Reveal User Guide

March 2014

Type Conventions Used in This Document

Convention	Meaning or Use
Bold	Items in the user interface that you select or click. Text that you type into the user interface.
<*Italic*>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
`Courier`	Code examples. Messages, reports, and prompts from the software.
`...`	Omitted material in a line of code.
. . .	Omitted lines in code and report examples.
`[ ]`	Optional items in syntax descriptions. In bus specifications, the brackets are required.
`( )`	Grouped items in syntax descriptions.
`{ }`	Repeatable items in syntax descriptions.
|	A choice between items in syntax descriptions.

Contents

Introduction

The Reveal software included in Diamond is a next-generation FPGA on-chip debug tool. It offers several key usability benefits including the following features:

- Integration with the Diamond design flow
- One-button operation to insert debug logic into the design
- Simple flow for modifying the original design or the debug configuration
- Advanced triggering capabilities for more flexible dynamic triggers
- Improved logic analyzer waveform usability

You can use Reveal with all FPGAs and with MachXO and MachXO2 devices of 1200 or more LUTs. But Reveal is not supported with Platform Manager or Platform Manager 2.

This user guide contains all the necessary information for getting started with the Reveal software. It contains two primary sections: the Reveal Inserter and the Reveal Analyzer.

The Reveal Inserter section contains information on how to add debug information to your design. It also contains detailed information on how to use and set up the triggering architecture used in Reveal. The triggering architecture, which is based on trigger units and trigger expressions, has some differences from other systems but offers increased capabilities and flexibility for an internal logic analyzer.

The Reveal Analyzer section contains information on how to use and connect the software to the design running on the target hardware.

Using the Reveal Example Project

If you want to get some hands-on experience with the Reveal tools, try using the example project while studying the online help.

The example is a simple 3-bit counter coded in Verilog. It already has a Reveal module inserted. It also already has trace data that can be viewed in Reveal Analyzer. A test board is not required to open the Reveal tools or to view the existing trace data. But, to actually run Reveal Analyzer to collect new data, you need a test board.

To start the Reveal example project:

1. Choose **File > Open > Design Example**.

   The Open Example dialog box opens.

2. Double-click **counta_reveal_XP2**.
3. Choose **count.ldf**.
4. Click **Open**.

   The count design project opens. At this point, you can open Reveal Inserter.

   Note

   If you want to preserve the original example for future experiments, choose **File > Archive Project** now.

5. In the Process view, double-click **Export Files** to implement the design.

   At this point, you can open Reveal Analyzer.

If you want to use the example project with a test board, change the device type to match the board by double-clicking the device in the File List view. Then run the design implementation process.

Reveal Inserter

Reveal Inserter enables you to select which design signals to use for debug tracing or triggering, then generate a core on the basis of these signals and their use. After generating the required core, it generates a modified design with the necessary debug connections and links it to the signals. Reveal Inserter supports VHDL, Verilog, mixed-HDL, and EDIF flows for debug insertion. Once the design has been modified for debug, it is mapped, placed, and routed with the normal design flow in Diamond.

After you generate the bitstream or JEDEC file, Reveal Analyzer helps you debug your FPGA circuitry by giving you access to internal nodes inside the device so you can observe their behavior. It enables you to set and change various values and combinations of trigger signals. Once the specified trigger condition is reached, the data values of the trace signals are saved in the trace buffer. After the data is captured, it is transferred from the FPGA through the JTAG ports to the PC.

You can also set up modules that allow you to adjust and monitor serdes functions in ECP5UM designs.

About Reveal Inserter

This section introduces some of the key features of Reveal Inserter: the devices that it supports, the steps in its design flow, its inputs, its outputs, and its limitations.

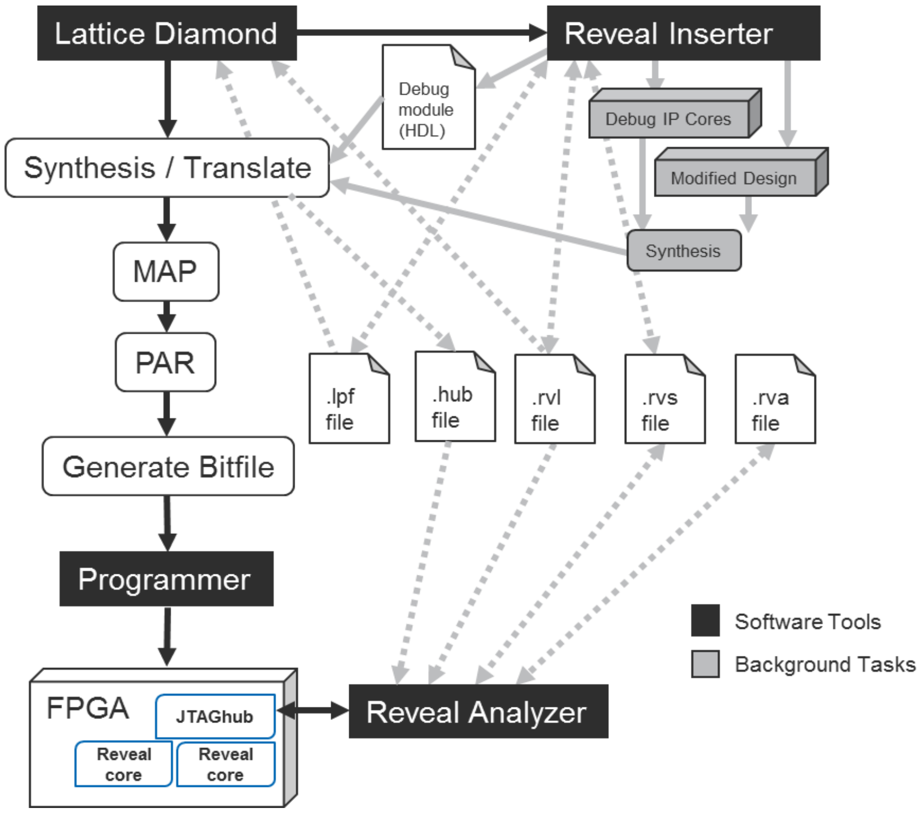
Reveal On-Chip Debug Design Flow

The Reveal Inserter design flow is shown in the following figure.

Note

Interactive synthesis is not compatible with the Reveal debugging flow. When you use Reveal, the interactive synthesis option is not available.

Figure 1: Reveal Inserter Design Flow



To generate and insert debug logic cores, follow these general steps:

1. Start Reveal Inserter.
2. Create a new Reveal Inserter project or open an existing Reveal Inserter project.
3. Add new cores to the project, if needed.
4. For each core, set up the trace signals in the Trace Signal Setup tab.
5. For each core, set up the trigger signals in the Trigger Signal Setup tab.
6. Insert the debug logic.

   This process generates and synthesizes the necessary debug logic.

   The generated .rvl is automatically imported into Diamond if you enabled the “Import Reveal file to Diamond project” option in the Insert Debug to Design dialog box.

7. Translate the design in Diamond.

   This process creates two or more .ngo files.

   When Reveal is used for your design, the Translate Design process performs two extra steps:

   - It builds a version of the design that contains the necessary connections for the debug logic to the signals that you are tracing or triggering.
   - It adds logic for communicating with the JTAG pins for Reveal Analyzer. Newer software and IP from Lattice Semiconductor support a mechanism to allow multiple elements to share the JTAG pins. However, if the design was built with earlier software or IP and contains a JTAG primitive, this logic conflicts with the Reveal flow and results in an error in the Translate Design process.

8. Map, place, and route the design.
9. Generate the bitstream data or JEDEC file.

   If you want to perform logic analysis with Reveal Analyzer, continue with these steps:

10. Set up the cable connection with Programmer.
11. Download the design onto the device.
12. Start Reveal Analyzer and perform logic analysis with it.

Inputs

The inputs to Reveal Inserter in the RTL flow are the following:

- VHDL, Verilog, and EDIF files
- A preference (.lpf) file, if it already exists

Outputs

Reveal Inserter generates the following files in the RTL flow:

- Reveal Inserter project (.rvl) file, which contains the signal connections for each core and some settings for the debugging logic, such as maximum sequence depth and maximum event counter. The information in this file is statically set in Reveal Inserter and cannot be changed in Reveal Analyzer.
- Reveal Inserter settings (.rvs) file, which contains settings that can be dynamically changed without regenerating the debug logic. This information includes trigger units, comparison types, values, and trigger expressions. The information in this file is dynamically set in either Reveal Analyzer or in both Reveal Analyzer and Reveal Inserter.
- Reveal Inserter parameter (.rvp) file, which contains information needed for debug logic generation, is produced during the design implementation process.

- JTAG hub (.hub) file produced by the Translate Design stage of the design implementation process. This file is used by Reveal Analyzer to sort the data coming from the different Reveal modules.

Reveal Inserter also modifies the logical preference (.lpf) file:

- Timing settings are modified for the debug logic.
- RVL_ALIAS preferences are added. RVL_ALIAS maps clock names generated by Reveal Inserter to the clock names used in the original design. So you do not need to change your preferences that refer to those clock signals.

Limitations

Reveal Inserter has the following limitations in the current release.

Unsupported VHDL and Verilog Features in Reveal Inserter

The following features that are valid in the VHDL and Verilog languages are not supported in Reveal Inserter when you use the RTL flow:

- Array types of two dimensions or more are not shown in the port or node section.
- Undeclared wires attached to instantiated component instances are not shown in the hierarchical design tree. You must declare these wires explicitly if you want to trace or trigger with them.
- Variables used in conditional statements like if-then-else statements are not available for tracing and triggering.
- Variables used in selection statements like the case statement are not available for tracing and triggering.
- If function calls are used in the array declaration, the actual size of the array is unknown to Reveal Inserter.
- Entity and architecture of the same design cannot be in different files.
- In Verilog, you must explicitly declare variables at the very beginning of a module body to avoid obtaining different results from various synthesis tools.
- In VHDL, you must declare synthesis attributes within an entity, not within an architecture, to avoid obtaining different results from various synthesis tools.

Syn_keep and Preserve_signal Attributes

In VHDL, always define the syn_keep and preserve_signal attributes as Boolean types when you declare them in your design. Synplify defines them as Boolean types, and Reveal Inserter will issue an error message if you define them as strings.

Constants

You can use signals that are set to constant values as trace or trigger signals. Although it is not recommended that you use these signals as trace or trigger signals, Reveal Inserter will not issue an error message if you do.

You cannot use a signal for the sample clock that has been set to a constant value. The value of the sample clock must be able to change. However, Reveal Inserter will not generate an error message if you use a signal set to a constant value for the sample clock. It does not know whether a signal has been set to a constant value before synthesis and can only avoid displaying those that were originally declared as constants in the hierarchy tree.

Dangling or Unconnected Nets

Dangling, or unconnected, nets in Verilog or VHDL code are available for use with Reveal Inserter. However, if you are using EDIF files as your source, dangling nets are probably not available because synthesis tools, which produce EDIF files, normally optimize out dangling logic.

Getting Started

After you create a project in Diamond, you can start Reveal Inserter and create a Reveal project. Or open an existing Reveal project for modification.

Starting Reveal Inserter

Reveal Inserter is started from the main window. Open the desired design project to have access to the tools.

To start Reveal Inserter:

- Do one of the following:
  - In the main window, choose **Tools > Reveal Inserter.**
  - In the toolbar, click the Reveal Inserter button.

When Reveal Inserter opens, it shows the active Reveal project or, if there are no existing projects, Reveal Inserter creates one.

When Reveal Inserter opens a design, it must parse and statically elaborate it. In some cases, code successfully synthesized with some synthesis tools may be flagged as having an error when Reveal Inserter tries to open the design. In these cases, Reveal Inserter is interpreting the HDL code more strictly than the chosen synthesis tool. It is likely that the code would not synthesize with a different synthesis tool or would have other compliance issues.

To correct this problem, see the reveal_error.log file in the project directory. This file contains information and error messages that enable you to see any problems found in the design.

Creating a New Reveal Inserter Project

After you create a project in Diamond, you can create a project in Reveal Inserter.

To create a new Reveal Inserter project:

1. Choose **File > New > File** or click in the toolbar and choose **File** from the drop-down menu.

   The New File dialog box appears.

2. Under Source Files, choose **Reveal Project File**.
3. Type in the base name for the .rvl file. The ".rvl" extension is added automatically.
4. If you do not want the file to be in the design project's folder, click **Browse** and browse to the desired location.
5. If you do *not* want the Reveal project to be part of the design project , clear **Add to project**. Not recommended.
6. Choose a design implementation.
7. Click **New**.

If you want to use a different set of trace and trigger signals after you create the first Reveal Inserter project, you can create a new Reveal Inserter project for the same design project.

Opening an Existing Reveal Inserter Project

To open an existing project, you must have available a Reveal Inserter project (.rvl) file and a Reveal Inserter settings (.rvs) file from a previous Reveal Inserter session.

To open an existing Reveal Inserter project:

1. Choose **File > Open > File** in the main window or click in the toolbar and choose **File**.
2. In the Open File dialog box, browse to the .rvl file of interest and click **Open**.

If the desired .rvl file has been recently opened, you can open it directly from the File menu.

To open a recently opened .rvl file directly from the File menu:

- Choose **File > Recent Files >** <*filename*> from the list of the four most recently opened files near the bottom of the File menu.

Managing the Cores in a Project

Each Reveal Inserter project can include up to 15 debug logic cores. Each core has its own settings for the debug logic, such as trace signals, trigger signals, sample clock, sample enable, and trigger output signal. These settings are called a dataset. In many cases, a single core is all that is required to debug a design. However, in designs with multiple clock regions, it may be necessary to sample different clock regions at the same time. For those types of designs, it is recommended that you use multple cores, one for each clock region where the clock is used as the sample clock for the core.

Adding a Core

When you open a new project, Reveal Inserter automatically adds the first debug logic core to the first dataset and gives it a name of <*top_module*>_LA<*number*>, where *top_module* is the name of the top module in the Reveal Inserter project, and *number* is a sequential number. For example, the third core added to the “counter” project would be named counter_LA3. The core name is case insensitive—for example, “core_LA0” is the same as “core_la0.”

All Reveal cores are listed in the Dataset pane in the Reveal Inserter window.

To add a core to a dataset:

1. Choose **Debug > Add New Core** or right-click in the Dataset pane and choose **Add New Core**.
2. Choose the type of core.

   Reveal Inserter creates a new core. The added cores are displayed in the Dataset pane.

Renaming a Core

You can rename a debug logic core if you want to change its initial name.

To rename a core or cores in a project:

1. Highlight the name of the core in the Dataset pane, and choose **Debug > Rename Core**, or right-click on the name of the core and choose **Rename Core** from the pop-up menu.
2. Type the new name of the module over the old name.

During the renaming process, Reveal Inserter verifies that:

- The core name begins with a letter and consists of letters, numbers, and underscores (_).
- The core name is not the same as that of any other core.
- The core name is not the same as that of any module or instance in the design.

Removing a Core

You can also remove a debug logic core.

To remove a core or cores from a project:

- Select the core in the Dataset pane, and choose **Debug > Remove Core**, or right-click on the name of the core and choose **Remove Core** from the pop-up menu.

Viewing Signals in the Design Tree Pane

In the Design Tree pane of the Reveal Inserter window, you can display the hierarchy of the whole design, including the ports and nodes in the top module and submodules, so that you can choose the signals to use for data tracing and triggering.

From the Design Tree pane, you can drag a signal to the upper half of the Trace Signal Setup tab to set it as a trace signal or drag it to the lower half of the tab to set it as a sample clock signal or a sample enable signal.

In the Design Tree pane, the names of trace, trigger, and control signals are in bold font if they are currently being used.

To view all signals in the design tree:

- Right-click on the design name in the Design Tree pane and choose **Expand All** from the pop-up menu.

To view the buses, ports, top-level signals, and top level of the hierarchy:

- Right-click on the design name in the Design Tree pane and choose **Collapse All** from the pop-up menu.

You can also view signals and buses in the Trace Data pane of the Trace Signal Setup tab.

Searching for Signals

You can search for a signal or signals and set the selected signals as trace signals, trigger unit signals, sample clock signals, or sample enable signals. You can search for signal names or patterns of characters.

To search for a signal:

1. In the Signal Search box in the Design Tree pane, enter the name of the signal or pattern to find. You can set a filter by using the case-insensitive alphanumeric characters and wildcards shown in the following table.
2. Click **Search**.

   If Reveal Inserter finds only one signal, it highlights it in the Design Tree pane.

   If Reveal Inserter finds multiple signals, it opens the Search Result dialog box to list all the signals found.
3. If you are searching for multiple signals, select the desired signals in the Search Result dialog box, and click **OK**.
   - Shift-click to select contiguous signals.
   - Control-click to select non-contiguous signals.

   The selected signals are now highlighted in the Design Tree pane.

