13. Design Debugging Using the SignalTap II Embedded Logic Analyzer

Introduction

The phenomenal growth in design size and complexity continues to make design verification a critical bottleneck for current FPGA systems. Limited access to internal signals, complex FPGA packages, and PCB electrical noise all contribute to making design debugging the most challenging process of the design cycle. More than 50% of the design cycle time can be spent on debugging and verifying the design. To help with the process of design debugging, Altera® provides a solution that enables a designer to examine the behavior of internal signals, without using extra I/O pins, while the design is running at full speed on an FPGA device.

The SignalTap® II Embedded Logic Analyzer is scalable, easy to use, and is included with the Quartus® II software subscription. This logic analyzer helps debug an FPGA design by probing the state of the internal signals in the design without the use of external equipment. Defining custom trigger-condition logic provides greater accuracy and improves the ability to isolate problems. The SignalTap II Embedded Logic Analyzer does not require external probes, or changes to the design files to capture the state of the internal nodes or I/O pins in the design. All captured signal data is conveniently stored in device memory until the designer is ready to read and analyze the data.

The topics in this chapter include:

- “On-Chip Debugging Tool Comparison” on page 13–5
- “Design Flow Using the SignalTap II Logic Analyzer” on page 13–7
- “SignalTap II Logic Analyzer Task Flow” on page 13–8
- “Add the SignalTap II Logic Analyzer to Your Design” on page 13–10
- “Configure the SignalTap II Logic Analyzer” on page 13–18
- “Define Triggers” on page 13–30
- “Program the Target Device or Devices” on page 13–57
- “Run the SignalTap II Logic Analyzer” on page 13–59
- “View, Analyze, and Use Captured Data” on page 13–63
- “Other Features” on page 13–67
- “SignalTap II Scripting Support” on page 13–72
- “Design Example: Using SignalTap II Logic Analyzers in SOPC Builder Systems” on page 13–77
- “Custom Triggering Flow Application Examples” on page 13–77
The SignalTap II Embedded Logic Analyzer is a next-generation, system-level debugging tool that captures and displays real-time signal behavior in a system-on-a-programmable-chip (SOPC) or any FPGA design. The SignalTap II Embedded Logic Analyzer supports the highest number of channels, largest sample depth, and fastest clock speeds of any embedded logic analyzer in the programmable logic market. Figure 13–1 shows a block diagram of the components that make up the SignalTap II Embedded Logic Analyzer.

**Figure 13–1. SignalTap II Logic Analyzer Block Diagram**

**Note (1)**

This diagram assumes that the SignalTap II Logic Analyzer was compiled with the design as a separate design partition using the Quartus II Incremental Compilation feature. This is the default setting for new projects in the Quartus II software. If incremental compilation is disabled or not used, the SignalTap II logic is integrated with the design. For information about the use of incremental compilation with SignalTap II, refer to “Faster Compilations with Quartus II Incremental Compilation” on page 13–51.

This chapter is intended for any designer who wants to debug their FPGA design during normal device operation without the need for external lab equipment. Because the SignalTap II Embedded Logic Analyzer is similar to traditional external logic analyzers, familiarity with external logic analyzer operations is helpful but not necessary. To take advantage of faster compile times when making changes to the SignalTap II Logic Analyzer, knowledge of the Quartus II Incremental Compilation feature is helpful.
For information about using the Quartus II Incremental Compilation feature, refer to the *Incremental Compilation for Hierarchical and Team-Based Design* chapter in the *Quartus II Handbook*.

### Hardware and Software Requirements

The following components are required to perform logic analysis with the SignalTap II Embedded Logic Analyzer:

- Quartus II design software
  - or
  - Quartus II Web Edition (with TalkBack feature enabled)
  - or
  - SignalTap II Logic Analyzer standalone software
- Download/Upload Cable
- Altera development kit or user design board with JTAG connection to device under test

Captured data is stored in the device’s memory blocks and transferred to the Quartus II software waveform display with a JTAG communication cable, such as EthernetBlaster or USB-Blaster™. Table 13–1 summarizes some of the features and benefits of the SignalTap II Embedded Logic Analyzer.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple logic analyzers in a single device</td>
<td>Captures data from multiple clock domains in a design at the same time</td>
</tr>
<tr>
<td>Multiple logic analyzers in multiple devices in a single JTAG chain</td>
<td>Simultaneously captures data from multiple devices in a JTAG chain</td>
</tr>
<tr>
<td>Plug-In Support</td>
<td>Easily specifies nodes, triggers, and signal mnemonics for IP, such as the Nios® II embedded processor</td>
</tr>
<tr>
<td>Up to 10 basic or advanced trigger conditions for each analyzer instance</td>
<td>Enables more complex data capture commands to be sent to the logic analyzer, providing greater accuracy and problem isolation</td>
</tr>
<tr>
<td>Power-Up Trigger</td>
<td>Captures signal data for triggers that occur after device programming but before manually starting the logic analyzer</td>
</tr>
<tr>
<td>State-Based Triggering Flow</td>
<td>Enables you to organize your triggering conditions to precisely define what your embedded logic analyzer will capture</td>
</tr>
<tr>
<td>Incremental Compilation</td>
<td>Modifies the SignalTap II Logic Analyzer monitored signals and triggers without performing a full compilation, saving time</td>
</tr>
<tr>
<td>Flexible buffer acquisition modes</td>
<td>Allows more accurate data collection by setting each trigger to sample at different ranges relative to the triggering event, in circular or segmented modes</td>
</tr>
<tr>
<td>Feature</td>
<td>Benefit</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MATLAB integration with included MEX function</td>
<td>Acquires the SignalTap II Logic Analyzer captured data into a MATLAB integer matrix</td>
</tr>
<tr>
<td>Up to 1,024 channels in each device</td>
<td>Samples many signals and wide bus structures</td>
</tr>
<tr>
<td>Up to 128K samples in each device</td>
<td>Captures a large sample set for each channel</td>
</tr>
<tr>
<td>Fast clock frequencies</td>
<td>Collects sample data at up to 270 MHz</td>
</tr>
<tr>
<td>Resource usage estimator</td>
<td>Provides estimate of logic and memory device resources used by SignalTap II Embedded Logic Analyzer configurations</td>
</tr>
<tr>
<td>No additional cost</td>
<td>The SignalTap II Logic Analyzer is included with a Quartus II subscription and with the Quartus II Web Edition (with TalkBack enabled)</td>
</tr>
</tbody>
</table>

For a list of supported device families, refer to the Quartus II Help.
On-Chip Debugging Tool Comparison

The Quartus II software provides a number of different ways to help debug your FPGA design after programming the device. The SignalTap II Logic Analyzer, SignalProbe, and the Logic Analyzer Interface (LAI) share some similar features, but each has its own advantages. In some debugging situations, it can be difficult to decide which tool is best to use or whether multiple tools are required. Table 13–2 compares common debugging features between these tools and provides suggestions about which is the best tool to use for a given feature.

<table>
<thead>
<tr>
<th>Feature</th>
<th>SignalProbe</th>
<th>Logic Analyzer Interface (LAI)</th>
<th>SignalTap II Embedded Analyzer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large Sample Depth</td>
<td>N/A</td>
<td>✓</td>
<td>—</td>
<td>An external logic analyzer used with the LAI has a bigger buffer to store more captured data than the SignalTap II Logic Analyzer. No data is captured or stored with SignalProbe.</td>
</tr>
<tr>
<td>Ease in Debugging Timing Issue</td>
<td>N/A</td>
<td>✓</td>
<td>—</td>
<td>An external logic analyzer used with the LAI provides you with access to timing mode, enabling you to debug combined streams of data.</td>
</tr>
<tr>
<td>Minimal Effect on Logic Design</td>
<td>✓</td>
<td>✓ (2)</td>
<td>✓ (2)</td>
<td>The LAI adds minimal logic to a design, requiring fewer device resources. The SignalTap II Logic Analyzer has little effect on the design when it is set as a separate design partition using incremental compilation. SignalProbe incrementally routes nodes to pins, not affecting the design at all.</td>
</tr>
<tr>
<td>Short Compile and Recompile Time</td>
<td>✓</td>
<td>✓ (2)</td>
<td>✓ (2)</td>
<td>SignalProbe attaches incrementally routed signals to previously reserved pins, requiring very little recompilation time to make changes to source signal selections. The SignalTap II Logic Analyzer and the LAI can take advantage of incremental compilation to refit their own design partitions to decrease recompilation time.</td>
</tr>
<tr>
<td>Triggering Capability</td>
<td>N/A</td>
<td>✓</td>
<td>✓</td>
<td>The SignalTap II Logic Analyzer offers triggering capabilities that are comparable to commercial logic analyzers.</td>
</tr>
<tr>
<td>I/O Usage</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>No additional output pins are required with the SignalTap II Logic Analyzer. Both the LAI and SignalProbe require I/O pin assignments.</td>
</tr>
</tbody>
</table>
If you have signals that you want to monitor with external equipment without adding the logic resources required by the SignalTap II Logic Analyzer, consider the use of these other tools available in the Quartus II software. Signals can be quickly routed out to reserved I/O pins as part of an ECO change using SignalProbe, while multiplexed banks of many signals can be made visible on only a few pins with the use of the LAI.

For information about the use of these tools, refer to the *Quick Design Debugging Using SignalProbe* and *In-System Debugging Using External Logic Analyzers* chapters in volume 3 of the *Quartus II Handbook*.

---

Table 13–2. Suggested On-Chip Debugging Tools for Common Debugging Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>SignalProbe</th>
<th>Logic Analyzer Interface (LAI)</th>
<th>SignalTap II Embedded Analyzer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acquisition Speed</td>
<td>N/A</td>
<td>—</td>
<td>☑</td>
<td>The SignalTap II Logic Analyzer can acquire data at speeds of over 200 MHz. The same acquisition speeds are obtainable with an external logic analyzer used with the LAI, but signal integrity issues may limit this.</td>
</tr>
<tr>
<td>No JTAG Connection Required</td>
<td>☑</td>
<td>—</td>
<td>—</td>
<td>An FPGA design with the SignalTap II Logic Analyzer or the LAI requires an active JTAG connection to a host running the Quartus II software. SignalProbe does not require a host for debugging purposes.</td>
</tr>
<tr>
<td>External Equipment</td>
<td>—</td>
<td>—</td>
<td>☑</td>
<td>The SignalTap II Logic Analyzer logic is completely internal to the programmed FPGA device. No extra equipment is required other than a JTAG connection from a host running the Quartus II software or the stand-alone SignalTap II software. SignalProbe and the LAI require the use of external debugging equipment, such as multimeters, oscilloscopes, or logic analyzers.</td>
</tr>
</tbody>
</table>

Notes to Table 13–2:

(1) ☑ indicates the suggested best tool for the feature. — indicates that while the tool is available for that feature, that tool may not give the best results. N/A indicates that the feature is not applicable for the selected tool.

(2) When used with incremental compilation.
Figure 13–2 shows a typical overall FPGA design flow for using the SignalTap II Logic Analyzer in your design. A SignalTap II file (.stp) is added to and enabled in your project, or a SignalTap II HDL function, created with the MegaWizard® Plug-In Manager, is instantiated in your design. The diagram shows the flow of operations from initially adding the SignalTap II Logic Analyzer to your design to the final device configuration, testing, and debugging.
SignalTap II Logic Analyzer Task Flow

To use the SignalTap II Logic Analyzer to debug your design, you perform a number of tasks to add, configure, and run the logic analyzer. Figure 13–3 shows a typical flow of the tasks you complete to debug your design. Refer to the appropriate section of this chapter for more information about each of these tasks.

Figure 13–3. SignalTap II Logic Analyzer Task Flow
Add the SignalTap II Logic Analyzer to Your Design

Create a SignalTap II file or create a parameterized HDL instance representation of the logic analyzer using the MegaWizard Plug-In Manager. If you want to monitor multiple clock domains simultaneously, you can add additional instances of the logic analyzer to your design, limited only by the available resources in your device.

Configure the SignalTap II Logic Analyzer

Once the SignalTap II Logic Analyzer is added to your design, you configure it to monitor the signals you want. You can manually add signals or use a plug-in, such as the Nios II plug-in, to quickly add entire sets of associated signals for a particular IP. You can also specify settings for the data capture buffer, such as its size, the method in which data is captured and stored, and the device memory type to use for the buffer in devices that support memory type selection.

Define Triggers

The SignalTap II Logic Analyzer continuously captures data while it is running. To capture and store specific signal data, you set up triggers that tell the logic analyzer under what conditions to stop capturing data. The SignalTap II Logic Analyzer lets you define Runtime Triggers that range from very simple, such as the rising edge of a single signal, to very complex, involving groups of signals, extra logic, and multiple conditions. Power-Up Triggers give you the ability to capture data from trigger events occurring immediately after the device enters user-mode after configuration.

Compile the Design

With the SignalTap II file configured and triggers defined, you compile your project as usual to include the logic analyzer in your design. Since you may need to frequently change monitored signal nodes or adjust trigger settings during debugging, it is recommended that you use the incremental compilation feature built into the SignalTap II Logic Analyzer, along with Quartus II incremental compilation, to reduce recompile times.

Program the Target Device or Devices

When you are debugging a design with the SignalTap II Logic Analyzer, you can program a target device directly from the SignalTap II file without using the Quartus II Programmer. You can also program multiple devices with different designs and simultaneously debug them.
Run the SignalTap II Logic Analyzer

In normal device operation, you control the logic analyzer through the JTAG connection, specifying when to start looking for your trigger conditions to begin capturing data. With Runtime or Power-Up Triggers, you read and transfer the captured data from the on-chip buffer to the SignalTap II file for analysis.

