UG0743
User Guide
PolarFire FPGA Debugging

Microsemi
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.

50200743. 4.0 3/18

Contents

Figures

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the current publication.

1.1 Revision 4.0

This version of the document is the update with respect to Libero® SoC PolarFire v2.1 release.

1.2 Revision 3.0

The following is a summary of the changes made in revision 3.0 of this document:

- Added the Demo Mode section (see Demo Mode, page 32).
- Updated the Eye Monitor section as part of the Debug TRANSCEIVER section (see Eye Monitor, page 19).
- Updated the sNVM Debug section for its features (see sNVM Debug, page 27).
- Updated the Signal Integrity sections for its features (see Signal Integrity, page 20).
- Updated the SmartBERT section for its features (see SmartBERT, page 15).

1.3 Revision 2.0

The following was a summary of the changes made in revision 2.0 of this document.

- Added additional information about SmartDebug. For more information, see SmartDebug, page 2.
- Updated the view device status section. For more information, see View Device Status, page 4.
- Updated the debug FPGA array section. For more information, see Debug FPGA Array, page 5.
- Updated the debug µPROM section. For more information, see Debug µPROM, page 12.
- Updated the debug transceiver section. For more information, see Debug TRANSCEIVER, page 14.

1.4 Revision 1.0

Revision 1.0 was the first publication of this document.

2 PolarFire FPGA Debugging

Microsemi PolarFire® devices support the following on-chip debug capabilities:

- Built-in dedicated probe interface in the fabric (hard block)
- Embedded logic analyzer (soft block)

Microsemi's SmartDebug tool is integrated in the Libero® SoC PolarFire Design Suite. SmartDebug enables hardware debugging using the in-built dedicated signal probe. It offers the following advantages:

- Enables live on-chip debugging
- Avoids extra FPGA resource utilization
- Reduces the FPGA design debug cycles
- Enables read-write capability to logic elements and memory blocks during debug

The Identify debug tool integrated in to Libero SoC PolarFire, enables hardware debugging using the embedded logic analyzer.

In addition to SmartDebug and Identify, external test equipments such as Oscilloscopes and Logic Analyzer are also supported. For functional debugging of the FPGA design, Libero SoC PolarFire also integrates the ModelSim simulator.

SmartDebug, Identify, and ModelSim are described in the following sections. For information about debugging examples, see Debugging Examples, page 45.

2.1 SmartDebug

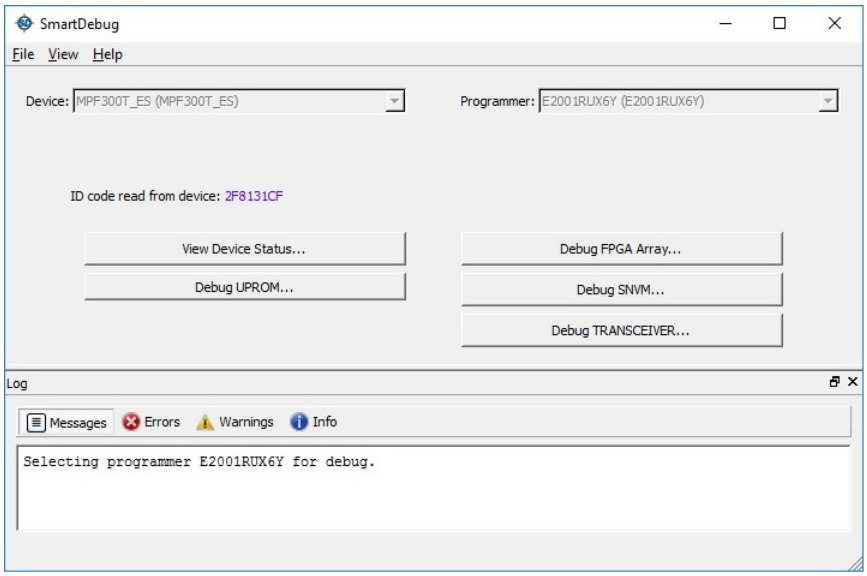
Design debugging is a critical phase of the FPGA design flow. SmartDebug enables you to debug the design by allowing verification and troubleshooting at the hardware level. It provides access to probe points, non-volatile memory (NVM), fabric and memory, transceivers, and the DDR controller.

SmartDebug uses the JTAG interface to retrieve information from the fabric probe points via the fabric control bus. As SmartDebug does not require any fabric logic or change in the design flow, it runs in two modes–standalone and integrated with Libero SoC PolarFire.

2.1.1 Standalone Mode

SmartDebug can also be installed separately with program and debug installer. This setup provides a lean installation that configures all of the programming and debugging tools in a lab environment for debugging. In this mode, SmartDebug opens as a separate tool. The debug process is invoked through SmartDebug after programming the FPGA with the programming file. When SmartDebug is invoked in standalone mode, create a new project and import the design debug data container file (`.ddc` file that needs to be exported from Libero SoC PolarFire) to access all of the debug features.

Figure 1 • **SmartDebug—Standalone Mode**

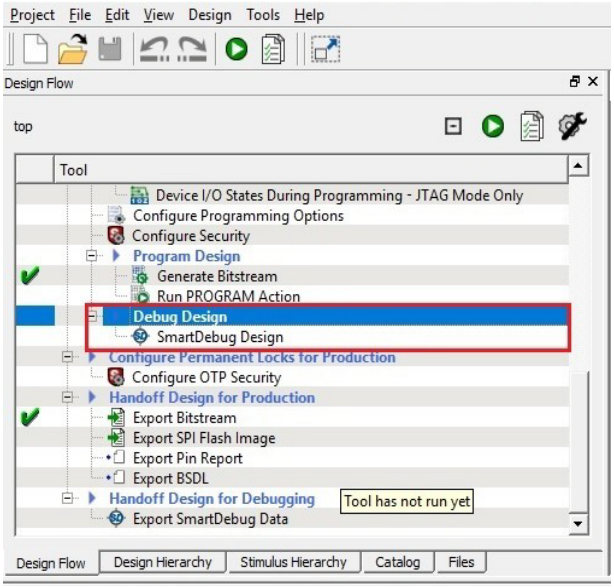


Note: In standalone mode, the probe insertion feature is not available for FPGA Array Debug, as this feature needs incremental place and route to connect the user net to the specified I/O.

2.1.2 Integrated Mode from the Libero SoC PolarFire Design Flow

SmartDebug has access to all design files that allow you to debug the device. Certain I/O states during programming, and JTAG clock frequency, however, cannot be changed through SmartDebug. To invoke SmartDebug in the Integrated mode, expand **Debug Design** and double-click **SmartDebug Design**.

Figure 2 • **SmartDebug–Integrated Mode**



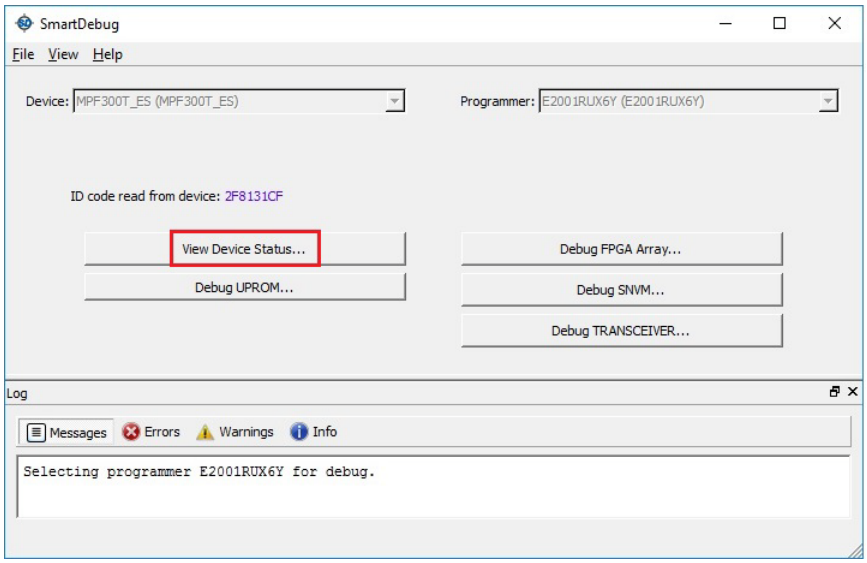
The following sections describe SmartDebug features, which include:

2.1.3 View Device Status

The **View Device Status** feature provides the device status report.

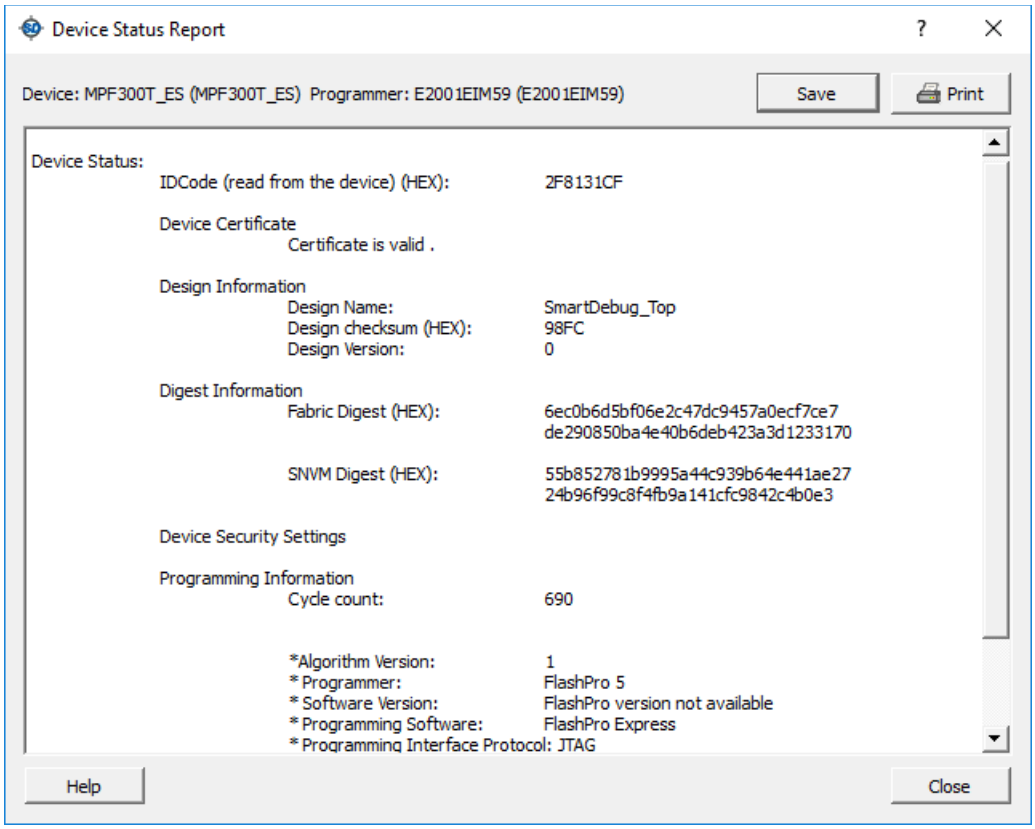
To view the device status report, click **View Device Status** in the **SmartDebug** window.

Figure 3 • **Viewing Device Status**



The device status report is displayed. It is a complete summary of ID Code, device certificate, design information, programming information, digest, and device security information, as shown in the following figure.

Figure 4 • **Viewing Device Status Report**



Device certificate—displays that certificate is valid if device certificate is installed on a device. If the device certificate is not installed, a message is displayed, stating that the device certificate needs to be installed to generate the device certificate related information.

2.1.4 Debug FPGA Array

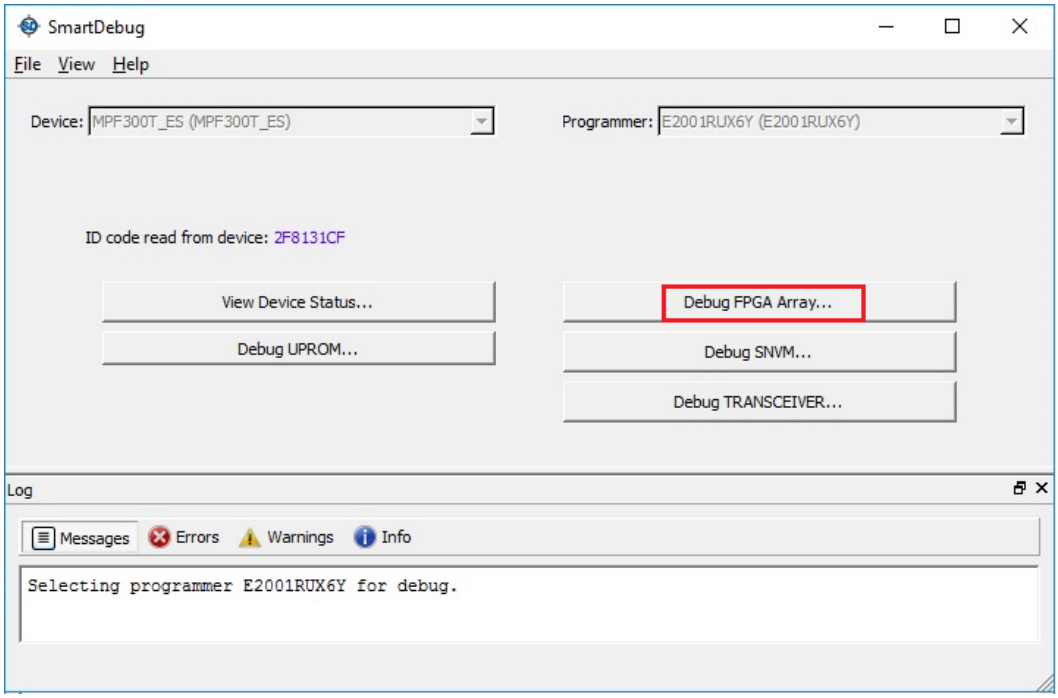
The Debug FPGA array provides an interface to probe the user logic implemented in the logic elements (LEs) of the FPGA using active and live probes, read-write access to the fabric flip-flops, and read-write access to the memories implemented using LSRAMs/µRAMs.

Probe insertion allows assignment of the internal signals to the assigned or unassigned pins. These signals can be monitored using the oscilloscope in real-time.

The Debug FPGA array supports the following four features:

- Active Probes, page 6
- Live Probes, page 8
- Memory Blocks, page 10
- Probe Insertion, page 11

Figure 5 • **Debug FPGA Array**



2.1.4.1 Active Probes

Active probes enable you to read or change the values of probe points in a design through JTAG.

The value of probe points maybe changed for various reasons, such as:

- To verify that a reset signal is in the active and required state.
- To test a logic function by writing to a probe point.
- To initiate a state machine transition by quickly setting an input value to isolate a control flow problem.

Active probes dynamically and asynchronously read or write to any logic element register bit. The probe points of a design are selected using active probes. Active probes are particularly useful for a quick observation of an internal signal. All of the probe points for the design are displayed in **Hierarchical View** and **Netlist View** in the left pane of the Active Probe window.

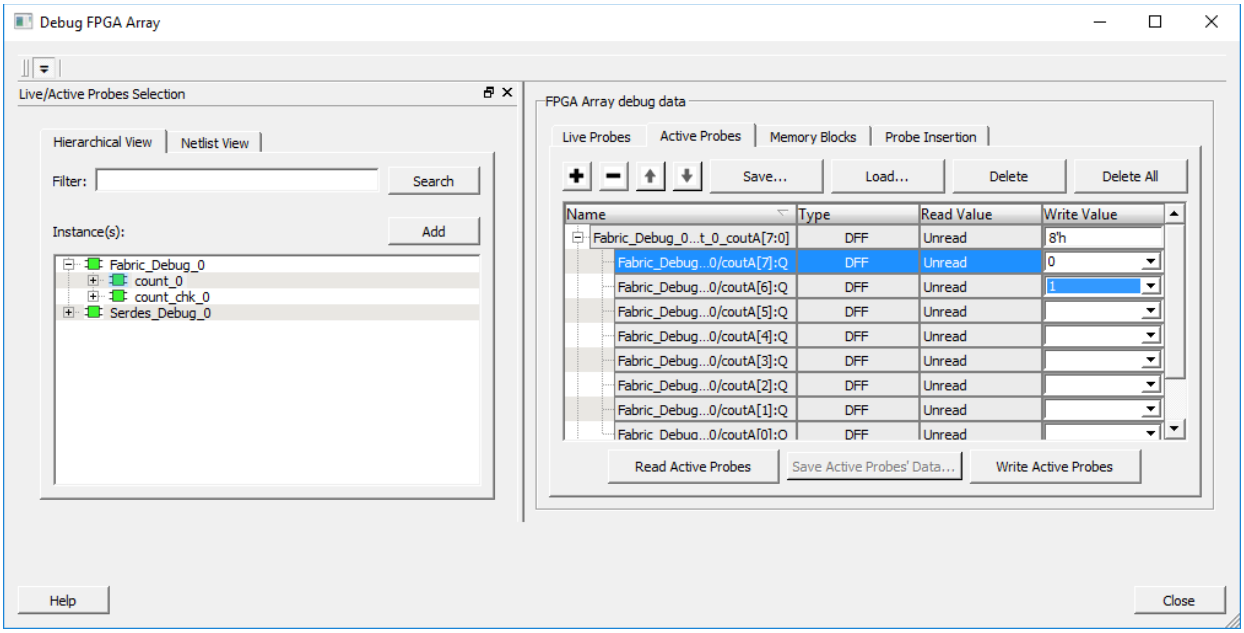
- **Hierarchical View**—available probe points are listed in hierarchical order.
- **Netlist View**—available probe points are listed with the Name and Type, which are physical locations of flip-flops.

To add probe points to a list:

1. Select the **Active Probes** tab in the right pane. The probe signals are displayed in the left pane.
2. Select the probe points that you want to add from the **Hierarchical View** or **Netlist View** in the left pane.

3. Right-click the selected points and click **Add** to add them to the **Active Probes**. You can also add the selected probe points by clicking **Add** in the top-right corner of the left pane. The probes signals can be filtered with the **Filter** option.

Figure 6 • **Adding Probe Points**



The added probe points appear in the active probe data chart and you can read or write multiple probe points at a time.

