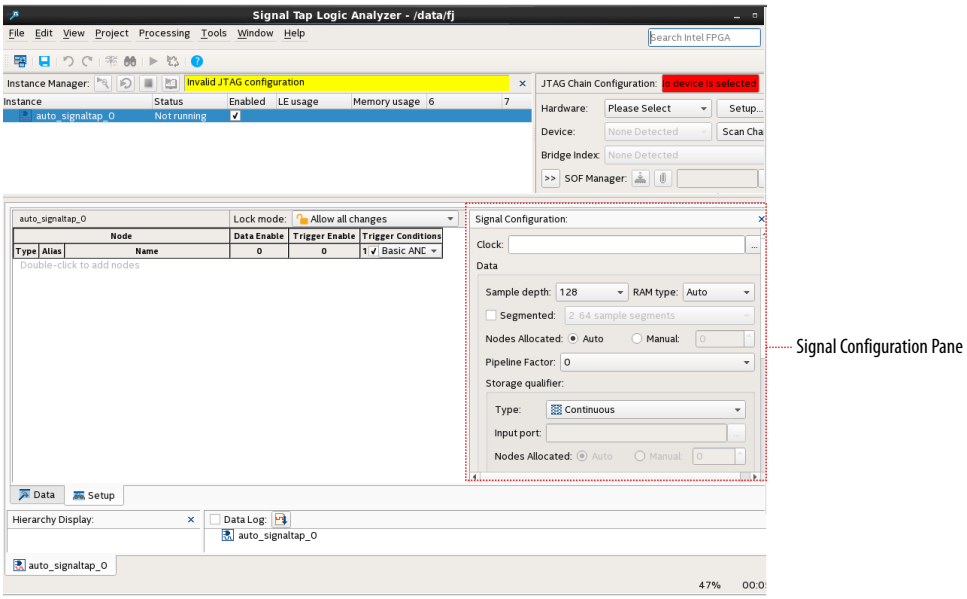
Related Information

Adding Signals to the Signal Tap File
In *Intel Quartus Prime Pro Edition User Guide: Debug Tools*

Step 5: Configuring Data Acquisition

You specify the acquisition parameters in the **Signal Configuration** pane of the Signal Tap Logic Analyzer.

Figure 15. Signal Configuration Pane



Add Acquisition Clock

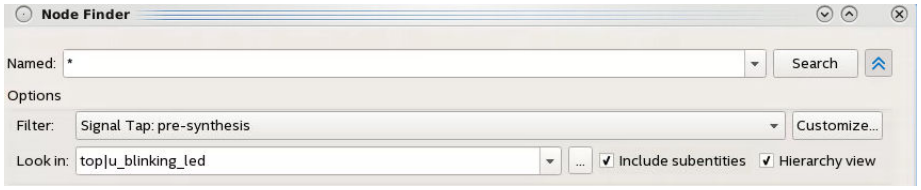
Specify the reference clock that Signal Tap uses during acquisition.

Perform the following steps in the **Signal Configuration** pane:

1. Next to **Clock**, click **...** to open the **Node Finder**.
2. Set the following search parameters:

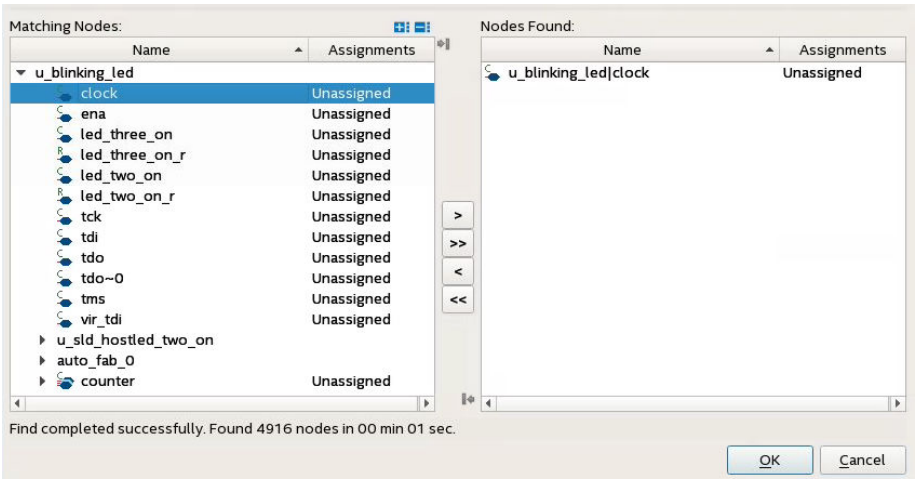
Field	Value
Named	*
Look in	`top|u_blinking_led`

