AC479
Application Note
Debugging PolarFire FPGA Using SmartDebug

a Microchip company

a MICROCHIP company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), 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. Learn more at www.microsemi.com.

51900479. 8.0 10/21

Contents

Figures

Tables

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 8.0

The following is a summary of the changes made in this revision.

- Updated for Libero SoC v2021.2.
- Updated Design Requirements, page 3.
- Added DDR Interface, page 6 in Demo Design, page 4.
- Updated Clocking Structure, page 8 and Reset Structure, page 8.
- Updated Enabling FPGA Hardware Breakpoint (FHB), page 9.
- Updated Programming the Device, page 9.
- Added information about applying eye mask on plotted eye diagram, see Eye Monitor, page 29.
- Added Register Access, page 31 and Start Record Actions, page 32.
- Added Debug DDR IO Margin, page 36.

1.2 Revision 7.0

Added Appendix 3: Running the TCL Script, page 42.

1.3 Revision 6.0

The following is a summary of the changes made in this revision.

- Updated the document for Libero SoC v12.2.
- Removed the references to Libero version numbers.

1.4 Revision 5.0

The following is a summary of the changes made in this revision.

- Updated the document for Libero® SoC v12.0
- The new FHB feature of SmartDebug was added, see Using FHB, page 17.
- Updated the supported Eye Scan Modes in Eye Monitor, page 29.

1.5 Revision 4.0

The following is a summary of the changes made in this revision.

- Converted this document from a tutorial (TU0804) to an application note (AC479).
- Updated the document for both Evaluation and SPLASH kits.
- Included the information from UG0743: PolarFire FPGA Debug User Guide.
- Updated the document for Libero® SoC PolarFire v2.3.

1.6 Revision 3.0

The document was updated for Libero SoC PolarFire v2.2.

1.7 Revision 2.0

The document was updated for Libero SoC PolarFire v2.1.

1.8 Revision 1.0

The first publication of this document.

2 Debugging PolarFire FPGA Designs Using SmartDebug

Design debug is a critical phase of the FPGA design flow. SmartDebug enables the debugging of designs by providing verification and troubleshooting features at the hardware level. It provides access to probe points, Non-Volatile Memory (NVM), fabric and fabric RAM blocks, transceivers, and the DDR controller. These features enable designers to check the state of inputs and outputs in real-time, without any design modification. For more information about SmartDebug features, see *PolarFire SmartDebug User Guide*.

This application note provides a demo design to demonstrate how SmartDebug is used for debugging Transceiver, DDR Memory, and Dual-Port SRAM (DPSRAM) in a PolarFire FPGA design.

2.1 Design Requirements

The following table lists the hardware and software requirements for this demo design.

Table 1 • **Design Requirements**

Requirement	Description
Operating system	64-bit Windows 7 or 10
Hardware	
PolarFire Evaluation Kit (MPF300T-1FCG1152I)	Rev D or later
2 SMA-to-SMA cables with 5 Gbps support	Only for Evaluation Kit
Software	
Libero® SoC	**Note:** Refer to the readme.txt file provided in the design files for the software versions used with this reference design.

Note: Libero SmartDesign and configuration screen shots shown in this guide are for illustration purpose only. Open the Libero design to see the latest updates.