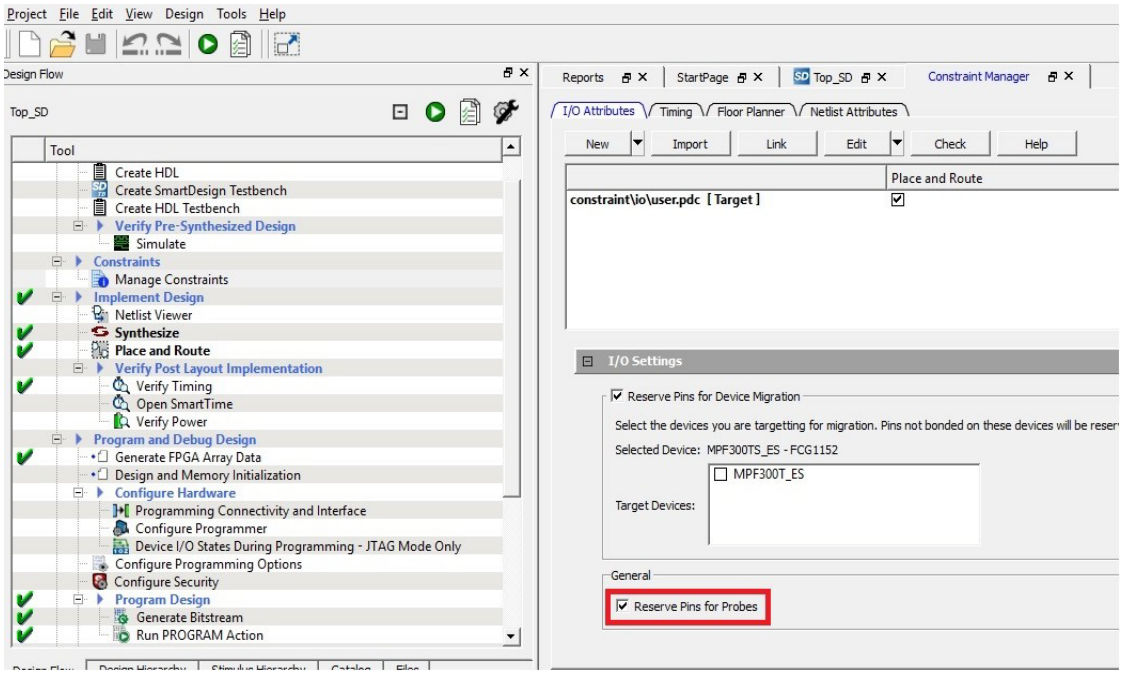
Note: All of the registered locations of a device are not accessible for writing new probe values due to a silicon limitation. This will be fixed in the next release of the PolarFire silicon.

2.1.4.2 Live Probes

Live probes enable the monitoring of two internal signals at a time in the design without having to rerun place and route.

PolarFire devices have two dedicated live probe channels (for example, pin H6 and G6 of PolarFire MPF300TS device). To use live probes, reserve pins, using **Reserve Pins for Probes** under **Constraints Manager** in the Libero SoC PolarFire. If you do not reserve pins for live probes, the live probe I/O's function as GPIOs and are used for routing nets in the design.

Figure 7 • **Reserve Pins for Live Probes**



Any probe point from the design can be routed to one of these channels without having to re-run place and route. The probe points assigned to live probe channels can be modified through the SmartDebug live probes **Assign** and **Unassign** options without having to recompile and reprogram the design.

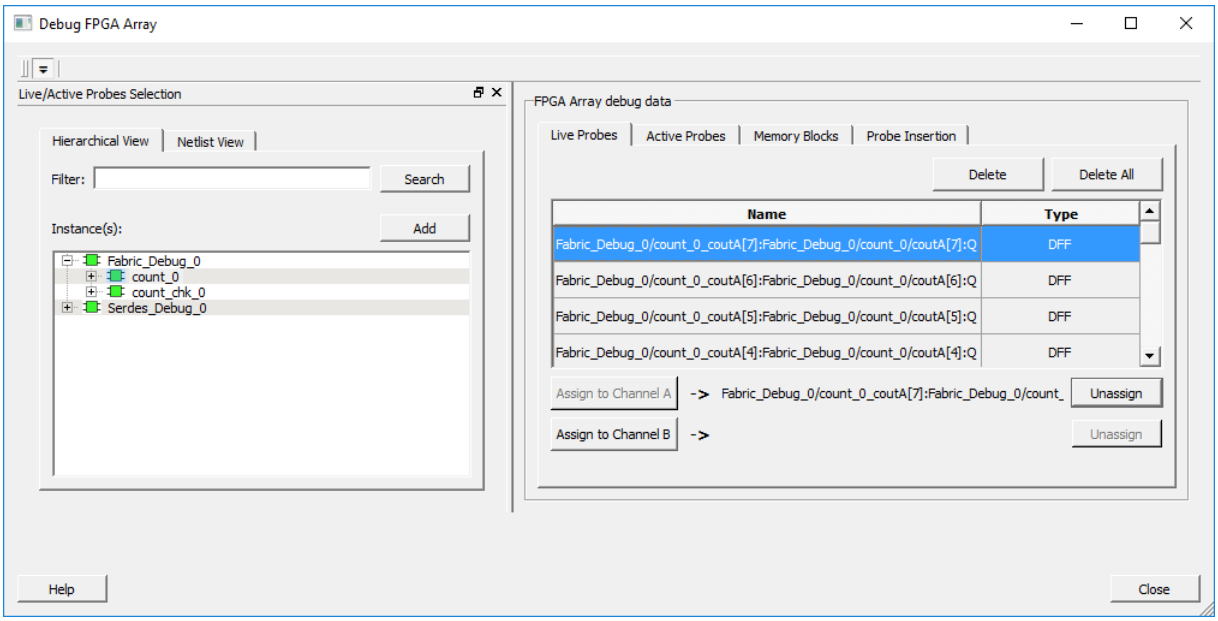
Channel A and Channel B are available in live probes. When an internal signal is selected, it can be assigned to either, Channel A or Channel B.

To assign and unassign probe points to Channel A or Channel B:

1. Select the required probe points in the left pane and click **Add** in the top-right corner. The signals are displayed in the right pane.
2. Click the signal to be monitored and click **Assign to Channel A** or **Assign to Channel B**. The signal is assigned to the selected channel. When the assignment is complete, the probe name appears next to channels. SmartDebug configures Channel A and Channel B I/O to monitor the desired probe points.

3. Click **Unassign Channels** to disconnect the selected internal net from the live probe channel that appears in the bottom-right corner of the live probes window.

Figure 8 • **Assign Live Probes**



2.1.4.3 Memory Blocks

SmartDebug provides the **Memory Blocks** tab to dynamically and asynchronously read from and write to a selected FPGA fabric SRAM block. Memory blocks are categorized into two views:

- Physical View—shows the actual memory view of the RAM in FPGA.
- Logical View—shows a logical representation of RAM block.

Using the **Memory Blocks** tab, you can select the required memory block to:

- Read
- Capture a snapshot of the memory
- Modify memory values, and then write the values back to that block.

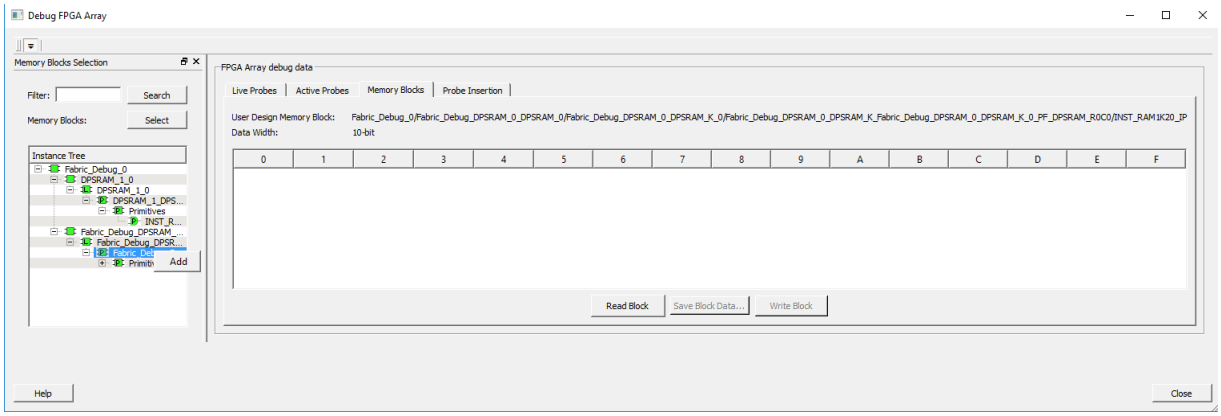
This feature is useful for checking or setting data buffers used in communication interfaces; debugging complex data dependent errors.

Note: For RAM blocks used in the design through RTL inference, logical representation of the memory blocks may not be available.

To read and write memory blocks:

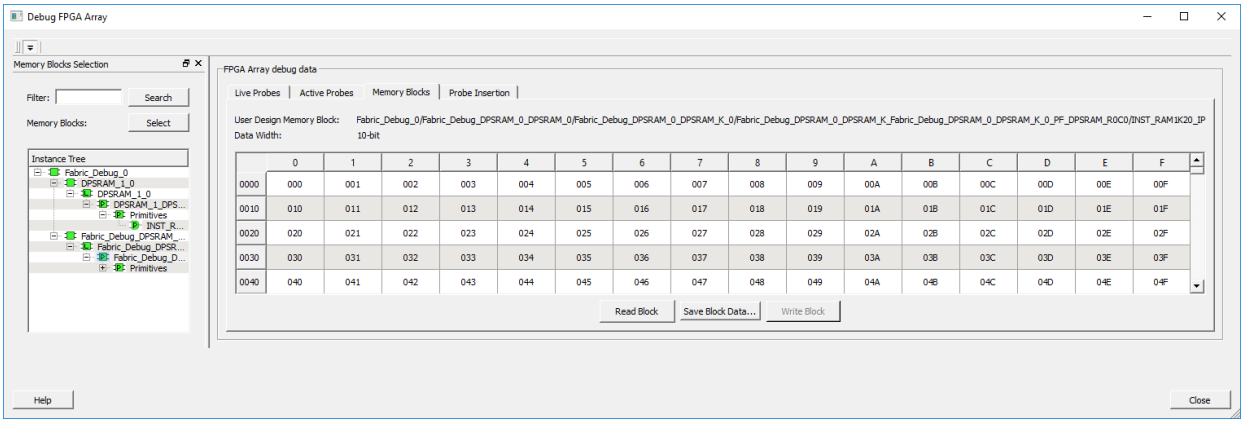
1. Select the **Memory Blocks** tab in the right pane of the **SmartDebug** window.
2. View the memory blocks in the left pane in the **Hierarchical View**.
3. Select the memory block in the left pane and click **Select** in the top-right corner of the pane.
4. Right-click the selected memory block and click **Add**.

Figure 9 • **Add Memory Block**



The memory blocks are displayed in the right pane.

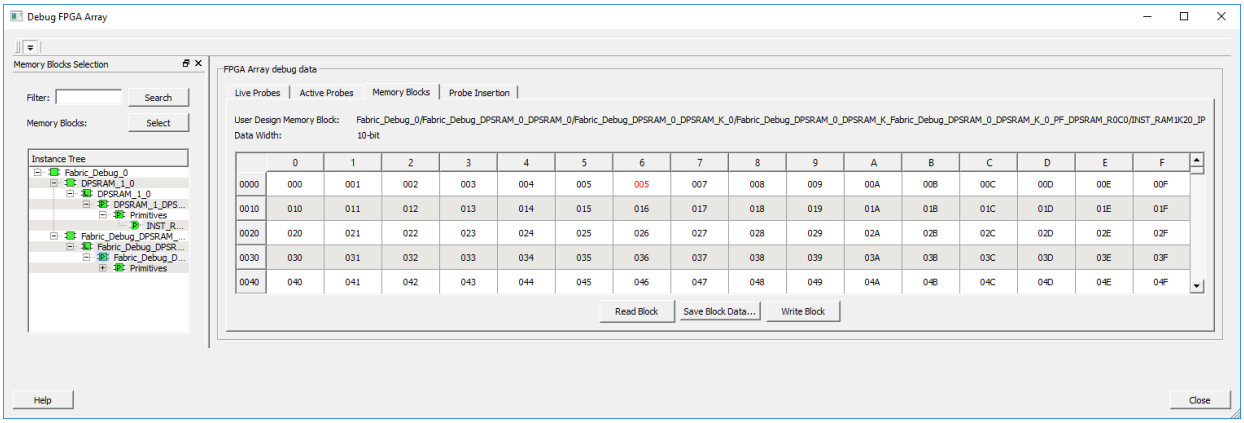
Figure 10 • **View Memory Blocks**



5. Click **Read Block**. The specified memory block is read.
6. Enter a hexadecimal value in the memory block locations and click **Write Block**. The memory blocks are replaced with the specified values.

7. Click **Save Block Data** to save the recent changes.

Figure 11 • **Read-Write Memory Blocks**



2.1.4.4 Probe Insertion

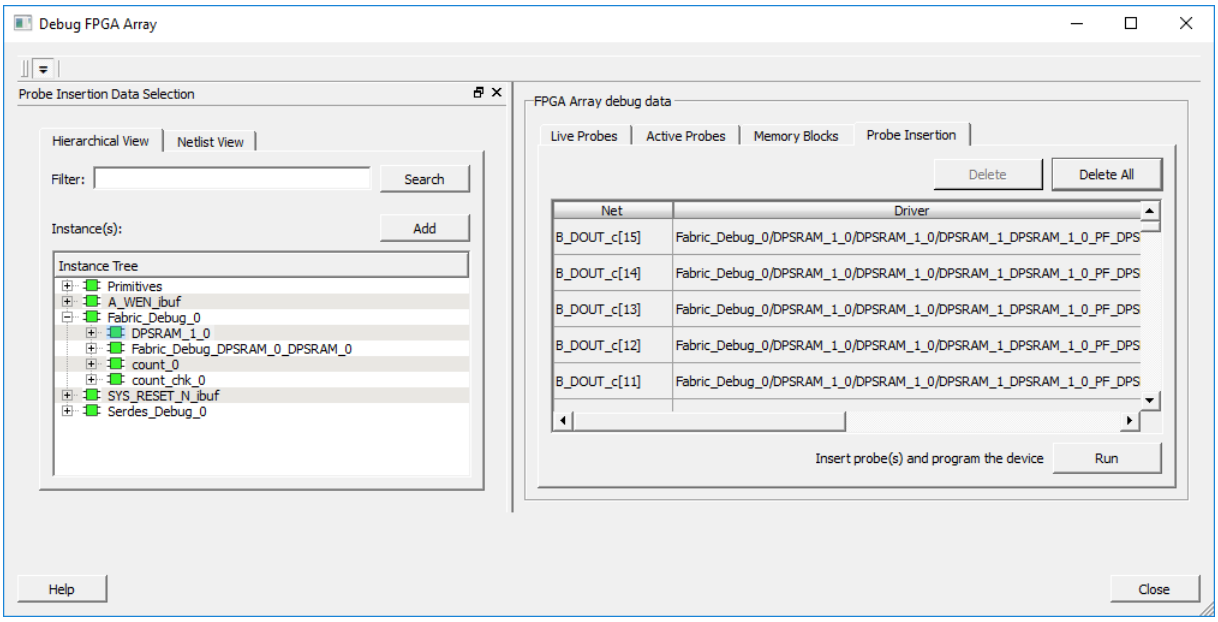
Probe insertion is a post-layout process that enables you to insert probes into the design and to bring signals out to the FPGA package pins. Probe insertion enables selecting internal ports in the design, connecting of those ports to used or unused pins, and then running the layout.

Note: FlashPro programmer must be connected to SmartDebug before inserting probes.

To insert probes into the design:

1. Double-click **SmartDebug Design** in the **Design Flow** window. The **SmartDebug Design** window is displayed.
2. Select **Debug FPGA Array** and click the **Probe Insertion** tab.

Figure 12 • **Inserting Probes**



3. Select the **Probe Insertion** tab in the right pane. The probe signals are displayed in the left pane.
4. Select probe points from the **Hierarchical View** or **Netlist View** in the left pane.
5. Select the probe points and right-click and then click **Add** to add them in the right pane. You can also add the selected probe points by clicking **Add** in the top-right corner in the left pane. The probes signals can be filtered with the **Filter** option.
6. Assign a package pin to the probe using the drop-down list in the package pin. You can assign a probe to any unused package pin or spare I/O.

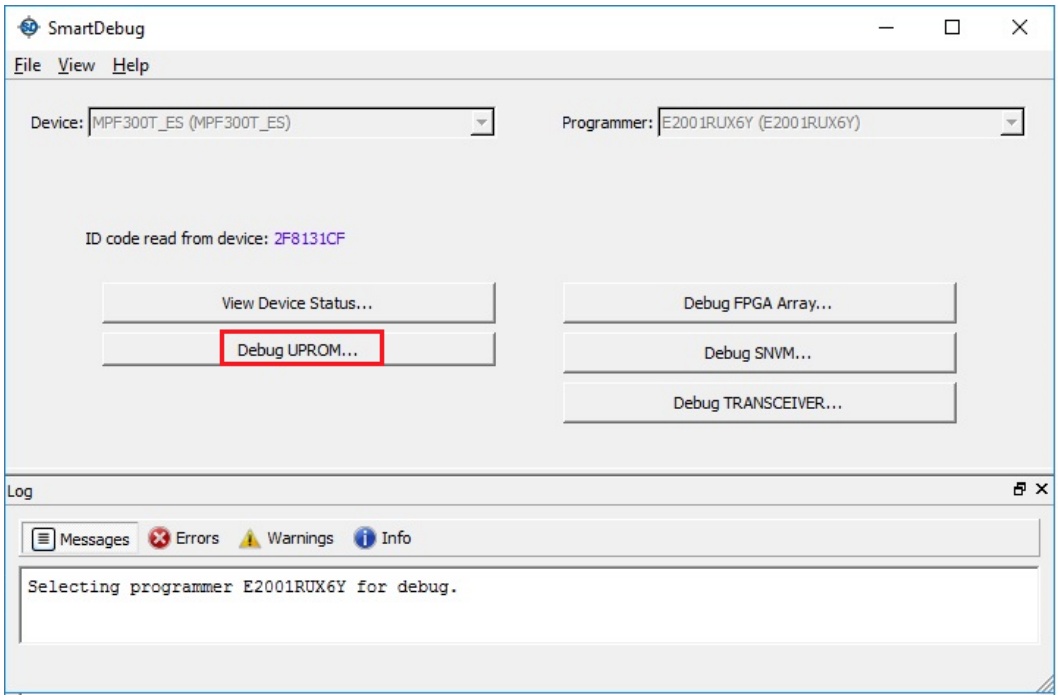
7. Click **Run** in the bottom-right corner of the window to run the **Place and Route** in the incremental mode. The selected probe nets are routed to the selected package pins. After incremental place and route, Libero SoC automatically reprograms the device with the added probes.
8. To delete the probe, select the probe and click **Delete or Delete All**. Deleting probes from the probe list without performing the **Run** functionality does not remove probes from the design.

