

Reveal User Guide for Radiant Software



December 11, 2025

Copyright

Copyright © 2025 Lattice Semiconductor Corporation. All rights reserved. This document may not, in whole or part, be reproduced, modified, distributed, or publicly displayed without prior written consent from Lattice Semiconductor Corporation (“Lattice”).

Trademarks

All Lattice trademarks are as listed at www.latticesemi.com/legal. Synopsys and Synplify Pro are trademarks of Synopsys, Inc. Aldec and Active-HDL are trademarks of Aldec, Inc. QuestaSim is a trademark or registered trademark of Siemens Industry Software Inc. or its subsidiaries in the United States or other countries. All other trademarks are the property of their respective owners.

Disclaimers

NO WARRANTIES: THE INFORMATION PROVIDED IN THIS DOCUMENT IS “AS IS” WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF ACCURACY, COMPLETENESS, MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL LATTICE OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (WHETHER DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION PROVIDED IN THIS DOCUMENT, EVEN IF LATTICE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF CERTAIN LIABILITY, SOME OF THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

Lattice may make changes to these materials, specifications, or information, or to the products described herein, at any time without notice. Lattice makes no commitment to update this documentation. Lattice reserves the right to discontinue any product or service without notice and assumes no obligation to correct any errors contained herein or to advise any user of this document of any correction if such be made. Lattice recommends its customers obtain the latest version of the relevant information to establish that the information being relied upon is current and before ordering any products.

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.
<i><Italic></i>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
<code>Courier</code>	Code examples. Messages, reports, and prompts from the software.
<code>...</code>	Omitted material in a line of code.
<code>.</code> <code>.</code> <code>.</code>	Omitted lines in code and report examples.
<code>[]</code>	Optional items in syntax descriptions. In bus specifications, the brackets are required.
<code>()</code>	Grouped items in syntax descriptions.
<code>{ }</code>	Repeatable items in syntax descriptions.
<code> </code>	A choice between items in syntax descriptions.

Contents

Chapter 1	Introduction	7
Chapter 2	Reveal Inserter	9
	Using JTAG Debugger	9
	Debug Flows	10
	Input and Output Files	16
	Limitations	17
	Getting Started	19
	Starting Reveal Inserter	19
	Creating a New Reveal Inserter Project	20
	Opening an Existing Reveal Inserter Project	22
	Using the Reveal Controller Simulation Model	22
	Managing the Cores in a Project	23
	Renaming a Core	24
	Removing a Core	24
	Viewing Signals in the Design Tree Pane	24
	Searching for Signals	25
	Setting Up the Trace Signals	26
	Selecting the Debug Logic Core	26
	Selecting the Trace Signals	26
	Viewing Trace Signals and Buses	27
	Grouping Trace Signals into a Bus	28
	Ungrouping Trace Signals in a Bus	28
	Removing Signals and Buses from the Trace Data Pane	28
	Renaming a Bus	28
	Setting Required Sample Parameters	29
	Power-on Reset (POR) Debug	30
	Setting Sample Options	30
	Setting Up the Trigger Signals	32
	Triggering	32
	Adding Trigger Units	41
	Renaming Trigger Units	41

Setting Up Trigger Units	42
Removing Trigger Units	44
Adding Trigger Expressions	44
Renaming Trigger Expressions	45
Setting Up Trigger Expressions	45
Removing Trigger Expressions	47
Setting Up Virtual Switch/LED Settings	48
Configuring User Memory Setup	48
Configuring User Control Register Setup	50
Configuring User Status Register Setup	52
Configuring Hard IP Setup	53
Checking the Debug Logic Settings	53
Saving a Project	55
Inserting the Debug Logic Cores	55
Removing Debug Logic from the Design	56
Closing a Project	56
Exiting Reveal Inserter	56
Performing Logic Analysis with Reveal Analyzer	57
Using JTAGhub	57
JTAGhub Input and Output Ports	61
JTAGhub Ecosystem	63
JTAGhub Usage and Design Examples	64
JTAGhub Addressing Scheme	65
JTAGhub CORES	65
Setting Parameters and Connectivity	66
JTAG BYPASS Instruction in SOFTJTAG Chain	67
Using JTAGMON	69
JTAGMON Input and Output Ports	69
JTAGMON Design Flow	70
User Interface Descriptions	71
Chapter 3 Reveal Analyzer	73
About Reveal Analyzer	74
Reveal On-Chip Debug Design Flow	74
Inputs	75
Outputs	75
Inserting the Debug Logic	76
Mapping, Placing, and Routing the Design	76
Generating a Bitstream File	76
Connecting to the Evaluation Board	77
Downloading a Design onto the Device	77
Starting Reveal Analyzer	78
Starting with a New File	79
Starting with an Existing File	80
Changing the Cable Connection	81
Selecting a Reveal Analyzer Core	81
Setting Up the Trace Signals	82
Setting the Trace Bus Radix	82
Adding Time Stamps to Trace Samples	82

Setting Up the Trigger Signals	83
Renaming Trigger Units	83
Setting Up Trigger Units	83
Renaming Trigger Expressions	85
Setting Up Trigger Expressions	85
Setting Trigger Options	86
Creating Token Sets	87
Debugging with Reveal Controller	88
Performing Logic Analysis	104
Data Capture with Sample Enable	105
Common Error Conditions	105
Stopping a Logic Analysis	105
Using Manual Triggering	106
Viewing Waveforms	106
Viewing Logic Analysis	106
Adjusting the Waveform Display	107
Panning	107
Zooming In and Out	107
Specifying the Clock Period	109
Placing, Moving, and Locating Cursors	110
Counting Samples	111
Exporting Waveform Data	111
Saving a Project	112
Exiting Reveal Analyzer	112
User Interface Descriptions	113
LA Trigger Tab	113
LA Waveform Tab	118
Viewing and Saving Waveforms	119
Using the Memory Controller Debug	120
Chapter 4 Best Practices	124
General Guidelines	125
Best Practices for Post-Synthesis Flow	126
Guidelines when Updating User Design in Post-Synthesis Debug Flow	126
Guidelines when Changing the Synthesis Tool	127
Revision History	128

Introduction

The Radiant software contains the Reveal Inserter and the Reveal Analyzer tools used for design debugging. This guide describes Reveal Inserter and Reveal Analyzer.

One of the most common activities in debugging is logic analysis. To do this, use Reveal Inserter and Reveal Analyzer. You can use both with all supported FPGA devices.

Reveal continuously monitors signals within the FPGA for specific conditions, which can range from simple to quite complex. When the trigger condition occurs, Reveal can save signal values preceding, during, and following the event for analysis, including a waveform presentation. The data can be saved to a value change dump file (.vcd), which can be used with tools such as QuestaSim, or to an ASCII tabular format that can be used with tools such as Excel.

Before running Reveal Analyzer, use Reveal Inserter to add Reveal modules to your design. In these modules, specify the signals to monitor, define the trigger conditions, and other options. Reveal supports multiple logic analyzer cores using hard/soft JTAG interface. You can have up to 15 modules, typically one for each clock region of interest. When the modules are set up, regenerate the bitstream data file to program the FPGA.

A feature is enabling an added module called Reveal Controller. This controller module enables:

- ▶ Access to the control and status registers of the hard IPs such as I2CFIFO, PLL, PCIe, CDR, and DPHY via the LMMI (Lattice Memory Mapped Interface) slave interface
- ▶ Virtual switches and LEDs emulating on-board switches and LEDs to control and monitor a user design. Up to 32 switches and 32 LEDs are supported.

- ▶ Read/write access to a bank of user registers and/or initialize memory post-configuration.

The main purpose is narrowing down to problem areas during debug cycles using a divide and conquer method into many small functional blocks to control and monitor the status of each block.

Before starting a test run, set up Reveal Analyzer. This includes setting a number of options, including modifying trigger conditions and customizing the waveform display. You can save these settings for later use. During and after a test run, view the incoming data in Reveal's LA Waveform view. You can also save the data to a .vcd or .txt file to analyze with other tools.

Note:

Reveal supports IEEE-P1735 encryption. If this encryption is applied to a design, the design tree will allow only the visible ports and signals that are not encrypted to be inserted by the Reveal Inserter for triggering purposes.

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, System Verilog and mixed-HDL flows for debug insertion. Once the design has been modified for debug, it is mapped, placed, and routed with the normal design flow in the Radiant software.

Using JTAG Debugger

JTAG can be used to access debug logic to test and analyze your design. Devices in Radiant support hard and soft JTAG cores. Hard JTAG is basically a JTAG block IP that is part of the device. Soft JTAG is built in RTL, which uses the device fabric logic.

In the iCE40 device, Reveal JTAG support is implemented using logic for JTAG state machine and GPIO pins for four JTAG pins (JTAG_TCK, JTAG_TDI, JTAG_TMS, and JTAG_TDO).

Consider the following recommendations:

- ▶ Lock the JTAG_TCK pin to PCLK or GR_PCLK to avoid using general routing, as clock general routing may violate the CLK 1-PLC rule. For an example of the `ldc_set_location` constraint:

```
ldc_set_location {JTAG_TCK} -site J2
```

- ▶ Lock pins JTAG_TCK, JTAG_TDI, JTAG_TMS, and JTAG_TDO on the same bank. Make other banks available for DDR, MIPI or LVDS usage.
- ▶ Set the frequency constraint as follows:

```
create_clock -name {JTAG_TCK} -period 166.67
```

Likewise, in CrossLink device, there is no hard JTAG block and Reveal uses soft JTAG debug logic and GPIO pins for the four JTAG pins.

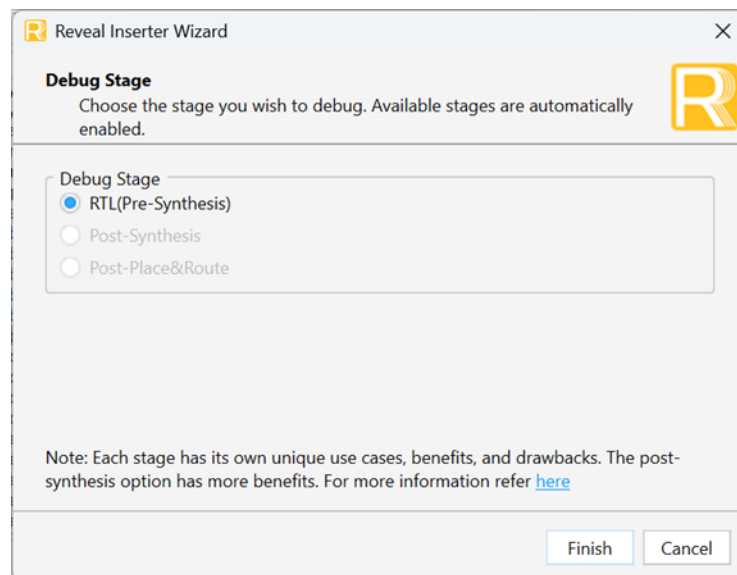
Refer to [“Using JTAGhub” on page 57](#) for information about the use and merging of JTAGhubs in Nexus and Avant devices.

Debug Flows

Reveal Inserter features two debug flows. When you open Reveal Inserter, the Reveal Inserter Wizard prompts you to choose the debug flow to use.

- ▶ **RTL (Pre-Synthesis)** – This is the standard flow for debugging a design all at once. The user inserts the logic core before the design is synthesized the first time. Refer to [“RTL \(Pre-Synthesis\) Debug Flow” on page 11](#) for details.
- ▶ **Post-Synthesis** – This flow allows you to insert debug logic after running synthesis. As such, this option is available if you have already run synthesis. It is useful for isolating or monitoring certain signals and focus your changes or improvements in specific areas of the design. Refer to [“Post-Synthesis Debug Flow” on page 11](#) for details.
- ▶ **Post-Place & Route** – This flow allows you to insert debug logic after running place and route. As such, this option is available if you have already completed synthesis, mapping, and placement and routing. Also referred to as non-intrusive debugging, you can add your debug core after place and route without having to alter your original design. The post-place & route debug flow is available when designing for Avant and Nexus devices. Refer to [“Post-Place & Route Debug Flow” on page 15](#) for details.

Figure 1: Reveal Inserter Wizard



RTL (Pre-Synthesis) Debug Flow

The following are the general steps in the Reveal Inserter RTL design flow.

1. Start Reveal Inserter.
2. In the Reveal Inserter Wizard, select RTL (Pre-Synthesis) as debug stage and click Finish.
3. Create a new Reveal Inserter project or open an existing Reveal Inserter project.
4. Add and set up analyzer and controller modules.
5. Insert the debug logic.

This process generates and synthesizes the necessary debug logic.

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

6. Map, place, and route the design.
7. Generate the .bin bitstream file.

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

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

Post-Synthesis Debug Flow

In post-synthesis debugging, it is required that you mark the specific signal objects that you want to monitor by adding the `syn_rvl_debug` synthesis attribute in the RTL source files. This allows you to keep track of these signals more easily as they are highlighted on the user interface. The `syn_rvl_debug` attribute is the same for both Synplify and LSE tools.

The `syn_rvl_debug` attribute performs two functions:

- ▶ This attribute highlights the signal so it can be easily identified in the user interface.
- ▶ After synthesis, this attribute tells the synthesis tool to preserve the signal without optimizing it. If it is a bus, the data width is also preserved. The synthesis tool passes the same attribute to the signal in the post-synthesis netlist (*.vm).

The syntax in Verilog/System-Verilog is:

```
/* synthesis syn_rvl_debug = 1 */;
```

The syntax in VHDL is:

```
attribute syn_rvl_debug : boolean;
attribute syn_rvl_debug of sig1 : signal is true
```

For example:

```
input clki /*synthesis syn_rvl_debug = 1 */,  
wire clk1 /* synthesis syn_rvl_debug = 1 */;  
reg [39:0] cnt /* synthesis syn_rvl_debug = 1 */;  
reg [31:0] cnti /* synthesis syn_rvl_debug = 1 */;
```

The synthesis tool may add a suffix to the name of a signal with the `syn_rvl_debug` attribute. This change is minimal and the signal remains recognizable to the user.

The `syn_rvl_debug` attribute may be attached to top level input/output ports, which will be connected to the input/output buffers. In case of an input port with the `syn_rvl_debug` attribute, in the .vm file, the output of the input buffer should get the passed attribute. In case of an output port with attribute, in the .vm file, the input of the output buffer should get the passed attribute.

Example 1

RTL Code:

```
input clki /* synthesis syn_rvl_debug = 1 */,
```

The expected output is:

```
wire clki_c /* synthesis syn_rvl_debug = 1 */;
```

where `clki` is driving the input port of IB and `clki_c` is connected to the output port. This is similar in the output port:

```
output dout /* synthesis syn_rvl_debug = 1 */,
```

The expected output is:

```
wire dout_c /* synthesis syn_rvl_debug = 1 */
```

Example 2

RTL Code:

```
reg [31:0] counter /*synthesis syn_rvl_debug = 1*/;
```

This is the expected .vm code when the exact signal name is preserved:

```
wire [31:0] counter /*synthesis syn_rvl_debug = 1*/;
```

When the exact signal name cannot be preserved, some suffix(*) may be added.

```
wire [31:0] counter_* /*synthesis syn_rvl_debug = 1*/;
```

Example 3

If the attribute is provided in the following way, it is applied to the last signal:

RTL Code:

```
reg sig1, sig2, sig3 /*synthesis syn_rvl_debug = 1*/;
```

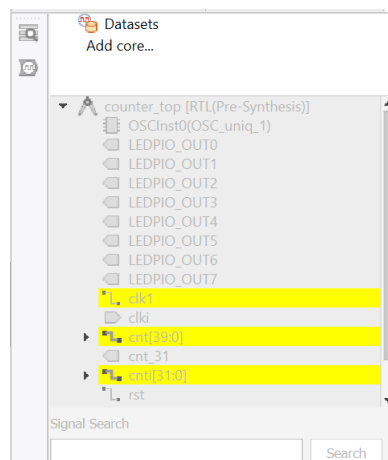
The expected .vm code is:

```
wire sig3 /*synthesis syn_rvl_debug = 1*/;
```

The synthesis tool may also expand/elaborate the RTL code if the attribute is attached to the states in the FSM. This depends on the type of encoding such as binary, one-hot and others.

Here is an example of highlighted signals attached with the `syn_rvl_debug` attribute.

Figure 2: Marked Debug Signals



Note:

Before you start post-synthesis debugging, make sure that there is no active .rvl project in your design. If there is, make it inactive. To change a file's status to inactive, right-click the file in the File List view and choose Set as Inactive.

The following are the general steps in the Reveal Inserter Post-Synthesis debug flow.

1. After running synthesis, start Reveal Inserter.
2. In the Reveal Inserter Wizard, select Post-Synthesis as debug stage and click Finish.

In the list of signals displayed in Reveal Inserter, the signals with the `syn_rvl_debug` attributes are highlighted for easy tracking.

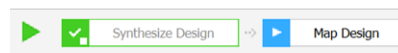
3. Add and set up analyzer and controller modules, if needed.

Note

When Reveal Controller is used for switches in post-synthesis, make sure the signals are not driven by other signals in the design. Multiple driver issues produce an error in Place & Route.

4. Click the **Design Rule Check** button and fix errors, if needed.
5. Click the **Insert Debug** button.
6. In the Insert Debug to Design dialog box, select the modules to insert. Select **Activate Reveal file in design project** to include it in synthesis. This also sets other .rvl files as inactive.
7. Click **OK**. The .rvl file is listed in the File List pane under Debug Files.

You will note that the Synthesize Design process bar now has a small white box to indicate that only Reveal is to be synthesized in post-synthesis stage.



8. Run **Synthesize Design**. Only the changes in Reveal are processed.
In the Task Detail View, you will observe the added Post-Synthesis Reveal task in progress.
9. Map, place, and route the design.
10. Generate the bitstream file.
If you want to perform logic analysis with Reveal Analyzer, continue with these steps:
11. Set up the cable connection with Programmer.
12. Download the design onto the device.
13. Start Reveal Analyzer and perform logic analysis with it.