2.2 Prerequisites

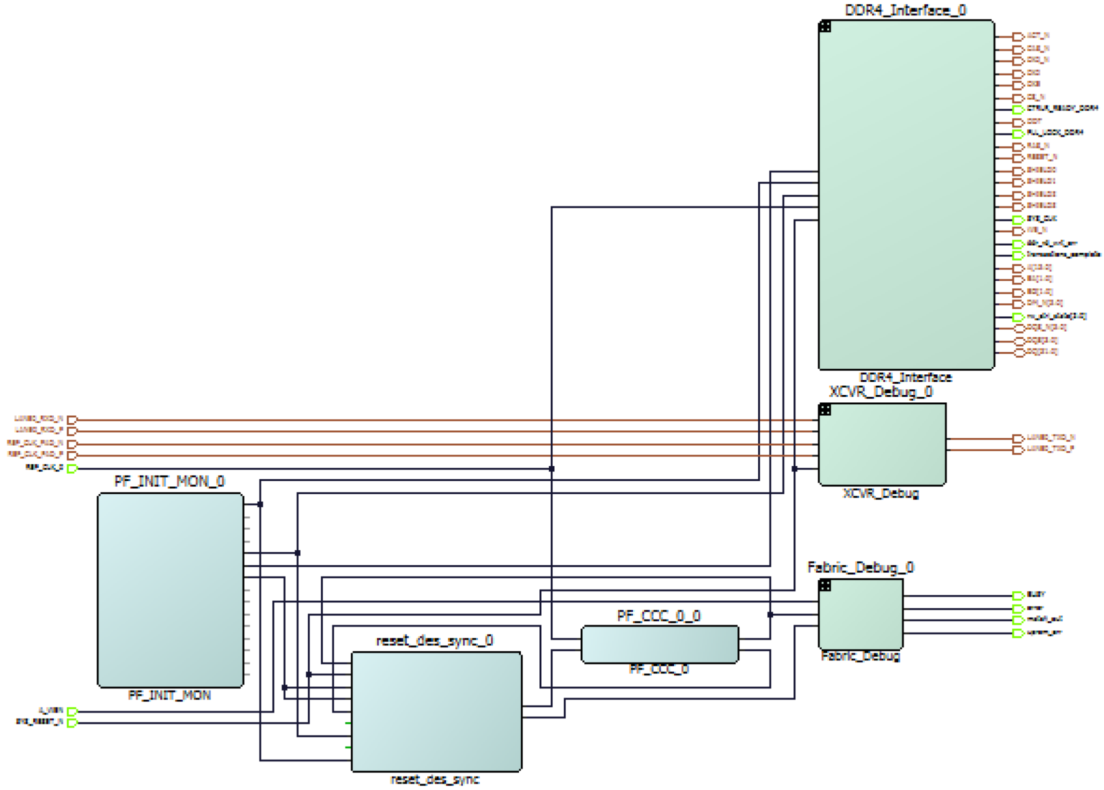
Before you begin:

1. Download and install Libero SoC (as indicated in the website for this design) from the following location:
   https://www.microsemi.com/product-directory/design-resources/1750-libero-soc#downloads
2. Demo design files download link:
   - http://soc.microsemi.com/download/rsc/?f=mpf_ac479_eval_df

2.3 Demo Design

This section describes the fabric, DDR interface, and XCVR design blocks implemented in Libero SoC.

Figure 1 • **SmartDebug Top-Level Blocks**



Note: The FHB feature is not enabled in the demo design. To enable FHB debugging, provide the "FHB_ENABLE" argument with the given tcl script. This creates the design without the DDR controller block because FHB debugging is currently not supported for designs that include DDR controller. For more information about enabling FHB debugging using the tcl script, see Appendix 3: Running the TCL Script, page 42.

The top level block contains the following blocks:

1. PF_CCC, page 4
2. PF_INIT_MON, page 4
3. reset_des_sync, page 4
4. XCVR_Debug, page 5
5. Fabric_Debug, page 5
6. DDR Interface, page 6

2.3.1 PF_CCC

The PF_CCC block generates 125 MHz clock. Fabric_Debug logic works on this clock.

2.3.2 PF_INIT_MON

The PF_INIT_MON block checks the status of device initialization. When the initialization of SRAM and µPROM is completed, the IP asserts DEVICE_INIT_DONE signal.

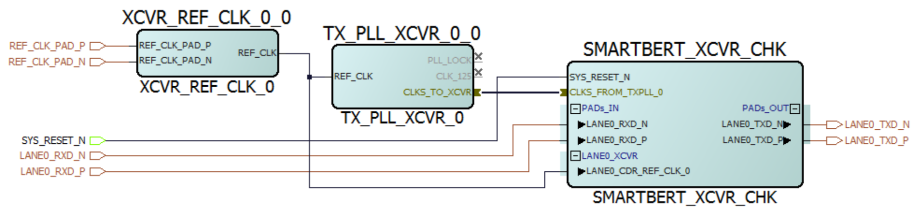
2.3.3 reset_des_sync

The reset_des_sync_0 block is an instantiation of CoreRESET_PF IP. It synchronizes the de-assertion of asynchronous reset.

2.3.4 XCVR_Debug

Figure 2 shows the IP blocks inside the XCVR_Debug block. The XCVR_Debug block demonstrates SmartDebug's real-time Signal Integrity (SI) testing and debugging capabilities to test and debug the PolarFire transceiver. The XCVR_Debug block contains CoreSmartBERT, TX_PLL, and XCVR_REF_CLK IP cores. CoreSmartBERT implements the PolarFire transceiver in the PMA mode.