View, Analyze, and Use Captured Data

Once you have captured data and read it into the SignalTap II file, it is available for analysis and use in the debugging process. Either manually or with a plug-in, you can set up mnemonic tables to make it easier to read and interpret the captured signal data. To speed up debugging, use the Locate feature in the SignalTap II node list to find the locations of problem nodes in other tools in the Quartus II software. Save the captured data for later analysis, or convert it to other formats for sharing and further study.

Add the SignalTap II Logic Analyzer to Your Design

Because the SignalTap II Logic Analyzer is implemented in logic on your target device, it must be added to your FPGA design as another part of the design itself. There are two ways to generate the SignalTap II Logic Analyzer and add it to your design for debugging:

- Create a SignalTap II file (.stp) and use the SignalTap II Editor to configure the details of the logic analyzer
- Create and configure the SignalTap II file with the MegaWizard Plug-In Manager and instantiate it in your design

Creating and Enabling a SignalTap II File

To create an embedded logic analyzer, you can use an existing SignalTap II file or create a new file. Once a file is created or selected, it must be enabled in the project where it is used.

Creating a SignalTap II File

The SignalTap II file contains the SignalTap II Logic Analyzer settings and the captured data for viewing and analysis. To create a new SignalTap II file, perform the following steps:

1. On the File menu, click New.
2. In the New dialog box, click the Other Files tab, and select SignalTap II Logic Analyzer File.
3. Click **OK**.

To open an existing SignalTap II file already associated with your project, on the Tools menu, click **SignalTap II Logic Analyzer**. You can also use this method to create a new SignalTap II file if no SignalTap II file exists for the current project.

To open an existing file, on the File menu, click **Open** and select a SignalTap II file (Figure 13–4).

**Figure 13–4. SignalTap II Editor**

![SignalTap II Editor](image)

**Enabling and Disabling a SignalTap II File for the Current Project**

Whenever you save a new SignalTap II file, the Quartus II software asks you if you want to enable the file for the current project. However, you can add this file manually, change the selected SignalTap II file, or completely disable the logic analyzer by performing the following steps:

1. On the Assignments menu, click **Settings**. The **Settings** dialog box appears.

2. In the **Category** list, select **SignalTap II Logic Analyzer**. The **SignalTap II Logic Analyzer** page appears.
3. Turn on Enable SignalTap II Logic Analyzer. Turn off this option to disable the logic analyzer, completely removing it from your design.

4. In the SignalTap II File name box, type the name of the SignalTap II file you want to include with your design, or browse to and select a file name.

5. Click OK.

**Using the MegaWizard Plug-In Manager to Create Your Embedded Logic Analyzer**

Alternatively, you can create a SignalTap II Logic Analyzer instance by using the MegaWizard Plug-In Manager. The MegaWizard Plug-In Manager generates an HDL file that you instantiate in your design. You can also use a hybrid approach in which you instantiate the MegaWizard Plug-In Manager file in your HDL, and then use the method described in “Creating and Enabling a SignalTap II File” on page 13–10.

**Creating an HDL Representation Using the MegaWizard Plug-In Manager**

The Quartus II software allows you to easily create your SignalTap II Logic Analyzer using the MegaWizard Plug-In Manager. To implement the SignalTap II megafunction, perform the following steps:

1. On the Tools menu, click MegaWizard Plug-In Manager. Page 1 of the MegaWizard Plug-In Manager dialog box appears.

2. Select Create a new custom megafunction variation.

3. Click Next.

4. In the Installed Plug-Ins list, expand the JTAG-accessible Extensions folder, and select SignalTap II Logic Analyzer. Select an output file type and enter the desired name of the SignalTap II megafunction. You can choose AHDL (.tdf), VHDL (.vhd), or Verilog HDL (.v) as the output file type (Figure 13–5).
5. Click Next.

6. Configure the analyzer by specifying the **Sample depth**, **RAM Type**, **Data input port width**, **Trigger levels**, **Trigger input port width**, and whether to enable an external **Trigger in** or **Trigger out** (Figure 13–6).

For information about these settings, refer to “Configure the SignalTap II Logic Analyzer” on page 13–18 and “Define Triggers” on page 13–30.
7. Click Next.

8. Set the Trigger level options by selecting Basic or Advanced (Figure 13–7). If you select Advanced for any trigger level, the next page of the MegaWizard Plug-In Manager displays the Advanced Trigger Condition Editor. You can configure an advanced trigger expression using the number of signals you specified for the trigger input port width.

You cannot define a Power-Up Trigger using the MegaWizard Plug-In Manager. Refer to “Define Triggers” on page 13–30 to learn how to do this using the SignalTap II file.
9. On the final page of the MegaWizard Plug-In Manager, select any additional files you want to create and click Finish to create an HDL representation of the SignalTap II Logic Analyzer.

For information about the configuration settings options in the MegaWizard Plug-In Manager, refer to “Configure the SignalTap II Logic Analyzer” on page 13–18. For information about defining triggers, refer to “Define Triggers” on page 13–30.
**SignalTap II MegafUNCTION Ports**

Table 13–3 provides information about the SignalTap II megafunction ports.

For the most current information about the ports and parameters for this megafunction, refer to the latest version of the Quartus II Help.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acq_data_in</td>
<td>Input</td>
<td>No</td>
<td>This set of signals represents the set of signals that are monitored in the SignalTap II Logic Analyzer.</td>
</tr>
<tr>
<td>acq_trigger_in</td>
<td>Input</td>
<td>No</td>
<td>This set of signals represents the set of signals that are used to trigger the analyzer.</td>
</tr>
<tr>
<td>acq_clk</td>
<td>Input</td>
<td>Yes</td>
<td>This port represents the sampling clock that the SignalTap II Logic Analyzer uses to capture data.</td>
</tr>
<tr>
<td>trigger_in</td>
<td>Input</td>
<td>No</td>
<td>This signal is used to trigger the SignalTap II Logic Analyzer.</td>
</tr>
<tr>
<td>trigger_out</td>
<td>Output</td>
<td>No</td>
<td>This signal is enabled when the trigger event occurs.</td>
</tr>
</tbody>
</table>

**Instantiating the SignalTap II Logic Analyzer in Your HDL**

Instantiating the logic analyzer in your HDL is similar to instantiating any other Verilog HDL or VHDL megafunction in your design. Add the code from the files that are generated by the MegaWizard Plug-In Manager to your design, mapping the signals in your design to the appropriate SignalTap II megafunction ports. You can instantiate up to 127 analyzers in your design, or as many as physically fit in the FPGA. Once you have instantiated the SignalTap II file in your HDL file, compile your Quartus II project to fit the logic analyzer in the target FPGA.

To capture and view the data, you must create a SignalTap II file from your SignalTap II HDL output file. To do this, on the File menu, point to Create/Update, and click **Create SignalTap II File from Design Instance(s)**.

If you make any changes to your design or the SignalTap II instance, recreate or update the SignalTap II file with this command. This ensures that the SignalTap II file is always compatible with the SignalTap II instance in your design. If the SignalTap II file is not compatible with the SignalTap II instance in your design, you may not be able to control the SignalTap II Logic Analyzer after it is programmed into your device.
For information about SignalTap II file compatibility with programmed SignalTap II instances, refer to “Program the Target Device or Devices” on page 13–57.

**Embedding Multiple Analyzers in One FPGA**

The SignalTap II Logic Analyzer includes support for multiple logic analyzers in an FPGA device. This feature allows you to create a unique logic analyzer for each clock domain in the design. As multiple instances of the analyzer are added to the SignalTap II file, the resource usage increases proportionally.

In addition to debugging multiple clock domains, this feature allows you to apply the same SignalTap II settings to a group of signals in the same clock domain. For example, if you have a set of signals that must use a sample depth of 64K and another set of signals in the same clock domain requires a sample depth of 1K, you can create two instances to meet these needs.

To create multiple analyzers, on the Edit menu, click **Create Instance**, or right-click in the Instance Manager window and click **Create Instance**.

Each instance of the SignalTap II Logic Analyzer can be configured independently. The icon in the Instance Manager for the currently active instance that is available for configuration is highlighted in color and surrounded by a blue box. To configure a different instance, double-click the icon or name of another instance in the Instance Manager.

**Monitoring FPGA Resources Used by the SignalTap II Logic Analyzer**

The SignalTap II Logic Analyzer has a built-in resource estimator that calculates the logic resources and amount of memory that each SignalTap II Logic Analyzer uses. You can see the resource usage of each logic analyzer instance and the total resources used in the columns of the Instance Manager section of the SignalTap II Editor. This feature is useful when device resources are limited and you must know what device resources the SignalTap II Logic Analyzer uses. The value reported in the resource usage estimator may vary by as much as 5% from the actual resource usage.
Table 13–4 shows the SignalTap II Logic Analyzer M4K memory block resource usage for the listed devices per signal width and sample depth.

<table>
<thead>
<tr>
<th>Signals (Width)</th>
<th>Samples (Depth)</th>
<th>256</th>
<th>512</th>
<th>2,048</th>
<th>8,192</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>&lt; 1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>4</td>
<td>8</td>
<td>32</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>16</td>
<td>32</td>
<td>128</td>
<td>512</td>
<td></td>
</tr>
</tbody>
</table>

*Note to Table 13–4:*
(1) When you configure a SignalTap II Logic Analyzer, the Instance Manager reports an estimate of the memory bits and logic elements required to implement the given configuration.

Configure the SignalTap II Logic Analyzer

The SignalTap II file provides many options for configuring instances of the logic analyzer. Some of the settings are similar to those found on traditional external logic analyzers. Other settings are unique to the SignalTap II Logic Analyzer because of the requirements for configuring an embedded logic analyzer. All settings give you the ability to configure the logic analyzer the way you want to help debug your design.

Some settings can only be adjusted when you are viewing Run-Time Trigger conditions instead of Power-Up Trigger conditions. To learn about Power-Up Triggers and viewing different trigger conditions, refer to “Creating a Power-Up Trigger” on page 13–45.

Assigning an Acquisition Clock

You must assign a clock signal to control the acquisition of data by the SignalTap II Logic Analyzer. The logic analyzer samples data on every rising edge of the acquisition clock. You can use any signal in your design as the acquisition clock. However, for best results, Altera recommends that you use a global, non-gated clock for data acquisition. Using a gated clock as your acquisition clock can result in unexpected data that does not accurately reflect the behavior of your design. The Quartus II Classic Timing Analyzer shows the maximum acquisition clock frequency at which you can run your design.
Configure the SignalTap II Logic Analyzer

To assign an acquisition clock, perform the following steps:

1. In the SignalTap II Logic Analyzer window, click the Setup tab.

2. Click Browse next to the Clock field in the Signal Configuration pane. The Node Finder dialog box appears.

3. From the Filter list, select SignalTap II: post-fitting or SignalTap II: pre-synthesis.

4. In the Named field, type the exact name of a node that you want to use as your sample clock, or search for a node using a partial name and wildcard characters.

5. To start the node search, click List.

6. In the Nodes Found list, select the node that represents the design’s global clock signal.

7. Add the selected node name to the Selected Nodes list by clicking “>” or by double-clicking the node name.

8. Click OK. The node is now specified as the acquisition clock in the SignalTap II Editor.

If you do not assign an acquisition clock in the SignalTap II Editor, the Quartus II software automatically creates a clock pin called auto_stp_external_clk.

You must make a pin assignment to this pin independently from the design. You must ensure that a clock signal in your design drives the acquisition clock.

For information about assigning signals to pins, refer to the I/O Management chapter in volume 2 of the Quartus II Handbook.

Adding Signals to the SignalTap II File

While configuring the logic analyzer, you add signals to the node list in the SignalTap II file to select which signals in your design you want to monitor. Selected signals are also used to define triggers. You can assign the following two types of signals to your SignalTap II file:

- **Pre-synthesis**—This signal exists after design elaboration, but before any synthesis optimizations are done. This set of signals should reflect your Register Transfer Level (RTL) signals.
Post-fitting—This signal exists after physical synthesis optimizations and place-and-route.

If you are not using incremental compilation, add only pre-synthesis signals to your SignalTap II file. Using pre-synthesis is particularly useful if you want to add a new node after you have made design changes. To do this, on the Processing menu, point to Start and click Start Analysis & Elaboration.

Signals shown in blue text are post-fit node names. Signals shown in black text are pre-synthesis node names.

After successful Analysis and Elaboration, the signals shown in red text are invalid signals. Unless you are certain that these signals are valid, you must remove them from the SignalTap II file for correct operation. The SignalTap II Health Monitor also indicates if an invalid node name exists in the SignalTap II file.

As a general guideline, signals can be tapped if a routing resource (row or column interconnects) exists to route the connection to the SignalTap II instance. For example, signals that exist in the I/O element (IOE) cannot be directly tapped because there are no direct routing resources from the signal in an IOE to a core logic element. For input pins, you can tap the signal that is driving a Logic Array Block (LAB) from an IOE, or, for output pins, you can tap the signal from the LAB that is driving an IOE.

When adding pre-synthesis signals, all connections made to the SignalTap II Logic Analyzer are made prior to synthesis. Logic and routing resources are allocated during recompilation to make the connection as if a change in your design files had been made. As such, pre-synthesis signal names for signals driving to and from IOEs will coincide with the signal names assigned to the pin.