Post-Place & Route Debug Flow

Similar to the post-synthesis debug flow, in post-place & route debugging, it is required that you mark the specific signal objects that you want to monitor by adding the `syn_rvl_debug` synthesis attribute in the RTL source files. The syntax for Verilog and VHDL are the same. See [“Post-Synthesis Debug Flow” on page 11](#).

The following are the general steps in the Reveal Inserter Post-Place & Route debug flow.

Note

When you initially run synthesis, map, and place & route, make sure there is no active Reveal (.rvl) project. If there is an active .rvl project, make it inactive by right-clicking the .rvl file in the File List view and choosing Set as Inactive.

1. After running synthesis, map, and place & route, start Reveal Inserter.
2. In the Reveal Inserter Wizard, select **Post-Place & Route** as debug stage and click **Finish**.

Note

In the Tcl Console, you will see that a new Reveal project is created in the post-PAR stage. The placed and routed design is preserved in the PAR output udb.

```
> rvl_new_project -stage postpar
```

In the design tree, the entire database after place and route is shown. The signals with the `syn_rvl_debug` attributes are highlighted.

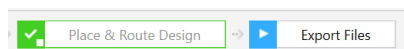
3. Add and set up logic analyzer cores

Note

Adding Reveal Controller cores is not supported in post-Place & Route debug flow.

4. Click the **Design Rule Check** button and fix errors, if needed.
5. Click the **Insert Debug** button.
6. In the Insert Debug to Design dialog box, select the modules to insert. Select **Activate Reveal file in design project** to include it in synthesis. This also sets other .rvl files, if any, as inactive.
Click **OK**.
7. In the Save Reveal Project dialog box, type the file name of the .rvl file and click **Save**. The .rvl file is listed in the File List view under Debug Files.

You will note that the Place & Route process bar now has a small white box to indicate that only Reveal is to be processed in post-place and route stage.



8. In the Task Detail View, you will observe that the following sub-tasks are unchecked. These will be run when you click the Place & Route button.
 - ▶ Place & Route Reveal
 - ▶ Place & Route Timing Analysis
9. Run **Place & Route**. Only the changes in Reveal are processed.
10. Generate the bitstream file.

If you want to perform logic analysis with Reveal Analyzer, continue with these steps:
11. Set up the cable connection with Programmer.
12. Download the design onto the device.
13. Start Reveal Analyzer and perform logic analysis with it.

Input and Output Files

The inputs to the Reveal Inserter flow are the following:

- ▶ VHDL, Verilog, and System Verilog files

Reveal Inserter generates the following files in the debug 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.

Limitations

Reveal Inserter has the following limitations in the current release.

Unsupported VHDL, Verilog and System Verilog Features in Reveal Inserter

The following features that are valid in the VHDL, Verilog and System 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.
- ▶ Signals used in VHDL “generate” statements are not available for tracing and triggering.
- ▶ Signals that are VHDL user-defined enumerated types, integer type, or Boolean type are not available for tracing and triggering.
- ▶ Some signals in a System Verilog design appear in the signal hierarchy but are not available for triggering or tracing. These signals include:
 - ▶ Array types of two dimensions or more are not shown in the port or node section
 - ▶ Signals that are user defined enumerated types, integer type, byte/shortint/int/longint type
 - ▶ Signals that belong to typedef and interface

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.

Signals Implemented as Hard Routes

Signals that are implemented as hard routes in the FPGA instead of using the routing fabric are not available for tracing or triggering. Examples are connections to IB and OB components. Many common hard routes are automatically shown as unavailable in Reveal Inserter, but some are not. If you select a signal for tracing or triggering that is implemented as a hard route, an error will occur during the synthesis, mapping, placement, or routing steps.

Dangling or Unconnected Nets

Dangling or unconnected nets in Verilog, System Verilog, or VHDL code are available for use with Reveal Inserter.

Synthesis Parameters

VHDL generics for synthesis must be added via HDL Parameters field in Project Properties. The current version does not support parameters via Command Line Options field in Synthesis Strategy setting.



Getting Started

After you create a project in the Radiant software, 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.

Choose the debug flow in the Reveal Inserter Wizard and click **Finish**.

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

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. The following steps can also be seen in Figure 3

To start Reveal Analyzer with a new file:

1. Issue the start command. To start:
 - ▶ For stand-alone in Windows, go to the Windows Start menu and choose **Programs > Lattice Radiant Reveal >  Reveal Logic Analyzer**.
 - ▶ For stand-alone in Linux, go to a command line and enter the following:

`<Reveal install path>/bin/linux64/revealrva`

The Reveal Analyzer Startup Wizard dialog box appears.

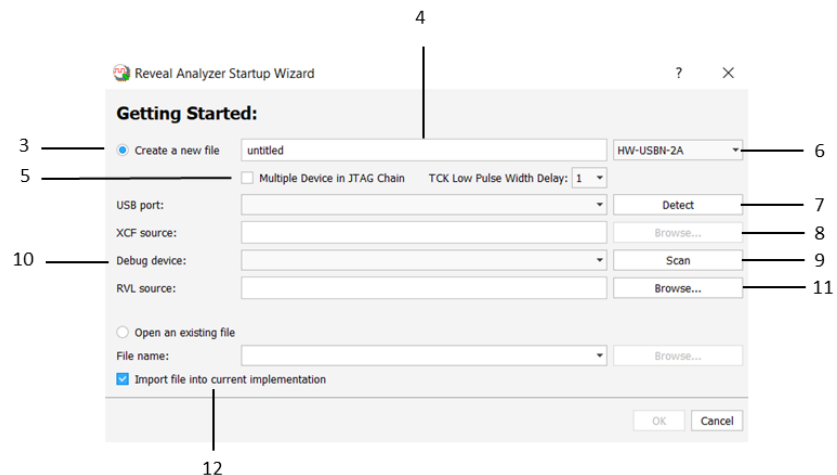
2. If Reveal Analyzer opens with an existing file, choose **File > Save As**.

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

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

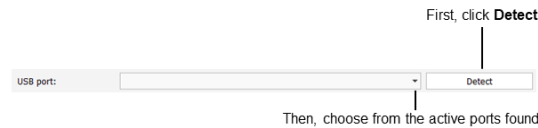
The dialog box presents a few rows of boxes that need to be filled in. In the figure below, the numbers match up with the steps in this procedure and show where the boxes, buttons, and menus are for each step.

Figure 3: Startup Wizard with Steps for a New File

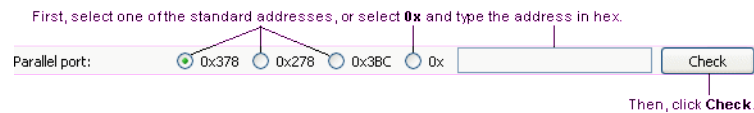



4. In the first second row, type in the base name of the file. The extension is added automatically.

5. If there are daisy-chained devices, select **Multiple Device in JTAG Chain**.
6. To the right of this row is a pulldown menu. Choose the type of cable that your board is connected to.
7. Select the port. The method depends on the cable type:
 - ▶ If USB, click **Detect**. Then choose from the active ports found. The following figure shows the row after choosing a USB type.



- ▶ 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 row after choosing a parallel type.




8. If there are daisy-chained devices, click **Browse** in the XCF source row to find the XCF source file.
9. Click **Scan** to find the FPGA.
10. 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.
11. Click **Browse** in the RVL source row to find the Reveal Inserter project (.rvl) file.
12. To add the new .rva file to the File List view, select **Import file into current implementation**. The .rva file works the same either way.
13. Click **OK**.

Opening an Existing Reveal Inserter Project

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

`<Reveal install path>/bin/linux64/revealrva`

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

Using the Reveal Controller Simulation Model

The Reveal Controller Simulation Model replicates the functionality of Reveal Controller in simulation.

To set up the Reveal Controller Simulation Model:

1. Choose **Tools > Reveal Inserter** to insert a reveal module into the design.
2. Run **Synthesize Design** to generate the reveal_workspace file.
 - ▶ If you insert Reveal Controller only, the tool will create the sim_src folder with the pre-synthesis Reveal generated source file under reveal_workspace.
 - ▶ If you insert Reveal Controller and Analyzer, the Reveal engine will not create the folder and no source files will be provided.
3. Click **Tools > Simulation Wizard** or click the  icon in the toolbar.
4. In the Simulator Project Name and Stage dialog box, enter the project name.

Click **Next**.

5. In the Add and Reorder Source dialog box, select and add the source files from the sim_src folder.

Click **Next**.

The following compiler directives are all enabled in the source file. You can use the OEM simulator or any stand-alone simulator to compile and simulate the Controller:

- ▶ ``define en_sw`
- ▶ ``define en_led`
- ▶ ``define en_user_mem`
- ▶ ``define en_user_creg`
- ▶ ``define en_user_sreg`

Managing the Cores in a Project

Each Reveal Inserter project can include multiple debug logic cores and one Reveal controller core. The 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.

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. 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.

Note:

For the Avant device, you need to add the PAR Strategy command line option “-exp WARNING_ON_PCLKPLC1=1” when using Reveal and JTAGH25 to avoid errors related to the jtck_N clock.

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
[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. This is either the regular debug logic core or the controller core.

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 25](#) 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.
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.

Power-on Reset (POR) Debug

An automatic "trigger enable" signal must be built into the Reveal module to monitor POR functions. Before Reveal Analyzer starts, these functions happen immediately after the power-on of the test board. When the trigger enables signal transitions to "active," the module will watch for the trigger and collect samples. It 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 not supported in iCE40UP devices.

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**.

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 discontinuous, 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.

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:

$$256 \times 2 = 512$$

$$512 = 2^9$$

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	✓	
	Remove	✓	✓
	Name	✓	✓
	Signals	✓	
	Operator	✓	✓
	Radix	✓	✓
	Value	✓	✓
Trigger Expressions	Add	✓	
	Remove	✓	✓
	Name	✓	✓
	Expression	✓	✓
	RAM type	✓	
	Maximum sequence depth	✓	
	Maximum event counter	✓	
Single Trigger Capture	Make available	✓	
Multiple Trigger Capture	Make available	✓	
	Number of samples per trigger	✓	✓
	Number of triggers		✓
Other Features	AND All versus OR All		✓
	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]` rise.
- ▶ 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** below 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** below for more information on NEXT statements.
- ▶ **# (count)** – Inserts a counter into a sequence. See **Sequences and Counters** below 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** below 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.

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.

Each core can support up to 16 trigger expressions. 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 25](#) 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 [“Triggering” on page 32](#).

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>`, information on these operators in Reveal Analyzer. See [“Creating Token Sets” on page 87](#) 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 [“Triggering” on page 32](#) 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 [“Triggering” on page 32](#) for more information on **NEXT** statements.
- ▶ **# (count)** – Inserts a counter into a sequence. See [“Triggering” on page 32](#) 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 [“Triggering” on page 32](#) for information on counters.

For more information on the precedence of these symbols in trigger expression syntax, see [“Triggering” on page 32](#).

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.

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

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**.

Setting Up Virtual Switch/LED Settings

The data pane consists of Virtual Switch/LED panels in which you drag and drop the RTL-defined switch and LED signals. A maximum of 32 bits can be selected for both Virtual Switch/LED signals.

To set up Virtual Switch/LED signals:

1. Select the top_Controller core in the Dataset view. (There can only be one Controller core.)
2. Click the **Virtual Switch & LED Setup** tab.
3. Select signals in the Design Tree by expanding the hierarchy or doing a name search in the Signal Search text box and clicking on Search.
4. Drag the desired signals to the Virtual Switch or LED data panels to the right. The width of the signals will automatically update between 1-32 entries at the top.
5. Beside the header title, Virtual Switch/LED Setting, there is a check box in which the Controller function for Virtual Switch/LED can be enabled/disabled.

Note

In order for this tab to display the Switch/LED names correctly and for the compilation tool flow to run through successfully, please use the **Reveal Controller template** from the Source Template Editor tool. See the help section: Entering the Design > HDL Design Entry > Using Templates.

Configuring User Memory Setup

Setting up the User Memory setting consists of assigning six mandatory ports (which must be defined in the user RTL design). The maximum data width is 32 bits.

The user design must include a memory block (EBR, Distributed Memory or PMI Memory) to be able set up User Memory.

The mandatory signals in the user design to set up User Memory are:

- ▶ Clock
- ▶ Clock_enable
- ▶ Wr_Rdn
- ▶ Address
- ▶ WData
- ▶ RData

To set up User Memory signals:

1. To select the top Controller core in the Dataset view, click on it. There can only be one Controller core. When you click on Add Controller, the control panel will appear. You can enable User Memory and connect to any lower-level hierarchy under the top-level, provided that all the ports are connected without any unconnected ports before and by selecting the corresponding memory port names. The Address and Data Width will be automatically updated.
2. Reveal Controller will add a mux between user logic and the Reveal Controller, which can be left blank or connected to the user logic. Once checked, click the **User Memory Setup** tab.

☒ User Memory Setting

Address: 0x90000000-0x9000FFFF

Address Width: 9 (4-16)

Data Width: 32 (4-32)

Setting List

	Memory Port
Clock	top_test1_inst/pmi_ram_dp_inst/Clock
Clock_enable	top_test1_inst/pmi_ram_dp_inst/ClockEn
Wr_Rdn	top_test1_inst/pmi_ram_dp_inst/WE_RDN
Address[8:0]	top_test1_inst/pmi_ram_dp_inst/Address[8:0]
WData[31:0]	top_test1_inst/pmi_ram_dp_inst/WData[31:0]
RData[31:0]	top_test1_inst/pmi_ram_dp_inst/Q[31:0]

Virtual Switch & LED Setup **User Memory Setup** Hard IP Setup User Status Register Setup User Control Register Setup

3. Select signals in the Design Tree and drag the signals to the desired position under Setting List in the User Memory Setup tab. As the pre-synthesis controller debug, no need to add any floating wires. For help finding signals see, [“Searching for Signals” on page 25](#).
4. Drag the desired signals into the User Memory data panel to the right. You do not need to provide a clock if there is no user logic connected while the width of the signals will automatically update between 4-32 entries at the top.
5. Beside the title User Memory Setting, there is a check box in which the Controller function for User Memory signals can be enabled/disabled. If there is a noticeable reset port, it should be connected to an inactive reset signal so that it does not impact the controller functionality and it should not be in a floating state.

Configuring User Control Register Setup

The User Control Register Setup tab displays a table-like panel where you can enable the Control Register. The Control Register address ranges from 0x81000000 up to 0x810000f8.

You can add up to 32-address locations with a maximum data-width of 8-bit. If the signal has a data width greater than 8, it splits into 8-bit per address location with a unique label.

The user design must include the control registers as wires, which should not have any other drivers. They should be treated as asynchronous switches as they are changed with the JTAG clock.

These signals are read-writable from the controller.

To set up User Control Register:

1. Click the **User Control Register Setup** tab.

The screenshot shows the 'Control Register' setup window. It features a table with the following data:

	Name	Control Signals (MSB:LSB)	Width
1	creg0	top_test1_inst/control_reg0:7, top_test1_inst/control_reg0:6, top_test1_inst/control_reg0:5, top_test1_inst/control_reg0:4, top_test1_inst/control_reg0:3, top_test1_inst/control_reg0:2, top_test1_inst/control_reg0:1, top_test1_inst/control_reg0:0	8
2	creg1	top_test1_inst/control_reg0_7_0[7:0]	8
3	creg2	top_test1_inst/control_reg0_15_8[7:0]	8
4	creg3	top_test1_inst/control_reg0_23_16[7:0]	8
5	creg4	top_test1_inst/control_reg1:7, top_test1_inst/control_reg1:6, top_test1_inst/control_reg1:5, top_test1_inst/control_reg1:4, top_test1_inst/control_reg1:3, top_test1_inst/control_reg1:2, top_test1_inst/control_reg1:1, top_test1_inst/control_reg1:0	8
6	creg5	top_test1_inst/count_reg1:15, top_test1_inst/count_reg1:14, top_test1_inst/count_reg1:13, top_test1_inst/count_reg1:12, top_test1_inst/count_reg1:11, top_test1_inst/count_reg1:10, top_test1_inst/count_reg1:9, top_test1_inst/count_reg1:8	8
7	creg6	top_test1_inst/control_reg1:23, top_test1_inst/control_reg1:22, top_test1_inst/control_reg1:21, top_test1_inst/control_reg1:20, top_test1_inst/control_reg1:19, top_test1_inst/control_reg1:18, top_test1_inst/control_reg1:17, top_test1_inst/control_reg1:16	8
8	creg7	top_test1_inst/count_reg1:31, top_test1_inst/count_reg1:30, top_test1_inst/count_reg1:29, top_test1_inst/count_reg1:28, top_test1_inst/count_reg1:27, top_test1_inst/count_reg1:26, top_test1_inst/count_reg1:25, top_test1_inst/count_reg1:24	8
9	creg8	top_test1_inst/control_reg2:7, top_test1_inst/control_reg2:6, top_test1_inst/control_reg2:5, top_test1_inst/control_reg2:4, top_test1_inst/control_reg2:3, top_test1_inst/control_reg2:2, top_test1_inst/control_reg2:1, top_test1_inst/control_reg2:0	8
10	creg9	top_test1_inst/control_reg2:15, top_test1_inst/control_reg2:14, top_test1_inst/control_reg2:13, top_test1_inst/control_reg2:12, top_test1_inst/control_reg2:11, top_test1_inst/control_reg2:10, top_test1_inst/control_reg2:9, top_test1_inst/control_reg2:8	8
11	creg10	top_test1_inst/control_reg2:23, top_test1_inst/control_reg2:22, top_test1_inst/control_reg2:21, top_test1_inst/control_reg2:20, top_test1_inst/control_reg2:19, top_test1_inst/control_reg2:18, top_test1_inst/control_reg2:17, top_test1_inst/control_reg2:16	8
12	creg11	top_test1_inst/control_reg2:31, top_test1_inst/control_reg2:30, top_test1_inst/control_reg2:29, top_test1_inst/control_reg2:28, top_test1_inst/control_reg2:27, top_test1_inst/control_reg2:26, top_test1_inst/control_reg2:25, top_test1_inst/control_reg2:24	8
13	creg12	top_test1_inst/control_reg3:7, top_test1_inst/control_reg3:6, top_test1_inst/control_reg3:5, top_test1_inst/control_reg3:4, top_test1_inst/control_reg3:3, top_test1_inst/control_reg3:2, top_test1_inst/control_reg3:1, top_test1_inst/control_reg3:0	8
14	creg13	top_test1_inst/control_reg3:15, top_test1_inst/control_reg3:14, top_test1_inst/control_reg3:13, top_test1_inst/control_reg3:12, top_test1_inst/control_reg3:11, top_test1_inst/control_reg3:10, top_test1_inst/control_reg3:9, top_test1_inst/control_reg3:8	8
15	creg14	top_test1_inst/control_reg3:23, top_test1_inst/control_reg3:22, top_test1_inst/control_reg3:21, top_test1_inst/control_reg3:20, top_test1_inst/control_reg3:19, top_test1_inst/control_reg3:18, top_test1_inst/control_reg3:17, top_test1_inst/control_reg3:16	8
16	creg15	top_test1_inst/control_reg3:31, top_test1_inst/control_reg3:29, top_test1_inst/control_reg3:28, top_test1_inst/control_reg3:27, top_test1_inst/control_reg3:26, top_test1_inst/control_reg3:25, top_test1_inst/control_reg3:24	8
17	creg16	top_test1_inst/count_reg4:7, top_test1_inst/count_reg4:6, top_test1_inst/count_reg4:5, top_test1_inst/count_reg4:4, top_test1_inst/count_reg4:3, top_test1_inst/count_reg4:2, top_test1_inst/count_reg4:1, top_test1_inst/count_reg4:0	8
18	creg17	top_test1_inst/control_reg4:15, top_test1_inst/control_reg4:14, top_test1_inst/control_reg4:13, top_test1_inst/control_reg4:12, top_test1_inst/control_reg4:11, top_test1_inst/control_reg4:10, top_test1_inst/control_reg4:9, top_test1_inst/control_reg4:8	8
19	creg18	top_test1_inst/control_reg4:23, top_test1_inst/control_reg4:22, top_test1_inst/control_reg4:21, top_test1_inst/control_reg4:20, top_test1_inst/control_reg4:19, top_test1_inst/control_reg4:18, top_test1_inst/control_reg4:17, top_test1_inst/control_reg4:16	8
20	creg19	top_test1_inst/control_reg4:31, top_test1_inst/control_reg4:30, top_test1_inst/control_reg4:29, top_test1_inst/control_reg4:28, top_test1_inst/control_reg4:27, top_test1_inst/control_reg4:26, top_test1_inst/control_reg4:25, top_test1_inst/control_reg4:24	8
21	creg20	top_test1_inst/count_reg5:7, top_test1_inst/count_reg5:6, top_test1_inst/count_reg5:5, top_test1_inst/count_reg5:4, top_test1_inst/count_reg5:3, top_test1_inst/count_reg5:2, top_test1_inst/count_reg5:1, top_test1_inst/count_reg5:0	8
22	creg21	top_test1_inst/count_reg5:15, top_test1_inst/count_reg5:14, top_test1_inst/count_reg5:13, top_test1_inst/count_reg5:12, top_test1_inst/count_reg5:11, top_test1_inst/count_reg5:10, top_test1_inst/count_reg5:9, top_test1_inst/count_reg5:8	8
23	creg22	top_test1_inst/count_reg5:23, top_test1_inst/count_reg5:22, top_test1_inst/count_reg5:21, top_test1_inst/count_reg5:20, top_test1_inst/count_reg5:19, top_test1_inst/count_reg5:18, top_test1_inst/count_reg5:17, top_test1_inst/count_reg5:16	8
24	creg23	top_test1_inst/control_reg5:31, top_test1_inst/control_reg5:30, top_test1_inst/control_reg5:29, top_test1_inst/control_reg5:28, top_test1_inst/control_reg5:27, top_test1_inst/control_reg5:26, top_test1_inst/control_reg5:25, top_test1_inst/control_reg5:24	8
25	creg24	top_test1_inst/control_reg6:7, top_test1_inst/control_reg6:6, top_test1_inst/control_reg6:5, top_test1_inst/control_reg6:4, top_test1_inst/control_reg6:3, top_test1_inst/control_reg6:2, top_test1_inst/control_reg6:1, top_test1_inst/control_reg6:0	8
26	creg25	top_test1_inst/count_reg6:15, top_test1_inst/count_reg6:14, top_test1_inst/count_reg6:13, top_test1_inst/count_reg6:12, top_test1_inst/count_reg6:11, top_test1_inst/count_reg6:10, top_test1_inst/count_reg6:9, top_test1_inst/count_reg6:8	8
27	creg26	top_test1_inst/count_reg6:23, top_test1_inst/count_reg6:22, top_test1_inst/count_reg6:21, top_test1_inst/count_reg6:20, top_test1_inst/count_reg6:19, top_test1_inst/count_reg6:18, top_test1_inst/count_reg6:17, top_test1_inst/count_reg6:16	8
28	creg27	top_test1_inst/control_reg6:31, top_test1_inst/control_reg6:30, top_test1_inst/control_reg6:29, top_test1_inst/control_reg6:28, top_test1_inst/control_reg6:27, top_test1_inst/control_reg6:26, top_test1_inst/control_reg6:25, top_test1_inst/control_reg6:24	8
29	creg28	top_test1_inst/control_reg7:7, top_test1_inst/control_reg7:6, top_test1_inst/control_reg7:5, top_test1_inst/control_reg7:4, top_test1_inst/control_reg7:3, top_test1_inst/control_reg7:2, top_test1_inst/control_reg7:1, top_test1_inst/control_reg7:0	8
30	creg29	top_test1_inst/count_reg7:15, top_test1_inst/count_reg7:14, top_test1_inst/count_reg7:13, top_test1_inst/count_reg7:12, top_test1_inst/count_reg7:11, top_test1_inst/count_reg7:10, top_test1_inst/count_reg7:9, top_test1_inst/count_reg7:8	8
31	creg30	top_test1_inst/count_reg7:23, top_test1_inst/count_reg7:22, top_test1_inst/count_reg7:21, top_test1_inst/count_reg7:20, top_test1_inst/count_reg7:19, top_test1_inst/count_reg7:18, top_test1_inst/count_reg7:17, top_test1_inst/count_reg7:16	8
32	creg31	top_test1_inst/count_reg7:31, top_test1_inst/count_reg7:30, top_test1_inst/count_reg7:29, top_test1_inst/count_reg7:28, top_test1_inst/count_reg7:27, top_test1_inst/count_reg7:26, top_test1_inst/count_reg7:25, top_test1_inst/count_reg7:24	8

At the bottom of the dialog, there are 'Add' and 'Remove' buttons. The status bar at the bottom right shows the address range: 0x81000000 ~ 0x81000008.

2. Click the **Add** button. By default, a unique label is assigned to the Name column. You can edit this label by double-clicking it.


3. Drag-and-drop the corresponding signals from Design View to the **Control Signals** column.

Or

4. Select the existing row and double-click the Control Signals column.

The **Edit Signals** dialog box appears.

5. In the Edit Signals dialog box, do the following:

- Choose signals from the **Select Signals** pane.
- Click the forward  button to move the signals to the right-hand pane.
- Change the order of the signals as desired.
- Click **OK**.

6. If you wish to delete a row from the Control Signals column, click the **Remove** button to delete the selected row.

Configuring User Status Register Setup

The User Status Register Setup tab displays a table-like panel where you can enable the Status Register. The Status Register address ranges from 0x81001000 up to 0x810010f8.

You can add up to 32-address locations with a maximum data-width of 8-bit. If the signal has a data width greater than 8, it splits into 8-bit per address location with a unique label.

The user design must include status registers as wires, which should be driven by registers to show the status of the different functional blocks of the design. These signals are read-only.

To set up User Status Register:


1. Click the **User Status Register Setup** tab.

Name	Status Signals (MSB:LSB)	Width
1	top_test1_inst/stat_reg0:7.top_test1_inst/stat_reg0:6.top_test1_inst/stat_reg0:5.top_test1_inst/stat_reg0:4.top_test1_inst/stat_reg0:3.top_test1_inst/stat_reg0:2.top_test1_inst/stat_reg0:1.top_test1_inst/stat_reg0:0	8
2	top_test1_inst/stat_reg0:15.top_test1_inst/stat_reg0:14.top_test1_inst/stat_reg0:13.top_test1_inst/stat_reg0:12.top_test1_inst/stat_reg0:11.top_test1_inst/stat_reg0:10.top_test1_inst/stat_reg0:9.top_test1_inst/stat_reg0:8	8
3	top_test1_inst/stat_reg0:23.top_test1_inst/stat_reg0:22.top_test1_inst/stat_reg0:21.top_test1_inst/stat_reg0:20.top_test1_inst/stat_reg0:19.top_test1_inst/stat_reg0:18.top_test1_inst/stat_reg0:17.top_test1_inst/stat_reg0:16	8
4	top_test1_inst/stat_reg0:31.top_test1_inst/stat_reg0:30.top_test1_inst/stat_reg0:29.top_test1_inst/stat_reg0:28.top_test1_inst/stat_reg0:27.top_test1_inst/stat_reg0:26.top_test1_inst/stat_reg0:25.top_test1_inst/stat_reg0:24	8
5	top_test1_inst/stat_reg1:7.top_test1_inst/stat_reg1:6.top_test1_inst/stat_reg1:5.top_test1_inst/stat_reg1:4.top_test1_inst/stat_reg1:3.top_test1_inst/stat_reg1:2.top_test1_inst/stat_reg1:1.top_test1_inst/stat_reg1:0	8
6	top_test1_inst/stat_reg1:15.top_test1_inst/stat_reg1:14.top_test1_inst/stat_reg1:13.top_test1_inst/stat_reg1:12.top_test1_inst/stat_reg1:11.top_test1_inst/stat_reg1:10.top_test1_inst/stat_reg1:9.top_test1_inst/stat_reg1:8	8
7	top_test1_inst/stat_reg2:23.top_test1_inst/stat_reg2:22.top_test1_inst/stat_reg2:21.top_test1_inst/stat_reg2:20.top_test1_inst/stat_reg2:19.top_test1_inst/stat_reg2:18.top_test1_inst/stat_reg2:17.top_test1_inst/stat_reg2:16	8
8	top_test1_inst/stat_reg3:31.top_test1_inst/stat_reg3:30.top_test1_inst/stat_reg3:29.top_test1_inst/stat_reg3:28.top_test1_inst/stat_reg3:27.top_test1_inst/stat_reg3:26.top_test1_inst/stat_reg3:25.top_test1_inst/stat_reg3:24	8
9	top_test1_inst/stat_reg4:31.top_test1_inst/stat_reg4:30.top_test1_inst/stat_reg4:29.top_test1_inst/stat_reg4:28.top_test1_inst/stat_reg4:27.top_test1_inst/stat_reg4:26.top_test1_inst/stat_reg4:25.top_test1_inst/stat_reg4:24	8
10	top_test1_inst/stat_reg5:31.top_test1_inst/stat_reg5:30.top_test1_inst/stat_reg5:29.top_test1_inst/stat_reg5:28.top_test1_inst/stat_reg5:27.top_test1_inst/stat_reg5:26.top_test1_inst/stat_reg5:25.top_test1_inst/stat_reg5:24	8
11	top_test1_inst/stat_reg6:31.top_test1_inst/stat_reg6:30.top_test1_inst/stat_reg6:29.top_test1_inst/stat_reg6:28.top_test1_inst/stat_reg6:27.top_test1_inst/stat_reg6:26.top_test1_inst/stat_reg6:25.top_test1_inst/stat_reg6:24	8
12	top_test1_inst/stat_reg7:31.top_test1_inst/stat_reg7:30.top_test1_inst/stat_reg7:29.top_test1_inst/stat_reg7:28.top_test1_inst/stat_reg7:27.top_test1_inst/stat_reg7:26.top_test1_inst/stat_reg7:25.top_test1_inst/stat_reg7:24	8
13	top_test1_inst/stat_reg8:31.top_test1_inst/stat_reg8:30.top_test1_inst/stat_reg8:29.top_test1_inst/stat_reg8:28.top_test1_inst/stat_reg8:27.top_test1_inst/stat_reg8:26.top_test1_inst/stat_reg8:25.top_test1_inst/stat_reg8:24	8
14	top_test1_inst/stat_reg9:31.top_test1_inst/stat_reg9:30.top_test1_inst/stat_reg9:29.top_test1_inst/stat_reg9:28.top_test1_inst/stat_reg9:27.top_test1_inst/stat_reg9:26.top_test1_inst/stat_reg9:25.top_test1_inst/stat_reg9:24	8
15	top_test1_inst/stat_reg10:31.top_test1_inst/stat_reg10:30.top_test1_inst/stat_reg10:29.top_test1_inst/stat_reg10:28.top_test1_inst/stat_reg10:27.top_test1_inst/stat_reg10:26.top_test1_inst/stat_reg10:25.top_test1_inst/stat_reg10:24	8
16	top_test1_inst/stat_reg11:31.top_test1_inst/stat_reg11:30.top_test1_inst/stat_reg11:29.top_test1_inst/stat_reg11:28.top_test1_inst/stat_reg11:27.top_test1_inst/stat_reg11:26.top_test1_inst/stat_reg11:25.top_test1_inst/stat_reg11:24	8
17	top_test1_inst/status_reg4:7.top_test1_inst/status_reg4:6.top_test1_inst/status_reg4:5.top_test1_inst/status_reg4:4.top_test1_inst/status_reg4:3.top_test1_inst/status_reg4:2.top_test1_inst/status_reg4:1.top_test1_inst/status_reg4:0	8
18	top_test1_inst/status_reg4:15.top_test1_inst/status_reg4:14.top_test1_inst/status_reg4:13.top_test1_inst/status_reg4:12.top_test1_inst/status_reg4:11.top_test1_inst/status_reg4:10.top_test1_inst/status_reg4:9.top_test1_inst/status_reg4:8	8
19	top_test1_inst/status_reg4:23.top_test1_inst/status_reg4:22.top_test1_inst/status_reg4:21.top_test1_inst/status_reg4:20.top_test1_inst/status_reg4:19.top_test1_inst/status_reg4:18.top_test1_inst/status_reg4:17.top_test1_inst/status_reg4:16	8
20	top_test1_inst/status_reg4:31.top_test1_inst/status_reg4:30.top_test1_inst/status_reg4:29.top_test1_inst/status_reg4:28.top_test1_inst/status_reg4:27.top_test1_inst/status_reg4:26.top_test1_inst/status_reg4:25.top_test1_inst/status_reg4:24	8
21	top_test1_inst/count_reg5:7.top_test1_inst/count_reg5:6.top_test1_inst/count_reg5:5.top_test1_inst/count_reg5:4.top_test1_inst/count_reg5:3.top_test1_inst/count_reg5:2.top_test1_inst/count_reg5:1.top_test1_inst/count_reg5:0	8
22	top_test1_inst/status_reg5:15.top_test1_inst/status_reg5:14.top_test1_inst/status_reg5:13.top_test1_inst/status_reg5:12.top_test1_inst/status_reg5:11.top_test1_inst/status_reg5:10.top_test1_inst/status_reg5:9.top_test1_inst/status_reg5:8	8
23	top_test1_inst/status_reg5:23.top_test1_inst/status_reg5:22.top_test1_inst/status_reg5:21.top_test1_inst/status_reg5:20.top_test1_inst/status_reg5:19.top_test1_inst/status_reg5:18.top_test1_inst/status_reg5:17.top_test1_inst/status_reg5:16	8
24	top_test1_inst/status_reg5:31.top_test1_inst/status_reg5:30.top_test1_inst/status_reg5:29.top_test1_inst/status_reg5:28.top_test1_inst/status_reg5:27.top_test1_inst/status_reg5:26.top_test1_inst/status_reg5:25.top_test1_inst/status_reg5:24	8
25	top_test1_inst/status_reg6:7.top_test1_inst/status_reg6:6.top_test1_inst/status_reg6:5.top_test1_inst/status_reg6:4.top_test1_inst/status_reg6:3.top_test1_inst/status_reg6:2.top_test1_inst/status_reg6:1.top_test1_inst/status_reg6:0	8
26	top_test1_inst/status_reg6:15.top_test1_inst/status_reg6:14.top_test1_inst/status_reg6:13.top_test1_inst/status_reg6:12.top_test1_inst/status_reg6:11.top_test1_inst/status_reg6:10.top_test1_inst/status_reg6:9.top_test1_inst/status_reg6:8	8
27	top_test1_inst/status_reg6:23.top_test1_inst/status_reg6:22.top_test1_inst/status_reg6:21.top_test1_inst/status_reg6:20.top_test1_inst/status_reg6:19.top_test1_inst/status_reg6:18.top_test1_inst/status_reg6:17.top_test1_inst/status_reg6:16	8
28	top_test1_inst/status_reg6:31.top_test1_inst/status_reg6:30.top_test1_inst/status_reg6:29.top_test1_inst/status_reg6:28.top_test1_inst/status_reg6:27.top_test1_inst/status_reg6:26.top_test1_inst/status_reg6:25.top_test1_inst/status_reg6:24	8
29	top_test1_inst/status_reg7:7.top_test1_inst/status_reg7:6.top_test1_inst/status_reg7:5.top_test1_inst/status_reg7:4.top_test1_inst/status_reg7:3.top_test1_inst/status_reg7:2.top_test1_inst/status_reg7:1.top_test1_inst/status_reg7:0	8
30	top_test1_inst/status_reg7:15.top_test1_inst/status_reg7:14.top_test1_inst/status_reg7:13.top_test1_inst/status_reg7:12.top_test1_inst/status_reg7:11.top_test1_inst/status_reg7:10.top_test1_inst/status_reg7:9.top_test1_inst/status_reg7:8	8
31	top_test1_inst/status_reg7:23.top_test1_inst/status_reg7:22.top_test1_inst/status_reg7:21.top_test1_inst/status_reg7:20.top_test1_inst/status_reg7:19.top_test1_inst/status_reg7:18.top_test1_inst/status_reg7:17.top_test1_inst/status_reg7:16	8
32	top_test1_inst/status_reg7:31.top_test1_inst/status_reg7:30.top_test1_inst/status_reg7:29.top_test1_inst/status_reg7:28.top_test1_inst/status_reg7:27.top_test1_inst/status_reg7:26.top_test1_inst/status_reg7:25.top_test1_inst/status_reg7:24	8

Address: 0x81001000 ~ 0x810010f8

2. Click the **Add** button. By default, a unique label is assigned to the Name column. You can edit this label by double-clicking it.
3. Drag-and-drop the corresponding signals from Design View to the **Status Signals** column.
- Or
4. Select the existing row and double-click the Status Signals column.