2.1.5 Debug µPROM

SmartDebug enables debugging µPROM and reading its µPROM contents. The clients added in the design can be debugged using the SmartDebug Debug uPROM feature.

To debug µPROM and its contents, select **Debug µPROM** in the **SmartDebug** window.

Figure 13 • **Debug µPROM**



The **µPROM Debug** dialog box displays µPROM instance used in the design.There are two tabs in the µPROM debug window:

- **User Design View**—lists all the µPROM clients configured in the design.
- **Direct Address View**—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.

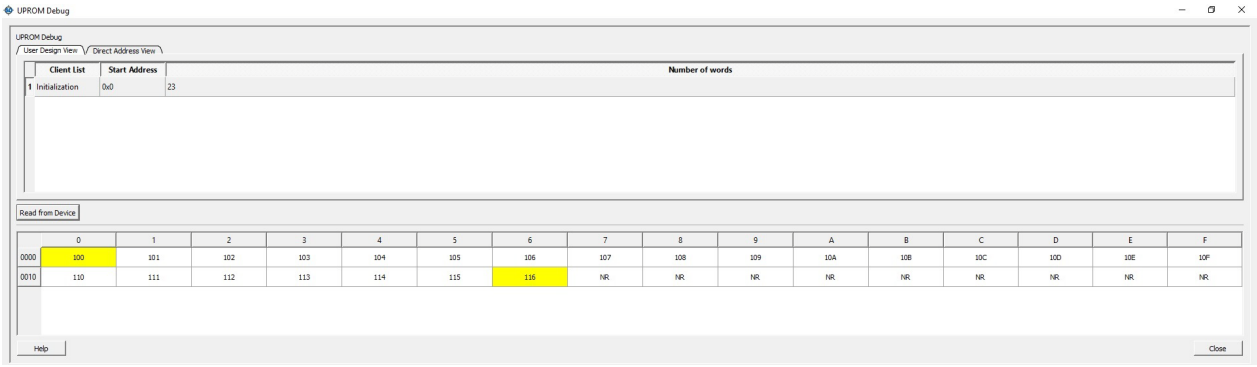
Note: µPROM clients added in **Design and Memory Initialization** tab and the UIC clients cannot be debugged through SmartDebug.

To view µPROM clients in the user design:

1. Select the **User Design View** tab.
2. Select the **Client Name** from the client list. The start address and number of words that are reprogrammed are displayed in the window.

3. Click **Read from Device**. The data of the selected client address is displayed. The client address is associated with a start address and number of 9-bit words. Therefore, the table contains as many locations as the number of 9-bit words.

Figure 14 • **View User Design**



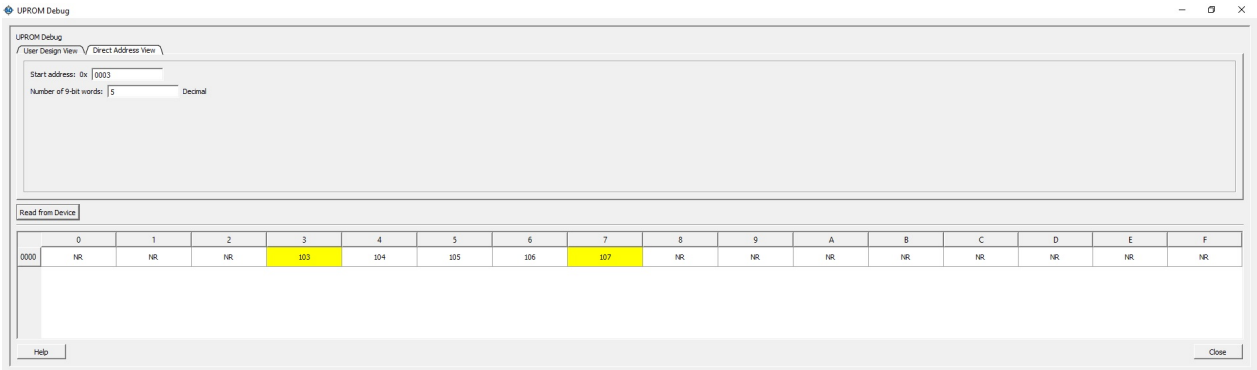
To view direct address of a µPROM client:

1. Select the **Direct Address View** tab.
2. Enter the **Start address** and **Number of 9-bit words**.

Note: The start address must be a hexadecimal value.

3. Click **Read from Device**. The data of the range specified is displayed.

Figure 15 • **Viewing Direct Address**



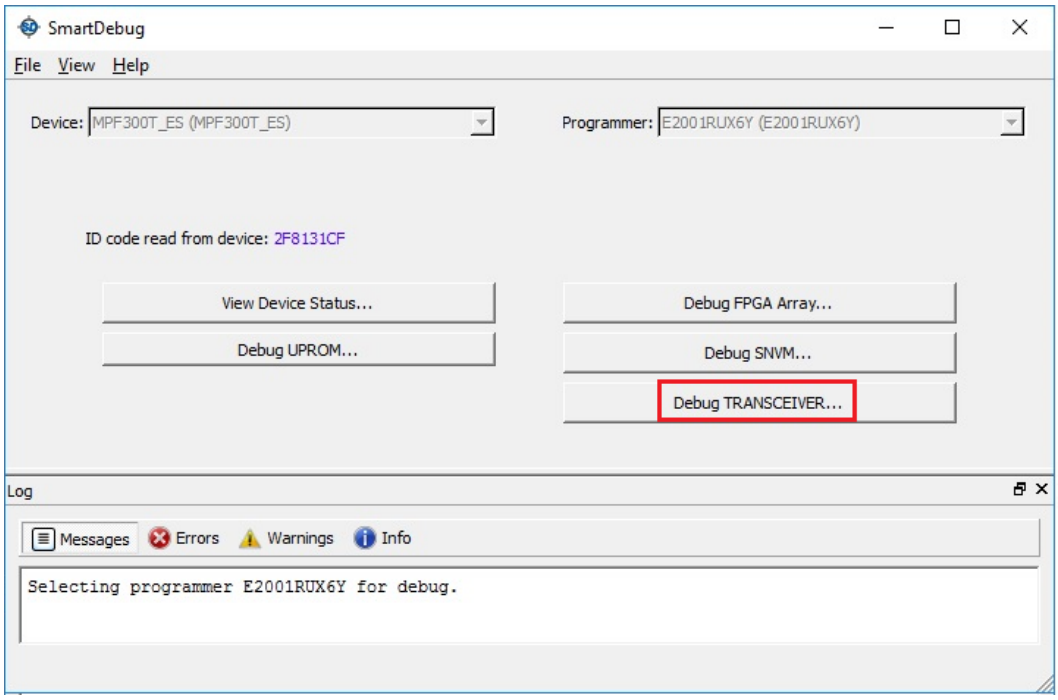
2.1.6 Debug TRANSCEIVER

SmartDebug enables transceiver debugging, which includes checking lane functionality and health for different settings of lane parameters. To access the debug transceiver feature, select **Debug TRANSCEIVER** in the **SmartDebug** window.

Debug Transceiver supports the following features:

- Configuration Report, page 15
- SmartBERT, page 15
- LoopBack Modes, page 18
- Static Pattern Transmit, page 18
- Eye Monitor, page 19

Figure 16 • **Debug Transceiver**

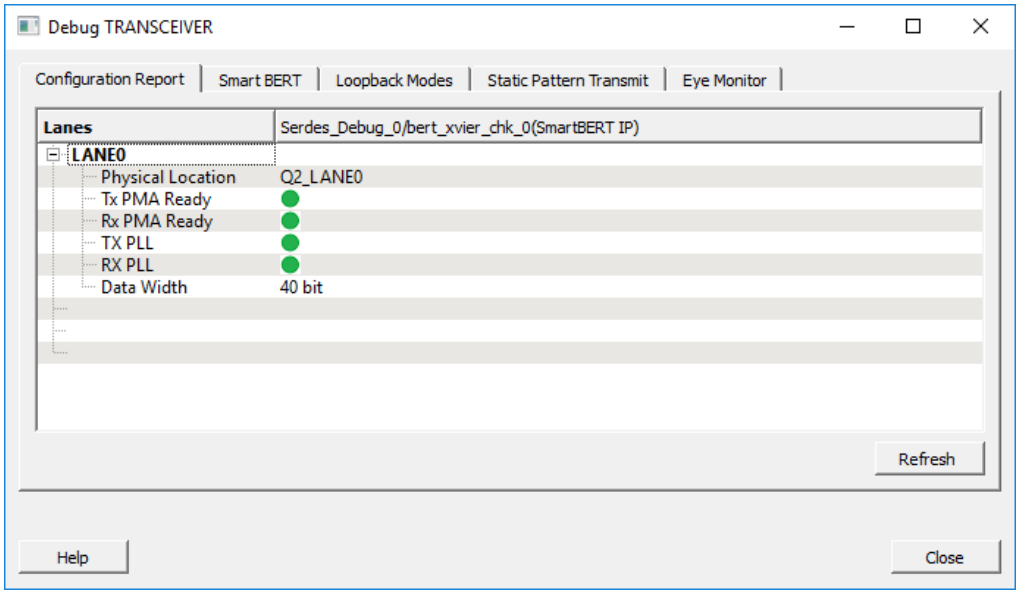


2.1.6.1 Configuration Report

The **Configuration Report** feature creates a report that shows the physical location, TX and RX PLL lock status, and data width of all enabled transceiver lanes. This report includes the following lane parameters:

- **Physical Location**—physical location of the transceiver lanes in the system.
- **Tx PMA Ready**—Tx lane of the transceiver is powered up and ready for transactions.
- **Rx PMA Ready**—Rx lane is powered up and ready for transactions.
- **TX PLL**—TX PLL of the transceiver is Locked.
- **Rx PLL**—Rx PLL of the transceiver is Locked.
- **Data Width**—configured data width of the corresponding lanes in the transceiver.

Figure 17 • **Configuration Report**



Note: Red indicates that the lanes are not configured in the current system.

2.1.6.2 SmartBERT

For any transceiver design, PRBS tests from XCVR PMA are available by default. SmartBERT enables you to run diagnostic tests on the transceiver lanes.

2.1.6.2.1 Running with PRBS generator

SmartBERT uses the PRBS generator and checker functionality available in each transceiver lane to determine the bit error rate (BER) of a lane. The various PRBS patterns supported are PRBS7, PRBS9, PRBS15, PRBS23, and PRBS31. Near-end loopback can be performed using one of these PRBS patterns. Bit Error Rate (BER) displays the BER for the PRBS test in progress.The formula for calculating BER is as follows:

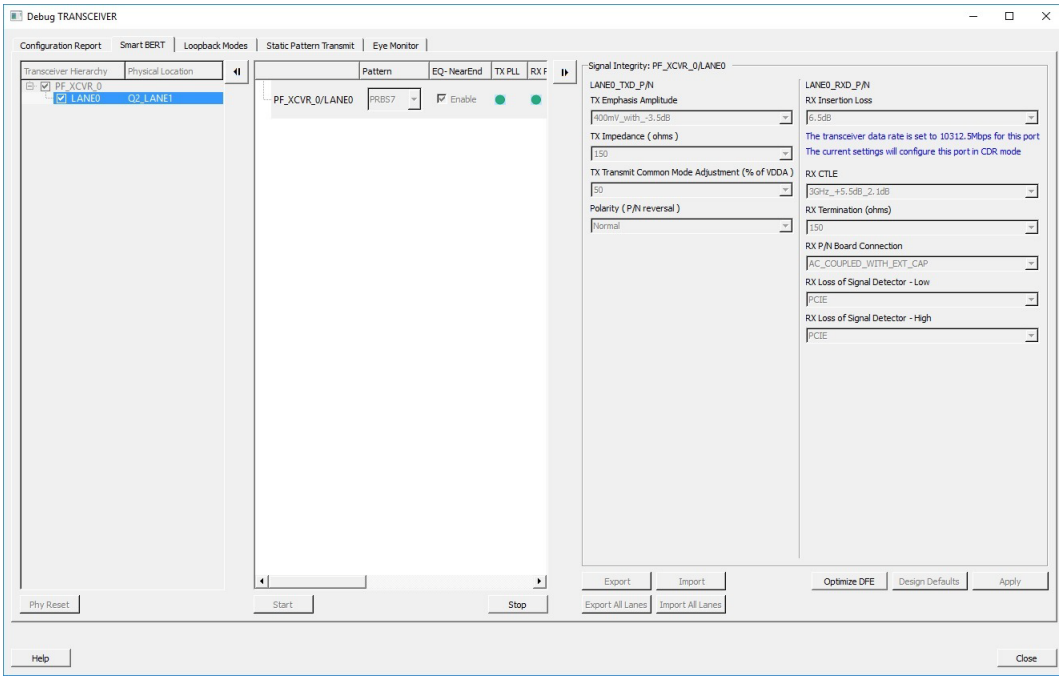
BER = (1 + Error Count)/(Data Rate × Seconds)

To run a PRBS pattern:

1. Select one of the **Patterns** from the drop-down list.
2. Select the **EQ-NearEnd** check box.
3. Click **Start** in the bottom-left corner of the window. The loopback cycle is initiated and the result is displayed.

4. Click **Stop** in the bottom-right corner of the window to stop the loopback.

Figure 18 • **SmartBert with PRBS Generator**



2.1.6.2.2 Running with SmartBERT IP

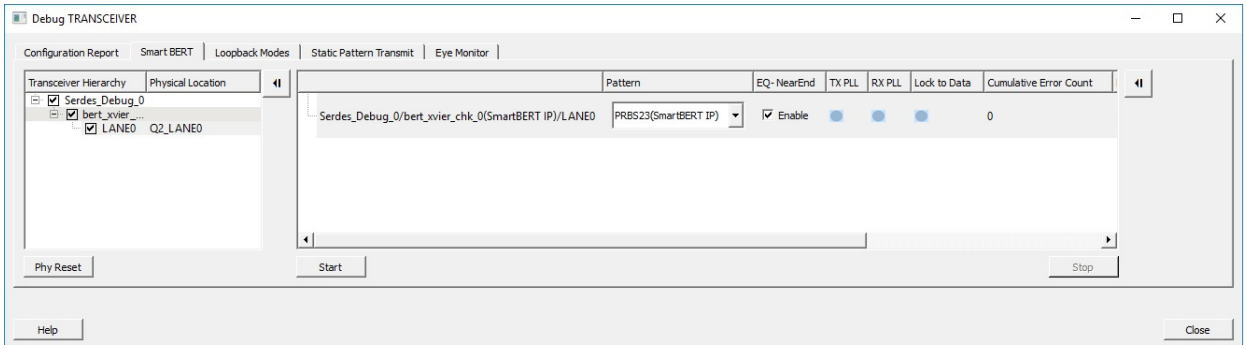
In addition to the PRBS generator and checker available on transceiver lane, SmartDebug provides the user interface to control the SmartBERT IP instantiated in the design. The transceiver lanes configured with SmartBERT IP show PRBS patterns generated from XCVR PMA as well as the PRBS patterns generated from IP residing in the fabric in the dropdown. Each SmartBERT IP can have 4 lanes configured and each lane can have PRBS7, PRBS9, PRBS23, and PRBS31 patterns configured in the design.

The Debug Transceiver feature shows all lanes that are configured in the design. SmartDebug automatically detects the presence of the CoreSmartBERT in the design. It also identifies the lanes that use SmartBERT IP and distinguish them by appending SmartBERT IP next to the SmartBERT IP instance.

To run SmartBERT in transceiver follow these steps:

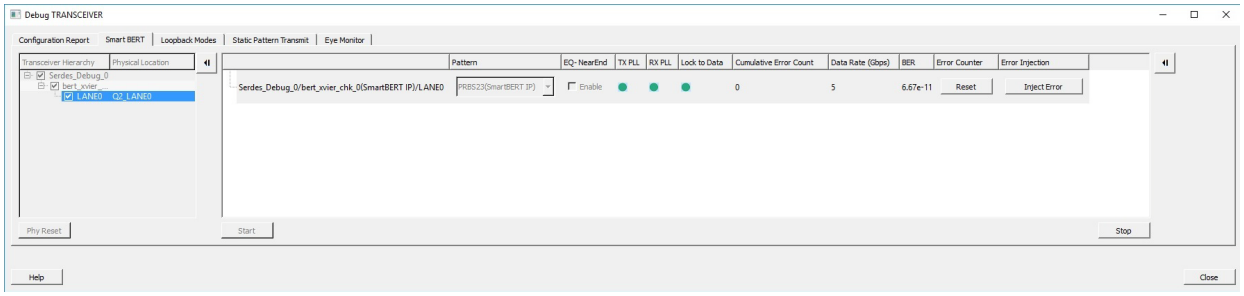
1. Select the **SmartBERT** tab in the **SmartDebug TRANSCEIVER** window.
2. Select the **Pattern** from the drop-down list where the lanes are visible.
3. Select the **EQ-Nearend** check box. When checked, the selected lane gets added to the right hand side where PRBS test can be performed. When unchecked, the selected lane gets removed from the added list.

Figure 19 • **SmartBERT in Transceiver**



4. Click **Start** on the lower-left corner of the pane. It enables both transmitter and the receiver for a particular lane and for a particular PRBS pattern.The GUI shows the status of the TXPLL, RXPLL,Lock to Data, Data rate, and the BER.

Figure 20 • **Viewing the Status of TXPLL, RXPLL Lock to Data, Data Rate, and BER**



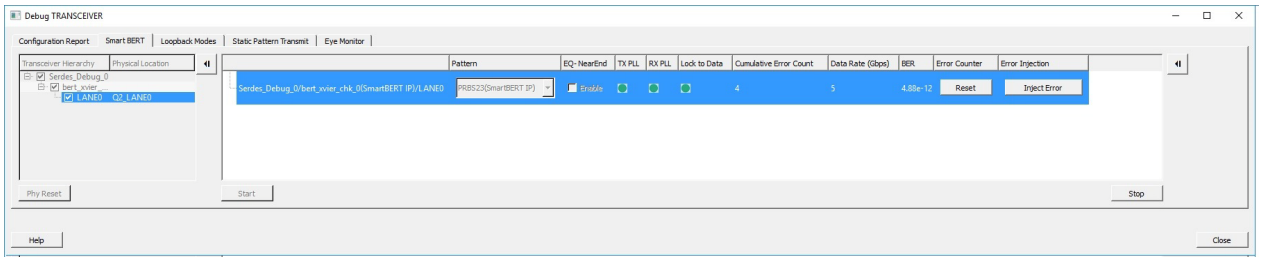
Error Injection

When a SmartBERT IP lane is added, the **Error Injection** column is displayed in the right pane. The Error Injection feature is provided to inject an error while running a PRBS pattern. This feature is unavailable if regular lanes are added. Also, this feature is disabled for a SmartBERT IP lane that has a non-configured PRBS pattern selected.