Figure 2 • **XCVR_Debug Overall Design Blocks**

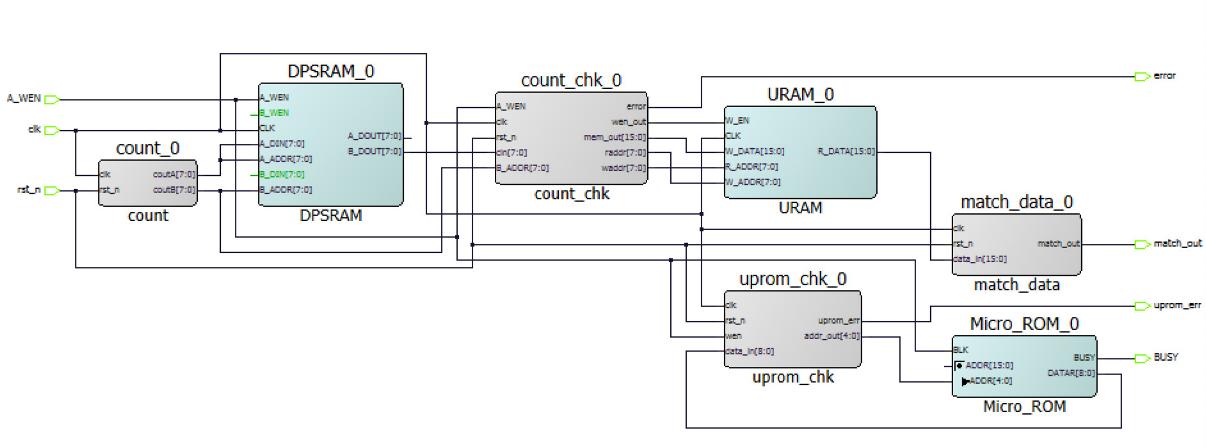


Note: SmartBERT includes the PolarFire Transceiver, which interfaces to the SmartDEBUG tool through a user control GUI to run the hardened PRBS generator and checkers. It also has fabric pattern generators and checks with more features like error injection.

2.3.5 Fabric_Debug

Figure 3 shows the IP blocks inside the Fabric_Debug block.

Figure 3 • **Fabric_Debug Overall Design Blocks**



The Fabric_Debug block demonstrates the following FPGA fabric debug features of SmartDebug.

- FPGA array debugging capabilities using a counter that loads a counting pattern into the DPSRAM instance. The data value of the DPSRAM block is the same as the address value of the block. On the read side of the DPSRAM, a count checker (count_chk) ensures that the count progresses as expected. If there is an error, the output (error) is driven high.
- µPROM debugging feature of SmartDebug using a µPROM instance.
- Live probes to monitor an internal user-selected point on the device in real time, and how to set active probes for dynamic asynchronous read and write to a flip-flop or probe point. These features help to quickly observe the output of the logic internally or quickly experiment to determine how the logic is affected by writes to a probe point.
- Capabilities to read and modify fabric SRAM content in real-time.

µPROM: This is the embedded non-volatile PROM arranged in a single row at the bottom of the fabric and is read only through the fabric interface. µPROM is programmed with the FPGA bitstream during fabric programming. µPROM is used to store the initialization data for DPSRAM and µSRAM and other user data. µPROM is initiated with the `uprom.mem` file.

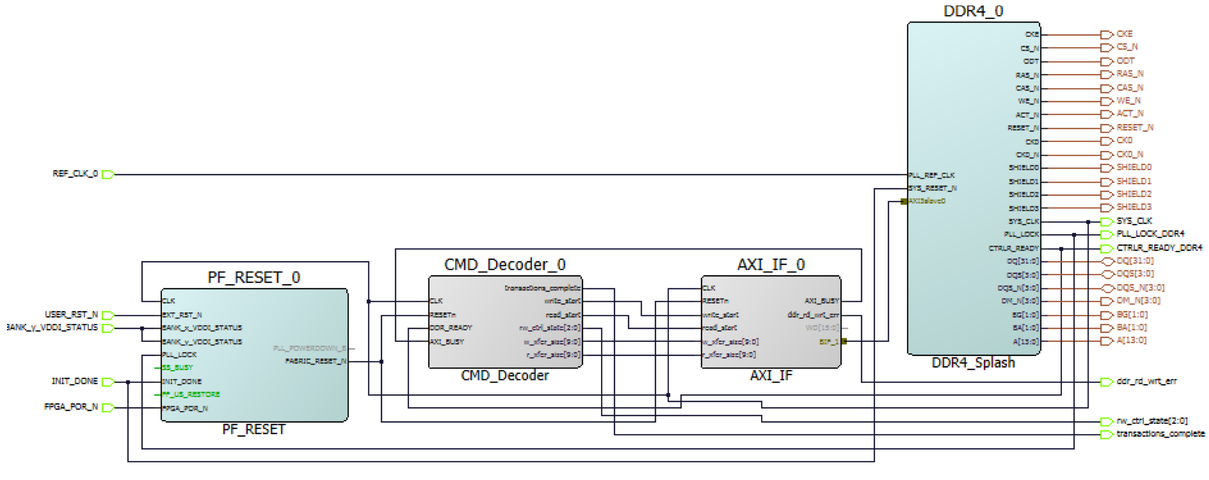
µSRAM: This is the fabric RAM block that is accessed using the PF_SRAM_AHBL_AXI IP. Generally, µSRAM is initialized with a user application executable at device power-up. In the example design, µSRAM is initialized with the `sram.hex` file.