In the case of post-fit signals, connections that you make to the SignalTap II Logic Analyzer are the signal names from the actual atoms in your post-fit netlist. A connection can only be made if the signals are part of the existing post-fit netlist and existing routing resources are available from the signal of interest to the SignalTap II Logic Analyzer. In the case of post-fit output signals, tap the COMBOUT or REGOUT signal that drives the IOE block. For post-fit input signals, signals driving into the core logic will coincide with the signal name assigned to the pin.
If you are tapping the signal from the atom that is driving an IOE, be aware that the signal may be inverted due to \textit{NOT}-gate push back. You can check this by locating the signal in either the Resource Property Editor or the Technology Map Viewer. The Technology Map viewer and the Resource Property Editor are also helpful in finding post-fit node names.

For information about cross-probing to source design file and other Quartus II windows, refer to the \textit{Analyzing Designs with Quartus II Netlist Viewers} chapter in volume 1 of the \textit{Quartus II Handbook}.

For more information about the use of incremental compilation with the SignalTap II Logic Analyzer, refer to “Faster Compilations with Quartus II Incremental Compilation” on page 13–51.

\textbf{Signal Preservation}

Many of your RTL signals are optimized during the process of synthesis and place-and-route. The RTL signal names frequently may not appear in the post-fit netlist after optimizations. This can cause a problem when you use the incremental compilation flow with the SignalTap II Logic Analyzer. Since only post-fitting signals can be added to the SignalTap II Logic Analyzer in partitions of type \textbf{post-fit}, RTL signals that you want to monitor may not be available, preventing their usage. To avoid this issue, you can use synthesis attributes to preserve signals during synthesis and place-and-route. When the Quartus II software encounters these synthesis attributes, it does not perform any optimization on the specified signals, forcing them to continue to exist in the post-fit netlist. However, if you do this, you could see an increase in resource utilization or a decrease in timing performance. The two attributes you can use are:

- \textbf{keep}—Ensures that combinational signals are not removed
- \textbf{preserve}—Ensures that registers are not removed

For more information about using these attributes, refer to the \textit{Quartus II Integrated Synthesis} chapter in volume 1 of the \textit{Quartus II Handbook}.

If you are debugging an IP core, such as the Nios II CPU, or other encrypted IP, you may need to preserve nodes from the core to make them available for debugging with the SignalTap II Logic Analyzer. This is often necessary when a plug-in is used to add a group of signals for a particular IP. To do this, on the Assignments menu, click \textbf{Settings}. In the \textbf{Category} list, select \textbf{Analysis \& Synthesis Settings}. Turn on \textbf{Create debugging nodes for IP cores} to make these nodes available to the SignalTap II Logic Analyzer.
Assigning Data Signals

To assign data signals, perform the following steps:

1. Perform Analysis and Elaboration, Analysis and Synthesis, or compile your design.

2. In the SignalTap II Logic Analyzer window, click the Setup tab.

3. Double-click anywhere in the node list of the SignalTap II Editor to open the Node Finder dialog box.

4. In the Fitter list, select SignalTap II: pre-synthesis or SignalTap II: post-fitting. Only signals listed under one of these filters can be added to the SignalTap II node list. Signals cannot be selected from any other filters.

   If you use Incremental Compilation flow with SignalTap II, pre-synthesis nodes will not be connected to the SignalTap II Logic Analyzer if the affected partition is of the post-fit type. Any pre-synthesis nodes added to a partition of type post-fit may not be connected to the SignalTap II Logic Analyzer. A critical warning is issued for all pre-synthesis node names that are not found in the post-fit netlist. Altera recommends that you do not add a mix of pre-synthesis and post-fitting signals within the same partition. For more details, refer to “Using Incremental Compilation with the SignalTap II Logic Analyzer” on page 13–52.

5. In the Named field, type a node name, or search for a particular node by entering a partial node name along with wildcard characters. To start the node name search, click List.

6. In the Nodes Found list, select the node or bus you want to add to the SignalTap II file.

7. Add the selected node name(s) to the Selected Nodes list by clicking “>” or by double-clicking the node name(s).

8. To insert the selected nodes in the SignalTap II file, click OK. With the default colors set for the SignalTap II Logic Analyzer, a pre-synthesis signal in the list is shown in black, and a post-fitting signal is shown in blue.

   You can also drag and drop signals from the Node Finder dialog box into a SignalTap II file.
Configure the SignalTap II Logic Analyzer

Node List Signal Use Options

Once a signal is added to the node list, you can select options that specify how the signal is used with the logic analyzer. You can turn off the ability of a signal to trigger the analyzer by disabling the Trigger Enable for that signal in the node list in the SignalTap II file. This option is useful when you want to see only the captured data for a signal, and you are not using that signal as part of a trigger.

You can turn off the ability to view data for a signal by disabling the Data Enable column. This option is useful when you want to trigger on a signal, but have no interest in viewing data for that signal.

For information about using signals in the node list to create SignalTap II trigger conditions, refer to “Define Triggers” on page 13–30.

Untappable Signals

Not all of the post-fitting signals in your design are available in the SignalTap II: post-fitting filter in the Node Finder dialog box. The following signal types cannot be tapped:

- Post-fit output pins—You cannot tap a post-fit output pin directly. To make an output signal visible, tap the register or buffer that drives the output pin.
- Signals that are part of a carry chain—You cannot tap the carry out (cout0 or cout1) signal of a logic element. Due to architectural restrictions, the carry out signal can only feed the carry in of another logic element (LE).
- JTAG Signals—You cannot tap the JTAG control (TCK, TDI, TDO, and TMS) signals.
- altgxb megafunction—You cannot directly tap any ports of an altgxb instantiation.
- LVDS—You cannot tap the data output from a serializer/deserializer (SERDES) block.

Adding Signals with a Plug-In

Instead of adding individual or grouped signals through the Node Finder, you can add groups of relevant signals of a particular type of IP through the use of a plug-in. The SignalTap II Logic Analyzer comes with one plug-in already installed for the Nios II processor. Besides easy signal addition, plug-ins also provide a number of other features, such as pre-designed mnemonic tables, useful for trigger creation and data viewing, as well as the ability to disassemble code in captured data.
The Nios II plug-in, for example, creates one mnemonic table in the Setup tab, and two tables in the Data tab:

- **Nios II Instruction** (Setup tab)—Capture all the required signals for triggering on a selected instruction address.
- **Nios II Instance Address** (Data tab)—Display address of executed instructions in hexadecimal format or as a programming symbol name if defined in an optional Executable and Linking Format (.elf) file.
- **Nios II Disassembly** (Data tab)—Displays disassembled code from the corresponding address.

For information about the other features plug-ins provided, refer to “Define Triggers” on page 13–30 and “View, Analyze, and Use Captured Data” on page 13–63.

To add signals to the SignalTap II file using a plug-in, perform the following steps after running Analysis and Elaboration on your design:

1. Right-click in the node list. On the **Add Nodes with Plug-In** submenu, click the name of the plug-in you want to use, such as the included plug-in named Nios II.

   ![If the intellectual property (IP) for the selected plug-in does not exist in your design, a message appears informing you that you cannot use the selected plug-in.](image)

2. The **Select Hierarchy Level** dialog box appears showing the IP hierarchy of your design (Figure 13–8). Select the IP that contains the signals you want to monitor with the plug-in, and click **OK**.
3. If all the signals in the plug-in are available, a dialog box may appear, depending on the plug-in selected, where you can set any available options for the plug-in. With the Nios II plug-in, you can optionally select an Executable and Linking Format (.elf) file containing program symbols from your Nios II Integrated Development Environment (IDE) software design. Set options for the selected plug-in as desired, and click OK.

To make sure all the required signals are available, turn on the Create debugging nodes for IP cores option in the Quartus II Analysis & Synthesis settings.

All the signals included in the plug-in are added to the node list.

**Specifying the Sample Depth**

The sample depth specifies the number of samples that are captured and stored for each signal in the captured data buffer. To set the sample depth, select the desired number of samples to store in the Sample Depth list. The sample depth ranges from 0 to 128K.

If device memory resources are limited, you may not be able to successfully compile your design with the sample buffer size you have selected. Try reducing the sample depth to reduce resource usage.
Capturing Data to a Specific RAM Type

When you use the SignalTap II Logic Analyzer with some devices, you have the option to select the RAM type where acquisition data is stored. RAM selection allows you to preserve a specific memory block for your design and allocate another portion of memory for SignalTap II data acquisition. For example, if your design implements a large buffering application such as a system cache, it is ideal to place this application into M-RAM blocks so that the remaining M512 or M4K blocks are used for SignalTap II data acquisition.

To select the RAM type to use for the SignalTap II buffer, select it from the RAM type list. Use this feature when the acquired data (as reported by the SignalTap II resource estimator) is not larger than the available memory of the memory type that you have selected in the FPGA.

Choosing the Buffer Acquisition Mode

The buffer acquisition type selection feature in the SignalTap II Logic Analyzer lets you choose how the captured data buffer is organized and can potentially reduce the amount of memory that is required for SignalTap II data acquisition. You can choose to use either a circular buffer, which allocates the entire sample depth to a single buffer, or a segmented buffer, which splits the buffer space into a number of separate even sized segments. Figure 13–9 illustrates the differences between the two buffer types.

Figure 13–9. Buffer Type Comparison in the SignalTap II Logic Analyzer Note (1)

![Buffer Type Comparison Diagram](image)

Note to Figure 13–9:
(1) Both circular and segmented buffers can use a predefined trigger position or define a custom trigger position using the State-Based Triggering tab. Refer to “Specifying the Trigger Position” on page 13–44 for more details.
Circular Buffer

The circular buffer (Figure 13–9 (a)) is the default buffer type used by the SignalTap II Logic Analyzer. While the logic analyzer is running, data is stored in the buffer until it fills up, at which point new data replaces the oldest data. This continues until a specified trigger event occurs. When this happens, the logic analyzer continues to capture data after the trigger event until the buffer is full, based on the circular buffer trigger position setting in the Signal Configuration pane in the SignalTap II file. Select a setting from the list to choose whether to capture the majority of the data before (Post trigger position) or after (Pre trigger position) the trigger occurs or to center the trigger position in the data (Center trigger position). Another option is to use the custom state-based triggering flow to define your desired triggering position precisely. You can also choose to continuously capture data until the logic analyzer is stopped.

For more information, refer to “Specifying the Trigger Position” on page 13–44.

Segmented Buffer

The segmented buffer (Figure 13–9 (b)) organizes the buffer into a number of separate, evenly sized segments. This type of buffer organization makes it easier to debug systems that contain relatively infrequent recurring events. Figure 13–10 shows an example of this type of buffer system.

Figure 13–10. Example System that Generates Recurring Events

The SignalTap II Logic Analyzer verifies the functionality of the design shown in Figure 13–10 to ensure that the correct data is written to the SRAM controller. The buffer acquisition in the SignalTap II Logic
Analyzer allows you to monitor the \texttt{RDATA} port when \texttt{H'0F0F0F0F} is sent into the \texttt{RADDR} port. You can monitor multiple read transactions from the SRAM device without running the SignalTap II Logic Analyzer again. The buffer acquisition feature allows you to segment the memory so that you can capture the same event multiple times without wasting the allocated memory. The number of cycles that are captured depends on the number of segments that you have specified under the \texttt{Data} settings.

To enable and configure buffer acquisition, select \textbf{Segmented} in the SignalTap II Editor, and select the number of segments to use. In the example, selecting sixty-four, 64-sample segments allows you to capture 64 read cycles when the \texttt{RADDR} signal is \texttt{H'0F0F0F0F}.

For more information about the buffer acquisition mode, refer to \textit{Setting the Buffer Acquisition Mode} in the Quartus II Help.

**Managing Multiple SignalTap II Files and Configurations**

In some cases you may have more than one SignalTap II file in one design. Each file potentially has a different group of monitored signals. These signal groups make it possible to debug different blocks in your design. In turn, each group of signals may also be used to define different sets of trigger conditions. Along with each SignalTap II file, there is also an associated programming file (SRAM Object File (SOF)). The settings in a selected SignalTap II file must match the SignalTap II logic design in the associated SOF file for the logic analyzer to run properly when the device is programmed. Managing all of the SignalTap II files and their associated settings and programming files is a challenging task. To help you manage everything, you can use the \texttt{Data Log} feature and the \texttt{SOF Manager}.

The \texttt{Data Log} allows you to store multiple SignalTap II configurations within a single SignalTap II file. \textbf{Figure 13–11} shows two signal set configurations with multiple trigger conditions in one SignalTap II file. To toggle between the active configurations, double-click on an entry in the \texttt{Data Log}. As you toggle between the different configurations, the signal list and trigger conditions change in the \texttt{Setup} tab of the SignalTap II file. The active configuration displayed in the SignalTap II file is indicated by the blue square around the signal set in the \texttt{Data log}. To store a configuration in the data log, on the Edit menu, click \texttt{Save to Data Log}, or click the \texttt{Save to Data Log} button at the top of the Data Log.
The SOF Manager allows you to embed multiple SOFs into one SignalTap II file. Embedding an SOF in a SignalTap II file lets you move the SignalTap II file to a different location, either on the same computer or across a network, without the need to include the associated SOF as a separate file. To embed a new SOF in the SignalTap II file, right-click in the SOF Manager, and click **Attach SOF File** (Figure 13–12).

As you switch between configurations in the Data Log, you can extract the SOF that is compatible with that particular configuration and use the programmer in the SignalTap II Logic Analyzer to download the new SOF to the FPGA. In this way, you ensure that the configuration of your SignalTap II file always matches the design programmed into the target device.
Define Triggers

To capture the data you want at the right time, you need to specify conditions under which the signals you are monitoring display that data. In the SignalTap II Logic Analyzer, these conditions are referred to as triggers, just as they are in conventional external logic analyzers and oscilloscopes. You have many options for creating different types of triggers to help in your debugging.