Figure 16. Search Parameters to Find the Clock



3. Click **Search**.

Figure 17. Select Clock in Node Finder



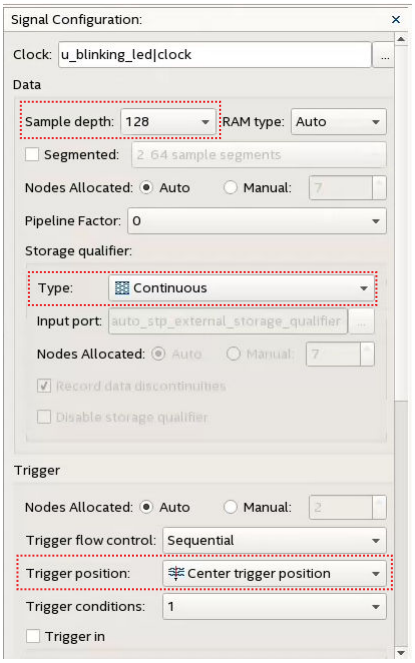
4. Select `clock`, click **>**, and then click **OK**.

Add Storage Parameters

Storage parameters define the number of samples the Signal Tap Logic Analyzer captures and stores, how to organize this samples, and the location of the sample with respect to the trigger activation.

1. In **Sample Depth**, select 128.
2. In **Storage Qualifier**, set **Type** as **Continuous**.
3. In **Trigger Position**, select **Center Trigger Position**.

Figure 18. Acquisition Settings for Tutorial

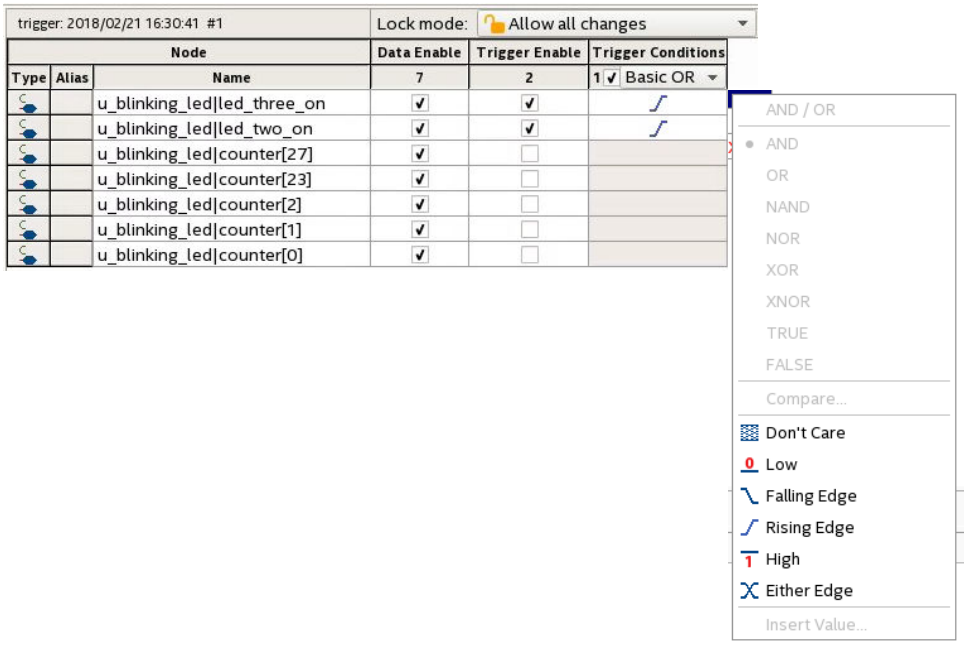


Step 6: Setting Trigger Conditions

These steps direct the Signal Tap Logic Analyzer to record data only after `u_blinking_led|led_three_on` or `u_blinking_led|led_two_on` does a rising edge transition.