Error Count

Error count is displayed when the lane is added and PRBS pattern is run. Click **Reset** to clear the error count under **Error Counter**.

Figure 21 • **Viewing Error Counter and Error Injection**

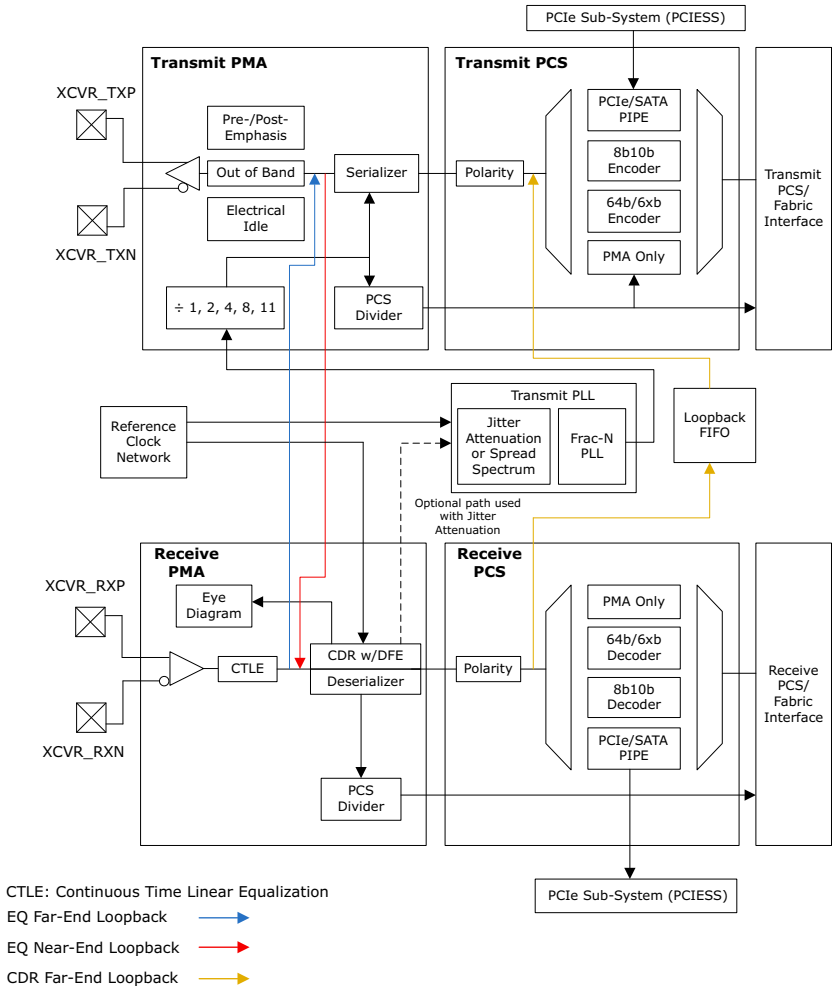


2.1.6.3 LoopBack Modes

Loopback modes help you perform the following types of loopback tests:

- **EQ-Near End Looback**—Serialized data from PMA is looped from Tx to Rx internally before the transmit buffer. This is called near-end serial loopback. EQ-Near End Looback supports data transmission rates of up to 10.315 Gbps.
- **EQ-Far End Looback**—Serialized data from Rx is looped back to Tx in PMA. This is called far-end serial loopback. EQ-Far End Looback supports data transmission rates of up to 1.25 Gbps.
- **CDR-Far End Looback**—De-serialized data from PCS Rx channel is looped back to Tx.
- **No-Loop Back**—Data is not looped internally.

Figure 22 • **Transceiver Loopback**



2.1.6.4 Static Pattern Transmit

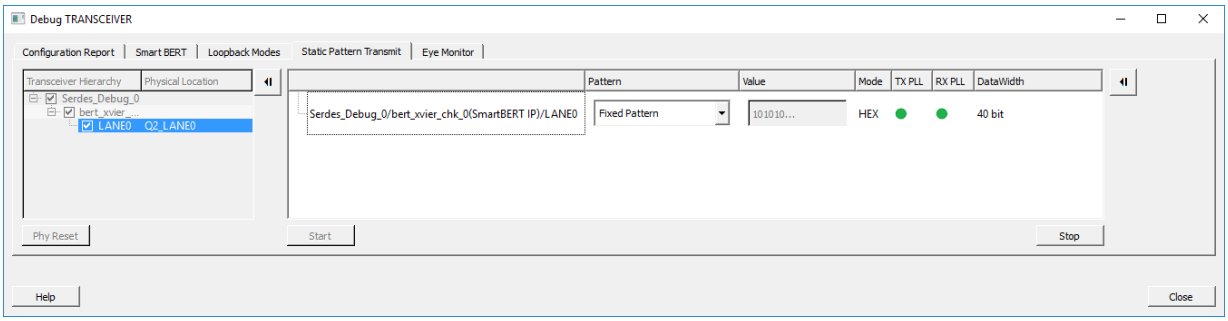
Static pattern transmit enables the selection of pattern to be transmitted on a specific transceiver (Tx) lane. The following patterns are supported:

- Fixed pattern
- Max run length pattern
- User pattern—The pattern is defined in the value column. It must be hex numbers and not greater than the configured data width.
- TX-PLL—Indicates lane lock onto TX PLL when a static pattern is transmitted.
- RX-PLL—Indicates RX PLL lock when a static pattern is transmitted.
- Data Width—Displays the data width configured for a transceiver lane.

To view static pattern transmit:

1. Select the **Static Pattern Transmit** tab.
2. Select the **Transceiver Hierarchy** in the left pane of the window. The selected lane data is displayed in the right pane.
3. Select a pattern from the **Pattern** drop-down list.
4. Click **Start**. The static pattern for the selected lanes is transmitted.
5. Click **Stop**. The static pattern transmission is stopped for the selected lanes.

Figure 23 • **Static Pattern Transmit**

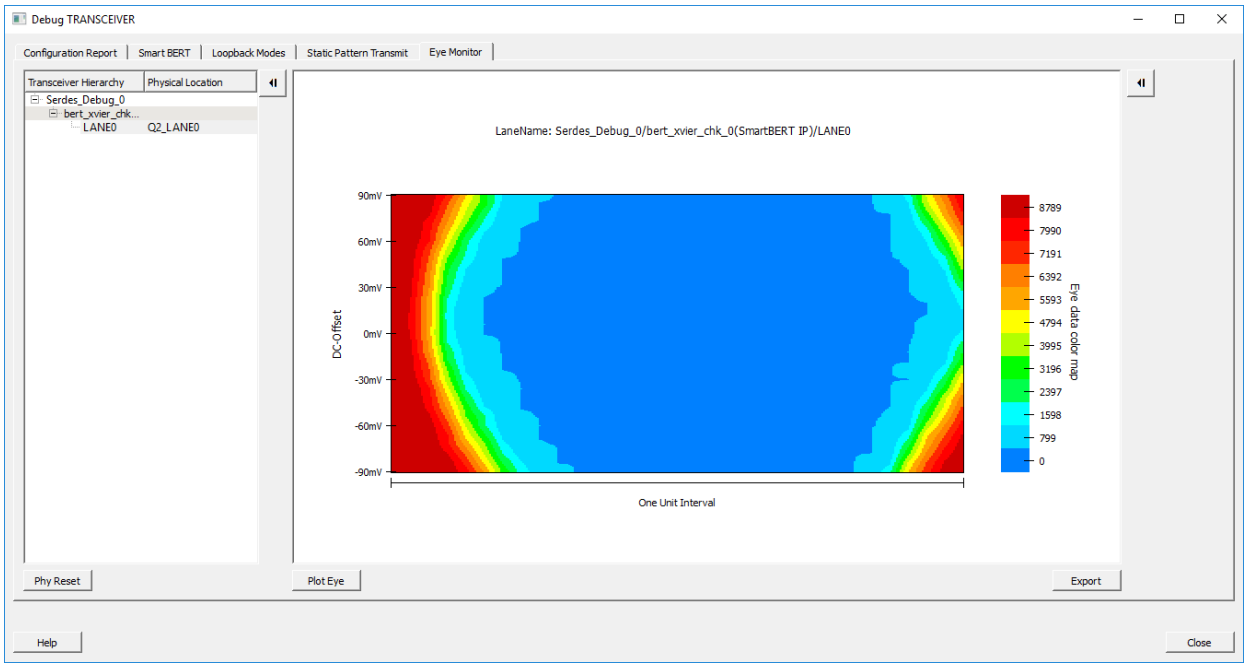


2.1.6.5 Eye Monitor

Eye monitor enables visualizing the eye diagram present within the receiver. This feature plots the receive eye after the CTLE and DFE functions. The diagram representation provides vertical and horizontal measurements of the eye and BER performance measurements. Eye Monitor is supported for data rates above 3.125 Gbps.

Click the **Eye Monitor** tab in the **Debug TRANSCEIVER** window to see the eye monitor representation within the receiver.

Figure 24 • **Viewing the Eye Monitor Diagram**



2.1.7 Signal Integrity

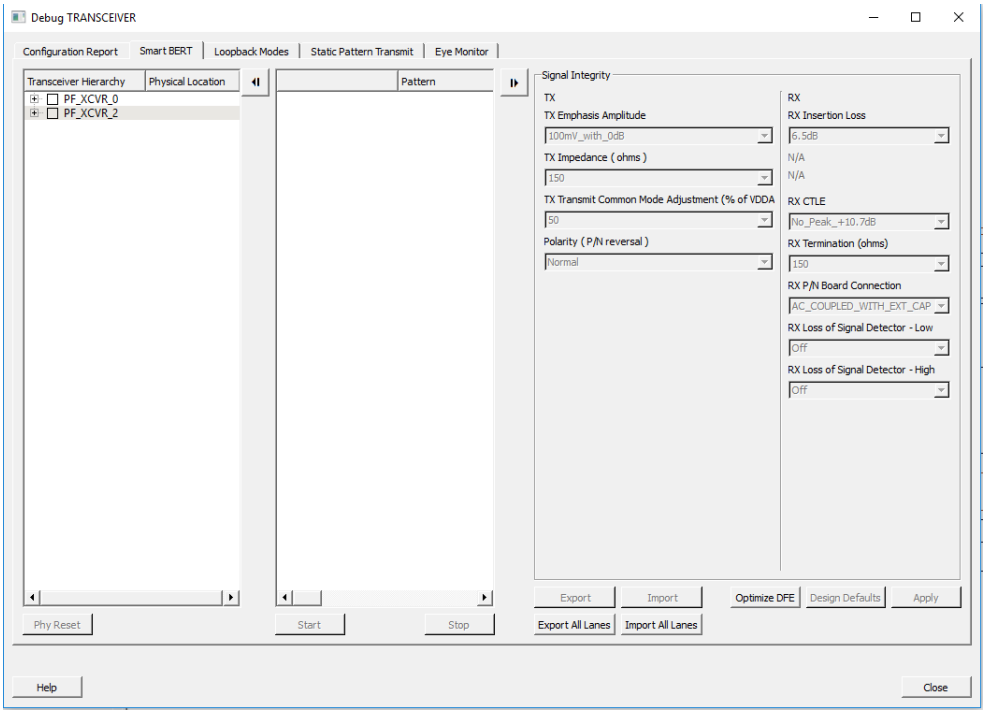
The signal integrity feature in SmartDebug works with Signal Integrity in the I/O Editor, allowing the import and export of `.pdc` files.

The **Signal Integrity** pane appears in the following SmartDebug pages:

- SmartBERT, page 15
- LoopBack Modes, page 18
- Static Pattern Transmit, page 18
- Eye Monitor, page 19

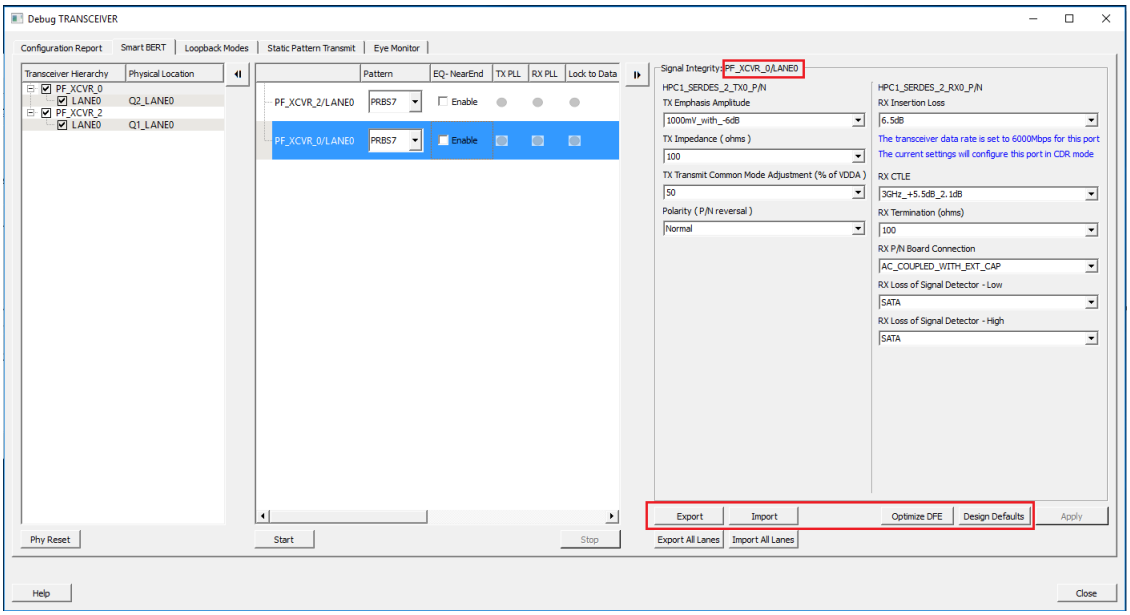
When you open Debug Transceiver in SmartDebug and click the **SmartBERT, Loopback Modes, Static Pattern Transmit,** or **Eye Monitor** tab, all parameters in the **Signal Integrity** pane are disabled. Only the **Export All Lanes** and **Import All Lanes** options are enabled, as shown in the following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

Figure 25 • **View Signal Integrity for SmartBERT**



When a lane is selected in the **SmartBERT, Loopback Modes, Static Pattern Transmit,** or **Eye Monitor** pages, the corresponding Signal Integrity parameters that are configured in the I/O Editor or changed in SmartDebug, are enabled, as shown in the following figure.

Figure 26 • **Viewing Signal Integrity for a Selected Lane**



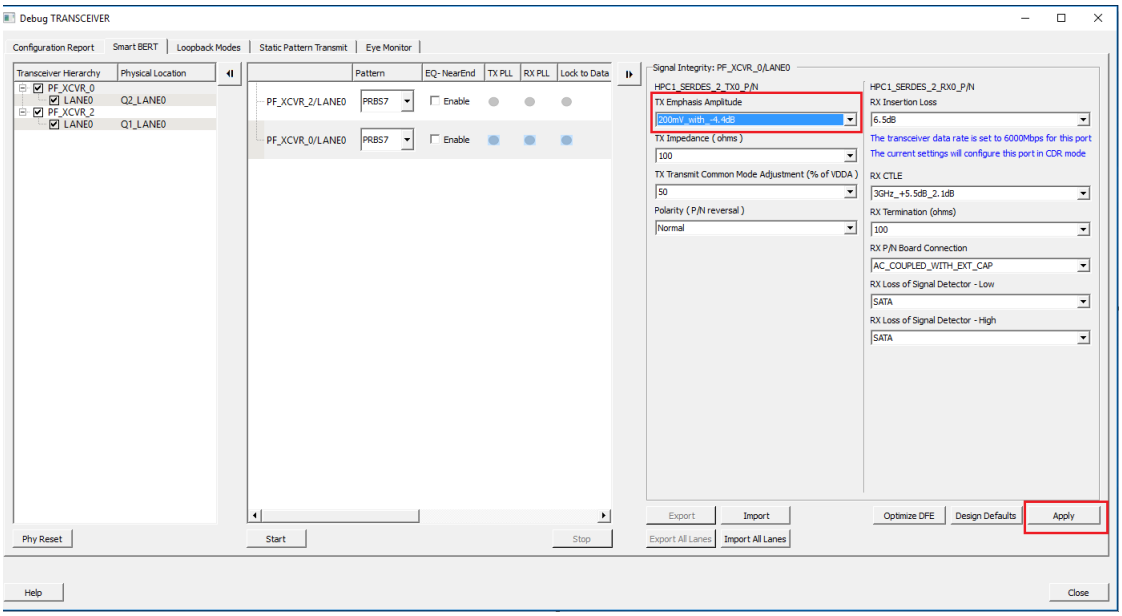
The selected lane instance name is displayed in the **Signal Integrity** group box, and the **Export, Import,** and **Design Defaults** options are enabled.

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.

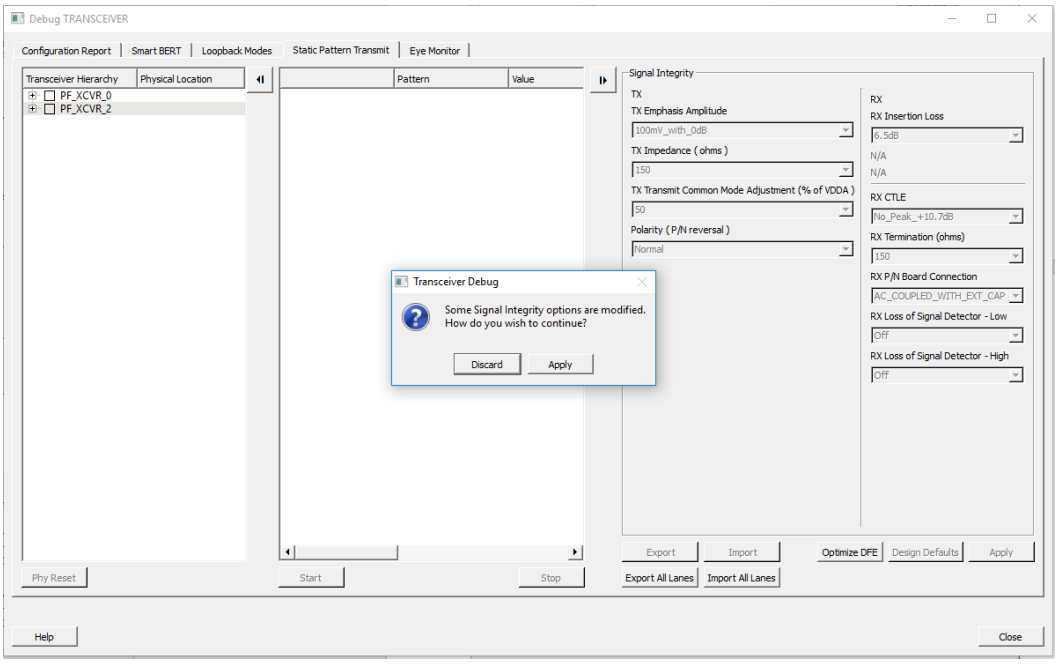
Note: The **Apply** button is enabled when you make a selection for any parameter, as shown in the following figure.

