UG0773
User Guide
PolarFire FPGA SmartDebug

NOTE: PDF files are intended to be viewed on the printed page; links and cross-references in this PDF file may point to external files and generate an error when clicked. **View the online help included with software to enable all linked content.**

Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: **sales.support@microsemi.com**
www.microsemi.com

About Microsemi

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at **www.microsemi.com.**

5-02-00773-4/08.18

Table of Contents

Table of Contents 2

Welcome to SmartDebug 4
Introduction to SmartDebug 4
Getting Started with SmartDebug 5
Using SmartDebug 5
Create Standalone SmartDebug Project 5

SmartDebug User Interface 8
Standalone SmartDebug User Interface 8
Programming Connectivity and Interface 9
View Device Status 13

Debugging 16
Debug FPGA Array 16
Hierarchical View 16
Netlist View 18
Live Probes 19
Active Probes 19
Probe Grouping (Active Probes Only) 20
Memory Blocks 24
Probe Insertion (Post-Layout) 29
Debug sNVM 32
Debug Transceiver 35
Debug uPROM 54

SmartDebug Tcl Commands 58
SmartDebug Tcl Support 58
add_probe_insertion_point 60
add_to_probe_group 60
construct_chain_automatically 60
create_probe_group 61
delete_active_probe 61
enable_device 62
get_programmer_info 62
load_active_probe_list 62
loopback_mode 63
move_to_probe_group 63
optimize_dfe 63
plot_eye 64
program_probe_insertion 64

read_active_probe 64
read_lsram 65
read_snvm_memory 66
read_usram 67
remove_from_probe_group 68
remove_probe_insertion_point 68
run_selected_actions 68
save_active_probe_list 69
scan_chain_prg 69
select_active_probe 69
set_debug_programmer 70
set_device_name 70
set_live_probe 71
set_programming_action 71
set_programming_file 72
smartbert_test 72
static_pattern_transmit 74
ungroup 75
unset_live_probe 75
uprom_read_memory 76
write_active_probe 76
write_lsram 77
write_usram 78
xcvr_read_register 79
xcvr_write_register 81

Frequently Asked Questions 84
How do I monitor a static or pseudo-static signal? 84
How do I force a signal to a new value? 84
How do I perform simple SmartBERT tests? 85
How do I read LSRAM or USRAM content? 86
How do I change the content of LSRAM or USRAM? 87
How do I read the health check of the Transceiver? 88

Welcome to SmartDebug

Introduction to SmartDebug

Design debug is a critical phase of FPGA design flow. Microsemi's SmartDebug tool complements design simulation by allowing verification and troubleshooting at the hardware level. SmartDebug provides access to non-volatile memory (sNVM), SRAM, transceiver, uPROM, and probe capabilities. Microsemi PolarFire FPGA devices have built-in probe logic that greatly enhance the ability to debug logic elements within the device. SmartDebug accesses the built-in probe points through the Active Probe and Live Probe features, which enables designers to check the state of inputs and outputs in real-time without re-layout of the design.

Use Models

SmartDebug can be run in the following modes:

- Integrated mode from the Libero Design Flow
- Standalone mode
- Demo mode

Integrated Mode

When run in integrated mode from Libero, SmartDebug can access all design and programming hardware information. No extra setup step is required. In addition, the Probe Insertion feature is available in Debug FPGA Array.

To open SmartDebug in the Libero Design Flow window, expand **Debug Design** and double-click **SmartDebug Design**.

Standalone Mode

SmartDebug can be installed separately in the setup containing FlashPro, FlashPro Express, and Job Manager. This provides a lean installation that includes all the programming and debug tools to be installed in a lab environment for debug. In this mode, SmartDebug is launched outside of the Libero Design Flow. When launched in standalone mode, you must to go through SmartDebug project creation and import a Design Debug Data Container (DDC) file, exported from Libero, to access all debug features in the supported devices.

Note: In standalone mode, the Probe Insertion feature is not available in FPGA Array Debug, as it requires incremental routing to connect the user net to the specified I/O.

Demo Mode

Demo mode allows you to experience SmartDebug features (Active Probe, Live Probe, Memory Blocks, Transceiver) without connecting a board to the system running SmartDebug.

Note: SmartDebug demo mode is for demonstration purposes only, and does not provide the functionality of integrated mode or standalone mode.

Note: You cannot switch between demo mode and normal mode while SmartDebug is running.

Standalone Mode Use Model Overview

In the main use model for standalone SmartDebug, the DDC file must be generated from Libero and imported into a SmartDebug project to obtain full access to the device debug features. Alternatively, SmartDebug can be used without a DDC file with a limited feature set.

Supported Families, Programmers, and Operating Systems

Programming and Debug: PolarFire

Programmers: FlashPro3, FlashPro4, and FlashPro5

Operating Systems: Windows 7, Windows 10, RHEL 6.x, RHEL 7.x, Cent OS 6, and Cent OS 7

Getting Started with SmartDebug

This topic introduces the basic elements and features of SmartDebug. If you are already familiar with the user interface, proceed to the Solutions to Common Issues Using SmartDebug or Frequently Asked Questions sections.

SmartDebug enables you to use JTAG to interrogate and view embedded silicon features and device status. SmartDebug is available as a part of the FlashPro programming tool.

See Using SmartDebug for an overview of the use flow.

You can use the debugger to:

- Get device status and view diagnostics

Using SmartDebug

The most common flow for SmartDebug is:

1. Create your design. You must have a FlashPro programmer connected to use SmartDebug.
2. Expand **Debug Design** and double-click **Smart Debug Design** in the Design Flow window. SmartDebug opens for your target device.
3. Click **View Device Status** to view the device status report and check for issues.
4. Examine individual silicon features, such as FPGA debug.

Create Standalone SmartDebug Project

A standalone SmartDebug project can be configured in two ways:

- Import DDC files exported from Libero
- Construct Automatically

From the SmartDebug main window, click **Project** and choose **New Project**. The Create SmartDebug Project dialog box opens.

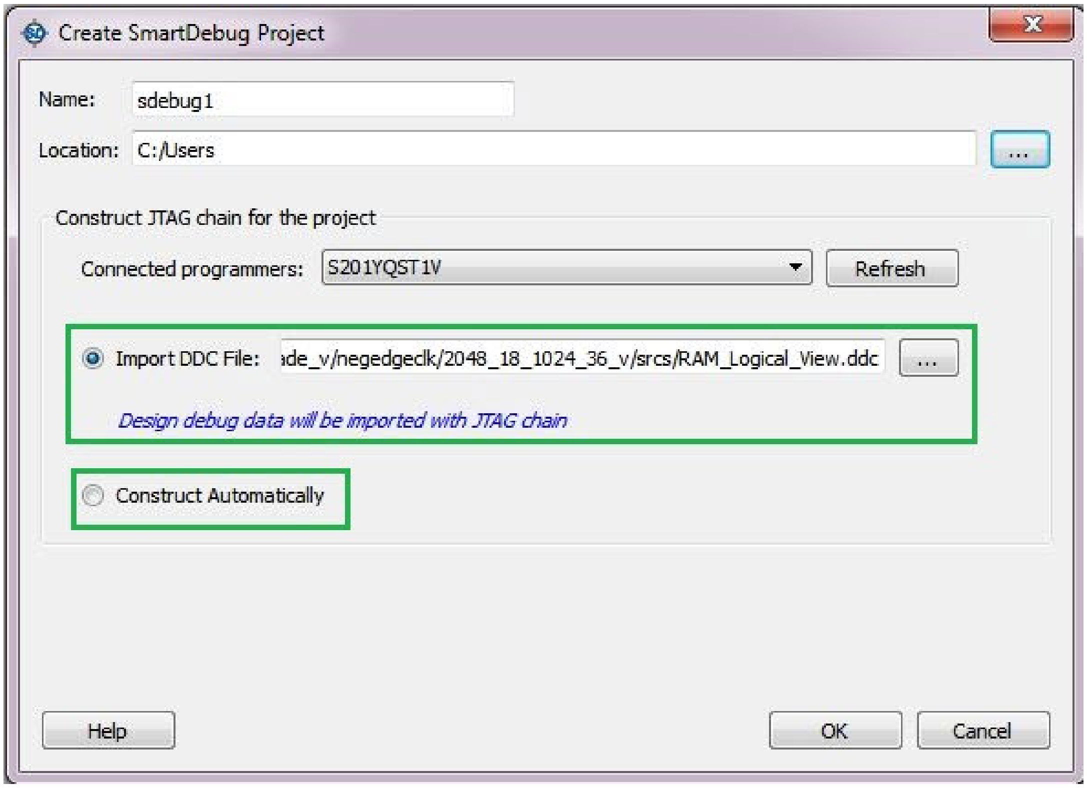


Figure 1 · Create SmartDebug Project Dialog Box

Import from DDC File (created from Libero)

When you select the **Import from DDC File** option in the Create SmartDebug Project dialog box, the Design Debug Data of the target device and all hardware and JTAG chain information present in the DDC file exported in Libero are automatically inherited by the SmartDebug project. The programming file information loaded onto other Microsemi devices in the chain is also transferred to the SmartDebug project.

Debug data is imported from the DDC file (created through Export SmartDebug Data in Libero) into the debug project, and the devices are configured using data from the DDC file.

Construct Automatically

When you select the **Construct Automatically** option, a debug project is created with all the devices connected in the chain for the selected programmer. This is equivalent to Construct Chain Automatically in FlashPRO.

Configuring a Generic Device

For Microsemi devices having the same JTAG IDCODE (i.e., multiple derivatives of the same Die), the device type must be configured for SmartDebug to enable relevant features for debug. The device can be configured by loading the programming file, by manually selecting the device using Configure Device, or by importing DDC files through Programming Connectivity and Interface. When the device is configured, all debug options are shown.

For debug projects created using Construct Automatically, you can use the following options to debug the devices:

- Load the programming file – Right-click the device in Programming Connectivity and Interface.
- Import Debug Data from DDC file – Right-click the device in Programming Connectivity and Interface.

The appropriate debug features of the targeted devices are enabled after the programming file or DDC file is imported.

Connected FlashPRO Programmers

The drop-down lists all FlashPro programmers connected to the device. Select the programmer connected to the chain with the debug device. At least one programmer must be connected to create a standalone SmartDebug project.

Before a debugging session or after a design change, program the device through Programming Connectivity and Interface.

See Also

Programming Connectivity and Interface

View Device Status

SmartDebug User Interface

Standalone SmartDebug User Interface

You can start standalone SmartDebug from the Libero installation folder or from the FlashPRO installation folder.

Windows:

<Libero Installation folder>/Designer/bin/sdebug.exe

<FlashPRO Installation folder>/bin/sdebug.exe

Linux:

<Libero Installation folder>/ bin/sdebug

<FlashPRO Installation folder>/bin/sdebug

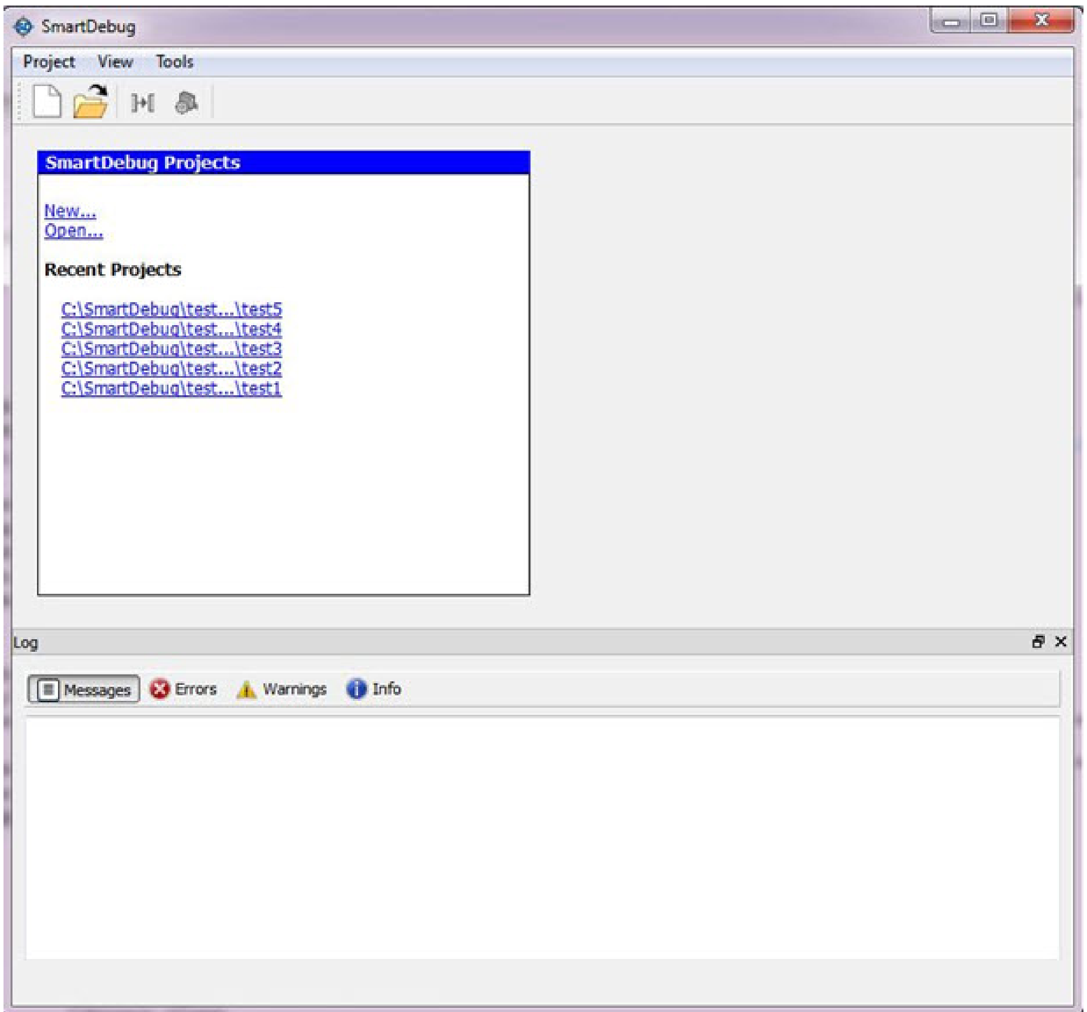


Figure 2 · Standalone SmartDebug Main Window

Project Menu

The Project menu allows you to do the following:

- Create new SmartDebug projects (**Project > New Project**)
- Open existing debug projects (**Project > Open Project**)

- Execute SmartDebug-specific Tcl scripts (**Project > Execute Script**)
- Export SmartDebug-specific commands to a script file (**Project > Export Script File**)
- See a list of recent SmartDebug projects (**Project > Recent Projects**).

Log Window

SmartDebug displays the Log window by default when it is invoked. To suppress the Log window display, click the View menu and toggle **View Log**.

The Log window has four tabs:

Messages – displays standard output messages

Errors – displays error messages

Warnings – displays warning messages

Info – displays general information

Tools Menu

The Tools menu includes Programming Connectivity and Interface and Programmer Settings options, which are enabled after creating or opening a SmartDebug project.

Programming Connectivity and Interface

To open the Programming Connectivity and Interface dialog box, from the standalone SmartDebug Tools menu, choose **Programming Connectivity and Interface**. The Programming Connectivity and Interface dialog box displays the physical chain from TDI to TDO.

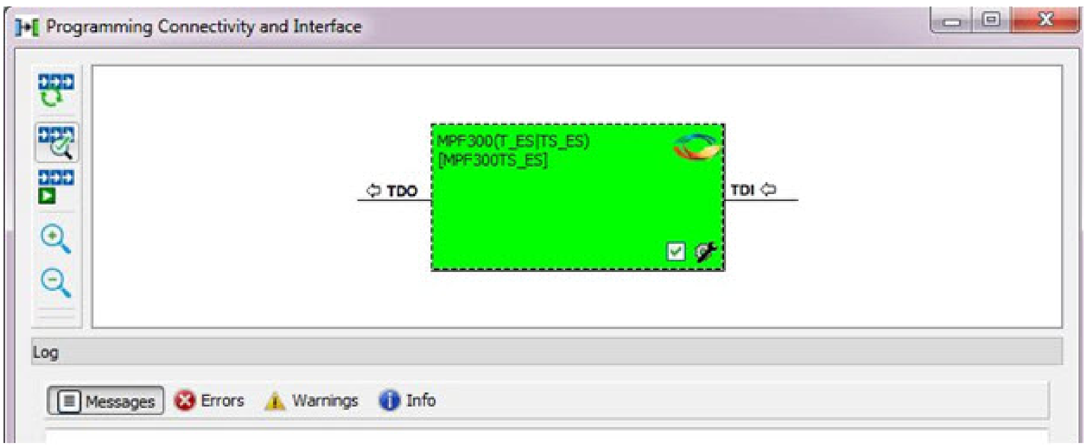


Figure 3 · Programming Connectivity and Interface Dialog Box – Project created using Import from DDC File

All devices in the chain are disabled by default when a standalone SmartDebug project is created using the **Construct Automatically** option in the Create SmartDebug Project dialog box.

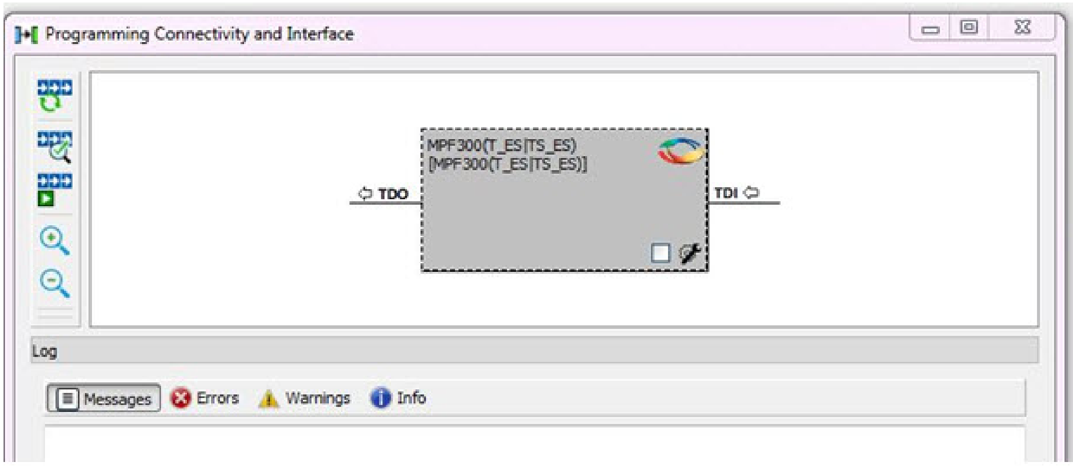


Figure 4 · Programming Connectivity and Interface window – Project created using Construct Automatically

The Programming Connectivity and Interface dialog box includes the following actions:

- **Construct Chain Automatically** - Automatically construct the physical chain.

  Running Construct Chain Automatically in the Programming Connectivity and Interface removes all existing debug/programming data included using DDC/programming files. The project is the same as a new project created using the Construct Chain Automatically option.

- **Scan and Check Chain** – Scan the physical chain connected to the programmer and check if it matches the chain constructed in the scan chain block diagram.
- **Run Programming Action** – Option to program the device with the selected programming procedure.

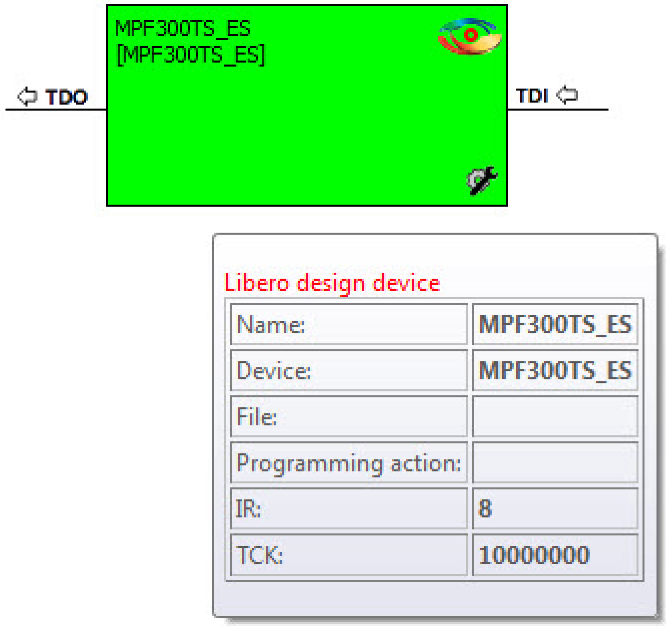
  When two devices are connected in the chain, the programming actions are independent of the device.

- **Zoom In** – Zoom into the scan chain block diagram.
- **Zoom Out** – Zoom out of the scan chain block diagram.

Hover Information

The device tooltip displays the following information if you hover your cursor over a device in the scan chain block diagram:

- **Name:** User-specified device name. This field indicates the unique name specified by the user in the Device Name field in Configure Device (right-click **Properties**).
- **Device:** Microsemi device name.
- **Programming File:** Programming file name.
- **Programming action:** The programming action selected for the device in the chain when a programming file is loaded.
- **IR:** Device instruction length.
- **TCK:** Maximum clock frequency in MHz to program a specific device; standalone SmartDebug uses this information to ensure that the programmer operates at a frequency lower than the slowest device in the chain.



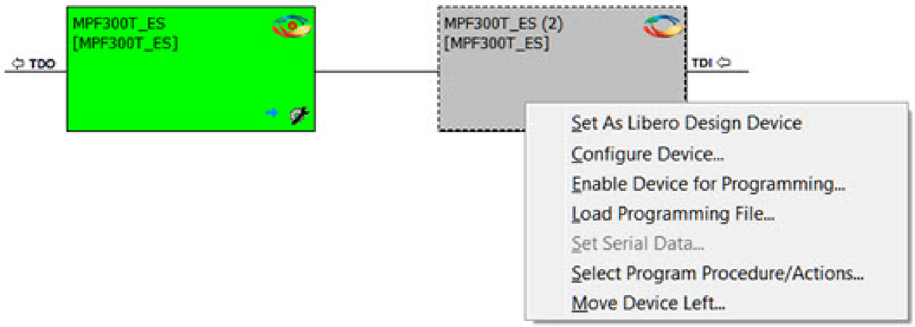
Device Chain Details

The device within the chain has the following details:

- User-specified device name
- Device name
- Programming file name
- Programming action – Select **Enable Device for Programming** to enable the device for programming. Enabled devices are green, and disabled devices are grayed out.

Right-click Properties

The following options are available when you right-click a device in the Programming Connectivity and Interface dialog box.



Set as Libero Design Device - The user needs to set Libero design device when there are multiple identical Libero design devices in the chain.

Configure Device - Ability to reconfigure the device.

- **Family and Die**: The device can be explicitly configured from the Family, Die drop-down.
- **Device Name**: Editable field for providing user-specified name for the device.

Enable Device for Programming - Select to enable the device for programming. Enabled devices are shown in green, and disabled devices are grayed out.

Load Programming File - Load the programming file for the selected device.

Select Programming Procedure/Actions- Option to select programming action/procedures for the devices connected in the chain.

- **Actions**: List of programming actions for your device.
- **Procedures**: Advanced option; enables you to customize the list of recommended and optional procedures for the selected action.

Import Debug Data from DDC File - Option to import debug data information from the DDC file.

Note: This option is supported when SmartDebug is invoked in standalone mode.

The DDC file selected for import into device must be created for a compatible device. When the DDC file is imported successfully, all current device debug data is removed and replaced with debug data from the imported DDC file.

The JTAG Chain configuration from the imported DDC file is ignored in this option.

If a programming file is already loaded into the device prior to importing debug data from the DDC file, the programming file content is replaced with the content of the DDC file (if programming file information is included in the DDC file).

Debug Context Save

Debug context refers to the user selections in debug options such as Debug FPGA Array, Debug Transceiver, and View Flash Memory Content. In standalone SmartDebug, the debug context of the current session is saved or reset depending on the user actions in Programming Connectivity and Interface.

The debug context of the current session is retained for the following actions in Programming Connectivity and Interface:

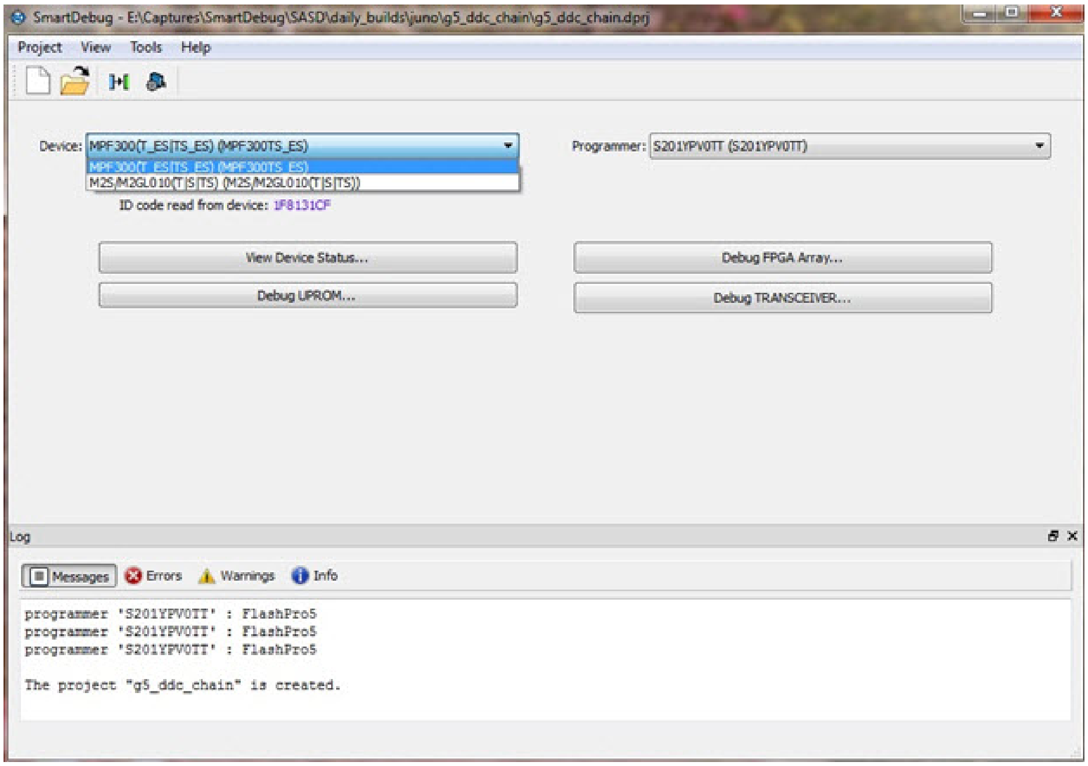
- Enable Device for Programming
- Select Programming Procedure/Actions
- Scan and Check Chain
- Run Programming Action

The debug context of the current session is reset for the following actions in Programming Connectivity and Interface:

- Auto Construct – Clears all the existing debug data. You need to reimport the debug data from DDC file.
- Import Debug Data from DDC file
- Configure Device – Renaming the device in the chain
- Configure Device – Family/Die change
- Load Programming File

Selecting Devices for Debug

Standalone SmartDebug provides an option to select the devices connected in the JTAG chain for debug. The device debug context is not saved when another debug device is selected.



View Device Status

Click **View Device Status** in the standalone SmartDebug main window to display the Device Status Report. The Device Status Report is a complete summary of IDCode, device certificate, design information, programming information, digest, and device security information. Use this dialog box to save or print your information for future reference.

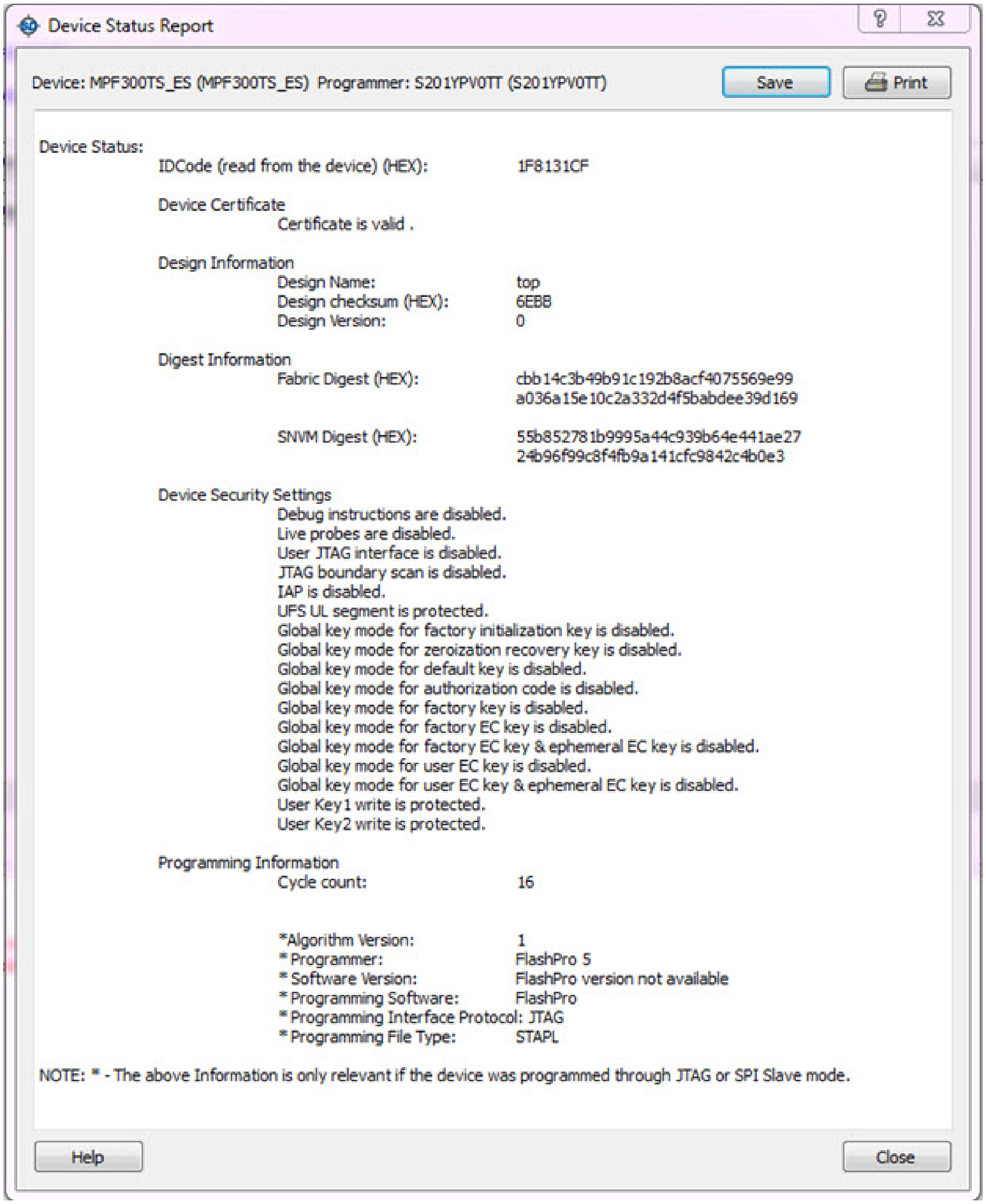


Figure 5 · Device Status Report

IdCode

IDCode read from the device under debug.