1. In the **Setup** tab of the Signal Tap Logic Analyzer window, turn on the box under the **Trigger Condition** column
2. Open the drop-down menu and select **Basic OR**.
3. For **u_blinking_led|led_three_on** and **u_blinking_led|led_two_on**, turn on **Trigger Enable** and select **Rising Edge** as the trigger type.
4. For all the other signals, turn off **Trigger Enable**.

Figure 19. Trigger Conditions



Defining trigger conditions completes the Signal Tap instance configuration. Now you can compile the design.

Step 7: Generating Programming Files

The design is now ready for compilation. The Intel Quartus Prime Compiler generates files that you then program into the FPGA. This Partial Reconfiguration design requires generating `.sof` and `.rbf` files.

1. Ensure the `blinking_led.qsf` contains the following assignments:

```
set_global_assignment -name GENERATE_PR_RBF_FILE ON
set_global_assignment -name ON_CHIP_BITSTREAM_DECOMPRESSION OFF
```

These assignments allow the assembler to automatically generate the required PR bitstreams.

2. In the Intel Quartus Prime GUI, click **Processing ➤ Start Compilation** to compile the base revision.
   Alternatively, type the following command:

```
quartus_sh --flow compile blinking_led -c blinking_led
```

3. Compile the implementation revision:

```
quartus_sh --flow compile blinking_led –c blinking_led_slow
quartus_sh --flow compile blinking_led –c blinking_led_default
quartus_sh --flow compile blinking_led –c blinking_led_empty
```

If the compilation succeeds, the output files are now in the `output_files` directory.

Related Information

- SRAM Object File (.sof) Definition
  In *Intel Quartus Prime Help*
- Raw Binary File (.rbf) Definition
  In *Intel Quartus Prime Help*
- Step 8: Preparing PR Implementation Revisions
  In *AN 797: Partially Reconfiguring a Design on Intel Arria 10 GX FPGA Development Board*

Step 8: Programming the Board

Before you begin:

1. Connect the power supply to the Intel Arria 10 GX FPGA development board.
2. Connect the USB Blaster cable between your PC USB port and the USB Blaster port on the development board.

Note: This tutorial utilizes the Intel Arria 10 GX FPGA development board on the bench, outside of the PCIe* slot in your host machine.

To program the design on the board:

1. In the Intel Quartus Prime software, click **Tools ➤ Programmer**.
2. In the Programmer, click **Hardware Setup** and select **USB-Blaster**.
3. Click **Auto Detect** and select the device, **10AT115S2F45**.
4. Click **OK**. The Intel Quartus Prime software detects and updates the Programmer with the three FPGA chips on the board.
5. Select the 10AT115S2F45 device, click **Change File** and load the `blinking_led.sof` file.
6. Enable **Program/Configure** for `blinking_led.sof` file.
7. Click **Start** and wait for the progress bar to reach 100%.
8. To program the PR persona that you want to debug, right-click the `blinking_led.sof` file in the Programmer, and click **Add PR Programming File**.
9. Select the `blinking_led_slow.pr_partition.rbf` file.

10. Disable **Program/Configure** for `blinking_led.sof` file.
11. Enable **Program/Configure** for `blinking_led_slow.pr_partition.rbf` file and click **Start**.
12. On the board, verify that two of the LEDs are blinking slower than the other two.

Related Information

Troubleshooting PR Programming Errors
In *Intel Quartus Prime Pro Edition User Guide: Partial Reconfiguration*

Step 9: Performing Data Acquisition

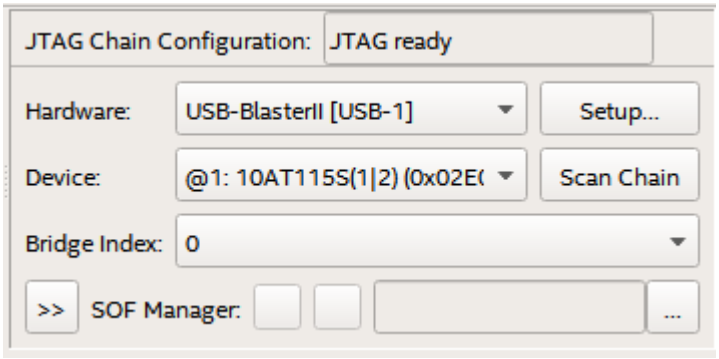
After loading the appropriate `.rbf` onto the board, you can start data acquisition on the Signal Tap logic analyzer.