From the Design Tree pane, you can drag signals to the Trace Data pane, the Sample Clock box, and the Sample Enable box in the Trace Signal Setup tab. You can also drag signals to the Signals (MSB:LSB) box in the Trigger Unit section of the Trigger Signal Setup tab.

Although the buses are displayed as "*busname*[*n*:*m*]" in the Design Tree pane, Reveal Inserter ignores the string after the bus name when it searches for buses. For example, if the design contains a bus called a[0:2], you can search for it by a pattern such as "a" or "a*," but you cannot use a pattern such as "a[*."

If a bus is named xyz, a search for xyz highlights the entire bus. A search for xyz* brings up the Search Result dialog box and all the individual signals in the xyz bus.

The following wildcards are supported in searches:

Wildcard Character	Characters to Replace	Example
?	Any single character	?a? where "a" is the middle character in a three-character string
*	Any sequence of characters	*a* where the string contains the "a" character

Wildcard Character	Characters to Replace	Example
[abc]	“a,” “b,” or “c”	[abc]* where the string begins with “a,” “b,” or “c”
[^abc]	Any character except “a,” “b,” or “c”	[^abc]* where the string does not begin with “a,” “b,” or “c”
[a-d]	Any character in the range of “a” through “d”	[a-d]* where the string begins with any character in the range of “a” through “d”
[^a-d]	Any character except those in the range of “a” through “d”	[a-d]* where the string does not begin with any character in the range of “a” through “d”

Setting Up the Trace Signals

The first step in performing a logic analysis is to specify how the data from the trace bus will be captured. Use the Trace Signal Setup tab in the Reveal Inserter window to choose the signals from which to collect sample data in the selected core.

Selecting the Debug Logic Core

Before you configure the trace signals, select the debug logic core to configure in the Dataset pane.

Selecting the Trace Signals

You can use either of two methods to select trace signals: dragging and dropping or using a search engine to find them. You can select up to 512 trace signals in each core.

To select trace signals by dragging and dropping:

▶ Select the desired signals in the Design Tree pane and drag them to the Trace Data pane in the Trace Signal Setup tab.

To select trace signals by using a search engine:

1. In the Signal Search box in the Design Tree pane, enter the name or pattern of the signal to find. You can set a filter by using case-insensitive alphanumeric characters and wildcards. See “Searching for Signals” on page 11 for information about the wildcards that you can use.
2. Click **Search**.

If Reveal Inserter finds only one signal, it highlights it in the Design Tree pane.

If Reveal Inserter finds multiple signals, it opens the Search Result dialog box to list all the signals found.

3. If you are searching for multiple signals, select the desired signals in the Search Result dialog box, and click **OK**.

   The signals are now selected in the Design Tree pane.

4. Drag them to Trace Data pane in the Trace Signal Setup tab.

Viewing Trace Signals and Buses

In the Trace Data pane in the Trace Signal Setup tab, you can display the signals in buses or remove them from view.

To display all the signals in all the buses:

▶ Right-click in the Trace Data pane, and choose **Expand All** from the pop-up menu.

To hide all the signals in all the buses:

▶ Right-click in the Trace Data pane, and choose **Collapse All** from the pop-up menu.

To display all the signals in an individual bus:

▶ Right-click on the bus and choose **Expand** from the pop-up menu.

To hide all the signals in an individual bus:

▶ Right-click on the bus and choose **Collapse** from the pop-up menu.

Grouping Trace Signals into a Bus

You can group trace signals, buses, or both into a bus.

To group signals or buses into a bus or to add signals or buses to a bus:

1. In the Trace Data pane of the Trace Signal Setup tab, select the signals, buses, or both to be grouped.
2. Choose **Debug > Group Trace Data**.
3. Double-click the new bus and type in the desired name.

Ungrouping Trace Signals in a Bus

You can ungroup the signals or buses in a bus.

To ungroup the signals, buses, or both in a bus:

1. In the Trace Data pane in the Trace Signal Setup tab, select the signals, buses, or both to be ungrouped from the bus.
2. Choose **Debug > UnGroup Trace Bus**.

Removing Signals and Buses from the Trace Data Pane

You can remove signals from the Trace Data pane in the Trace Signal Setup tab.

To remove a signal or a bus from the Trace Data pane:

1. In the Trace Signal Setup tab, select the signals to be removed from the Trace Data pane.
2. Choose **Debug > Remove Trace Data**, or right-click and choose **Remove** from the pop-up menu. You can also press the Delete key.

Renaming a Bus

You can rename a bus.

To rename a bus:

1. In the Trace Data pane of the Trace Signal Setup tab, select the bus.
2. Choose **Debug > Rename Trace Bus**, or right-click and choose **Rename** from the pop-up menu.
3. Type the new name of the module over the old name.

Setting Required Sample Parameters

For each core, you must set the certain sample parameters for the trace signals.

To set the required sample parameters:

1. In the **Sample Clock** box in the Trace Signal Setup tab, type the name of the clock signal or drag the clock signal from the design tree shown in the Design Tree pane.

   Note

   On the board, make sure that the minimum sample clock frequency is at least that of the JTAG clock. If the sample clock speed is too slow, you will be unable to complete logic analysis with Reveal Analyzer.

   The sample clock frequency should be no more than 200 MHz.

2. In the **Buffer Depth** box, specify the size of the trace memory buffer.

   This parameter defines the number of trace bus samples that a core can capture. It can be set to a minimum of 16 or to powers of 2 from 16 to 65536. The buffer size is determined by the amount of embedded memory in the FPGA.

3. In the **Implementation** box, specify how the debug logic is to be implemented in the FPGA. You can choose one of the following:

   - EBR – Implements the debugging logic as embedded block RAM (EBR). This setting is the default.
   - DistRAM – Implements the debugging logic as distributed RAM.

4. In the **Data Capture Mode** box, select Single Trigger Capture or Multiple Trigger Capture. Single Trigger Capture is enabled by default.

5. If you choose Multiple Trigger Capture, you must also set the **Minimum samples per trigger** option, which specifies the minimum number of data samples to collect per trigger. The minimum is either 8 or 1/256 of the total buffer depth, whichever is greater. The maximum number of samples depends on the design.

6. If you select Multiple Trigger Capture and are creating a POR module, choose the “Number of triggers for POR.” For an explanation of POR modules, see “POR Debug” on page 16.

Setting Sample Options

In addition to the required parameters, you can set options for the data sample.

Using a Sample Enable

A sample enable is an optional signal used to capture data only when the sample enable is active, either high or low. If you do not specify a sample enable signal, trace data is collected on every sample clock after the trigger.

You may want to use a sample enable in cases where you need to capture a lot of data, but the data is only important during certain times, not whenever the sample clock is running. In these cases, the sample enable is a “gate” that allows you to turn the capturing of data on and off. An example is a design that contains many different sections, but some sections only work during certain clock phases. You typically use a master clock and generate different signals for the phases. You could use one of the phases as the sample enable.

To set the sample enable:

- In the **Sample Enable** checkbox, indicate whether a sample enable signal is to be used. If you want to use a sample enable:

  a. Select the checkbox to indicate that a sample enable signal will be used. The checkbox is deselected by default.

b. Enter the name of the sample enable signal in the box beneath the checkbox, or drag the signal from the Design Tree pane.

c. In the box to the right of the signal name box, select either **Active High**, which means that trace data is captured when the sample enable is high and the sample clock occurs, or **Active Low**, which means that trace data is captured when the sample enable is low and the sample clock occurs. Active High is the default.

Each sample shown in the trace buffer is only captured when the sample enable is active and there is a sample clock. Data samples can be discontiguous, unlike those in a normal data capture.

Additionally, it is possible that the actual trigger condition may occur when the sample enable is not active. This causes two changes from a normal data capture:

- The actual data values for the trigger condition may not be visible, because the data cannot be captured when the sample enable is inactive.
- Reveal Analyzer cannot accurately calculate the trigger point, since the trigger point may have occurred when the sample enable is inactive. Normally a trigger point is shown as a single marker on the clock on which the trigger occurred. If a sample enable is used, a trigger region that spans 5 clock cycles is shown instead. Reveal Analyzer can guarantee that the trigger occurred in this region, but it cannot determine during which clock cycle the trigger occurred.

The sample enable is a very useful feature, but it takes more understanding than a normal data capture.

POR Debug

To monitor power-on reset (POR) functions an automatic “trigger enable” signal must be built into the Reveal module. This is because POR functions happen immediately after power-on of the test board, before Reveal Analyzer can be started. When the trigger enable signal transitions to active, the module will watch for the trigger and collect samples. This is similar to clicking the Run button in Reveal Analyzer. When Reveal Analyzer starts, it loads and displays any data collected by the POR modules.

POR modules are only supported with LatticeECP3, LatticeSC, LatticeXP2, and MachXO2.

To set a POR trigger enable:

1. In the POR Debug section, select **Trigger Enable**.
2. Find the POR trigger signal in the Design Tree view and drag it to the text box in the POR Debug section.
3. Choose whether the signal is **Active High** or **Active Low**.

Adding Trigger Signals to Trace Signals

You can add trigger signals to the trace signals so that the data from the trigger signals is included in the trace data. Tracing trigger signals increases the amount of logic used by the trace buffer.

To add the trigger signals to the trace signals:

- Select the **Include trigger signals in trace data** option. This option is turned off by default.

Adding Time Stamps to Trace Samples

In Reveal Inserter, you can optionally specify a sample clock count value to be stored with each trace sample to indicate the sample count clock value at which the sample was captured. This count is extra data (bits) captured into the trace buffer that increase the trace buffer’s width. This time stamp enables you to see how many sample clock intervals have elapsed between data captures when you use a sample enable. It is useful in some cases when it is necessary to know if you captured the right data. A time stamp is also useful when you try to synchronize data between multiple cores, off-chip data, or both. For example, if you trigger two cores at the same time, you can use the time stamps on the trace samples to calculate how the data between the cores compares.

To add time stamps to the trace samples:

1. Select the **Timestamp** box in the Trace Signal Setup tab.
2. In the drop-down menu in the Bits box next to the Timestamp box, select the amount of trace memory storage needed by the time stamp, in bits.

   The number of bits for the timestamp is the number of bits in the maximum count of the timestamp. But each bit is equivalent to adding another signal to be traced, so the amount of trace memory needed is therefore much larger. The minimum number of bits that appears in the drop-down menu is obtained by multiplying the value in the Buffer Depth box by 2 and converting the result to an exponential value. For example, if the value in the Buffer Depth box is 256, the minimum number of bits in the Bits drop-down menu is calculated as follows:

   256 X 2 = 512

   512 = 2^9

   So the minimum number of bits available in the Bits menu in this case is 9.

   The maximum number of bits available in the Bits menu is always 63.

Setting Up the Trigger Signals

The Reveal software has some similarities to and some differences from external logic analyzers. An external logic analyzer typically offers up to a few dozen signals or channels and megabits worth of capture data depth. Internal or embedded logic analyzers have different constraints. An internal logic analyzer can offer thousands of signal connections, since no extra pins are required to connect to the signal. But the resources inside an FPGA force a limitation on the amount of data that can be captured, typically constrained to several thousand bits. This difference drives different requirements. An internal logic analyzer requires the ability to accurately pinpoint the desired event in order to capture a smaller amount of data around that precise event. The capabilities in the Reveal software are designed specifically for the triggering requirements of an internal logic analyzer.

Triggering

With the Reveal software, it is easy to set up simple triggering conditions, as well as extremely complex triggers. Triggering in Reveal is based on the trigger unit and the trigger expression. A trigger unit is used to compare signals to a value, and a trigger expression is used to combine trigger units to form a trigger.

Some of Reveal's triggering features are static and some are dynamic. Static features can only be changed in Reveal Inserter and require the design to be re-implemented by synthesis, map, place, and route. Although you can set most of the dynamic features in Reveal Inserter, you can change all dynamic features when Reveal Analyzer is running, and you do not have to re-implement the design.

Table 1: Where Trigger Features Can Be Changed

Feature		Reveal Inserter	Reveal Analyzer
Trigger Units	Add	✓	
	Name	✓	✓
	Signals	✓	
	Operator	✓	✓
	Radix	✓	✓
	Value	✓	✓
Trigger Expressions	Add	✓	
	Remove	✓	✓
	Name	✓	✓
	Expression	✓	✓
	RAM type	✓	
	Maximum sequence depth	✓	
	Maximum event counter	✓	

Table 1: Where Trigger Features Can Be Changed (Continued)

Feature		**Reveal Inserter**	**Reveal Analyzer**
Multiple Trigger Capture	Make available	✓	
	Number of samples per trigger	✓	✓
	Number of triggers		✓
Other Features	AND All versus OR All		✓
	Final event counter size	✓	✓
	Trace buffer depth	✓	
	Timestamp	✓	
	Trigger position		✓

Trigger Units

The trigger unit is used to compare a number of input signals to a value. A number of different operators are available for comparison and can be dynamically changed during analysis, along with the comparison value and the trigger unit name.

You can change the signals in a trigger unit only in Reveal Inserter. Changing the input signals requires the design to be re-implemented.

You can specify up to 16 trigger units for each debug core. A common technique is to group associated input signals into a trigger unit. For example, you might use a trigger unit for the address bus in a design, another for the data bus, and another for the control signals.

Most of the trigger unit operators use standard logical comparisons between the current value of the combined signals of the trigger unit and a specified value. But some of the operators are unusual and need some explanation.

With the exception of “serial compare,” the operators can be changed in Reveal Analyzer.

Standard Logical Operators Reveal includes the following operators:

- == equal to
- != not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

Rising-Edge and Falling-Edge Operators The “rising edge” and “falling edge” operators check for change in the signal value, not the value itself. So the trigger unit’s specified value is a bit mask showing which signals should have a rising or falling edge. A 1 means “look for the edge;” a 0 means “ignore this bit.” A multiple-bit value is true if any of the specified bits has the edge.

For example, consider a trigger unit defined as cout[3:0], rising edge, 1110. This trigger unit will be true only when cout[3], cout[2], or cout[1] have a rising edge. What happens on cout[0] does not matter.

- 0000 > 1110

  True because cout[3], cout[2], and cout[1] rose.

- 0000 > 1111

  True for the same reason. It does not matter whether cout[0] rises or not.

- 0000 > 0100

  True because a rising edge on any of the specified bits is sufficient.

- 1000 > 1000

  False because cout[3] did not rise. It just stayed high.

Serial Compare The “serial compare” operator checks for a series of values on a single signal. For example, if a trigger unit’s specified value is 1011, the “serial compare” operator looks for a 1 on the first clock, a 0 on the next clock, a 1 on the next clock, and a 1 on the last clock. Only after those four conditions are met in those four clock cycles is the trigger unit true.

Serial compare is available only when a single signal is listed in the trigger unit’s signal list. The radix is automatically binary.

You can only set the serial compare operator in Reveal Inserter. You cannot change it or select it in Reveal Analyzer as you can the other operators.

Trigger Expressions

Trigger expressions are combinations of trigger units. Trigger units can be combined in combinatorial, sequential, and mixed combinatorial and sequential patterns. A trigger expression can be dynamically changed at any time. Each core supports up to 16 trigger expressions that can be dynamically enabled or disabled in Reveal Analyzer. Trigger expressions support AND, OR, XOR, NOT, parentheses (for grouping), THEN, NEXT, # (count), and ## (consecutive count) operators. Each part of a trigger expression, called a sequence, can also be required to be valid a number of times before continuing to the next sequence in the trigger expression.

Detailed Trigger Expression Syntax Trigger expressions in both Reveal Inserter and Reveal Analyzer use the same syntax.

Operators You can use the following operators to connect trigger units:

- & (AND) – Combines trigger units using an AND operator.
- | (OR) – Combines trigger units using an OR operator.
- ^ (XOR) – Combines trigger units using a XOR operator.
- ! (NOT) – Combines a trigger unit with a NOT operator.

- Parentheses – Groups and orders trigger units.
- THEN – Creates a sequence of wait conditions. For example, the following statement:

  ```
  TU1 THEN TU2
  ```

  means “wait for TU1 to be true, then wait for TU2 to be true.”

  The following expression:

  ```
  (TU1 & TU2) THEN TU3
  ```

  means “wait for TU1 and TU2 to be true, then wait for TU3 to be true.”

  Reveal supports up to 16 sequence levels.

  See “Sequences and Counters” on page 21 for more information on THEN statements.
- NEXT – Creates a sequence of wait conditions, like THEN, except the second trigger unit must come immediately after the first. That is, the second trigger unit must occur in the next clock cycle after the first trigger unit. See “Sequences and Counters” on page 21 for more information on NEXT statements.
- # (count) – Inserts a counter into a sequence. See “Sequences and Counters” on page 21 for information on counters.
- ## (consecutive count) – Inserts a counter into a sequence. Like # (count) except that the trigger units must come in consecutive clock cycles. That is, one trigger unit immediately after another with no delay between them. See “Sequences and Counters” on page 21 for information on counters.