The **Edit Signals** dialog box appears.

5. In the Edit Signals dialog box, do the following:
 - a. Choose signals from the **Select Signals** pane.
 - b. Click the forward  button to move the signals to the right-hand pane.
 - c. Change the order of the signals as desired.
 - d. Click **OK**.
6. If you wish to delete a row from the Status Signals column, click the **Remove** button to delete the selected row.

Configuring Hard IP Setup

The Hard IPs are automatically extracted from the RTL design and displayed in the data pane.

For the Inserter, the starting addresses for the IPs as shown is only for informational purpose. The addresses can be experimented in the Analyzer.

To set up Hard IP:

1. Click the **Hard IP Setup** tab.
2. Select Controller function for Hard IPs for analysis by enabling check boxes.

When implementing Reveal Controller for Hard IPs of PCS and PCIE, the implementation will not remove the user logic but will mux it with Reveal Controller logic. The following information explains this process:

- ▶ The user logic will access the LMMI interface and be active in the beginning of design for functions like link state machine. After that, the function needs to be inactive.
- ▶ When Reveal Controller gets access to the LMMI registers, it will select the mux during that time only. When there is no Reveal read/write command, it will de-select the mux.

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 40](#).
- ▶ 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.

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

When you finish setting up the trace and trigger signals, you can insert the Reveal modules into the design.

Note

Interactive and stand-alone synthesis are not compatible with Reveal modules. Reveal Inserter automatically uses the integrated synthesis option. Make sure your design project is set up for integrated synthesis if not done already.

To insert the debug logic modules into the design:

1. Choose **Debug >  Insert Debug**.
2. In the Insert Debug to Design dialog box, select the modules to insert.
3. Select **Activate Reveal file in design project**.

If the .rvl file is not active in the design project, the Reveal modules will not be included during synthesis.

4. Click **OK**.

Reveal Inserter performs a design rule check and saves the Reveal (.rvl) file. The Output view shows resource requirements and the DRC report for the modules. The .rvl file is listed in the File List pane under Debug Files.

5. Implement the design in the usual way.

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.

Performing Logic Analysis with Reveal Analyzer

After you have created your design project database with the Radiant software, generated a debug logic/controller 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 Analyzer” on page 73](#) for more information about performing logic analysis.

Using JTAGhub

A JTAGhub is a module that contains a JTAG core, which is either hard or soft, and around it is soft logic. A hard JTAG core wrapped by soft logic is a hard JTAGhub. A soft JTAG RTL core wrapped by soft logic is a soft JTAGhub.

In Nexus, a hard JTAGhub is referred to as JTAGH19 and a soft JTAGhub is called JTAGH19SOFT. In Avant, a hard JTAGhub is referred to as JTAGH25 and a soft JTAGhub is called JTAGH25SOFT. In pre-synthesis, you can have a maximum of 19 connected cores in Nexus and 25 connected cores in Avant. For details on the actual cores usable by Reveal or open to user instantiation and those reserved for certain other applications, refer to [“JTAGhub Addressing Scheme” on page 65](#) and [“JTAGhub CORES” on page 65](#).

In Reveal Inserter, JTAGhub is treated as a primitive which can be instantiated directly in user RTL.

Note

You can have several Reveal debug cores but only one controller core in your design.

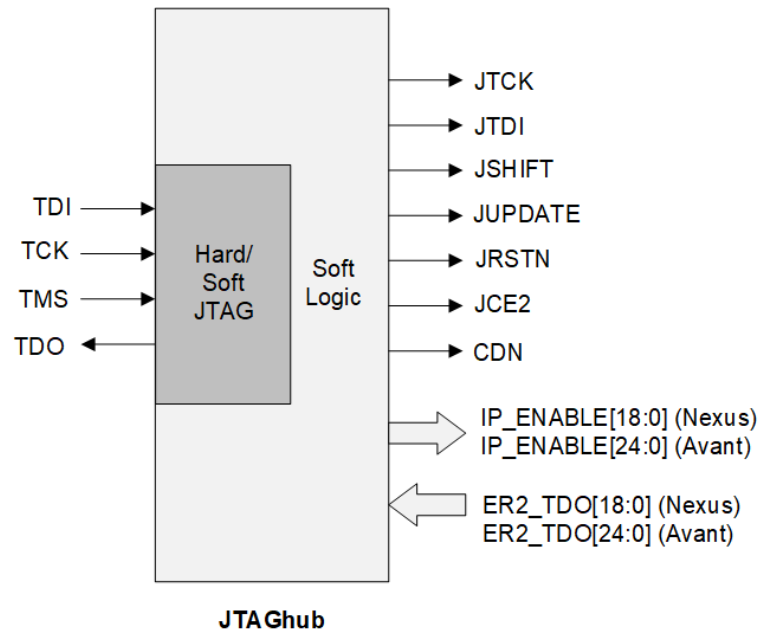
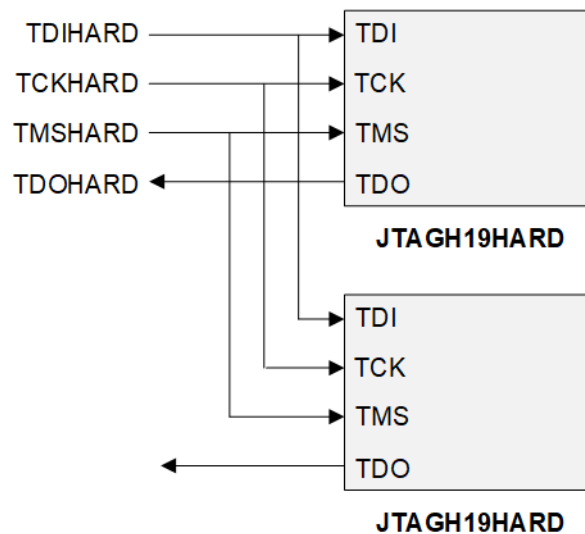
There can only be one hard JTAG in a design. So if there are multiple hard JTAGhub instances, all these are merged by the post-synthesis utility. Multiple soft JTAGhubs, with or without merging, are acceptable.

A JTAGhub, either soft or hard is inserted as a black-box by Reveal Inserter for normal debugging and treated as a primitive for synthesis. It is added to the synthesis header file as a black-box. The logic is not optimized by the synthesis tool. The user can also manually instantiate the JTAGhub in the design.

Hard JTAGhub – Merged

If there is already one hard JTAGhub in the design and the user inserts another hard JTAGhub in Reveal, then the two hard JTAGhubs are merged.

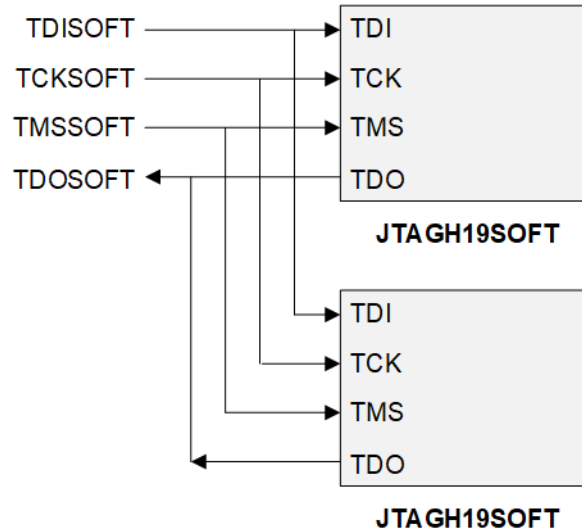
On the other hand, there can be multiple soft JTAGhub instances in a design. These soft JTAGhubs may or may not be merged. Each soft JTAGhub needs four I/O ports and a JTAG cable.

Figure 4: JTAGhub Architecture**Figure 5: Hard JTAGhub - Merged****Soft JTAGhub – Merged**

This is an example of two soft JTAG instances that are merged. In this example, all the JTAG ports are connected. TDISOFT, TCKSOFT, TMSSOFT and TDOSOFT are the top-level ports of the design. If the user needs to insert Reveal debugger, then the user must use TDISOFT, TCKSOFT, TMSSOFT, TDOSOFT as the top level port names.

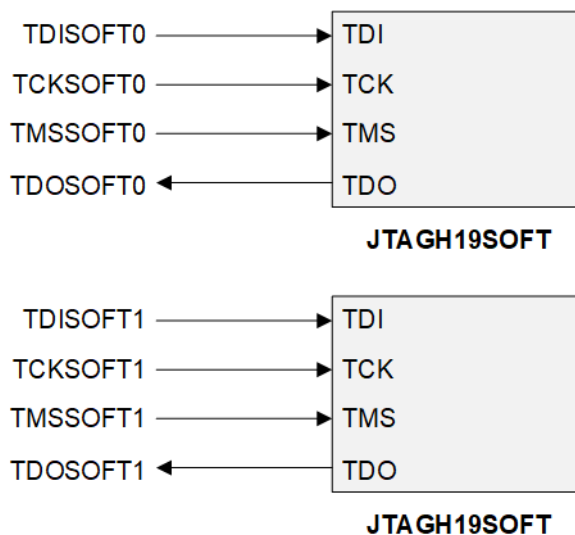
By merging JTAGhubs, only one set of I/O pins is required and only one set of logic is involved. Since merging minimizes logic resources, the design utilizes a minimal number LUTs.

Figure 6: Soft JTAGhub - Merged



Soft JTAGhub – Not Merged

This is an example of two soft JTAGhub instances that are not merged. In this diagram, the TDI, TCK, TMS, and TDO ports are connected to their corresponding top-level ports. The design has two independent sets of soft JTAG ports that are connected to two different cables. Hard JTAG is used for both programming and debugging. Both of these operations can be performed using the same cable and connection. Note that this is not true for soft JTAG.

Figure 7: Soft JTAGhub - Not Merged**Notes**

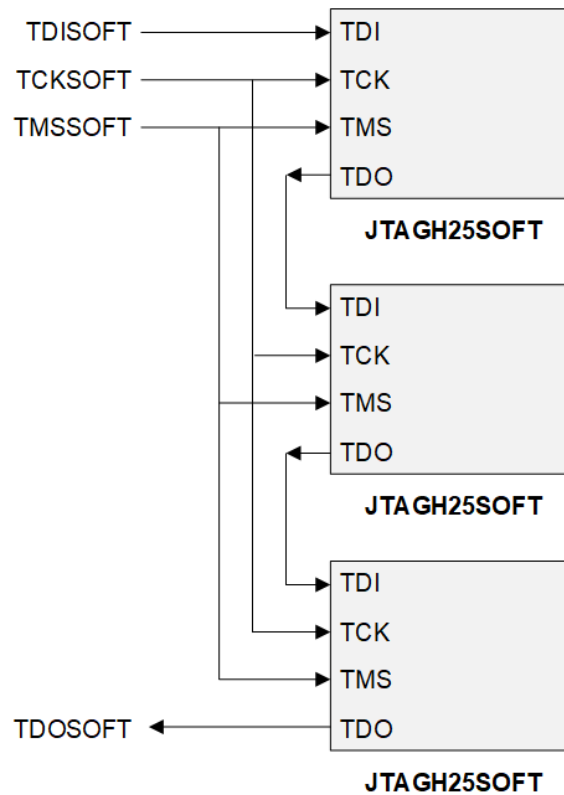
- ▶ Designs generated in Lattice Propel can use hard JTAG, soft JTAG, or a mix of the two types, depending on the processor. If there are two processors, two independent debugging GDBs are used. In some devices, however, Propel can use only one soft JTAGhub.
- ▶ The JTAG primitive and JTAGH19/JTAGH25 cannot co-exist in a design. Likewise, JTAGA and JTAGH19SOFT/JTAGH25SOFT cannot be added together in one design. These result in an error.

Support for BYPASS Instruction in Soft JTAGhub

In JTAG, the BYPASS register is a single-bit pass-through register that connects the TDO and the TDI ports.

The user can create SOFT JTAGhub chains wherein the TDO port of the first device is connected to the TDI port of the next device. The TCK and TMS ports are all connected together.

Up to 22 devices can be linked in one chain. These devices can be Avant only, Nexus only, or a mix of the two. BYPASS instruction allows the other devices in the JTAG chain to be tested with minimal overhead.

Figure 8: Soft JTAGhub Chain

JTAGhub Input and Output Ports

In JTAGhub, the standard TCK, TMS, TDI, and TDO ports are connected to the I/Os and go off-chip. All the other ports are connected to the FPGA fabric. When multiple sets of JTAGhubs are in the same design, all the JTAG ports are unique. Each set connects to one JTAG cable.

Signal	Description
TDI	Data shifted into the test or programming logic of the device.
TCK	Signal that synchronizes the internal state machine operations. In JTAG, TCK is not like a regular clock. It is pulsed when data is transmitted/received.
TMS	Signal sampled at the rising edge of TCK to determine the next state.
TDO	Data shifted out of the test or programming logic of the device.
JTCK	Signal similar to TCK and controlled by IP_ENABLE and ER2_TDO.
JTDI	Signal similar to TDI and controlled by IP_ENABLE and ER2_TDO.

Signal	Description
JSHIFT	Signal asserted and de-asserted at the pos-edge of JTCK. It remains high as long as valid data is available in JTDI.
JUPDATE	Signal that gives a positive pulse of one-cycle after JSHIFT is de-asserted. It should be used to capture the shifted-data (JTDI). It is asserted and de-asserted at the pos-edge of JTCK.
JRSTN	An active low reset signal. Initially set low and then remains high.
JCE2	Signal that goes high when valid data becomes available in JTDI for the selected core. It remains high as long as the data is valid. It is asserted one clock cycle before JSHIFT is asserted and de-asserted at the same time as JSHIFT.
CDN	Not used
IP_ENABLE[18:0] (Nexus) IP_ENABLE[24:0] (Avant)	Communicates with 19 cores in Nexus and 25 cores in Avant. Each bit enables one core. When one core is enabled, others are disabled. When IP_ENABLE bit and JCE2 and JSHIFT are high, then valid data is shifted in with JTDI and shifted out with corresponding bit of ER2_TDO.
ER2_TDO[18:0] (Nexus) ER2_TDO[24:0] (Avant)	Input data to be shifted out of TDO. Each bit corresponds to equivalent bit of IP_ENABLE.

A soft JTAGhub uses four GPIO ports with locate constraints. Hard JTAGhub pins are fixed in the device.

The TDI, TCK, TMS and TDO ports can be connected to the top-level of the user design.

The user or the tools must connect top-level ports to the TDI, TCK, and TMS of the JTAGhub. The TDO port may be connected to the top-level port for synthesis and may not be connected to the top-level port for simulation to avoid having multiple drivers.

If simulation is not required, it is recommended that the TDI, TCK, TMS input ports be tied to low and the TDO be unconnected.

Notes

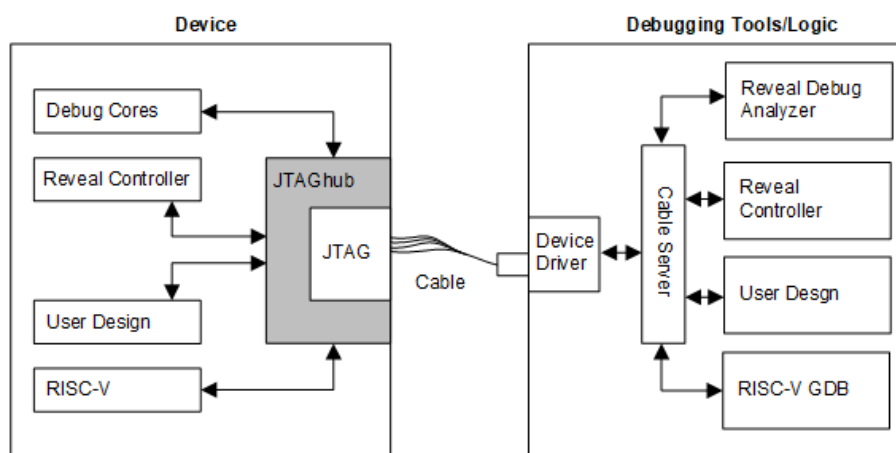
- ▶ For debug cores inserted by Reveal Inserter, there is no support for simulation.
- ▶ For netlists generated in System Builder and Propel, it is possible to simulate the user design.
- ▶ Encrypted source files for JTAGH19/JTAGH25 and JTAGH19SOFT/JTAGH25SOFT are provided to compile and run the simulation. In an Avant project, the models will also be available in the OEM simulator.
- ▶ If both JTAGH25 and JTAGMON are present in the design, the merge tool will intelligently combine them, ensuring compatibility and avoiding duplication.