Figure 27 • **Applying the Changes for the Signal Integrity**



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, an alert message is displayed.

Figure 28 • **Viewing a Confirmation Message**

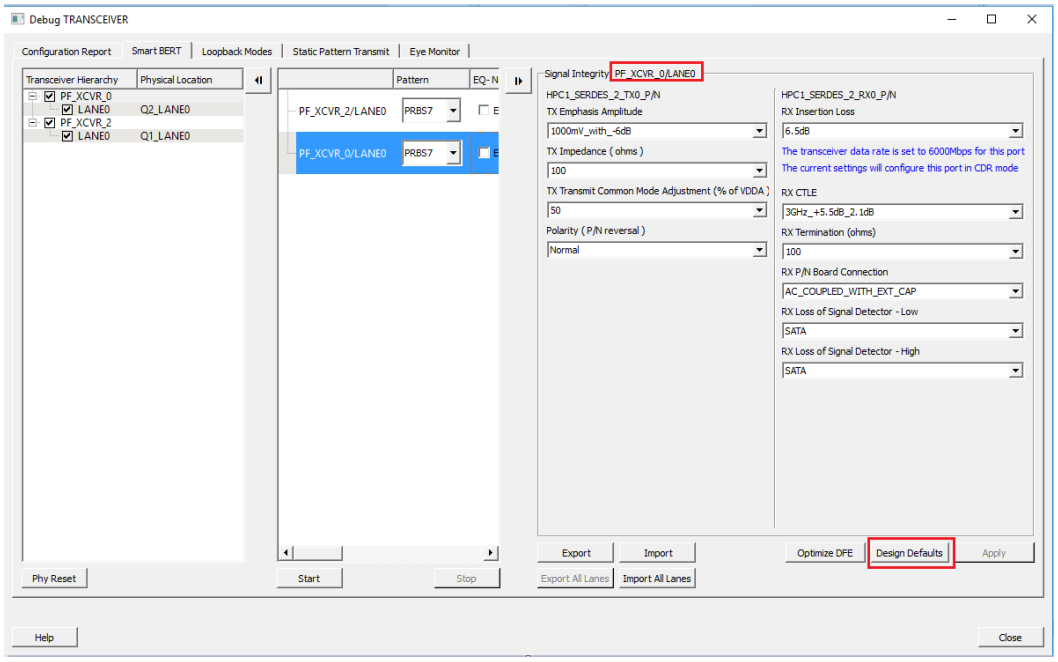


Click **Apply** to apply or click **Discard** to discard the changes.

2.1.7.1 Design Defaults

Click **Design Defaults** to load the Signal Integrity parameter options for the selected lane instance.

Figure 29 • **Viewing the Design Default Constraints**



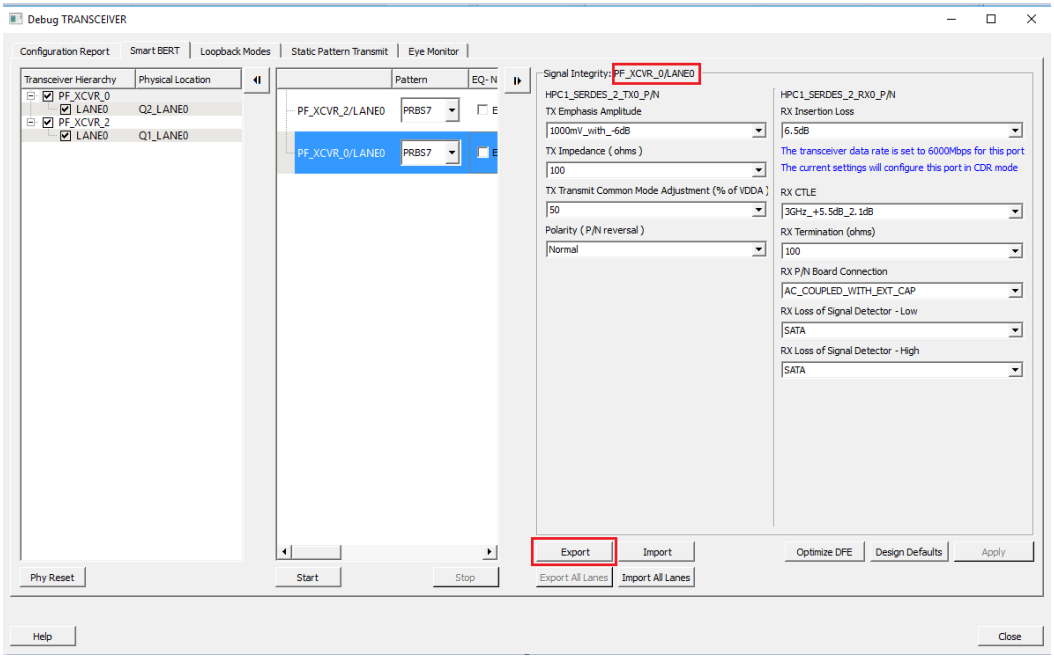
Design Default parameter options are applied to the device and updated in Modified Constraints.

Note: Modified Constraints is a list of I/O constraints set on the TXP and RXP 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.

2.1.7.2 Export

Click **Export** to export the current selected parameter options along with other physical information of the selected lane instance to an external PDC file.

Figure 30 • **Exporting the Selected Lane Details**

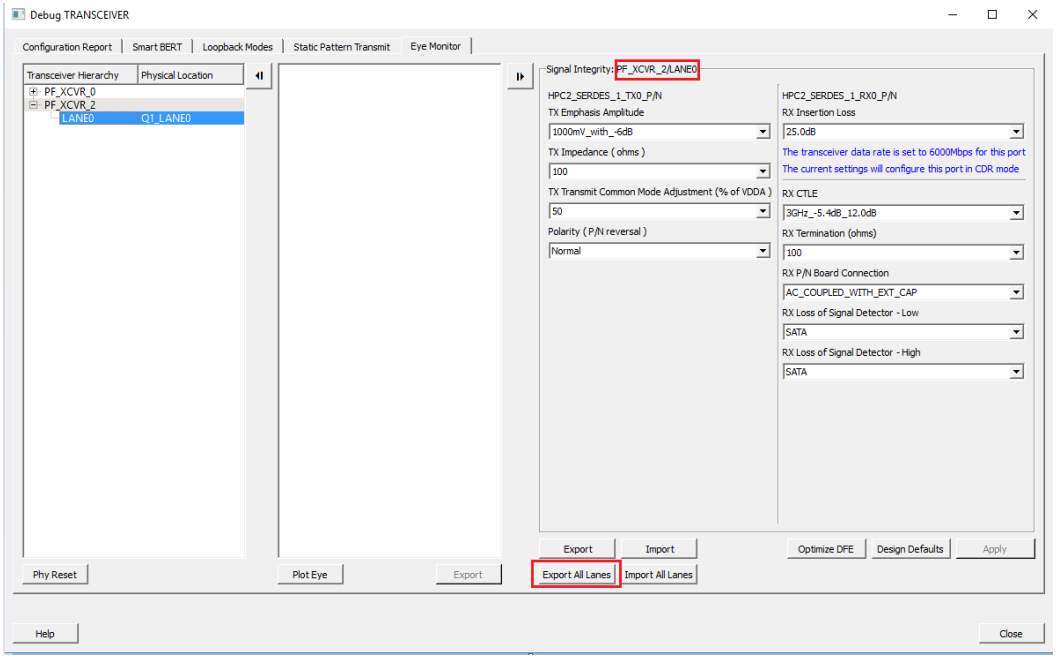


The exported content is in the form of two set_io commands, one for the TXP port and one for the RXP port of the selected lane instance.

2.1.7.3 Export All Lanes

Click **Export All Lanes** to export the current selected parameter options and other physical information for all lane instances in the design to an external PDC file.

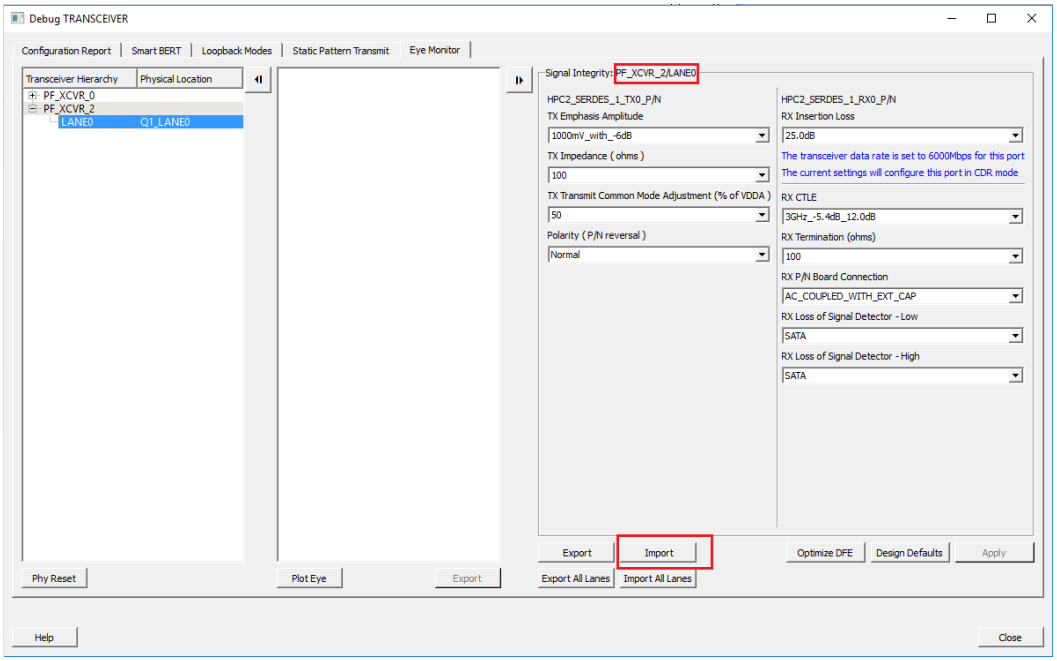
Figure 31 • **Exporting All Lane Details**



2.1.7.4 Import

Click **Import** to import Signal Integrity parameter options and other physical information for the selected lane from an external PDC file.

Figure 32 • **Importing a Selected Lane Details**

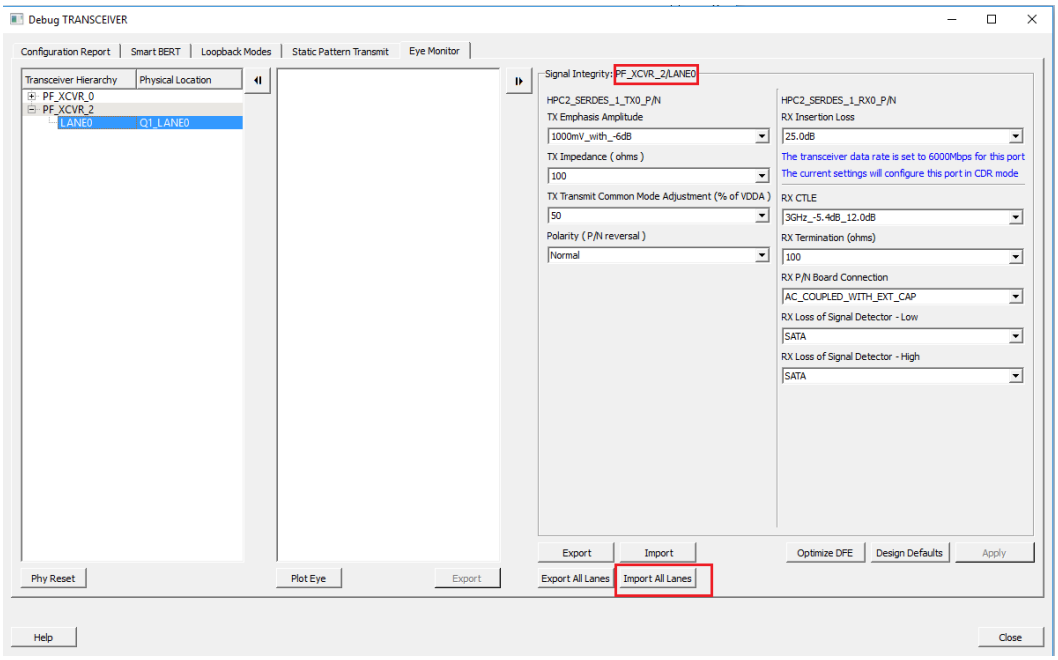


The Signal Integrity parameter options are applied to the device and updated in Modified Constraints.

2.1.7.5 Import All Lanes

Click **Import All Lanes** to import Signal Integrity parameter options and other physical information for all lanes from an external PDC file.

Figure 33 • **Importing All Lane Details**

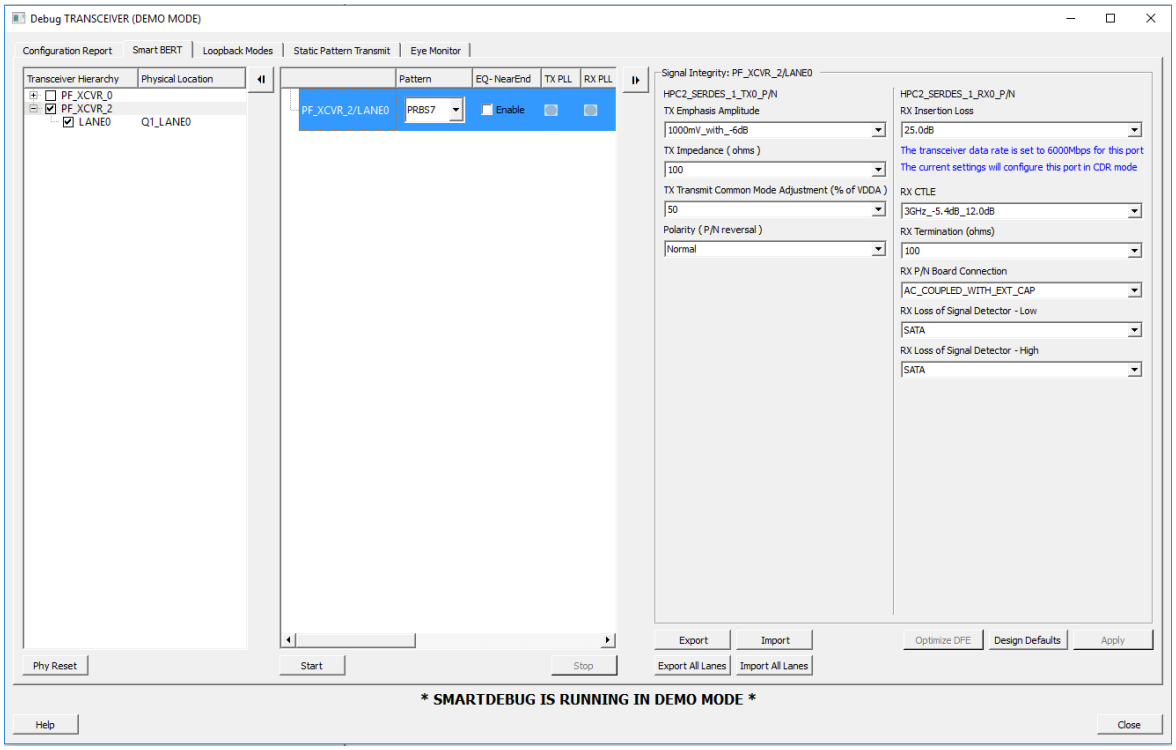


The Signal Integrity parameter options are applied to the device and updated in Modified Constraints.

2.1.7.6 Demo Mode

Signal Integrity in **Demo Mode,** lets users experience and understand the debug activities that can be performed with the Signal Integrity feature in SmartDebug. All debug activities except Apply are available in demo mode.

Figure 34 • **Viewing Signal Integrity Features in Demo Mode**



2.1.8 sNVM Debug

sNVM Debug enables reading from and writing to the sNVM during debug. It can be done by reading each page or reading multiple pages based on the authentication. Debug Pass Key is required to carry out SNVM_DEBUG instruction.This feature supports debugging of plain text non-authenticated, authenticated plain text, and cipher authenticated plain text.

Following are the two ways of sNVM debugging which can be performed in SmartDebug:

- When the USK is programmed using USK client, it applies to all the authenticated pages or clients.
  - SmartDebug has to read the USK client and store it as the key whenever the content is read from a client or a page.
- You can override the USK at runtime by changing it for a specific client or page using system services (IP).
  - If you select the option of using sNVM client as ROM, then USK cannot be overridden by system services where the authenticated pages always use the USK client to unlock and read.
  - You can override the type of page using system services. For example, if a page is non-authenticated, then it can be authenticated using system service routines at runtime.

You can also perform the following operations even after the design is programmed into the device:

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

Note: The preceding operations are not possible if the page is used as ROM.