Case Sensitivity Trigger expressions are case-insensitive.

Spaces You can use spaces anywhere in a trigger expression.

Sequences and Counters Sequences are sequential states connected by THEN or NEXT operators. A counter counts how many times a state must occur before a THEN or NEXT statement or the end of the sequence. The maximum value of this count is determined by the Max Event Counter value. This value must be specified in Reveal Inserter and cannot be changed in Reveal Analyzer.

Here is an example of a trigger expression with a THEN operator:

```
TU1 THEN TU2
```

This trigger expression is interpreted as “wait for TU1 to be true, then wait for TU2 to be true.”

If the same example were written with a NEXT operator:

```
TU1 NEXT TU2
```

it is interpreted as “wait for TU1 to be true, then wait *one clock cycle* for TU2 to be true.” If TU2 is not true in the next clock cycle, the sequence fails and starts over, waiting for TU1 again.

The next trigger expression:

```
TU1 THEN TU2 #2
```

is interpreted as “wait for TU1 to be true, then wait for TU2 to be true for two sample clocks.” TU2 may be true on consecutive or non-consecutive sample clocks and still meet this condition.

The following statement:

```
TU1 ##5 THEN TU2
```

means that TU1 must occur for five consecutive sample clocks before TU2 is evaluated. If there are any extra delays between any of the five occurrences of TU1, the sequence fails and starts over.

The next expression:

```
(TU1 & TU2)#2 THEN TU3
```

means “wait for the second occurrence of TU1 and TU2 to be true, then wait for TU3.”

The last expression:

```
TU1 THEN (1)#200
```

means “wait for TU1 to be true, then wait for 200 sample clocks.” This expression is useful if you know that an event occurs a certain time after a condition.

You can only use one count (# or ##) operator per sequence. For example, the following statement is not valid, because it uses two counts in a sequence:

```
TU1 #5 & TU2 #2
```

Multiple count values are allowed for a single trigger expression, but only one per sequence. For two count operators to be valid in a trigger expression, the expression must contain at least one THEN or NEXT operator, as in the following example:

```
(TU1 & TU2) #5 THEN TU2 #2
```

This expression means “wait for TU1 and TU2 to be true for five sample clocks, then wait for TU2 to be true for two sample clocks.”

Also, the count operator must be applied to the entire sequence expression, as indicated by parentheses in the expression just given. The following is not allowed:

```
TU1 #5 & TU2 THEN TU2 #2
```

The count (#) operator cannot be used as part of a sequence following a NEXT operator. A consecutive count (##) operator may be used after a NEXT operator. The following is not allowed:

```
TU1 NEXT TU2 #2
```

The count (# or ##) operators can only be used in one of two areas:

- Immediately after a trigger unit or parentheses(). However, if the trigger unit is combined with another trigger unit without parentheses, a # cannot be used.
- After a closing parenthesis.

Precedence The symbols used in trigger expression syntax take the following precedence:

- Because it inserts a sequence, the THEN and NEXT operators always take the highest precedence in trigger expressions.
- Between THEN or NEXT statements, the order is defined by parentheses that you insert. For example, the following trigger expression:

  ```
  TU1 & (TU2|TU3)
  ```

  means “wait for either TU1 and TU2 or TU1 and TU3 to be true.”

  If you do not place any parentheses in the trigger expression, precedence is left to right until a THEN or NEXT statement is reached.

  For example, the following trigger expression:

  ```
  TU1 & TU2|TU3
  ```

  is interpreted as “wait for TU1 & TU2 to be true or wait for TU3 to be true.”
- The precedence of the ^ operator is same as that of the & operator and the | operator.
- The logic negation operator (!) has a higher precedence than the ^ operator, & operator, or | operator, for example:

  ```
  !TU1 & TU2
  ```

  means “not TU1 and TU2.”
- The # and ## operators have the same precedence as the ^ operator, & operator, or | operator. However, they can only be used in one of two areas:
  - Immediately after a trigger unit or trigger units combined in parentheses. However, if the trigger unit is combined with another trigger unit without parentheses, a # or ## operator cannot be used.

    Here is an example of correct syntax using the count (#) operator:

    ```
    TU1 #2 THEN TU3
    ```

    This statement means “wait for TU1 to be true for two sample clocks, then wait for TU3.”

    However, the following syntax is incorrect, because the count operator is applied to multiple trigger units combined without parentheses:

    ```
    TU1 & TU2#2 THEN TU3
    ```
  - After a closing parenthesis. Use parentheses to combine multiple trigger units and then apply a count, as in the following example:

    ```
    (TU1 & TU2)#2 THEN TU3
    ```

This statement means “wait for the combination of TU1 and TU2 to be true for two sample clocks, then wait for TU3.”

Following is a series of examples that demonstrate the flexibility of trigger expressions.

Example 1: Simplest Trigger Expression Following is the simplest trigger expression:

TU1

This trigger expression is true, causing a trigger to occur when the TU1 trigger unit is matched. The value and operator for the trigger unit is defined in the trigger unit, not in the trigger expression.

Example 2: Combinatorial Trigger Expression An example of a combinatorial trigger expression is as follows:

TU1 & TU2 | TU3

This trigger expression is true when (TU1 and TU2) or TU3 are matched. If no precedence ordering is specified, the order is left to right.

Example 3: Combinatorial Trigger Expression with Precedence Ordering In the following example of a combinatorial trigger expression, precedence makes a difference:

TU1 & (TU2 | TU3)

This trigger expression gives different results than the previous one. In this case, the trigger expression is true if (TU1 and TU2) or (TU1 and TU3) are matched.

Example 4: Simple Sequential Trigger Expression Following is an example of a simple sequential trigger expression:

TU1 THEN TU2

This trigger expression looks for a match of TU1, then waits for a match on TU2 a minimum of one sample clock later. Since this expression uses a THEN statement, it is considered to have multiple sequences. The first sequence is “TU1,” since it must be matched first. The second sequence is “TU2,” because it is only checked for a match after the first sequence has been found. The “sequence depth” is therefore 2.

The sequence depth is an important concept to understand for trigger expressions. Since the debug logic is inserted into the design, logic must be used to support the required sequence depth. Matching the depth to the entered expression can be used to minimize the logic. However, if you try to define a trigger expression that has a greater sequence depth than is available in the FPGA, an error will prevent the trigger expression from running. The dynamic capabilities of the trigger expression can therefore be limited. To allow more flexibility, you can specify the maximum sequence depth when you set up the debug logic in Reveal Inserter. You can reserve more room for the trigger expression than is required for the trigger expression currently entered. If you specify multiple trigger expressions, each trigger expression can have its own maximum sequence depth.

Example 5: Mixed Combinatorial and Sequential Trigger Expression Here is an example showing how you can mix combinatorial and sequential elements in a trigger expression:

TU1 & TU2 THEN TU3 THEN TU4 | TU5

This trigger expression only generates a trigger if (TU1 AND TU2) match, then TU3 matches, then (TU4 or TU5) match. You can set precedence for any sequence, but not across sequences. The expression (TU1 & TU2) | TU3 THEN TU4 is correct. The expression (TU1 & TU2 THEN TU3) | TU4 is invalid and is not allowed.

Example 6: Sequential Trigger Expression with Sequence Counts The next trigger expression shows two new features, the sequence count and a true operator to count sample clocks:

(TU1 & TU2)#2 THEN TU3 THEN TU4#5 THEN (1)#200

This trigger expression means wait for (TU1 and TU2) to be true two times, then wait for TU3 to be true, then wait for TU4 to be true five times, then wait 200 sample clocks. The count (# followed by number) operator can only be applied to a whole sequence, not part of a sequence. When the count operator is used in a sequence, the count may or may not be contiguous. The always true operator (1) can be used to wait or delay for a number of contiguous sample clocks. It is useful if you knew that an event that you wanted to capture occurred a certain time after a condition but you did not know the state of the trigger signals at that time.

However, there is a limitation on the maximum size of the counter. This depends on how much hardware is reserved for the sequence counter. When you define a trigger expression, the Max Event Counter setting in the Trigger Expression section of Reveal Inserter and Reveal Analyzer specifies how large a count value is allowed in the trigger expression. Each trigger expression can have a unique Max Event Counter setting.

Trigger Expression and Trigger Unit Naming Conventions

You can rename trigger units and trigger expressions. The names can be a mixture of lower-case or upper-case letters, underscores, and digits from 0 through 9. The first character must be either an underscore or a letter. The names can be any length.

Final Event Counter

The final event counter allows a counter to be added to the final trigger of one or more trigger expressions. In order to use the final event counter during logic analysis, you must specify it during insertion, along with the maximum count allowed. The actual count used by the counter during triggering can be dynamically changed during logic analysis.

Multiple Core Support for Triggers

Each core in a design has its own triggers and traced signals. When a design is “triggered” (meaning that the triggers are enabled and active so that the debug logic is running and looking for the trigger condition), you can specify which individual cores are enabled.

Simultaneous Trigger Activation in Multiple Cores Since each core is typically a different clock region, you can specify whether the triggering is enabled for each core when triggering for the device is activated.

For each core, you can indicate if the trigger or triggers should be enabled when triggering for the device is activated.

Cross-Triggering To support triggers based on multiple sample clocks, cross-triggering is available between different debug cores.

- Reveal provides an optional trigger-out signal in the triggering section for every core.
- If a design has multiple cores, trigger-out signals from other cores are listed as an available signal for triggers in another core. To use a trigger-out signal as an input to another core, you must specify it as a “net” or “both” type. The I/O type is only used for connecting the trigger-out to an external I/O. Trigger-out signals are listed in the Trigger Output pane at the bottom left of the Reveal Inserter window.

Adding Trigger Units

You can add trigger units only in Reveal Inserter. You cannot add them in Reveal Analyzer. You can change some of the trigger conditions defined in Reveal Inserter in Reveal Analyzer during hardware debugging.

All trigger units are automatically available for use in all trigger expressions defined.

You can add up to 16 trigger units per core. Each trigger unit consists of the following:

- Trigger unit name (label)
- Signals in the trigger unit
- Comparison function
- Radix of the trigger unit value

- Value of the trigger unit

To add a trigger unit:

1. If you want the buses in the new trigger units that you will add to have a certain radix by default, set that radix in the **Default Trigger Radix** box in the Trigger Unit section of the Trigger Signal Setup tab before you add any trigger units.

   Changing the trigger radix value does not affect any trigger units that were created before you made the change.

2. To add a new trigger unit, click **Add** in the Trigger Unit section of the Trigger Signal Setup tab.

   A line now appears in the Trigger Unit section, with a default trigger unit named TU<*number*>, where *number* is a sequential number. The first trigger unit is named TU1 by default.

Renaming Trigger Units

You can rename a trigger unit.

To rename a trigger unit:

- Double-click in the appropriate box in the Name column of the Trigger Unit section of the Trigger Signal Setup tab, backspace over the existing name, and type in the new name.

Setting Up Trigger Units

All signals must be defined for a trigger unit in Reveal Inserter. You cannot change them in Reveal Analyzer.

To set up a trigger unit:

1. If you want to change the default name of the trigger unit, backspace over the default name in the Name box in the Trigger Unit section of the Trigger Signal Setup tab and type the new name.

2. Specify the signals in the trigger unit:

   a. Double-click in the box in the **Signals (MSB:LSB)** column.

      The TU Signals dialog box appears.

   b. In the Select Signals box of the dialog box, highlight the signal or signals that you want to use in the trigger unit, and click **>** to move them to the box on the right. (Shift-click to select multiple signals.)

      Each trigger unit can have up to 256 signals. Since there are 16 allowable trigger units, each core can have a maximum of 4096 trigger signals.

c. If you want to change the order of a signal in the list of signals, highlight its name and click the up arrow to move it up one line or the down arrow to move it down one line.

   The order of the signals affects how the comparison is performed.

d. Click **OK**.

As an alternative to this procedure, you can drag and drop signals from the Design Tree pane to the Signals (MSB:LSB) box in a trigger unit.

If you want to select certain signals by using a search engine:

a. In the Signal Search box in the Design Tree pane, enter the name or pattern of the signal to find. You can set a filter by using case-insensitive alphanumeric characters and wildcards. See “Searching for Signals” on page 11 for information about the wildcards that you can use.

b. Click **Search**.

   If Reveal Inserter finds only one signal, it highlights it in the Design Tree pane.

   If Reveal Inserter finds multiple signals, it opens the Search Result dialog box to list all the signals found.

c. If you are searching for multiple signals, select the desired signals in the Search Result dialog box, and click **OK**.

   The signals are now selected in the Design Tree pane.

d. Drag them to Signals (MSB:LSB) box in the Trigger Unit section of the Trigger Signal Setup tab.

If you move the cursor over a trigger-unit line in the Signals box, the software displays a complete list of the signals in that trigger unit.

3. In the **Operator** column, set the comparators for the trigger condition. You can choose from the following states:

   - == equal to
   - != not equal to
   - > greater than
   - >= greater than or equal to
   - < less than
   - <= less than or equal to
   - Rising edge – compares on the rising edge of the clock
   - Falling edge – compares on the falling edge of the clock
   - Serial compare – compares until the trigger condition is met. For example, if the trigger condition is 10011, the serial compare option looks for a 1 on the first clock, a 0 on the next clock, a 0 on the next clock, a 1 on the next clock, and a 1 on the last clock. Only if those five conditions are met in those five clock cycles will the serial compare output be active.

The serial comparator is available only when a single signal is listed in the Trigger Unit signal list. If you choose this option, you must choose Binary in the Radix box.

You can only set the serial compare operator in Reveal Inserter. You cannot change it as you can other operators in Reveal Analyzer.

The default comparator is == (equal to).

For more information on the effect of the “Rising edge” and “Falling edge” operators, see “Rising-Edge and Falling-Edge Operators” on page 19.

Both the operator type and the trigger unit value can be changed in Reveal Analyzer during hardware debugging.

4. In the **Radix** column, set the radix of the trigger unit value given in the Value box by selecting a radix from the drop-down menu. You can choose one of the following:

   - Binary. This is the default. You must choose Binary if you selected “Serial compare” as a comparator.
   - Octal
   - Decimal
   - Hexadecimal
   - <*token_set_name*>. To select <*token_set_name*>, you must have created token sets in Reveal Analyzer. See “Creating Token Sets” on page 58 for instructions on creating token sets.

5. In the **Value** column, enter the comparison value.

   This value is the pattern of highs and lows that you want on the trigger unit that will initiate collection of the trace data. The default is binary, unless you selected <*token_set_name*> in the Radix column.

   If you selected <*token_set_name*> in the Radix column, a drop-down menu opens in the Value column. This menu lists all the tokens that you entered in the Token Manager dialog box for the chosen token set. Select any name. The only token sets available for a given bus must match the bit width of the bus. Other token sets will not be listed as choices for that bus.

   You can use “x” for a don’t-care value in the Value column if you selected Binary, Octal or Hexadecimal in the Radix column and if you selected the ==, !=, or serial compare operators in the Operator column.

Removing Trigger Units

You can remove trigger units in Reveal Inserter, but you cannot remove them in Reveal Analyzer.

To remove a trigger unit:

1. In the Trigger Unit section of the Trigger Signal Setup tab, click in any box in the line representing the trigger unit that you want to remove.
2. Click **Remove**.

Adding Trigger Expressions

Trigger expressions are combinatorial or sequential equations of trigger units or both. Trigger expressions can be defined during insertion and changed in Reveal Analyzer. You can add up to 16 trigger expressions.

You can add trigger expressions only in Reveal Inserter. You cannot add them in Reveal Analyzer.

You can dynamically enable or disable individual trigger expressions before triggering is activated during hardware debugging.

To add a trigger expression:

▶ In the Trigger Expression section of the Trigger Signal Setup tab, click **Add**.

A line appears with the default trigger expression called TE<*number*>, where <*number*> is a sequential number. The first trigger expression is named TE1 by default. You can rename the trigger expression by backspacing over the name and typing a new name.

Renaming Trigger Expressions

You can rename a trigger expression.

To rename a trigger expression:

▶ Double-click in the appropriate box in the Name column of the Trigger Expression section of the Trigger Signal Setup tab, backspace over the existing name, and type in the new name.

Setting Up Trigger Expressions

You set up the initial trigger expressions in Reveal Inserter, but you can change them and their names in Reveal Analyzer. You can also enable or disable trigger expressions in Reveal Analyzer. However, you cannot change the sequence depth, the maximum sequence depth, or the maximum event counter of the trigger expressions in Reveal Analyzer.

To set up a trigger expression:

1. If you want to change the default name of the trigger expression, backspace over the default name in the **Name** box in the Trigger Expression section of the Trigger Signal Setup tab and type the new name.

   You can also change the name of a trigger expression in Reveal Analyzer.

2. In the **Expression** box, enter the names of the trigger units and the operators that you want to use to connect them.