JTAGhub Ecosystem

In a typical design, several Reveal debug cores and, normally, only one Reveal controller core may be present. There may likewise be user design logic and one or more RISC-V cores. These can all be connected to one JTAGhub and can communicate with the JTAG one at a time.

The JTAG cable connects to the machine with the Debug Analyzer and Reveal Controller tools. To set parameters and connectivity, the user can use the interface or run Tcl commands since these are all standard JTAG protocol commands. The RISC-V GDB debugging protocol can also be included. All these communicate with the cables server, then the device driver, and onto the device through the JTAG cable.

Figure 9: JTAGhub Ecosystem



JTAGhub Usage and Design Examples

This section provides some examples of how JTAGhub is used in a design and recommendations for debugging. A general rule is to avoid using both hard and soft JTAGhubs if using only one cable. With multiple cables, hard and soft JTAGhubs can be mixed in the design.

Note: A design with RISC-V does not necessarily come from Propel.

Example 1. The design has no existing JTAGhub

In this case, the user can insert either a hard JTAGhub or a soft JTAGhub.

Example 2. The design contains one RISC-V with hard JTAGhub

In this case, the user should use hard JTAGhub for Reveal debugging with just one cable. If soft JTAG is used for Reveal debugging, another cable is required.

Example 3. The design contains one RISC-V with soft JTAGhub

In this case, the user should use soft JTAGhub for Reveal debugging. The soft JTAGhubs are merged and only one cable is required.

Example 4. The design contains two RISC-V cores with hard JTAGhub

In this case, the user should use hard JTAGhub for Reveal debugging. The hard JTAGhubs are merged and only one cable is required.

Example 5. The design contains two RISC-V cores with soft JTAGhub

In this case, the user should use soft JTAGhub for Reveal debugging.

Example 6. The design contains one RISC-V with hard JTAGhub and the user instantiates another one hard JTAGhub

In this case, the three hard JTAGhubs are merged and the user should use hard JTAGhub for Reveal debugging.

Example 7. The design contains one RISC-V with soft JTAGhub and the user instantiates another one soft JTAGhub

In this case, the three soft JTAGhubs are merged and the user should use soft JTAGhub for Reveal debugging.

Example 8. The design contains multiple RISC-V cores with independent soft JTAGhubs

In this case, the user must use multiple cables, one for each RISC-V processor and its associated soft JTAGhub.

JTAGhub Addressing Scheme

The following addressing scheme is used to address the individual cores from the Cablesrvr clients (Reveal Debug, Controller, RISCv GDB, User Instance - Avant). If the user manually instantiates JTAGhub in the design, it is recommended to use only CORE-17 and CORE-18.

The address length is 24 bits for Nexus and 32 bits for Avant.

Address	Description
Nexus	
Bit[23]	Bit[23]
Bit[22:4]	19 bits for IP_ENABLE[18:0], one-bit for each core. To enable a core, the IP_ENABLE bit of that core is set high and other IP_ENABLE bits are set low.
Bit[3]	Reserved bit for future use to clear data.
Bit[2:0]	3-bit control, value of 6 means read
Avant	
Bit[29]	Bit[29]
Bit[28:4]	25 bits for IP_ENABLE[24:0], one bit for each core. To enable a core, the IP_ENABLE bit of that core is set high and other IP_ENABLE bits are set low.
Bit[3]	Reserved bit for future use to clear data
Bit[2:0]	3-bit control, value of 6 means read
Bit[31:30]	2 bits reserved

JTAGhub CORES

Example Commands

To enable read of the ID of a CORE

```
! enable ER1
SIR 8 TDI (32);          // 8-bit data with hex value 32
! set reveal core
SDR 24 TDI (800006);     // 24-bit data with hex value 800006
```

To enable a CORE (CORE0 n this case)

```
! enable ER1
SIR 8 TDI (32);
SDR 24 TDI (000016);
```

This table lists the CORE hex values for Nexus and Avant devices. In Nexus, CORE17 and CORE18 are open to any user instantiation. In Avant, CORE17 to CORE 24 are cores open to any user instantiation.

Nexus	Avant
CORE17 => SDR 24 TDI (200006);	CORE17 => SDR 32 TDI (00200006);
CORE18 => SDR 24 TDI (400006);	CORE18 => SDR 32 TDI (00400006);
	CORE19 => SDR 32 TDI (00800006);
	CORE20 => SDR 32 TDI (01000006);
	CORE21 => SDR 32 TDI (02000006);
	CORE22 => SDR 32 TDI (04000006);
	CORE23 => SDR 32 TDI (08000006);
	CORE24 => SDR 32 TDI (10000006);

Setting Parameters and Connectivity

When Reveal Inserter inserts a core, it sets the HUB_* value as “0b1” and connects the IP_ENABLE bit and the ER2_TDO bit.

Examples:

- ▶ For any user instantiation, the settings and connections are as follows for CORE-17

```
parameter HUB_17 = "0b1";
```

IP_ENABLE[17] is connected to the IP_ENABLE input port of the core.

ER2_TDO[17] is connected to the ER2_TDO output port of the core.

- ▶ For any user instantiation, the settings and connections are as follows for CORE-18

```
parameter HUB_18 = "0b1";
```

IP_ENABLE[18] is connected to the IP_ENABLE input port of the core.

ER2_TDO[18] is connected to the ER2_TDO output port of the core.

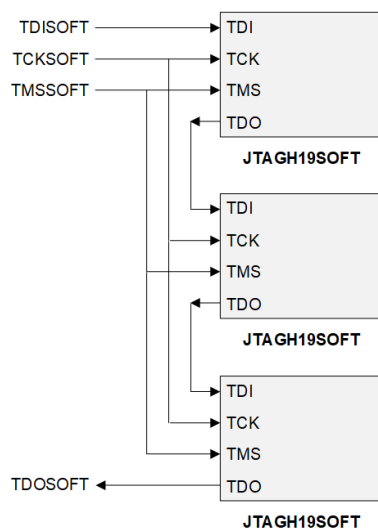
JTAG BYPASS Instruction in SOFTJTAG Chain

JTAG BYPASS instruction allows you to reduce overhead by passing over devices in a chain that do not need to be involved in the current action. These devices remain in functional mode but allow data to flow through to the next device in the chain. When debugging, you can also use BYPASS instruction to isolate what may be offending devices in the chain.

You can build a SOFTJTAG chain of Nexus devices, Avant devices, or a mix of both Nexus and Avant devices. Radiant supports up to 22 devices in a single chain.

The TDO port of the first device is connected to the TDI port of the next device in the chain. The TCK and TMS ports are connected together.

Figure 10: Soft JTAGhub Chain of Three Nexus Devices



Each device with softJTAG interface can be debugged in Reveal when connected in a chain. For example, if multiple devices are connected in a chain, a specific target device can be selected to be debugged. You can mix hardJTAG and softJTAG interfaces as long as they are connected in a chain. The softJTAG interface only has access to the FPGA fabric and that is how debugging is done.

Note:

The softJTAG interface does not have access to configuration logic, so there are no programming and scanning functions.

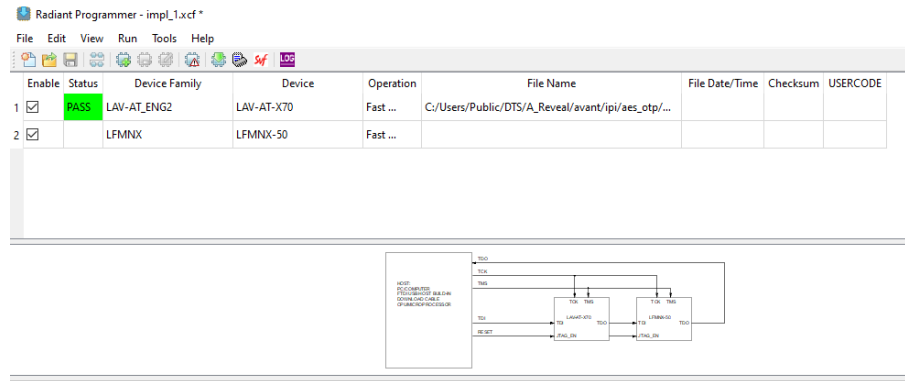
To scan debug devices in a chain with hardJTAG and softJTAG interfaces:

1. Scan the devices in Radiant Programmer tool. The hardJTAG interface will be used in scanning these devices. Open Radiant Programmer and

choose **Run > Scan Device**. The tool scans and lists the devices in a new .xcf file.

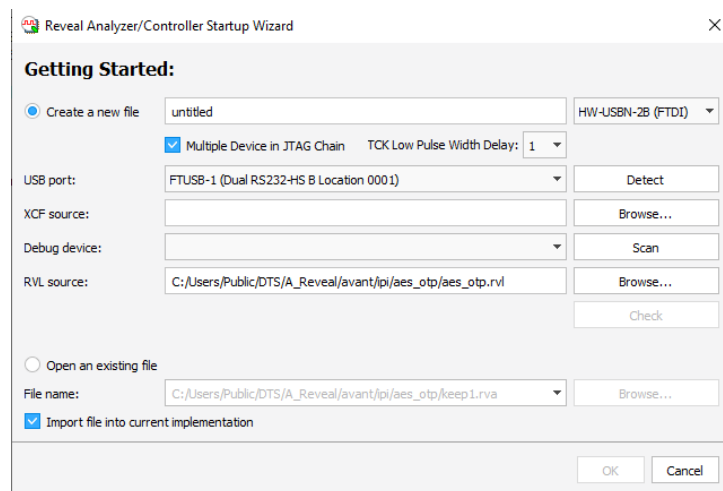
- If there are devices connected in the chain using softJTAG interface, manually add these devices using **Edit > Add Device**.

Figure 11: Manually Adding Device



- Create the .xcf file. This .xcf file will be common for all the Reveal projects.
- Open Reveal Analyzer and create a new project.
- In the Reveal Analyzer/Controller Startup Wizard, select the **Multiple Device in JTAG Chain** option.

Figure 12: Multiple Devices in JTAG Chain Option



- In the XCF source row, click **Browse** to locate and load the .xcf file that you created in Step 2.
- In the Debug device row, click **Scan** to find the FPGA device.

Note:

Always load the .xcf source file before running scan. If you do not specify the .xcf source file, there will be an error message.

Using JTAGMON

The JTAGMON primitive is a hardware-level anti-tamper mechanism for Avant devices. It monitors the JTAG interface against unauthorized access or any security risk.

JTAGMON instantiates IB (Input Buffers) to receive the JTAG signals. It can log, analyze, or trigger alerts based on JTAG activity. If an unexpected JTAG activity is detected, for example during runtime, it can trigger a system response such as a reset, a lockdown, or an alert.

JTAGMON Input and Output Ports

The following ports allow the primitive to sit inline with the JTAG chain, observing and potentially modifying or blocking signals.

The top level port names of the design connecting to the TDI, TCK, TMS ports of the JTAGMON primitive must be TDI, TCK, and TMS respectively. These can drive both JTAG ports and Fabric routing.

Input Ports	Description
TDI	Test Data Input, one of the pins for the Test Access Port (TAP), JTAG serial interface to the device, connected directly from the pad of device TDI pin.
TCK	Test Clock, one of the pins for the Test Access Port (TAP), JTAG serial interface to the device, connected directly from the pad of device TCK pin. Clocks registers and TAP Controller.
TMS	Test Mode Select, one of the pins for the Test Access Port (TAP), JTAG serial interface to the device, connected directly from the pad of device TMS pin. Controls state machine switching for TAP Controller

Output Ports	Description
TDIO	Monitored Test Data Out
TCKO	Monitored Test Clock Out
TMSO	Monitored Test Mode Select Out

JTAGMON Design Flow

This section details how JTAGMON is implemented in the design flow.

1. RTL Design Instantiation

The RTL design includes an instantiation of the JTAGMON primitive.

Top-Level Inputs: The JTAG interface signals—TDI (Test Data In), TMS (Test Mode Select), and TCK (Test Clock)—are exposed at the top level of the design.

Output Connections: The outputs of the JTAGMON primitive are connected to a soft IP block that monitors these signals. This soft IP could be used for debugging, test access, or signal analysis.

2. Synthesis Phase

In synthesis, the primitive is declared in the synthesis header file.

It uses:

- ▶ `syn_black_box` attribute to indicate it is a black box during synthesis.
- ▶ `black_box_pad_pin` to preserve pin names and prevent optimization.

During synthesis, the tool is configured not to infer input buffers (IBs) for the JTAG signals. This is to preserve the integrity of the JTAG path and allow for proper monitoring and routing.

This ensures that the JTAG signals remain unbuffered and directly accessible, which is critical for accurate signal observation and debugging.

3. Post-Synthesis / Mapping

JTAGMON Expansion: After synthesis and during the mapping phase, the JTAGMON primitive is expanded as input buffers using data from the post-synthesis .udb file.

4. Optional Merge with JTAGhub

Reveal Inserter automatically inserts the current JTAGhub (JTAGH25). If both JTAGH25 and JTAGMON are present, the merge tool combines them, ensuring compatibility and avoiding duplication.


This step ensures that all JTAG-related components are correctly interconnected and that the monitoring infrastructure is unified.

5. Place and Route (PAR)

During the Place and Route (PAR) stage, the tool ensures that the JTAG ports are correctly located on the physical device.

This includes assigning the correct I/O pads and ensuring signal integrity and timing closure for the JTAG interface.

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 25](#).

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.

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.

5. Map, place, and route the design.
6. Generate the bitstream data (.bin 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.

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.

Mapping, Placing, and Routing the Design

Once you build and translate the design, you map, place, and route 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 for successful hardware debugging.

Generating a Bitstream File

Now you generate a bitstream (.bin) file, as appropriate, to download into the device.

To generate a bitstream file:

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

This process creates a .bin 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 the Radiant software, choose **Tools > Programmer**.
6. The Cable Setup Window view is default open.
7. Click **Detect Cable** button.
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. 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 programmer project window appears, enter Filename.
2. Click on Detect Cable button.
3. If there are multiple cables, select one from the Multiple Cables Detected dialog drop down list.
4. Click the ... button under File Name section.

5. Select the `<design_name>.bin` file, and click **Open**.
6. Double click In the Operation box, a new Device Properties dialog appears. Select **Fast Configuration**, in the Operation drop down list.
7. Click **OK** to close the Device Properties dialog box.
8. Choose **Run > Program Device**, or click the **Program Device** button on the toolbar.

After a few moments, the download and programming activity will end. A green PASS button appears in the Status field.
9. Select **File > Save Project As** to save the configuration setup as an .xcf file.
10. The .xcf file must reside in the design project directory for Reveal Analyzer to use it. In the File Name box in the dialog box that appears, type in `<design_name>.xcf`, and click **Save**.
11. 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 80.](#))

How you start Reveal Analyzer also depends on whether you are using it integrated with the Radiant software 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 the Radiant software, go to the Radiant software main window and choose **Tools** >  **Reveal Analyzer**.
 - ▶ For stand-alone in Windows, go to the Windows Start menu and choose **Programs** > **Lattice Radiant Reveal** > **Lattice Reveal Logic Analyzer**.
 - ▶ For stand-alone in Linux, go to a command line and enter the following:

`<Reveal install path>/bin/linux64/revealrva`

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.

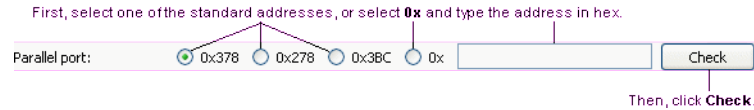
5. In the next row, type in the base name of the file. The extension is added automatically.
6. To the right of this row is a drop-down menu. Choose the type of cable that your board is connected to.

Another row in the dialog box changes to select the port.

7. 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 a USB type.



- ▶ 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 a parallel type.




8. Click **Browse** in the RVL source row to find the Reveal Inserter project (.rvl) file.
9. To add the new .rva file to the File List view, select **Import file into current implementation**. (Not available in stand-alone.) The .rva file works the same either way.
10. 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 Radiant software main window, choose **Tools** >  **Reveal Analyzer**.
 - ▶ The stand-alone Reveal Analyzer in Windows, go to the Windows Start menu and choose **Programs** > **Lattice Radiant Reveal** > **Lattice Reveal Logic Analyzer**.
 - ▶ The stand-alone Reveal Analyzer in Linux, enter the following on a command line:

`<Reveal install path>/bin/linux64/revealrva`

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. To add the new .rva file to the File List view, select **Import file into current implementation**. (Not available in stand-alone.) The .rva file works the same either way.
7. 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 81](#).

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 80](#).
3. Choose **Design > Cable Connection Manager**.
The Cable Connection Manager dialog box opens.
4. In the dialog box, choose the cable type.
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. To change the clock speed of the cable connection, adjust the value of TCK Low Pulse Width Delay.
7. Click **OK**.

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.

To display a Reveal Analyzer core:

- ▶ Choose the only core from the check box. Only one core is supported.

Setting Up the Trace Signals

Although you can add trace signals only in Reveal Inserter, you can set radices 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 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 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 [“Triggering” on page 32](#) 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 [“Triggering” on page 32](#) 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. In the POR Debug section, if you wish to connect POR when Analyzer opens, click the **Connect POR when Analyzer opens** check box.
6. 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

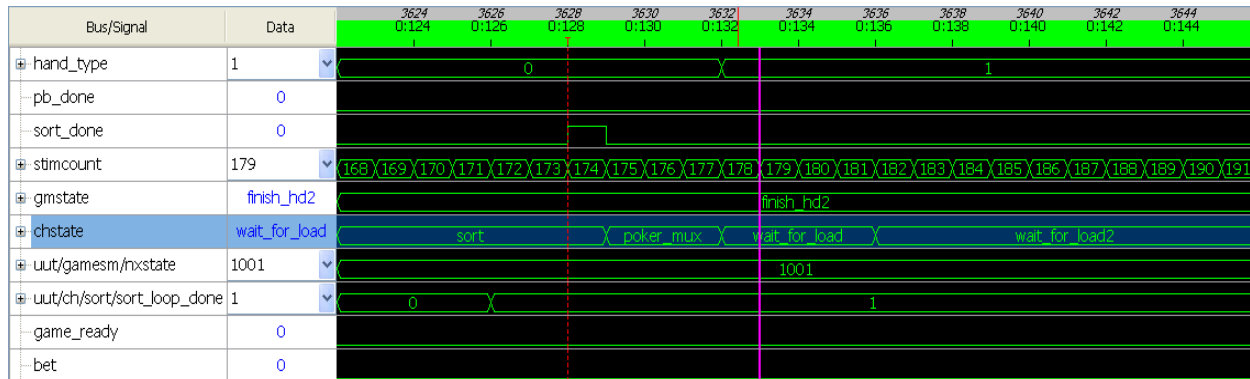
You can set a number of options to control the triggering.