Device Certificate

Device certificate displays Family and Die information if device certificate is installed on the device.

If the device certificate is not installed on the device, a message indicating that the device certificate may not have been installed is shown.

Design Information

Design Information displays the following:

- Design Name
- Design Checksum
- Design Version

Digest Information

Digest Information displays Fabric Digest, sNVM Digest (if applicable) computed from the device during programming. sNVM Digest is shown when sNVM is used in the design.

Device Security Settings

Device Security Settings displays information about your security settings, including live probes, JTAG boundary scan, global key modes, and user keys.

Programming Information

Programming Information displays the following:

- Cycle Count
- Algorithm Version
- Programmer
- Software Version
- Programming Software
- Programming Interface Protocol
- Programming File Type

Debugging

Debug FPGA Array

In the Debug FPGA Array dialog box, you can view your Live Probes, Active Probes, Memory Blocks, and Insert Probes (Probe Insertion).

The Debug FPGA Array dialog box includes the following four tabs:

- Live Probes
- Active Probes
- Memory Blocks
- Probe Insertion

Hierarchical View

The Hierarchical View lets you view the instance level hierarchy of the design programmed on the device and select the signals to add to the Live Probes, Active Probes, and Probe Insertion tabs in the Debug FPGA Array dialog box. Logical and physical Memory Blocks can also be selected.

- **Instance** – Displays the probe points available at the instance level.
- **Primitives** – Displays the lowest level of probeable points in the hierarchy for the corresponding component —i.e., leaf cells (hard macros on the device).

You can expand the hierarchy tree to see lower level logic.

Signals with the same name are grouped automatically into a bus that is presented at instance level in the instance tree.

The probe points can be added by selecting any instance or the leaf level instance in the Hierarchical View. Adding an instance adds all the probe able points available in the instance to Live Probes, Active Probes, and Probe Insertion.

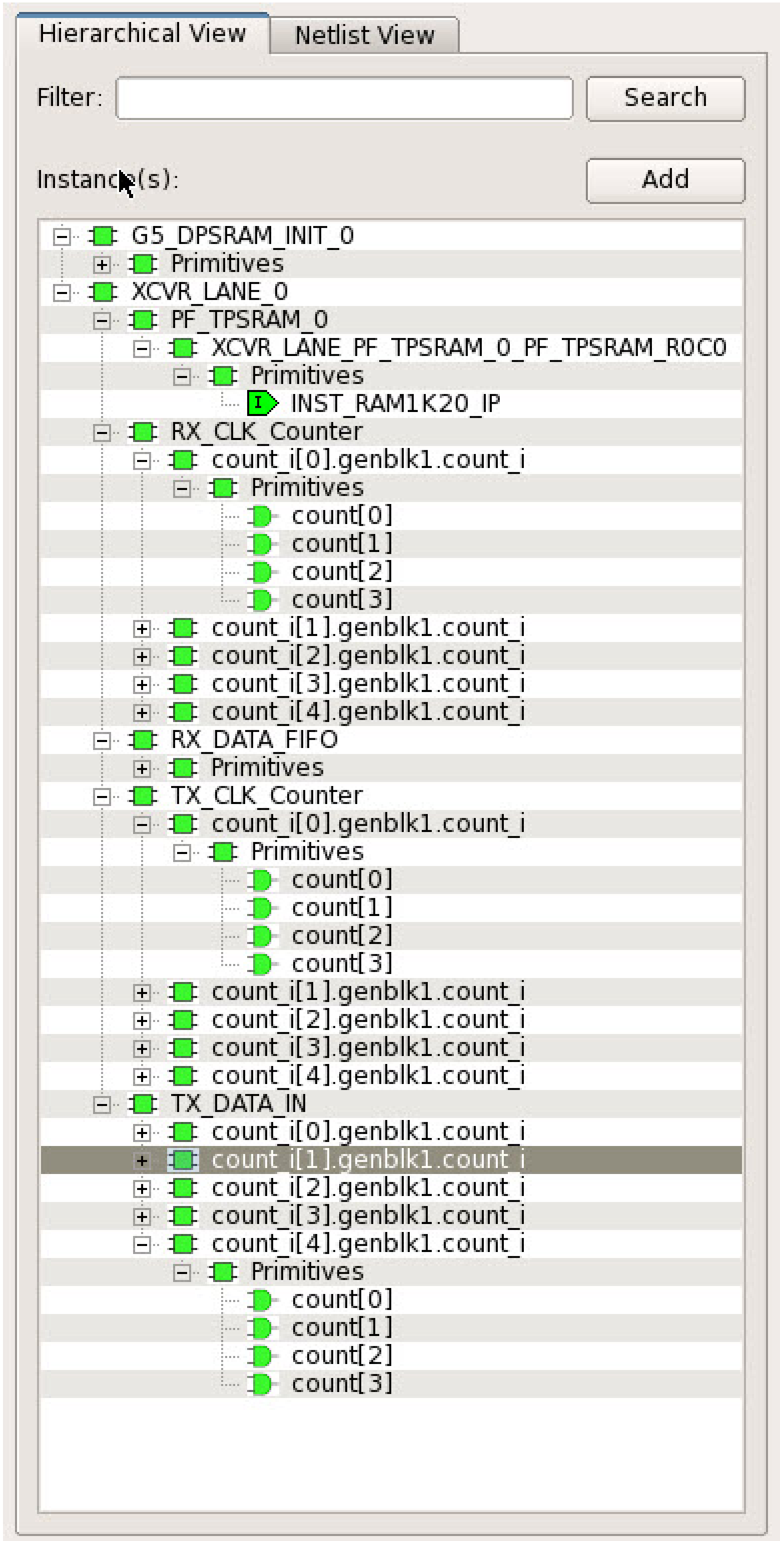


Figure 6 · Hierarchical View

Search

In Live Probes, Active Probes, Memory Blocks, and the Probe Insertion UI, a search option is available in the Hierarchical View. You can use wildcard characters such as * or ? in the search column for wildcard matching.

Probe points of leaf level instances resulting from a search pattern can only be added to Live Probes, Active Probes, and the Probe Insertion UI. You cannot add instances of search results in the Hierarchical View.

Netlist View

The Netlist View displays a flattened net view of all the probe-able points present in the design, along with the associated cell type.

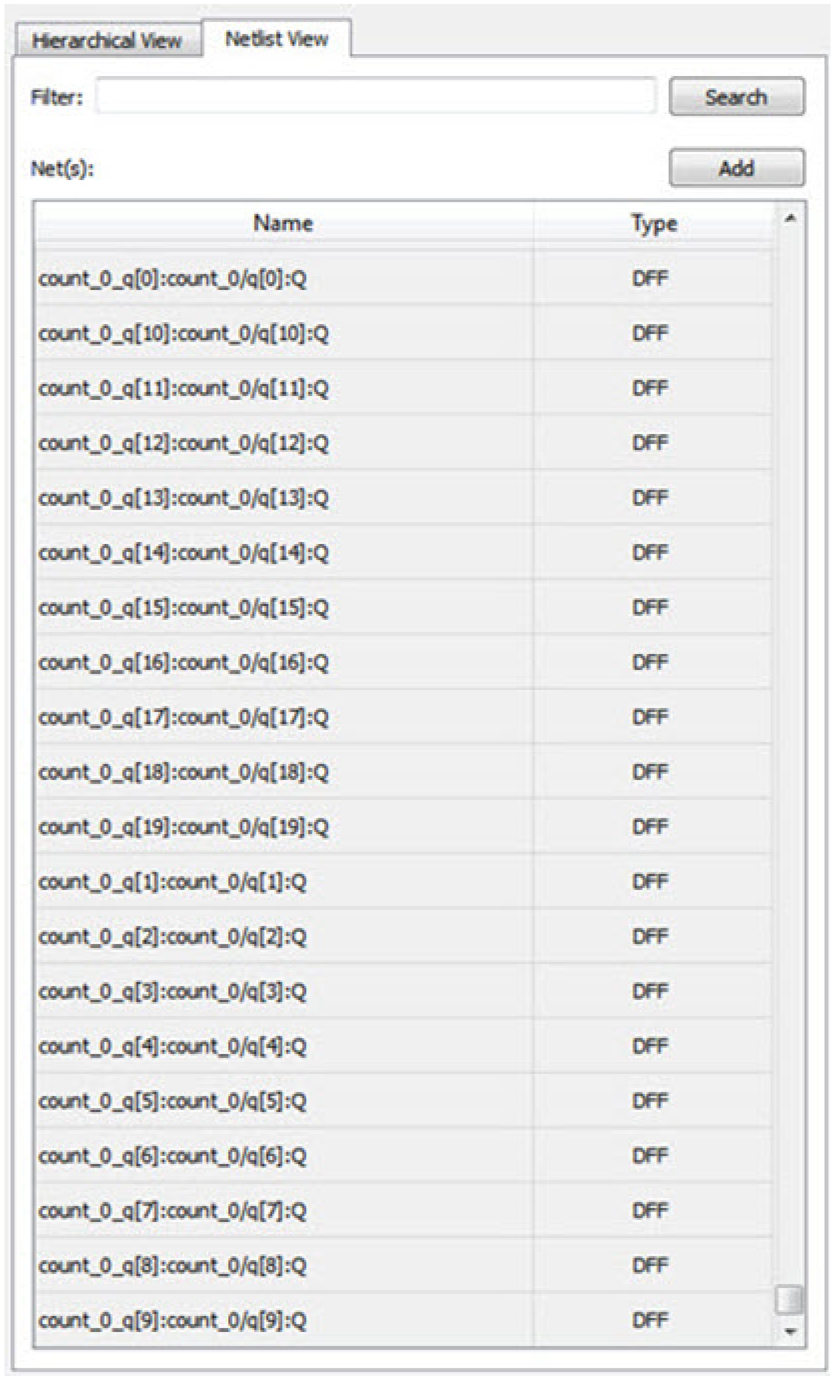


Figure 7 · Netlist View

Search

A search option is available in the Netlist View for Live Probes, Active Probes, and Probe Insertion. You can use wildcard characters such as * or ? in the search column for wildcard matching.

Live Probes

Live Probes is a design debug option that uses non-intrusive real time scoping of up to two probe points with no design changes.

The Live Probes tab in the Debug FPGA Array dialog box displays a table with the probe names and pin types.

There are two channels, and Live Probe can be assigned/unassigned independently.

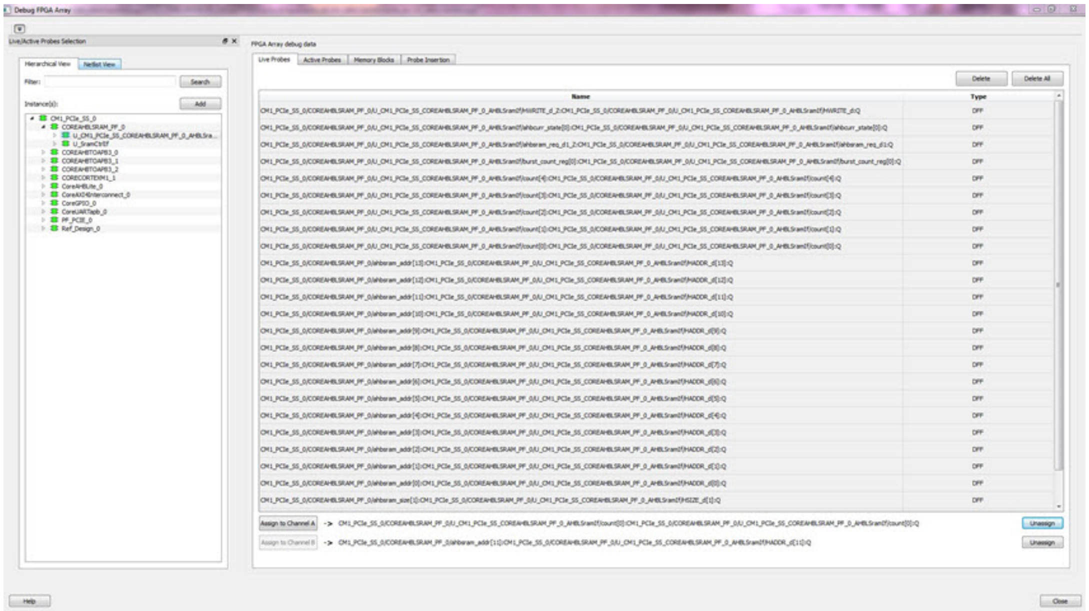


Figure 8 · Live Probes Tab in SmartDebug FPGA Array Dialog Box

Two probe channels (ChannelA and ChannelB) are available. When a probe name is selected, it can be assigned to either ChannelA or ChannelB.

You can assign a probe to a channel by doing either of the following:

- Right-click a probe in the table and choose **Assign to Channel A** or **Assign to Channel B**.
- Click the **Assign to Channel A** or **Assign to Channel B** button to assign the probe selected in the table to the channel. The buttons are located below the table.

When the assignment is complete, the probe name appears to the right of the button for that channel, and SmartDebug configures the ChannelA and ChannelB I/Os to monitor the desired probe points. Because there are only two channels, a maximum of two internal signals can be probed simultaneously.

Click the **Unassign Channels** button to clear the live probe names to the right of the channel buttons and discontinue the live probe function during debug.

Note: Both probes can be assigned/unassigned independently.

Live Probes in Demo Mode

You can assign and unassign Live Probes ChannelA and ChannelB in Demo Mode.

Active Probes

Active Probes is a design debug option to read and write to one or many probe points in the design through JTAG.

In the left pane of the Active Probes tab, all available Probe Points are listed in instance level hierarchy in the Hierarchical View. All Probe Names are listed with the Name and Type (which is the physical location of the flip-flop) in the Netlist View.

Select probe points from the Hierarchical View or Netlist View, right-click and choose **Add** to add them to the Active Probes UI. You can also add the selected probe points by clicking the **Add** button. The probes list can be filtered with the Filter box.

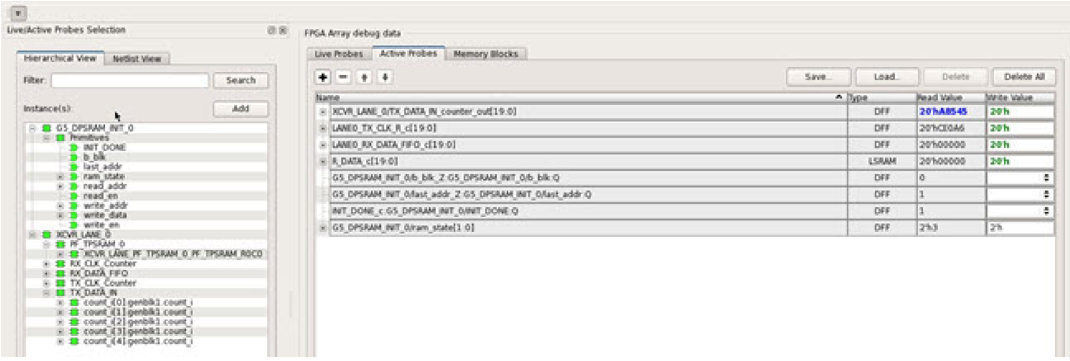


Figure 9 · Active Probes Tab in SmartDebug FPGA Array Dialog Box

When you have selected the desired probe, points appear in the Active Probe Data chart and you can read and write multiple probes (as shown in the figure below).

You can use the following options in the Write Value column to modify the probe signal added to the UI:

- Drop-down menu with values ‘0’ and ‘1’ for individual probe signals
- Editable field to enter data in hex or binary for a probe group or a bus

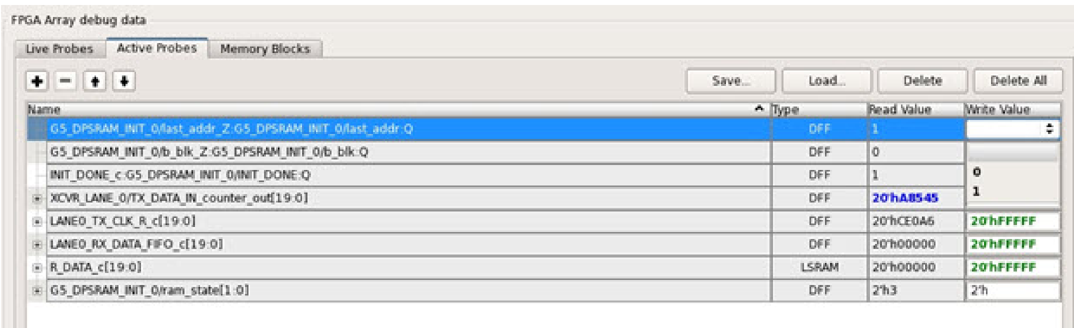


Figure 10 · Active Probes Tab - Write Value Column Options

Active Probes in Demo Mode

In demo mode, a temporary probe data file with details of current and previous values of probes added in the active probes tab is created in the designer folder. The write values of probes are updated to this file, and the GUI is updated with values from this file when you click Write Active Probes. Data is read from this file when you click Read Active Probes. If there is no existing data for a probe in the file, the read value displays all 0s. The value is updated based on your changes.

Probe Grouping (Active Probes Only)

During the debug cycle of the design, designers often want to examine the different signals. In large designs, there can be many signals to manage. The Probe Grouping feature assists in comprehending multiple signals as a single entity. This feature is applicable to Active Probes only. Probe nets with the same name are automatically grouped in a bus when they are added to the Active Probes tab. Custom probe groups can also be created by manually selecting probe nets of a different name and adding them into the group.

The Active Probes tab provides the following options for probe points that are added from the Hierarchical View/Netlist View:

- Display bus name. An automatically generated bus name cannot be modified. Only custom bus names can be modified.
- Expand/collapse bus or probe group
- Move Up/Down the signal, bus, or probe group
- Save (Active Probes list)
- Load (already saved Active Probes list)
- Delete (applicable to a single probe point added to the Active Probes tab
- Delete All (deletes all probe points added to the Active Probes tab)
- In addition, the context (right-click) menu provides the following operations:
    - Create Group, Add/Move signals to Group, Remove signals from Group,
    - Ungroup
    - Reverse bit order, Change Radix for a bus or probe group
    - Read, Write, or Delete the signal or bus or probe group

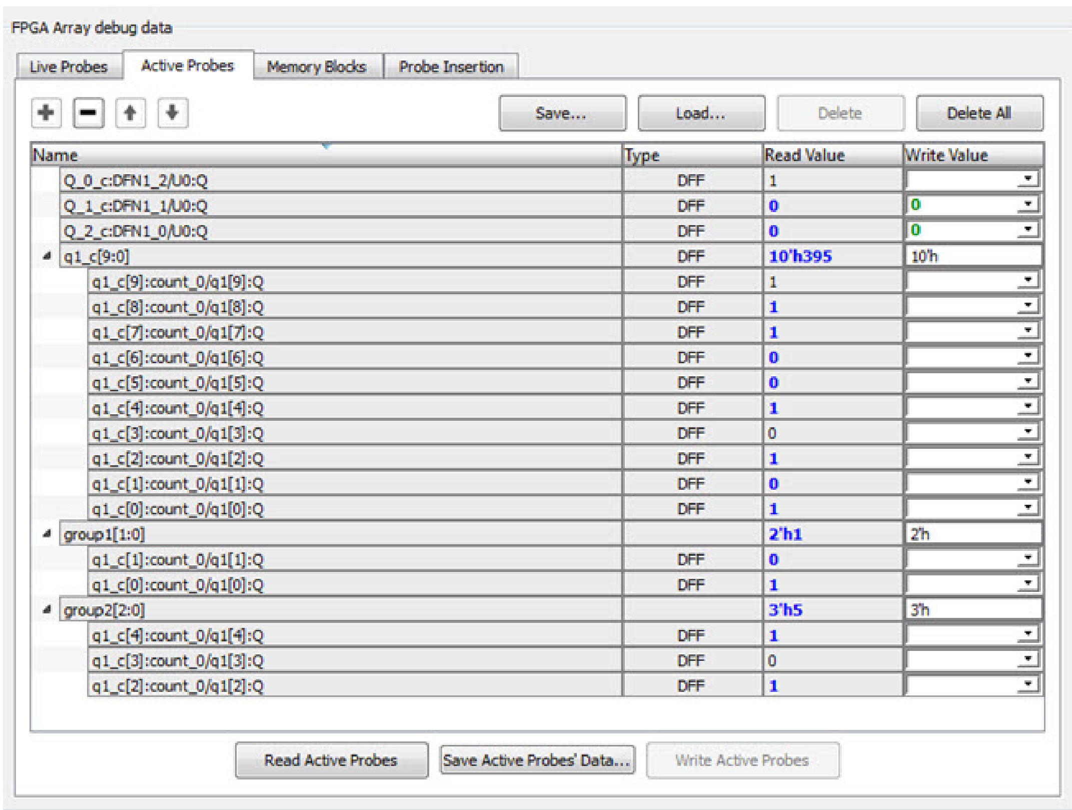


Figure 11 · Active Probes Tab

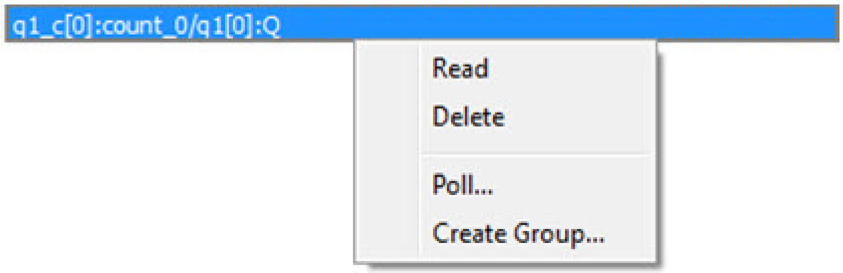
- Green entries in the “Write Value” column indicate that the operation was successful.
- Blue entries in the “Read Value” column indicate values that have changed since the last read.

Context Menu of Probe Points Added to the Active Probes UI

When you right-click a signal or bus, you will see the following menu options:

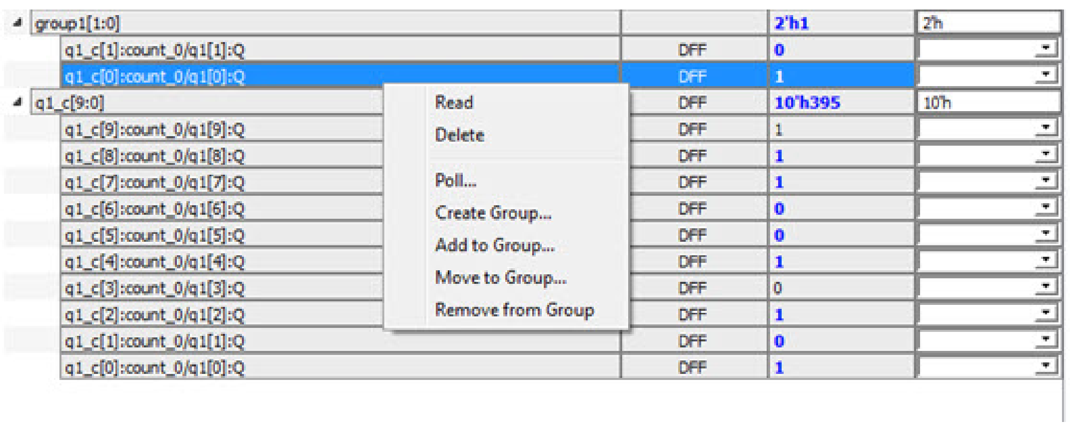
For individual signals that are not part of a probe group or bus:

- Read
- Delete
- Poll
- Create Group



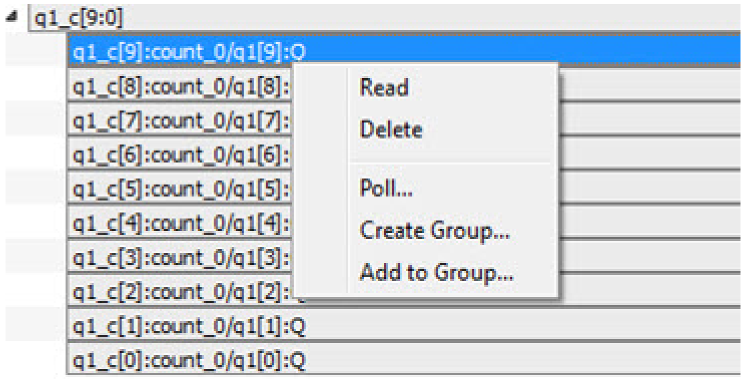
For individual signals in a probe group:

- Read
- Delete
- Poll
- Create Group
- Add to Group
- Move to Group
- Remove from Group



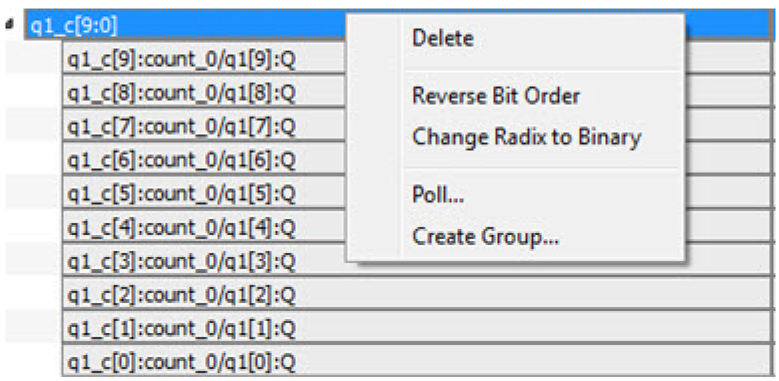
For individual signals in a bus:

- Read
- Delete
- Poll
- Create Group
- Add to Group



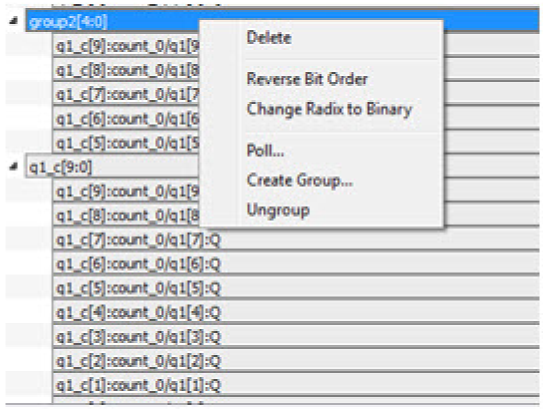
For a bus:

- Delete
- Reverse Bit Order
- Change Radix to Binary
- Poll
- Create Group



For a probe group:

- Delete
- Reverse Bit Order
- Change Radix to Binary
- Poll
- Create Group
- Ungroup



Differences Between a Bus and a Probe Group

A bus is created automatically by grouping selected probe nets with the same name into a bus. A bus *cannot* be ungrouped.

A Probe Group is a custom group created by adding a group of signals in the Active Probes tab into the group. The members of a Probe Group are not associated by their names. A Probe Group *can* be ungrouped.

In addition, certain operations are also restricted to the member of a bus, whereas they are allowed in a probe group.

The following operations are not allowed in a bus:

• **Move to Group**: Moving a signal to a probe group

• **Remove from Group**: Removing a signal from a probe group

Memory Blocks

The Memory Blocks tab in the Debug FPGA Array dialog box shows the hierarchical view of all memory blocks in the design. The depth and width of blocks shown in the logical view are determined by the user in SmartDesign, RTL, or IP cores using memory blocks.

Notes:

- RAM is not accessible to the user when SmartDebug is accessing RAM blocks.
- RAM is not accessible to the user during a read or write operation.
    - o During a single location write, the RAM block is not accessible. If multiple locations are written, the RAM block is accessed and released for each write.
    - o When each write is completed, access returns to the user, so the access time is a single write operation time.

The example figure that follows shows the hierarchical view of the Memory Blocks tab. You can view logical blocks and physical blocks. Logical blocks are shown with an **L** (), and physical blocks are shown with a **P** ( ).

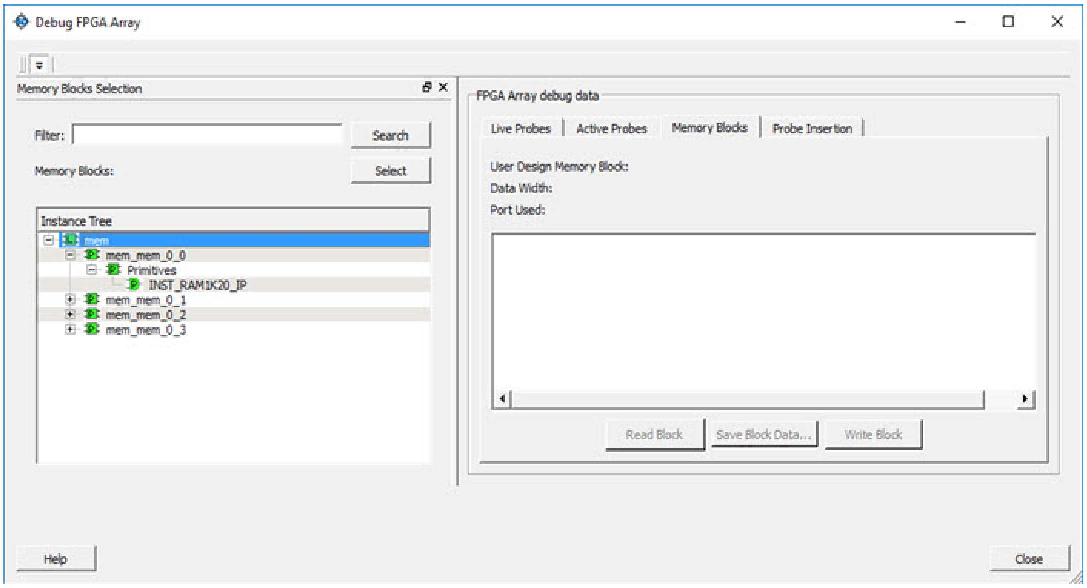


Figure 12 · Memory Blocks Tab - Hierarchical View

You can only select one block at a time. You can select and add blocks in the following ways:

- Right-click the name of a memory block and click **Add** as shown in the following figure.