2.3.6 DDR Interface

The DDR_Interface block demonstrates Debug DDR IO Margin feature in SmartDebug, select the **Debug DDR Memory** option in the main SmartDebug window. **Debug DDR Memory** is available only for DDR3/DDR4/LPDDR3 memory configurations on PolarFire and PolarFire SoC devices. This option is not visible when DDR memory is not used in the design.

Note: For detailed information on DDR Debug, see *PolarFire SmartDebug User Guide*.

Figure 4 • **DDR Interface Overall Blocks**

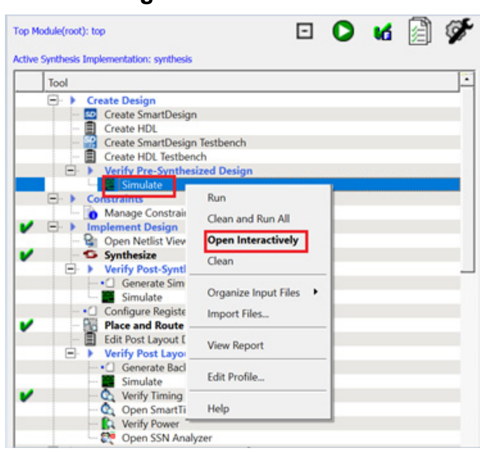


2.3.6.1 Simulation Using Micron DDR4 SDRAM Model

Follow these steps to simulate the DDR4 Model:

1. The PolarFire Evaluation Kit features the DDR4 SDRAM from Micron with the part number MT40A1G8WE083E. The DDR4 simulation model files are available at the design files path \<Libero project directory>\stimulus.
2. For running simulation, select **Simulate** from **Design Flow** > **Verify Pre-Synthesized Design** as shown in the following figure.

Figure 5 • **Simulating Pre-Synthesized Design**



The AXI_IF block initiates 1K reads and writes to DDR4 memory via the CMD_Decoder block. The following figures show the simulation waveforms.