To perform data acquisition:

1. Make sure that the Signal Tap Logic Analyzer loads the `.stp` file in the current active revision.
2. In the top right corner of the Signal Tap window, set up the JTAG connection to the board with the following options:

Option	Description
Hardware	**USB-BlasterII**
Device	**10AT115S**
Bridge Index	**0**

Figure 20. JTAG Configuration



3. On the Signal Tap toolbar, click **Run Analysis**.

   The analysis may take a few minutes.

   When the analysis finishes, the Signal Tap Logic Analyzer loads the waveforms to the window.

The following section displays the resultant waveforms for all PR configurations.

Tutorial Results

By looking at the data acquisition waveform, you can verify whether the signals behave consistently with your expectations. As a reference, the waveforms include counter[2:0] signals.

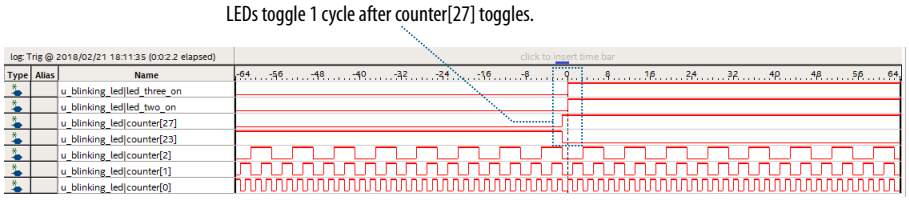
Related Information

Tutorial Design Description on page 5

Waveforms for Slow Implementation

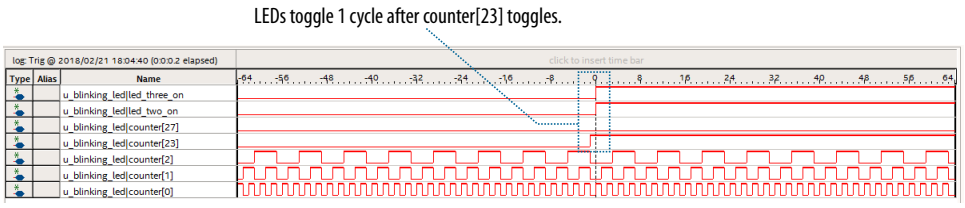
In the Figure, signals `led_three_on` and `led_two_on` show a rising edge one clock cycle after `counter[27]` has a rising edge.

Figure 21. Slow Implementation



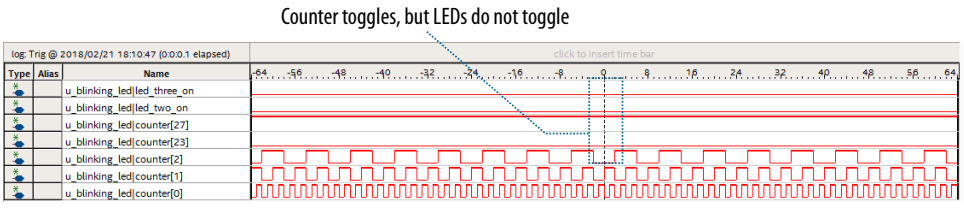
Waveforms for Default Implementation

Figure 22. Default Implementation



Waveforms for Empty Implementation

Figure 23. Empty Implementation



Document Revision History for AN 845: Signal Tap Tutorial for Intel Arria 10 Partial Reconfiguration Design

This document has the following revision history:

Table 2. Document Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.10.08	18.1.0	• Updated contents with the new PR flow, which eliminates the need for manual export of finalized snapshot of the static region. • Updated the list of files in the `start` and `finish` folders.
2018.05.07	18.0.0	Initial release.