   You can use the following operators to connect trigger units:

- & (AND) – Combines trigger units using an & operator.
- | (OR) – Combines trigger units using an OR operator.
- ^ (XOR) – Combines trigger units using a XOR operator.
- ! (NOT) – Combines a trigger unit with a NOT operator.
- Parentheses – Groups and orders trigger units.
- THEN – Creates a sequence of wait conditions. For example, the following statement:

  ```
  TU1 THEN TU2
  ```

  means "wait for TU1 to be true," then "wait for TU2 to be true."

  The following expression:

  ```
  (TU1 & TU2) THEN TU3
  ```

  means "wait for TU1 and TU2 to be true, then wait for TU3 to be true."

  Reveal supports up to 16 sequence levels.

  See "Sequences and Counters" on page 21 for more information on THEN statements.
- NEXT – Creates a sequence of wait conditions, like THEN, except the second trigger unit must come immediately after the first. That is, the second trigger unit must occur in the next clock cycle after the first trigger unit. See "Sequences and Counters" on page 21 for more information on NEXT statements.
- # (count) – Inserts a counter into a sequence. See "Sequences and Counters" on page 21 for information on counters.
- ## (consecutive count) – Inserts a counter into a sequence. Like # (count) except that the trigger units must come in consecutive clock cycles. That is, one trigger unit immediately after another with no delay between them. See "Sequences and Counters" on page 21 for information on counters.

For more information on the precedence of these symbols in trigger expression syntax, see "Precedence" on page 23.

Reveal Inserter checks the syntax and displays the syntax in red font if it is erroneous.

Both the trigger units and operators associated with a trigger expression can be changed in Reveal Analyzer during hardware debugging.

3. From the drop-down menu in the **Ram Type** box, specify how the trigger expression is to be implemented in the debug logic. You can choose one of the following:

   - EBR – Implements the trigger expression as embedded block RAM (EBR). Reveal Inserter calculates the appropriate number of EBRs. By default, the trigger expression is implemented as EBR.
   - <*number*> Slices – Implements the trigger expression as slices. Reveal Inserter calculates the appropriate number of slices.

   The **Sequence Depth** box is read-only, so you do not need to enter data in this box.

4. From the drop-down menu in the **Max Sequence Depth** box, specify the maximum number of sequences, or trigger units connected by THEN operators, that can be used in a trigger expression.

   You can choose 1, 2, 4, 8, or 16. Reveal supports up to 16 maximum sequence levels.

   If the number in the Sequence Depth box is higher than that set in the Max Sequence Depth box, the number in the Max Sequence Depth box appears in red to indicate an error.

   The Max Sequence Depth value is set statically in Reveal Inserter and cannot be changed in Reveal Analyzer.

5. From the drop-down menu in the **Max Event Counter** box, specify the maximum size of the count in the trigger expression (the count is how many times a sequence must occur before a THEN statement). You can choose 1 and powers of 2 from 2 to 65,536. The maximum is 65,536. The default is 1. If the largest counter value used in the trigger expression is larger than that set in the Max Event Counter box, the number in the Max Event Counter box appears in red.

   You cannot change the Max Event Counter setting in Reveal Analyzer. You can only change it in Reveal Inserter.

   You can also add a counter to the output of the final trigger from all the trigger expressions. This counter adds an option to the final trigger output that combines all the trigger expressions. It is similar to the AND All and OR All options in the Analyzer.

6. To add a counter to the output of the final trigger, do the following:

   a. Select the **Enable final trigger counter** checkbox in the lower left portion of the Trigger Signal Setup tab.

   b. From the drop-down menu in the **Event Counter Value** box, select the maximum size of the count of all the trigger expression outputs combined. You can choose powers of 2 between 2 and 65536.

   Leaving the “Enable final trigger counter” option unselected is equivalent to setting the counter to a value of 1.

   You can change the value of this parameter in Reveal Analyzer, but you cannot make the value bigger, so be sure to reserve enough space for the count that you think you will need.

7. If you want create a trigger-out signal, do the following in the **Trigger Out** section:

   a. Select the **Enable Trigger Out** option.

   b. If you want to create a net, type in the name of the signal that you want to use as the trigger output signal in the Net box.

      The default name of the trigger output signal is reveal_debug_<*default_core_name*>_net. An example is reveal_debug_count_LA0_net.

   c. In the drop-down menu next to the Net box, select one of the following:

- **NET** – Creates a net signal that can be connected to the input of a trigger unit of another core. This setting is the default.
- **IO** – Creates an I/O signal that can trigger outside the chip.
- **BOTH** – Creates a signal that can both connect to the input of a trigger unit of another core and trigger outside the chip.

d. In the Polarity box, select the polarity of the trigger output signal from the drop-down menu, either **Active High** or **Active Low**.

e. In the “Minimum pulse width” box, enter the minimum pulse width of the trigger output signal, measured in cycles of the sample clock. You can input any value of 0 or greater as the minimum pulse.

f. If you chose IO or BOTH for the Net type, go into the .lpf file and add a LOCATE preference to specify which pin the trigger-out should go to. For example:

```
LOCATE COMP "reveal_debug_count_LA0_net" SITE "J2" ;
```

Once you create a net as a trigger output signal, its name appears in the Trigger Output pane beneath the Design Tree pane.

Removing Trigger Expressions

You can disable a trigger expression from being used by deselecting the checkbox to the left of the trigger expression name in Reveal Analyzer, but you can remove a trigger expression only in Reveal Inserter.

To remove a trigger expression:

1. Click in any box in the line representing the expression that you want to remove.
2. Click **Remove**.

Checking the Debug Logic Settings

Reveal Inserter automatically checks the settings of the debug logic before saving the project or inserting the debug logic cores, but you may want to check them independently beforehand. With one DRC command (Debug > Design Rule Check), you can verify the following:

- The core names begin with a letter and consist of letters, numbers, and underscores (_).
- A core name is not the same as that of any other core.
- The core name is not the same as that of any module already defined in the design.
- The number of cores is between 1 and 15.
- The number of trace signals is between 1 and 512.
- The number of trigger signals is between 1 and 4096.

- The number of trigger units and trigger expressions is between 1 and 16.
- The number of trigger signals in a trigger unit is between 1 and 256.
- A sample clock is specified.
- The sample clock signal is a 1-bit signal already defined in the design.
- A sample enable is specified.
- The sample enable signal is a 1-bit signal already defined in the design.
- The name of the trigger-out signal is given if this signal is enabled.
- The name of the trigger-out signal is not the same as any signal already defined in the design.
- The number of EBRs needed does not exceed the number available.
- The design includes an input signal.
- The syntax of the trigger expressions is correct.
- The trigger expression sequence is less than or equal to the maximum sequence.
- The trigger output signal is specified, if the Enable Trigger Out option is enabled.
- The trigger output signal is not the same as the name of any signal in the design.
- The values of the trigger unit are correct.
- The names of the trigger units and the trigger expressions conform to the guidelines given in the “Trigger Expression and Trigger Unit Naming Conventions” on page 25.
- The bit widths of the token values are the same as the bit widths of the trigger unit signals.

To check the logic debugging settings:

- Choose **Debug > Design Ruler Check** or click in the toolbar.

The results of the check are displayed in the Message tab. The Message tab also displays the total resource utilization, as in the following example:

```
The number of EBRs needed is 2.
The number of DistRAM (logic/ROM/RAM) slices needed is 0.
```

Creating SERDES Debug Modules

SERDES Debug modules help in debugging the serdes function by giving you read and write access to control registers. In Reveal Analyzer you can monitor what is happening in the serdes and experiment with different control settings. These modules are only available with ECP5UM designs that use the DCU block.

To add a SERDES Debug module:

1. Choose **Debug > Add New Core > Add SERDES Debug**.

   A new module appears in the Dataset view and the Serdes Debub Setup tab appears.

2. Select the sample clock in the Design Tree view and drag it to the Serdes Debug Setup tab. For help finding signals, see “Searching for Signals” on page 11. For more about sample clocks, see “Setting Required Sample Parameters” on page 14.

3. Select the serdes reset signal in the Design Tree view and drag it to the Serdes Debug Setup tab.

4. When you have set up all your modules, add them to the design project by choosing **Debug >** **Insert Debug** and running the design implementation process. See “Inserting the Debug Logic Cores” on page 36.

Saving a Project

Once you set the debug options, save the project so that the project information is saved in an .rvl and an .rvs file. Reveal Inserter automatically performs a design rule check before it saves these files.

When you select Debug > Insert Debug or click the button, Reveal Inserter saves the project information in an .rvl and an .rvs file.

Note

Reveal Inserter generates a “signature” or tracking mechanism each time that debug logic is inserted into the design. The signature is placed into the project file and into the debug logic. Reveal Analyzer reads this signature to ensure that the FPGA has been programmed with the latest debug logic. Reveal Inserter generates a new signature every time the .rvl file is written, and Reveal Analyzer checks this signature each time that it runs the design. If you save the project in Reveal Inserter without re-running the implementation process, Reveal Analyzer issues an error message, even if the debug logic was not changed.

To save the project settings in the current directory:

- Choose **File > Save** or click in the toolbar to save the project in .rvl and .rvs files in your current directory.

To save the project settings in another directory:

- Choose **File > Save As** to save the project in .rvl and .rvs files in a directory other than the current directory. In the Select Project dialog box, browse to the desired directory, enter the name of the .rvl file in the File Name box, select **.rvl** in the Files of Type box, and click **Save**.

Inserting the Debug Logic Cores

Once you set all the options in the tabs in Reveal Inserter window, you can insert the debug logic cores into the design.

To insert the debug logic cores into the design:

1. Choose **Debug > Insert Debug** or click in the Reveal Inserter toolbar.
2. In the Insert Debug to Design dialog box, select the cores to insert.
3. In the box beneath “Please select design .lpf file you want to use,” enter the path and name of the .lpf file to use.

   You must specify the .lpf file, which contains the timing preferences, to avoid generating false errors in the timing between the JTAG clock and the sample clocks. By default, Reveal Inserter uses the default .lpf file for the project.
4. If you want to import the .rvl file into the Diamond project, select **Import Reveal file to Diamond project**.

   Importing the .rvl file is the default, and the “Import Reveal file to Diamond project option” should not normally be left unchecked. If the .rvl file is not present in the design project, Reveal Inserter does not insert the debug logic into the design. If the .rvl file is changed and it is not re-imported into the design project, the design will not be implemented correctly.

   Importing the .rvl file enables Diamond to include the debug logic in the implementation. Only one .rvl file can be imported into a Diamond project. If you have previously imported an .rvl file, Diamond asks if it should replace the file. Reveal Inserter automatically saves the .rvl file before it inserts the debug logic.
5. Click **Insert**.

   This step automatically performs a design rule check, saves the debug options, generates and builds the debug logic cores, invokes the synthesis tool to synthesize the cores, and imports the .rvl file into the design project. A progress bar in the lower right of the Reveal Inserter window shows when the insertion is complete. If Reveal Inserter encounters no errors, you should see the following message in the Resource Usage tab, indicating that the core insertion was successful:

   ```
   Reveal Project imported.
   ```

   The Resource Usage tab also displays the total resource utilization, as in the following example:

   ```
   The number of EBRs needed is 2.
   The number of DistRAM (logic/ROM/RAM) slices needed is 0.
   ```

   In addition, a message box opens, informing you that the insertion was successful.
6. Click **OK** in the message box.

You should now see the .rvl file listed in the File List view, if it was not listed there before.

Removing Debug Logic from the Design

You may want to remove the debug logic cores in pre-production versions of your device to free block RAM resources and LUT-based logic and to expand the design. If you want to remove the debug logic cores from your design, you must remove the .rvl file or set it as inactive. Otherwise, the cores will continue to be inserted.

To remove the debug logic cores from the design:

1. In the File List view, highlight the .rvl file and right-click.
2. Do one of the following:
    - To remove the Reveal modules but keep the project, choose **Set as Inactive**.
    - To delete the Reveal project, choose **Remove**.

The .rvl file is now removed from the design.

Closing a Project

To close a Reveal Inserter project:

- Choose **File > Close**.

Exiting Reveal Inserter

To exit Reveal Inserter:

- Click ☒ in the Reveal Inserter tab.

Translating the Design

To build the design project database:

- Double-click the **Translate Design** process in the Process view.

During this process, Diamond invokes the synthesis tool.

Note

If your design contains an unlicensed IP block, the Hardtimer mechanism enables you to evaluate the IP. You can control this mechanism by highlighting the Translate Design Process, selecting Properties, and setting the Hardware Evaluation option to Enable.

If you use the Reveal tools on a design that includes an unlicensed IP block, you cannot disable the Hardtimer mechanism. It is required to generate the bitstream data or JEDEC file for Reveal Analyzer. If you set the Hardware Evaluation option to Disable, the Reveal flow overwrites the Hardtimer mechanism. However, the option automatically reverts to Disable when you exit the Reveal flow.

Mapping, Placing, and Routing the Design

To map, place, and route the design:

1. Double-click the **Map Design** process in the Process view.
2. Double-click the **Place & Route Design** process in the Process view.

All core clock pins must be located and driven by valid signals to ensure successful hardware debugging.

Generating a Bitstream or JEDEC File

To generate a bitstream:

- Double-click the **Export Files** process.

  This process creates a .bit or .jed file for FPGAs that is ready for downloading into the device.

Connecting to the Evaluation Board

Reveal Analyzer requires that a Lattice Semiconductor or USB download cable and a power supply be installed between your computer and evaluation board. Refer to the Programmer online Help for more information about setting up a cable connection.

Downloading Design onto the Device

For more information about downloading your design onto the device, refer to the Programmer online Help.

Performing Logic Analysis with Reveal Analyzer

After you have created your design project database with Diamond software, generated a debug logic core with Reveal Inserter, mapped, placed, and routed your design, and downloaded the design to the evaluation board, you can perform a logic analysis with Reveal Analyzer. Refer to "Reveal Logic Analyzer" on page 75 for more information about performing logic analysis.

User Interface Descriptions

The Reveal Inserter window appears when you first choose Tools > Reveal Inserter or click on the icon.

The Reveal Inserter window includes the following features:

Dataset pane Lists the cores in the current dataset. You debug a design with Reveal Inserter debug logic, using a certain sample clock. If you want to debug a multi-clock design, you can create a core for each sample clock region. These cores are listed in the Dataset pane. This pane can be detached as a separate window and can be hidden using the View menu.

Design Tree pane Lists all the buses and signals in the design. The names of trace, trigger, and control signals are in bold font if they are currently being used.

One of the following strings appears after each signal name to indicate its use:

- @Tc indicates that the signal is a trace signal.
- @Tg indicates that the signal is a trigger signal.
- @C indicates that the signal is a control signal.

Similarly, one of the following strings appears after each bus name to indicate its use:

- @Tc indicates that all the signals in the bus are used only as trace signals.
- @Tg indicates that all the signals in the bus are used only as trigger signals.
- @Tc, Tg indicates that all the signals in the bus are used as trace signals and trigger signals. It also appears if all the signals are used as trigger signals and none of the signals in the bus are used as control signals and you selected the "Include trigger signals in trace data" option.
- @Mx indicates the following:
  - At least one signal in the bus is used as a control signal.
  - Some signals in the bus are used both as trigger signals and as other kinds of signals.
  - Some signals in the bus are used both as trace signals and as other kinds of signals, except that all the signals are used as trigger signals,

none of the signals are used as control signals, and you selected the “Include trigger signals in trace data” option.

If you select or deselect the “Include trigger signals in trace data” option, the signal and bus names are immediately updated in the Design Tree pane. If you set a signal as a trigger signal and select the “Include trigger signals in trace data” option, the use of the signal is displayed as Tc, Tg, even though you did not drag the signal name to the Trace Data pane.

If you select a signal in the hierarchy, the Signal Information tab at the bottom of the Reveal Inserter window displays information about how it is used.

You can enlarge the width of this pane to see longer signal names by dragging the splitter at the right edge of the pane. This pane can be detached as a separate window and can be hidden using the View menu.

Signal Search box Enables you to search for a signal or a group of signals. You can enter a signal name or pattern. You can set a filter by using the case-insensitive alphanumeric characters and wildcards described in “Searching for Signals” on page 11.

If Reveal Inserter finds only one signal, it highlights it in the Design Tree pane. If it finds multiple signals, it opens the Search Signals dialog box to list all the signals found. When you click OK, the selected signals are highlighted in the Design Tree pane. From the Design Tree pane, you can drag signals to the Trace Data pane, the Sample Clock box, and the Sample Enable box in the Trace Signal Setup tab. You can also drag signals to the Signals (MSB:LSB) box in the Trigger Unit section of the Trigger Signals Setup tab.

Trigger Output pane Displays the names of the trigger output signals defined in the Trigger Out box in the Trigger Signal Setup tab.