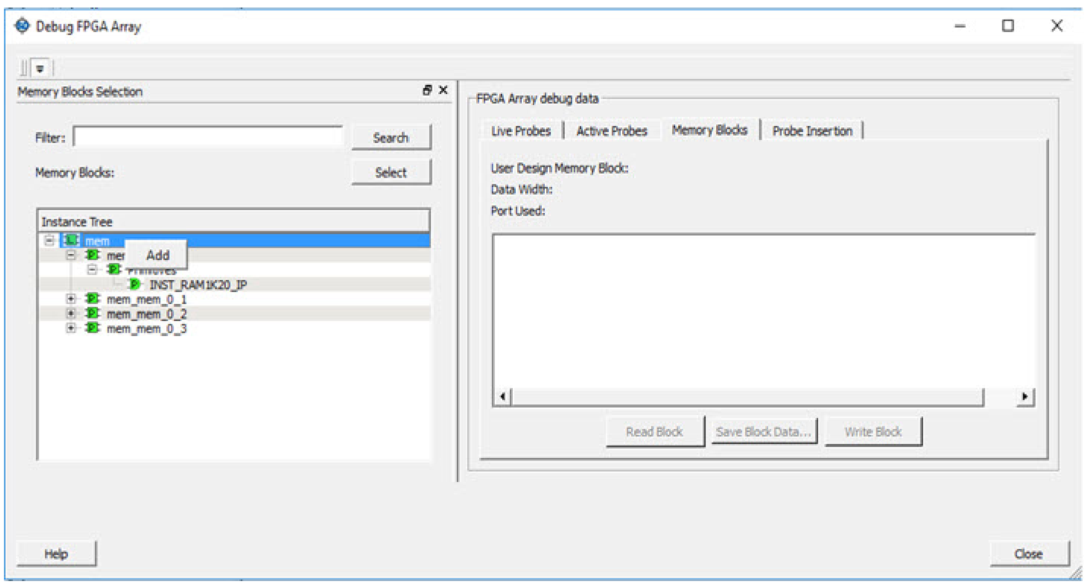


Figure 13 · Adding a Memory Block

- Click on a name in the list and then click **Select**.
- Select a name, drag it to the right, and drop it into the Memory Blocks tab.
- Enter a memory block name in the Filter box and click **Search** or press **Enter**. Wildcard search is supported.

Note: Only memory blocks with an **L** or **P** icon can be selected in the hierarchical view.

Memory Block Fields

The following memory block fields appear in the Memory Blocks tab.

User Design Memory Block

The selected block name appears on the right side. If the block selected is logical, the name from top of the block is shown.

Data Width

If a block is logical, the depth and width is retrieved from each physical block, consolidated, and displayed. If the block is physical, the value of "Depth X Width" is 64 X 12 for uSRAM blocks, 16384 X 1, 8192 X 2, 4096 X 5, 2048 X 10, and 1024 X 20 for LSRAM blocks.

Note: LSRAM physical block configuration of 40-bits is not supported in SmartDebug. This will be available in a future release.

Port Used

This field is displayed only in the logical block view. Because configurators can have asymmetric ports, memory location can have different widths. The port shown can either be Port A or Port B. For TPSRAM, where both ports are used for reading, Port A is used. This field is hidden for physical blocks, as the values shown will be irrespective of read ports.

The following figure shows the Memory Blocks tab fields for a logical block view.

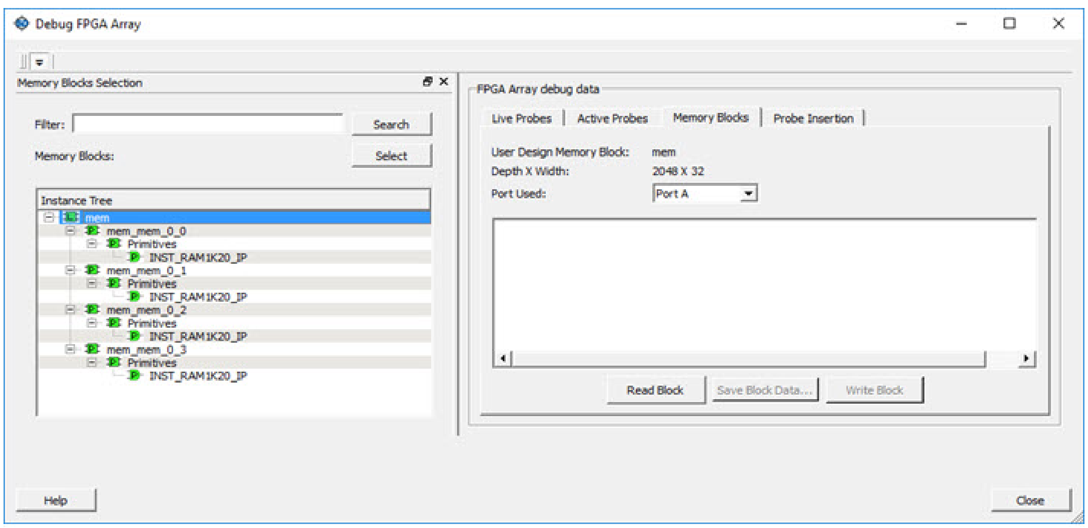


Figure 14 · Memory Blocks Tab Fields for Logical Block View

The following figure shows the Memory Blocks tab fields for a physical block view.

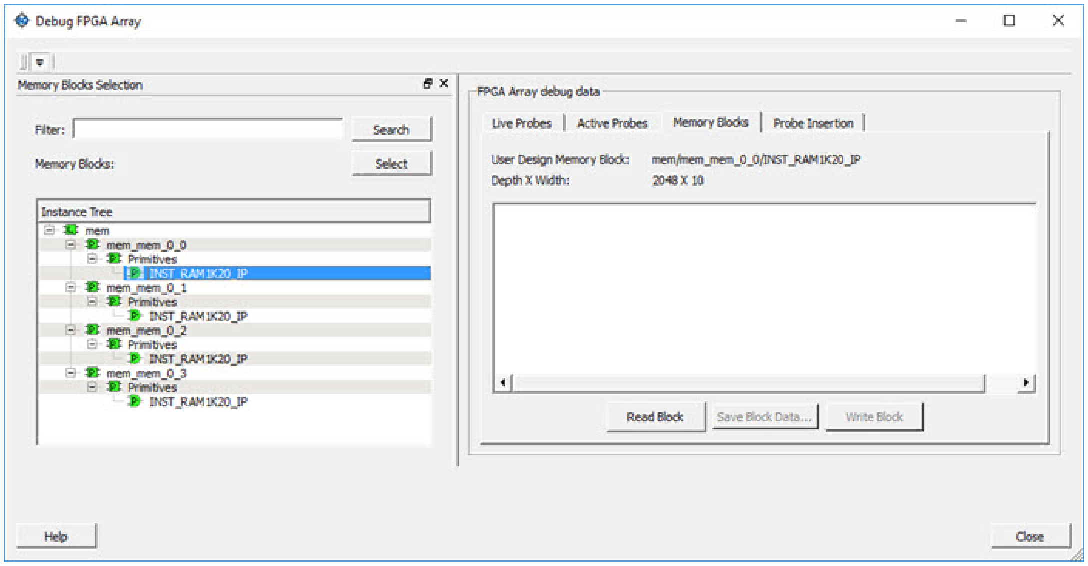


Figure 15 · Memory Blocks Tab Fields for Physical Block View

Read Block

Memory blocks can be read once they are selected. If the block name appears on the right-hand side, the Read Block button is enabled. Click **Read Block** to read the memory block.

Logical Block Read

A logical block shows three fields. User Design Memory Block and Depth X Width are read only fields, and the Port Used field has options. If the design uses both ports, Port A and Port B are shown under options. If only one port is used, only that port is shown.

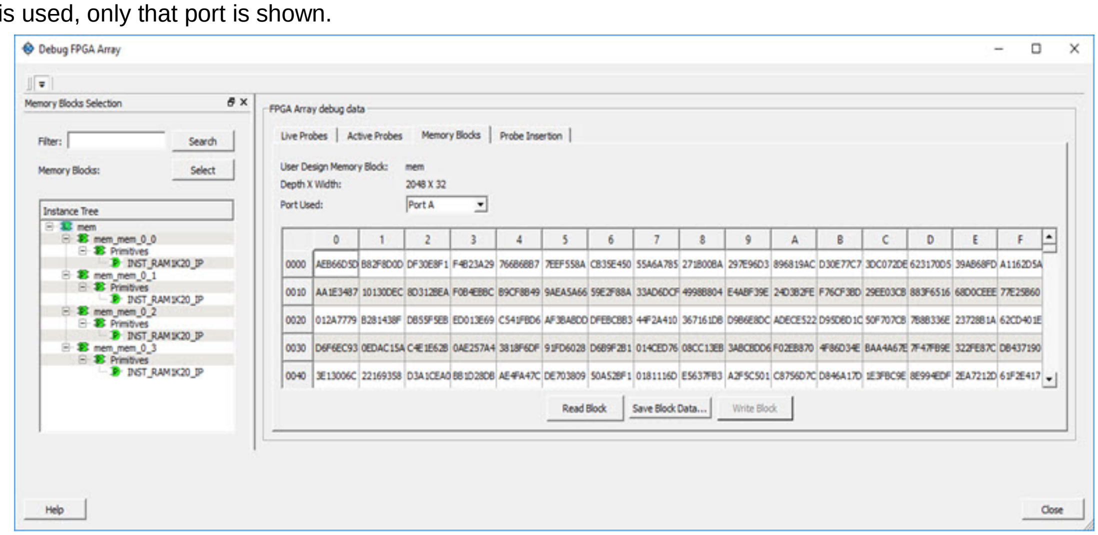


Figure 16 · Logical Block Read

The data shown is in Hexadecimal format. In the example figure above, data width is 32. Because each hexadecimal character has 4 bits of information, you can see 8 characters corresponding to 32 bits. Each row has 16 locations (shown in the column headers) which are numbered in hexadecimal from 0 to F.

Note: For all logical blocks that cannot be inferred from physical blocks, the corresponding icon does not contain a letter.

Physical Block Read

When a Physical block is selected, only the User Design Memory Block and Depth X Width fields are shown.

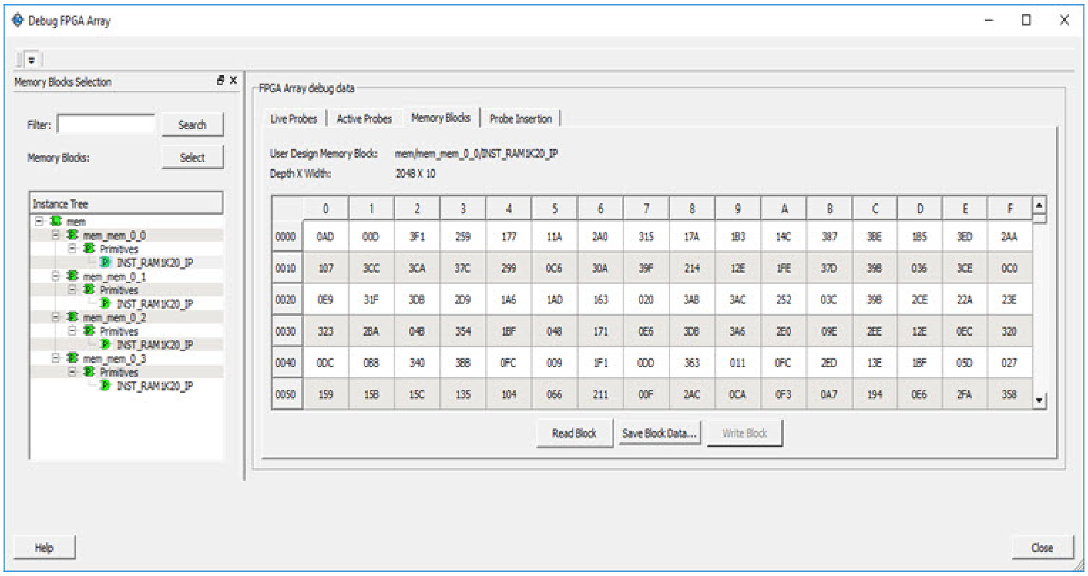


Figure 17 · Physical Block Read

Write Block

Logical Block Write

A memory block write can be done on each location individually. A logical block shows each location of width. The written format is hexadecimal numbers from 0 to F. Width is shown in bits, and values are shown in hexadecimal format. If an entered value exceeds the maximum value, SmartDebug displays a pop-up message showing the range of allowed values.

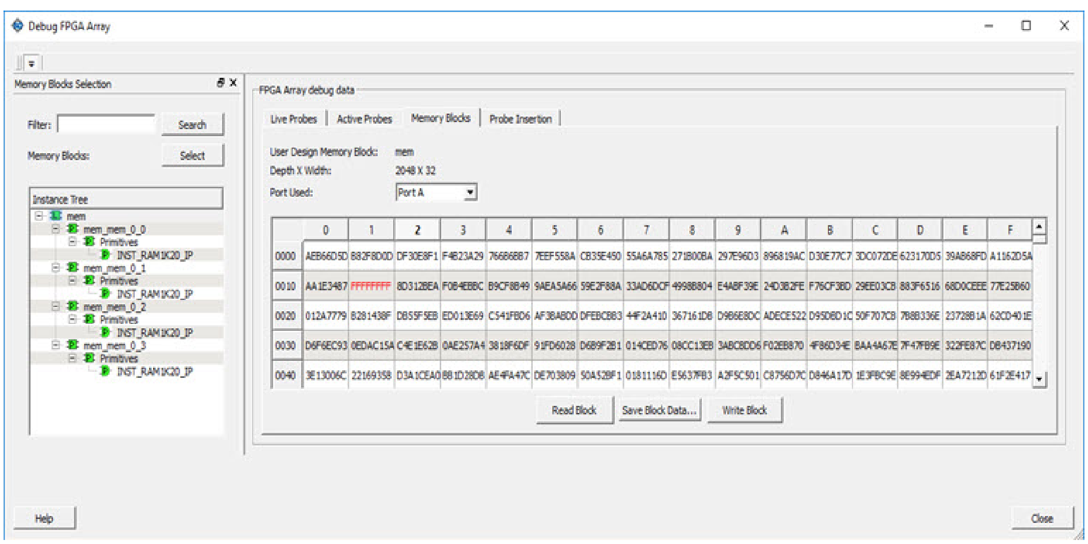


Figure 18 · Logical Block Write

Physical Block Write

Physical blocks have a fixed width of 129 bits for uSRAM and the maximum value that can be written in hexadecimal format is FFF. Similarly, for LSRAM blocks, a range of values are possible (1, 2, 5, 10, and 20) and the maximum values can be 1, 3, 1F, 3FF, and FFFFF, respectively. If an entered value exceeds the limit, SmartDebug displays a popup message showing the range of values that can be entered.

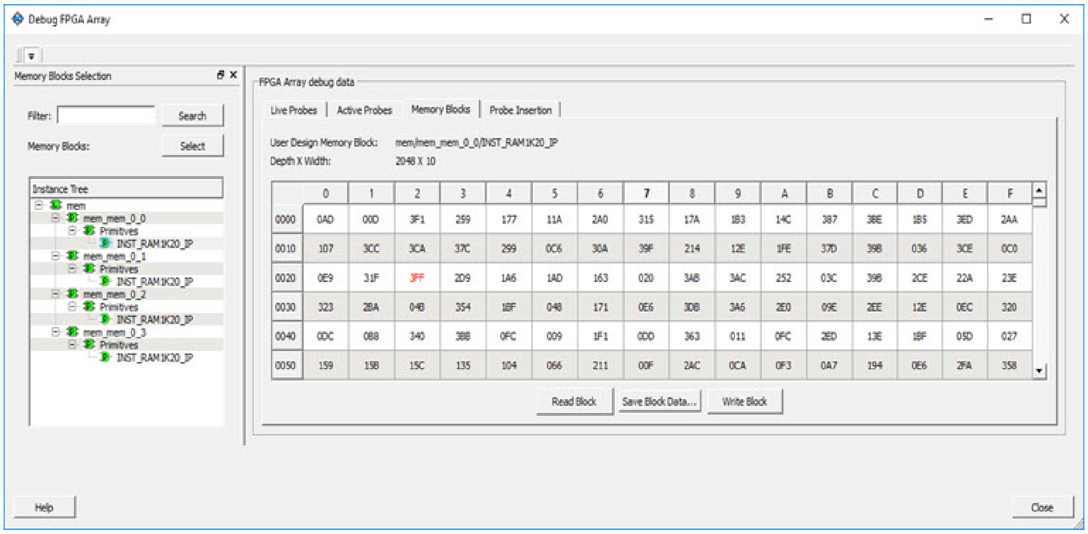


Figure 19 · Physical Block Write

Unsupported Memory Blocks

If RTL is used to configure memory blocks, it is recommended that you follow RAM block inference guidelines provided by Microsemi.

SmartDebug may or may not be able to support logical view for memory blocks that are inferred using RTL coding not specified in the above document.

Memory Blocks in Demo Mode

A temporary memory data file is created in the designer folder for each type of RAM selected. All memory data of all instances of USRAM, LSRAM, and other RAM types is written to their respective data files. The default value of all memory locations is shown as 0s, and is updated based on your changes.

Both physical block view and logical block view are supported.

Probe Insertion (Post-Layout)

Introduction

Probe insertion is a post-layout debug process that enables internal nets in the FPGA design to be routed to unused I/Os. Nets are selected and assigned to probes using the Probe Insertion window in SmartDebug. The rerouted design can then be programmed into the FPGA, where an external logic analyzer or oscilloscope can be used to view the activity of the probed signal.

Note: This feature is not available in standalone mode because of the need to run incremental routing.

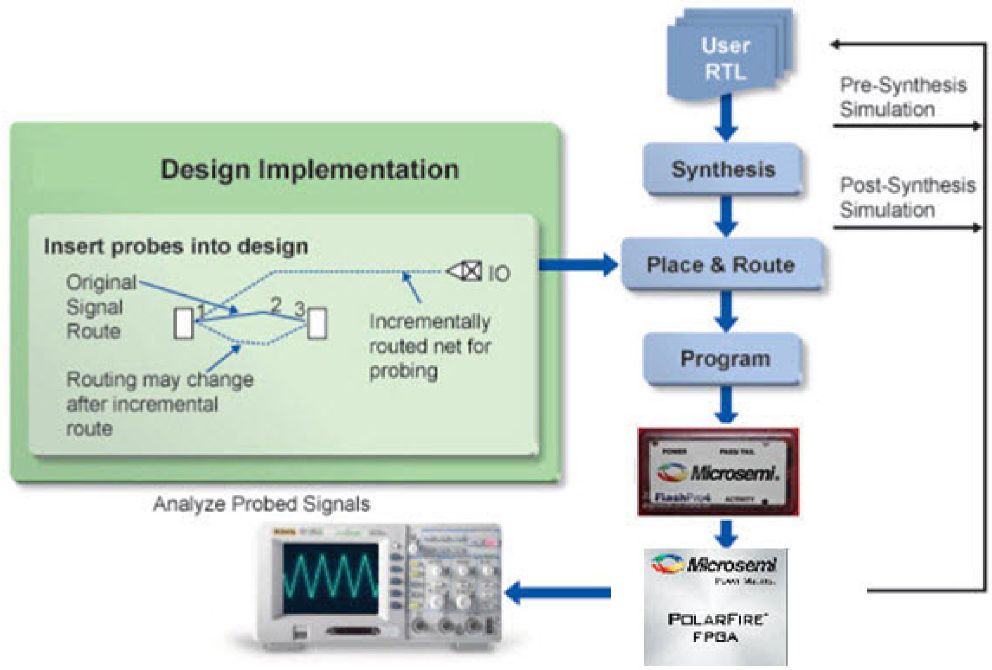


Figure 20 · Probe Insertion in the Design Process

The Probe Insertion debug feature is complementary to Live Probes and Active Probes. Live Probes and Active Probes use a special dedicated probe circuitry.

Probe Insertion

1. Double-click **SmartDebug Design** in the Design Flow window to open the SmartDebug main window.

   Note: FlashPro Programmer must be connected for SmartDebug.

2. Select **Debug FPGA Array** and then select the Probe Insertion tab.

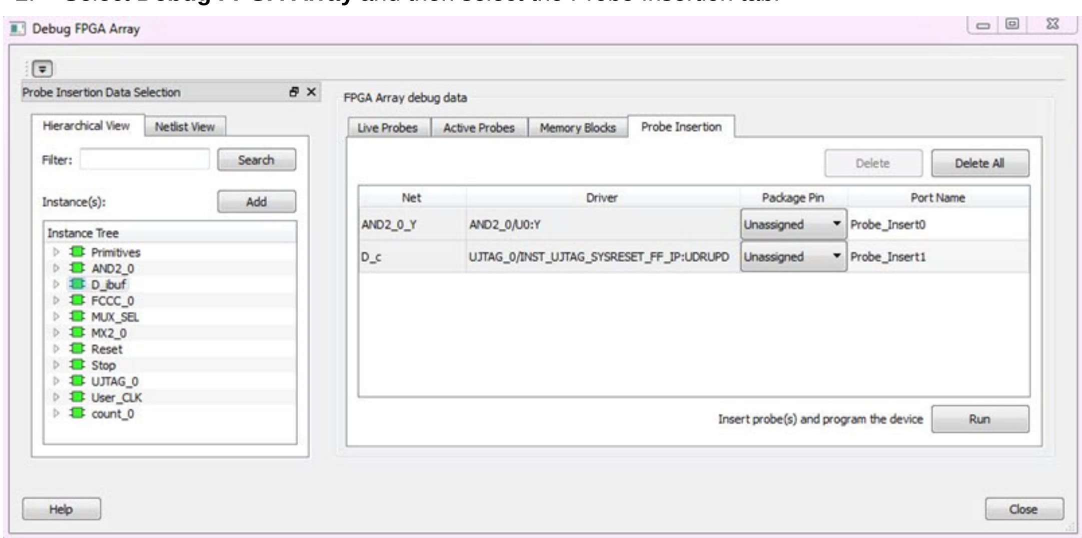


Figure 21 · Probe Insertion Tab

In the left pane of the Probe Insertion tab, all available Probe Points are listed in instance level hierarchy in the Hierarchical View. All Probe Names are shown with the Name and Type in the Netlist View.

3. Select probe points from the Hierarchical View or Netlist View, right-click and choose **Add** to add them to the Active Probes UI. You can also add the selected probe points by clicking the **Add** button. The probes list can be filtered with the Filter box.

Each entry has a Net and Driver name which identifies that probe point.
The selected net(s) appear in the Probes table in the Probe Insertion tab, as shown in the figure below. SmartDebug automatically generates the Port Name for the probe. You can change the Port Name from the default if desired.

4. Assign a package pin to the probe using the drop-down list in the Package Pin column. You can assign the probe to any unused package pin (spare I/O).

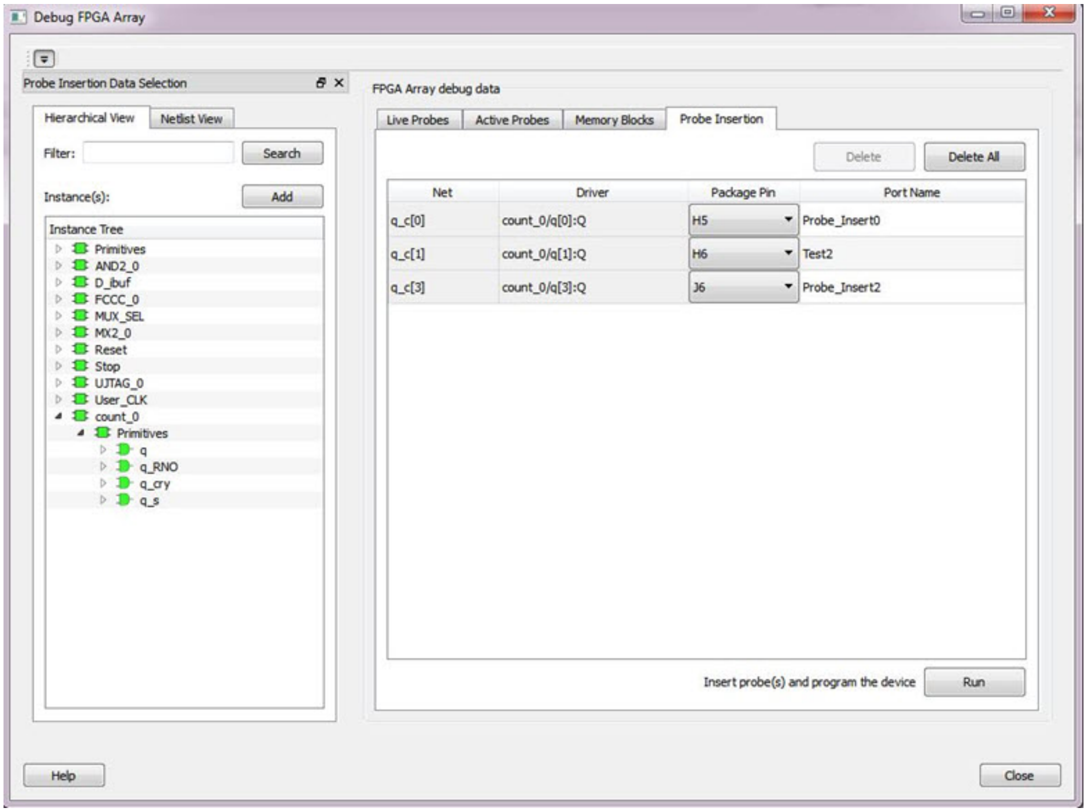


Figure 22 · Debug FPGA Array > Probe Insertion > Add Probe

5. Click **Run**.

   This triggers Place and Route in incremental mode, and the selected probe nets are routed to the selected package pin. After incremental Place and Route, Libero automatically reprograms the device with the added probes.
   The log window shows the status of the Probe Insertion run.

Probe Deletion

To delete a probe, select the probe and click **Delete**. To delete all probes, click **Delete All**.

Note: Deleting probes from the probes list without clicking **Run** does not automatically remove the probes from the design.

Reverting to the Original Design

To revert to the original design after you have finished debugging:

1. In SmartDebug, click **Delete All** to delete all probes.
2. Click **Run**.
3. Wait until the action has completed by monitoring the activity indicator (spinning blue circle). Action is completed when the activity indicator disappears.
4. Close SmartDebug.

Debug sNVM

The sNVM block stores User data and UIC data. This data is stored as clients and can be configured in the Libero design. The USK (User Secret Key) security key secures pages within the memory. Authenticated data can be plain text or encrypted text, and non-authenticated data is plain text. SmartDebug helps the user read the page content of the sNVM block.

The sNVM Debug window has two tabs – Client View and Page View.

Client View

When you open the sNVM window, two tabs are visible. Client information appears in the Client View tab when it is configured in the Libero design. Select a client to expand the table and see pages and page status inside the client. Click the **Read From Device** button to view the memory content.

You can select only one client at a time. Pages inside the client cannot be selected.

Start Page, End Page, and Number of Bytes are displayed for the selected client.

Click the **View All Page Status** button to see information for all pages in the client. See the following example figures.

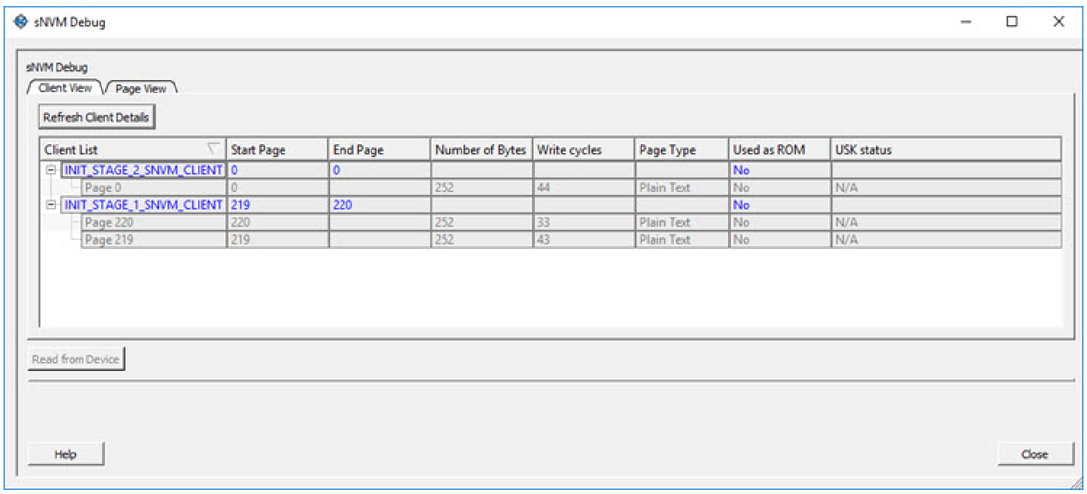


Figure 23 · Client View - expanded list

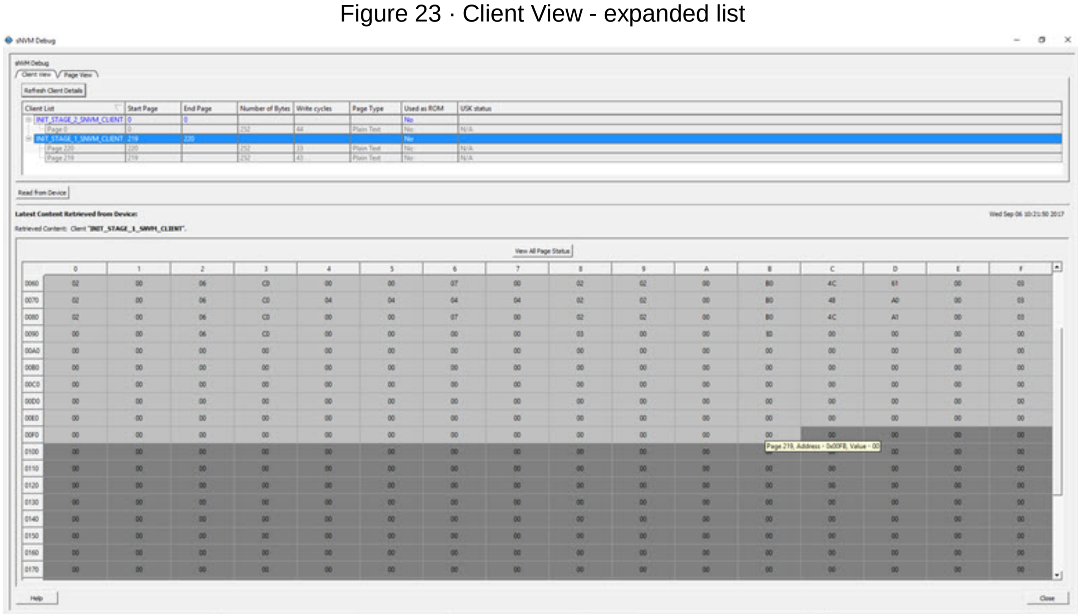


Figure 24 · Client View - Memory

Page View

Page View is used to read a range of pages where start and end page have been specified.