1. **To set trigger options in Reveal Analyzer:**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.
2. 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.
3. In the **Trigger Position** field, select **Pre-selected Position** or **User-selected Position**.

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 14: LA Waveform View Using Tokens



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, click **Add 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 click **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'	x'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 click **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 **Close** to close the dialog box. The token sets are automatically applied to the current Reveal project.

Debugging with Reveal Controller

Reveal Controller is another divide-and-conquer mechanism for you to emulate an otherwise unavailable environment for power debug. For example, your evaluation board would only have a limited number of LEDs or switches but the virtual environment enables up to 32 bits. Register memory mapping and dumping of values is also easily manifested while visibility into Hard IPs is also enabled.

Virtual LED Switch Console

Once Reveal Analyzer is set up and started, you are presented with three tabs, the first of which is a virtual LED/Switch console to emulate the board.

The top section, Virtual LED shows all the LEDs that were defined in the insertion stage. Once the board is running, you can see the LEDs flash in red and green as they would in the real hardware environment.

The lower section shows the virtual switch in which you can either enter the data as a hex value or can set the virtual dip switches.

Select Direct Mode to see the data and switches in real time instead of waiting until the Apply button is clicked.

Note

Switches are of type in/out. When defining switches in RTL, use the WIRE definition otherwise multiple input driver errors will be encountered later in the flow.

Assigning Colors to Virtual Switch/LED Signals

To assign colors to virtual switch/LED signals:

1. In the Switch List and LED List tables, click the color of the signal.
2. This opens the Select Color dialog box. Choose the color to assign to the signal and click **OK**.

The colors you assign are also reflected in the virtual LED/Switch console in Reveal Analyzer.

User Memory Analysis

The second tab is for User Memory Analysis. The title indicates the range of the User Memory map. Analysis follows no particular steps or order.

- ▶ Default Data: A value that will initialize all memory locations.
- ▶ Write Address/Data: Address and data in hex to be written.
- ▶ Read Address/Data: Enter address. The data is read back when 'Read' is selected.
- ▶ Memory File: You can dump from/to a range of memory addresses to a .mem file. A user can also **Load MemFile** to load a pre-configured memory file.

Running User Control Register

1. In the **User Control Register** tab, select a label from the **Name** column, then click the **Rd** button.

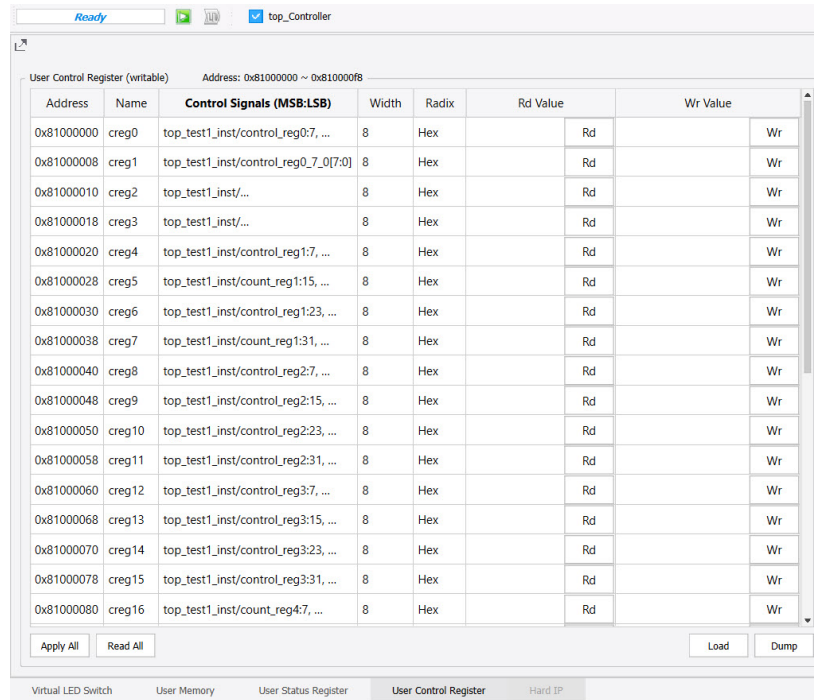
User Control Register reads the value of the corresponding address location and executes the following TCL command:

```
rva_run_controller -read_control "label"
```

2. Select a label from the Name column.
3. In the **WR Value** column, do the following:
 - a. Enter desired value.
 - b. Click the **Wr** button.

User Control Register executes the following TCL command:

```
rva_run_controller -write_control "label" -data value
```



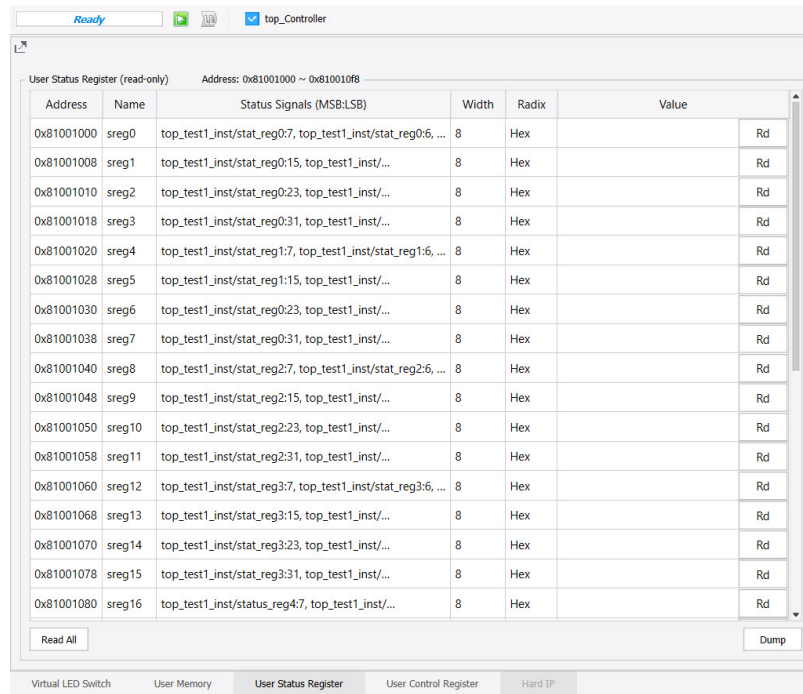
- Double-click a row in the **Control Signals** column to display the list of signals with their corresponding value.
- In the User Control Register tab, click the **Apply All** button to apply the value of all address locations assigned by the Name column and execute multiple TCL commands sequentially.
- Click the **Read All** button to read the value of all address locations assigned by the Name column and execute multiple TCL commands sequentially.
- Click the **Load** button to load the value from the register file to all address locations of Control Register.
- Click the **Dump** button to save the value of all address locations to a register file.

Running User Status Register

- In the **User Status Register** tab, select a label from the **Name** column, then click the **Rd** button.

User Status Register reads the value of the corresponding address location and executes the following TCL command:

```
rva_run_controller -read_status "label"
```



- On the bottom left corner, click the **Read All** button.

The Read All button reads the value of all address locations assigned by the Name column and executes multiple TCL commands sequentially.

- On the bottom right corner, click the **Dump** button.

The Dump button saves the value of all address locations to a register file.

- Double-click a row in the **Status Signals** column to display the list of signals with their corresponding value.

Hard IP Debug

All the IP selected for analysis in Reveal Inserter are displayed here.

Main operations are reading/writing to memory locations.

Note

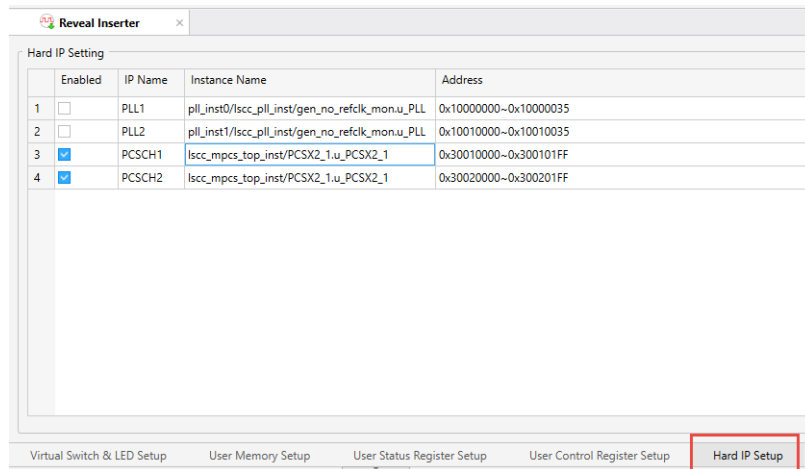
The detailed settings of each IP is beyond the scope of this help. Links will be provided to Application Notes to explain its usage.

Debugging Nexus PCS/SerDes

Radiant supports PCS/SerDes debugging for devices built on the Nexus platform such as CertusPro-NX.

To debug Nexus PCS/SerDes:

1. Use the Reveal Inserter tool to add controller cores to the design. These cores allow writing and reading the control and status registers and change the parameters for PCS/Serdes operations.
2. If your design has a hard IP, the Hard IP Setup tab displays all your hard IPs. Under the Enable column, click the check box of an IP to enable the debug function.



3. Generate the updated bitstream with the inserted debug cores and program the device.
4. Open Reveal Analyzer/Controller and connect to the programmed device through USB/JTAG.

5. In the Hard IP tab, you will see the instantiated hard IP blocks on the Nexus device. Scroll to the IP core to debug..

▼ PCSCH1

Address (0x30010000 ~ 0x300101FF)
Instance Hierarchy: lsc_mpcs_top_inst/PCXSX2_1.u_PCSX2_1

Transmit Settings

Differential Amplitude: 0mV
Output Termination: 100 ohm
Tx Post-Cursor Ratio: 0
Tx Pre-Cursor Ratio: 0
Invert Tx Data Polarity: off
Tx PLL Loss of Lock Status: unlocked

Receive Settings

Input Termination: 100 ohm
Rx EQ: SS_LMS 2.5GT/s
Invert Rx Data Polarity: off
CDR Loss of Lock Status: unlocked
Data Loss of Lock Status: unlocked

Loopback Mode Settings Note: The "Reload" function will not update th
Loopback Mode: Disabled

Serdes/PCS Reset

MPCS Tx Reset: off
MPCS Rx Reset: off
PMA Tx Reset: off
PMA Rx Reset: off
PMA Tx HiZ: off
PMA Rx HiZ: off

Apply Reload

Write Address: 0x30010000
Write Data: 0x0 Write

Read Address: 0x30010000
Read Data: Read

Dump From (Offset): 0x00000000 Range: Dump
Dump MemFile: ...
Load MemFile: ... Load

Virtual LED Switch User Memory User Status Register User Control Register **Hard IP**

6. Set the control parameters and click the **Apply** button.
7. Run your design.
8. Analyze the captured data. You can modify settings if allowed.

Click the **Eye Opening Monitor** button to view a graphical eye diagram that details the signal eye width and height. See [“Running Reveal Eye-Opening Monitor” on page 101](#).

Debugging Avant with SerDes Toolkit

In designs using Avant MPPHY module, Radiant supports SerDes debugging using the SerDes toolkit. This feature allows you to evaluate and debug the performance of transceivers. It generates and checks data patterns to measure the bit error rate (BER), which helps in identifying signal integrity issues. In Reveal Inserter, set up hard IP by clicking the Hard IP Setup tab and then select functions by enabling check boxes.

You can test and validate transceiver links. You can verify that the transceivers are functioning correctly and can handle the data rates required in your design. By analyzing the BER, you can identify and diagnose issues such as signal degradation, noise, or incorrect settings. You can also optimize transceiver configurations by adjusting various transceiver parameters.

SerDes toolkit features:

- ▶ Real-time access to transceiver settings. This allows you to monitor and modify the settings of the transceiver on-the-fly while the system is running.
 - ▶ Adjust Parameters: Use different transceiver and clocking topologies. Change settings such as pre-emphasis, equalization, and data rate in real-time to see how they affect the signal quality.
 - ▶ Optimize Performance: Fine-tune the transceiver settings to achieve the best possible performance for your specific board configuration.
 - ▶ Troubleshoot Issues: Quickly identify and resolve issues by making adjustments and immediately observing the impact on the link quality.
- ▶ Sweep testing. Automate the process of testing various settings to find the optimal configuration. The AutoSweep feature with interval-stepping capability allows quick testing of many combinations of physical medium attachment (PMA) analog settings to find the optimal setting for a particular transceiver link.

Note

You can capture Eye Diagram data while auto-sweeping the PMA settings.

- ▶ Data pattern generators and checkers select transceiver.
- ▶ Multiple different PRBS (Pseudo-Random Binary Sequence) and clock patterns transmitted over the channels.
- ▶ Channel manager interface monitor and track status while testing multiple channels simultaneously.
- ▶ Running link tests between multiple devices across one or more boards.
- ▶ Save and Load capability that enables comparison of different sets of signal integrity results including the eye diagram. This capability enables customers to compare the signal integrity results for different PMA settings.

After the device is programmed, SerDes Toolbox, TX, and RX settings are displayed in the Hard IP tab of the Reveal Controller interface.

Start Page | Reports | Reveal Analyzer/Controller | top_Controller

Ready

MPPCIEX8AQ1

MPPX4 Address (0x30000000 ~ 0x3000FFFF)

Instance Hierarchy: inst_m

Assigned Quad: Quad-A

SerDes Toolbox

	Lane0	Lane1	Lane2	Lane3
SerDes Toolbox	Read	Read	Read	Read
Loopback Mode	Disabled	Disabled	Disabled	Disabled
TX PRBS Pattern	Disabled	Disabled	Disabled	Disabled
RX PRBS Pattern	Disabled	Disabled	Disabled	Disabled
PAT0	0	0	0	0
Restart	Restart	Restart	Restart	Restart
Inject 1-bit Error	Inject	Inject	Inject	Inject
PRBS Error Count	0	0	0	0

TX Settings

	Lane0	Lane1	Lane2	Lane3
TX	Read	Read	Read	Read
PLL Lock Status	Unlocked	Unlocked	Unlocked	Unlocked
TX Data Polarity	Non-invert	Non-invert	Non-invert	Non-invert
TX Data Width	8-bit	8-bit	8-bit	8-bit
TX Data Enable	Off	Off	Off	Off
TX Equal Pre-cursor	0	0	0	0
TX Equal Main-cursor	0	0	0	0
TX Equal Post-cursor	0	0	0	0
TX Termination	54-ohm	54-ohm	54-ohm	54-ohm
TX Diff Amplitude Enable	Off	Off	Off	Off
TX Diff Amplitude (iboot)	0	0	0	0
TX Diff Amplitude (vboost)	0	0	0	0

RX Settings

	Lane0	Lane1	Lane2	Lane3
RX	Read	Read	Read	Read
CDR Lock Status	Unlocked	Unlocked	Unlocked	Unlocked

Virtual LED Switch | User Memory | User Status Register | User Control Register | Hard IP

Clicking the **Read** button of a specific lane shows the required settings from the device and populates the values for the lane. Inactive lanes are grayed out.

Table 2: SerDes Toolbox Settings

Settings	Display/Values	Description
Loopback Mode	DISABLED (Default)	This is a testing configuration where the transmitted signal is looped back to the receiver without leaving the device.
	Near-End Serial	In Near-End Serial, the transmitted signal is looped back at the near end of the communication link, close to the transmitter.
	Parallel	In Parallel, the transmitted data is looped back in parallel.
		Multiple data lanes are used simultaneously for the loopback test.
TX PRBS Pattern	DISABLED (Default)	This is the PRBS (Pseudo-Random Binary Sequence) pattern generated and sent by the transmitter side of the communication link.
	PRBS31	The PRBS pattern is used to test the integrity of the link by sending a known sequence of bits and checking if the received sequence matches the expected pattern.
	PRBS23_18	
	PRBS23_21	
	PRBS16	<ul style="list-style-type: none"> ▶ PRBS<BITS> PRBS patterns with more number of bits (long pattern) are particularly useful for jitter measurement. ▶ FIXED WORD is a single unchanging predefined sequence. ▶ DC balanced refers to a pattern that includes an equal number of ones and zeros. This balance helps keep the DC level steady and reduces signal degradation, which is particularly important when the signal passes through AC-coupling circuits.
	PRBS15	
	PRBS11	
	PRBS9	
	PRBS7	
	FIXED WORD (PAT0)	
	DC BALANCED (PAT0, ~PAT0)	
	FIXED PATTERN (000, PAT0, 3FF, ~PAT0)	<ul style="list-style-type: none"> ▶ FIXED PATTERN is a predefined sequence that is a combination of multiple fixed words or sequences.
		When selected, the Restart button takes on the value.
RX PRBS Pattern	DISABLED (Default)	This is the PRBS pattern generated and sent by the receiver side of the communication link.
	PRBS31	The PRBS pattern is used to test the integrity of the link by sending a known sequence of bits and checking if the received sequence matches the expected pattern.
	PRBS23_18	
	PRBS23_21	
	PRBS16	<ul style="list-style-type: none"> ▶ PRBS<BITS> PRBS patterns with more number of bits (long pattern) are particularly useful for jitter measurement. ▶ FIXED WORD is a single unchanging predefined sequence. ▶ DC balanced refers to a pattern that includes an equal number of ones and zeros. This balance helps keep the DC level steady and reduces signal degradation, which is particularly important when the signal passes through AC-coupling circuits.
	PRBS15	
	PRBS11	
	PRBS9	
	PRBS7	
	FIXED WORD (PAT0)	
	DC BALANCED (PAT0, ~PAT0)	
	FIXED PATTERN (000, PAT0, 3FF, ~PAT0)	<ul style="list-style-type: none"> ▶ FIXED PATTERN is a predefined sequence that is a combination of multiple fixed words or sequences.
		When selected, the Restart button takes on the value.
PAT0		Enter the pattern:
		▶ 8 bits for 16-bit data width
		▶ 10 bits for 20-bit data-width

Table 2: SerDes Toolbox Settings

Settings	Display/Values	Description
Restart	Restart button	Clicking this button clears the TX and RX pattern settings and sets the selected Patterns and sync.
Inject 1-bit Error	Inject 1-bit Error button	Clicking this button inserts a 1-bit error in LSB of TX word.
PRBS Error Count	PRBS Error Count button	Clicking this button reads the error count and displays it in the cell.