This field displays the trigger output signals for all but the first core and only those output signals for which NET or BOTH were chosen. From the Trigger Out Nets box, you can drag the signal names to the top half of the Trace Signal Setup tab. This pane can be detached as a separate window and can be hidden using the View menu.

Trace Signal Setup Activates the Trace Signal Setup tab.

Trigger Signal Setup Activates the Trigger Signal Setup tab.

Trace Data pane Displays the selected trace signals in the Trace Signal Setup tab.

Serdes Debug Setup Serdes Debug Setup is where you specify signals to control the current SERDES Debug module. Select signals by dragging them from the Design Tree view.

Reveal Analyzer

Logic analyzers enable you to view signal information to debug design functionality. With external logic analyzers, you connect to pins on a board, set one or more trigger conditions, and sample and view collected data. Internal logic analyzers, such as Reveal Analyzer, depend on additional logic placed into the design for triggering and tracing, then transferring the data to a PC, usually through a JTAG connection, for viewing and analysis.

Reveal Inserter handles the task of inserting debug logic into your design. Before using Reveal Analyzer, you must use Reveal Inserter to allow debug access.

Reveal Analyzer enables you to configure trigger settings and extract information from a programmed device through the JTAG ports. It interfaces directly to the Reveal cores in the design. You can set up triggers, select capture modes, and run or stop the triggers. Reveal Analyzer displays the data captured on the silicon according to the settings that you specify.

Reveal Analyzer's graphical user interface enables you to view the trace data of a signal or bus in a waveform viewer.

Although an evaluation board is normally required to run Reveal Analyzer, Reveal Analyzer includes a demonstration design that you can run without the evaluation board so that you can learn how to use the tool.

Reveal Analyzer requires the Programmer programming software to configure the specified device. The acquired data is displayed in the waveform viewer.

You can export waveform data to a value change dump (.vcd) file, which can be imported by such third-party tools as ModelSim or Active-HDL. You can also output a file in ASCII tabular format for exporting the data into other tools such as Excel.

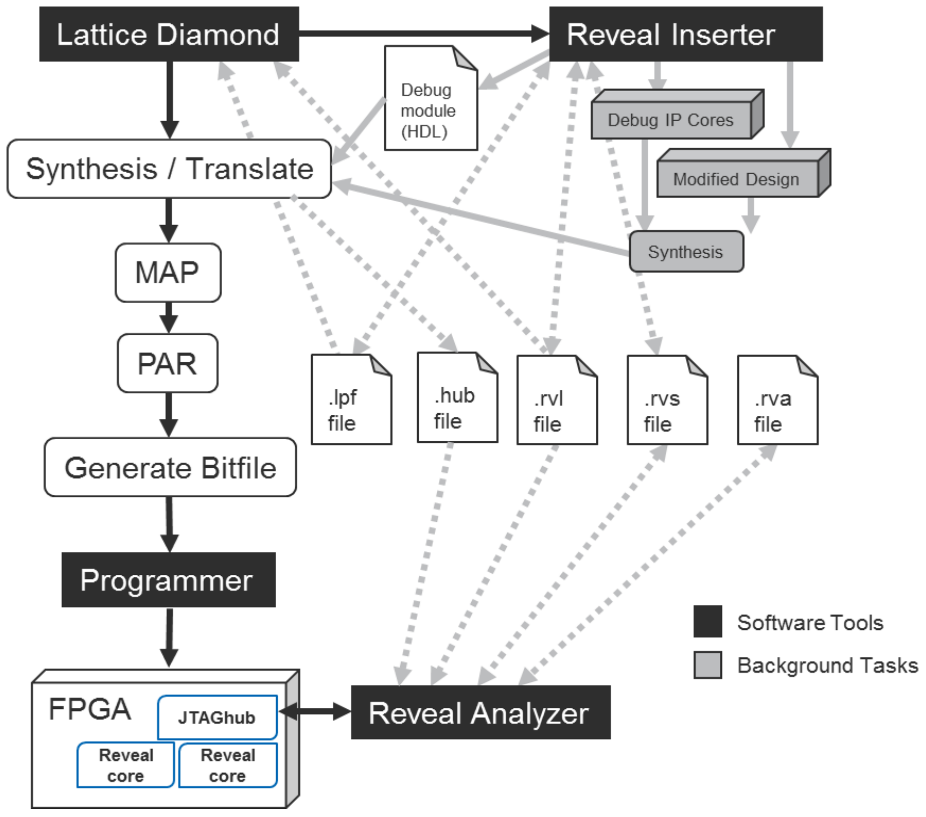
About Reveal Analyzer

This section introduces some of the key features of Reveal Analyzer: the devices that it supports, the steps in its design flow, its inputs, and its outputs.

Reveal On-Chip Debug Design Flow

The following figure shows the Reveal insertion and logic analysis design flow.

Figure 2: Reveal Design Flow



Before accessing Reveal Analyzer, you must install a Lattice Semiconductor or USB download cable and a power supply between your computer and the evaluation board. Refer to “Connecting to the Evaluation Board” on page 45 for information on this procedure.

You do not need to install a cable and a power supply if you want to run the demonstration design that comes with Reveal Analyzer so that you can learn how to use the tool. See “Using the Reveal Example Project” on page 2.

You must also have a design project that has had on-chip debug logic inserted by the Reveal Inserter software.

The general steps involved in performing a logic analysis are the following:

1. Start Reveal Inserter.
2. Configure the trace and trigger signal settings in Reveal Inserter.
3. Insert the debug logic with Reveal Inserter.
4. Build the database in the Process view.
5. Map, place, and route the design.
6. Generate the bitstream data or JEDEC file.
7. Set up a cable connection.
8. Download the design onto the device by using Programmer.
9. Start Reveal Analyzer.
10. Create a new Reveal Analyzer project or open an existing one.
11. Configure the trigger settings for each core in each device that you want to use to perform logic analysis of the design.
12. Click the Run ▶ button to perform the logic analysis, and wait for the design to trigger and download the trace information into Reveal Analyzer from the board.
13. View the resulting waveforms for each core.
14. Optionally, you can export the waveform data for each core in a value change dump (.vcd) file for use in third-party tools or in an ASCII-format text (.txt) file.

Inputs

Reveal Analyzer requires the following as input:

- A design project
- A Reveal Analyzer settings (.rvs) file, which is output by Reveal Inserter or Reveal Analyzer. It contains all the dynamically changeable trigger settings, such as trigger unit operators, trigger unit values, and any trigger expressions.
- An existing Reveal Inserter project (.rvl file), which contains the connections for each core and all the static settings of the debugging logic. The information in this file is statically set in Reveal Inserter and cannot be changed in Reveal Analyzer.
- A Reveal Analyzer project (.rva) file, which is the project file output by Reveal Analyzer in a previous session. It contains the information used by Reveal Analyzer, such as window settings, waveform trace signal positions, radixes, markers, and signal colors.
- JTAG hub (.hub) file, which is used by Reveal Analyzer to sort the data coming from the different Reveal modules.
- Optionally, a scan chain configuration (.xcf) file, which is generated by Programmer for programming devices in a JTAG daisy chain. The .xcf file contains information about each device, the data files targeted, and the operations to be performed. The .xcf file must reside in the design project directory for Reveal Analyzer to use it.

An .xcf file is required as input only if you will be programming devices in a JTAG daisy chain. It is not required if you will be programming a single device. It is recommended that an .xcf file be used since extra information such as the TCLKDelay value can be directly read from this file avoiding having to manually set this within Reveal Analyzer.

Note

Only one device can be debugged in a JTAG daisy chain.

Outputs

Reveal Analyzer generates the following files:

- A Reveal Analyzer project (.rva) file, which contains the information such as window settings, waveform trace signal positions, radixes, markers, and signal colors. This file is also an input file when you re-open a project that you previously saved.
- A Reveal Analyzer trace (.trc) file, which contains the waveform information acquired from previous runs of Reveal Analyzer. When you first open Reveal Analyzer, the waveform displays this information until you press the Run button. If the debug signals have been changed from a previous Reveal Analyzer run, incorrect information is displayed in the waveform when it is first opened. Once a run has been completed, the waveform contains valid information with the changed debug configuration.
- Optionally, a value change dump (.vcd) file, in which you can export waveform data for display in third-party tools such as ModelSim and Active-HDL. The .vcd file is an ASCII file containing header information, variable definitions, and variable value changes. Its format is specified by the IEEE 1364 standard.
- Optionally, an ASCII-format text (.txt) file, in which you can export waveform data for display in third-party tools such as ModelSim and Active-HDL. The .txt file is in a simple ASCII character-tab-delimited format. It includes a header line with the signal names, then each line contains the value for each signal, one line per each sample clock.

Inserting the Debug Logic

Before performing logic analysis with Reveal Analyzer, you must use Reveal Inserter to generate the debug logic and insert it into your design. You must also set up the trace and trigger signals to be used in Reveal Inserter.

Translating the Design

After you insert the debug logic in the Reveal Inserter, you return to the Process view to translate the design.

To translate the design:

- Double-click the **Translate Design** process.

During this process, Diamond invokes the synthesis tool.

Mapping, Placing, and Routing the Design

Once you build the translate the design, you map, place, and route the design.

To map, place, and route the design:

1. Double-click the **Map Design** process.
2. Double-click the **Place & Route Design** process.

All core clock pins must be located and driven by valid signals for successful hardware debugging.

Generating a Bitstream or JEDEC File

Now you generate a bitstream or JEDEC file, as appropriate, to download into the device.

To generate a bitstream or JEDEC file:

- Double-click the **Export Files** process in the Process view.

  This process creates a .bit or .jed file that is ready for downloading into the device.

Connecting to the Evaluation Board

Reveal Analyzer requires that a Lattice Semiconductor parallel port cable or USB download cable and a power supply be installed between your computer and the evaluation board so that you can program the device with Programmer.

To connect the evaluation board to your computer:

1. Install a driver for the download cable, if it has not been previously installed.
2. Reboot your computer, if the driver was not previously installed.
3. Attach the parallel port or USB cable to the parallel port or USB port of your system.

4. Plug in the AC adapter to a wall outlet, and plug the other end into the power jack provided on the evaluation board.

   Note

   You should follow the handling and power-up advice provided in the Lattice Semiconductor device evaluation board documentation when using the evaluation board.

5. In Diamond, choose **Tools > Programmer**.
6. Select **Options > Cable and IO Port Setup**.
7. Click **Auto Detect**, then click **OK**.
8. Attach the JTAG connector cable to the appropriate JTAG programming header of the evaluation board. See the device evaluation board documentation for details.

Refer to the Programmer Help for more information about your cable connection.

Downloading a Design onto the Device

To download a design onto the device, use Programmer. This process creates a scan chain configuration (.xcf) file. An .xcf file is not necessary to use Reveal Analyzer, unless you will be programming or debugging devices in a daisy chain (see “Programming and Debugging Devices in a Daisy Chain” on page 51 for more information). Reveal Analyzer derives the information in the downloaded design directly from the device on the board.

To download the design onto the device:

1. In Programmer, select **File > New**.

   A new chain configuration window appears.
2. Choose **ispTools > Scan Chain** or click the Scan toolbar icon.

   Programmer detects the device that you are using and adds it to the list.
3. Select the first device in the New Scan Configuration Setup list.
4. In the popup box labeled Multi Match Device’s ID List, select the appropriate device.
5. Highlight the selected device and choose **Edit Device** from the pop-up menu.
6. In the Device Information dialog box, click the **Select** button of the Device section to open the Select Device dialog box.
7. In the Select Device dialog box, do the following:
   a. In the Device Family box, select the appropriate device family.
   b. In the Device box, select the device, as appropriate for your revision of the standard evaluation board.

   c. In the Package box, select the appropriate package.
   d. Click **OK**.

8. Click the **Browse** button of the Data File section.
9. Select the <*design_name*>**.bit** or **.jed** file, and click **Open**.
10. In the Operation box, select **Fast Program**, if it is not already selected.
11. Click **OK** to close the Device Information dialog box.
12. Choose **Project > Download**, or click the **GO** button on the toolbar.

    After a few moments, the download and programming activity will end. A green PASS button appears in the New Scan Configuration Setup dialog box.

13. Select **File > Save As** to save the configuration setup as an .xcf file.

    The .xcf file must reside in the design project directory for Reveal Analyzer to use it.

    Note

    An .xcf file is not required unless you will be programming or debugging devices in a daisy chain. See "Programming and Debugging Devices in a Daisy Chain" on page 51 for more information.

14. In the File Name box in the dialog box that appears, type in <*design_name*>**.xcf**, and click **Save**.
15. Choose **File > Exit** in Programmer.

See the Programmer Help for detailed instructions on the downloading process.

Starting Reveal Analyzer

Before starting Reveal Analyzer you need to decide if you want to work with a new Reveal Analyzer (.rva) file or an existing one. The .rva file defines the Reveal Analyzer project and contains data about the display of signals in the LA Waveform view. You may want to start Reveal Analyzer with a new file to set up a new test. Start with an existing file to rerun a test, to set up a new test based on existing settings, or to just view the waveforms from an earlier test. (See "Starting with an Existing File" on page 49.)

How you start Reveal analyzer also depends on whether you are using it integrated with Diamond or using the stand-alone version, and on your operating system.

Starting with a New File

Before you can start Reveal Analyzer with a new .rva file, you need to be connected to your evaluation board with a download cable and have the board's power turned on.

To start Reveal Analyzer with a new file:

1. Issue the start command:

   - For integrated with Diamond, go to the Diamond main window and choose **Tools >** **Reveal Analyzer**.
   - For stand-alone in Windows, go to the Windows Start menu and choose **Programs > Lattice Diamond Reveal > Reveal Logic Analyzer**.
   - For stand-alone in Linux, go to a command line and enter the following:

     <*Reveal install path*>/bin/lin/**rvamain**

   If Reveal Analyzer finds just one .rva file in the active implementation, Reveal Analyzer opens with the data from that file. Otherwise, the Reveal Analyzer Startup Wizard dialog box appears.

2. If Reveal Analyzer opens with an existing file, choose **File > Save** <*file*> **As**.

   The Save Reveal Analyzer File dialog box opens. Change the filename and click **Save**. You now have a new .rva file ready to work with.

3. (Stand-alone only) In the Reveal Analyzer Startup Wizard dialog box, browse to the implementation directory. This is where the Reveal Inserter project (.rvl) file should be and where the .rva file will be created.

4. In the Reveal Analyzer Startup Wizard dialog box, select **Create a new file** (at the upper-left of the dialog box).

   The dialog box presents a few rows of boxes that need to be filled in.

5. In the first row, type in the base name of the file. The extension is added automatically.

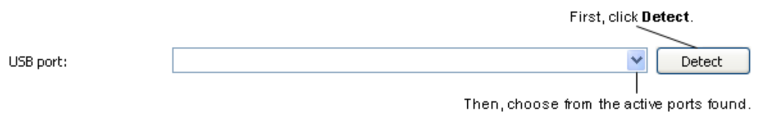
6. To the right of the first row is a drop-down menu. Choose whether your board is connected to a USB or parallel port. The next row in the dialog box changes to select the specific port.

7. Select options:

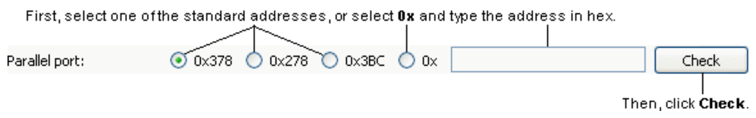
   - If there are daisy-chained devices, select **Multiple Device in JTAG Chain**.
   - If you are using the USB2A cable and want to avoid the test-logic-reset JTAG state, select **Avoid TLR State**. JTAG chains that require the assertion of TRST can get out of sync when using Reveal.

8. Select the specific port. The method depends on the port type:

   - If USB, click **Detect**. Then choose from the active ports found. The following figure shows the second row after choosing USB.



- If Parallel, select the port address. If it's not one of the standard addresses given, select **0x** and type in the hexadecimal address. Then click **Check** to verify that the connection is working. The following figure shows the second row after choosing Parallel.


First, select one of the standard addresses, or select 0x and type the address in hex.
Parallel port:
0x378
0x278
0x3BC
0x
Check
Then, click Check.

9. If there are daisy-chained devices, click **Browse** in the XCF source row to find the XCF source file.
10. Click **Scan** to find the FPGA.
11. If there is more than one FPGA on your board, go to the "Debug device" menu and choose one that has a Reveal icon. The icon indicates the presence of a Reveal module.
12. Click **Browse** in the RVL source row to find the Reveal Inserter project (.rvl) file.
13. Click **OK**.

Starting with an Existing File

If you want to start with an existing file, you just need to have that .rva file in the design project. You need to be connected to the evaluation board only if you want to run a test and capture data.