If a page is secured, the default USK is used by SmartDebug to get the page status. If successful, the USK automatically reads the page. If a different USK has been set using system services, use the option to enter the USK, as shown in the example figure below.

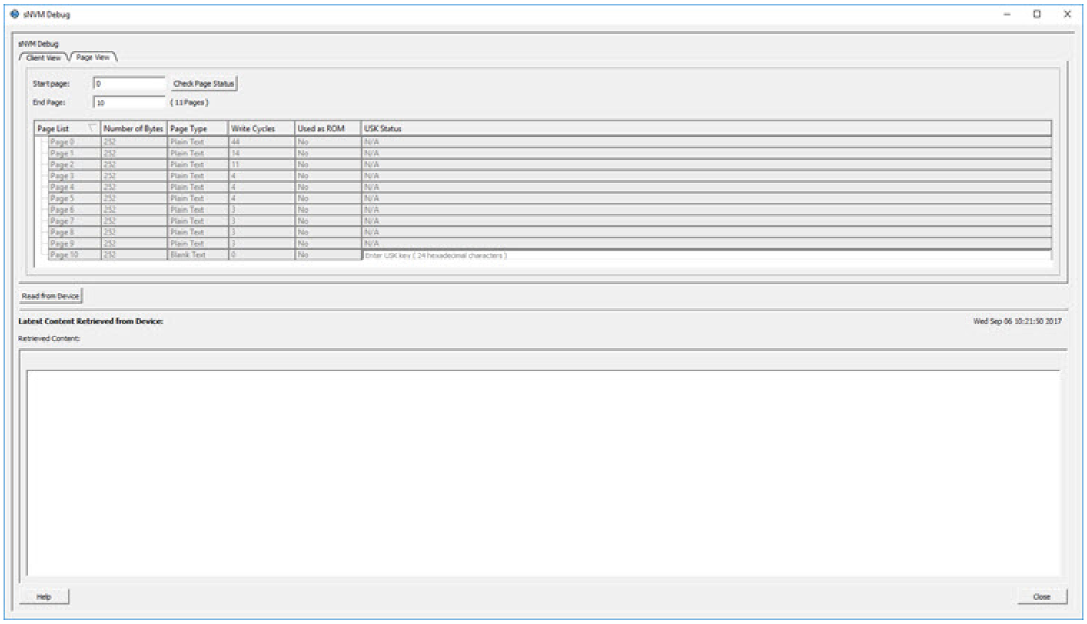


Figure 25 · Page View - Enter USK highlighted

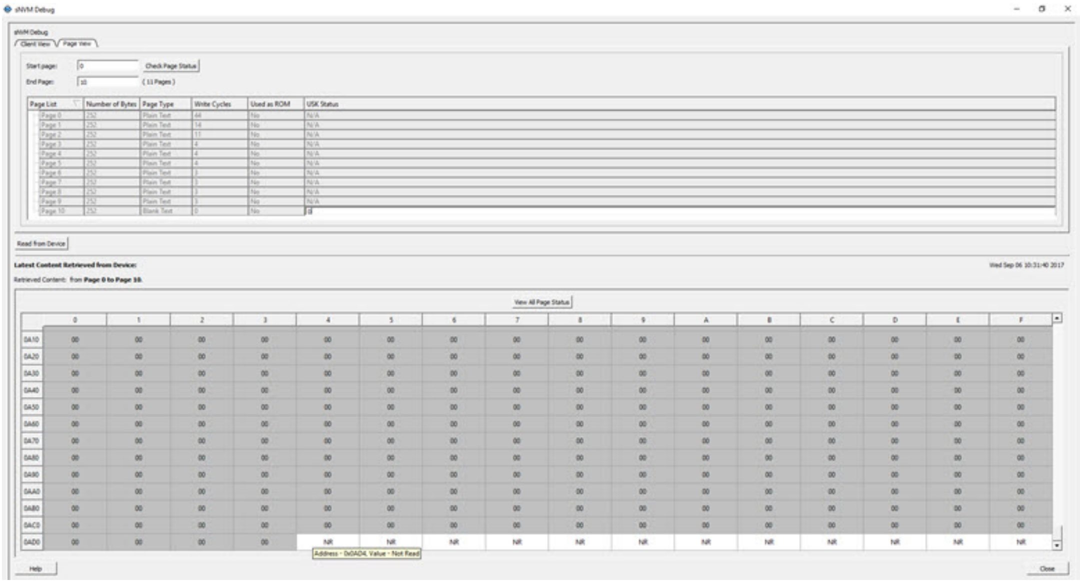


Figure 26 · Memory View - Page Range

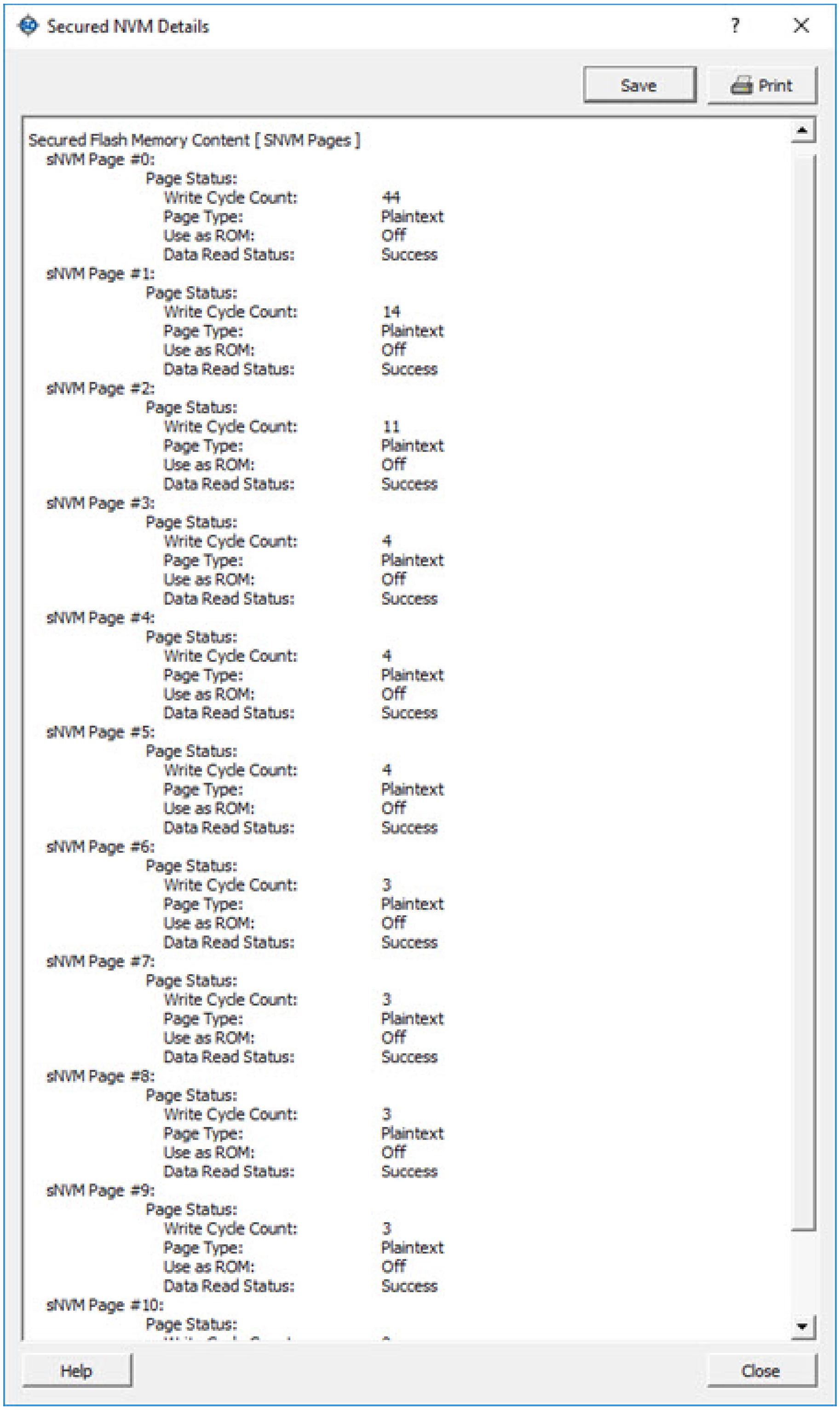


Figure 27 · View All Page Status

Read Operation

Client View

The Client View displays all the clients that are configured in the design. When a client is expanded, a table listing all pages is displayed.

When a client is selected, the Read from Device button is enabled. Click **Read from Device** to read the content of the client. A client can have one or more pages. Refresh Client Details option is given to the user to refresh the table. Click **Refresh Client Details** to update the information in SmartDebug and refresh the table. This is helpful when a client configuration is changed using system services.

Page View

When valid parameters are entered and **Check Page Status** is clicked, a table of all pages is shown with page status information. Pages in the table are read-only and cannot be selected. The page range included in Start Page and End Page is validated, and the Read from Device button is enabled. Click **Read from Device** to read the content.

Runtime Operations

After a design is programmed into the device, you can do the following:

- Change the content of a page
- Authenticate a page
- Change the security key of each configured page

The above operations are not possible if the page is used as ROM.

You can refresh page status in SmartDebug:

- Click the **Refresh Client Details** button in the Client View tab to refresh the client view table and update it with the latest changes.
- Click the **Check Page Status** button in the Page View tab to refresh the pages in the table.

If the security key has been changed, SmartDebug prompts you to enter the USK manually.

Enter the USK in the USK Status column (Client View tab and Page View tab).

By default, the USK entered in the configurator as the USK client is used to authenticate the page.

Debug Transceiver

Debug Transceiver

The Debug Transceiver feature in SmartDebug checks the lane functionality and health for different settings of the lane parameters.

To access the Debug Transceiver feature in SmartDebug, click **Debug Transceiver** in the main SmartDebug window.

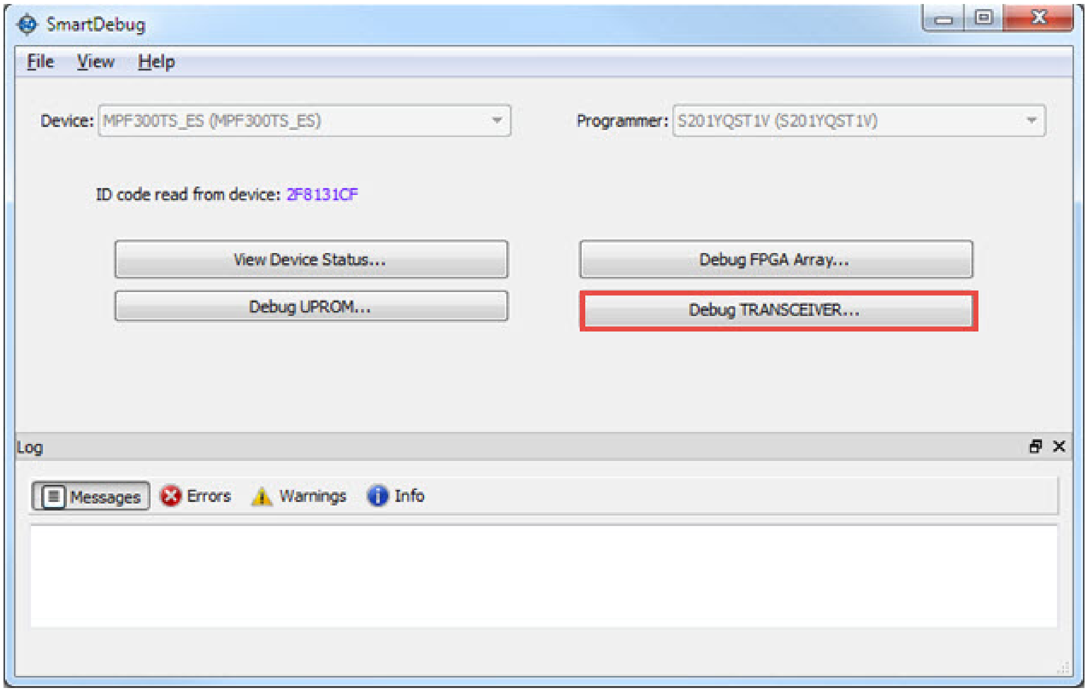


Figure 28 · SmartDebug Window - Debug TRANSCEIVER

This opens the Debug TRANSCEIVER dialog box, which is shown in the following example.

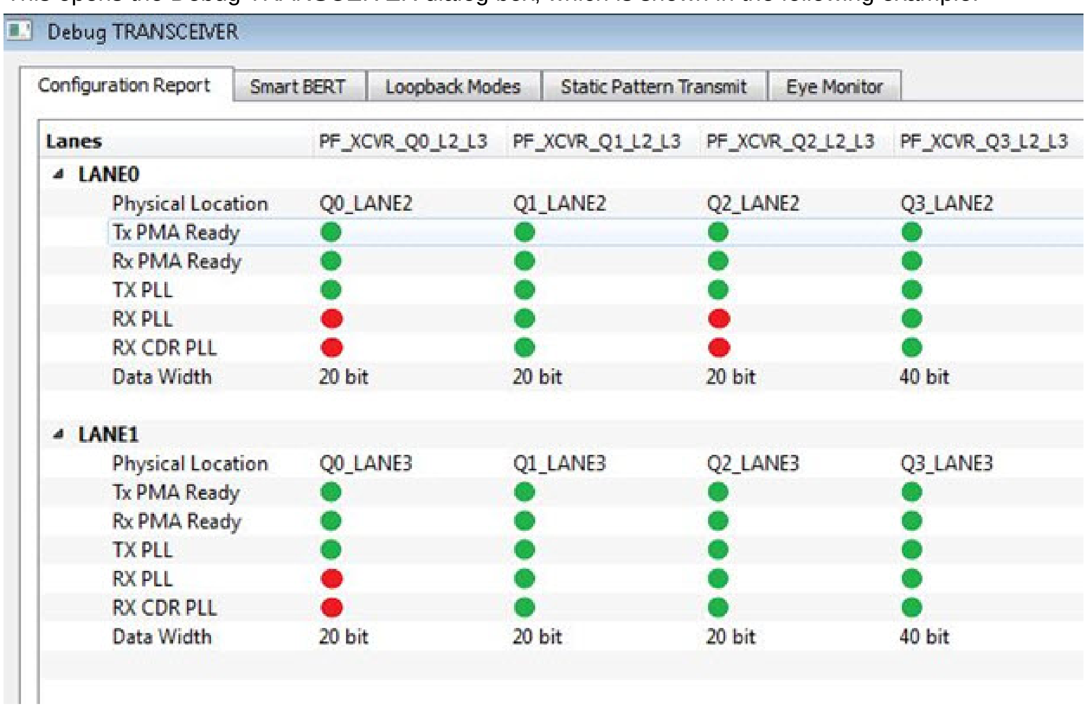


Figure 29 · Debug Transceiver Dialog Box

Debug Transceiver has five distinct debug features, which are represented by tabs in the Debug TRANSCEIVER dialog box:

- Configuration Report (shown by default when the dialog box opens)
- SmartBERT
- Loopback Modes

- [Static Pattern Transmit]
- [Eye Monitor]

Configuration Report

The Configuration Report is the first tab in the Debug TRANSCEIVER dialog box, and is shown by default when the dialog box opens. The Configuration Report shows the lane status/health properties of all lanes of Quads in the design.

Click the **Refresh** button to refresh the information.

Note: The report refreshes automatically when you navigate from another tab.

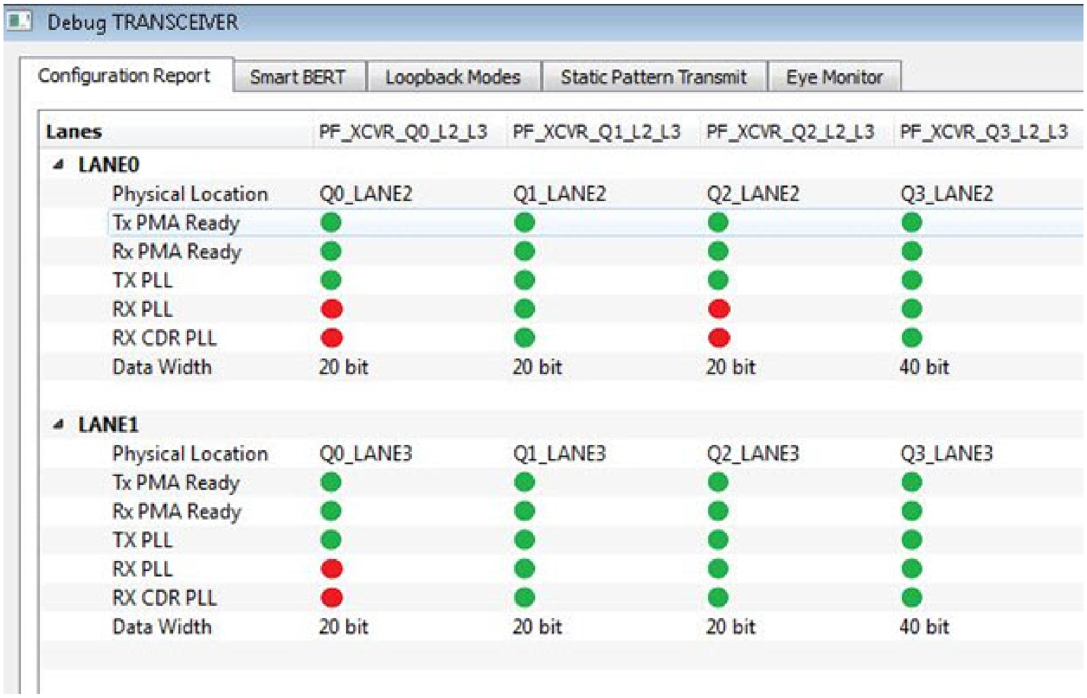


Figure 30 · Debug TRANSCEIVER - Configuration Report

The Configuration Report shows the physical location, status/health, and data width for all lanes of all the quads enabled in the system controller.

Parameter information is shown in table format, with lane numbers as rows and transceiver instance names as columns.

The lane parameters are as follows:

Physical Location - Physical block and lane location in the system controller.

Tx PMA Ready - Indicates if the Tx of the lane is powered up and ready for transactions.

Rx PMA Ready - Indicates if the Rx of the lane is powered up and ready for transactions.

TX PLL - Indicates if the lane is locked onto TX PLL.

RX PLL - Indicates if the lane is locked onto RX PLL.

RX CDR PLL - Indicates if the lane is locked onto the incoming data..

For the parameters above, green indicates true and red indicates false.

Notes:

Click the **Refresh** button to update the lane status.

The report refreshes automatically when you navigate from another tab.

Transceiver Hierarchy

Transceiver Hierarchy is a lane hierarchy with all the lanes instantiated in the design shown with respect to top level instance.

Transceiver Hierarchy is shown in the following tabs: "SmartBERT" on page 39, "Loopback Modes" on page 42, "Static Pattern Transmit" on page 43, and "Eye Monitor" on page 44.

In the SmartBERT, Loopback Modes, and Static Pattern Transmit pages, check boxes allow multiple lanes to be selected for debug, as shown in the following example.

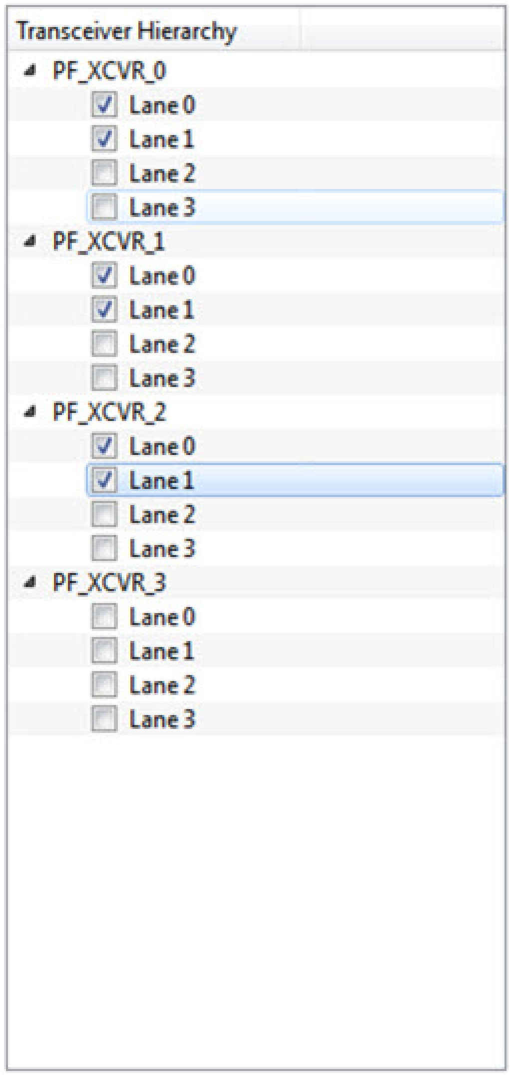


Figure 31 · Transceiver Hierarchy Lane Selection Example - SmartBERT, Loopback Modes, Static Pattern Transmit Pages

In the Eye Monitor page, eye monitoring is done one lane at a time, as shown in the following example.

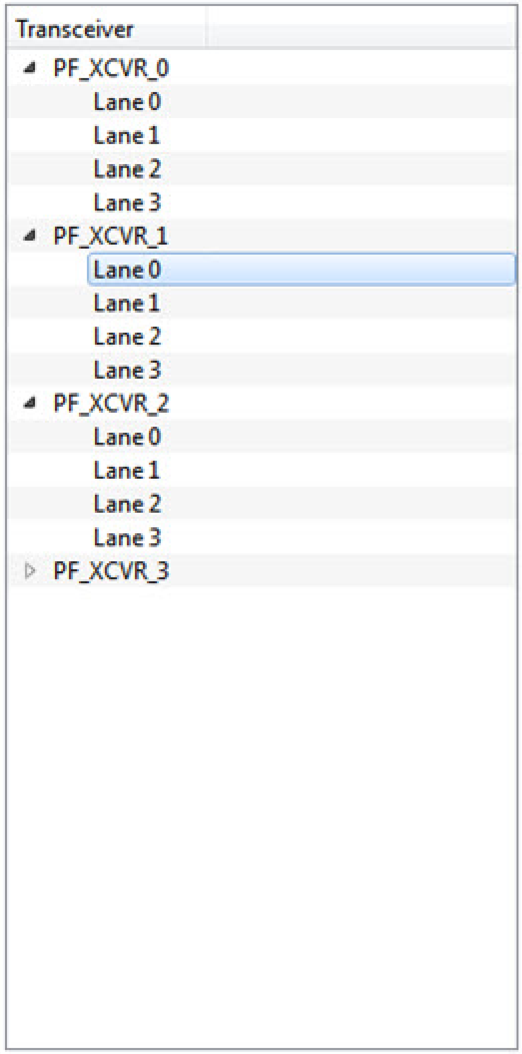


Figure 32 · Transceiver Hierarchy Lane Selection Example - Eye Monitor Page

SmartBERT

In the SmartBERT page of the Debug TRANSCEIVER dialog box, you can select lanes in the Transceiver Hierarchy and use debug options to run Smart BERT tests.

Click the **SmartBERT** tab in the Debug TRANSCEIVER dialog box to open the SmartBERT page.

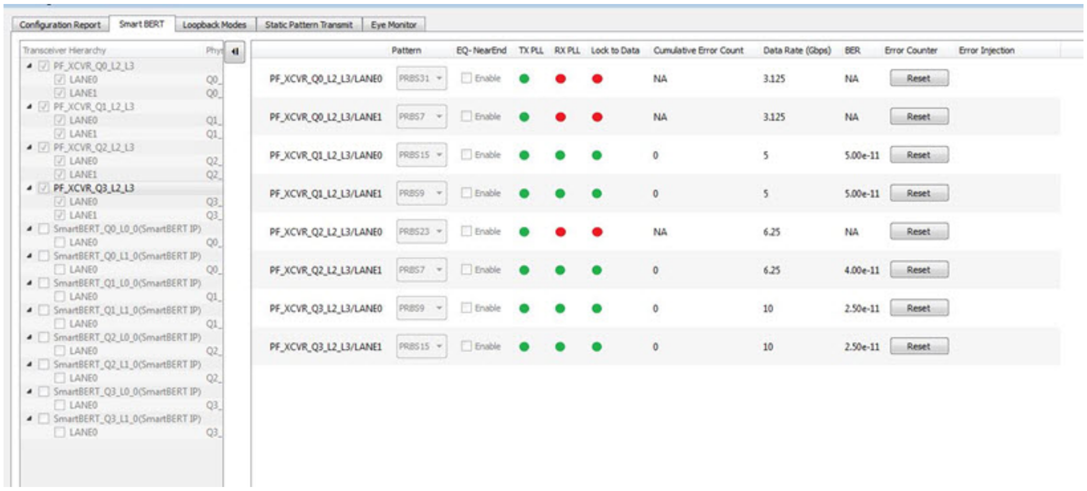


Figure 33 · Debug TRANSCEIVER - SmartBERT

The following input options and outputs are represented as columns:

Pattern – Input option. Select a PRBS pattern type from the drop-down list: PRBS7, PRBS9, PRBS15, PRBS23, or PRBS31. The default is PRBS7.

EQ-NearEnd – Input option. When checked, enables EQ-NearEnd loopback from Lane Tx to Lane Rx.

TX PLL – Indicates if lane is locked onto TX PLL when the SmartBERT test is in progress.

Gray – Indicates test is not in progress

Green – Indicates lane is locked onto TX PLL

Red – Indicates lane is not locked onto TX PLL

RX PLL – Indicates if lane is locked onto RX PLL when the SmartBERT test is in progress.

Gray – Indicates test is not in progress

Green – Indicates lane is locked onto TX PLL

Red – Indicates lane is not locked onto TX PLL

Lock to Data – Indicates if lane is locked onto incoming data / RX CDR PLL when the SmartBERT test is in progress.

Gray – Indicates test is not in progress

Green – Indicates lane is locked onto TX PLL

Red – Indicates lane is not locked onto TX PLL

.**Cumulative Error Count** – Displays the error count when the SmartBERT test is in progress.

Data Rate – Indicates the data rate (in Gbps) configured in the lane.

BER – Calculates the Bit Error Rate (BER) from the cumulative error count and data rate and displays it in the column.

Error Counter Reset – Resets the error counter and BER of the lane. A reset can be done at any time.

All output parameters are updated approximately once per second, with their values retrieved from the device.

To add lanes, in the Transceiver Hierarchy, check the boxes next to the lanes to be added. To remove lanes, uncheck the boxes next to the lanes to be removed.

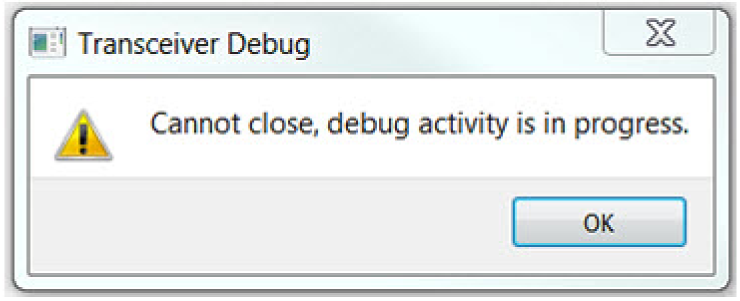
Select the desired options and click **Start** to start the Smart BERT test on all selected lanes.

Note: A popup message appears if a test cannot be started on one lane, multiple lanes, or all lanes. Tests will start normally on all unaffected lanes.

Click the **Phy Reset** button to do a Phy reset on all checked lanes in the Transceiver Hierarchy. This button is disabled when a PRBS test is in progress.

Note: You can navigate to other tabs when a SmartBERT test is in progress, but you cannot perform any debug activity except to use Plot Eye for any lane on the Eye Monitor page.

Note: You cannot close the SmartBERT window when a test is in progress. Attempting to do so will result in the following message:



Click the **Stop** button to stop the SmartBERT test on all lanes simultaneously.