Creating Basic Trigger Conditions

The simplest kind of trigger condition you can use is a basic trigger. You select this from the list at the top of the Trigger Conditions column in the node list in the SignalTap II Editor. With the trigger type set to Basic, you must set the trigger pattern for each signal you have added in the SignalTap II file. To set the trigger pattern, right-click in the Trigger Conditions column and click the desired pattern. You can set the trigger pattern to any of the following conditions:

- Don’t Care
- Low
- High
- Falling Edge
- Rising Edge
- Either Edge

For buses, you can type a pattern in binary, or right-click and select Insert Value to enter the pattern in other number formats. For signals added to the SignalTap II file that have an associated mnemonic table, you can right-click and select an entry from the table to set pre-defined conditions for the trigger.

For more information about the creation and use of mnemonic tables, refer to “View, Analyze, and Use Captured Data” on page 13–63 and in the Quartus II Help.

For signals added with certain plug-ins, you can easily create basic triggers using pre-defined mnemonic table entries. For example, with the Nios II plug-in, if you have specified an executable software (.elf) file from your Nios II IDE design, you can type the name of a function from your Nios II code. The logic analyzer triggers when the Nios II instruction address matches the address of the specified code function name.

Data capture stops and the data is stored in the buffer when the logical AND of all the signals for a given trigger condition evaluates to TRUE.
Creating Advanced Trigger Conditions

Along with the SignalTap II Logic Analyzer’s basic triggering capabilities, you can build more complex triggers utilizing extra logic that enable you to capture data when a particular combination of conditions exist. If you set the trigger type to **Advanced** at the top of the Trigger Conditions column in the node list of the SignalTap II Editor, a new tab named **Advanced Trigger** appears where you can build a complex trigger expression using a simple GUI. You can drag and drop operators into the Advanced Trigger Configuration Editor window to build the complex trigger condition in an expression tree. Double-click operators that you have placed or right-click them and select Properties to configure the operator’s settings. Table 13–5 lists the operators you can use.

### Table 13–5. Advanced Triggering Operators

<table>
<thead>
<tr>
<th>Name of Operator</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less Than</td>
<td>Comparison</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>Comparison</td>
</tr>
<tr>
<td>Equality</td>
<td>Comparison</td>
</tr>
<tr>
<td>Inequality</td>
<td>Comparison</td>
</tr>
<tr>
<td>Greater Than</td>
<td>Comparison</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>Comparison</td>
</tr>
<tr>
<td>Logical NOT</td>
<td>Logical</td>
</tr>
<tr>
<td>Logical AND</td>
<td>Logical</td>
</tr>
<tr>
<td>Logical OR</td>
<td>Logical</td>
</tr>
<tr>
<td>Logical XOR</td>
<td>Logical</td>
</tr>
<tr>
<td>Reduction AND</td>
<td>Reduction</td>
</tr>
<tr>
<td>Reduction OR</td>
<td>Reduction</td>
</tr>
<tr>
<td>Reduction XOR</td>
<td>Reduction</td>
</tr>
<tr>
<td>Left Shift</td>
<td>Shift</td>
</tr>
<tr>
<td>Right Shift</td>
<td>Shift</td>
</tr>
<tr>
<td>Bitwise Complement</td>
<td>Bitwise</td>
</tr>
<tr>
<td>Bitwise AND</td>
<td>Bitwise</td>
</tr>
<tr>
<td>Bitwise OR</td>
<td>Bitwise</td>
</tr>
<tr>
<td>Bitwise XOR</td>
<td>Bitwise</td>
</tr>
<tr>
<td>Edge and Level Detector</td>
<td>Signal Detection</td>
</tr>
</tbody>
</table>

*Note to Table 13–5:*  
(1) For more information about each of these operators, refer to the Quartus II Help.
You can configure some of the settings for certain operators at run-time. This enables you to change one operator type to another operator type or adjust other settings for an operator without recompiling your design. Operator settings that have a white background on the operator symbol can be changed without recompiling the design.

Adding many objects to the Advanced Trigger Condition Editor can make the workspace cluttered and difficult to read. To keep objects organized while you build your advanced trigger condition, use the right-click menu and select **Arrange All Objects**. You can also use the **Zoom-Out** command to fit more objects into the Advanced Trigger Condition editor window.

**Examples of Advanced Triggering Expressions**

The following examples show how to use Advanced Triggering:

- Trigger when bus `outa` is greater than or equal to `outb` (Figure 13–13).

**Figure 13–13. Bus outa is Greater Than or Equal to Bus outb**

- Trigger when bus `outa` is greater than or equal to `outb`, and when the enable signal has a rising edge (Figure 13–14).
Define Triggers

Figure 13–14. Enable Signal has a Rising Edge

Result: outa>=outb&&ELD(enable)

- Trigger when bus outa is greater than or equal to bus outb, or when the enable signal has a rising edge. Or, when a bitwise AND operation has been performed between bus outc and bus outd, and all bits of the result of that operation are equal to 1 (Figure 13–15).

Figure 13–15. Bitwise AND Operation

Result: outa>=outb||ELD(enable)||(outc&&outd)
Trigger Condition Flow Control

SignalTap II offers multiple triggering conditions to give you more precise control of the method in which data is captured into the acquisition buffers. Trigger Condition Flow control allows you to define the relationship between a set of triggering conditions. SignalTap II gives you two flow control mechanisms for organizing trigger conditions:

- Sequential Triggering—The default triggering flow. This flow allows you to define up to ten triggering levels that must be satisfied before the acquisition buffer finishes capturing.
- Custom State-Based Triggering—This flow allows you the greatest control over your acquisition buffer. This method allows you to organize trigger conditions into states based on a conditional flow that you define.

Both methods can be used with either a circular buffer or a segmented buffer.

Sequential Triggering

The sequential triggering flow allows you to cascade up to ten levels of triggering conditions. The SignalTap II Logic Analyzer sequentially evaluates each of the triggering conditions. When the last triggering condition evaluates to TRUE, the SignalTap II Logic Analyzer triggers the acquisition buffer. For segmented buffers, every acquisition segment after the first segment triggers on the last triggering condition that you have specified. You can use the simple sequential triggering feature with basic triggers, advanced triggers, or a mix of both. Figure 13–16 illustrates the simple sequential triggering flow for circular and segmented buffers.

Note that the external trigger in is considered as trigger level 0. The external trigger must be evaluated before the main trigger levels are evaluated.
To configure the SignalTap II Logic Analyzer for Sequential triggering, on the Trigger flow control list in the SignalTap II editor, select **Sequential**. You can select the desired number of trigger conditions by using the **Trigger Conditions** pull-down list. After you select the desired number of trigger conditions, you can configure each trigger condition in the node list. To disable any trigger condition, click the check box next to the trigger condition at the top of the column in the node list. Figure 13–17 shows the setup tab for Sequential Triggering.

**Note to Figure 13–16:**

1. The Acquisition buffer stops capture when all \( n \) triggering levels are satisfied, where \( n \leq 10 \).
2. An external trigger input, if defined, will be evaluated before all other defined trigger conditions are evaluated. For more information about external triggers refer to “Using External Triggers” on page 13–47.
Figure 13–17. Setup Tab

Custom State-Based Triggering

The custom state-based triggering method gives you the most control of triggering condition arrangement. This flow gives you the ability to describe the relationship between triggering conditions precisely, using an intuitive GUI and the SignalTap II Trigger Flow Description Language, a simple description language based upon conditional expressions. Tooltips within the custom triggering flow GUI allow you to describe your desired flow quickly. The custom state-based triggering flow allows for more efficient use of the space available in the acquisition buffer because only specific samples of interest are captured.

Figure 13–18 illustrates the custom state-based triggering flow. Events that trigger the acquisition buffer are organized by a user-defined state diagram. All actions performed by the acquisition buffer are captured by the states and all the transition conditions between the states are defined by the conditional expressions that you specify within each state.
Each state allows you to define a set of conditional expressions. Each conditional expression is a Boolean expression dependent upon a combination of triggering conditions (configured within the Setup tab), counters, and status flags. Counters and status flags are resources provided by the Signal Tap II custom-based triggering flow.

Within each conditional expression you define a set of “actions”. Actions include triggering the acquisition buffer to stop capture, a modification to either a counter or status flag, or a state transition.

Trigger actions can apply to either a single segment of a segmented acquisition buffer or to the entire circular acquisition buffer. Each trigger action provides you with an optional count that specifies the number of samples to be captured before stopping acquisition of the current segment. The count argument allows you to control the amount of data captured precisely before and after triggering event.

Resource manipulation actions allow you to increment and decrement counters or set and clear status flags. The counter and status flag resources are used as optional inputs in the conditional expressions. Counters and status flags are useful for counting the number of occurrences of particular events and for aiding in the triggering flow control.
This SignalTap II custom state-based triggering flow allows you to capture a sequence of events that may not necessarily be contiguous in time; for example, capturing a communication transaction between two devices that includes a handshaking protocol containing a sequence of acknowledgements.

The **State-Based Trigger Flow** tab is the control interface for the custom state-based triggering flow. To enable this tab, on the Trigger Flow Control pull-down list, select **State-based**. (Note that when the Trigger Flow control option is set to **Sequential**, the **State-Based Trigger Flow** tab is hidden.)

Figure 13–19 shows the **Custom Trigger Flow** tab.

![State-Based Trigger Flow Tab](image)

The State-Based Trigger Flow tab is partitioned into the following three panes.

- **State Diagram Pane**
- **Resources Pane**
- **State Machine Pane**
State Diagram Pane
The State Diagram pane provides a graphical overview of the triggering flow that you define. It shows the number of states available and the state transitions between all of the states. You can adjust the number of available states by using the pull-down menu above the graphical overview.

State Machine Pane
The State Machine pane contains the text entry boxes where you can define the triggering flow and the actions associated with each state. You can define the triggering flow using the SignalTap II Trigger Flow Description Language, a simple language based upon if-else conditional statements. Tooltips appear when you move the mouse over the cursor, to guide command entry into the state boxes. The GUI provides a syntax check on your flow description in real-time and highlights any errors in the text flow.

Refer to “SignalTap II Trigger Flow Description Language” on page 13–40 for a full description of the SignalTap II Trigger Flow Description Language. You can also refer to the Quartus II Help.

The State Machine description text boxes default to show one text box per state. You can optionally have the entire flow description be shown in a single text field. This option can be useful when copying and pasting a flow description from a template or an external text editor. To toggle between one window per state, or all states in one window, select the appropriate option under State Display mode.

Resources Pane
The Resources pane allows you to declare Status Flags and Counters for use in the conditional expressions in the Custom Triggering Flow. Actions to decrement and increment counters or to set and clear status flags are performed within the triggering flow that you define.

You can set up to 20 counters and 20 status flags for use. Counter and status flags values may be initialized by right-clicking the status flag or counter name after selecting a number of them from the respective drop-down list, and selecting Set Initial Value. Counter width can be set by right-clicking the counter name and selecting Set Width.
Runtime Reconfigurability—The **configurable at runtime** options in the Resources pane allows you to configure the custom-flow control options that can be changed at runtime without requiring a recompilation. Table 13–6 contains a description of options that can be reconfigured at runtime.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination of goto action</td>
<td>Allows you to modify the destination of the state transition at runtime.</td>
</tr>
<tr>
<td>Comparison values</td>
<td>Allows comparison values in Boolean expressions to be modifiable at runtime. In addition, it allows the <em>segment_trigger</em> and trigger action post-fill count argument to be modifiable at runtime.</td>
</tr>
<tr>
<td>Comparison operators</td>
<td>Allows comparison operators in Boolean expressions to be modifiable at runtime.</td>
</tr>
<tr>
<td>Logical operators</td>
<td>Allows the logical operators in Boolean expressions to be modifiable at runtime.</td>
</tr>
</tbody>
</table>

You can restrict changes to your SignalTap configuration to include only the options that do not require a recompilation by using the pull-down menu above the trigger list in the Setup tab. The option **Allow trigger condition changes only** restricts changes to only the configuration settings that the runtime configurable option set. You can then modify Trigger Flow conditions in the Custom Trigger Flow tab by clicking the desired parameter in the text box, and selecting a new parameter from the menu that appears.

The runtime configurable settings for the Custom Trigger Flow tab are on by default. You may get some performance advantages by disabling some of the runtime configurable options. Refer to “Performance and Resource Considerations” on page 13–55 for details about the effects of turning off the runtime modifiable options.

### SignalTap II Trigger Flow Description Language

The Trigger Flow Description Language is based on a list of conditional expressions per state to define a set of actions. Each line in Example 13–1 shows a language format. Keywords are shown in bold. Non-terminals are delimited by “<>” and are further explained in the following sections. Optional arguments are delimited by “[]”.

Examples of Triggering Flow descriptions for common scenarios using the Signal Tap II Custom Triggering Flow are provided in the section, “Custom Triggering Flow Application Examples” on page 13–77.
**Example 13–1. Trigger Flow Description Language Format Note (1)**

```plaintext
state <State_label>:  
<action_list>

or

state <State_label>:  
if ( <Boolean_expression> )  
<action_list>  
[else if ( <boolean_expression> )  
<action_list>] (1)  
[else  
<action_list>]
```

Notes to Example 13–1:
(1) Multiple else if conditions are allowed.