Table 3: TX Settings

Settings	Display/Values	Description
PLL Lock Status	Green = Locked Red = Unlocked	When the PLL (Phase-Locked Loop) is locked, the output signal stays in sync with the input signal, maintaining a steady phase relationship. If the PLL becomes "unlocked," it indicates that synchronization failed, and the output signal is no longer aligned with the input signal.
TX Data Polarity	Non-inverted = Unchecked Inverted = Checked	Non-Inverted Data Polarity means that the data signal is transmitted without any change in its original form. The logical '1' (high) and logical '0' (low) levels of the signal are maintained. Inverted Data Polarity means that the data signal is flipped or transmitted in an inverted form. In this case, the logical '1' becomes '0' and the logical '0' becomes '1'.
TX Data Width	8-bit 10-bit 16-bit 20-bit 32-bit 40-bit	TX data width is the number of bits that can be transmitted simultaneously over a channel.
TX Data Enable	Off On	When the TX Data Enable signal is On, the data on the TX line is valid and ready for transmission. When the signal is Off, the data on the TX line is not valid, and the transmission should be ignored.
TX Equal Pre-cursor	0.0 to 6.75 with 0.25 each step	The adjustment of the signal before the main data pulse to compensate for distortions and signal degradation caused by the transmission channel. Pre-conditioning the signal with pre-cursor equalization reduces inter-symbol interference (ISI) and enhances overall signal quality at the receiver's end.
TX Equal Main-cursor	0 to 24	The primary coefficient (main cursor) is used as the key value in determining the total equalization. The main cursor coefficient represents the central, most significant part of the transmitted signal. It is the primary reference point in the signal that should have the highest amplitude.

Table 3: TX Settings

Settings	Display/Values	Description
TX Equal Post-cursor	0.0 to 8.75 with 0.25 each step	The adjustment of the signal after the main data pulse to compensate for distortions and signal degradation caused by the transmission channel. Pre-conditioning the signal with post-cursor equalization reduces inter-symbol interference (ISI) and enhances overall signal quality at the receiver's end.
TX Termination	54-ohms 52-ohms 50-ohms 48-ohms 46-ohms 44-ohms 42-ohms 40-ohms	The termination resistor value on the transmission side of the communication link. This resistor matches the transmission line impedance and minimizes signal reflections.
TX Diff Amplitude Enable	Off On	When the TX Diff Amplitude Enable setting is turned On, the transmitter can generate a differential signal with the appropriate amplitude so that the signal can be accurately interpreted by the receiver, even in the presence of noise.
TX Diff Amplitude (iboot)	0 to 15	The strength of the amplitude boost applied to the differential signal. This help helps the transmitted signal to be accurately received despite noise or interference.
TX Diff Amplitude (vboost)	0 to 7	

Table 4: RX Settings

Settings	Display/Values	Description
CDR Lock Satus	Green = Locked Red = Unlocked	When the CDR (Clock and Data Recovery) the data is correctly sampled and the clock is properly aligned with the data transitions. If the CDR is unlocked, data errors may occur due to misalignment between the clock and data signals.
RX Signal Status	Green = Valid Red = Invalid	A valid RX signal status indicates that the received data is correctly aligned with the clock signal. An invalid RX signal means that the received data is not correct or cannot be properly interpreted. This may be due to signal integrity issues, clock misalignment, protocol errors, or data corruption.

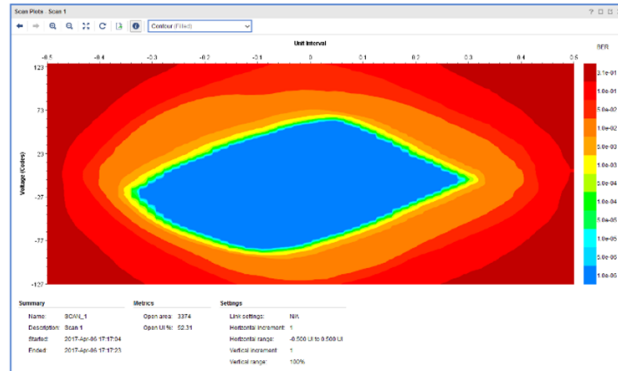
Table 4: RX Settings

Settings	Display/Values	Description
RX Loss of Signal Threshold	60mVpp	This value is the minimum signal level required to maintain reliable data transmission. When the signal falls below this threshold, the system detects a loss of signal.
	80mVpp	
	100mVpp	
	120mVpp	
	140mVpp	
	160mVpp	
	180mVpp	
RX Data Polarity	Non-inverted = Unchecked	Non-Inverted Data Polarity means that the data signal is received without any change in its original form. The logical '1' (high) and logical '0' (low) levels of the signal are maintained. Inverted Data Polarity means that the data signal is flipped or received in an inverted form. In this case, the logical '1' becomes '0' and the logical '0' becomes '1'.
	Inverted = Checked	
RX Data Width	8-bit	RX data width is the number of bits that can be received simultaneously over a channel.
	10-bit	
	16-bit	
	20-bit	
	32-bit	
	40-bit	
RX Equal Afe Rate	0 to 7	The "adaptive equalization filter rate" setting that adjusts the equalization parameters dynamically to optimize signal integrity and minimize bit errors.
RX Equal Atten Level	0 to 7	The receiver equalization attenuation level setting adjusts the attenuation applied to the received signal to optimize signal integrity and minimize bit errors.
RX Equal Ctle Pole	0 to 3	This value indicates the pole location of the Continuous-Time Linear Equalizer (CTLE) in the receiver path.
RX Equal Ctle Boost	0 to 31	This value refers to the CTLE boost in the receiver path. Boosting the CTLE increases its gain, especially at higher frequencies.
RX Termination	54-ohms	The termination resistor value on the receiver side of the communication link. This resistor matches the transmission line impedance and minimizes signal reflections
	52-ohms	
	50-ohms	
	48-ohms	
	46-ohms	
	44-ohms	
	42-ohms	
	40-ohms	
RX Termination Acdc	DC (Default)	This indicates whether the receiver termination is set to an alternating current (AC) or a direct current (DC) coupling mode.
	AC	

Table 4: RX Settings

Settings	Display/Values	Description
RX Equal Vga1 Gain	0 to 7	The value indicates the amplification level of the signal received by VGA2.
RX Equal Vga2 Gain	0 to 7	The value indicates the amplification level of the signal received by VGA2.

Clicking the Eye Opening Monitor button provides a graphical eye diagram that details the signal eye width and height.



See [“Running Reveal Eye-Opening Monitor” on page 101](#) for more information.

Running Reveal Eye-Opening Monitor

The Reveal Eye-Opening Monitor can be launched from the Reveal Analyzer/Controller. You need to be connected to a board to be able to run the Eye-Opening Monitor.

To launch the Eye-Opening Monitor:

1. In the Radiant software main window choose **Tools > Reveal Analyzer/Controller**

In the drop-down menu, choose **top_Controller** to display the **Hard IP** tab.

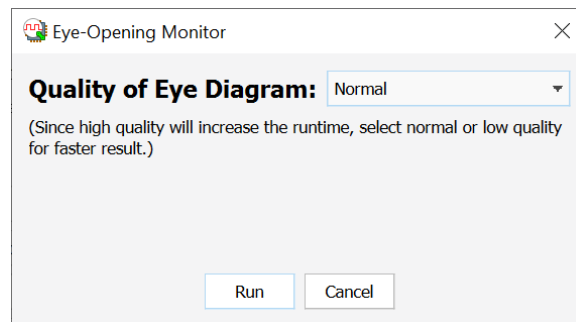
2. Click the **Hard IP** tab.

If debugging a non-Avant device, scroll down to the PCS channels to locate the **Eye-Opening Monitor** button.

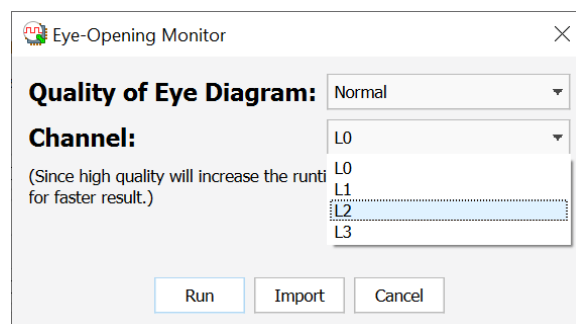
If debugging an Avant device, the **Eye-Opening Monitor** button is located at the upper-right of the current IP instance.

3. Click the **Eye-Opening Monitor** button. The Eye-Opening Monitor dialog box is displayed.

The figure below shows the Eye-Opening Monitor dialog box when debugging a non-Avant device.



The figure below shows the Eye-Opening Monitor dialog box when debugging an Avant device. The Channel and Import options are added.



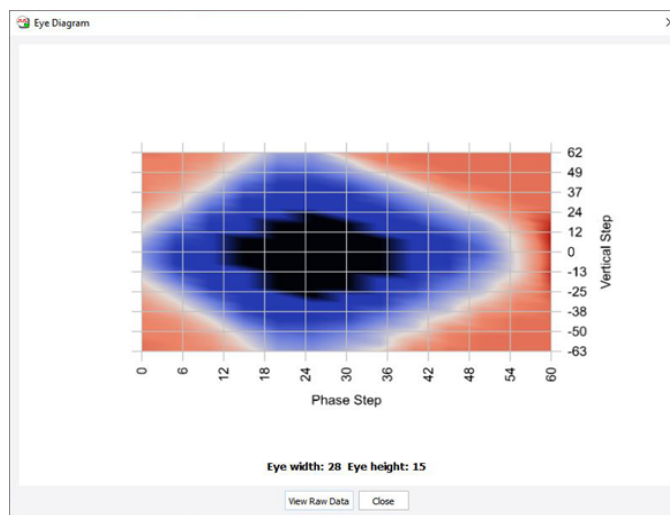
4. In the Eye-Opening Monitor dialog box, select the quality of the eye diagram from the drop-down menu.

It is recommended to select normal or low quality for a faster result, and select high quality for the final result. The default is normal quality.

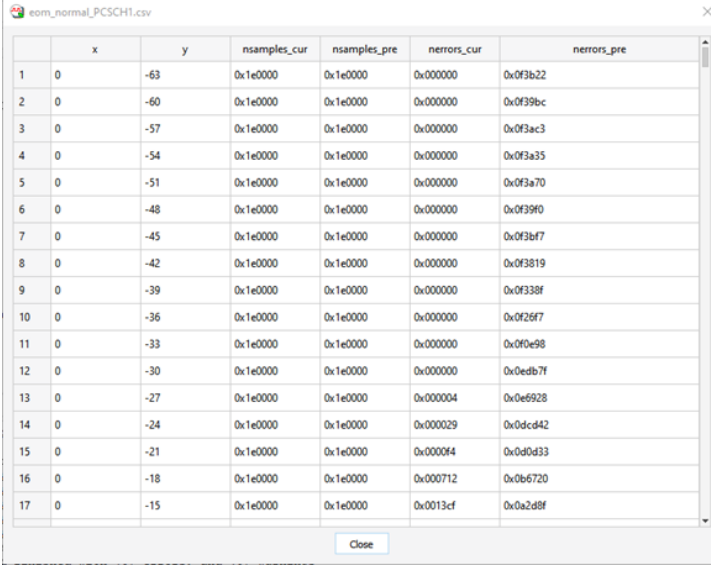
If you are debugging Avant SerDes multiple channels, select the channel to debug in the Channel drop-down menu.

5. Click the **Run** button to start the EOM process. You can stop the process at any time by clicking the **Stop** button.
6. After the EOM has finished running, it will display the eye diagram for the current channel. The figure below shows the Eye Diagram when debugging non- Avant devices.

Figure 15: Example of Eye Diagram for Non-Avant Devices



At the bottom of the eye diagram for non-Avant devices, you can click the **View Raw Data** button to open the raw data file containing the number of samples and errors used in calculating the density of the eye diagram.



	x	y	nsamples_cur	nsamples_pre	nerrors_cur	nerrors_pre
1	0	-63	0x1e0000	0x1e0000	0x000000	0x0f3b22
2	0	-60	0x1e0000	0x1e0000	0x000000	0x0f39bc
3	0	-57	0x1e0000	0x1e0000	0x000000	0x0f3ac3
4	0	-54	0x1e0000	0x1e0000	0x000000	0x0f3a35
5	0	-51	0x1e0000	0x1e0000	0x000000	0x0f3a70
6	0	-48	0x1e0000	0x1e0000	0x000000	0x0f39f0
7	0	-45	0x1e0000	0x1e0000	0x000000	0x0f3bf7
8	0	-42	0x1e0000	0x1e0000	0x000000	0x0f3819
9	0	-39	0x1e0000	0x1e0000	0x000000	0x0f38f8
10	0	-36	0x1e0000	0x1e0000	0x000000	0x0f26f7
11	0	-33	0x1e0000	0x1e0000	0x000000	0x0f0e98
12	0	-30	0x1e0000	0x1e0000	0x000000	0x0edb7f
13	0	-27	0x1e0000	0x1e0000	0x000004	0x0e9928
14	0	-24	0x1e0000	0x1e0000	0x000029	0x0dcd42
15	0	-21	0x1e0000	0x1e0000	0x0000f4	0x0d0d33
16	0	-18	0x1e0000	0x1e0000	0x000712	0x0b6720
17	0	-15	0x1e0000	0x1e0000	0x0013cf	0x0a2d8f

- ▶ The raw data comes from the eom register.
- ▶ The numbers from nsample_cur, nsample_pre, nerrors_cur, and nerrors_pre columns calculate the result.
- ▶ The numbers are collected by reading the eom register of the PCS IP.

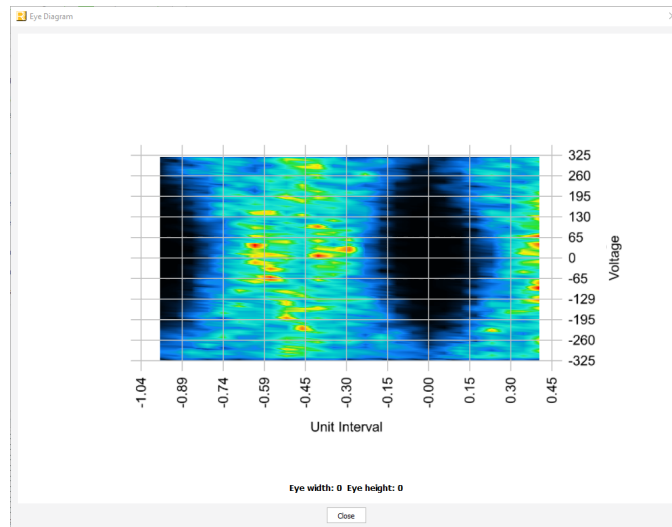
If you change the Reveal Controller options, you can re-run the EOM to display the new eye diagram and view the new raw data until the desired result is achieved.

Importing an Existing EOM Result

If you are debugging Avant SerDes, each time you run the EOM, the result is automatically saved as a .txt file in the project folder. The file name provides information on the EOM settings. For example, the file name *eom_normal_MPPHYX4Q1_L2.txt* tells you that the EOM result was generated using normal quality for the MPPHYX4Q1 instance with L2 or lane 2 as the selected channel.

To import an existing EOM result:


1. Click the **Import** button in the Eye-Opening Monitor dialog box.
2. The Select Existing EOM File dialog box is displayed. Selecting AVANT EOM (*.txt) displays only the EOM results for debugging the Avant device.
3. Select the .txt file and click **Open**. The Eye Diagram is displayed. The figure below shows the Eye Diagram when debugging Avant devices.


Figure 16: Example of Eye Diagram for Avant Devices

Performing Logic Analysis

After you have configured trigger settings in the LA Trigger tab, you can perform a logic analysis on a single core.

To perform logic analysis:

1. In the `<design_name>_LA<core_number>` check box in the toolbar, select the core 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 106.

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 discontinuous, 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

Please refer to the Reveal Troubleshooting guide for more information.

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

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

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 Analysis

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 the Radiant software, choose **Tools > Options**. Then, in the Options dialog box, choose **Color** in the left panel and then click on the **Reveal Analyzer** tab.
 - ▶ 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 18 on page 118. 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. 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 109](#).
2. 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.
3. Right-click in the waveform and choose **Export Waveform**.
The Export Waveform dialog box opens.
4. Browse to the location where you want to export the file.
5. Type in a name in the **File name** box.
6. Choose a file type.
7. If you are exporting only some of the signals, choose **Selected signals** in the Range box.

8. 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>".
9. 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**.