To start Reveal Analyzer with an existing file:

1. Issue the start command. To start:

   - In the Diamond main window, choose **Tools > Reveal Analyzer**.
   - The stand-alone Reveal Analyzer in Windows, go to the Windows Start menu and choose **Programs > Lattice Diamond Reveal > Reveal Logic Analyzer**.
   - The stand-alone Reveal Analyzer in Linux, enter the following on a command line:

     <*Reveal install path*>/bin/lin/**rvamain**

   If Reveal Analyzer finds just one .rva file in the active implementation, Reveal Analyzer opens with the data from that file. Otherwise, the Reveal Analyzer Startup Wizard dialog box appears.

   If Reveal Analyzer opens with the .rva file you want to use, you're ready to go. Otherwise continue with the following steps.

2. (Stand-alone only) In the Reveal Analyzer Startup Wizard dialog box, browse to the implementation directory. This is where the Reveal Inserter project (.rvl) file and where the .rva file should be.

3. In the Reveal Analyzer Startup Wizard dialog box, select **Open an existing file** (in the lower part of the dialog box).
4. In the “File name” box, choose one of the available .rva files.
5. If the file you want is not in the menu, click **Browse** and browse to the desired .rva file.
6. Click **OK**.

If the connection to your evaluation board has changed, either in the cable type or the computer port used, you need to tell Reveal Analyzer about the new connection. See “Changing the Cable Connection” on page 50.

Changing the Cable Connection

If you need to change how your evaluation board is connected to your computer, go ahead and make the change. Then go through the following procedure to change the Reveal Analyzer project.

To change the cable setting in a Reveal Analyzer project:

1. Make sure your evaluation board is connected and that its power is on.
2. If Reveal Analyzer is not already open, start it as described in “Starting with an Existing File” on page 49.
3. Choose **Design > Cable Connection Manager**.

   The Cable Connection Manager dialog box opens.
4. In the dialog box, choose the cable type: USB or parallel.

   The second row in the dialog box changes to select the specific port.
5. Select the specific port. The method depends on the port type:
   - If USB, click **Detect**. Then choose from the active ports found.
   - If Parallel, select the port address. If it’s not one of the standard addresses given, select **0x** and type in the hexadecimal address. Then click **Check** to verify that the connection is working.
6. Click **OK**.
7. To change the clock speed of the cable connection, choose **Tools > Options** and in the Reveal Analyzer section change the TCKDelay value. Note that the Reveal Analyzer tool must be open for this value to be saved.

Creating a New Reveal Analyzer Project

In order to create a new Reveal Analyzer project, you must first have available a design directory with a Reveal project (.rvl) file generated by Reveal Inserter.

To open a new Reveal Analyzer project:

1. Choose **File > New > File** or click in the toolbar and choose **File** from the drop-down menu.

   The New File dialog box appears.

2. Under Categories, choose **Other Files**.

3. Under Source Files, choose **Reveal Analyzer Files**.

4. Type in the base name for the .rva file. The ".rva" extension is added automatically.

5. If you do not want the file to be in the design project's folder, click **Browse** and browse to the desired location.

   It is recommended that you save the Reveal Analyzer project files in the project directory for the design into which the debug logic has been inserted.

6. Click **New**.

   The Reveal Analyzer Startup Wizard opens.

7. Type in the base name of the file. The extension is added automatically.

8. From the menu, choose the port that is connected to your board and click **Scan** to find the Reveal module.

9. Browse to the Reveal Inserter project (.rvl) file.

10. If you will be programming devices in a JTAG daisy chain, choose the name of the .xcf file from the drop-down menu in the **XCF source** box. See "Programming and Debugging Devices in a Daisy Chain" on page 51.

    The .xcf file must reside in the design project directory for Reveal Analyzer to use it.

    Note

    Only one device can be debugged in a JTAG daisy chain.

11. Click **OK**.

    On the basis of the settings in the .rvl file, Reveal Analyzer now verifies that the project directory contains the correct design .hub file and creates the new project.

Programming and Debugging Devices in a Daisy Chain

For an evaluation board using multiple FPGAs in a JTAG daisy chain, Reveal Analyzer has the following requirements:

- Only one device can be debugged at a time.
- The .xcf file must be in the design project directory.

To program an FPGA with a Reveal module in a daisy chain:

1. In Programmer, uncheck the Process column of the devices that do not get the Reveal module. This sets the operation of these devices to Bypass mode.
2. Ensure that the Process column of the device that does get the Reveal module is checked.
3. Double-click in the row of the device that does get the Reveal module.

   The Device Properties dialog box opens.
4. In the Device Properties dialog box, choose **Fast Program** from the drop-down menu.
5. Click **OK**.

For an FPGA that contains its own SPI flash, flash programming is also an option.

Opening an Existing Reveal Analyzer Project

To open an existing project, you must have available a Reveal Analyzer project (.rva) file from a previous Reveal Analyzer session.

Opening an Existing Project

You can open an existing project in Reveal Analyzer by using the File menu to open a dialog box.

To open an existing Reveal Analyzer project:

1. Choose **File > Open > File**.
2. In the Open File dialog box, browse to the desired .rva file.
3. Click **Open**.

Opening a Recently Opened Project

If the desired .rva file has been recently opened, you can open it directly from the File menu.

To open a recently opened .rva file directly from the File menu:

- Choose **File > Recent Files >** <*filename*> from the list of the four most recently opened files near the bottom of the File menu.

Selecting a Reveal Analyzer Core

After you have created a project, each Reveal core in the design will have a Reveal Analyzer window available to set triggers and view captured data.

Note

The darker gray background in certain fields in the LA Trigger tab indicates that you can change these values only in Reveal Inserter. You cannot change them in Reveal Analyzer.

To display a Reveal Analyzer core:

▶ Choose the core from the drop-down menu in the Reveal Analyzer toolbar.

Setting Up the Trace Signals

Although you can add trace signals only in Reveal Inserter, you can set radixes for them by using the LA Waveform tab.

Setting the Trace Bus Radix

You can set the radix of a trace bus displayed in the LA Waveform tab. You can choose a binary, octal, decimal, or hexadecimal radix. You can also use any token set whose bit width matches the bus.

To set the bus radix of a signal bus:

1. In the LA Waveform tab, select one or more buses.

   To select one bus, click on it. To select more than one, Control-click on each one. To select all buses in a range, click on one end of the range and Shift-click the other end. If you want to change all the signals in the waveform to the same radix, you do not need to select anything.

2. Right-click in one of the selected waveforms and choose **Set Bus Radix**. Be careful to click in the same row as one of your selections, or you will change the selection.

   The Set Bus Radix dialog box opens.

3. In the drop-down menu, choose the radix or token set.

4. In the Range drop-down menu, choose **Selected signals** or, if you want to change all the signals in the waveform to the same radix, choose **All signals**.

5. Click **OK**.

Adding Time Stamps to Trace Samples

In the Reveal Inserter, you can optionally specify a sample clock count value to be stored with each trace sample to indicate the sample count clock value at which the sample was captured. This count is extra data (bits) captured into the trace buffer that increase the trace buffer's width. This time stamp enables you to see how many sample clock intervals have elapsed between data captures when you use a sample enable. It is useful in some cases when it is necessary to know if you captured the right data. A time stamp is also useful when you try to synchronize data between multiple cores, off-chip data, or both. For example, if you trigger two cores at the same time, you can use the time stamps on the trace samples to calculate how the data between the cores compares.

See Adding Time Stamps to Trace Samples in the Reveal Inserter online Help for information on adding time stamps to trace samples in Reveal Inserter.

Setting Up the Trigger Signals

Before you perform logic analysis, you must define the conditions under which the trigger will start or stop the collection of data on the trace signals specified in Reveal Inserter. You must define these triggers for each core. Use the LA Trigger tab of the Reveal Inserter window to specify the trigger units and trigger expressions that start the collection of the sample data for the selected core. In Reveal Analyzer, you cannot add or remove new trigger units or trigger expressions, but you can change the values and operators in the trigger units and trigger expressions. In addition, you can disable a trigger expression from being used by clearing the checkbox to the left of the trigger expression name. You must make sure in Reveal Inserter that all signals that you might want to trigger on are included in the trigger units. In addition, you may want to create several trigger expressions ahead of time.

Renaming Trigger Units

You can rename a trigger unit.

To rename a trigger unit:

- Click in the appropriate box in the Name column of the Trigger Unit section of the LA Trigger tab, backspace over the existing name, and type in the new name.

Setting Up Trigger Units

All signals for a trigger unit must be defined in Reveal Inserter. You cannot change them in Reveal Analyzer.

To set up a trigger unit:

1. If you want to change the default name of the trigger unit, backspace over the default name in the Name column and type the new name.
2. If you want to add, change, or remove the signals in the Signals (MSB:LSB) column, you must add, change, or remove them in Reveal Inserter. You cannot add, change, or remove signals in Reveal Analyzer.
3. In the **Operator** column, set the comparators for the trigger condition. You can choose from the following states:
   - == equal to
   - != not equal to
   - > greater than
   - >= greater than or equal to
   - < less than
   - <= less than or equal to
   - Rising edge – compares on the rising edge of the clock
   - Falling edge – compares on the falling edge of the clock
   - Serial compare – compares until the trigger condition is met. For example, if the trigger condition is 10011, the serial compare option looks for a 1 on the first clock, a 0 on the next clock, a 0 on the next clock, a 1 on the next clock, and a 1 on the last clock. Only if those five conditions are met in those five clock cycles will the serial compare output be active.

     Other operators cannot be changed to a serial compare, and a serial compare cannot be changed to another operator in Reveal Analyzer. These can only be changed in Reveal Inserter.

     The serial comparator is available only when a single signal is listed in the Trigger Unit signal list. If you choose this option, you must choose Binary in the Radix box.

   The default comparator is == (equal to).
4. In the **Radix** column, select a radix from the drop-down menu to set the radix of the trigger bus value given in the Value box. You can choose one of the following:
   - Binary. This is the default. You must choose Binary if you selected “Serial compare” as a comparator.
   - Octal
   - Decimal
   - Hexadecimal
   - <*token_set_name*>. To select <*token_set_name*>, you must have created token sets in Reveal Inserter. See the Reveal Inserter online Help for information on this procedure.
5. In the **Value** column, enter the comparison value.

This value is the pattern of highs and lows that you want on the trigger unit that will initiate collection of the trace data. The default is binary, unless you selected <*token_set_name*> in the Radix column.

If you selected <*token_set_name*> in the Radix column, a drop-down menu opens in the Value column. This menu lists all the tokens that you entered in the Token Manager dialog box for the chosen token set. Select any name.

You can use ”x” for a don’t-care value in the Value column if you selected Binary, Octal or Hexadecimal in the Radix column and if you selected the ==, !=, or serial compare operators in the Operator column.

Renaming Trigger Expressions

You can rename a trigger expression.

To rename a trigger expression:

- Click in the appropriate box in the Name column of the Trigger Expression section of the LA Trigger tab, backspace over the existing name, and type in the new name.

Setting Up Trigger Expressions

You must set up the initial trigger expressions in Reveal Inserter, but you can change their names and some values in Reveal Analyzer. You can also enable or disable trigger expressions in Reveal Analyzer. However, you cannot change the sequence depth, the maximum sequence depth, or the maximum event counter of the trigger expressions in Reveal Analyzer.

To set up a trigger expression:

1. To enable a trigger expression, click the checkbox in the **Enable** column.
2. In the **Expression** box, enter the names of the trigger units that you want to use and the operators that you want to use to connect them.

   You can use the following operators to connect trigger units:

   - & (AND)
   - | (OR)
   - ^ (XOR)
   - ! (NOT)
   - Parentheses
   - THEN
   - NEXT
   - # (count)
   - ## (consecutive count)

See “Trigger Expressions” on page 20 for information on these operators.

Reveal Analyzer checks the syntax and displays the syntax in red font if it is erroneous.

The setting in the **Sequence Depth** box is set by the software, so it is read-only. See “Trigger Expressions” on page 20 for more information on this parameter.

3. If you want to change the setting in the **Max Sequence Depth** box, you must change it in Reveal Inserter; you cannot change it in Reveal Analyzer. The number of sequences in the Trigger Expression box cannot exceed the number specified in the Max Sequence Depth box.
4. If you want to change the setting in the **Max Event Counter** box, you must change it in Reveal Inserter; you cannot change it in Reveal Analyzer.
5. Specify whether the final trigger occurs when one or all of the conditions specified by the trigger expressions is met before trace data is captured:
   - AND All indicates that the conditions specified by all the trigger expressions must be met before the trace data is captured.
   - OR All indicates that the conditions of one of the trigger expressions must be met before the trace data is captured. This option is the default.

   Only trigger expressions whose checkboxes are enabled are included in the AND or OR.

Setting Trigger Options

You can set a number of options to control the triggering.

To set trigger options in Reveal Analyzer:

1. If you want to add a counter to the output of the final trigger, do the following:
   a. Select the **Final Event Counter** checkbox in the lower left portion of the LA Trigger tab.
   b. In the drop-down menu next to Event Counter, select the number of times that the conditions specified by AND All or OR (whichever you have selected) must occur before trace data capture begins. The lowest value of the counter is 1 time, and the maximum value is the value set in Reveal Inserter. The default value is 1 time. Leaving this option unselected is equivalent to setting the counter to a value of 1 time.
2. In the **Samples Per Trigger** box, select the number of samples to collect per trigger. The minimum is 16. The maximum is the trace buffer depth chosen in Reveal Inserter when the core is generated. The values available in the Samples Per Trigger box also change according to the number of triggers. If the number of triggers is set higher than 1, the samples per trigger multiplied by the selected number of triggers cannot exceed the trace buffer depth. Reveal Analyzer adjusts the Samples Per

Trigger value, if necessary. The default number of samples per trigger is the sample buffer depth. For example, if the sample buffer depth is 2048, the default sample per trigger is 2048.

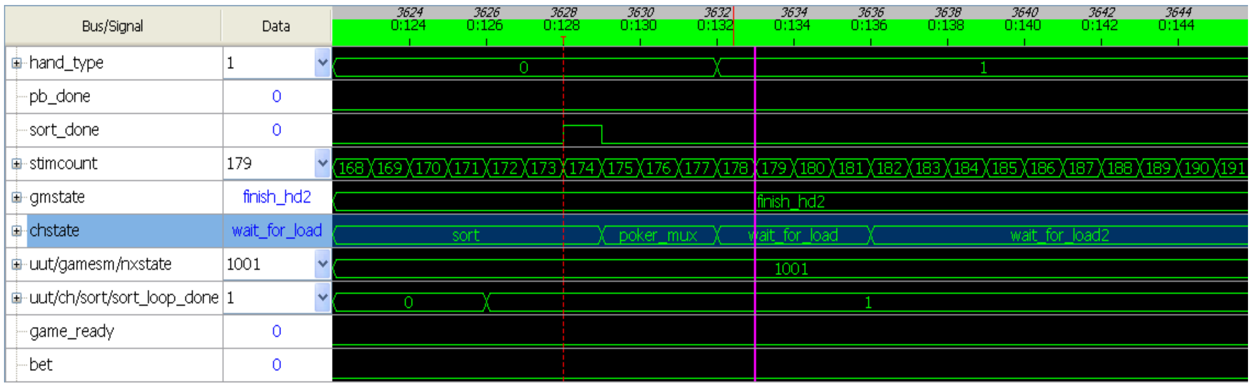
3. In the **Number of Triggers** box, select the trace buffer depth divided by the samples per trigger. The trace buffer depth is specified by the setting of the Buffer Depth parameter in the Trace Signal Setup tab in Reveal Inserter. The default is 1.

4. In the **Trigger Position** field, select **Pre-selected Position** or **User-selected Position**. See “Trigger Position” on page 75 for information on these options.

Creating Token Sets

You can create sets of “tokens,” or text labels, for values that might appear on trace buses. You can create tokens such as ONE, TWO, THREE, or Reset, Boot, Load. Tokens can make reading the waveforms in Reveal Analyzer easier and can highlight the occurrence of key values. See the following figure for an example. The row for the chstate bus uses tokens.

Figure 3: LA Waveform View Using Tokens



Bus/Signal	Data
hand_type	1
pb_done	0
sort_done	0
stimcount	179
gmstate	finish_hd2
chstate	wait_for_load
uut/gamesm/nxstate	1001
uut/ch/sort/sort_loop_done	1
game_ready	0
bet	0

To create or modify a token set:

1. Choose **Design > Token Set Manager**.

   The Token Manager dialog box opens. If the Reveal project already has token sets defined, they are listed in the dialog box.

2. If you want to use token sets that were previously saved to a separate file, right-click in the dialog box and choose **Import**. In the Import Tokens dialog box, browse to the token (.rvt) file and click **Open**.

   The token sets in the .rvt file are added to the list in Token Manager.

3. To create a new token set, right-click and choose **Add Token Set**.

   A new token set is started with default values. But it has no tokens defined yet.