The priority for evaluation of conditional statements is assigned from top to bottom. The `<boolean_expression>` in an `if` statement can contain a single event, or it can contain multiple event conditions. The `action_list` embedded within an `if` or an `else if` clause must be delimited by the `begin` and `end` tokens when the action list contains multiple statements. When the boolean expression is evaluated true, the logic analyzer analyzes all of the commands in the action list concurrently. The possible actions include:

- Triggering the acquisition buffer
- Manipulating a counter or status flag resource
- Defining a state transition

**State Labels**

State Labels are identifiers that can be used in the action `goto`.

```plaintext
state <state_label>:  begins the description of the actions evaluated when this state is reached.
```

The description of a state ends with the beginning of another state or the end of the whole trigger flow description.

**Boolean_expression**

Boolean_expression is a collection of logical operators, relational operators, and their operands that evaluate into a Boolean result. Depending on the operator, the operand can be a reference to a trigger condition, a counter and a register, or a numeric value. Within an expression, parentheses can be used to group a set of operands.
Logical operators accept any boolean expression as an operand. The supported Logical operators are shown in Table 13–7.

### Table 13–7. Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>NOT operator</td>
<td>expr1</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>AND operator</td>
<td>expr1 &amp;&amp; expr2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Relational operators are performed on counters or status flags. The comparison value—the right operator—must be a numerical value. The supported Relational operators are shown in Table 13–8.

### Table 13–8. Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>&lt;identifier&gt; &gt; &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or Equal to</td>
<td>&lt;identifier&gt; &gt;= &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>==</td>
<td>Equals</td>
<td>&lt;identifier&gt; == &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>!=</td>
<td>Does not equal</td>
<td>&lt;identifier&gt; != &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>&lt;identifier&gt; &lt;= &lt;numerical_value&gt;</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>&lt;identifier&gt; &lt; &lt;numerical_value&gt;</td>
</tr>
</tbody>
</table>

**Notes to Table 13–8:**

1. `<identifier>` indicates a counter or status flag
2. `<numerical_value>` indicates an integer

**Action_list**

Action_list is a list of actions that can be performed when a state is reached and a condition is also satisfied. If more than one action is specified, they must be enclosed by begin and end. The actions can be categorized as resource manipulation actions, buffer control actions and state transition actions. Actions must be embedded within a condition statement if condition statements are used in a state. Each action is terminated by a semicolon.
Resource Manipulation Action

The resources used in the trigger flow description can be either counters or status flags. Table 13–9 shows the description and syntax of each action.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increment</td>
<td>Increments a counter resource by 1</td>
<td>increment &lt;counter_identifier&gt;;</td>
</tr>
<tr>
<td>Decrement</td>
<td>Decrements a counter resource by 1</td>
<td>decrement &lt;counter_identifier&gt;;</td>
</tr>
<tr>
<td>Reset</td>
<td>Resets counter resource to initial value</td>
<td>reset &lt;counter_identifier&gt;;</td>
</tr>
<tr>
<td>Set</td>
<td>Sets a status Flag to 1</td>
<td>set &lt;register_flag_identifier&gt;;</td>
</tr>
<tr>
<td>Clear</td>
<td>Sets a status Flag to 0</td>
<td>clear &lt;register_flag_identifier&gt;</td>
</tr>
</tbody>
</table>

Buffer Control Action

Buffer control actions specify an action to control the acquisition buffer. Table 13–10 shows the description and syntax of each action.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger</td>
<td>Stops the acquisition for the current buffer and ends analysis. This command is required in every flow definition.</td>
<td>trigger &lt;post-fill_count&gt;;</td>
</tr>
<tr>
<td>segment_trigger</td>
<td>Ends the acquisition of the current segment. The Signal Tap II Logic Analyzer starts acquiring from the next segment upon evaluating this command. If all segments are filled, the oldest segment is overwritten with the latest sample. The acquisition stops when a trigger action is evaluated. This action cannot be used in non-segmented acquisition mode.</td>
<td>segment_trigger &lt;post-fill_count&gt;;</td>
</tr>
</tbody>
</table>

Both trigger and segment_trigger actions accept an optional post-fill count argument. If provided, the current acquisition acquires the number of samples provided by post-fill count and then stops acquisition. If no post-count value is specified, the trigger position for the affected buffer defaults to the trigger position specified in the setup tab.
Note that in the case of `segment_trigger`, acquisition of the current buffer stops immediately if a subsequent triggering action is issued in the next state, regardless of whether or not the post-fill count has been satisfied for the current buffer. The remaining unfilled post-count acquisitions in the current buffer are discarded and displayed as grayed-out samples in the data window.