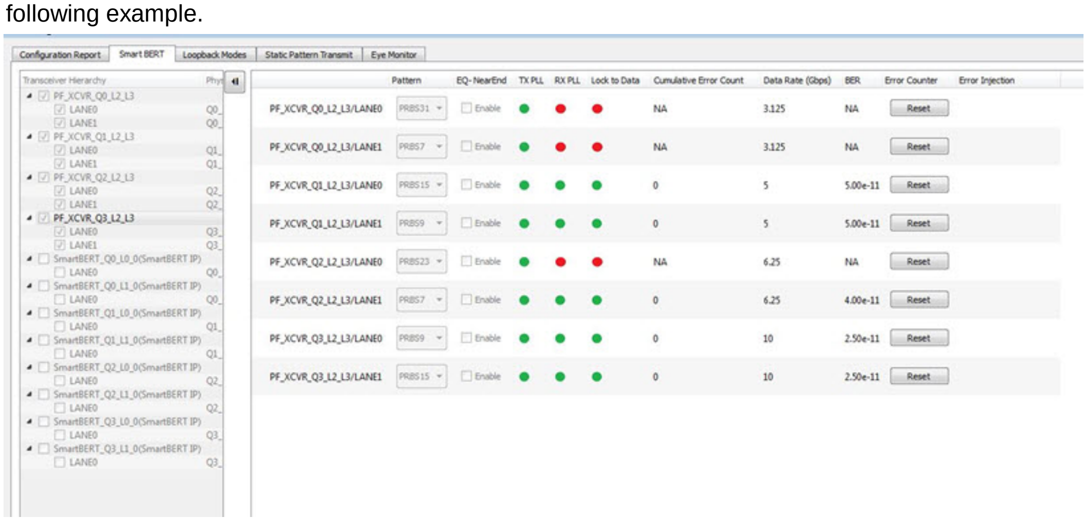
SmartBERT IP

The CoreSmartBERT core provides a broad-based evaluation and demonstration platform for PolarFire transceivers (PF_XCVR). Parameterizable to use different transceivers and clocking topologies, the SmartBERT core can also be customized to use different line rates and reference clock rates. Data pattern generators and checkers are included for each PF_XCVR, giving several different Pseudo-random binary sequences PRBS (2**7**,2**23**, 2**15** and 2**31**).

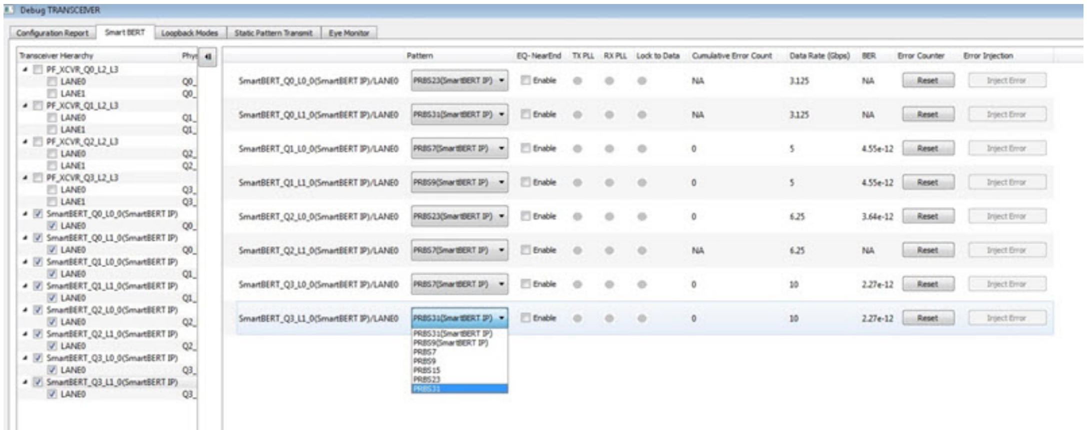
Each SmartBERT IP can have four lanes configured. Each Lane can have the pattern type PRBS7, PRBS9, PRBS23, or PRBS31 configured.

SmartDebug identifies the lanes that are used by the SmartBERT IP and distinguishes them by adding "_IP" to the SmartBERT IP instance name in the Transceiver Hierarchy. See the following example.

You can expand a SmartBERT IP instance to see all the lanes. Check the checkbox next to a lane to add it to the SmartBERT IP page and include the lane in a PRBS test. If the box is unchecked, it will not be added. See the following example.



You can select patterns for the added lane(s) from a drop-down list. See the following example.



After the lane(s) have been added and the patterns(s) selected, click **Start** to enable the transmitter and receiver for the added lanes and patterns.

Error Injection

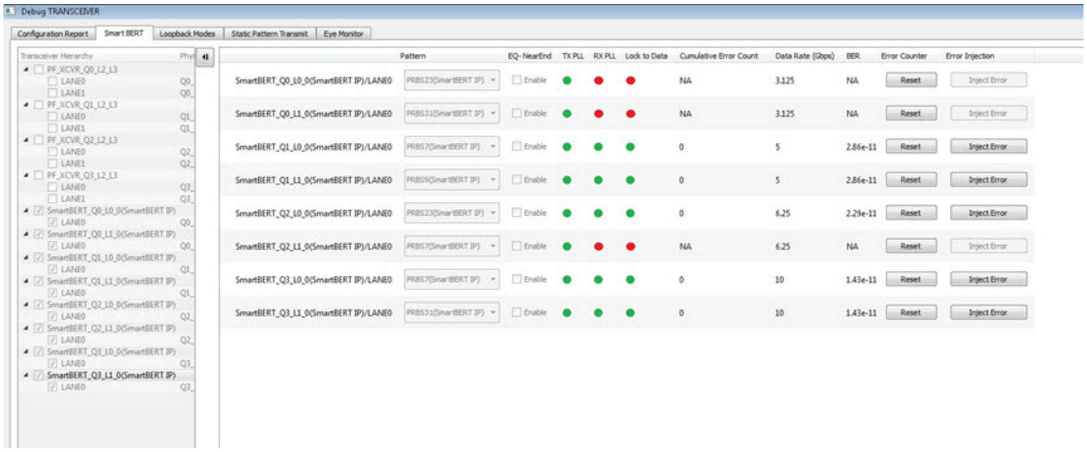
When SmartBERT IP lanes are added, you will see the Error Injection column and Error Inject button. Errors can be injected by clicking the **Error Inject** button when a PRBS test is running. This feature tests whether the error is identified by the pattern checker.

Note: This column does not appear for non-SmartBERT IP lanes, or if a non-configured PRBS pattern has been selected.

Error Count

Error Count is shown when a lane is added and a PRBS pattern is run. The error count can be cleared by clicking the **Reset** button under the Error Counter column.

The following example shows the Reset and Inject Error buttons.



Loopback Modes

The Loopback Modes page in the Debug TRANSCEIVER dialog box allows you to select lanes from the Transceiver Hierarchy and use Loopback Mode debug options.

Click the **Loopback Modes** tab in the Debug TRANSCEIVER dialog box.

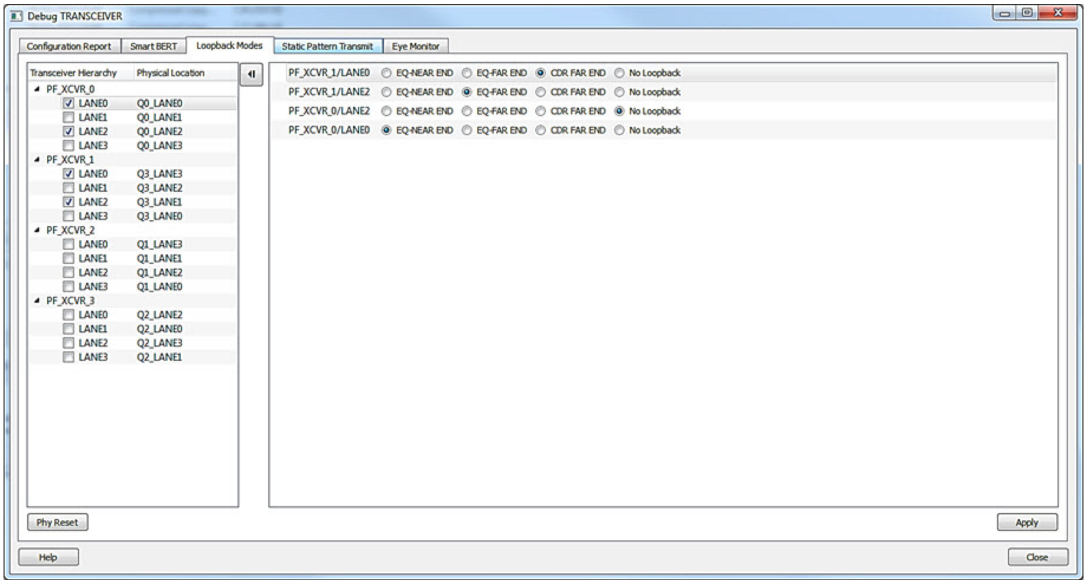


Figure 34 · Debug TRANSCEIVER - Loopback Modes

You can select the desired loopback type (EQ-NEAREND, EQ-FAREND, CDRFAREND, or No Loopback) for each lane.

EQ-NEAR END – Set EQ-Near End loopback from Lane Tx to Lane Rx. This loopback mode is supported up to 10.3125 Gbps.

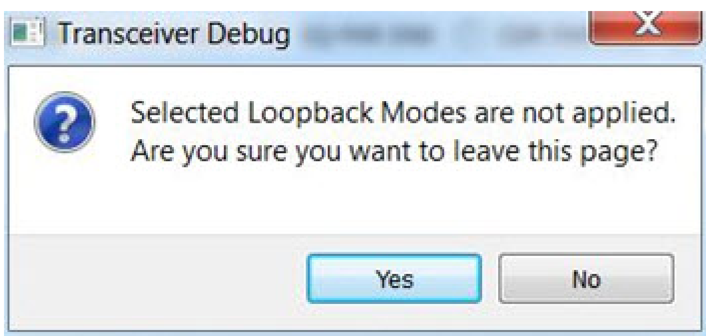
EQ-FAR END – Set EQ-Far End loopback from Lane Tx to Lane Rx.

CDR FAR END – Set CDR Far End loopback from Lane Rx to Lane Tx.

No Loopback – Set this option to have no loopback between Lane Tx and Lane Rx. (For external loopback using PCB backplane or High Speed Loopback cables.)

When you have selected the desired options, click **Apply** to enable the selected loopback mode on the lane(s).

Note: If you proceed to another tab without applying your changes to loopback modes, the following popup message appears:



Click **Yes** to ignore the changed selections and move to another selected page.

Click **No** to remain on the current page.

Static Pattern Transmit

In the Static Pattern Transmit page of the Debug TRANSCEIVER dialog box, you can select lanes from the Transceiver Hierarchy and use Static Pattern Transmit debug options.

Click the **Static Pattern Transmit** tab in the Debug TRANSCEIVER dialog box to open the Static Pattern Transmit page.

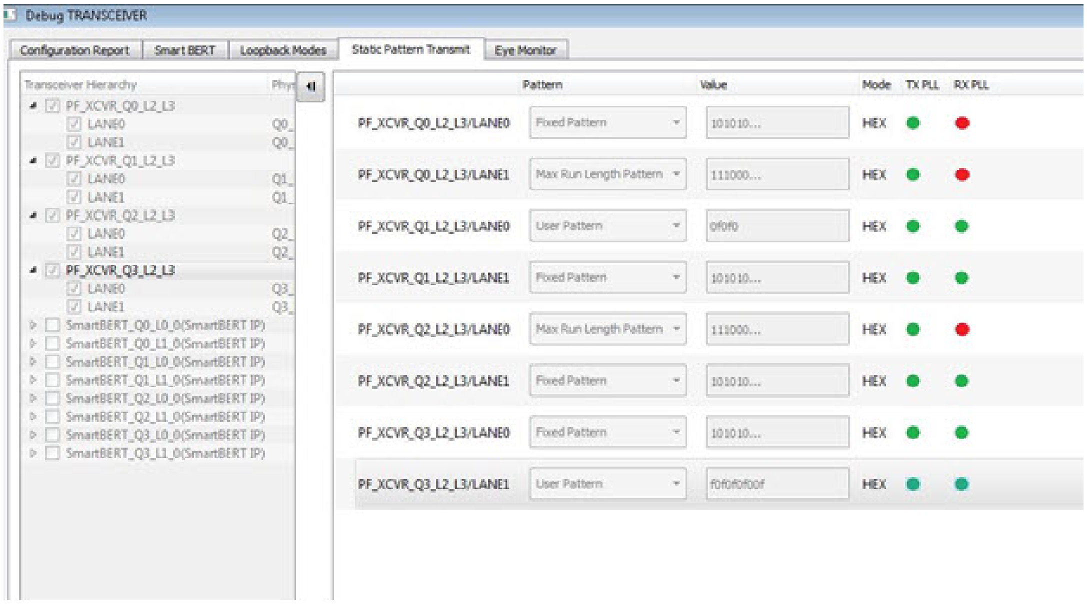


Figure 35 · Debug TRANSCEIVER – Static Pattern Transmit

When a lane is added from the Transceiver Hierarchy, the following debugging options can be selected:

Pattern – Fixed Pattern, Max Run Length Pattern, and User Pattern can be selected from the drop-down list.

- Fixed Pattern is a 10101010... pattern. Length is equal to the data width of the Tx Lane.
- Max Run Length Pattern is a 1111000... pattern. Length is equal to the data width of the Tx Lane, with half 1s and half 0s.
- User Pattern is a user defined pattern in the value column. Length is equal to the data width.

Value – Editor available only with the User Pattern pattern type. For other pattern type selections, it is disabled.

- Takes the input pattern to transmit from the Lane Tx of selected lanes.
- Pattern type should be Hex numbers, and not larger than the data width selected.
- Internal validators dynamically check the pattern and indicate when an incorrect pattern is given as input.

Mode – Currently, HEX mode is supported for pattern type.

TX PLL – Indicates Lane lock onto TX PLL when Static Pattern Transmit is in progress

- *Gray* – Test is not in progress
- *Green* – Lane is locked onto TXPLL
- *Red* – Lane is not locked onto TXPLL

RX PLL – Indicates Lane lock onto RX PLL when Static Pattern Transmit is in progress

- *Gray* – Test is not in progress
- *Green* – Lane is locked onto RXPLL
- *Red* – Lane is not locked onto RXPLL

Click **Start** to start Static Pattern Transmit on selected lanes.

Click **Stop** to stop Static Pattern Transmit test on selected lanes.

Eye Monitor

You can determine signal integrity with the Eye Monitor feature. It allows you to create an eye diagram to measure signal quality. Eye Monitoring estimates the horizontal eye-opening at the receiver serial data sampling point and helps you select an optimum data sampling point at the receiver.

To use the Eye Monitor feature, do the following:

1. Invoke SmartDebug from Libero.
2. Click the **Eye Monitor** tab in the Debug TRANSCEIVER dialog box.

In the Eye Monitor page, you can select a lane and click **Plot Eye** to start eye monitoring for that lane. The eye diagram displays, as shown in the following example.

Note: Ensure data transmission on Lane Rx for successful monitoring.

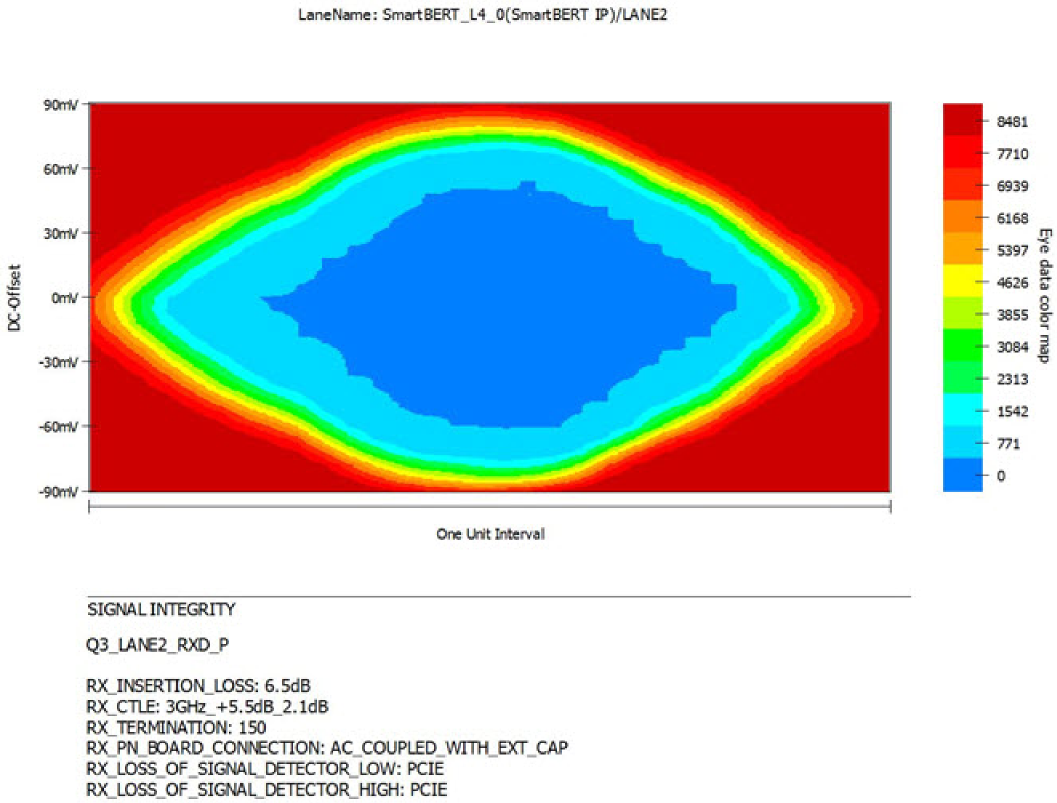


Figure 36 · Eye Monitor Example

You can move to the SmartBERT or Static Pattern Transmit page and start a SmartBERT test or Static Pattern Transmit (using external high speed board cables), respectively, which sends traffic in Lane Rx. You can then return to the Eye Monitor page and click **Plot Eye**.

Signal Integrity

The Signal Integrity feature in SmartDebug works with Signal Integrity in the I/O Editor, allowing the import and export of .pdc files.

The Signal Integrity pane appears in the following SmartDebug pages:

- SmartBERT
- Loopback Modes
- Static Pattern Transmit
- Eye Monitor

When you open Debug Transceiver in SmartDebug and click the SmartBERT, Loopback Modes, Static Pattern Transmit, or Eye Monitor tab, all parameters in the Signal Integrity pane are shown as Undefined. Only the Export All Lanes and Import All Lanes buttons are enabled. See the following example.

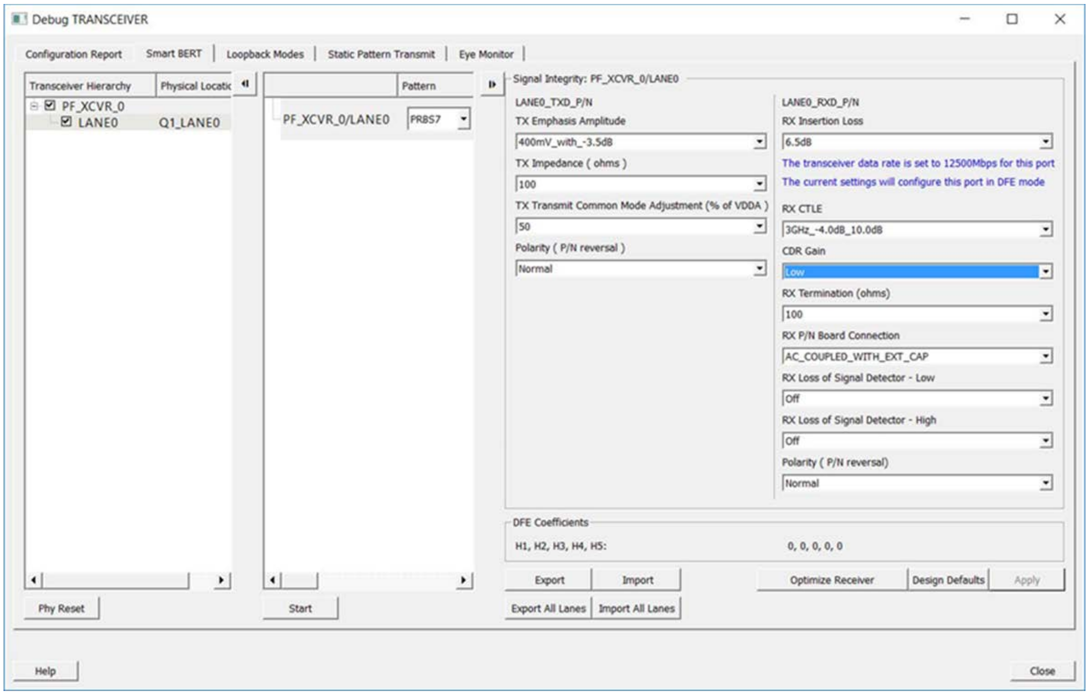


Figure 37 · Debug TRANSCEIVER - Signal Integrity

When a lane is selected in the SmartBERT, Loopback Modes, Static Pattern Transmit, or Eye Monitor pages, the corresponding Signal Integrity parameters (configured in the I/O Editor or changed in SmartDebug) are enabled and shown in the Signal Integrity pane.

The selected lane instance name is displayed in the Signal Integrity group box, and the Export, Import, and Design Defaults buttons are enabled.

You can select options for each parameter from the drop-down for that parameter. Click **Apply** to set the selected transceiver instance with the selected options.

The *Polarity (P/N reversal)* parameter has been added. You can choose Normal or Inverted from the drop-down. Note that this parameter is not available for MPF300T_ES (Rev C) or MPF300T_XT (Rev E) devices.

The *CDR Gain* parameter has been added for MPF300T, MPF100T, MPF200T, MPF500T devices, and you can select the High or Low option from the drop-down. This parameter is supported for Export, Export All, Import, Import All, Design Defaults, and Apply flows of Signal Integrity. Note that this parameter is not available for MPF300T_ES (Rev C) or MPF300T_XT (Rev E) devices.

Note: The Apply button is enabled when you make a selection for any parameter.

If you change parameter options and click another lane, move to another tab, or click Import, Import All, or Design Defaults without applying the changes, you will see the following message stating "*Some Signal Integrity options are modified. How do you wish to continue?*"

Click **Apply** to apply the changes or **Discard** to discard the changes.

Design Defaults

Clicking the **Design Defaults** button loads the Signal Integrity parameter options for the selected lane instance. These are the signal integrity settings that were selected in the Libero design flow run and reside in the STPL file. Design Default parameter options are applied to the device and updated in Modified Constraints.

Note: Modified Constraints is a list of I/O constraints set on the TXP/N and RXP/N lane ports. For a selected lane, this set is created in the SmartDebug session and is updated when a Signal Integrity parameter option is modified and applied or an external PDC file is imported.

Export

Clicking the **Export** button exports the current selected parameter options and other physical information for the selected lane instance to an external PDC file. A pop-up box prompts you to choose the location where you want the .pdc file to be exported.

The exported content will be in the form of two set_io commands, one for the TXP port and one for the RXP port of the selected lane instance.

Export All Lanes

Clicking the **Export All Lanes** button exports the current selected parameter options and other physical information for all lane instances in the design to an external PDC file. A pop-up box prompts you to choose the location where you want the .pdc file to be exported.

Import

Clicking the **Import** button imports Signal Integrity parameter options and other physical information for the selected lane from an external PDC file.

The Signal Integrity parameter options are applied to the device and updated in Modified Constraints.

Import All Lanes

Clicking the Import All Lanes button imports Signal Integrity parameter options and other physical information for all lanes from an external PDC file.

The Signal Integrity parameter options are applied to the device and updated in Modified Constraints.

Signal Integrity and Calibration Report

You can generate and extract an additional report containing Signal Integrity parameters and options, CTLE register settings, and DFE coefficients by clicking **Export** or **Export all** in SmartDebug. Click **Export** to export the report only for the selected lane. Click **Export all** to export a report for all the lanes. This report is a text file that contains the Signal Integrity parameters and options, CTLE register values {CST1, RST1, CST2, RST2}, and DFE coefficient values {H1, H2, H3, H4, H5}. The exported file has a .txt extension with the same name as the .pdc file, and is exported in the same location. DFE Coefficients are exported only for DFE configured lanes. See the following example report.

```
SIGNAL INTEGRITY AND CALIBRATION REPORT
==========================================================================
PF_XCVR_3/LANE0

Signal Integrity:
TX_EMPHASIS_AMPLITUDE=400mV_with_-3.5dB
TX_IMPEDANCE=150
TX_TRANSMIT_COMMON_MODE_ADJUSTMENT=50
TX_POLARITY=Normal
TXPLL_BANDWIDTH=LOW
RX_INSERTION_LOSS=6.5dB
RX_CTLE=3GHz_+5.5dB_2.1dB
RX_TERMINATION=100
RX_PN_BOARD_CONNECTION=AC_COUPLED_WITH_EXT_CAP
RX_LOSS_OF_SIGNAL_DETECTOR_LOW=PCIE
RX_LOSS_OF_SIGNAL_DETECTOR_HIGH=PCIE
RX_POLARITY=Normal
```

```
----------------------------------------------------------------------------------
CTLE Coefficients:
CST1, RST1, CST2, RST2 = 3, 3, 1, 2
----------------------------------------------------------------------------------
DFE Coefficients:
H1, H2, H3, H4, H5 = 0, -1, 2, 6, 3
==============================================================================
======

PF_XCVR_3/LANE0
Signal Integrity:
TX_EMPHASIS_AMPLITUDE=400mV_with_-3.5dB
TX_IMPEDANCE=100
TX_TRANSMIT_COMMON_MODE_ADJUSTMENT=80
TX_POLARITY=Normal
TXPLL_BANDWIDTH=LOW
RX_INSERTION_LOSS=17.0dB
RX_CTLE=3GHz_+5.5dB_2.1dB
RX_TERMINATION=100
RX_PN_BOARD_CONNECTION=AC_COUPLED_WITH_EXT_CAP
RX_LOSS_OF_SIGNAL_DETECTOR_LOW=PCIE
RX_LOSS_OF_SIGNAL_DETECTOR_HIGH=PCIE
RX_POLARITY=Normal
----------------------------------------------------------------------------------
CTLE Coefficients:
CST1, RST1, CST2, RST2 = 3, 1, 2, 2
----------------------------------------------------------------------------------
DFE Coefficients:
H1, H2, H3, H4, H5 = Not applicable for CDR configured lane
```

Optimize Receiver

Note: This feature is available for MPF300T, MPF100T, MPF200T, and MPF500T devices.

The Optimize Receiver function allows you to optimize the DFE coefficients and/or CTLE settings for the selected lanes, depending on receiver mode. For CDR mode receivers, CTLE settings can be optimized. For DFE mode receivers, CTLE settings and DFE coefficients can be optimized.

For DFE coefficients, the optimize function runs through an algorithm for each lane and sets the best available coefficients for each selected lane for the current temperature, voltage, and data pattern conditions. After the optimization is complete, the transceiver lanes are set to these coefficients for the user to continue debugging.

For information about how to use the optimized coefficients without SmartDebug, see the PolarFire FPGA Transceiver User Guide.

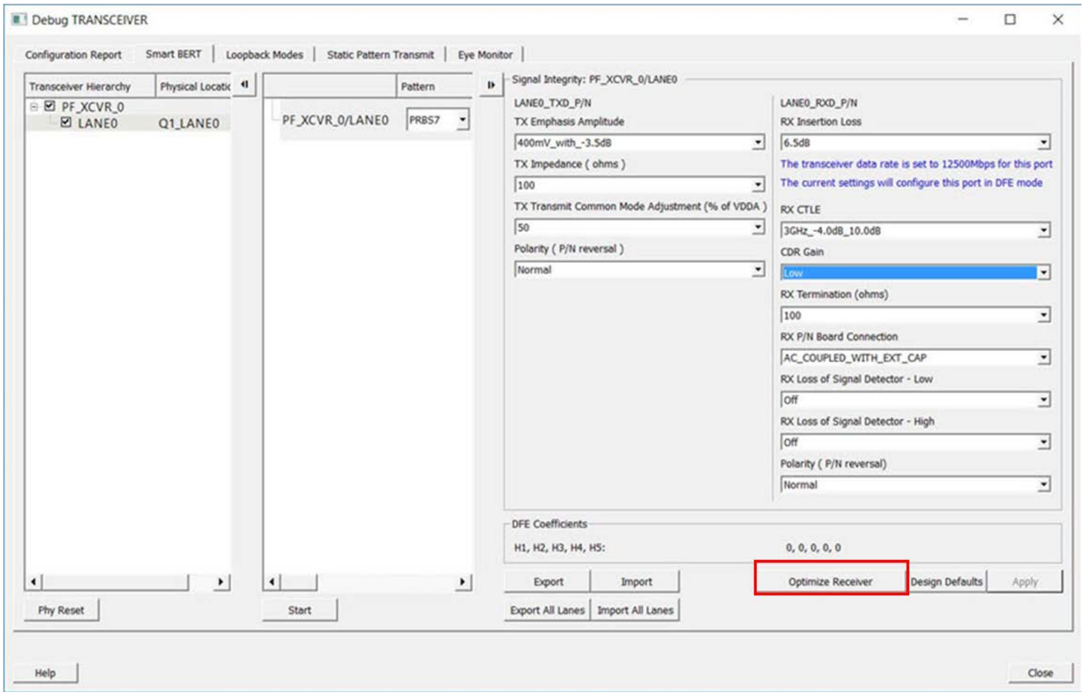


Figure 38 · Debug TRANSCEIVER - Optimize Receiver

Click **Optimize Receiver** to open the Optimize Receiver dialog box. This dialog box shows the lanes that are configured in the design.