The following sections describe the two views of sNVM Debug:

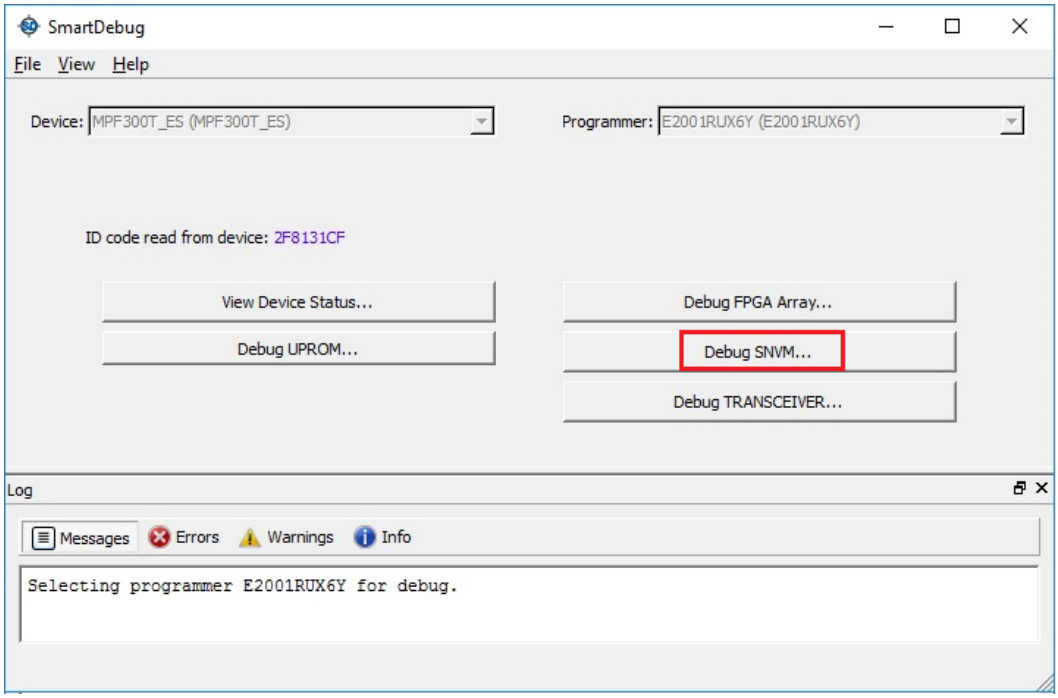
- Client View, page 28
- Page View, page 31

2.1.8.1 Client View

To view the client view, follow these steps:

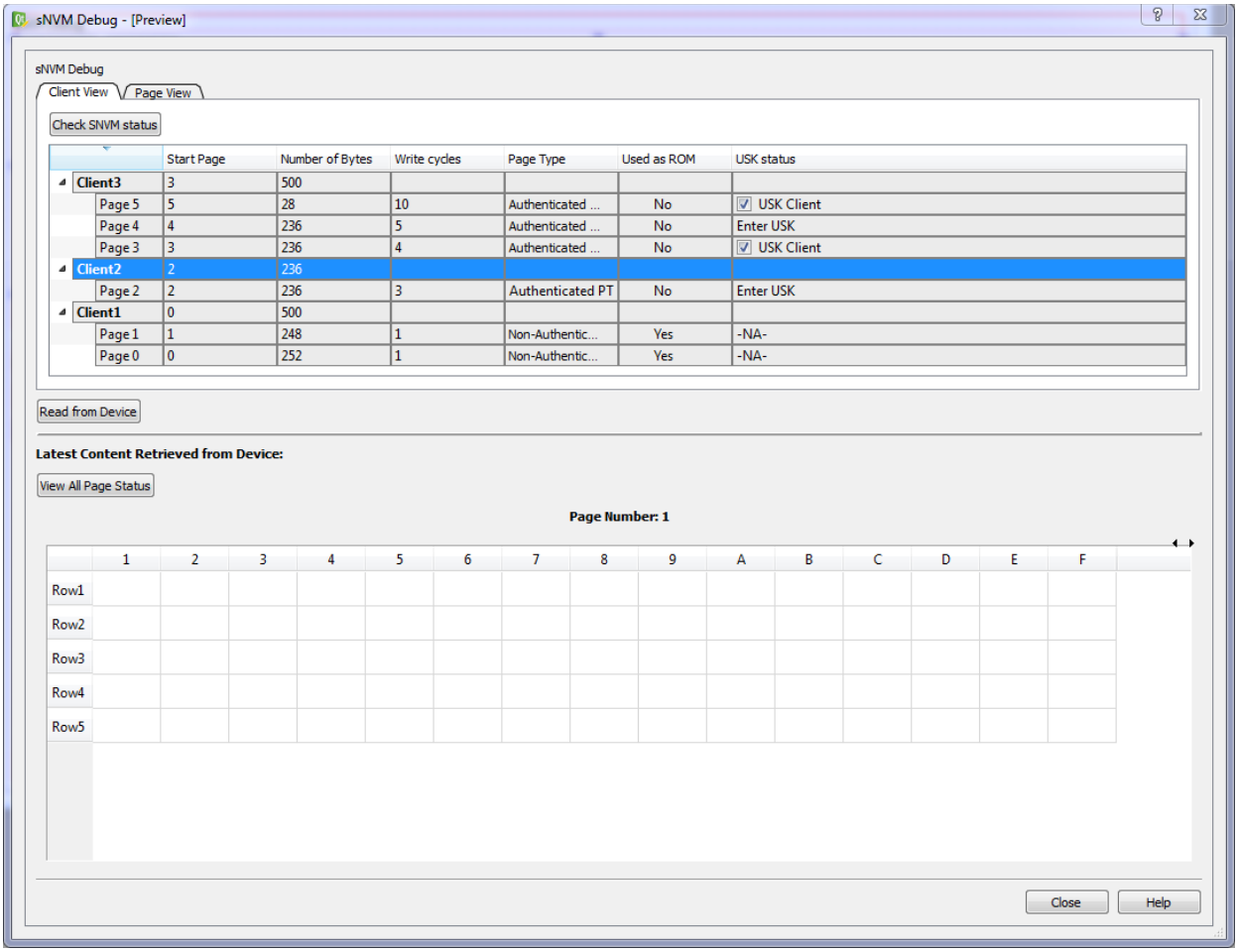
1. Click **Debug SNVM** in the **SmartDebug** window. The **sNVM Debug** window is displayed.

Figure 35 • **SmartDebug—Debug SNVM**



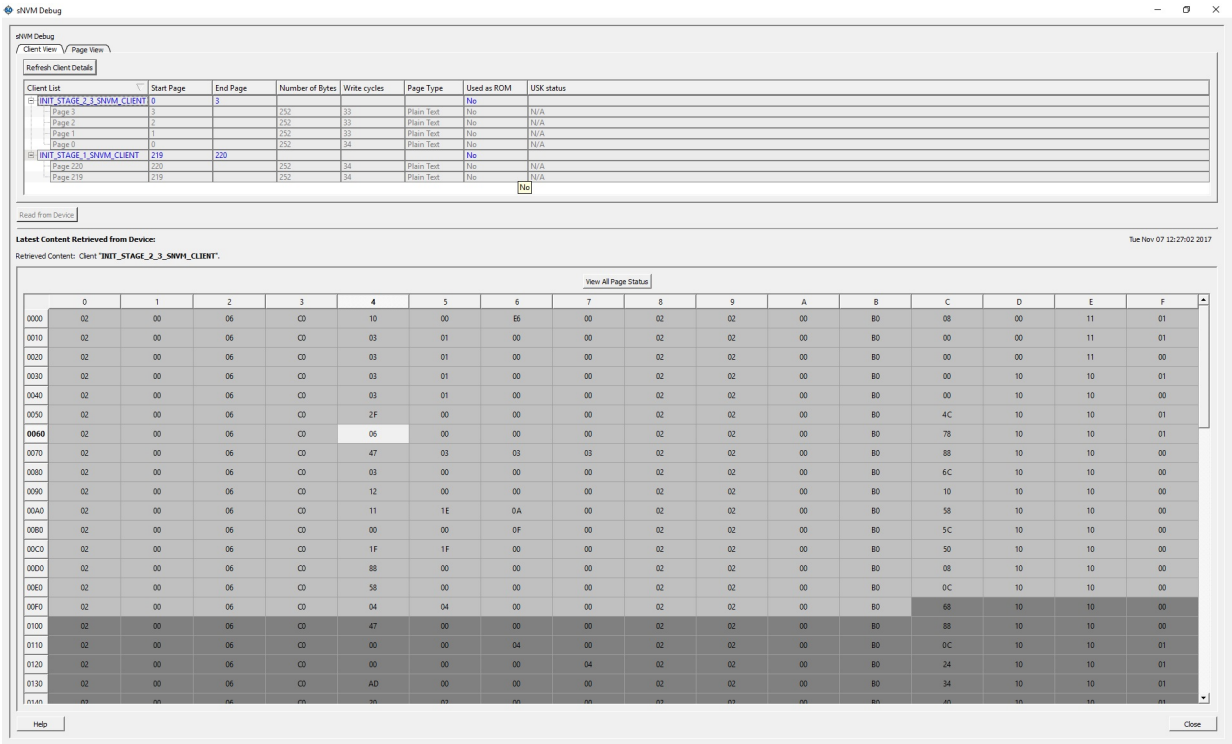
2. Click the **Client View** tab. The client view details are listed. It shows **Client Names, Start Page, Number of Bytes, Write Cycles, Page Type, Used as ROM,** and **USK Status** as shown in the following figure.

Figure 36 • **Viewing the Client View**



3. Select a client from the list in the **Client View** and click **Read From Device**. The client details are listed in the second pane.

Figure 37 • **Reading the Client View Details**



Note: Only one client can be selected at a time to read the client details. Pages inside the client cannot be selected.

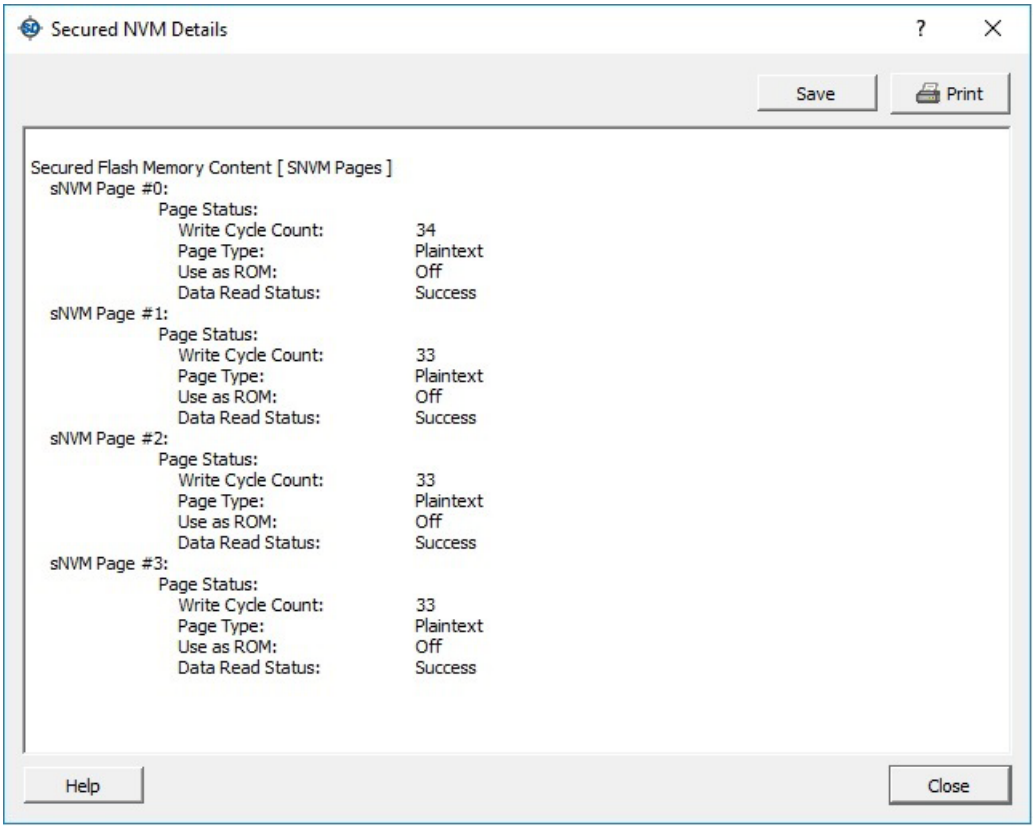
2.1.8.1.1 Refreshing sNVM Status

To view the latest details in the client page, Click **Check SNVM status**. The client pages are refreshed and display the recently updated details.

2.1.8.1.2 Viewing the Page Status

Click **View All Page Status** to view the page status such as Write Cycle Count, Page Type, Use as ROM, and Data Read Status.

Figure 38 • **Viewing the Page Status Report**



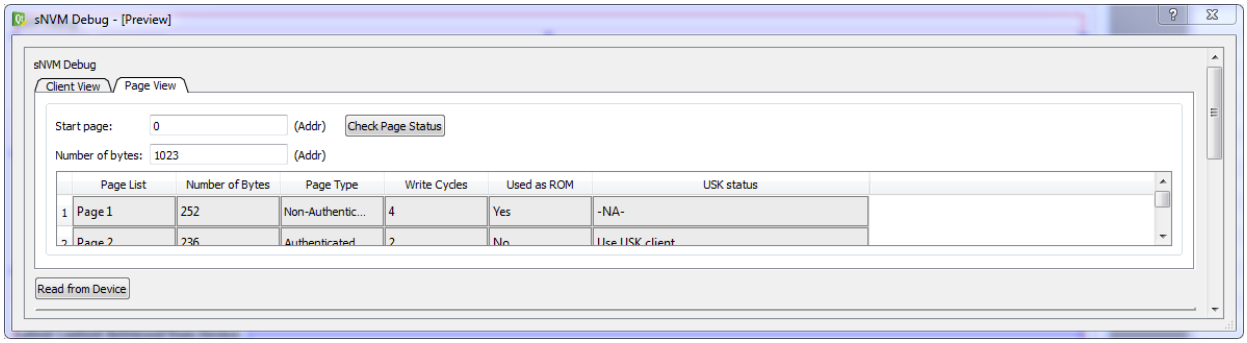
2.1.8.2 Page View

Page view displays the client details of the required pages. You can read pages from 0-220 in the page view.

To read data in page view:

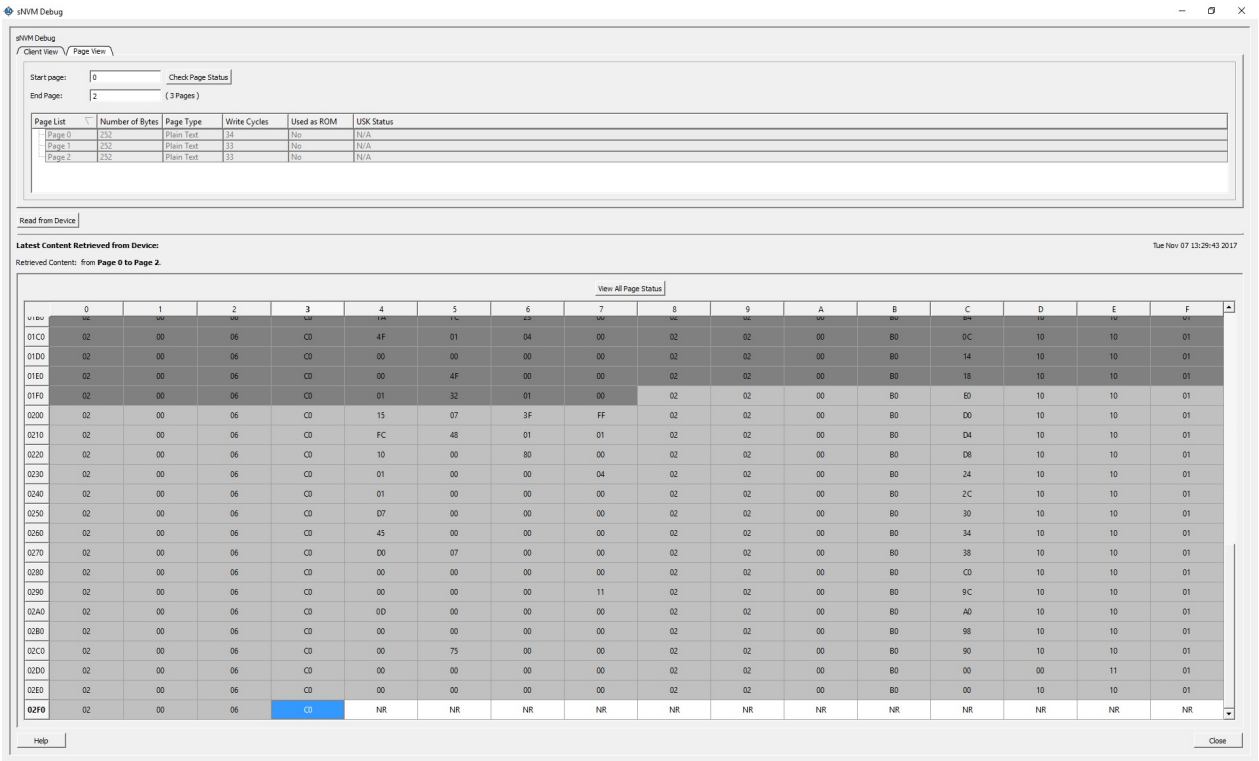
1. Click the **Page View** in the **sNVM Debug** window.
2. Enter the page number that you want to read in the **Start Page** and **Number of Bytes** in the respective boxes.
3. Click **Check Page Status**. The page status information is displayed.

Figure 39 • **Viewing Page View**



4. Select pages from the list, and click **Read from Device**. The page details are displayed.

Figure 40 • **Viewing Page View Details**



5. Click **View All Page Status**. The page status details are displayed.

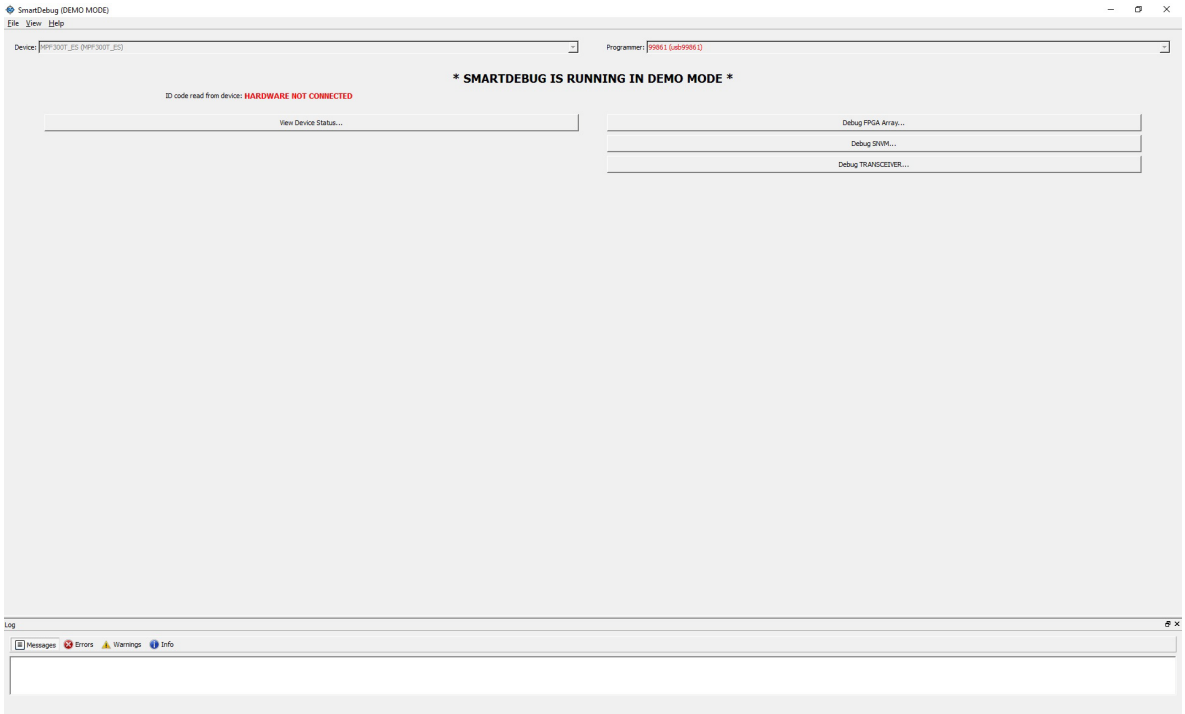
2.2 Demo Mode

The demo mode allows you to experience SmartDebug functionalities (Active Probe, Live Probe, Memory Blocks, and Debug Transceiver) without having to connect a board to the system running SmartDebug.