Figure 6 • **AXI Master signal write operation**

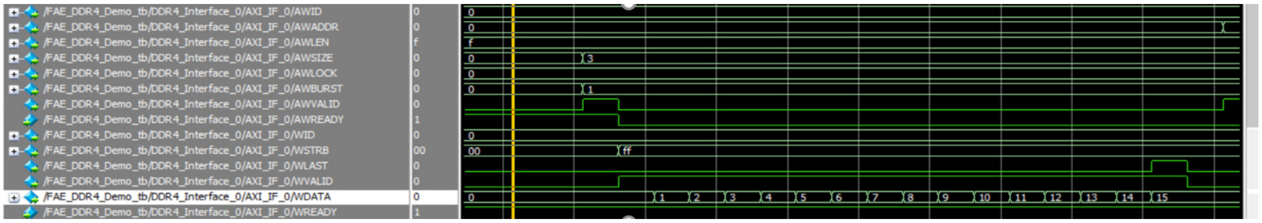


Figure 7 • **DDR4 signals**

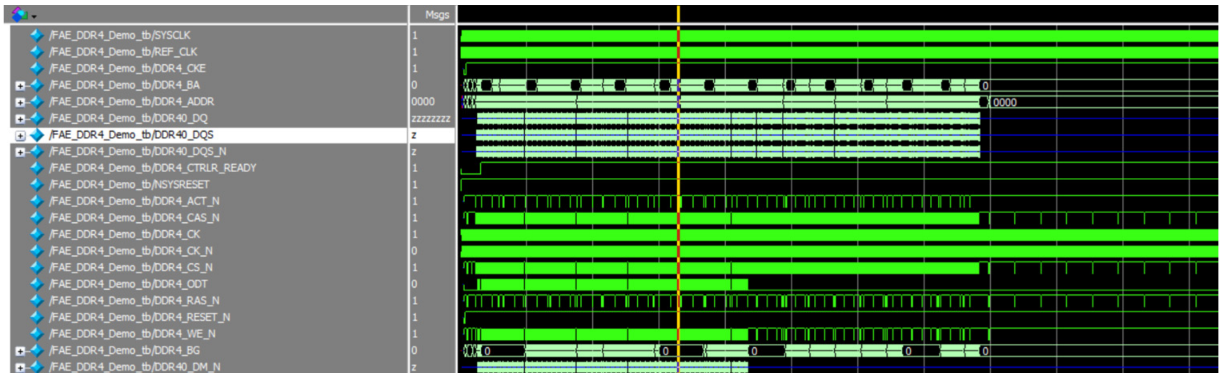
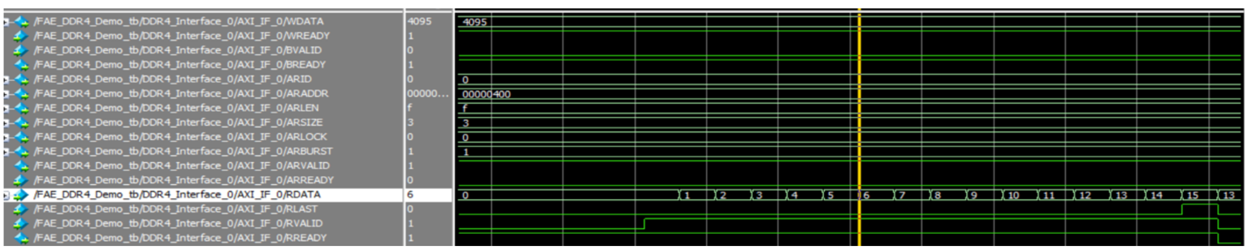


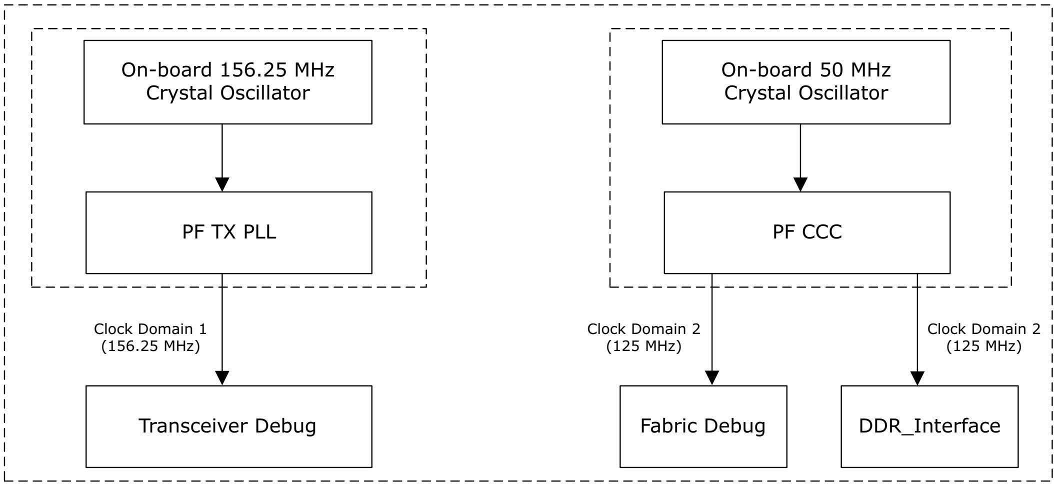
Figure 8 • **AXI Master signals read operation**



2.4 Clocking Structure

The reference design has two clock domains. As shown in the following illustration, clock domain 1, used for transceiver debug, runs at 156.25 MHz, and clock domain 2, used for fabric debug, runs at 125 MHz.