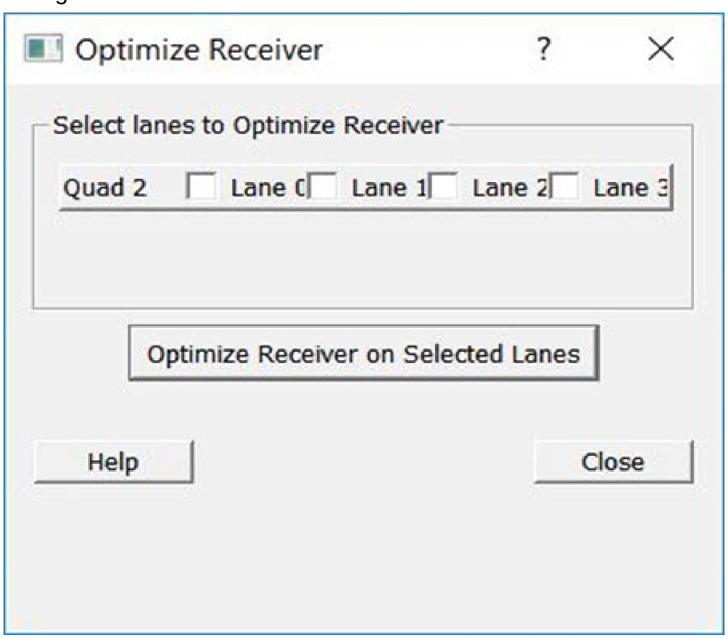


Figure 39 · Optimize Receiver Dialog Box

Select the lanes on which to run Optimize Receiver and click **Optimize Receiver on Selected Lanes**. You can select any combination of lanes including those configured in CDR or DFE. The hardware will perform CTLE calibration for CDR receivers and Full calibration for DFE receivers.

Display DFE Coefficient values

The DFE coefficients H1, H2, H3, H4, and H5 are displayed as non-editable in the Signal Integrity pane for any lane configured in DFE mode. See the following example figure.

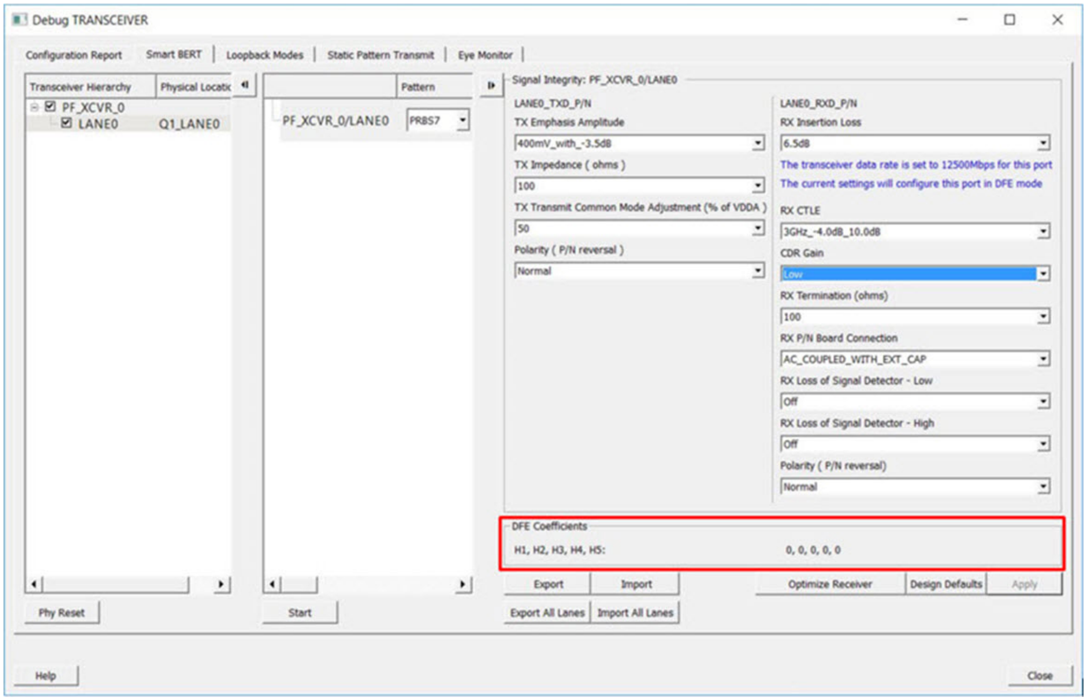


Figure 40 · Signal Integrity Pane - DFE Coefficients

DFE coefficients for CDR lanes are displayed as NA.

The DFE coefficients are read from the register fields as follows:

H1 = H1_MON

H2 = H2_MON

H3 = H3_MON

H4 = H4_MON

H5 = H5_MON

DFE coefficients are read back from the device in the following scenarios:

- When the lane is selected in the SmartBert, Loopback Modes, Static Pattern Transmit, and Eye Monitor pages.
- When a test is started/stopped on selected lane in the SmartBert page
- When a test is started/stopped on selected lane in the Static Pattern Transmit page
- When Optimize Receiver is executed on the selected lane.

Signal Integrity and Calibration Report

You can generate and extract an additional report containing Signal Integrity parameters and options, CTLE register settings, and DFE coefficients by clicking **Export** or **Export all** in SmartDebug. Click **Export** to export the report only for the selected lane. Click **Export all** to export a report for all the lanes. This report is a text file that contains the Signal Integrity parameters and options, CTLE register values {CST1, RST1, CST2, RST2}, and DFE coefficient values {H1, H2, H3, H4, H5}. The exported file has a .txt extension with the same name as the .pdc file, and is exported in the same location. DFE Coefficients are exported only for DFE configured lanes. See the following example report.

```
SIGNAL INTEGRITY AND CALIBRATION REPORT
================================================================================
PF_XCVR_3/LANE0

Signal Integrity:
TX_EMPHASIS_AMPLITUDE=400mV_with_-3.5dB
TX_IMPEDANCE=150
TX_TRANSMIT_COMMON_MODE_ADJUSTMENT=50
TX_POLARITY=Normal
TXPLL_BANDWIDTH=LOW
RX_INSERTION_LOSS=6.5dB
RX_CTLE=3GHz_+5.5dB_2.1dB
RX_TERMINATION=100
RX_PN_BOARD_CONNECTION=AC_COUPLED_WITH_EXT_CAP
RX_LOSS_OF_SIGNAL_DETECTOR_LOW=PCIE
RX_LOSS_OF_SIGNAL_DETECTOR_HIGH=PCIE
RX_POLARITY=Normal
--------------------------------------------------------------------------------
CTLE Coefficients:
CST1, RST1, CST2, RST2 = 3, 3, 1, 2
--------------------------------------------------------------------------------
DFE Coefficients:
H1, H2, H3, H4, H5 = 0, -1, 2, 6, 3
==================================================================================
======

PF_XCVR_3/LANE0
Signal Integrity:
TX_EMPHASIS_AMPLITUDE=400mV_with_-3.5dB
TX_IMPEDANCE=100
TX_TRANSMIT_COMMON_MODE_ADJUSTMENT=80
TX_POLARITY=Normal
TXPLL_BANDWIDTH=LOW
RX_INSERTION_LOSS=17.0dB
RX_CTLE=3GHz_+5.5dB_2.1dB
RX_TERMINATION=100
RX_PN_BOARD_CONNECTION=AC_COUPLED_WITH_EXT_CAP
RX_LOSS_OF_SIGNAL_DETECTOR_LOW=PCIE
RX_LOSS_OF_SIGNAL_DETECTOR_HIGH=PCIE
RX_POLARITY=Normal
--------------------------------------------------------------------------------
CTLE Coefficients:
CST1, RST1, CST2, RST2 = 3, 1, 2, 2
--------------------------------------------------------------------------------
DFE Coefficients:
H1, H2, H3, H4, H5 = Not applicable for CDR configured lane
```

Optimize DFE Coefficients

Note: This feature is available for MPF300T_ES (Rev C) and MPF300T_XT (Rev E) devices.

The Optimize DFE Coefficients function allows you to optimize the DFE coefficients for the selected lanes. The optimize function runs through an algorithm for each lane and programs the best available coefficients for each selected lane for the current temperature, voltage, and data pattern conditions. After the optimization is complete, the transceiver lanes are programmed with these coefficients for the user to continue debugging.

Note: Optimize DFE Coefficients is only supported for data rates >= 5Gbps.

For information about how to use the optimized coefficients without SmartDebug, see the PolarFire FPGA Transceiver User Guide.

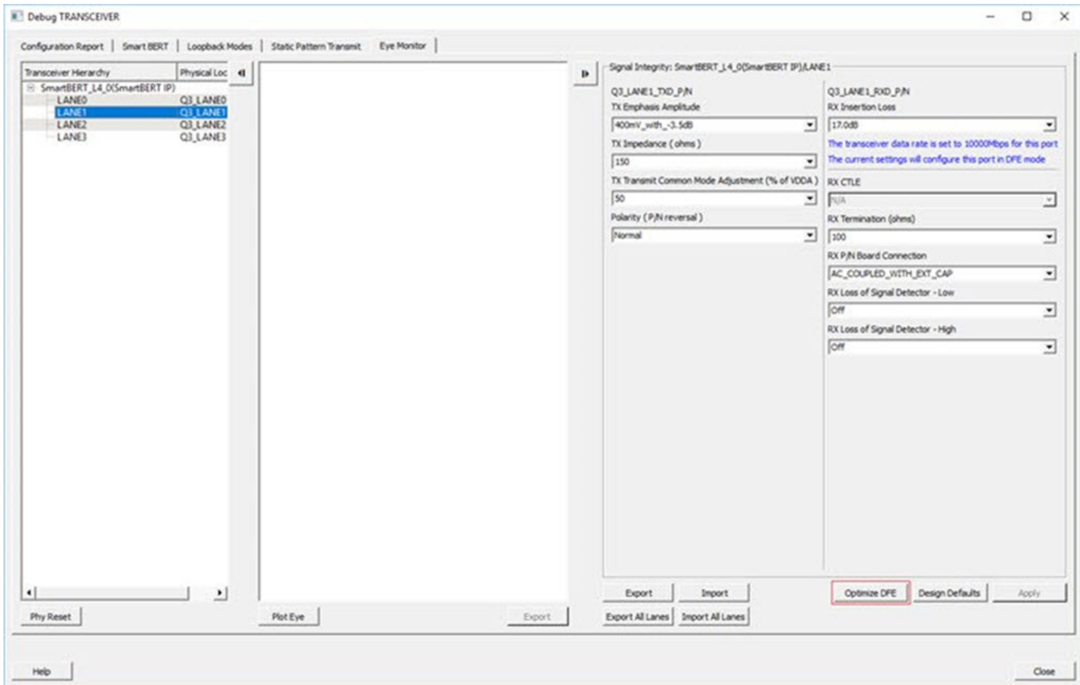


Figure 41 · Debug TRANSCEIVER - Optimize DFE

Click **Optimize DFE** to open the Optimize DFE dialog box. This dialog box shows the lanes that are configured in the design.

There are two options to choose from in the DFE Algorithm pull-down menu:

- **Software Based** - executes DfeSs.tcl script on all selected lanes

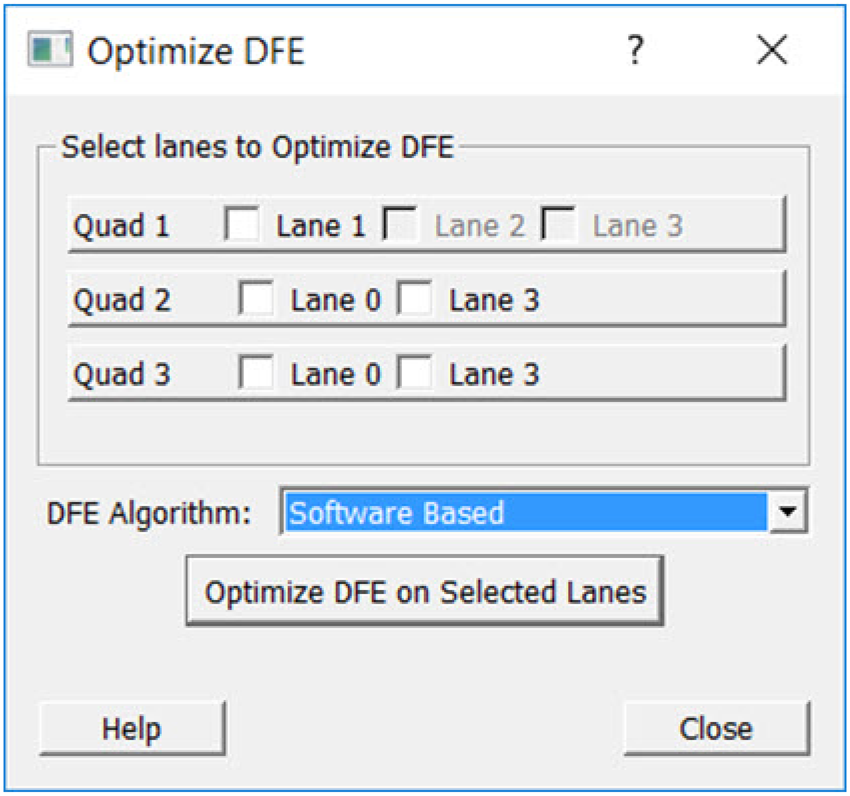


Figure 42 · Optimize DFE Dialog Box- Software Based DFE Algorithm

- **XCVR Based (Built-in)** - executes internal Dfe Auto Calibration on all selected lanes

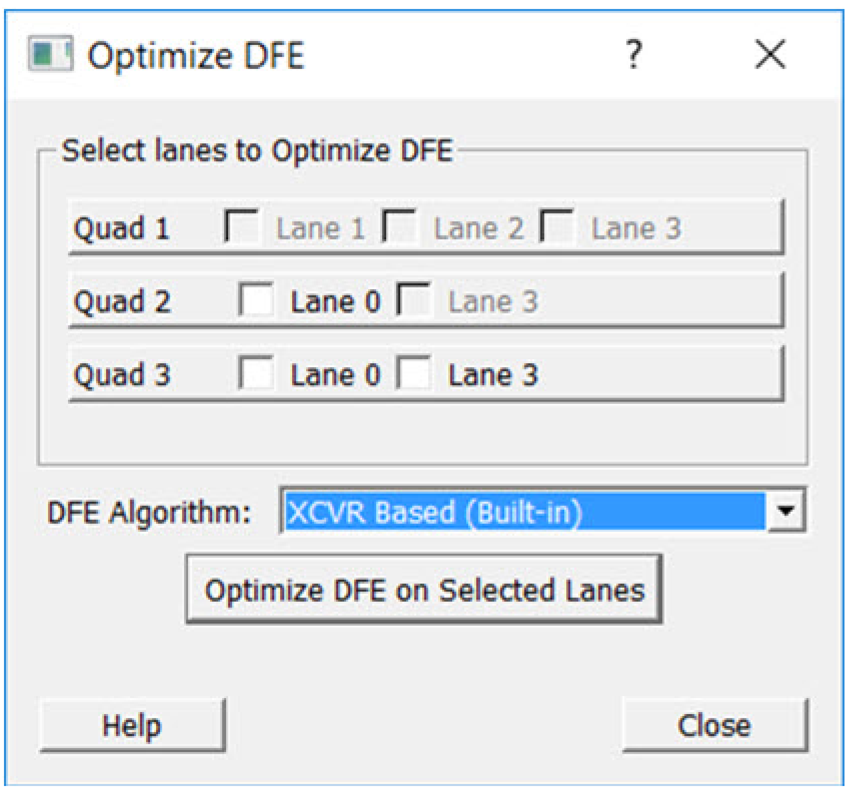


Figure 43 · Optimize DFE Dialog Box- XCVR Based (Built-in) DFE Algorithm

Note: When the XCVR Based (built-in) algorithm is selected, the lanes configured in CDR mode are disabled, as internal Dfe Auto Calibration is not supported for CDR Configured Devices.

Select the lanes on which to run Optimize DFE and click **Optimize DFE on Selected Lanes**.

A Tcl script runs on the selected lanes to optimize the DFE coefficients.

To use these optimized coefficients directly in the STAPL file from Libero, you need to use a generated file. After the TCL script completes, an override file (<Design_Name>_SD_DFE.txt) is generated. This file contains all the registers that were updated with the new coefficients.

The override file has the format <Reg. Address> <Reg. Value> and is saved in integrated flow in the designer folder. For standalone SmartDebug, the file is saved in the directory where the contents of ddc were extracted to when the standalone project was created.

In Libero, select the **Configure Design Initialization Data and Memories** tool, select the generated override file as the override file, and generate a new STAPL file.

Note: Additional design work must be done to use the DFE feature. See the PolarFire FPGA Transceiver User Guide for more information.

In the Libero **Configure Design Initialization Data and Memories** tool, select the generated UIC file as the override file and generate a new STAPL file.

Debug uPROM

You can debug clients configured in a design and debug µPROM memory address information with the Debug µPROM feature.

In the main SmartDebug window, click **Debug uPROM**.

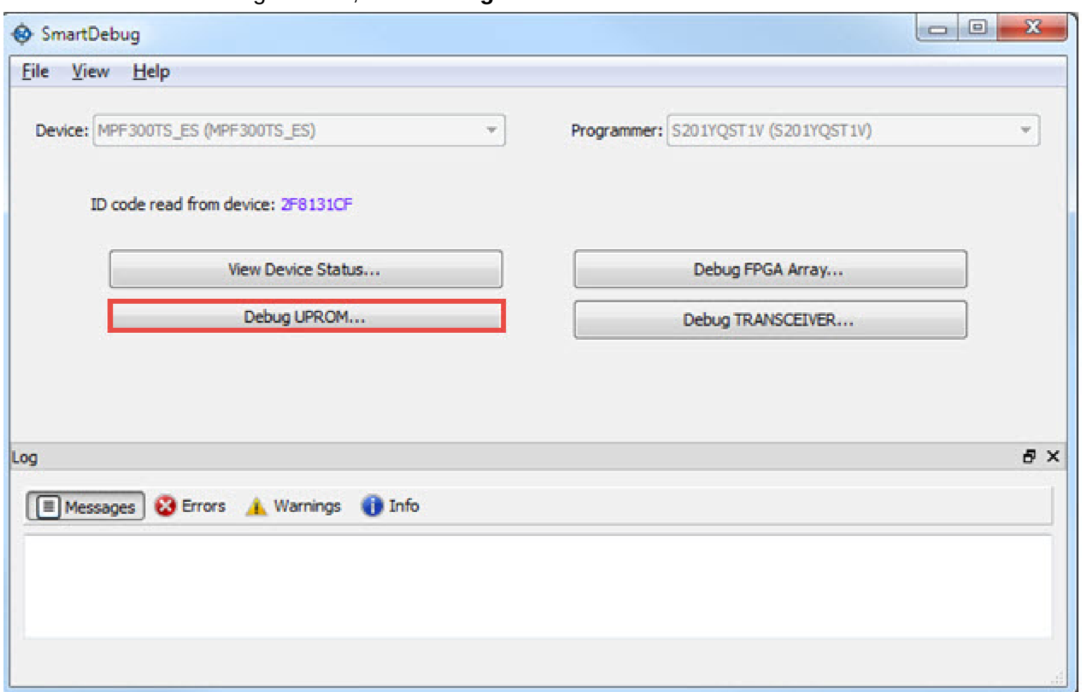


Figure 44 · SmartDebug Window - Debug uPROM

If a µPROM memory block is used in the Libero design, the µPROM Debug window appears.

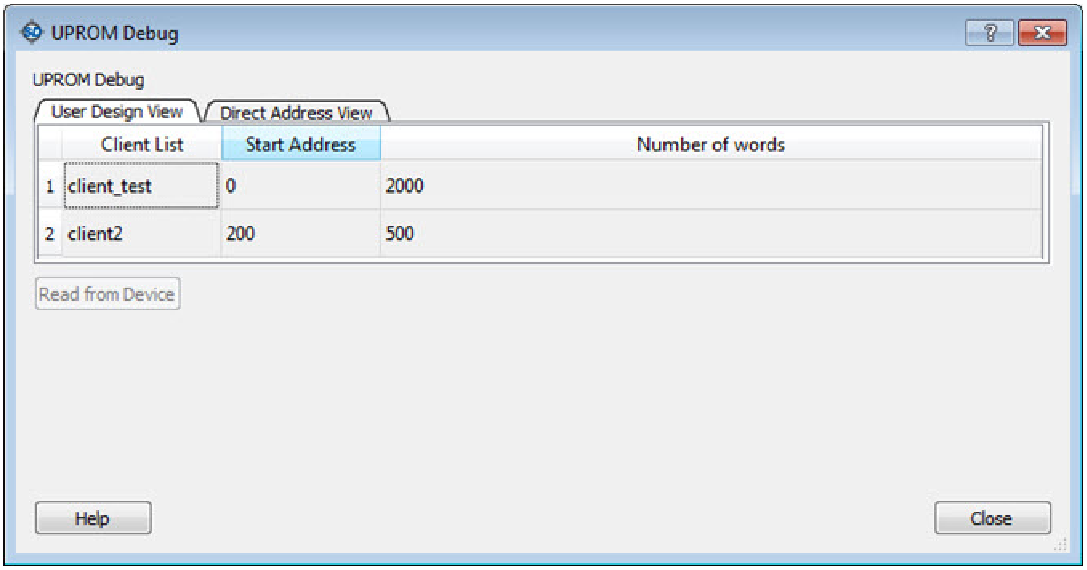
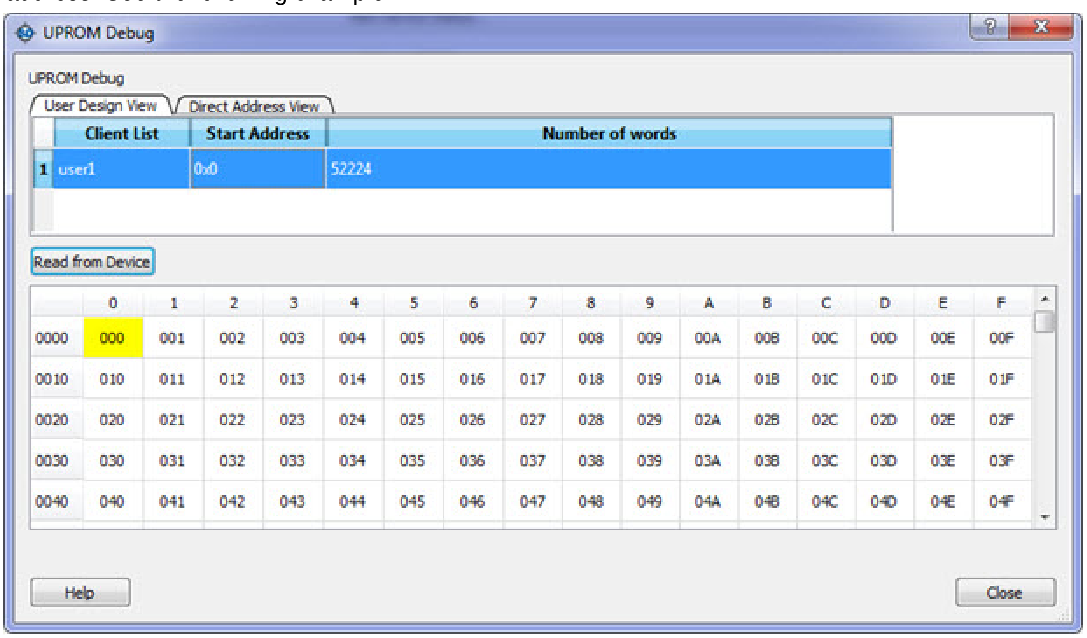


Figure 45 · µPROM Debug Window

User Design View

The User Design View tab in the µPROM Debug window lists all clients configured in the design. Selecting a client in the list enables the **Read from Device** button.

Clicking the **Read from Device** button displays a table showing the data in the location at the selected client address. See the following example.

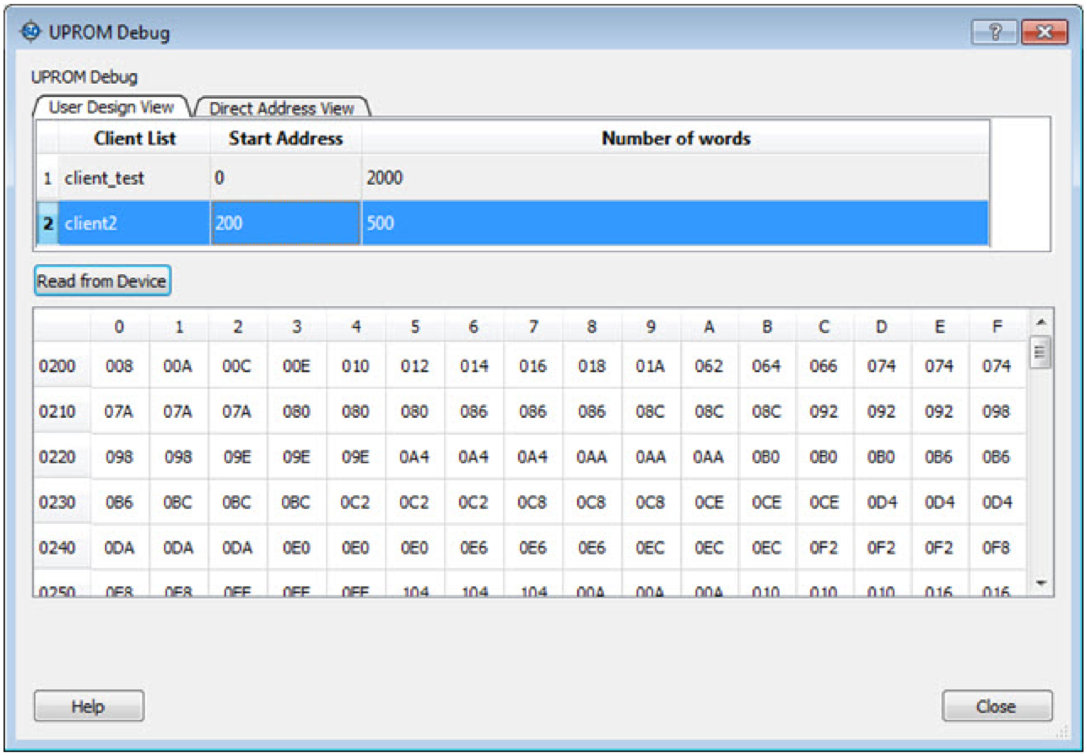


The Client address is associated with *Start Address* and *Number of 9-bit words*. Therefore, the table will contain as many locations as the number of 9-bit words.

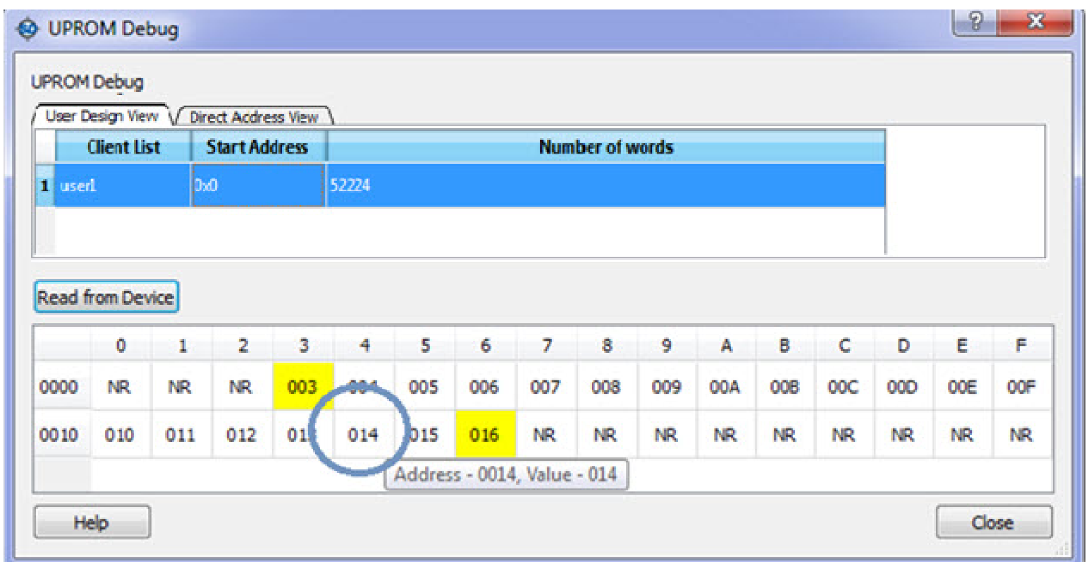
In the example above, *Number of 9-bit words* is 52224, so 52224 words will be shown in the table.

Column headers are numbered 0 to F in hexadecimal format, representing 16 words in a row.

Row addresses begin with a word address associated with *Start Address*. For example, if the *Start Address* is 0x15 (hex), the starting row has an address of 0x0010.



You can hover over a cell to see its address and value, as shown in the following example.



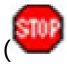
Direct Address View

The Direct Address View tab in the µPROM Debug window provides access to µPROM memory. You can read a part of a client or more than one client by specifying the *Start Address* and *Number of 9-bit words*.

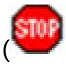
Start Address - hexadecimal value (0 -9, A-F, upper/lower case)

Values are validated and errors are indicated by a red "STOP" icon (). The error message displays when you hover over the icon.

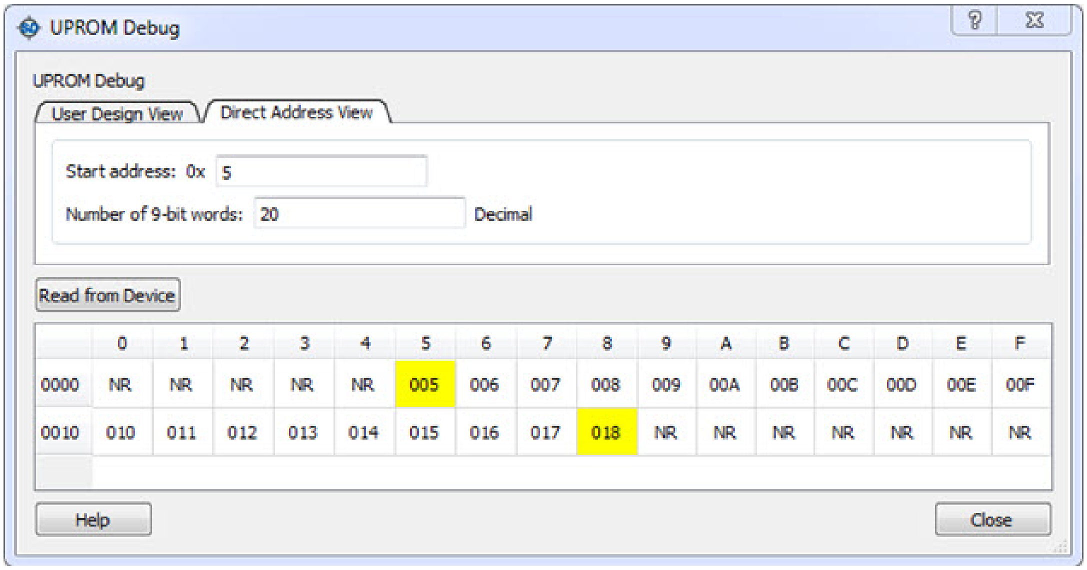
Number of 9-bit words - positive integer value

Values are validated and errors are indicated by a red "STOP" icon (). The error message displays when you hover over the icon.