Note: The demo mode is for demonstration purposes only, and does not provide the full functionality of the standalone mode.
You cannot switch between demo mode and normal mode while SmartDebug is running.

The following figure is displayed when you open SmartDebug in demo mode.

Figure 41 • **Launching SmartDebug in Demo Mode**



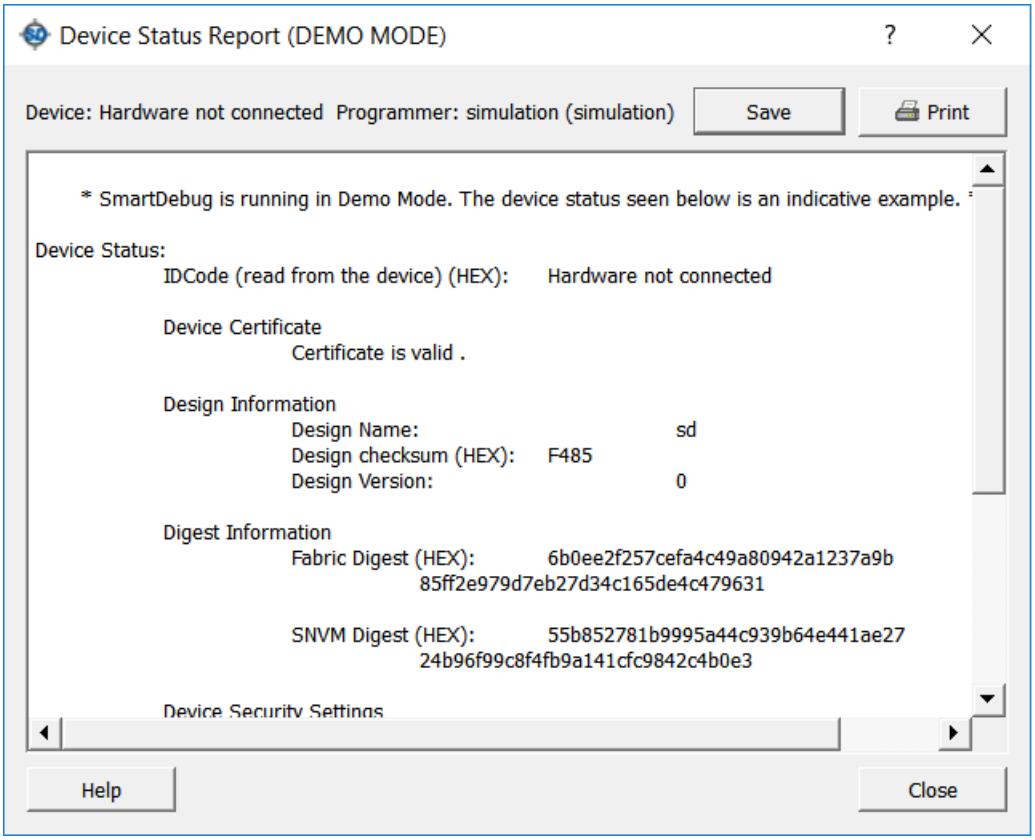
Demo mode supports all the following functionalities that Standalone mode supports:

- View Device Status, page 34
- Debug FPGA Array, page 34
- Debug TRANSCEIVER, page 37

2.2.1 View Device Status

The following figure shows an example of the **Device Status Report in Demo Mode**.

Figure 42 • **Viewing Device Status**



2.2.2 Debug FPGA Array

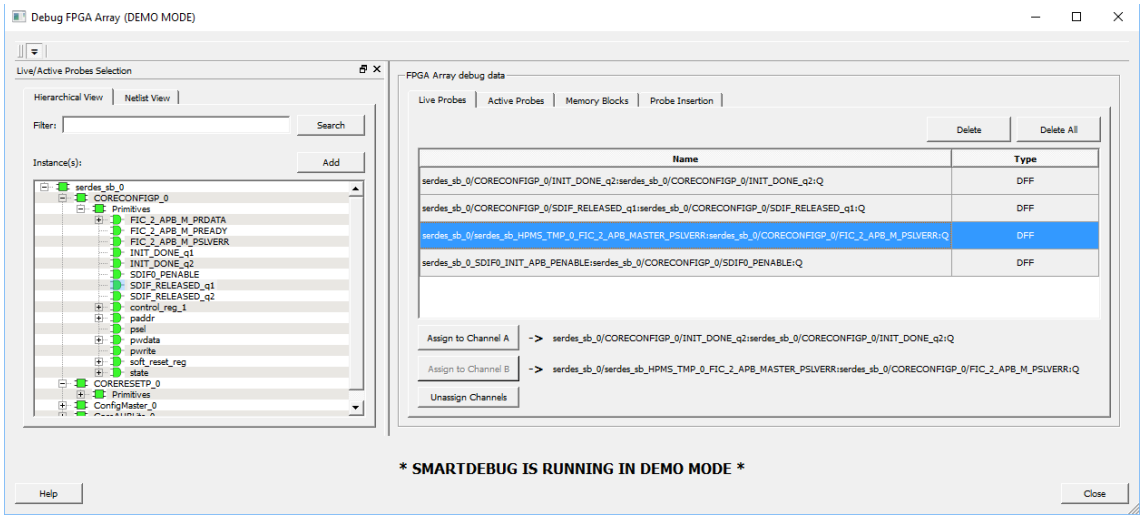
In the Debug FPGA Array dialog box, you can view your Live Probes, Active Probes, Memory Blocks, and Insert Probes (Probe Insertion).

Note: Insert Probes (Probe Insertion) is not supported in Demo mode.

2.2.2.1 Live Probes

You can assign and unassign Live Probes to Channel A and Channel B. The following figure shows an example of Live Probes assignment.

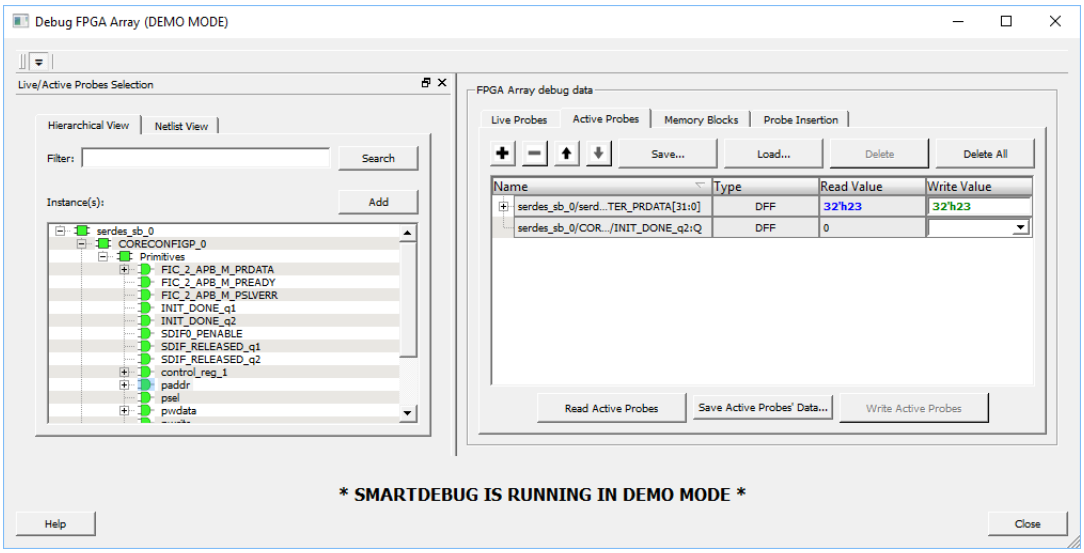
Figure 43 • **Assigning Live Probes to Channels**



2.2.2.2 Active Probes

In demo mode, a temporary probe data file with details of current and previous values of probes is added in the active probes. The write values of probes are updated to this file and the GUI is updated with values from this file when you click Write Probes. The values are read when you click **Read Probes**. If there is no existing data for a probe in the file, the read values are all 0s.

Figure 44 • **Reading or Writing Live Probe Values**

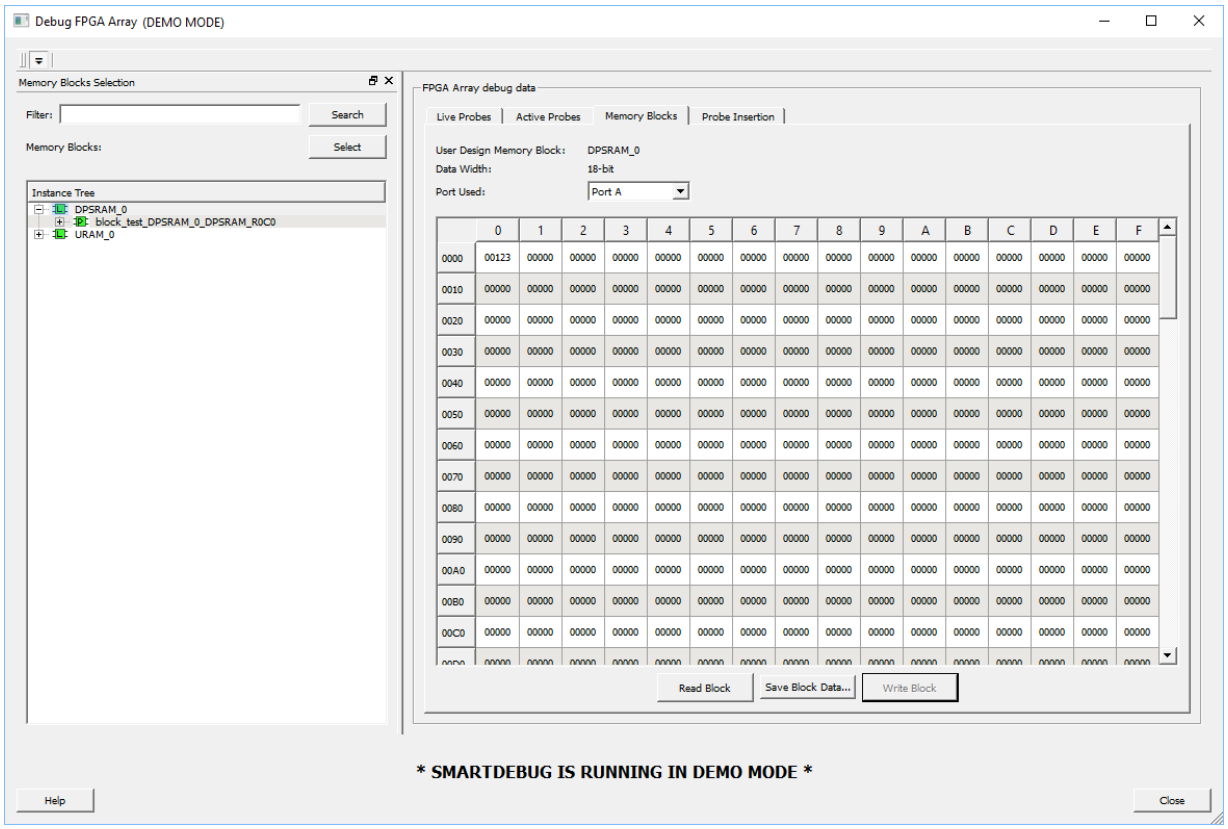


2.2.2.3 Memory Blocks

A temporary memory data file is created in the designer folder for each type of RAM selected. Different data file is created based on the memory type, for example, separate files are created in the µSRAM, LSRAM, and other RAM types in the designer folder. All memory data of all instances of µSRAM, LSRAM, and other RAM types are written to their respective data files. The default value of all memory locations is shown as 0s, and is updated based on your changes. The demo mode supports physical view as well as logical view of RAM blocks (LSRAM and µRAM).

The following figure shows an example of memory blocks in demo mode.

Figure 45 • **Memory Blocks in Demo Mode**



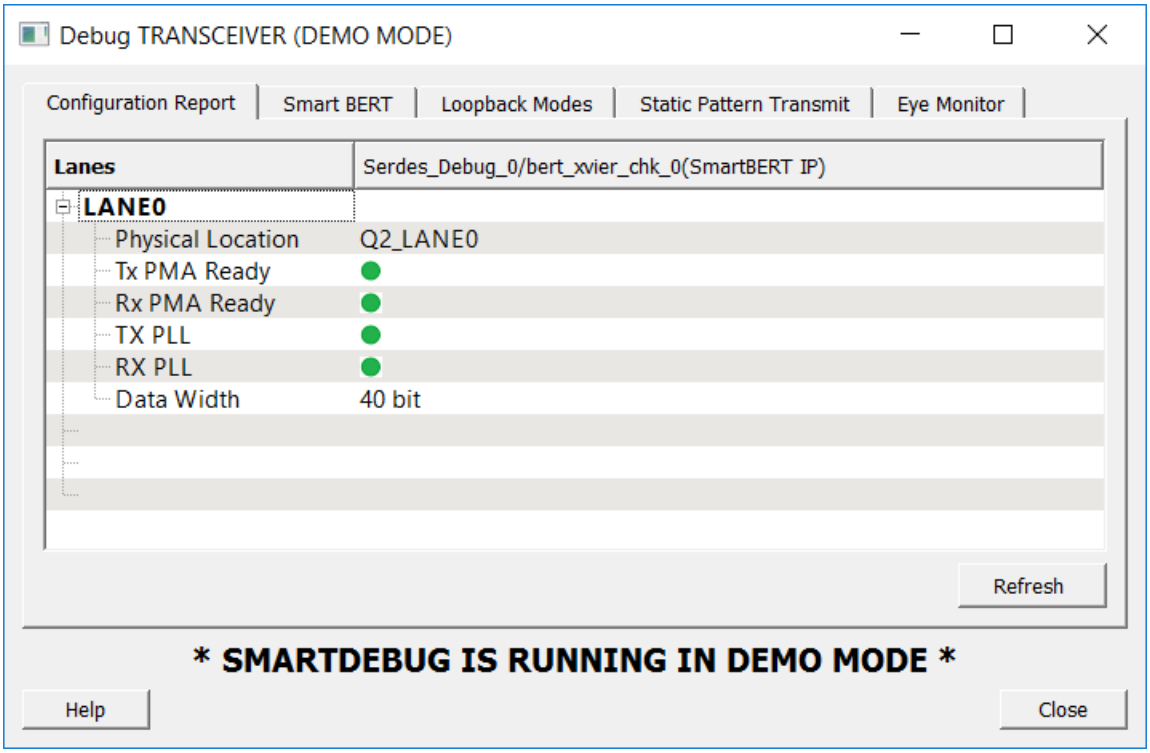
2.2.3 Debug TRANSCEIVER

Transceiver demo mode does not create any temporary data file and is provided to give the user a feel of all the GUI features of Debug Transceiver.

2.2.3.1 Configuration Report

The following figure shows the configuration report of the debug transceiver.

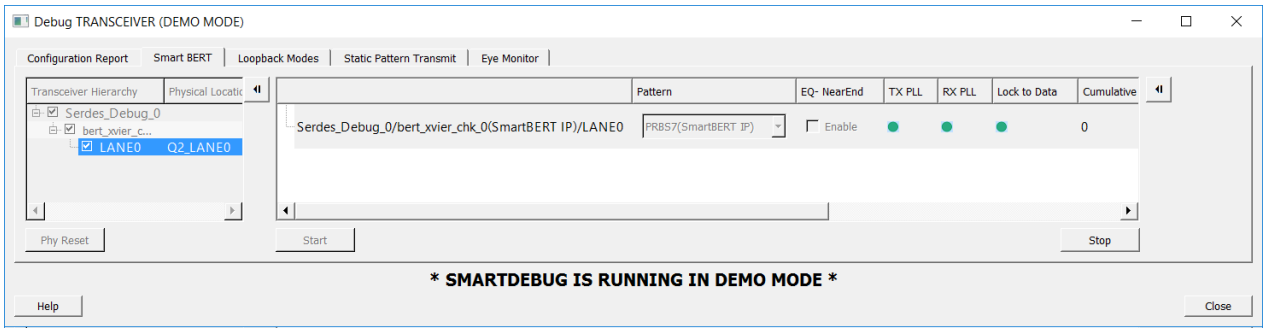
Figure 46 • **Debug TRANSCEIVER—Configuration Report**



2.2.3.2 SmartBERT

The following figure shows the SmartBERT options of the debug transceiver.