Exiting Reveal Analyzer

To exit Reveal Analyzer:

- ▶ Choose **File** > **Close**.

User Interface Descriptions

The Reveal Analyzer window consists of the LA Trigger, LA Waveform.

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:

Name Specifies the name of the trigger unit. See [“Trigger Expression and Trigger Unit Naming Conventions” on page 118](#) 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 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 five 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 17: Trigger Expression Sequences

Enable	Name	Expression	Sequence Depth	Max Sequence Depth	Max Event Counter
<input checked="" type="checkbox"/>	TE1	TU1 & TU2	1	2	1
<input checked="" type="checkbox"/>	TE2	TU1 TU2 THEN TU3 & TU2	2	2	1
<input checked="" type="checkbox"/>	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 65,536. The maximum is 65,536. 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.

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.

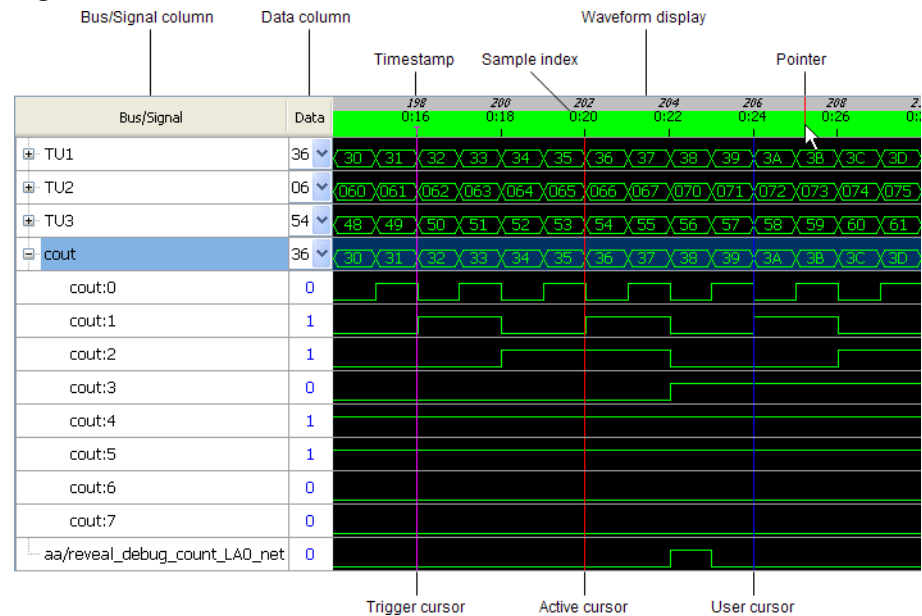
Trigger Expression and Trigger Unit Naming Conventions

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 18 along with several features to help you find and analyze the data.

Figure 18: Elements of the LA Waveform View



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

Note:

When you copy and paste a duplicate or incorrect signal, you can delete it using either of the following keyboard shortcuts:

- ▶ Shift + Delete
- ▶ Ctrl + X

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 82](#).

Waveform Display Displays the trace data in waveform format. When there is room, bus values are included in 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 82](#).
- ▶ **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 110](#).

Viewing and Saving Waveforms

After capturing data, you can view the trace data in waveform format in the LA Waveform tab. Whenever the trace stops, Reveal Analyzer reads the trace samples and automatically updates the signal waveforms.

To view a waveform:

1. Click the **LA Waveform** tab.
2. Choose a module from the pulldown menu in the Reveal Analyzer tool bar.

Sometimes the waveform includes fewer clock cycles than you expect; in particular, fewer before the trigger. This happens if the trigger occurs so quickly after the test starts that there are not enough clock cycles before the trigger to fill that part of the trace buffer.

You can also save the captured waveform as an RVA file and reload it when you reopen Reveal Analyzer. The saved waveform carries most of the

customization that you applied such as color changes, bus grouping, cursors, and others (except for zoom in/out information.)

To save a waveform:

1. Choose **File > Save <file> As**.
The Save Reveal Analyzer File dialog box appears.
2. Browse to the directory in which to save the project.
Make sure to save the .rva in the same directory as .rvl project file.
3. In the File name box, type the file name.
4. Click **Save**.

Using the Memory Controller Debug

The memory controller debug feature for Avant devices provides an interactive and targeted method for diagnosing and resolving DDR interface issues or stress-testing your design.

- ▶ Initiate various types of memory training routines such as write leveling, read gate training, and VREF training.
- ▶ Write directly to DDR mode registers and fine-tuning memory parameters such as drive strength, termination, and on-die VREF settings.
- ▶ Immediately assess the impact of your changes through the Margin Report and VREF Report. These are visual and quantitative feedback on signal margins and voltage reference calibration.
- ▶ Stress-test your design, even if not currently experiencing issues. You can create traffic patterns based on your use cases. Test different types of memory access patterns and see how the memory controller handles them.

The following section describes the memory controller debug feature. For complete information on the IP, see [FPGA-IPUG-02195, DDR Memory PHY Module](#).

Enabling the Memory Controller Debug

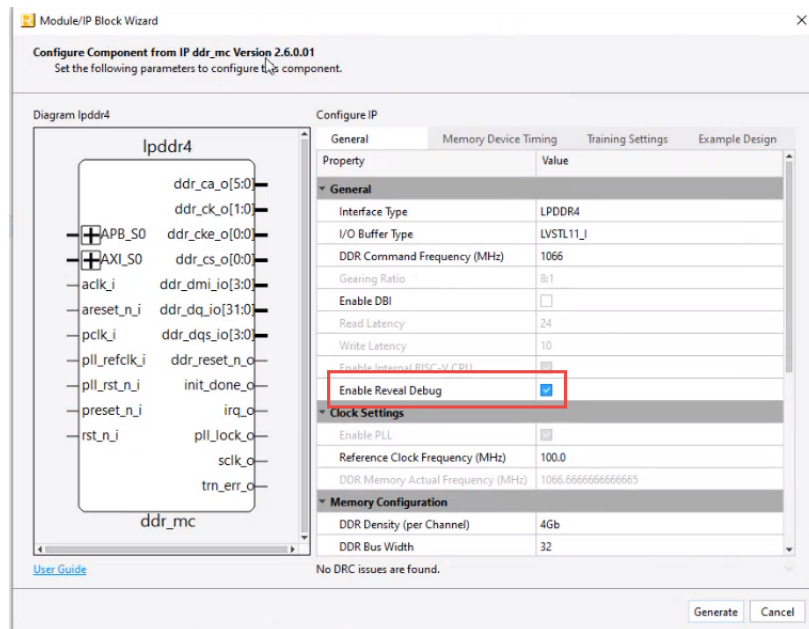
Enabling the memory controller debug is a two-step process:

- ▶ Enable Reveal debug during IP configuration
- ▶ Enable memory controller debug in Reveal Inserter

To enable Reveal debug during IP configuration:

1. In the Module/IP tree, select the component to generate.
2. Configure your settings in the Module/IP Block Wizard.

Check the **Enable Reveal Debug** option.



3. Click **Generate**.

To automatically import the .ipx file into your project when the component is generated, select **Insert to project** in the Check Generating Result dialog box.

4. Click **Finish**.

To enable the memory controller debug in Reveal Inserter:

1. Open Reveal Inserter. select **RTL (Pre-Synthesis)** in Debug Stage.
2. Click **Finish**.
3. In Datasets, click **Add Controller** to add a Controller core.
4. Click the **Hard IP Setup** tab.
5. Under Hard IP Setting, enable the IP by selecting its check box on the Enabled column.
6. Click the **Insert Debug** button.
7. In the Insert Debug to Design dialog box, select the modules to insert.
8. Select **Activate Reveal file in design project**.

If the .rvl file is not active in the design project, the Reveal modules will not be included during synthesis.

9. Click **OK**.
10. In the Save Reveal Project dialog box, indicate the file name and click **Save**.

The active .rvl file is listed in the File List view under Debug Files.

11. Click **Export Files** to run the design flow and insert the Reveal controller.

Downloading the Bitstream and Starting a Reveal Analyzer Project

If the correct setup process is followed and Reveal inserter is properly configured, download the bitstream and start a new project in Reveal Analyzer/Controller.

Connect the board with the Avant device to your computer. Using Radiant Programmer, download the bitstream to the board. Refer to [“Programming the FPGA” on page 445](#).

After downloading the bitstream and making sure that there are no errors, open the Reveal Analyzer/Controller to create a new project.

Running Calibration

To open the memory controller debug interface and run calibration:

1. In Reveal Analyzer/Controller, choose **top_Controller** in the drop-down menu.
2. Click the **Hard IP** tab.

You can separate the tab by clicking the detach icon at the upper right corner.

3. The **Memory Interface Calibration** tab opens.

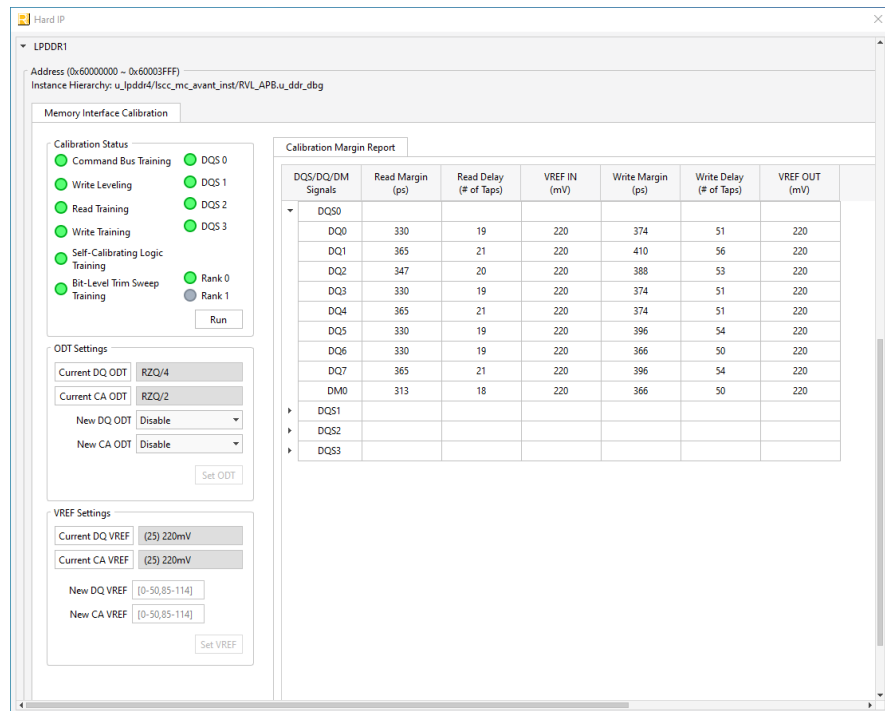
You can view the status of various memory controller debug tuning tests such as command bus training, write leveling, read training, and write training. You can also read and write from the mode registers of their memory controller IP.

4. Click the **Run** button.

This shows the calibration status, calibration margin report, current ODT, and VREF settings.

The Calibration Margin Report tab shows the margin for various DQS/DQ, command/control, and address signals.

For detailed information on the available options, refer to [FPGA-IPUG-02195, DDR Memory PHY Module](#).



When you run the calibration, a .csv file of the results is automatically saved in the directory of your .rva file.

Note

This feature is repeatable with TCL commands, with or without the GUI.

Best Practices

This section presents a set of best practices for leveraging the Lattice Radiant Reveal debug tool in FPGA development workflows. Reveal offers visibility into internal signal behavior, enabling precise timing analysis, logic verification, and system-level debugging. Designed for engineers familiar with RTL design and synthesis, this best practices guide focuses on optimizing trace configurations, minimizing resource overhead, and integrating Reveal into iterative development cycles. By applying these practices, you can enhance debugging efficiency, accelerate root cause analysis, and improve overall design robustness.

This documents includes the following sections:

- ▶ [“General Guidelines” on page 125](#)
- ▶ [“Best Practices for Post-Synthesis Flow” on page 126](#)

This best practices guide is a **work in progress**. As Reveal continues to evolve and new use cases emerge, additional recommendations, clarifications, and examples will be incorporated. Users are encouraged to refer to the latest version of this document and provide feedback to help improve its accuracy and relevance.

General Guidelines

To maximize the effectiveness of the Reveal debug tool and ensure accurate signal capture and analysis, follow the general usage guidelines in this section.

▶ **Prefer Registered Signals for Trigger and Trace**

Always select registered signals (such as signals driven by flip-flops for example) for both trigger and trace. Avoid combinatorial signals, as they are prone to glitches and may produce unreliable or misleading trace data due to their asynchronous nature.

▶ **Use `syn_rvl_debug` Attributes in RTL**

Annotate the registered signals intended for debugging with the `syn_rvl_debug` attribute directly in the RTL code. This highlights the signals and ensures that they are preserved through synthesis and made available for Reveal debug configuration.

▶ **Debug at the Post-Synthesis Stage**

Perform debugging at the post-synthesis stage to ensure that the signals selected for tracing reflect the actual synthesized netlist. This stage provides a more accurate representation of the design and avoids discrepancies between RTL and implementation.

▶ **Select the Sample Clock First in Inserter GUI**

In the Reveal Inserter GUI, begin by selecting the sample clock. This clock determines the timing domain for signal sampling and must be chosen before selecting any signals.

▶ **Group Signals by Clock Domain**

After selecting the sample clock, choose only the signals that belong to the same clock domain for a single analyzer core. Mixing signals from different domains within the same core can lead to incorrect timing analysis and data corruption.

▶ **Use Separate Analyzer Cores for Different Clocks**

For designs with multiple clock domains, instantiate separate analyzer cores for each domain. This ensures proper synchronization and avoids clock domain crossing issues during trace capture.

▶ **Review the Trace Report for CDC Issues**

After instrumentation, carefully examine the trace report for any clock domain crossing (CDC) warnings or errors. These issues can compromise the integrity of captured data and should be resolved before proceeding with debug.

▶ **Minimize Trigger Unit Complexity in Tight Timing Designs**

If your design is close to failing timing constraints, reduce the number of signals in the trigger unit. Each additional signal can increase logic depth and routing complexity, potentially pushing the design over timing limits.

Best Practices for Post-Synthesis Flow

The post-synthesis debug flow in Reveal provides a more accurate and implementation-aware environment for signal tracing and analysis. At this stage, the design has undergone synthesis, allowing you to work with actual netlist data rather than abstract RTL representations. This section outlines key strategies for configuring Reveal effectively during post-synthesis.

Guidelines when Updating User Design in Post-Synthesis Debug Flow

▶ **Disable All Active Reveal Projects (*.rvl)**

Before making any changes to the design, deactivate all active Reveal projects. This prevents conflicts between the existing debug configuration and the updated design, and ensures that Reveal does not attempt to instrument outdated netlist data.

▶ **Synthesize the User Design After Design Update**

Once the design has been modified, run synthesis to generate an updated netlist. This step is crucial because Reveal operates on post-synthesis data, and any changes to the RTL must be reflected in the synthesized output.

▶ **Create a New Reveal Project at the Post-Synthesis Stage**

After synthesis, create a new Reveal project and select the post-synthesis debug stage. This ensures that the tool uses the latest netlist and allows you to configure signal tracing based on the updated design.

▶ **Activate Existing Post-Synthesis Reveal Project (If Applicable)**

If a Reveal project already exists for the post-synthesis stage and matches the updated design, it can be reactivated. This saves time by reusing previous configurations, provided they are still valid for the current netlist.

▶ **Run Synthesis to Generate Reveal Core Only**

After configuring the Reveal project, run synthesis again, but this time it will synthesize only the Reveal analyzer core. This step integrates the debug logic into the design without re-synthesizing the entire user logic, streamlining the process.

Guidelines when Changing the Synthesis Tool

▶ **Disable All Active Reveal Projects (*.rvl)**

Before switching synthesis tools, deactivate any active Reveal projects. This prevents conflicts between the previous synthesis output and the new tool's netlist structure. Reveal projects are tightly coupled with the synthesis tool used, so disabling them ensures a clean transition.

▶ **Synthesize the User Design After Selecting a Different Synthesis Tool**

Once the new synthesis tool is selected, re-run synthesis on the user design. This step generates a fresh netlist compatible with the selected tool and ensures that Reveal can correctly interpret the design structure for instrumentation.

▶ **Create a New Reveal Project at the Post-Synthesis Stage**

After synthesis is complete, create a new Reveal project and select the post-synthesis debug stage. This ensures that the Reveal configuration aligns with the netlist generated by the new synthesis tool, allowing accurate signal tracing and core insertion.

Revision History

The following table provides the revision history for this document.

Date	Version	Description
12/11/2025	2025.2	Updated to reflect changes in Radiant 2025.2 software.
09/26/2025	2025.1.1	Updated to reflect changes in Radiant 2025.1.1 software.
06/26/2025	2025.1	Updated to reflect changes in Radiant 2025.1 software.
03/31/2025	2024.2.1	Updated to reflect changes in Radiant 2024.2.1 software.
12/20/2024	2024.2	Updated to reflect changes in Radiant 2024.2 software.
06/28/2024	2024.1	Updated to reflect changes in Radiant 2024.1 software.
03/29/2024	2023.2.1	Updated to reflect changes in Radiant 2023.2.1 software.
11/27/2023	2023.2	Updated to reflect changes in Radiant 2023.2 software.
06/5/2023	2023.1	Updated to reflect changes in Radiant 2023.1 software.
02/27/2023	2022.1.1	Updated to reflect changes in Radiant 2022.1.1 software.
11/7/2022	2022.1	Updated to reflect changes in Radiant 2022.1 software.
06/6/2022	3.2	Updated to reflect changes in Radiant 3.2 software.
12/7/2021	3.1	Updated to reflect changes in Radiant 3.1 software.
05/1/2020	2.1	Updated to reflect changes in Radiant 2.1 software.
10/23/2019	2.0	Initial Release. Content was previously Appendix A of "Radiant Software User Guide."