4. To change the size of the token values, double-click the value in the Num. of Bits column and type in the new width, in bits. The width can be up to

256. The width must be the same as the bus that the token set will be used with.

The Num. of Bits value can only be changed when the token set is empty. If there are any tokens, you will get an error message.

5. To create a new token, select a token set. Then right-click and choose **Add Token**.

   A new token is created with default values. Repeat for as many new tokens needed.

6. You can modify token sets by doing any of the following:

   - To change the name of a token or token set, double-click the name and type a new name. The name can consist of letters, numbers, and underscores (_). It must start with a letter.
   - To change the value of a token, double-click the value and type in a new value. Token values must be prefixed by one of the radix indicators shown in the following table:

Radix	Prefix	Example
Binary	b’	b’110x0
Octal	o’	o’53
Decimal	d’	d’123
Hexadecimal	h’	h’0F2

   If a value does not have a prefix, its radix is assumed to be binary. You can use an “x” in binary numbers as a don’t-care value.

   - To remove a token or token set, select it. Then right-click and choose **Remove**.

7. You can save the collection of token sets showing in the dialog box to a separate file for use in another project. To save the token sets, right-click and choose **Export**. In the Export Tokens dialog box, browse to the desired location and type in the name of the new token (.rvt) file. Click **Save**.

8. When you are done, click the Close ❎ button to close the dialog box. The token sets are automatically applied to the current Reveal project.

Performing Logic Analysis

After you have configured trigger settings in the LA Trigger tab, you can perform a logic analysis using up to 15 cores, depending on the number of cores you created in Reveal Inserter, and view the trace buffer data in waveform format in the LA Waveform tab.

To perform logic analysis:

1. In the <*design_name*>_LA<*core_number*> check box in the toolbar, select the cores on which to perform logic analysis.
2. Click ▶ on the Reveal Analyzer toolbar.

   The Run button changes into the Stop ■ button and the status bar next to the button shows the progress.

Reveal Analyzer first configures the cores selected for the correct trigger condition, then waits for the trigger conditions to occur. Once the specified trigger has occurred, the data is downloaded to the PC. The resulting waveforms appear in the LA Waveform tab.

If the trigger condition is not met, Reveal Analyzer will continue running. In that case, you can use manual triggering, described in “Using Manual Triggering” on page 62.

Data Capture with Sample Enable

Triggers occur at every sample clock edge when the condition is met. Trace data is also captured on the sample clock edge. If a sample enable is used, each sample shown in the trace buffer is only captured when the sample enable is active and there is a sample clock. Data samples can be discontiguous, unlike those in a normal data capture.

It is also possible that the actual trigger condition may occur when the sample enable is not active, causing two changes from a normal data capture:

- The actual data values for the trigger condition may not be visible, because the data cannot be captured when the sample enable is inactive.
- Reveal Analyzer cannot accurately calculate the trigger point, since the trigger point may have occurred when the sample enable was inactive. Normally a trigger point is shown as a single marker on the clock on which the trigger occurred. If a sample enable is used, a trigger region that spans five clock cycles is shown instead. Reveal Analyzer can guarantee that the trigger occurred in this region, but it cannot determine during which clock cycle the trigger occurred.

The sample enable is a very useful feature, but it takes more understanding than a normal data capture.

Common Error Conditions

Reveal Analyzer may fail to generate waveforms and instead issue an error message because of problems with the inserted core or cores.

Clock Causes Core to Malfunction

If Reveal Analyzer issues the following error message, the core may not be functional because of a clock problem:

```
Incorrect HUB ID. Check the clock or cable setup.
```

In this case, be sure the clock is running on the target hardware.

Signature in .rvl File Does Not Match Signature in Bitstream

If you have defined a token set and Reveal Analyzer issues an error message similar to the following, the core may be functional, but the signature in the .rvl file does not match the signature in the bitstream:

```
Incorrect core signature 1234 from core(core1), expected
signature is 1235.
```

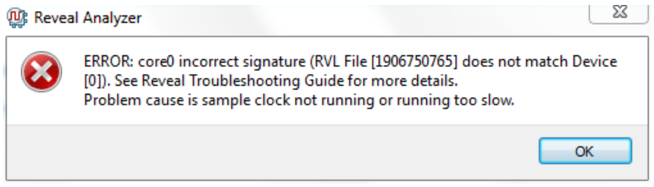
This problem can be caused by saving the Reveal Inserter project file, even if you have made no changes, without re-implementing the design.

If you receive this message, regenerate the bitstream or JEDEC file, or download another file whose signature matches that of the .rvl file.

Sample Clock Runs Too Slowly

The sample clock is used by Reveal debug logic to clock data into the trace buffer and in the triggering logic. The sample clock is also needed when Reveal Analyzer communicates with the debug logic through JTAG. If the sample clock is not running or is running too slow, Reveal Analyzer cannot detect that the Reveal debug logic is available. This information is especially important when you create a new Reveal Analyzer project. Reveal Analyzer checks the debug logic for a signature to make sure that the bitstream matches the design. If Reveal Analyzer cannot communicate with the debug logic because the sample clock is not running, the project creation or the Reveal Analyzer run command will fail with an error, and Reveal Analyzer issues an error message similar to that shown in Figure 4. For these reasons, the sample clock should be a signal with a reasonably regular frequency rather than a signal with intermittent pulses. The frequency of the sample clock should also be faster than the speed of the JTAG clock that is used.

Figure 4: Reveal Analyzer Sample Clock Error Message



ERROR: core0 incorrect signature (RVL File [1906750765] does not match Device [0]). See Reveal Troubleshooting Guide for more details.
Problem cause is sample clock not running or running too slow.

Performing Logic Analysis on Multiple Devices

You can perform logic analysis on one device in a scan chain of multiple devices on a board or multiple boards. The maximum number of boards is specified by Programmer.

When you perform logic analysis on a device in a scan chain, you must generate a scan chain configuration (.xcf) file. See the Programmer online Help for instructions on generating this file.

Stopping a Logic Analysis

You can stop a logic analysis while it is running.

To stop a logic analysis:

- Click .

This command only stops the logic analysis on the current core in the active window. You must stop each core separately.

Using Manual Triggering

If you set up a trigger but triggering fails to occur or you want to trigger manually instead of triggering when a signal condition occurs, you can use manual triggering to capture data. The captured data may then help you find out why triggering did not occur as you originally intended.

When you select manual triggering, Reveal Analyzer fills the buffer with data captured from that moment. In single-trigger capture mode, it fills the buffer and stops. In multiple-trigger capture mode, it captures one trigger and data. You can then continue to manually trigger as many times as the original triggering setup specified. If you want to capture fewer triggers, you can manually trigger the desired number of times, then press the Stop () button to stop the logic analysis. The buffer starts downloading the data.

To use manual triggering:

Note

A logic analysis must be running before you can use the Manual Trigger command.

1. After you start the logic analysis with the button, click .

   This command only applies to the logic analysis on the current core in the active window. You must trigger each core separately.

2. When you have captured the desired number of triggers in multiple trigger capture mode, click .

Viewing Waveforms

After running your logic analysis, you can view the trace buffer data in waveform format in the LA Waveform tab. Whenever the trace stops, Reveal Analyzer reads the trace samples back from the trace memory and automatically updates the signal waveforms.

If you perform a logic analysis, exit Reveal Analyzer, and then reopen it, the old data is displayed in the waveform until you perform a new logic analysis.

Viewing Logic Analyses

To view a logic analysis:

1. Click the **LA Waveform** tab.
2. Choose a module from the drop-down menu in the Reveal Analyzer toolbar.

Adjusting the Waveform Display

You can adjust the waveform display by panning and zooming. You can also adjust the colors.

Panning

You can move the waveform display in the LA Waveform tab so that you can view any part of it.

To pan the waveform display:

1. Right-click in the waveform and choose **Pan Mode**.
2. Press and drag the left mouse button to the left or the right.

Zooming In and Out

You can zoom in and out on a waveform in the Reveal Analyzer LA Waveform tab to increase or decrease the displayed time interval.

To zoom in on a waveform:

▶ Choose **View > Zoom In**, click 🔍, or right-click on the waveform and choose **Zoom > Zoom In**.

To zoom in on a specified area:

1. Right-click the waveform and choose **Zoom Mode**.

   The pointer changes to a cross: +.

2. Hold down the left mouse button and drag the pointer across the area you want to zoom in on.

   A shaded area appears on the waveform display.

3. Release the mouse button.

   The shaded area expands to fill the display.

To zoom out on a waveform:

▶ Choose **View > Zoom Out**, click , or right-click on the waveform and choose **Zoom > Zoom Out**.

To show the entire waveform in the window:

▶ Choose **View >** **Zoom Fit**.

To zoom to the trigger point:

▶ Right-click in the waveform and choose **Zoom > Zoom Trigger**

To zoom to the start of the display:

▶ Right-click in the waveform and choose **Zoom > Zoom Start**

To zoom to the end of the display:

▶ Right-click in the waveform and choose **Zoom > Zoom End**

Setting a Trace Bus Radix

You can set the radix of a trace bus displayed in the LA Waveform tab. You can choose a binary, octal, decimal, or hexadecimal radix. You can also use any token set whose bit width matches the bus.

To set the bus radix of a signal or bus:

1. In the LA Waveform tab, click in the Data cell of the signal or bus.

   A menu appears showing the different radices and any token sets that fit.

2. Choose the desired radix or token set.

To set the bus radix of multiple signals and buses:

This method can set several signals to the same radix but cannot use tokens.

1. In the LA Waveform tab, select one or more buses.

To select one bus, click on it. To select more than one, Control-click on each one. To select all buses in a range, click on one end of the range and Shift-click the other end. If you want to change all the signals in the waveform to the same radix, you do not need to select anything.

2. Right-click in one of the selected waveforms and choose **Set Bus Radix**. Be careful to click in the same row as one of your selections, or you will change the selection.

   The Set Bus Radix dialog box opens.

3. In the drop-down menu, choose the radix.
4. In the Range drop-down menu, choose **Selected signals** or, if you want to change all the signals in the waveform to the same radix, choose **All signals**.
5. Click **OK**.

Changing LA Waveform Colors

You can change the colors used by the waveform.

To change the colors:

1. Open the Options dialog box. Depending on which version of Reveal Analyzer you're using, do one of the following:
   - If integrated with Diamond, choose **Tools > Options**. Then, in the Options dialog box, choose **Reveal Analyzer > Colors**.
   - If stand-alone, choose **Design > Options**.
2. Click on the color sample for the desired part of the LA Waveform view.

   The Select Color dialog box opens.

3. Select a color.
4. In the Select Color dialog box, click **OK**.
5. To see the effect of the change, click **Apply**.
6. Change other colors if desired.
7. Click **OK**.

Specifying the Clock Period

You can specify a clock period for your logic analysis. Setting the frequency enables you to determine the location of your cursors, as well as the distance between them. Frequency is determined by dividing 1 by the period.

To set the clock frequency:

1. Right-click the waveform and choose **Set Clock Period**.

2. In the Specify Clock Period dialog box, choose either picoseconds or nanoseconds for the period interval selector. Click the box next to Period to select picoseconds (ps) or nanoseconds (ns).
3. Place the cursor in either the Period or Frequency text box and type in the desired value. The other text box fills in automatically.

   Only integers are allowed. If you try to specify a frequency that would require a non-integer period, the period is truncated to an integer and the frequency is automatically adjusted. For example, typing 150 in the Frequency text box gives you a period of 6 and a frequency of 166.
4. Click **OK**.

Placing, Moving, and Locating Cursors

The LA Waveform view comes with three types of “cursors” to highlight moments in the waveform. The cursors are vertical lines cutting through all the signals at the leading edge of a clock cycle. See Figure 6 on page 77. The three types are:

- Trigger. A purple line with a “T” at the top, trigger cursors are automatically placed at the moment of each final trigger event. If the module used a sample enable signal and the exact moment of the trigger is unknown, the waveform shows a trigger cursor five clock cycles before the sample enable signal turned inactive and sampling stopped.
- Active. A red line appears wherever you click in the waveform. The Data column shows the values of the signals and buses at the moment highlighted by the active cursor.
- User. A blue line can be placed anywhere you want. Use these cursors to mark moments of interest. You can also use these cursors to maneuver about a long waveform with the Go to Cursor command.

Most cursor functions require that the LA Waveform view be in Select mode: right-click in the LA Waveform view and choose **Select Mode**.

To create a user cursor:

1. Click in the desired clock cycle.

   The active cursor appears. Make sure it is where you want the user cursor to be.
2. Right-click and choose **Add Cursor**.

To move a user cursor:

1. Zoom in so you can easily see and click in individual samples.
2. Click in the desired location.

   The active cursor appears. Make sure it is where you want the user cursor to be.
3. Carefully click in the sample to the right of the user cursor.

You must click on or to the right of the user cursor. Otherwise you are just moving the active cursor to a neighboring sample.

The user cursor and the active cursor exchange locations.

To jump to a user cursor:

- Right-click in the waveform and choose **Go to Cursor >** <*cursor*>. Cursors are identified by the sample index as shown in the green bar at the top of the waveform display.

To remove a user cursor:

1. Click on or near the cursor.

   The active cursor appears. Make sure it is on or next to the user cursor you want to remove.

2. Right-click and choose **Remove Cursor**.

To remove all user cursors:

- Right-click in the waveform and choose **Clear All Cursor**.

Counting Samples

You can easily count the number of samples in a range on the display.

To count samples:

- Click where you want to start counting and drag to the end of the range.

  While you're dragging, the LA Waveform view shows two red lines and the number of samples between the lines.

Exporting Waveform Data

You can export waveform data to a value change dump (.vcd) file, which can be imported by such third-party tools as ModelSim or Active-HDL, or to an ASCII-format text (.txt) file. You must have performed a logic analysis, implemented a trigger on hardware, and captured data that is shown in the waveform display before you can export data.

To export data:

1. Choose the module from the drop-down menu in the Reveal Analyzer tool bar.
2. If you want the data to include an approximate measure of time instead of a simple count of clock cycles, right-click the waveform and choose **Set Clock Period**. See "Specifying the Clock Period" on page 65.

3. If you want to export only some of the signals, select them in the waveform. You can only export whole buses. If you select only some of the signals in a bus, you get the whole bus.
4. Right-click in the waveform and choose **Export Waveform**.

   The Export Waveform dialog box opens.
5. Browse to the location where you want to export the file.
6. Type in a name in the **File name** box.
7. Choose a file type.
8. If you are exporting only some of the signals, choose **Selected signals** in the Range box.
9. If you are exporting to .vcd, type in a module name. This will form the title in the .vcd file. If you leave the field empty, the module name will be "<unknown>".
10. Click **Save**.

Saving a Project

You can save the trigger settings and waveform setup settings in a Reveal Analyzer project (.rva) file that you can use as an input file in the future. You can also save an existing .rva file in a file with a different name.

To save a Reveal Analyzer project:

- Choose **File >** **Save** <*file*>.

The project data is now output into an .rva file.

To save the project file with a different name:

1. Choose **File > Save** <*file*> **As**.

   The Save Reveal Analyzer File dialog box appears.
2. Browse to the directory in which you want to save the project.
3. In the File name box, type the file name.
4. Click **Save**.

Controlling Serdes

Reveal Analyzer provides two methods for controlling serdes functions in an ECP5UM device. You can use:

- A Serdes Debug module created with Reveal Inserter to experiment with control registers and monitor system status
- A Wishbone bus in the design to set and monitor register values

Setting Serdes Debug Registers

You can adjust a large variety of serdes settings while running tests. Settings can be changed individually or by importing previously saved settings. You can also restore the design's original values from the configuration SRAM.

You can also monitor the following items in the Serdes Debug tab (green for good, red for bad):

- PLL lock status
- Receive input signal status
- CDR loss status

To set the serdes registers:

1. Click the **Serdes Debug** tab.
2. Click the DCUA drop-down menu and choose the block.
3. Click the Channel drop-down menu and choose the channel.
4. Change settings as desired.
5. Click **Apply**.

   The new values are immediately written to the FPGA.

To use previously saved settings:

1. Click the **Serdes Debug** tab.
2. Click the DCUA drop-down menu and choose the block.
3. Click the Channel drop-down menu and choose the channel.
4. Click **Import**.
5. In the Import SERDES File dialog box, browse to the desired serdes register (.srv) file.
6. Click **Open**.

   The file's settings are appear in the Serdes Debug tab.
7. Click **Apply**.

   The new values are immediately written to the FPGA.

To restore the design's original values:

1. Click the **Serdes Debug** tab.
2. Click the DCUA drop-down menu and choose the block.
3. Click the Channel drop-down menu and choose the channel.
4. Click **Config SRAM Reload**.

   The original settings are appear in the Serdes Debug tab.
5. Click **Apply**.

   The new values are immediately written to the FPGA.

Saving Serdes Debug Settings

You can save your Serdes Debug settings for later use.