Read from Device - Disabled until valid values are entered in the fields.

Invalid or blank values are indicated by a red "STOP" icon (). The error message displays when you hover over the icon.

Note: If the word falls within the 16 words that are placed in a row, the start location and the end location are highlighted in the row to show the starting point of the data. All preceding locations show ‘NR’ (Not Read). See the following example.



Notes:

When one field is entered, both fields are validated to enable the Read from Device button.

If fields change after enabling Read from Device, values are validated again and Read from Device may be disabled if invalid values are entered.

If the µPROM Debug window is closed and reopened, the session is retained. The µPROM Debug session is lost only if the main SmartDebug window is closed.

SmartDebug Tcl Commands

SmartDebug Tcl Support

The following table lists the Tcl commands related to SmartDebug for PolarFire. Click the command to view more information.

Table 1 · SmartDebug Tcl Commands

Command	Action
Probe	
add_probe_insertion_point	Adds probe points to be connected to user-specified I/Os for probe insertion flow.
add_to_probe_group	Adds the specified probe points to the specified probe group
create_probe_group	Creates a new probe group.
delete_active_probe	Deletes either all or the selected active probes.
load_active_probe_list	Loads the list of probes from the file.
move_to_probe_group	Moves the specified probe points to the specified probe group.
program_probe_insertion	Runs the probe insertion flow on the selected nets.
remove_probe_insertion_point	Deletes an added probe from the probe insertion UI.
set_live_probe	Set Live probe channels A and/or B to the specified probe point (or points).
select_active_probe	Manages the current selection of active probe points to be used by active probe READ operations.
read_active_probe	Reads active probe values from the device.
remove_from_probe_group	Move out the specified probe points from the group.
save_active_probe_list	Saves the list of active probes to a file.
select_active_probe	Manages the current selection of active probe points to be used by active probe READ operations.
ungroup	Disassociates the probes as group.
unset_live_probe	Discontinues the debug function and clears live probe channels.
write_active_probe	Sets the target probe point on the device to the specified value.
LSRAM	
read_lsram	Reads a specified block of large SRAM from the device.
write_lsram	Writes a seven bit word into the specified large SRAM location.

Command	Action
Probe	
uSRAM	
read_usram	Reads a uSRAM block from the device.
write_usram	Writes a seven bit word into the specified uSRAM location.
Transceiver	
loopback_mode	Applies loopback to a specified lane.
smartbert_test	Starts and stops a Smart BERT test and resets error counter.
static_pattern_transmit	Starts and stops a Static Pattern Transmit.
plot_eye	Plots eye and exports eye plots.
xcvr_read_register	Reads SCB registers and their field values.
xcvr_write_register	Writes SCB registers and their field values.
Additional Commands	
get_programmer_info	Lists the IDs of all FlashPRO programmers connected to the machine.
uprom_read_memory	Reads uPROM memory block from the device.
Standalone SmartDebug Commands	
construct_chain_automatically	Automatically starts chain construction for the specified programmer.
scan_chain_prg	In single mode, this Tcl command runs scan chain on a programmer. In chain mode, this Tcl command runs scan and check chain on a programmer if devices have been added in the grid.
enable_device	Enables or disables a device in the chain.
set_debug_programmer	Sets the debug programmer.
set_device_name	Sets the device name.
set_programming_file	Sets the programming file for a device.
set_programming_action	Selects tehe action for a device.
run_selected_actions	Runs the selected action for a device.

add_probe_insertion_point

This Tcl command adds probe points to be connected to user-specified I/Os for probe insertion flow.

```
add_probe_insertion_point –net net_name -driver driver -pin package_pin_name -port port name
```

Arguments

`-net net_name`

Name of the net used for probe insertion.

`-driver driver`

Driver of the net.

`-pin package_pin_name`

Package pin name (i.e. I/O to which the net will be routed during probe insertion).

`-port port_name`

User-specified name for the probe insertion point.

Example

```
add_probe_insertion_point -net {count_out_c[0]} -driver {Counter_8bit_0_count_out[0]:Q} -
pin {H5} -port {Probe_Insert0}
```

add_to_probe_group

Tcl command; adds the specified probe points to the specified probe group.

```
add_to_probe_group -name probe_name -group group_name
```

Arguments

`-name probe_name`

Specifies one or more probes to add.

`-group group_name`

Specifies name of the probe group.

Example

```
add_to_probe_group -name out[5]:out[5]:Q \
                   -name grp1.out[3]:out[3]:Q \
                   -name out.out[1].out[1]:Q \
                   -group my_new_grp
```

construct_chain_automatically

This Tcl command automatically starts chain construction for the specified programmer.

```
construct_chain_automatically -name {programmer_name}
```

Arguments

`-name`

Specify the device (programmer) name. This argument is mandatory.

Example

For a single programmer:

```
construct_chain_automatically -name {21428}
```

See Also

scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_file

set_programming_action

run_selected_actions

create_probe_group

Tcl command; creates a new probe group.

```
create_probe_group -name group_name
```

Arguments

`-name group_name`

Specifies the name of the new probe group.

Example

```
create_probe_group -name my_new_grp
```

delete_active_probe

Tcl command; deletes either all or the selected active probes.

Note: You cannot delete an individual probe from the Probe Bus.

```
delete_active_probe -all | -name probe_name
```

Arguments

`-all`

Deletes all active probe names.

`-name probe_name`

Deletes the selected probe names.

Example

```
delete –all      <- deletes all active probe names
delete -name out[5]:out[5]:Q \
       -name my_grp1.out[1]:out[1]:Q            #deletes the selected probe names
delete -name my_grp1 \
       -name my_bus             #deletes the group, bus and their members.
```

enable_device

This Tcl command enables or disables a device in the chain. When the device is disabled, it is bypassed. The device must be a Microsemi device.

```
enable_device -name {device_name} -enable {1 | 0}
```

Arguments

`-name`

Specify the device name. This argument is mandatory.

`-enable`

Specify the enable device. This argument is mandatory.

Example

```
enable_device -name {MPF300 (T_ES|TS_ES)} -enable 1
```

See Also

construct_chain_automatically

scan_chain_prg

set_debug_programmer

set_device_name

set_programming_file

set_programming_action

run_selected_actions

get_programmer_info

This Tcl command lists the IDs of all FlashPRO programmers connected to the machine.

```
get_programmer_info
```

This command takes no arguments.

Example

```
set a [get_programmer_info]
```

load_active_probe_list

Tcl command; loads the list of probes from the file.

```
load_active_probe_list –file file_path
```

Arguments

`-file file_path`

The input file location.

Example

```
load_active_probe_list -file "./my_probes.txt"
```

loopback_mode

This Tcl command applies loopback to a specified lane.

```
loopback_mode -lane {Physical_Location} -apply -type {loopback_type}
```

Arguments

`-lane {Physical_Location}`

Specify the physical location of the lane.

`-apply`

Apply specified loopback to specified lane.

`-type {loopback_type}`

Specify the loopback type to apply.

Examples

```
loopback_mode -lane {Q3_LANE2} -apply -type {EQ-NearEnd}
loopback_mode -lane {Q3_LANE0} -apply -type {EQ-FarEnd}
loopback_mode -lane {Q0_LANE0} -apply -type {CDRFarEnd}
loopback_mode -lane {Q0_LANE1} -apply -type {NoLpbk}
loopback_mode -lane {Q1_LANE2} -apply -type {EQ-FarEnd}
loopback_mode -lane {Q1_LANE0} -apply -type {NoLpbk}
loopback_mode -lane {Q2_LANE2} -apply -type {EQ-NearEnd}
loopback_mode -lane {Q2_LANE3} -apply -type {CDRFarEnd}
```

move_to_probe_group

Tcl command; moves the specified probe points to the specified probe group.

Note: Probe points related to a bus cannot be moved to another group.

```
move_to_probe_group -name probe_name -group group_name
```

Arguments

`-name probe_name`

Specifies one or more probes to move.

`-group group_name`

Specifies name of the probe group.

Example

```
move_to_probe_group -name out[5]:out[5]:Q \
                    -name grp1.out[3]:out[3]:Q \
                    -group my_grp2
```

optimize_dfe

This Tcl command supports the Optimize DFE feature in SmartDebug.

```
optimize_dfe -dfe_algorithm <type of dfe algorithm> -lane <lane(s) configured in the design>
```

Arguments

`-dfe_algorithm`

This command executes Dfe Algorithm with type of dfe algorithm and lanes as input. Algorithm selection has two options:

`software_based` – executes DfeSs.tcl script

`xcvr_based` –executes internal Dfe Auto Calibration.

This argument is mandatory.

`-lane`

List of lane(s) configured in the design.

This argument is mandatory.

Examples

```
optimize_dfe -lane {"Q2_LANE0"} -dfe_algorithm {software_based}
optimize_dfe -lane {"Q2_LANE0"} -dfe_algorithm {xcvr_based}
optimize_dfe -lane {"Q2_LANE0" “Q0_LANE0”} -dfe_algorithm {xcvr_based}
```

plot_eye

This Tcl command is used to plot eye and export eye plots.

```
plot eye -lane {lane_instance_name} -export_dir {location_path}
```

Arguments

`-lane`

Specify the lane instance name.

`-export_dir`

Specify the path to the location where the file is to be exported.

Supported Families

PolarFire

Example

```
plot_eye -lane {Q2_LANE0} - export_dir {E:\designs\G5\SERDES\ export.txt}
```

program_probe_insertion

This Tcl command runs the probe insertion flow on the selected nets.

```
program_probe_insertion
```

This command takes no arguments.

read_active_probe

Tcl command; reads active probe values from the device. The target probe points are selected by the select_active_probe command.

```
read_active_probe [-deviceName device_name] [-name probe_name] [-group_name bus_name|group_name] [-value_type b|h][-file file_path]
```

Arguments

`-deviceName device_name`

Parameter is optional if only one device is available in the current configuration.

-name *probe_name*

Instead of all probes, read only the probes specified. The probe name should be prefixed with bus or group name if the probe is in the bus or group.

-group_name *bus_name* | *group_name*

Instead of all probes, reads only the specified buses or groups specified here.

-value_type b | h

Optional parameter, used when the read value is stored into a variable as a string.

b = binary

h = hex

-file *file_path*

Optional. If specified, redirects output with probe point values read from the device to the specified file.

Note: When the user tries to read at least one signal from the bus/group, the complete bus or group is read. The user is presented with the latest value for all the signals in the bus/group.

Example

```
read_active_probe -group_name {bus1}
read_active_probe -group_name {group1}
```

To save into variable:

`set a [read_active_probe -group_name {bus_name} -value_type h]` #save read data in hex string

If read values are stored into a variable without specifying value_type parameter, it saves values as a binary string by default.

Example

`set a [read_active_probe ]` #sets variable a as binary string of read values after read_active_probe command.

read_lsram

Tcl command; reads a specified block of large SRAM from the device.

Physical block

```
read_lsram -name block_name -fileName file_name
```

Arguments

-name *block_name*

Specifies the name for the target block.

-fileName *file_name*

Optional; specifies the output file name for the data read from the device.

Exceptions

- Array must be programmed and active
- Security locks may disable this function

Example

Reads the LSRAM Block Fabric_Logic_0/U2/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM1K20_IP from the PolarFire device and writes it to the file output.txt.

```
read_lsram -name {Fabric_Logic_0/U2/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM1K20_IP} –
fileName {output.txt}
```

Logical block

```
read_lsram -logicalBlockName block_name -port port_name
```

Arguments

`-logicalBlockName block_name`

Specifies the name for the user defined memory block.

`-port port_name`

Specifies the port for the memory block selected. Can be either Port A or Port B.

Example

```
read_lsram -logicalBlockName {Fabric_Logic_0/U2/F_0_F0_U1} -port {Port A}
```

read_snvm_memory

This Tcl command reads pages present in sector 1 of sNVM memory. 221 pages can be accessed through SmartDebug tool. This command can also read encrypted and authenticated pages.

```
read_snvm_memory [-deviceName device_name] [-client client_name] -startpage integer_value -
endpage integer_value [-fileName snvm_data_file_name] -uskKey usk_key
```

Arguments

`-deviceName device_name`

Specifies the device to which a sNVM read is requested. This parameter is optional.

`-client client_name`

Specifies the client name that was configured in the design. This parameter is optional. This will be recorded if client view is requested.

`-startpage integer_value`

Specifies the page value that is the beginning of the range of pages to be read.

`-endpage integer_value`

Specifies the page value that is the end of the pages to be read.

`-fileName snvm_data_file_name`

Specifies the file name to which the output will be redirected. This parameter is optional.

`-uskKey usk_key`

Specifies the user secret key for each page. This field can have multiple keys that will be equal to the number of pages requested. The key must be 24 hexadecimal characters or 0. Multiple keys can be entered by separating them with a colon (:).

If the page is plaintext, the USK value should be set to 0.

Examples

Single page read from client:

read_snvm_memory -deviceName {MPF300TS_ES} -client {client_auth_PT} -startpage {1} -endpage {1} -uskKey {0123456789ABCDEF01234567} –fileName {output.log}

Multiple pages read from client:

read_snvm_memory -deviceName {MPF300TS_ES} -client {client_PlainText} -startpage {0} -endpage {1} -uskKey {0:0} –fileName {output.log}

Page range read without client:

read_snvm_memory -deviceName {MPF300TS_ES} -startpage {0} -endpage {4} -uskKey {0:0:0:0: 0123456789ABCDEF01234567} –fileName {output.log}

read_usram

Tcl command; reads a uSRAM block from the device.

Physical block

```
read_usram [-name block_name] -fileName file_name
```

Arguments

`-name block_name`

Specifies the name for the target block.

`-fileName file_name`

Optional; specifies the output file name for the data read from the device.

Exceptions

- Array must be programmed and active
- Security locks may disable this function

Example

Reads the uSRAM Block Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x12_IP from the PolarFire device and writes it to the file sram_block_output.txt.

```
read_usram -name {Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x12_IP} –
fileName {output.txt}
```

Logical block

```
read_usram -logicalBlockName block_name -port port_name
```

Arguments

`-logicalBlockName block_name`

Specifies the name of the user defined memory block.

`-port port_name`

Specifies the port of the memory block selected. Can be either Port A or Port B.

Example

```
read_usram -logicalBlockName {Fabric_Logic_0/U3/F_0_F0_U1} -port {Port A}
```

remove_from_probe_group

Tcl command; removes the specified probe points from the group. That is, the removed probe points won't be associated with any probe group.

Note: Probes cannot be removed from the bus.

```
remove_from_probe_group -name probe_name
```

Arguments

`-name` *`probe_name`*

Specifies one or more probe points to remove from the probe group.

Example

The following command removes two probes from my_grp2.

```
Move_out_of_probe_group -name my_grp2.out[3]:out[3]:Q \
                        -name my_grp2.out[3]:out[3]:Q
```

remove_probe_insertion_point

This Tcl command deletes an added probe from the probe insertion UI.

```
remove_probe_insertion_point –net net_name -driver driver
```

Arguments

`-net` *`net_name`*

Name of the existing net which is added using the `add_probe_insertion_point` command.

`-driver` *`driver`*

Driver of the net.

Example

```
remove_probe_insertion_point -net {count_out_c[0]} -driver
{Counter_8bit_0_count_out[0]:Q}
```

run_selected_actions

This Tcl command is used to run the selected action for a device.

```
run_selected_actions
```

This command takes no arguments.

Example

```
set_programming_action -name {MPF300(T_ES|TS_ES)} -action {DEVICE_INFO}
set_programming_action -name {M2S/M2GL090(T|TS|TV)} -action {ERASE}
```

See Also

construct_chain_automatically

scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_file
set_programming_action

save_active_probe_list

Tcl command; saves the list of active probes to a file.

```
save_active_probe_list -file file_path
```

Arguments

-file *file_path*

The output file location.

Example

```
save_active_probe_list -file "./my_probes.txt"
```

scan_chain_prg

In single mode, this Tcl command runs scan chain on a programmer. In chain mode, this Tcl command runs scan and check chain on a programmer if devices have been added in the grid.

```
scan_chain_prg -name {programmer_name}
```

Arguments

-name

Specify the device (programmer) name. This argument is mandatory.

Example

```
scan_chain_prg -name {21428}
```

See Also

construct_chain_automatically
enable_device
set_debug_programmer
set_device_name.htm
set_programming_file
set_programming_action
run_selected_actions

select_active_probe

Tcl command; manages the current selection of active probe points to be used by active probe READ operations. This command extends or replaces your current selection with the probe points found using the search pattern.

```
select_active_probe [-deviceName device_name] [–name probe_name_pattern] [-reset true|false]
```

Arguments

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration..

-name *probe_name_pattern*

Specifies the name of the probe. Optionally, search pattern string can specify one or multiple probe points. The pattern search characters "*" and "?" also can be specified to filter out the probe names.

-reset *true | false*

Optional parameter; resets all previously selected probe points. If name is not specified, empties out current selection.

Example

The following command selects three probes. In the below example, "grp1" is a group and "out" is a bus..

```
Select_active_probe -name out[5]:out[5]:Q
Select_active_probe -name out.out[1]:out[1]:Q \
                   -name out.out[3]:out[3]:Q \
                   -name out.out[5]:out[5]:Q
```

set_debug_programmer

This Tcl command is used to set the debug programmer.

```
set_debug_programmer -name {programmer_name}
```

Arguments

-name

Specify the programmer. This argument is mandatory.

Example

```
set_debug_programmer -name {S201YQST1V}
```

See Also

construct_chain_automatically

scan_chain_prg

enable_device

set_device_name

set_programming_file

set_programming_action

run_selected_actions

set_device_name

Tcl command that is used to set the device name.

```
set_device_name -name {device_name} -new_name {new_name}
```

Arguments

-name

Specify the device name. This argument is mandatory.

-new_name

Specify the new name for the device. This argument is mandatory.

Example

```
set_device_name -name {MPF300(T_ES|TS_ES)} -new_name {Polarfire}
```

See Also

construct_chain_automatically

scan_chain_prg

enable_device

set_debug_programmer

set_programming_file

set_programming_action

run_selected_actions

set_live_probe

Tcl command; set_live_probe channels A and/or B to the specified probe point(s). At least one probe point must be specified. Only exact probe name is allowed (i.e. no search pattern that may return multiple points).

```
set_live_probe [-deviceName device_name] [–probeA probe_name] [–probeB probe_name]
```

Arguments

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug .

-probeA *probe_ name*

Specifies target probe point for the probe channel A.

-probeB *probe_ name*

Specifies target probe point for the probe channel B.

Exceptions

- The array must be programmed and active
- Active probe read or write operation will affect current settings of Live probe since they use same probe circuitry inside the device
- Setting only one Live probe channel affects the other one, so if both channels need to be set, they must be set from the same call to set_live_probe
- Security locks may disable this function
- In order to be available for Live probe, ProbeA and ProbeB I/O's must be reserved for Live probe respectively

Example

Sets the Live probe channel A to the probe point A12 on device MPF300TS_ES.

```
set_live_probe [-deviceName MPF300TS_ES] [–probeA A12]
```

set_programming_action

This Tcl command is used to select the action for a device.

```
set_programming_action [-name {device_name}] -action {procedure_action}
```

Arguments

-name

Specify the device name. This argument is mandatory.

-action

Specify the programming action. This argument is mandatory.

Example

```
set_programming_action -name {MPF300(T_ES|TS_ES)} -action {DEVICE_INFO}
set_programming_action -name {M2S/M2GL090(T|TS|TV)} -action {ERASE}
```

See Also

construct_chain_automatically

scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_file

run_selected_actions

set_programming_file

This Tcl command is used to set the programming file for a device. Either the file or the no_file flag must specified. A programming file must be loaded. The device must be a Microsemi device.

```
set_programming_file –name {device_name} –file {stapl_file_name_with_path}
```

Arguments

-name

Specify the device name. This argument is mandatory.

-file

Specify the *file* path. This argument is mandatory.

Example

```
set_programming_file -name {MPF300(T_ES|TS_ES)} -file
{D:/export/CM1_PCIE_TOP_default_uic_12_200_0_12.stp}
```

See Also

construct_chain_automatically

scan_chain_prg

enable_device

set_debug_programmer

set_device_name

set_programming_action

run_selected_actions

smartbert_test

This Tcl command is used for the following:

- Start a Smart BERT test
- Stop a Smart BERT test
- Reset error count

smartbert_test -start

This Tcl command starts a Smart BERT test with a specified pattern on a specified lane.

```
smartbert_test -start -pattern {pattern_type} -lane {Physical_Location}
```

Arguments

`-start`

Start the Smart BERT test.

`pattern {pattern_type}`

Specify the pattern type of the Smart BERT test.

`-lane{Physical_Location}`

Specify the physical location of the lane.

`-EQ-NearEndLoopback`

Enable EQ-Near End Loopback on specified lane.

Examples

```
smartbert_test -start -pattern {prbs9} -lane {Q0_LANE3}
smartbert_test -start -pattern {prbs23} -lane {Q3_LANE2}
smartbert_test -start -pattern {prbs7} -lane {Q3_LANE1}
smartbert_test -start -pattern {prbs31} -lane {Q1_LANE2} -EQ-NearEndLoopback
smartbert_test -start -pattern {prbs9} -lane {Q2_LANE2} -EQ-NearEndLoopback
smartbert_test -start -pattern {prbs15} -lane {Q2_LANE3} -EQ-NearEndLoopback
```

smartbert_test -stop

This Tcl command stops a Smart BERT test on a specified lane.

```
smartbert_test -stop -lane {Physical_Location}
```

Arguments

`-stop`

Stop the smart BERT test.

`-lane {Physical_Location}`

Specify the physical location of the lane.

Examples

```
smartbert_test -stop -lane {Q0_LANE0}
smartbert_test -stop -lane {Q0_LANE3}
smartbert_test -stop -lane {Q3_LANE2}
smartbert_test -stop -lane {Q3_LANE1}
smartbert_test -stop -lane {Q1_LANE2}
smartbert_test -stop -lane {Q2_LANE2}
smartbert_test -stop -lane {Q2_LANE3}
```

smartbert_test -reset_counter

This Tcl command resets a lane error counter.

```
smartbert_test -reset_counter -lane {Physical_Location}
```

Arguments

-reset_counter

Reset lane error counter on hardware and cumulative error count on the UI.

-lane {*Physical_Location*}

Specify the physical location of the lane.

Examples

```
smartbert_test -reset_counter -lane {Q0_LANE0}
smartbert_test -reset_counter -lane {Q3_LANE2}
smartbert_test -reset_counter -lane {Q2_LANE3}
smartbert_test -reset_counter -lane {Q2_LANE2}
smartbert_test -reset_counter -lane {Q1_LANE2}
smartbert_test -reset_counter -lane {Q3_LANE1}
```

static_pattern_transmit

This Tcl command starts and stops a Static Pattern Transmit.

static_pattern_transmit -start

```
static_pattern_transmit -start -lane {Physical_Location} -pattern {pattern_type} -value
{user_pattern_value}
```

Parameters

-start

Start the Static Pattern Transmit.

-lane {*Physical_Location*}

Specify physical location of lane.

-pattern {*pattern_type*}

Specify pattern_type of Static Pattern Transmit.

-value {*user_pattern_value*

Specify user_pattern_value in hex if pattern_type selected is custom.

Examples

```
static_pattern_transmit -start -lane {Q0_LANE0} -pattern {fixed}
static_pattern_transmit -start -lane {Q0_LANE2} -pattern {maxrunlength} -value {}
static_pattern_transmit -start -lane {Q3_LANE2} -pattern {custom} -value {df}
static_pattern_transmit -start -lane {Q3_LANE0} -pattern {fixed} -value {}
static_pattern_transmit -start -lane {Q1_LANE1} -pattern {custom} -value {4578}
static_pattern_transmit -start -lane {Q1_LANE2} -pattern {fixed} -value {}
static_pattern_transmit -start -lane {Q2_LANE2} -pattern {maxrunlength} -value {}
static_pattern_transmit -start -lane {Q2_LANE1} -pattern {custom} -value {abcdef56}
```

static_pattern_transmit -stop

```
static_pattern_transmit -stop -lane {Physical_Location}
```

Parameters

`-stop`

Stop the Static Pattern Transmit.

`-lane {Physical_Location}`

Specify physical location of lane.

Examples

```
static_pattern_transmit -stop -lane {Q0_LANE0}
static_pattern_transmit -stop -lane {Q0_LANE2}
static_pattern_transmit -stop -lane {Q3_LANE2}
static_pattern_transmit -stop -lane {Q3_LANE0}
static_pattern_transmit -stop -lane {Q1_LANE1}
static_pattern_transmit -stop -lane {Q1_LANE2}
static_pattern_transmit -stop -lane {Q2_LANE2}
static_pattern_transmit -stop -lane {Q2_LANE1}
```

ungroup

Tcl command; disassociates the probes as a group.

```
nngroup -name group_name
```

Arguments

`-name group_name`

Name of the group.

Example

```
ungroup –name my_grp4
```

unset_live_probe

Tcl command; discontinues the debug function and clears live probe A, live probe B, or both probes (Channel A/Channel B). An all zeros value is shown in the oscilloscope.

```
unset_live_probe -probeA 1 -probeB 1  [-deviceName device_name]
```

Arguments

`-probeA`

Live probe Channel A.

`-probeB`

Live probe Channel B.

`-deviceName device_name`

Parameter is optional if only one device is available in the current configuration or set for debug (see the SmartDebug User's Guide for details).

Exceptions

- The array must be programmed and active.
- Active probe read or write operation affects current of Live Probe settings, because they use the same probe circuitry inside the device.
- Security locks may disable this function.

Example

The following example unsets live probe Channel A from the device MPF300TS_ES.

```
unset_live_probe -probeA 1[-deviceName MPF300TS_ES]
```

uprom_read_memory

This Tcl command reads a uPROM memory block from the device.

```
read_uprom_memory –startAddress {hex_value} –words {integer_value}
```

Arguments

`-startAddress hex_value`

Specifies the start address of the uPROM memory block.

`-words integer_value`

Specifies the number of 9-bit words.

Example

```
read_uprom_memory –startAddress {0xA} –words {100}
```

write_active_probe

Tcl command; sets the target probe point on the device to the specified value. The target probe point name must be specified.

```
write_active_probe [-deviceName device_name] –name probe_name -value true|false
-group_name group_bus_name -group_value "hex-value" | "binary-value"
```

Arguments

`-deviceName device_name`

Parameter is optional if only one device is available in the current configuration.

`-name probe_name`

Specifies the name for the target probe point. Cannot be a search pattern.

`-value true | false hex-value | binary-value`

Specifies values to be written.

True = High

False = Low

`-group_name group_bus_name`

Specify the group or bus name to write to complete group or bus.

`-group_value "hex-value" | "binary-value"`

Specify the value for the complete group or bus.

Hex-value format : " <size>'h<value>"

Binary-value format: " <size>'b<value>"

Example

write_active_probe –name out[5]:out[5]:Q –value true <-- write to a single probe
write_active_probe -name grp1.out[3]:out[3]:Q -value low <-- write to a probe in the group
write_active_probe -group_name grp1 –group_value "8'hF0" <-- write the value to complete group
write_active_probe –group_name out –group_value "8'b11110000" \
-name out[2]:out[2]:Q –value true <-- write multiple probes at the same time.

write_lsram

Tcl command; writes a word into the specified large SRAM location.

Physical block

```
write_lsram –name block_name] –offset offset_value –value integer_value
```

Arguments

`-name` *`block_name`*

Specifies the name for the target block.

`-offset` *`offset_value`*

Offset (address) of the target word within the memory block.

`-value` *`integer_value`*

Word to be written to the target location. Depending on the configuration of memory blocks, the width can be 1, 2, 5, 10, or 20 bits.

Exceptions

- Array must be programmed and active
- The maximum value that can be written depends on the configuration of memory blocks
- Security locks may disable this function

Example

```
write_lsram -name {Fabric_Logic_0/U2/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM1K20_IP} -offset
0 -value 291
```

Logical block

```
write_lsram -logicalBlockName block_name -port port_name -offset 1 offset_value -logicalValue
hexadecimal_value
```

Arguments

`-logicalBlockName` *`block_name`*

Specifies the name of the user defined memory block.

`-port` *`port_name`*

Specifies the port of the memory block selected. Can be either Port A or Port B.

`-offset` *`offset_value`*

Offset (address) of the target word within the memory block.

`-logicalValue` *`hexadecimal_value`*

Specifies the hexadecimal value to be written to the memory block. Size of the value is equal to the width of the output port selected.

Example

```
write_lsram -logicalBlockName {Fabric_Logic_0/U2/F_0_F0_U1} -port {Port A} -offset 1 -
logicalValue {00FFF}
```

write_usram

Tcl command; writes a 12-bit word into the specified uSRAM location.

Physical block

```
write_usram –name block_name] –offset offset_value –value integer_value
```

Arguments

-name *block_name*

Specifies the name for the target block.

-offset *offset_value*

Offset (address) of the target word within the memory block.

-value *integer_value*

12-bit value to be written.

Exceptions

- Array must be programmed and active
- The maximum value that can be written is 0x1FF
- Security locks may disable this function

Example

Writes a value of 0x291 to the device PolarFire in the Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x12_IP with an offset of 0.

```
write_lsram -name {Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x12_IP} -
offset 0 -value 291
```

Logical block

```
write_usram -logicalBlockName block_name -port port_name -offset offset_value -logicalValue
hexadecimal_value
```

Arguments

-logicalBlockName *block_name*

Specifies the name of the user defined memory block.

-port *port_name*

Specifies the port of the memory block selected. Can be either Port A or Port B.