**State Transition Action**

State transition action specifies the next state in the custom state control flow. It is specified by the `goto` command. The syntax is as follows:

```
goto <state_label>;```

**Specifying the Trigger Position**

The SignalTap II Logic Analyzer allows you to specify the amount of data that is acquired before and after a trigger event. You can set the trigger position independently between a Runtime and Power-Up Trigger. Select the desired ratio of pre-trigger data to post-trigger data by choosing one of the following ratios:

- **Pre**—This selection saves signal activity that occurred after the trigger (12% pre-trigger, 88% post-trigger).
- **Center**—This selection saves 50% pre-trigger and 50% post-trigger data.
- **Post**—This selection saves signal activity that occurred before the trigger (88% pre-trigger, 12% post-trigger).

These pre-defined ratios apply to both circular buffers and segmented buffers.

If you use the custom-state based triggering flow, you can specify a custom trigger position. The `segment_trigger` and `trigger` actions accept a post-fill count argument. The post-fill count specifies the number of samples to capture before stopping data acquisition for the circular buffer or a data segment when using the `trigger` and `segment_trigger` commands, respectively. When the captured data is displayed in the SignalTap II data window, the trigger position appears as the number of post-count samples from the end of the acquisition segment or buffer. Refer to Equation 1:

\[
(1) \quad \text{Sample Number of Trigger Position} = (N - \text{Post-Fill Count})
\]

In this case, $N$ is the sample depth of either the acquisition segment or circular buffer.
For segmented buffers, the acquisition segments that have a post-count argument defined use the post-count setting. Segments that do not have a post-count setting default to the trigger position ratios defined in the Setup tab.

For more details about the Custom-State based triggering flow, refer to “Custom State-Based Triggering” on page 13–36.

Creating a Power-Up Trigger

Typically, the SignalTap II Logic Analyzer is used to trigger on events that occur during normal device operation. You start an analysis manually once the target device is fully powered on and the device’s JTAG connection is available. However, there may be cases when you would like to capture trigger events that occur during device initialization immediately after the FPGA is powered on or reset. With the SignalTap II Power-Up Trigger feature, you can capture data from triggers that occur after device programming but before the logic analyzer is started manually.

Enabling a Power-Up Trigger

A different Power-Up Trigger can be added to each logic analyzer instance in the SignalTap II Instance Manager. To enable the Power-Up Trigger for a logic analyzer instance, right-click the instance, and click Enable Power-Up Trigger, or select the instance, and on the Edit menu, click Enable Power-Up Trigger. To disable a Power-Up Trigger, click Disable Power-Up Trigger in the same locations. Power-Up Trigger is shown as a child instance below the name of the selected instance with the default trigger conditions set in the node list. Figure 13–20 shows the SignalTap II Editor when a Power-Up Trigger is enabled.
Managing and Configuring Power-Up and Runtime Trigger Conditions

When the Power-Up Trigger is enabled for a logic analyzer instance, you create basic and advanced trigger conditions for it in the same way you do with the regular trigger, also called the Runtime Trigger. Power-Up Trigger conditions that you can adjust are color coded light blue, while Run-Time Trigger conditions remain white. Since each instance now has two sets of trigger conditions, the Power-Up Trigger and the Run-Time Trigger, you can differentiate between the two with the color coding. To switch between the trigger conditions of the Power-Up Trigger and the Runtime Trigger, double-click the instance name or the Power-Up Trigger name in the Instance Manager.

You cannot make changes to the Power-Up Trigger conditions that would normally require a full recompile with Runtime Trigger conditions, such as adding signals, deleting signals, or changing between basic and advanced triggers. For these changes to be applied to the Power-up Trigger conditions, you must first make the changes using the Runtime Trigger conditions.
Any change made to the Power-Up Trigger conditions requires that the SignalTap II Logic Analyzer be recompiled, even if a similar change to the Runtime Trigger conditions does not require a recompilation.

While creating or making changes to the trigger conditions for the Run-Time Trigger or the Power-Up Trigger, you may want to copy these conditions to the other trigger. This makes it easy to look for the same trigger during both power-up and runtime. To do this, right-click the instance name or the Power-Up Trigger name in the Instance Manager, and click **Duplicate Trigger**, or select the instance name or the Power-Up Trigger name and, on the Edit menu, click **Duplicate Trigger**.

For information about running the SignalTap II Logic Analyzer instance with a Power-Up Trigger enabled, refer to “Running with a Power-Up Trigger” on page 13–60.

**Using External Triggers**

You can create a trigger input that allows you to trigger the SignalTap II Logic Analyzer from an external source. The external trigger input behaves like trigger condition 1. It is evaluated and must be true before any other configured trigger conditions are evaluated. The analyzer can also supply a signal to trigger external devices or other SignalTap II instances. These features allow you to synchronize external logic analysis equipment with the internal logic analyzer. Power-Up Triggers can use the external triggers feature, but they must use the same source or target signal as their associated Run-Time Trigger.

**Trigger In**

To use Trigger In, perform the following steps:

1. In the SignalTap II Editor, click the **Setup** tab.

2. If a Power-Up Trigger is enabled, make sure you are viewing the Runtime Trigger conditions.

3. In the **Signal Configuration** pane, turn on **Trigger In**.

4. In the **Pattern** list, select the condition you want to act as your trigger event. You can set this separately for a Runtime or a Power-Up Trigger.

5. Click **Browse** next to the **Source** field in the Trigger In pane (Figure 13–22 on page 13–50). The **Node Finder** dialog box appears.
6. In the **Node Finder** dialog box, select the signal (either an input pin or an internal signal) that you want to drive the Trigger In source, and click **OK**.

   If you type a new signal name in the **Source** field, you create a new node that you can assign to an input pin in the Pin Planner or Assignment editor. If you leave the **Source** field blank, a default name is entered in the form `auto_stp_trigger_in_<SignalTap instance number>`.

**Trigger Out**

To use Trigger Out, perform the following steps:

1. In the SignalTap II Editor, click the **Setup** tab.

2. If a Power-Up trigger is enabled, make sure you are viewing the Runtime Trigger conditions.

3. In the **Signal Configuration** pane, turn on **Trigger Out** (refer to Figure 13–21 on page 13–49).

4. In the **Level** list, select the condition you want to signify that the trigger event is occurring. You can set this separately for a Run-Time or a Power-Up Trigger.

5. Type a new signal name in the **Target** field. A new node name is created that you must assign to an output pin in the Pin Planner or Assignment editor.

   If you leave the **Target** field blank, a default name is entered in the form `auto_stp_trigger_out_<SignalTap instance number>`. When the logic analyzer triggers, a signal at the level you indicated will be output on the pin you assigned to the new node.

**Using the Trigger Out of One Analyzer as the Trigger In of Another Analyzer**

An advanced feature of the SignalTap II Logic Analyzer is the ability to use the Trigger Out of one analyzer as the Trigger In to another analyzer. This feature allows you to synchronize and debug events that occur across multiple clock domains.
To perform this operation, first enable the **Trigger Out** of the source logic analyzer instance. On the Trigger out **Target** list, select the targeted logic analyzer instance. For example, if the instance named `auto_signaltap_0` should trigger `auto_signaltap_1`, select `auto_signaltap_1|trigger_in` from the list (Figure 13–21).

**Figure 13–21. Configuring the Trigger Out Signal**

This automatically enables the Trigger In of the targeted logic analyzer instance and fills in the Trigger In **Source** field with the Trigger Out signal from the source logic analyzer instance. In this example, `auto_signaltap_0` is targeting `auto_signaltap_1`. The Trigger In **Source** field of `auto_signaltap_1` is automatically filled in with `auto_signaltap_0|trigger_out` (Figure 13–22).
When you add a SignalTap II file to your project, the SignalTap II Logic Analyzer becomes part of your design. You must compile your project to incorporate the SignalTap II logic and enable the JTAG connection that is used to control the logic analyzer. When you are debugging with a traditional external logic analyzer, it is often necessary to make changes to the signals monitored as well as the trigger conditions. Since these adjustments often translate into recompilation time when using the SignalTap II Logic Analyzer, you can use the SignalTap II Logic Analyzer feature along with incremental compilation in the Quartus II software to reduce time spent recompiling.
Faster Compilations with Quartus II Incremental Compilation

To use Incremental compilation with the SignalTap II Logic Analyzer, you must perform the following steps:

- Enable Full Incremental Compilation for your design
- Assign design partitions
- Set partitions to the proper preservation levels
- Enable SignalTap for your design
- Add signals to SignalTap using the appropriate netlist filter in the node finder (either SignalTap II: pre-synthesis or SignalTap II: post-fitting).

When you compile your design with a SignalTap II file, the sld_signaltap and sld_hub entities are automatically added to the compilation hierarchy. These two entities are the main components of the SignalTap II Logic Analyzer, providing the trigger logic and JTAG interface required for operation.

Incremental compilation enables you to preserve the synthesis and fitting results of your original design and add the SignalTap II Logic Analyzer to your design without recompiling your original source code. This feature is also useful when you want to modify the configuration of the SignalTap II file. For example, you can modify the buffer sample depth or memory type without performing a full compilation after the change is made. Only the SignalTap II Logic Analyzer, configured as its own design partition, must be recompiled to reflect the changes.

To use incremental compilation, you must first enable **Full Incremental Compilation** for your design if it is not already enabled, assign design partitions if necessary, and set the design partitions to the correct preservation levels. Incremental compilation is the default setting for new projects in the Quartus II software, so you can establish design partitions immediately in a new project. However, it is not necessary to create any design partitions to use the SignalTap II Incremental Compilation feature. Once your design is set up to use full incremental compilation, the SignalTap II Logic Analyzer acts as its own separate design partition. You can begin taking advantage of incremental compilation by using the **SignalTap II: post-fitting filter** in the **Node Finder** to add signals for logic analysis.

**Enabling Incremental Compilation for your Design**

To enable Incremental Compilation if it is not already enabled, perform the following steps:

1. On the Assignments menu, click **Design Partitions window**.
2. In the **Incremental Compilation** list, select **Full Incremental Compilation**.

3. Create user-defined partitions if desired and set the **Netlist Type** to **Post-fit** for all partitions. The netlist type for the top-level partition defaults to source. To take advantage of incremental compilation, you must set the Netlist types for the partitions you wish to tap as post-fit.

4. On the **Processing** menu, click **Start Compilation**, or click **Start Compilation** on the toolbar.

Your project is fully compiled the first time, establishing the design partitions you have created. When enabled for your design, the SignalTap II Logic Analyzer will always be a separate partition. After the first compilation, you can use the SignalTap II Logic Analyzer to analyze signals from the post-fit netlist. If your partitions are set correctly, subsequent compilations due to SignalTap II settings are able to take advantage of the shorter compilation times.

For more information about configuring and performing Incremental Compilation, refer to the *Quartus II Incremental Compilation for Hierarchical and Team-Based Design* chapter in volume 1 of the *Quartus II Handbook*.

**Using Incremental Compilation with the SignalTap II Logic Analyzer**

The SignalTap II Logic Analyzer is automatically configured to work with the incremental compilation flow. For all signals that you want to connect to the SignalTap II Logic Analyzer from the post-fit netlist, set the netlist type of the partition containing the desired signals to Post-Fit or Post-Fit (Strict) with a Fitter Preservation Level of Placement and Routing using the Design Partitions window. Use the **SignalTap II: post-fitting filter** in the **Node Finder** to add the signals of interest to your SignalTap II configuration file. If you want to add signals from the pre-synthesis netlist, set the netlist type to Source File and use the **SignalTap II: pre-synthesis filter** in the **Node Finder**. Do not use the netlist type **Post-Synthesis** with the SignalTap II Logic Analyzer.

**CAUTION**

Be sure to conform to the following guidelines when using post-fit/pre-synthesis nodes:

- Read all incremental compilation guidelines to ensure the proper partition of a project.
- To speed compile time, use only post-fit nodes for partitions set to preservation level post-fit.
Compile the Design

- Do not mix pre-synthesis and post-fit nodes in any partition. If you must tap presynthesis nodes for a particular partition, make all tapped nodes in that partition presynthesis nodes and change the netlist type to source in the design partitions window.

Node names may be different between a pre-synthesis netlist and a post-fit netlist. In general, registers and user input signals share common names between the two netlists. During compilation, certain optimizations will change the names of combinational signals in your RTL. If the type of node name chosen does not match the netlist type, the compiler may not be able to find the signal to connect to your SignalTap II Logic Analyzer instance for analysis. The compiler will issue a critical warning to warn you of this scenario. The signal that is not connected is tied to ground in the SignalTap II data tab.

If you do use incremental compile flow with the SignalTap II Logic Analyzer and source file changes are necessary, be aware that you may have to remove compiler-generated post-fit net names. Source code changes force the affected partition to go through a resynthesis. During synthesis, the compiler cannot find compiler-generated net names from a previous compilation. Altera recommends you use only registered and user-supplied input signals as debugging taps in your STP file whenever possible. Both registered and user-supplied input signals share common node names in the pre-synthesis and post-fit netlist. As a result, using only registered and user-supplied input signals in your STP file limits the changes you need to make to your SignalTap configuration.

To verify that your original design was not modified, examine the messages in the Partition Merge section of the Compilation Report. Figure 13–23 shows an example of the messages displayed.
 Unless you make changes to your design partitions that require recompilation, only the SignalTap II design partition is recompiled. If you make subsequent changes to only the SignalTap II file, only the SignalTap II design partition must be recompiled, reducing your recompilation time.

### Preventing Changes Requiring Recompilation

You can configure the SignalTap II file to prevent changes that normally require a recompilation. You do this by selecting a lock mode from above the node list in the **Setup** tab. Whether or not you are using incremental compilation, you can lock your configuration by choosing to allow only trigger condition changes.

For more information about the use of lock modes, refer to the Quartus II Help.

### Timing Preservation with the SignalTap II Logic Analyzer

In addition to verifying functionality, timing closure is one of the most crucial processes in successfully completing a design. When you compile a project with a SignalTap II Logic Analyzer without the use of incremental compilation, you add IP to your existing design. Therefore, you can affect the existing placement, routing, and timing of your design. To minimize the effect that the SignalTap II Logic Analyzer has on your design, Altera recommends that you use incremental compilation for
your project. Incremental compilation is the default setting in new designs and can be easily enabled and configured in existing designs. With the SignalTap II Logic Analyzer in its own design partition, it has little to no affect on your design.

In addition to using the incremental compilation flow for your design, you can use the following techniques to help maintain timing:

- Avoid adding critical path signals to your SignalTap II file.
- Minimize the number of combinational signals you add to your SignalTap II file, and add registers whenever possible.
- Specify an \( f_{\text{MAX}} \) constraint for each clock in your design.

For an example of timing preservation with the SignalTap II Logic Analyzer, refer to the *Area and Timing Optimization* chapter in volume 2 of the *Quartus II Handbook*.

**Performance and Resource Considerations**

There is an inherent trade-off between runtime flexibility of the SignalTap II Logic Analyzer, timing performance of the Signal Tap II Logic Analyzer, and the resource usage. The SignalTap II Logic Analyzer allows you to select the runtime configurable parameters to balance the need for runtime flexibility, speed, and area. The default values have been chosen to provide maximum flexibility so you can reach debugging closure as quickly as possible; however, you can adjust these settings to determine whether there is a more optimal configuration for your design.

The suggestions in this section provide some tips to provide extra timing slack if you have determined that the SignalTap II logic is in your critical path, or to alleviate the resource requirements that the SignalTap II Logic Analyzer consumes if your design is resource-constrained.

If the SignalTap II logic is part of your critical path, the following suggestions can help to speed up the performance of the SignalTap II Logic Analyzer:

- **Disable runtime configurable options**—Certain resources are allocated to accommodate for run-time flexibility. If you are using either advanced triggers or the state-based triggering flow, you can disable run-time configurable parameters for a boost in \( f_{\text{MAX}} \) of the SignalTap II logic. If you are using the state-based triggering flow, try disabling the *Goto state destination* option and performing a recompilation before disabling the other runtime configurable options. The *Goto state destination* option has the greatest impact on \( f_{\text{MAX}} \) as compared to the other runtime configurable options.
Minimize the number of signals that have Trigger Enable selected—All of the signals that you add to the SignalTap II file have Trigger Enable turned on. Turn off Trigger Enable for signals that you do not plan to use as triggers.

Turn on Physical Synthesis for register retiming—If you have a large number of triggering signals enabled (greater than the number of inputs that would fit in a LAB) that fan-in to logic gate-based triggering condition, such as a basic trigger condition or a logical reduction operator in the advanced trigger tab, turn on the Perform register retiming. This can help balance combinational logic across LABs.

If your design is resource constrained, the following suggestions can help to reduce the amount of logic or memory used by the SignalTap II Logic Analyzer:

Disable runtime configurable options—Disabling runtime configurability for the advanced trigger conditions or the runtime configurable options in the state-based triggering flow will result in less LE usage.

Minimize the number of segments in the acquisition buffer—You can reduce the number of logic resources used for the SignalTap II Logic Analyzer by limiting the number of segments in your sampling buffer to only that which is required.

Disable the Data Enable for signals that are used for triggering only—By default, both the data enable and trigger enable options are selected for all signals. Turning off the data enable option for signals used as trigger inputs only will save on memory resources used by the SignalTap II Logic Analyzer.

Because performance results are design-dependent, try these options in different combinations until you achieve the desired balance between functionality, performance, and utilization.

For more information about area and timing optimization, refer the Area and Timing Optimization chapter in volume 2 of the Quartus II Handbook.
Program the Target Device or Devices