To save the Serdes Debug settings:

1. In the Serdes Debug tab, click **Export**.

   The Export SERDES Register File dialog box opens.
2. Browse to where you want to save the file.
3. Type in a name in the **File name** box.
4. Click **Save**.

Reading and Writing with a Wishbone Bus

If you have a Wishbone bus in your design, you can write directly into the serdes registers, manually or using a file, and read values from them.

To write to a serdes register:

1. Click the **Wishbone Debug** tab.
2. Enter the address of the register.
3. In the Data box, type a 1-byte value.
4. Click **Write**.

   The value in the Data box is written to the register.

To write to several serdes registers from a file:

1. Click the **Wishbone Debug** tab.
2. Click **Load**.

   The Load SERDES File dialog box opens.
3. Browse to the desired Tcl file.
4. Click **Open**.

   The addresses and values defined in the file are written to the FPGA.

To read a serdes register:

1. Click the **Wishbone Debug** tab.
2. Enter the address of the register.
3. If you want to monitor the value in the register continuously, select **Continuous**.
4. Click **Read**.

The value in the register appears in the Data box. If you selected Continuous, the value changes as the register changes.

Exiting Reveal Analyzer

To exit Reveal Analyzer:

- Choose **File** > **Close**.

User Interface Descriptions

The Reveal Analyzer window consists of the LA Trigger, LA Waveform, Serdes Debug, and Wishbone Debug tabs.

LA Trigger Tab

The LA Trigger tab enables you to select the trigger signals and define the data values or pattern of data values that cause trace data collection to begin.

Trigger Unit

The parameters in the Trigger Unit section enable you to configure the trigger units, which are the basic trigger comparison mechanism in Reveal Inserter and Reveal Logic Analyzer. Trigger units allow comparison of the signal to a value that is entered during hardware debug. You can include up to 16 trigger units in a core. Each trigger unit consists of the following information:

Note

The darker gray background in certain fields in the Trigger Signal Setup tab indicates that you can change these values only in Reveal Inserter. You cannot change them in Reveal Logic Analyzer.

Name Specifies the name of the trigger unit. See “Trigger Expression and Trigger Unit Naming Conventions” on page 76 for the guidelines governing trigger unit names. The default name is TU<*number*>, where <*number*> is a sequential number. The first trigger unit is named TU1 by default.

Signals Lists the signals in the trigger unit. You can select up to 4096 trigger signals. You can specify these signals only in Reveal Inserter.

Operator Specifies the comparators that Reveal will use to compare the states of the trigger bus signals to the pattern of signal states that you set in the Trigger Signal Setup tab of Reveal Logic Analyzer. You can choose from the following states:

- == equal to. This comparator is the default.
- != not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to
- Rising edge – Compares on the rising edge of the clock
- Falling edge – Compares on the falling edge of the clock
- Serial compare – Compares until the trigger condition is met. For example, if the trigger condition is 10011, the serial compare option looks for a 1 on the first clock, a 0 on the next clock, a 0 on the next clock, a 1 on the next clock, and a 1 on the last clock. Only if those five conditions are met in those five clock cycles will the serial compare output be active.

  Other operators cannot be changed to a serial compare, and a serial compare cannot be changed to another operator in Reveal Logic Analyzer. These can only be changed in Reveal Inserter.

  The serial comparator is available only when a single signal is listed in the Trigger Unit signal list. If you choose this option, you must choose Binary in the Radix box.

Radix Specifies the radix of the trigger bus. It can be one of the following:

- Binary. This is the default. You must choose Binary if you selected “Serial compare” as a comparator.
- Octal
- Decimal
- Hexadecimal
- <*token_set_names*>. To select <*token_set_name*>, you must have created token sets in Reveal Inserter. See the Reveal Inserter online Help for information on this procedure.

Value Specifies the comparison value. This value is the pattern of highs and lows that you want on the trigger unit that will initiate collection of the trace data. The default is binary.

If you selected a <*token_set_name*> in the Radix column, a drop-down menu opens in the Value column. This menu lists all the tokens that you entered in the Token Manager dialog box for the chosen token set. Select any name.

You can use “x” for a don’t-care value in the Value column if you selected Binary, Octal or Hexadecimal in the Radix column and if you selected the ==, !=, or serial compare operators in the Operator column.

Trigger Expression

The parameters in the Trigger Expression section of the Trigger Signal Setup tab enable you to configure the trigger expressions, which are combinatorial, sequential, or both combinatorial and sequential equations of trigger units that define when the collection of the trace data samples begins. You can add up to 16 trigger expressions. Each trigger expression consists of the following information:

Enable Determines whether the trigger expression is active when the triggering is enabled by the Run button.

Name Specifies the name of the trigger expression. The default name is TE<*number*>, where <*number*> is a sequential number. The first trigger expression is named TE1 by default.

Expression Specifies the trigger units and the operator or operators that indicate the relationship of one trigger unit to other trigger units. You can use the following operators to connect trigger units:

- & (AND) – Combines trigger units using an & operator.
- | (OR) – Combines trigger units using an OR operator.
- ^ (XOR) – Combines trigger units using a XOR operator.
- ! (not) – Combines a trigger unit with a NOT operator.
- Parentheses – Groups and orders trigger units.
- THEN – Creates a sequence of wait conditions. For example, the following statement:

  ```
  TU1 THEN TU2
  ```

  means “wait for TU1 to be true,” then “wait for TU2 to be true.”

  The following expression:

  ```
  (TU1 & TU2) THEN TU3
  ```

  means “wait for TU1 and TU2 to be true, then wait for TU3 to be true.”

  Reveal supports up to 16 sequence levels.
- NEXT – Creates a sequence of wait conditions, like THEN, except the second trigger unit must come immediately after the first. That is, the second trigger unit must occur in the next clock cycle after the first trigger unit.
- # (count) – Inserts a counter into a sequence. Sequences are groups of combinatorial operations connected by THEN operators. The counter counts how many times a sequence must occur before a THEN statement. The maximum value of this count is determined by the Max Event Counter value. It must be specified in Reveal Inserter and cannot be changed in Reveal Logic Analyzer.

  Here are some examples.

  The following statement:

  ```
  TU1 #5 THEN TU2
  ```

means that TU1 must be true for five consecutive or non-consecutive sample clocks before TU2 is evaluated. The counts do not have to be sequential. TU1 does not have to be true five times in a row to satisfy this condition. It only has to be true fives times to meet this condition.

The next expression:

```
(TU1 & TU2)#2 THEN TU3
```

means “wait for the second occurrence of TU1 and TU2 being true, then wait for TU3.”

The last expression:

```
TU1 THEN (1)#200
```

means “wait for TU1 to be true, then wait for 200 sample clocks.” This expression is useful if you know that an event occurs a certain time after a condition.

You can only use one count (#) operator per sequence. For example, the following statement is not valid, because it uses two counts in a sequence:

```
TU1 #5 & TU2 #2
```

- ## (consecutive count) – Inserts a counter into a sequence. Like # (count) except that the trigger units must come in consecutive clock cycles. That is, one trigger unit immediately after another with no delay between them.

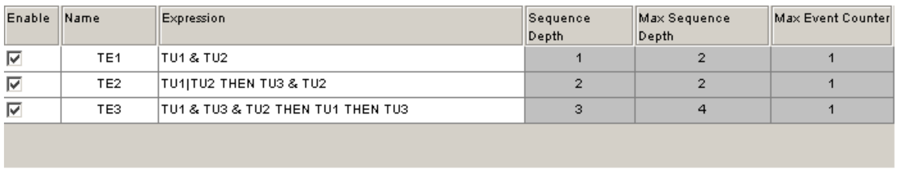
Sequence Depth Specifies the number of sequences, which are groups of combinatorial operations connected by THEN operators, used in a trigger expression. Reveal supports up to 16 sequence levels.

For example, in the following figure, TE1 consists of one sequence, since it has no THEN operator. It therefore has a sequence depth of 1. TE2 has two sequences, TU1|TU2 and TU3 & TU2, linked by a THEN operator, so its sequence depth is 2. TE3 has three sequences:

- TU1 & TU3 & TU2 followed by THEN
- TU1 followed by THEN
- TU3

TE3 therefore has a sequence depth of 3.

Figure 5: Trigger Expression Sequences



Enable	Name	Expression	Sequence Depth	Max Sequence Depth	Max Event Counter
✓	TE1	TU1 & TU2	1	2	1
✓	TE2	TU1|TU2 THEN TU3 & TU2	2	2	1
✓	TE3	TU1 & TU3 & TU2 THEN TU1 THEN TU3	3	4	1

Max Sequence Depth Specifies the maximum number of sequences, or trigger units connected by THEN operators, that can be used in a trigger expression. You can set this option only in Reveal Inserter.

Max Event Counter Determines the maximum size of the count in the trigger expression (the count is how many times a sequence must occur before a THEN statement). You can set this option from 1 to 64,000. The maximum is 64,000. The default is 1. You can set this option only in Reveal Inserter.

AND All Indicates that the conditions specified by all the trigger expressions must be met before the trace data is captured.

OR All Indicates that the conditions of one of the trigger expressions must be met before the trace data is captured. This option is the default.

Final Event Counter Specifies the number of times that AND All or OR all must be met before triggering occurs and trace data is captured. In effect, it adds an extra counter to the final trigger logic. The lowest value of the counter is 2 times, and the maximum value is 65536 times. The default value is 8 times. Leaving this option unselected is equivalent to setting the counter to a value of 1 time. This option is unselected by default.

Samples Per Trigger Specifies the number of samples to collect per trigger. The minimum is 16. The maximum is the trace buffer depth chosen in Reveal Inserter when the core is generated. The values available in the Samples Per Trigger box also change according to the number of triggers. If the number of triggers is set higher than 1, the samples per trigger multiplied by the selected number of triggers cannot exceed the trace buffer depth. Reveal Logic Analyzer adjusts the Samples Per Trigger value, if necessary. The default number of samples per trigger is the sample buffer depth. For example, if the sample buffer depth is 2048, the default sample per trigger is 2048.

Number of Triggers Specifies the trace buffer depth divided by the samples per trigger. The trace buffer depth is specified by the setting of the Buffer Depth parameter in the Trace Signal Setup tab in Reveal Inserter. The default is 1.

Trigger Position

Pre-selected Position Specifies the position of the trigger point in the data stream. For example, 32/512 means the trigger point is the 32nd sample in the trace memory that has a total depth of 512 samples (from sample 0 to sample 511). You can use one of the following samples in the Position box:

- Pre-Trigger – Sets the trigger point at 1/16 of the total number of samples in the trace memory. For example, when the trace memory has a total number of 512 samples, 1/16 of these would be 32. The 32 setting means that 32 data samples are collected before the trigger occurs. The Pre-Trigger setting is helpful if you are mostly interested in the data states that occurred after the trigger event. Using the example just given, only 32 samples of data (from sample 0 to sample 31) that occur before the trigger are stored, but 480 samples of data (from sample 32 to sample 511) that occur after the trigger are stored.
- Center-Trigger – Sets the trigger point at 50 percent (1/2) of the total number of samples in the trace memory. For example, when the trace

memory has a total number of 512 samples, 50 percent of these would be 256, so the trigger point would be set at 256. The Center-Trigger setting provides equal amounts of trace data before and after the trigger event. Using the example just given, 256 samples of data (from sample 0 to sample 255) that occur before the trigger are stored, and 256 samples of data (from sample 256 to sample 511) that occur after the trigger are stored.

- Post-Trigger – Sets the trigger point at 15/16 of the total number of samples in the trace memory. For example, when the trace memory has a total number of 512 samples, 15/16 of these would be 480. The Post-Trigger setting is helpful if you are mostly interested in the data states that occurred before the trigger event. Using the example just given, 480 samples of data (from sample 0 to sample 479) that occur before the trigger are stored, but only 31 samples of data that occur after the trigger are stored.

Note

The actual trigger position in the captured samples may not match the trigger position that you set in the Trigger Signal Setup tab. For example, if the trigger condition occurs before the trigger position set in the Trigger Signal Setup tab, the actual trigger position is earlier than that specified in the Trigger Signal Setup tab. Since the trigger occurred before enough samples were captured to fill the buffer, both the position is different and the total number of captured samples will be less than the set value. This condition is more likely to occur using the post-trigger setting. To avoid this, do not use a trigger condition that occurs immediately or very frequently.

User-Selected Position Enables you to choose the trigger point from certain points selected by the tool.

Trigger Position Shows the position of the trigger point in relation to the number of samples per trigger. It is in read-only notation at the very bottom of the Trigger Position section. For example, if you selected the Center-Trigger setting for the Pre-selected Position option and you selected 1024 samples in the Samples Per Trigger box, the Trigger Position field would display a trigger point equal to half the samples, 512/1024.

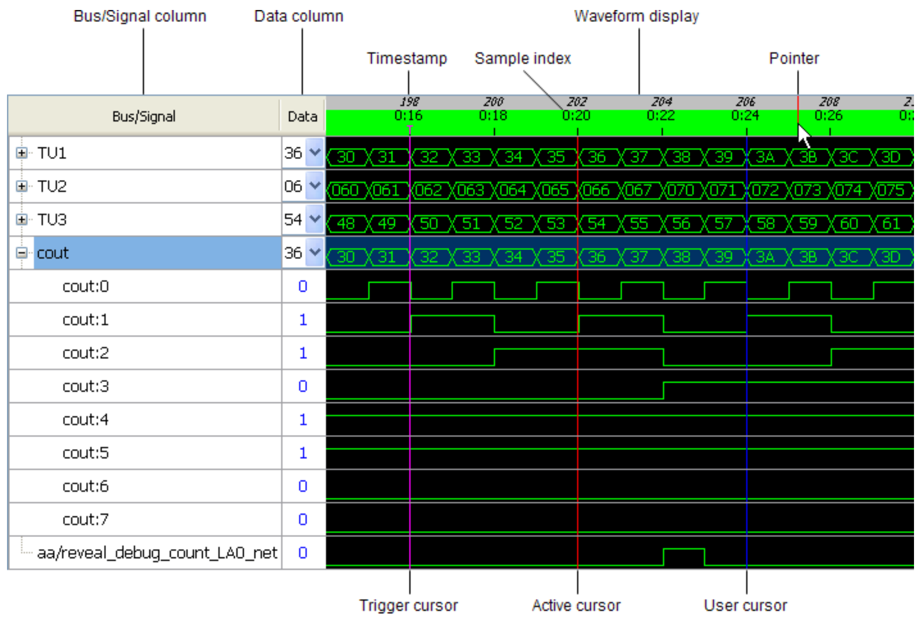
Trigger Expression and Trigger Unit Naming Conventions

You can rename trigger units and trigger expressions. The names can be a mixture of lower-case or upper-case letters, underscores, and digits from 0 through 9. The first character must be either an underscore or a letter. The names can be any length.

LA Waveform Tab

Waveforms are presented in a grid layout as shown in Figure 6 along with several features to help you find and analyze the data.

Figure 6: Elements of the LA Waveform View



Bus/Signal Column Displays the names of the trace buses and signals in the selected module.

Data Column Displays the value of the bus or signal at the active cursor (a solid, red line that you can set in the waveform display). Buses also have a drop-down menu for setting the radix used in the Data column and in the waveform display. The menu includes token sets whose bit width matches the bus. See “Setting the Trace Bus Radix” on page 53.

Waveform Display Displays the trace data in waveform format. When there is room, bus values are included the display using the radix set in the Data column. You can zoom in and out, pan, and jump to various points.

The waveform display includes several other elements to help you read the display and analyze the data:

- Timestamp. The gray bar at the top of the display shows “timestamps” of the trace frames (actually, a simple count of the clock cycles). Timestamps are shown only if the Timestamp trace option was selected for the module in Reveal Inserter. See “Adding Time Stamps to Trace Samples” on page 54.
- Sample Index. The green bar near the top of the display shows a count of triggers and trace samples within each trigger’s data set. The sample indexes have the form <*trigger*>:<*sample*>. For example, 0:2 indicates the first trigger and the third trace sample for that trigger (the counts are zero-based). 2:10 indicates the third trigger and the eleventh trace sample for that trigger.
- Pointer. A red line that cuts across the timestamp and sample index bars, the pointer follows the horizontal movement of the mouse pointer across the waveform display. Use the pointer to see where you are in time as you examine the waveform.

- Cursors. Vertical lines cutting through all the signals, cursors mark moments in the waveform. See “Placing, Moving, and Locating Cursors” on page 66.

Serdes Debug Tab

You can adjust a large variety of serdes settings while running tests on an ECP5UM device. Settings can be changed individually or by importing previously saved settings. You can also restore the design’s original values from the configuration SRAM.

You can also monitor the following items in the Serdes Debug tab (green for good, red for bad):

- PLL lock status
- Receive input signal status
- CDR loss status

Wishbone Debug Tab

If you have a Wishbone bus in an ECP5UM design, you can write directly into the serdes registers, manually or using a file, and read values from them.