-offset *offset_value*

Offset (address) of the target word within the memory block.

`-logicalValue hexadecimal_value`

Specifies the hexadecimal value to be written to the memory block. Size of the value is equal to the width of the output port selected.

Example

```
write_usram -logicalBlockName {Fabric_Logic_0/U3/F_0_F0_U1} -port {Port A} -offset 1 -
logicalValue {00FFF}
```

xcvr_read_register

This Tcl command reads SCB registers and their field values. Read value is in hex format. This command is used in SmartDebug Signal Integrity.

```
xcvr_read_register –inst_name <inst_name> –reg_name [<reg_name> | <reg_name:field_name>]
```

Arguments

`-inst_name <inst_name>`

Specify the lane instance name used in the design.

`-reg_name <reg_name> or <reg_name:field_name>`

Specify the <*reg_name*> for register name or <*reg_name*>:<*field_name*> for the register's field.

Examples

Reading pcslane's 32-bit register LNTV_R0:

```
xcvr_read_register -inst_name {CM1_PCIe_SS_0/PF_PCIE_0/LANE1} -reg_name {LNTV_R0}
```

Output:

```
Register Name: LNTV_R0 value: 0x12
The 'xcvr_write_register' command succeeded.
```

Reading Register LNTV_R0 field LNTV_RX_GEAR (i.e. 0th bit of 32-bit register):

```
xcvr_read_register -inst_name {CM1_PCIe_SS_0/PF_PCIE_0/LANE1} -reg_name
{LNTV_R0:LNTV_RX_GEAR}
```

Output:

```
Register Name: LNTV_R0:LNTV_RX_GEAR, Value: 0x0
The 'xcvr_read_register' command succeeded.
```

Exception:

SOFT_RESET Register

The SOFT_RESET register is an SCB read/write register containing information such as block ID and Map IDs. It is also used to provide a pulsed reset to the SCB registers. It is a group-specific register.

The SOFT_RESET register is available with the four groups (`pma_lane`, `pma_cmn`, `pcslane`, and `pcscmn`). To read or write this register or its field value, "group name" must be added before "SOFT_RESET".

`-reg_name <group name>_<SOFT_RESET>` for register name

or

`[<group name>_<SOFT_RESET>:field_name]` for register field name

where <*group name*> can be PCS, PCSCMN, PMA, or PMA_CMN.

Examples

Reading all four groups' SOFT_RESET register and its field BLOCKID

Reading the PCS SOFT_RESET register and its field BLOCKID (i.e. 16th to 31st bit):

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PCS_SOFT_RESET}
```

Output:

```
Register Name: PCS_SOFT_RESET, Value: 0x300100
The 'xcvr_read_register' command succeeded.
```

Reading field BLOCKID:

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCS_SOFT_RESET:BLOCKID}
```

Output:

```
Register Name: PCS_SOFT_RESET:BLOCKID, Value: 0x30
The 'xcvr_read_register' command succeeded.
```

Reading PCSCMN's SOFT_RESET register and its field BLOCKID (i.e. 16th to 31st bit):

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCSCMN_SOFT_RESET}
Register Name: PCSCMN_SOFT_RESET, Value: 0x340100
The 'xcvr_read_register' command succeeded.
```

Reading field BLOCKID:

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCSCMN_SOFT_RESET:BLOCKID}
```

Output:

```
Register Name: PCSCMN_SOFT_RESET:BLOCKID, Value: 0x34
The 'xcvr_read_register' command succeeded.
```

Reading PMA's SOFT_RESET register and its field BLOCKID (i.e. 16th to 31st bit):

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PMA_SOFT_RESET}
```

Output:

```
Register Name: PMA_SOFT_RESET, Value: 0x1300100
The 'xcvr_read_register' command succeeded.
```

Reading field BLOCKID:

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_SOFT_RESET:BLOCKID}
```

Output:

```
Register Name: PMA_SOFT_RESET:BLOCKID, Value: 0x130
The 'xcvr_read_register' command succeeded.
```

Reading PMA_CMN's SOFT_RESET register and it's field BLOCKID (i.e. 16th to 31st bit):

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_CMN_SOFT_RESET}
```

Output:

```
Register Name: PMA_CMN_SOFT_RESET, Value: 0x1340100
The 'xcvr_read_register' command succeeded.
```

Reading field BLOCKID:

```
xcvr_read_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_CMN_SOFT_RESET:BLOCKID}
```

Output:

```
Register Name: PMA_CMN_SOFT_RESET:BLOCKID, Value: 0x134
The 'xcvr_read_register' command succeeded.
```

See Also

xcvr_write_register

xcvr_write_register

This Tcl command writes SCB registers and their field values. Write value is in hex format. This command is used in SmartDebug Signal Integrity.

```
xcvr_write_register –inst_name <inst_name> –reg_name [<reg_name> | <reg_name:field_name>] –
value {write_value}
```

Arguments

–inst_name <*inst_name*>

Specify the lane instance name used in the design.

–reg_name <*reg_name*> *or* <*reg_name*:*field_name*>

Specify the <*reg_name*> for register name or <*reg_name*>:<*field_name*> for the register's field.

-value <*write_value*>

Specify the value in hex format.

Examples

Writing pcscmn's 32-bit register GSSCLK_CTRL

```
xcvr_write_register -inst_name {CM1_PCIe_SS_0/PF_PCIE_0/LANE1} -reg_name {GSSCLK_CTRL} -
value 0xffffffff
```

Output:

```
Register Name: GSSCLK_CTRL value: 0xffffffff
The 'xcvr_write_register' command succeeded.
```

Writing Register GSSCLK_CTRL field MCLK_GSSCLK_2_SEL i.e. 16th to 20th bits (5 bits) of 32-bit register

```
xcvr_write_register -inst_name {CM1_PCIe_SS_0/PF_PCIE_0/LANE1} \
-reg_name {GSSCLK_CTRL:MCLK_GSSCLK_2_SEL} -value 0x6
```

Output:

```
Register Name: GSSCLK_CTRL:MCLK_GSSCLK_2_SEL value: 0x6
The 'xcvr_write_register' command succeeded.
```

Exception:

SOFT_RESET Register

The SOFT_RESET register is an SCB read/write register containing information such as block ID and Map IDs. It is also used to provide a pulsed reset to the SCB registers. It is a group-specific register.

The SOFT_RESET register is available with the four groups (`pma_lane`, `pma_cmn`, `pcslane`, and `pcscmn`). To read or write this register or its field value, “group name” must be added before “SOFT_RESET”.

`-reg_name <group name>_<SOFT_RESET>` for register name

or

`[<group name>_<SOFT_RESET>:field_name]` for register field name

where <*group name*> can be PCS, PCSCMN, PMA, or PMA_CMN

Examples

Writing all four groups' SOFT_RESET register and its field PERIPH

Writing to the PCS SOFT_RESET register (32-bits) and its field PERIPH (i.e. 8th bit):

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PCS_SOFT_RESET}
-value 0xffffffff
```

Output:

```
Register Name: PCS_SOFT_RESET value: 0xffffffff
The 'xcvr_write_register' command succeeded.
```

Writing to field PERIPH:

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCS_SOFT_RESET:PERIPH} -value 0x1
```

Output:

```
Register Name: PCS_SOFT_RESET:PERIPH value: 0x1
The 'xcvr_write_register' command succeeded.
```

Writing to PCSCMN’s SOFT_RESET register (32-bits) its field PERIPH (i.e. 8th bit):

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCSCMN_SOFT_RESET} -value 0xffffffff
```

Output:

```
Register Name: PCSCMN_SOFT_RESET value: 0xffffffff
The 'xcvr_write_register' command succeeded.
```

Writing to field PERIPH:

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PCSCMN_SOFT_RESET:PERIPH} -value 0x1
```

Output:

```
Register Name: PCSCMN_SOFT_RESET:PERIPH value: 0x1
The 'xcvr_write_register' command succeeded.
```

Writing to PMA's SOFT_RESET register its field PERIPH (i.e. 8th bit):

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name {PMA_SOFT_RESET}
-value 0xffffffff
```

Output:

```
Register Name: PMA_SOFT_RESET value: 0xffffffff
The 'xcvr_write_register' command succeeded.
```

Writing to field PERIPH:

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_SOFT_RESET:PERIPH} -value 0x1
```

Output:

```
Register Name: PMA_SOFT_RESET:PERIPH value: 0x1
The 'xcvr_write_register' command succeeded.
```

Writing to PMA_CMN's SOFT_RESET register its field PERIPH (i.e. 8th bit):

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_CMN_SOFT_RESET} -value 0xffffffff
```

Output:

```
Register Name: PMA_CMN_SOFT_RESET value: 0xffffffff
The 'xcvr_write_register' command succeeded.
```

Writing to field PERIPH:

```
xcvr_write_register -inst_name SmartBERT_L4_0/PF_XCVR_0/LANE0 -reg_name
{PMA_CMN_SOFT_RESET:PERIPH} -value 0x1
```

Output:

```
Register Name: PMA_CMN_SOFT_RESET:PERIPH value: 0x1
The 'xcvr_write_register' command succeeded.
```

See Also

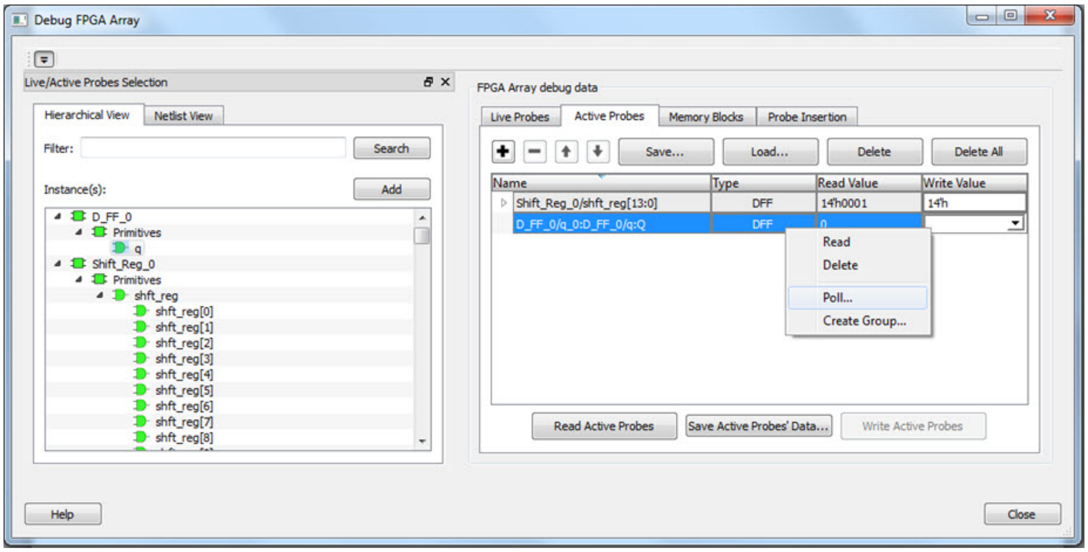
xcvr_read_register

Frequently Asked Questions

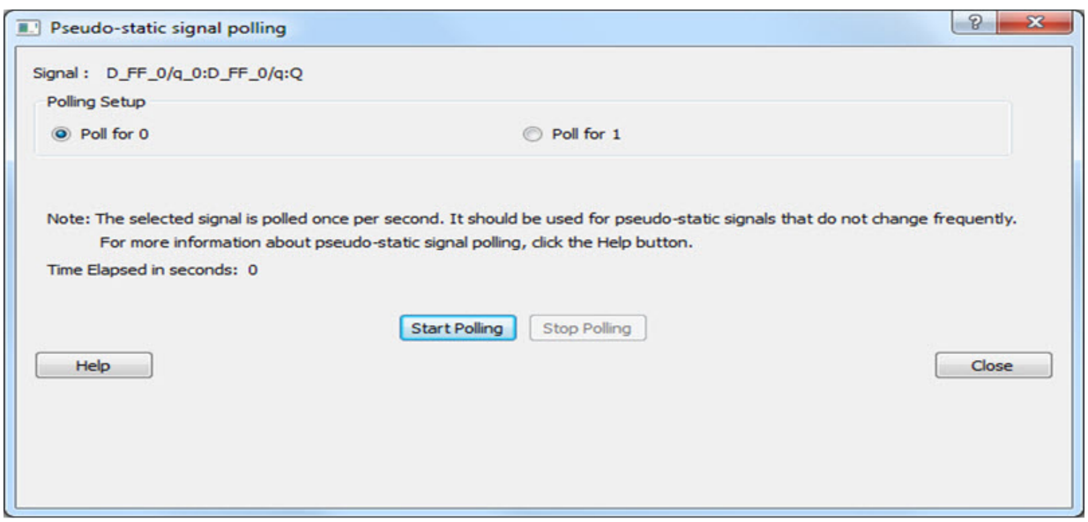
How do I monitor a static or pseudo-static signal?

To monitor a static or pseudo-static signal:

1. Add the signal to the **Active Probes** tab.
2. Select the signal in the **Active Probes** tab, right-click, and choose **Poll...**.



3. In the Pseudo-static Signal Polling dialog box, choose a value in Polling Setup and click **Start Polling**.

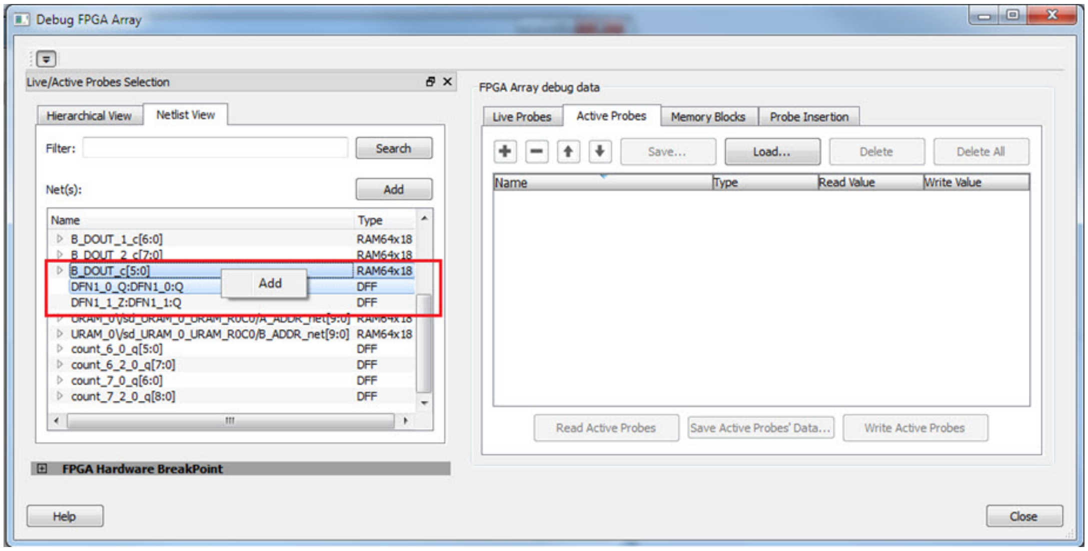


How do I force a signal to a new value?

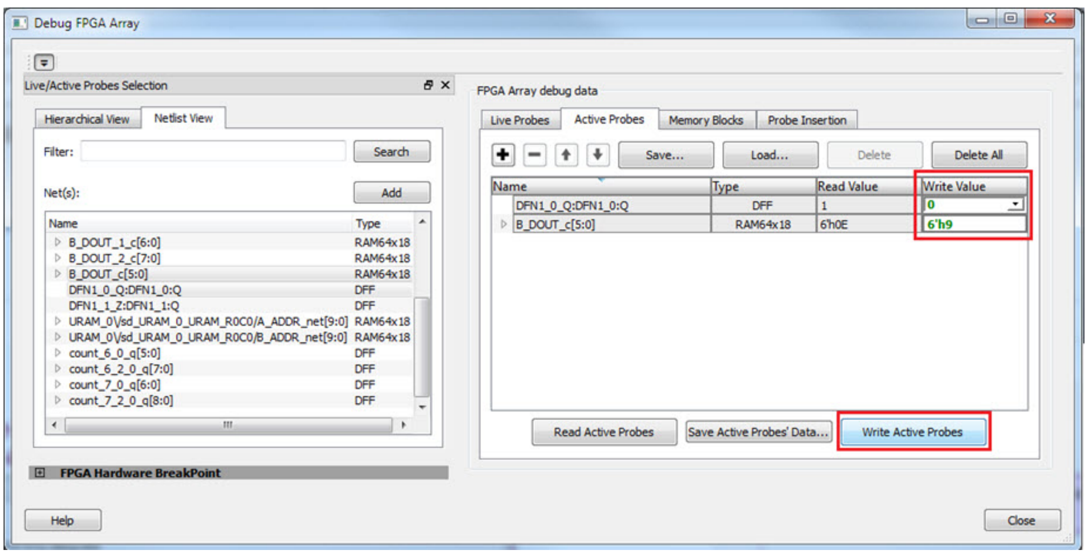
To force a signal to a new value:

1. In the SmartDebug window, click **Debug FPGA Array**.
2. Click the **Active Probes** tab.

3. Select the signal from the selection panel and add it to Active Probes tab.



1. Click **Read Active Probe** to read the value.
2. In the Write Value column, enter the value to write to the signal and then click **Write Active Probes**.



How do I perform simple SmartBERT tests?

You can perform SmartBERT tests using the Debug Transceiver option in SmartDebug.

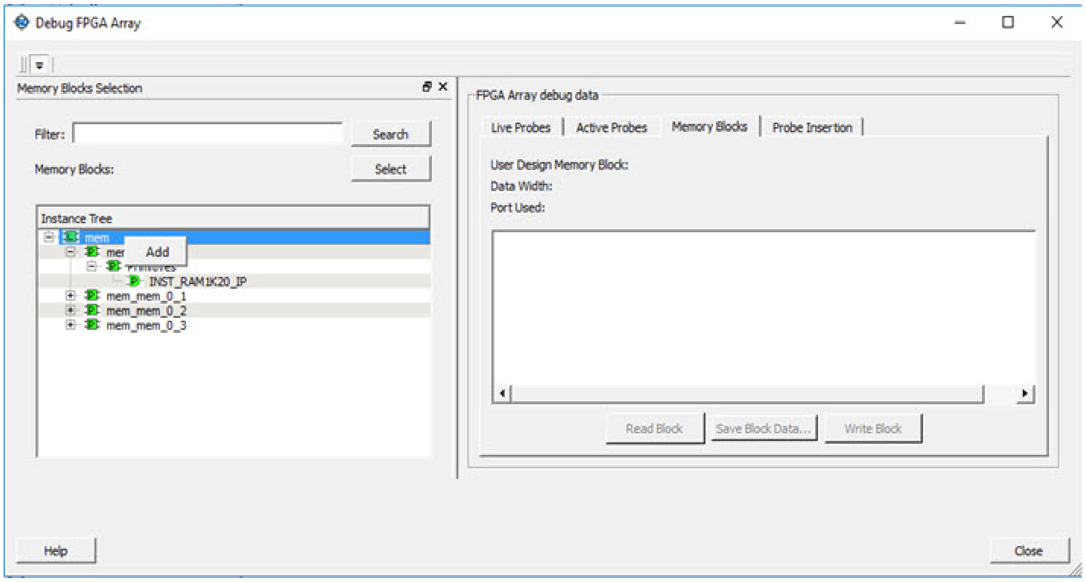
To perform a SmartBERT test, in the SmartBERT page of the Debug Transceiver dialog box, select to run a PRBS test on-die or off-die with EQ-NEAREND checked or unchecked. For more information, see "SmartBERT" on page 39 .

To perform a SmartBERT test, in the Smart BERT page of the Debug Transceiver dialog box, select your options and click **Start** to run a Smart BERT test on-die or off-die with EQ-NEAREND checked or unchecked. For more information, see "SmartBERT" on page 39.

How do I read LSRAM or USRAM content?

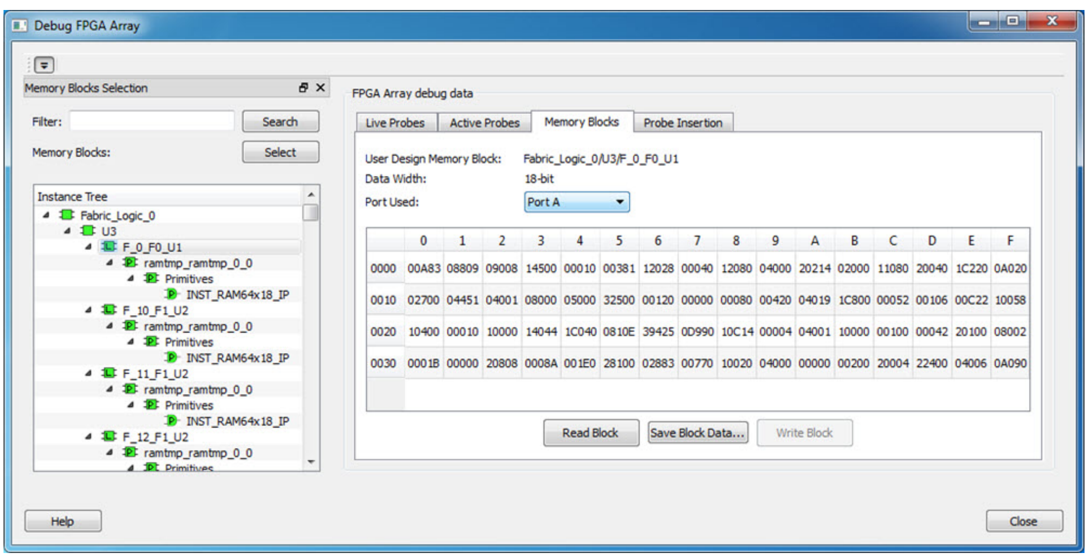
To read RAM content:

1. In the Debug FPGA Array dialog box, click the **Memory Blocks** tab.
2. Select the memory block to be read from the selection panel on the left of the window.



An "L" in the icon next to the block name indicates that it is a logical block, and a "P" in the icon indicates that it is a physical block. A logical block displays three fields in the Memory Blocks tab: User Design Memory Blocks, Data Width, and Port Used. A physical block displays two fields in the Memory Blocks tab: User Design Memory Block and Data Width.

3. Add the block in one of the following ways:
   a. Click **Select**.
   b. Right-click and choose **Add**.
   c. Drag the block to the **Memory Blocks** tab.
4. Click **Read Block** to read the content of the block.



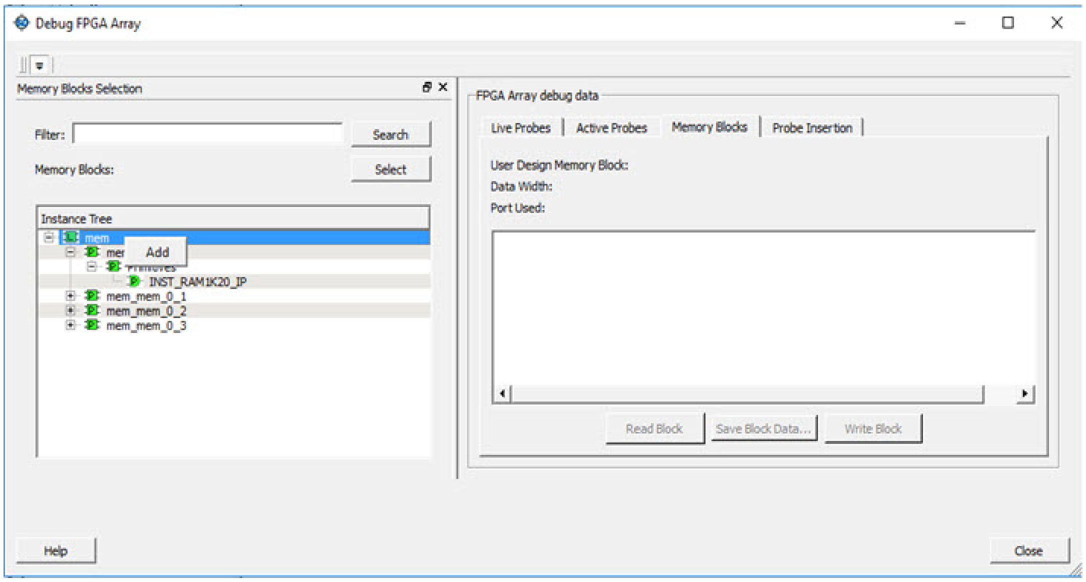
See Also

"Memory Blocks " on page 24

How do I change the content of LSRAM or USRAM?

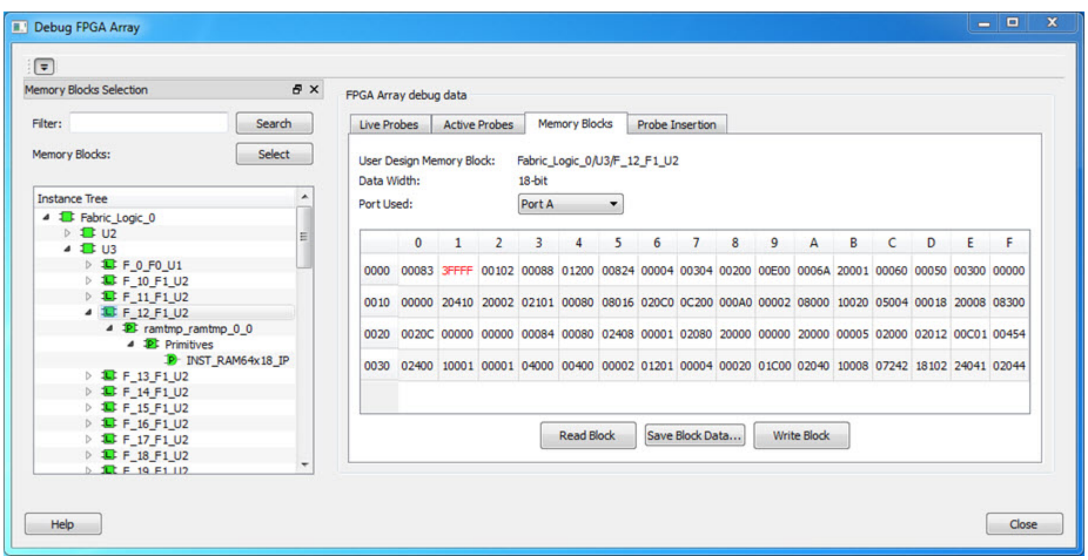
To change the content of LSRAM or USRAM:

1. In the SmartDebug window, click **Debug FPGA Array**.
2. Click the **Memory Blocks** tab.
3. Select the memory block from the selection panel on the left of the window.



An "L" in the icon next to the block name indicates that it is a logical block, and a "P" in the icon indicates that it is a physical block. A logical block displays three fields in the Memory Blocks tab: User Design Memory Blocks, Data Width, and Port Used. A physical block displays two fields in the Memory Blocks tab: User Design Memory Block and Data Width.

4. Add the memory block in one of the following ways:
   a. Click **Select**.
   b. Right-click and choose **Add**.
   c. Drag the block to the **Memory Blocks** tab.
5. Click **Read Block**. The memory content matrix is displayed.
6. Select the memory cell value that you want to change and update the value.
7. Click **Write Block** to write to the device.



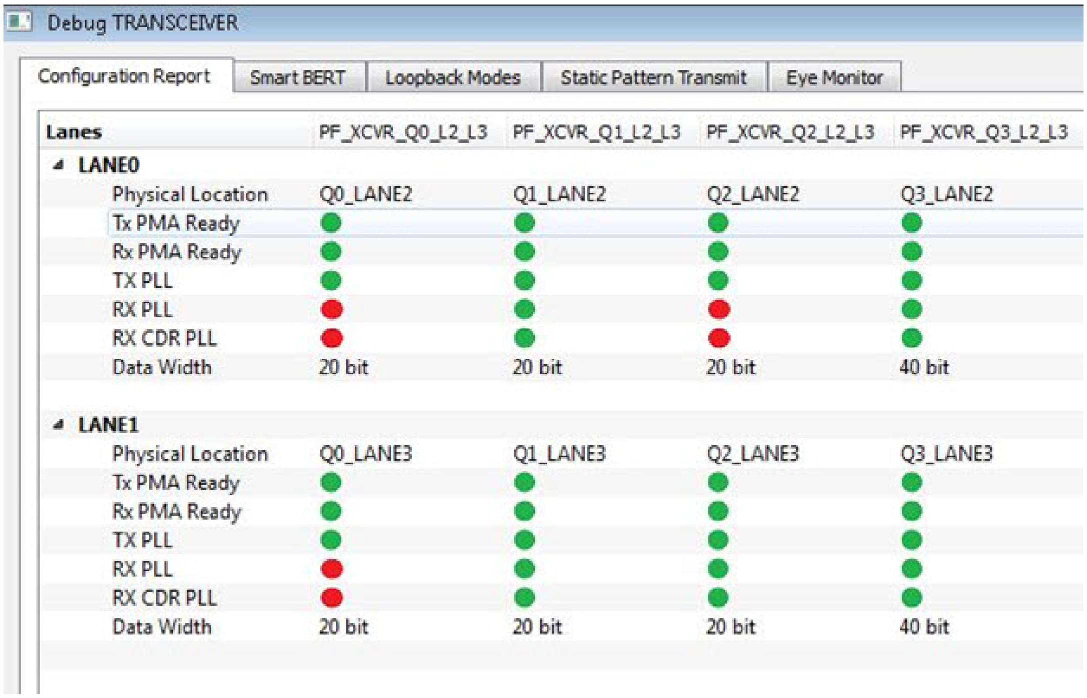
See Also

"Memory Blocks " on page 24

How do I read the health check of the Transceiver?

You can read the transceiver health check using the following Debug Transceiver options:

1. Review the **Configuration Report**, which returns Tx PMA Ready, Rx PMA Ready, TxPLL status, and RxPLL status. For the transceiver to function correctly, all four should be green. The Configuration Report can be found in the Debug TRANSCEIVER dialog box under Configuration Report. See "Debug Transceiver" on page 35.



2. Run the SmartBERT Test, with EQ-NEAR END checked or with external loopback connection from Tx to Rx on selected lanes. This should result in 0 errors in the Cumulative Error Count column. See "SmartBERT" on page 39.