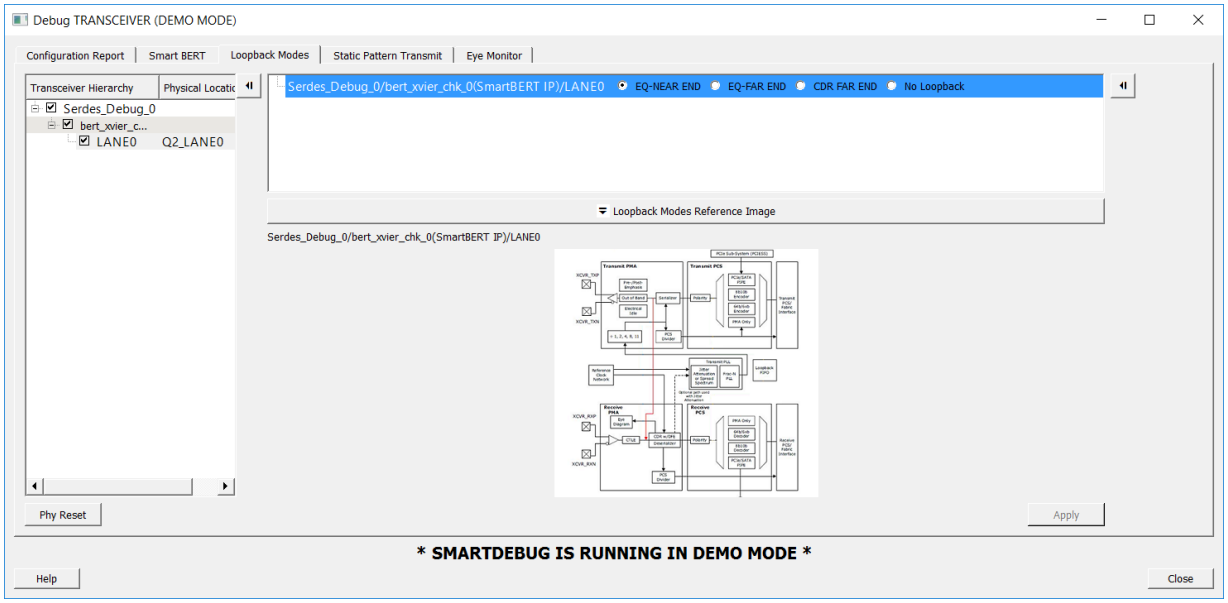
Figure 47 • **Debug TRANSCEIVER—SmartBERT**



2.2.3.3 Loopback Modes

The following figure shows the loopback modes of the debug transceiver.

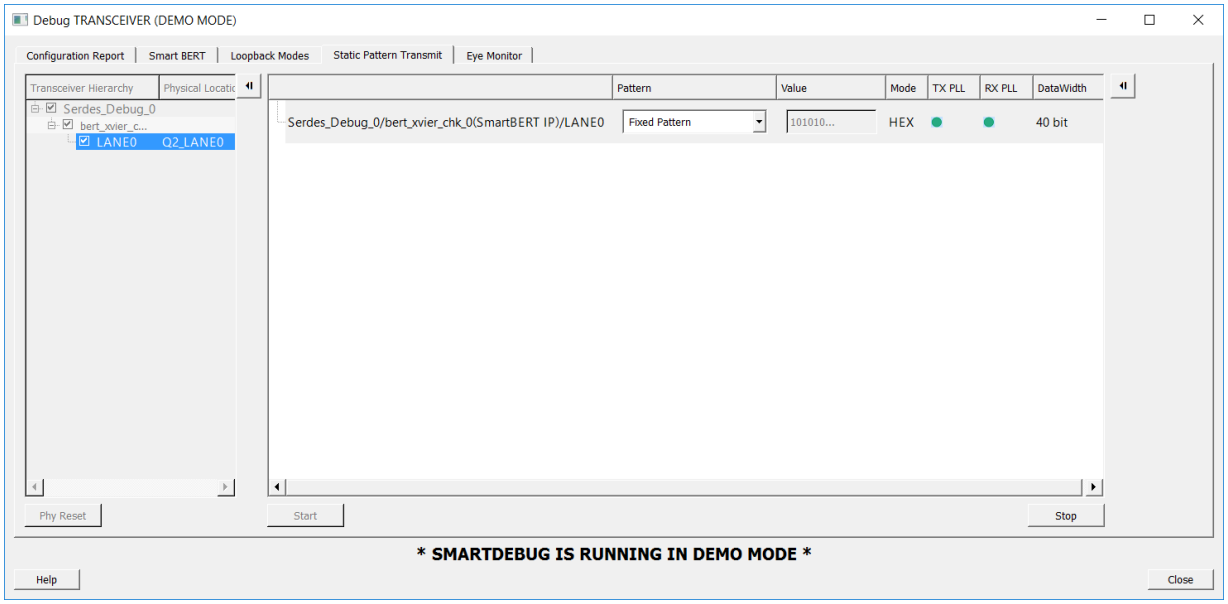
Figure 48 • **Debug TRANSCEIVER—Loopback Modes**



2.2.3.4 Static Pattern Transmit

The following figure shows the Static Pattern Transmit of the debug transceiver.

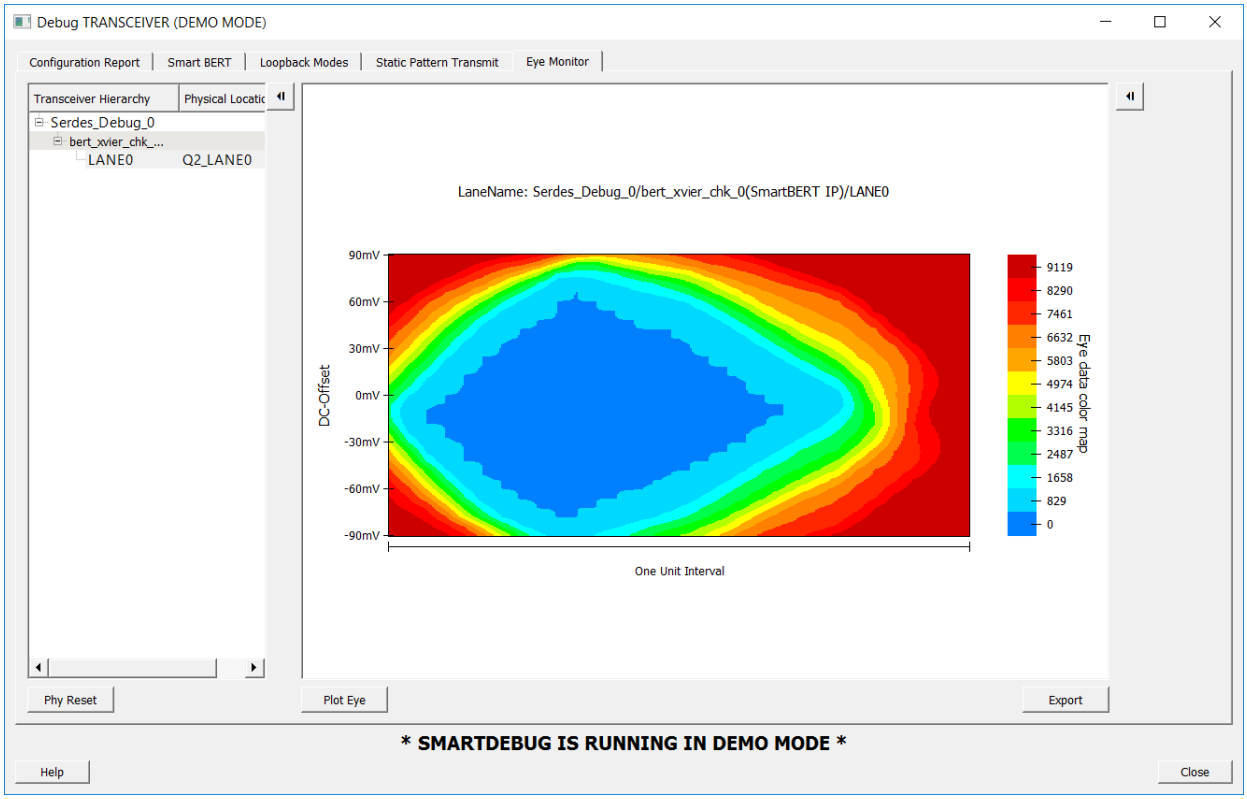
Figure 49 • **Debug TRANSCEIVER—Static Pattern Transmit**



2.2.3.5 Eye Monitor

The following figure shows the Eye Monitor of the debug transceiver.

Figure 50 • **Debug TRANSCEIVER—Eye Monitor**



2.3 DDR Debug

DDR Debug enables reading the key DDR registers to assess the configuration and status of the DDR controller.

Note: This feature is currently not supported and will be provided in future releases.

2.4 Identify

Identify embeds a logic analyzer. This logic analyzer functionality is embedded in the design using the FPGA fabric and embedded memory blocks. This is triggered when:

- The hardware state machine enters a certain stage
- The critical RTL signals are set to invalid values
- The input pin goes high or low

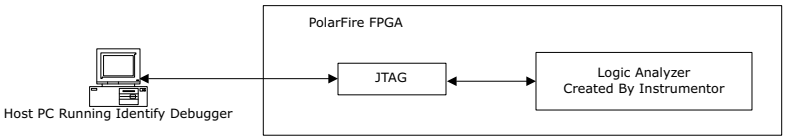
When triggered, Identify selects and captures complex signal interactions. The trace data of the RTL signal before, after, or around the trigger event is stored in the on-chip, or fabric memory. The trigger event normally reflects a hardware error condition and may be calculated as a combinatorial function of several signals in the RTL.

To simplify interface requirements, the logic analyzer is accessed through the standard JTAG port for control and data transfer. The captured data can be displayed on a PC using common viewing software, and typically mirrors a logic simulator waveform output.

Identify adds and configures a logic analyzer to monitor RTL signals within the FPGA design. This tool lets you debug the hardware at the HDL level in the target environment. It is a dual-component system and part of the HDL design flow process. It can be integrated easily with your existing Libero SoC PolarFire design flow. Minor modifications are required for this integration.

The following figure shows the interface between the host PC running the identify Debugger and logic analyzer created in the FPGA fabric.

Figure 51 • **Host PC-Identify Interface**



Using Identify, the following activities can be performed in the HDL code:

- Activate breakpoints
- Set watch-points
- View captured data related to the original source code or as waveforms
- Set trigger conditions that determine when to capture signal data
  A trigger condition is a set of on-chip signal values or events. When a trigger condition occurs, data is transferred from the hardware device to Identify RTL through the JTAG communication cable.

Identify has the following advantages:

- Additional FPGA I/O pins are not required, only standard JTAG signals are used.
- Identify is integrated with the Libero SoC PolarFire design flow.

2.4.1 System Components

The Identify system consists of the following components:

- IICE, page 41
- Identify Instrumentor, page 41
- Identify Debugger, page 42

2.4.1.1 IICE

The Intelligent In-Circuit Emulator (IICE) is a custom block inserted into the design. It is connected to signals in the design by the Identify Instrumentor according to the interface specifications. The IICE block samples internal signals and feeds this information back to the Identify Debugger, where the data is transformed for interpretation at the HDL level.

The IICE block comprises of the Probe block and the Controller block. The probe block samples internal signal data and communicates with the controller block. It contains the sample buffer where signal value data is stored. The probe block also contains the trigger logic that determines when signal data is stored in the sample buffer.

The controller block receives sample data from the probe block and sends to Identify Debugger through the JTAG port.

2.4.1.2 Identify Instrumentor

Identify Instrumentor reads and analyzes the pre-synthesis HDL design and provides detailed information about the signals that can be observed. Based on this analysis, Identify Instrumentor enables you to specify how to control signal observation.

Identify Instrumentor uses HDL design files and user-selection information to create a custom logic analyzer block. It connects the logic analyzer to the appropriate signals in the design.

Identify Instrumentor enables the following:

- Selecting the required design signals at the HDL level
- Creating an on-chip logic analyzer to give access to the Identify Debugger

2.4.1.3 Identify Debugger

Identify Debugger lets you interact with the debug-enabled hardware at the HDL level.

Using Identify Debugger, trigger conditions are set that determine when to capture the signal data. A trigger condition is a set of on-chip signal values or events. When triggered, data is transferred from the hardware device to the Identify Debugger through the JTAG communication cable.

Identify Debugger enables the following:

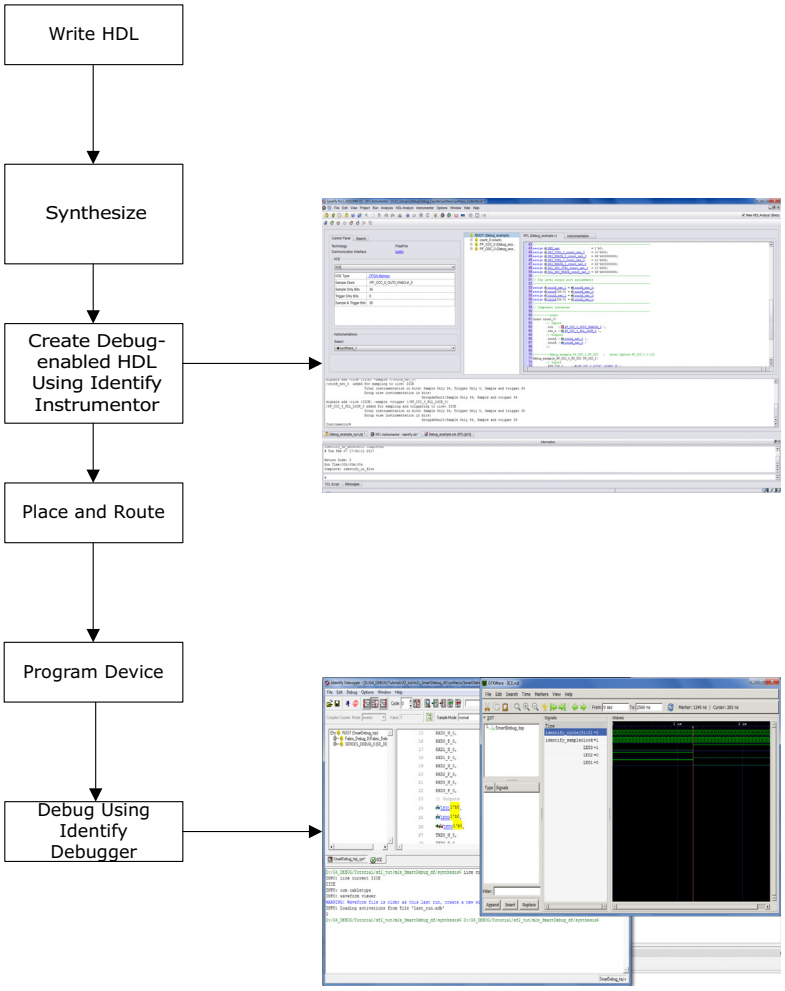
- Interacting with the on-chip logic analyzer
- Getting information about the required signals in the design

With these components, Identify debugs your design faster, easier, and more efficiently.

2.4.2 Libero Design Flow With Identify

This section describes the Libero SoC PolarFire design flow with Identify.The following figure shows the Libero SoC PolarFire design flow with the Identify system components.

Figure 52 • **Libero Design Flow With Identify**



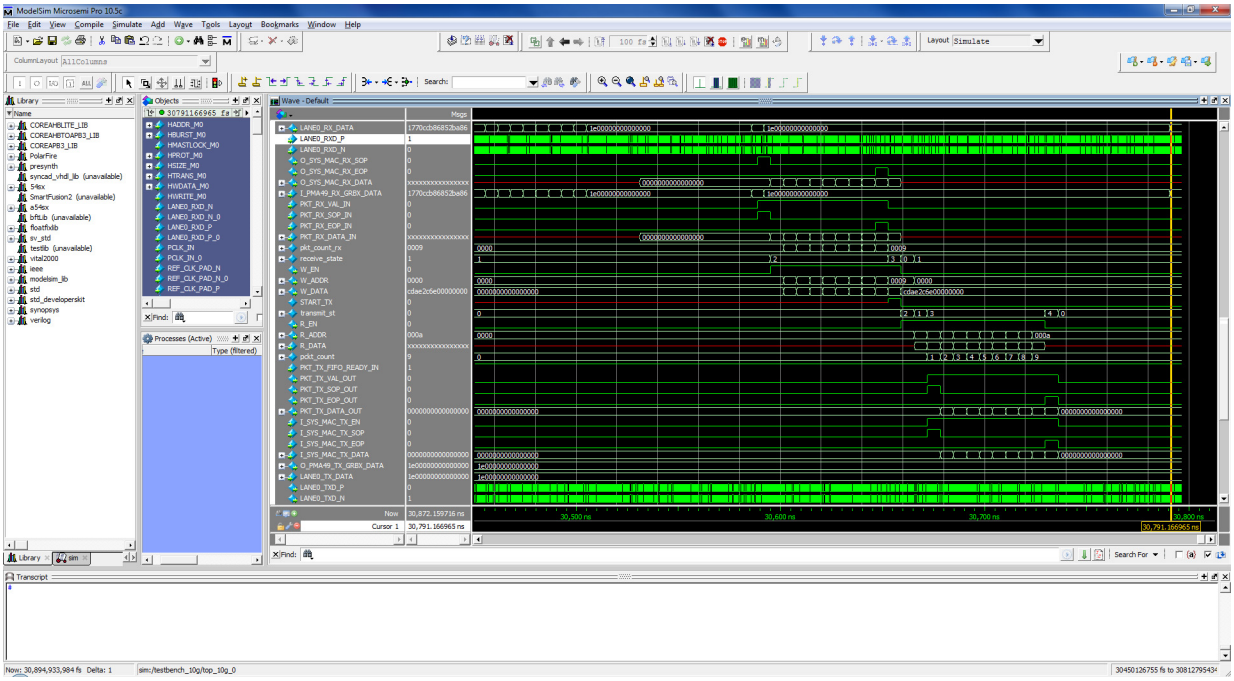
The Libero SoC PolarFire design flow with Identify comprises the following steps:

1. Create the HDL design.
2. Synthesize the design to the target device.
3. Use Identify Instrumentor to create a debug-enabled HDL design.
4. Place and route the debug-enabled design on the target device.
5. Implement the debug-enabled design on the device by programming it.
6. Use Identify Debugger to debug the design while it is running on the target device.

2.5 ModelSim

ModelSim enables debugging of the FPGA design. In ModelSim design simulation, all design components are modeled mathematically as software processes that are executed sequentially to represent the functional, logical, or timing information of the design. A wide-range of stimuli are applied to the design and the expected output is checked against the design requirements. A typical ModelSim window is shown in the following figure.

Figure 53 • **ModelSim Window**



Simulation tools (like ModelSim) have the following advantages:

- Simulation is performed using software and testbench stimuli to represent the design requirements. Design hardware is not required.
- Simulators with testbenches catch many design errors including:
  - Incorrect specifications
  - Incorrect Interface requirements
  - Function errors
  - Other gross errors detected by stimulus vectors

Simulation is particularly effective when extensive stimulus combinations are used, and when the results are well known. In these cases, simulation can do an exhaustive test of a design.

Simulation tools have the following disadvantages:

- Most designs do not have easy access to extensive test suites and creating them can be time consuming and often impossible for large FPGA designs.
- Execution is slow, when many iterations are involved, and rendering them is time-consuming and expensive during the development process.

3 Debugging Examples

This section explains the uses of different tools for debugging the following peripherals:

- Fabric
- LSRAM/µSRAM
- Transceiver
- DDR Memories
- Coretex-M1 based embedded system

Note: More information will be provided in future releases.

4 Board Design Recommendations For Probes

This section describes the board design recommendations for board level debugging and the guidelines for hardware debugging.

Note: More information will be provided in future releases.