Once your project, including the SignalTap II Logic Analyzer, is compiled, you must configure the FPGA target device. When you are using the SignalTap II Logic Analyzer for debugging, you can configure the device from the SignalTap II file instead of the Quartus II Programmer. Because you configure from the SignalTap II file, you can open more than one SignalTap II file and program multiple devices to debug multiple designs simultaneously.

The settings in a SignalTap II file must be compatible with the programming (SOF) file used to program the device. A SignalTap II file is considered compatible with an SOF when the settings for the logic analyzer, such as the size of the capture buffer and the signals selected for monitoring or triggering, match the way the target device will be programmed. If the files are not compatible, you will still be able to program the device, but you will not be able to run or control the logic analyzer from the SignalTap II Editor.

To ensure programming compatibility, make sure to program your device with the latest SOF created from the most recent compilation.

Before starting a debugging session, do not make any changes to the SignalTap II file settings that would require the project to be recompiled. You can check the SignalTap II status display at the top of the Instance Manager to see if a change you made requires the design to be recompiled, producing a new SOF. This gives you the opportunity to undo the change, so that a recompilation is not necessary. To prevent any such changes, enable a lock mode in the SignalTap II file.

Programming a Single Device

To configure a single device for use with the SignalTap II Logic Analyzer, perform the follow steps:

1. In the JTAG Chain Configuration pane in the SignalTap II Editor, select the connection you use to communicate with the device from the Hardware list. If you need to add your communication cable to the list, click Setup to configure your connection.

2. Click Browse in the JTAG Chain Configuration pane, and select the SOF file that includes the compatible SignalTap II Logic Analyzer.

3. Click Scan Chain. The Scan Chain operation enumerates all of the JTAG devices within your JTAG chain.
4. In the **Device** list, select the device to which you want to download the design. The device list shows an ordered list of all devices in the JTAG chain.

   All of the devices are numbered sequentially according to their position in the JTAG chain, prefixed with the “@”. For example: 
   @1 : EP3C25 (0x020F30DD) lists a Cyclone III device as the first device in the chain, with the JTAG ID code of 0x020F30DD.

5. Click the **Program Device** icon.

**Programming Multiple Devices to Debug Multiple Designs**

You can simultaneously debug multiple designs using one instance of the Quartus II software by performing the following steps:

1. Create, configure, and compile each project that includes a SignalTap II file.

2. Open each SignalTap II file.

   You do not have to open a Quartus II project to open a SignalTap II file.

3. Use the **JTAG Chain Configuration** pane controls to select the target device in each SignalTap II file.

4. Program each FPGA.

5. Run each analyzer independently.

**Figure 13–24** shows a JTAG chain and its associated SignalTap II files.
Run the SignalTap II Logic Analyzer

After the device is configured with your design that includes the SignalTap II Logic Analyzer, you can perform debugging operations in a manner similar to the use of an external logic analyzer. You “arm” the logic analyzer by starting an analysis. When your trigger event occurs, the captured data is stored in the memory buffer on the device and then transferred to the SignalTap II file over the JTAG connection. You can also perform the equivalent of a “force trigger” that lets you view the captured data currently in the buffer without a trigger event occurring. Figure 13–25 illustrates a flow that shows how you operate the SignalTap II Logic Analyzer. The flowchart indicates where Power-Up and Run-Time Trigger events occur and when captured data from these events is available for analysis.

Figure 13–25. Power-Up and runtime Trigger Events Flowchart
The SignalTap II toolbar in the Instance Manager has four options for running the analyzer:

- **Run Analysis**—The SignalTap II Logic Analyzer runs until the trigger event occurs. When the trigger event occurs, monitoring and data capture stops once the acquisition buffer is full.
- **AutoRun Analysis**—The SignalTap II Logic Analyzer continuously captures data until the **Stop Analysis** button is clicked, ignoring all trigger event conditions.
- **Stop Analysis**—SignalTap II analysis stops. The acquired data does not appear automatically if the trigger event has not occurred.
- **Read Data**—Captured data is displayed. This button is useful if you want to view the acquired data even if the trigger has not occurred.

**Running with a Power-Up Trigger**

If you have enabled and set up a Power-Up Trigger for an instance of the SignalTap II Logic Analyzer, the captured data may already be available for viewing if the trigger event occurred after device configuration. To download the captured data or to check if the Power-Up Trigger is still running, click **Run Analysis** in the Instance Manager. If the Power-Up Trigger occurred, the logic analyzer immediately stops, and the captured data is downloaded from the device. The data can now be viewed on the **Data** tab of the SignalTap II Editor. If the Power-Up Trigger did not occur, no captured data is downloaded, and the logic analyzer continues to run. You can wait for the Power-Up Trigger event to occur, or, to stop the logic analyzer, click **Stop Analysis**.

**Running with Runtime Triggers**

You can arm and run the SignalTap II Logic Analyzer manually after device configuration to capture data samples based on the Runtime Trigger. You can do this immediately if there is no Power-Up Trigger enabled. If a Power-Up Trigger is enabled, you can do this after the Power-Up Trigger data is downloaded from the device or once the logic analyzer is stopped because the Power-Up Trigger event did not occur. Click **Run Analysis** in the SignalTap II Editor to start monitoring for the trigger event. You can start multiple SignalTap II instances at the same time by selecting all of the required instances before you click **Run Analysis** on the toolbar.

Unless the logic analyzer is stopped manually, data capture begins when the trigger event evaluates to **true**. When this happens, the captured data is downloaded from the buffer. You can view the data in the **Data** tab of the SignalTap II Editor.
Performing a Force Trigger

Sometimes when you use an external logic analyzer or oscilloscope, you want to see the current state of signals without setting up or waiting for a trigger event to occur. This is referred to as a “force trigger” operation, because you are forcing the test equipment to capture data without regard to any set trigger conditions. With the SignalTap II Logic Analyzer, you can choose to run the analyzer and capture data immediately or run the analyzer and capture data when you want.

For more information, refer to the Design Debugging Using In-System Sources and Probes chapter in volume 3 of the Quartus II Handbook.

To run the analyzer and immediately capture data, disable the trigger conditions by turning off each Trigger Condition column in the node list. This operation does not require a recompilation. Click Run Analysis in the Instance Manager. The SignalTap II Logic Analyzer immediately triggers, captures, and downloads the data to the Data tab of the SignalTap II Editor. If the data does not download automatically, click Read Data in the Instance Manager.

If you want to choose when to capture data manually, it is not required that you disable the trigger conditions. Click Autorun Analysis to start the logic analyzer, and click Stop Analysis to capture data. If the data does not download to the Data tab of the SignalTap II Editor automatically, click Read Data.

Finally, you can choose to capture data manually after a trigger event has occurred. This is useful if you still want the trigger event to occur, but you want to capture data about the signals at some point after the trigger without capturing the trigger event itself. To do this, set the Buffer acquisition mode to Circular and Continuous, and click Run Analysis. When the trigger event occurs, the status in the SignalTap II Health Monitor is shown as Acquiring post-trigger data, but the logic analyzer does not stop. When you want to capture and download the data, click Stop Analysis. If the data does not download automatically, click Read Data.

You can also use In-System Sources and Probes in conjunction with the SignalTap II Logic Analyzer to force trigger conditions. The In-System Sources and Probes feature allows you to drive and sample values on to selected nets over the JTAG chain. For more information, refer to the Design Debugging Using In-System Sources and Probes chapter in volume 3 of the Quartus II Handbook.
SignalTap II Status Messages

Table 13–11 describes the text messages that may appear in the SignalTap II Health Monitor in the Instance Manager before, during, and after a data acquisition. Use these messages to know the state of the logic analyzer or what operation it is performing.

<table>
<thead>
<tr>
<th>Message</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not running</td>
<td>The SignalTap II Logic Analyzer is not running. There is no connection to a device or the device is not configured.</td>
</tr>
<tr>
<td>(Power-Up Trigger) Waiting for clock (1)</td>
<td>The SignalTap II Logic Analyzer is performing a Runtime or Power-Up Trigger acquisition and is waiting for the clock signal to transition.</td>
</tr>
<tr>
<td>Acquiring (Power-Up) pre-trigger data (1)</td>
<td>The trigger condition has not been evaluated yet. A full buffer of data is collected if the circular buffer acquisition mode is selected.</td>
</tr>
<tr>
<td>Trigger In conditions met</td>
<td>Trigger In condition has occurred. The SignalTap II Logic Analyzer is waiting for the condition of the first trigger condition to occur. This can appear if Trigger In is specified.</td>
</tr>
<tr>
<td>Waiting for (Power-up) trigger (1)</td>
<td>The SignalTap II Logic Analyzer is now waiting for the trigger event to occur.</td>
</tr>
<tr>
<td>Trigger level $x$ met</td>
<td>The condition of trigger condition $x$ has occurred. The SignalTap II Logic Analyzer is waiting for the condition specified in condition $x+1$ to occur.</td>
</tr>
<tr>
<td>Acquiring (power-up) post-trigger data (1)</td>
<td>The whole trigger event has occurred. The SignalTap II Logic Analyzer is acquiring the post-trigger data. The amount of post-trigger data collected is user-defined between 12%, 50%, and 88% when the circular buffer acquisition mode is selected.</td>
</tr>
<tr>
<td>Offload acquired (Power-Up) data (1)</td>
<td>Data is being transmitted to the Quartus II software through the JTAG chain.</td>
</tr>
<tr>
<td>Ready to acquire</td>
<td>The SignalTap II Logic Analyzer is waiting for the user to arm the analyzer.</td>
</tr>
</tbody>
</table>

Note to Table 13–11:
(1) This message can appear for both Runtime and Power-Up Trigger events. When referring to a Power-Up Trigger, the text in parentheses is added.

In segmented acquisition mode, pre-trigger and post-trigger do not apply.
View, Analyze, and Use Captured Data

Once a trigger event has occurred or you capture data manually, you can use the SignalTap II interface to examine the data, and use your findings to help debug your design. The SignalTap II Logic Analyzer provides a number of features that makes it easy to do this.

Viewing Captured Data

You can view captured SignalTap II data in the Data tab of the SignalTap II file (Figure 13–26). Each row of the Data tab displays the captured data for one signal or bus. Buses can be expanded to show the data for each individual signal on the bus. Click on the data waveforms to zoom in on the captured data samples, and right-click to zoom out.

Figure 13–26. Captured SignalTap II Data

When you are viewing captured data, it is often useful to know the time interval between two events. Time bars enable you to see the number of clock cycles between two samples of captured data in your system. There are two types of time bars:

- **Master Time Bar**—The master time bar’s label displays the absolute time of its location in bold. The master time bar is a thick black line in the Data tab. The captured data has only one master time bar.

- **Reference Time Bar**—The reference time bar’s label displays the time relative to the master time bar. You can create an unlimited number of reference time bars.

To help you find a transition of signals relative to the master time bar location, use either the Next Transition or the Previous Transition button. This aligns the master time bar with the next or previous
transition of a selected signal or group of selected signals. This feature is very useful when the sample depth is very large and the rate at which signals toggle is very low.

Creating Mnemonics for Bit Patterns

The mnemonic table feature allows you to assign a meaningful name to a set of bit patterns, such as a bus. To create a mnemonic table, right-click in the Setup or Data tab of a SignalTap II file, and click Mnemonic Table Setup. You create a mnemonic table by entering sets of bit patterns and specifying a label to represent each pattern. Once you have created a mnemonic table, you assign it to a group of signals. To assign a mnemonic table, right-click on the group, click Bus Display Format, and select the desired mnemonic table.

The labels you create in a table are used in different ways on the Setup and Data tabs. On the Setup tab, you can create basic triggers with meaningful names by right-clicking an entry in any Trigger Conditions column and selecting a label from the table you assigned to the signal group. On the Data tab, if any captured data matches a bit pattern contained in an assigned mnemonic table, the signal group data is replaced with the appropriate label, making it easy to see when expected data patterns occur.

Automatic Mnemonics with a Plug-In

When you use a plug-in to add signals to a SignalTap II file, mnemonic tables for the added signals are automatically created and assigned to the signals defined in the plug-in. If you ever need to manually enable these mnemonic tables, right-click on the name of the signal or signal group. On the Bus Display Format submenu, click the name of the mnemonic table that matches the plug-in.

As an example, the Nios II plug-in makes it easy to monitor your design’s signal activity as code is executed. If you have set up the logic analyzer to trigger on a function name in your Nios II code based on data from an ELF file, you can see the function name in the Instance Address signal group at the trigger sample, along with the corresponding disassembled code in the Disassembly signal group, as shown in Figure 13–27 on page 13–65. Captured data samples around the trigger are referenced as offset addresses from the trigger function name.
Locating a Node in the Design

When you find the source of a bug in your design using the SignalTap II Logic Analyzer, you can use the node locate feature to locate that signal in many of the tools found in the Quartus II software, as well as in your design files. This lets you find the source of the problem quickly so you can modify your design to correct the flaw. To locate a signal from the SignalTap II Logic Analyzer in one of the Quartus II software tools or your design files, right-click on the signal in the SignalTap II file, and click Locate in <tool name>. You can locate a signal from the node list in any of the following locations:

- Assignment Editor
- Pin Planner
- Timing Closure Floorplan
- Chip Planner
- Resource Property Editor
- Technology Map Viewer
- RTL Viewer
- Design File

For more information about using these tools, refer to the appropriate chapters in the Quartus II Handbook.
Saving Captured Data

The data log shows the history of captured data and the triggers used to capture the data. The analyzer acquires data, stores it in a log, and displays it as waveforms. When the logic analyzer is in auto-run mode and a trigger event occurs more than once, captured data for each time the trigger occurred is stored as a separate entry in the data log. This makes it easy to go back and review the captured data for each trigger event. The default name for a log is based on the time when the data was acquired. Altera recommends that you rename the data log with a more meaningful name.

The logs are organized in a hierarchical manner; similar logs of captured data are grouped together in trigger sets. If the Data Log pane is closed, on the View menu, select Data Log to reopen it. To enable data logging, turn on Enable data log in the Data Log (Figure 13–11). To recall a data log for a given trigger set and make it active, double-click the name of the data log in the list.

The Data Log feature is useful for organizing different sets of trigger conditions and different sets of signal configurations. Refer to “Managing Multiple SignalTap II Files and Configurations” on page 13–28.

Converting Captured Data to Other File Formats

You can export captured data in the following file formats, some of which can be used with other EDA simulation tools:

- Comma Separated Values File (.csv)
- Table File (.tbl)
- Value Change Dump File (.vcd)
- Vector Waveform File (.vwf)
- Graphics format files (.jpg, .bmp)

To export the SignalTap II Logic Analyzer’s captured data, on the File menu, click Export and specify the File Name, the Export Format, and the Clock Period.
Creating a SignalTap II List File

Captured data can also be viewed in a SignalTap II list file. A SignalTap II list file is a text file that lists all the data captured by the logic analyzer for a trigger event. Each row of the list file corresponds to one captured sample in the buffer. Columns correspond to the value of each of the captured signals or signal groups for that sample. If a mnemonic table was created for the captured data, the numerical values in the list are replaced with a matching entry from the table. This is especially useful with the use of a plug-in that includes instruction code disassembly. You can immediately see the order in which the instruction code was executed during the same time period of the trigger event. To create a SignalTap II list file, on the File menu, select **Create/Update**, and click **Create SignalTap II List File**.

Other Features

The SignalTap II Logic Analyzer has a number of other features that do not necessarily belong to a particular task in the task flow.

Using the SignalTap II MATLAB MEX Function to Capture Data

If you use MATLAB for DSP design, you can call the MATLAB MEX function `alt_signaltap_run`, built into the Quartus II software, to acquire data from the SignalTap II Logic Analyzer directly into a matrix in the MATLAB environment. If you use the MEX function repeatedly in a loop, you can perform as many acquisitions as you can when using SignalTap II in the Quartus II software environment in the same amount of time.

The SignalTap II MATLAB MEX function is available only in the Windows version of the Quartus II software. It is compatible with MATLAB Release 14 Original Release Version 7 and later.

To set up the Quartus II software and the MATLAB environment to perform SignalTap II acquisitions, perform the following steps:

1. In the Quartus II software, create a SignalTap II file.

2. In the node list in the **Data** tab of the SignalTap II Editor, organize the signals and groups of signals into the order in which you want them to appear in the MATLAB matrix. Each column of the imported matrix represents a single SignalTap II acquisition sample, while each row represents a signal or group of signals in the order they are organized in the **Data** tab.
Signal groups acquired from the SignalTap II Logic Analyzer and transferred into the MATLAB environment with the MEX function are limited to a width of 32 signals. If you want to use the MEX function with a bus or signal group that contains more than 32 signals, split the group up into smaller groups that do not exceed the 32 signal limit.

3. Save the SignalTap II file and compile your design. Program your device and run the SignalTap II Logic Analyzer to make sure your trigger conditions and signal acquisition are working correctly.

4. In the MATLAB environment, add the Quartus II binary directory to your path with the following command:

    `addpath <Quartus install directory>\win`

You can view the help file for the MEX function by entering `alt_signaltap_run` in MATLAB without any operators.

You use the MEX function in the MATLAB environment to open the JTAG connection to the device and run the SignalTap II Logic Analyzer to acquire data. When you finish acquiring data, you must close the connection.

To open the JTAG connection and begin acquiring captured data directly into a MATLAB matrix called `stp`, use the following command:

    `stp = alt_signaltap_run('<stp filename>[,('signed'\'unsigned')[,<instance names>[,/<signalset name>[,'<trigger name>'][,])]])';`

When capturing data, `<stp filename>` is the name of the SignalTap II file you want to use. This is required for using the MEX function. The other MEX function options are defined in Table 13–12.

### Table 13–12. SignalTap II MATLAB MEX Function Options (Part 1 of 2)

<table>
<thead>
<tr>
<th>Option</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed</td>
<td>'signed'</td>
<td>The signed option turns signal group data into 32-bit two's complement signed integers. The most significant bit (MSB) of the group as defined in the SignalTap II Data tab is the sign bit. The unsigned option keeps the data as an unsigned integer. The default is signed.</td>
</tr>
<tr>
<td>unsigned</td>
<td>'unsigned'</td>
<td></td>
</tr>
</tbody>
</table>
You can enable or disable verbose mode to see the status of the logic analyzer while it is acquiring data. To enable or disable verbose mode, use the following commands:

```matlab
alt_signaltap_run('VERBOSE_ON');
alt_signaltap_run('VERBOSE_OFF');
```

When you finish acquiring data, you must close the JTAG connection. Use the following command to close the connection:

```matlab
alt_signaltap_run('END_CONNECTION');
```

For more information about the use of MEX functions in MATLAB, refer to the MATLAB Help.

### Using SignalTap II in a Lab Environment

You can install a stand-alone version of the SignalTap II Logic Analyzer. This version is particularly useful in a lab environment where you do not have a workstation that meets the requirements for a complete Quartus II installation, or if you do not have a license for a full installation of the Quartus II software. The stand-alone version of the SignalTap II Logic Analyzer is included with the Quartus II stand-alone Programmer and is available from the Downloads page of the Altera website, [www.altera.com](http://www.altera.com).

### Remote Debugging Using the SignalTap II Logic Analyzer

You can use the SignalTap II Logic Analyzer to debug a design that is running on a device attached to a PC in a remote location.
To perform a remote debugging session, you must have the following setup:

- The Quartus II software installed on the local PC
- Stand-alone SignalTap II Logic Analyzer or the full version of the Quartus II software installed on the remote PC
- Programming hardware connected to the device on the PCB at the remote location
- TCP/IP protocol connection

**Equipment Setup**

On the PC in the remote location, install the stand-alone version of the SignalTap II Logic Analyzer or the full version of the Quartus II software. This remote computer must have Altera programming hardware connected, such as the EthernetBlaster or USB-Blaster.

On the local PC, install the full version of the Quartus II software. This local PC must be connected to the remote PC across a LAN with the TCP/IP protocol.

**Software Setup on the Remote PC**

To setup the software on the remote PC, perform the following steps:

1. In the Quartus II programmer, click **Hardware Setup**.
2. Click the **JTAG Settings** tab (Figure 13–28 on page 13–70).

![Configure JTAG on Remote PC](image)
3. Click Configure local JTAG Server.

4. In the Configure Local JTAG Server dialog box (Figure 13–29), turn on Enable remote clients to connect to the local JTAG server, and type your password in the password box. Type your password again in the Confirm Password box and click OK.

**Figure 13–29. Configure Local JTAG Server on Remote**

---

**Software Setup on the Local PC**

To set up your software on your local PC, perform the following steps:

1. Launch the Quartus II programmer.

2. Click Hardware Setup.

3. On the JTAG settings tab, click Add server.

4. In the Add Server dialog box (Figure 13–30), type the network name or IP address of the server you want to use and the password for the JTAG server that you created on the remote PC.

**Figure 13–30. Add Server Dialog Box**

---

5. Click OK.
SignalTap II Setup on the Local PC

To connect to the hardware on the remote PC, perform the following steps:

1. Click the Hardware Settings tab and select the hardware on the remote PC (Figure 13–31).

![Figure 13–31. Selecting Hardware on Remote PC](image)

2. Click Close.

You can now control the logic analyzer on the device attached to the remote PC as if it was connected directly to the local PC.

SignalTap II Scripting Support

You can run procedures and make settings described in this chapter in a Tcl script. You can also run some procedures at a command prompt. For detailed information about scripting command options, refer to the Quartus II Command-Line and Tcl API Help browser. To run the Help browser, type the following command at the command prompt:

```sh
quartus_sh --qhelp
```

The Quartus II Scripting Reference Manual includes the same information in PDF format.
SignalTap II Scripting Support

For more information about Tcl scripting, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*.

For more information about command-line scripting, refer to the *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook*.

**SignalTap II Command Line Options**

To compile your design with the SignalTap II Logic Analyzer using the command prompt, you must use the `quartus_stp` command. Table 13–13 shows the options that help you better understand how to use the `quartus_stp` executable.

<table>
<thead>
<tr>
<th>Option</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stp_file</td>
<td><code>quartus_stp --stp_file &lt;stp_filename&gt;</code></td>
<td>Assigns the specified SignalTap II file to the USE_SIGNALTAP_FILE in the Quartus II Settings File (QSF).</td>
</tr>
<tr>
<td>enable</td>
<td><code>quartus_stp --enable</code></td>
<td>Creates assignments to the specified SignalTap II file in the QSF, and changes ENABLE_SIGNALTAP to ON. The SignalTap II Logic Analyzer is included in your design the next time the project is compiled. If no SignalTap II file is specified in the QSF, the <code>--stp_file</code> option must be used. If the <code>--enable</code> option is omitted, the current value of ENABLE_SIGNALTAP in the QSF is used.</td>
</tr>
</tbody>
</table>
Example 13–2 illustrates how to compile a design with the SignalTap II Logic Analyzer at the command line:

**Example 13–2.**

```bash
quartus_stp filtref --stp_file stp1.stp --enable
quartus_map filtref --source=filtref.bdf --family=CYCLONE
quartus_fit filtref --part=EP1C12Q240C6 --fmax=80MHz --tsu=8ns
quartus_tan filtref
quartus_asm filtref
```

The `quartus_stp --stp_file stp1.stp --enable` command creates the QSF variable and instructs the Quartus II software to compile the `stp1.stp` file with your design.
Example 13–3 shows how to create a new SignalTap II file after building the SignalTap II Logic Analyzer instance with the MegaWizard Plug-In Manager:

```
Example 13–3.
quartus_stp filtref --create_signaltap_hdl_file --stp_file stp1.stp
```

For information about the other command line executables and options refer to the Command-Line Scripting chapter in volume 2 of the Quartus II Handbook.

**SignalTap II Tcl Commands**

The `quartus_stp` executable supports a Tcl interface that allows you to capture data without running the Quartus II GUI. You cannot execute SignalTap II Tcl commands from within the Tcl console in the GUI. They must be run from the command line with the `quartus_stp` executable. To run a Tcl file that has SignalTap II Tcl commands, use the following command:

```
quartus_stp -t <Tcl file>
```

Table 13–14 shows the Tcl commands that you can use with SignalTap II.

<table>
<thead>
<tr>
<th>Command</th>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open_session</td>
<td>-name &lt;stp_filename&gt;</td>
<td>Opens the specified SignalTap II file. All captured data is stored in this file.</td>
</tr>
<tr>
<td>run</td>
<td>-instance &lt;instance_name&gt;</td>
<td>Starts the analyzer. This command must be followed by all the required arguments to properly start the analyzer. You can optionally specify the name of the data log you want to create. If the Trigger condition is not met, you can specify a timeout value to stop the analyzer.</td>
</tr>
<tr>
<td></td>
<td>-signal_set &lt;signal_set&gt;</td>
<td>(optional)</td>
</tr>
<tr>
<td></td>
<td>-trigger &lt;trigger_name&gt;</td>
<td>(optional)</td>
</tr>
<tr>
<td></td>
<td>-data_log &lt;data_log_name&gt;</td>
<td>(optional)</td>
</tr>
<tr>
<td></td>
<td>-timeout &lt;seconds&gt;</td>
<td>(optional)</td>
</tr>
</tbody>
</table>
For more information about SignalTap II Tcl commands, refer to the Quartus II Help.

Example 13–4 is an excerpt from a script that is used to continuously capture data. Once the trigger condition is met, the data is captured and stored in the data log.

**Example 13–4.**

```tcl
# opens signaltap session
open_session -name stp1.stp
# start acquisition of instance auto_signaltap_0 and auto_signaltap_1 at the same time
# calling run_multiple_end will start all instances
# run after run_multiple_start call
run_multiple_start
run -instance auto_signaltap_0 -signal_set signal_set_1 -trigger /trigger_1 -data_log log_1 -timeout 5
run -instance auto_signaltap_1 -signal_set signal_set_1 -trigger /trigger_1 -data_log log_1 -timeout 5
run_multiple_end
# close signaltap session
close_session
```
Once the script is completed, you should open the SignalTap II file that you used to capture data to examine the contents of the Data log.

Design Example: Using SignalTap II Logic Analyzers in SOPC Builder Systems

Altera Application Note, AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems describes how to use the SignalTap II Logic Analyzer to monitor signals located inside a system module generated by SOPC Builder. The system in this example contains many components, including a Nios processor, a direct memory access (DMA) controller, on-chip memory, and an interface to external SDRAM memory. In this example, the Nios processor executes a simple C program from on-chip memory and waits for a button push. After a button is pushed, the processor initiates a DMA transfer, which you analyze using the SignalTap II Logic Analyzer.

For more information about this example and using the SignalTap II Logic Analyzer with SOPC builder systems refer to AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder Systems and AN 446: Debugging NIOS II Systems with the SignalTap II Logic Analyzer.

Custom Triggering Flow Application Examples

The custom triggering flow in the SignalTap II Logic Analyzer is most useful for organizing a number of triggering conditions and for precise control over the acquisition buffer. This section provides two application examples for defining a custom triggering flow within the SignalTap II Logic Analyzer. Both examples can be easily copied and pasted directly into the state machine description box by using the state display mode All states in one window.

For additional triggering flow design examples, refer to the Quartus II On-Chip Debugging Support Resources page for on-chip debugging.

Design Example 1: Specifying a Custom Trigger Position

Actions to the acquisition buffer can accept an optional post-count argument. This post-count argument enables you to define a custom triggering position for each segment in the acquisition buffer. Example 13–5 shows an example that applies a trigger position to all segments in the acquisition buffer. The example describes a triggering flow for an acquisition buffer split into four segments. If each acquisition segment is 64 samples in depth, the trigger position for each buffer will be at sample #34. The acquisition stops after all four segments are filled once.
Example 13–5.

```c
if (c1 == 3 && condition1)
    trigger 30;
else if (condition1)
    begin
        segment_trigger 30;
        increment c1;
    end
```

Each segment acts as a circular buffer, that continuously updates the memory contents with the signal values. The last acquisition before stopping the buffer is the displayed on the data tab as the last sample number in the affected segment. The trigger position in the affected segment is then defined by \( N – post\ count\ fill \), where \( N \) is the number of samples per segment. Figure 13–32 illustrates the triggering position.

**Figure 13–32. Specifying a Custom Trigger Position**

Design Example 2: Trigger When triggercond1 Occurs Ten Times between triggercond2 and triggercond3

The custom trigger flow description is often useful to count a sequence of events before triggering the acquisition buffer. Example 13–6 on page 13–79 shows such a sample flow. This example uses three basic triggering conditions configured in the SignalTap II setup tab.
This example triggers the acquisition buffer when condition1 occurs after condition3 and occurs ten times prior to condition3. If condition3 occurs prior to ten repetitions of condition1, the state machine transitions to a permanent wait state.

Example 13–6.

state ST1:

if ( condition2  )
begin
    reset c1;
    goto ST2;
end

State ST2 : 
if ( condition1 )
    increment c1;

else if (condition3 && c1 < 10)
    goto ST3;

else if ( condition3 && c1 >= 10)
    trigger;

ST3:
    goto ST3;
Conclusion

As the FPGA industry continues to make technological advancements, outdated methodologies need to be replaced with new technologies that maximize productivity. The SignalTap II Logic Analyzer gives you the same benefits as a traditional logic analyzer, without the many shortcomings of a piece of dedicated test equipment. This version of the SignalTap II Logic Analyzer provides many new and innovative features that allow you to capture and analyze internal signals in your FPGA, allowing you to find the source of a design flaw in the shortest amount of time.

Referenced Documents

This chapter references the following documents:

- AN 323: Using SignalTap II Embedded Logic Analyzers in SOPC Builder System
- Area and Timing Optimization chapter in volume 2 of the Quartus II Handbook
- Command-Line Scripting chapter in volume 2 of the Quartus II Handbook
- I/O Management chapter in volume 2 of the Quartus II Handbook
- In-System Debugging Using External Logic Analyzers chapter in volume 3 of the Quartus II Handbook
- Quartus II Incremental Compilation for Hierarchical and Team-Based Design chapter in volume 1 of the Quartus II Handbook
- Quartus II Integrated Synthesis chapter in volume 1 of the Quartus II Handbook
- Quartus II Settings File Reference Manual
- Quick Design Debugging Using SignalProbe chapter in volume 3 of the Quartus II Handbook
- Tcl Scripting chapter in volume 2 of the Quartus II Handbook
Table 13–15 shows the revision history for this chapter.

### Table 13–15. Document Revision History

<table>
<thead>
<tr>
<th>Date and Document Version</th>
<th>Changes Made</th>
<th>Summary of Changes</th>
</tr>
</thead>
</table>
| October 2007 v.7.2.0       | Updated for the Quartus II software version 7.2:  
- Added new section: “Trigger Condition Flow Control” on page 13–34  
- Documented the new feature for State-machine-based triggering  
- Documented changes to “Using Incremental Compilation with the SignalTap II Logic Analyzer” on page 13–52  
- Added additional information about node tappability  
- Added section “Performance and Resource Considerations” on page 13–55, with information about performance and resource utilization considerations for the SignalTap II Logic Analyzer | Updated for the Quartus II software version 7.2 |
| May 2007 v7.1.0            | Added “Referenced Documents” on page 13–71, minor updates to address ADoQS issues. | — |
| March 2007 v7.0.0          | Added Cyclone III device support listed on page 13–4. | — |
| November 2006 v6.1.0       | Updated for the Quartus II software version 6.1:  
- Updated Figure 13-4, 13-11,13-16, 13-17, 13-18. Added new Figure 13-23.  
- Miscellaneous changes throughout.  
- Removed information about incremental routing (feature removed).  
- Added more detail about the use of incremental compilation.  
- Added more detail about the use of the Nios II plug-in.  
- Added more information about SignalTap II file/SOF compatibility.  
- Updated method for triggering one logic analyzer with another using trigger in/out. | Updated for the Quartus II software version 6.1. |
| May 2006 v6.0.0            | Updated for the Quartus II software version 6.0. | — |
| October 2005 v5.1.0        | Updated for the Quartus II software version 5.1. | — |
| May 2005 v5.0.0            | Updated information.  
- Updated figures.  
- New functionality for Quartus II software 5.0. | — |
| December 2004 v1.0         | Initial release. | — |