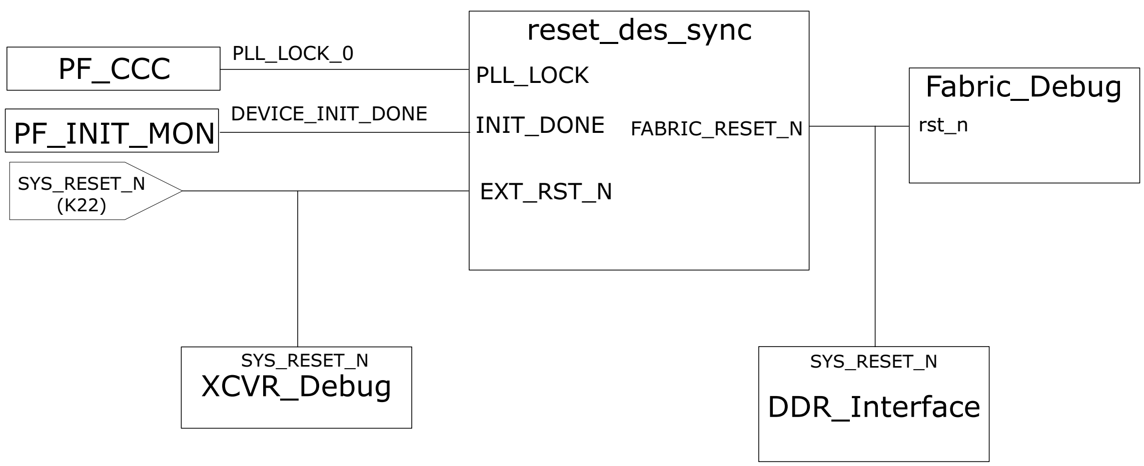
Figure 9 • **Clocking Structure**



2.5 Reset Structure

Figure 10 shows the reset structure used in the design.

Figure 10 • **Reset Structure**



2.6 Enabling FPGA Hardware Breakpoint (FHB)

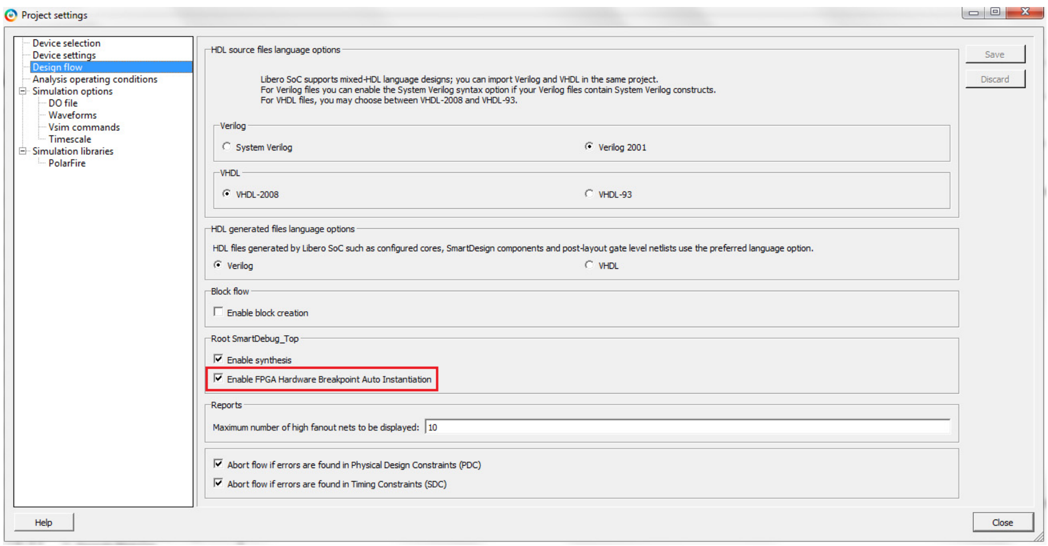
Using tcl script, a Libero SoC project with FHB enabled can be created. A tcl script is provided in the design files folder under the TCL_Scripts directory for creating a Libero SoC Project with FHB enabled.

Follow these steps to create the Libero project with FHB enabled:

1. Launch Libero SoC.
2. Select **Project** > **Execute Script....**
3. Select **Browse** and then select `script.tcl` from the downloaded **TCL_Scripts** directory.
4. In the **Argument** tab, provide the `FHB_ENABLE` argument.
5. Click **Run**.

After successful execution of the TCL script, Libero SoC project is created within TCL_Scripts directory. This can be confirmed by going to **Project > Project Settings** as shown in Figure 11.

Figure 11 • **Enabling FHB**



2.7 Programming the Device

The following steps describe how to program the device on a PolarFire Evaluation Kit.

1. Ensure that the following jumper settings are followed.

Note: Power-down the board before making the jumper connections.

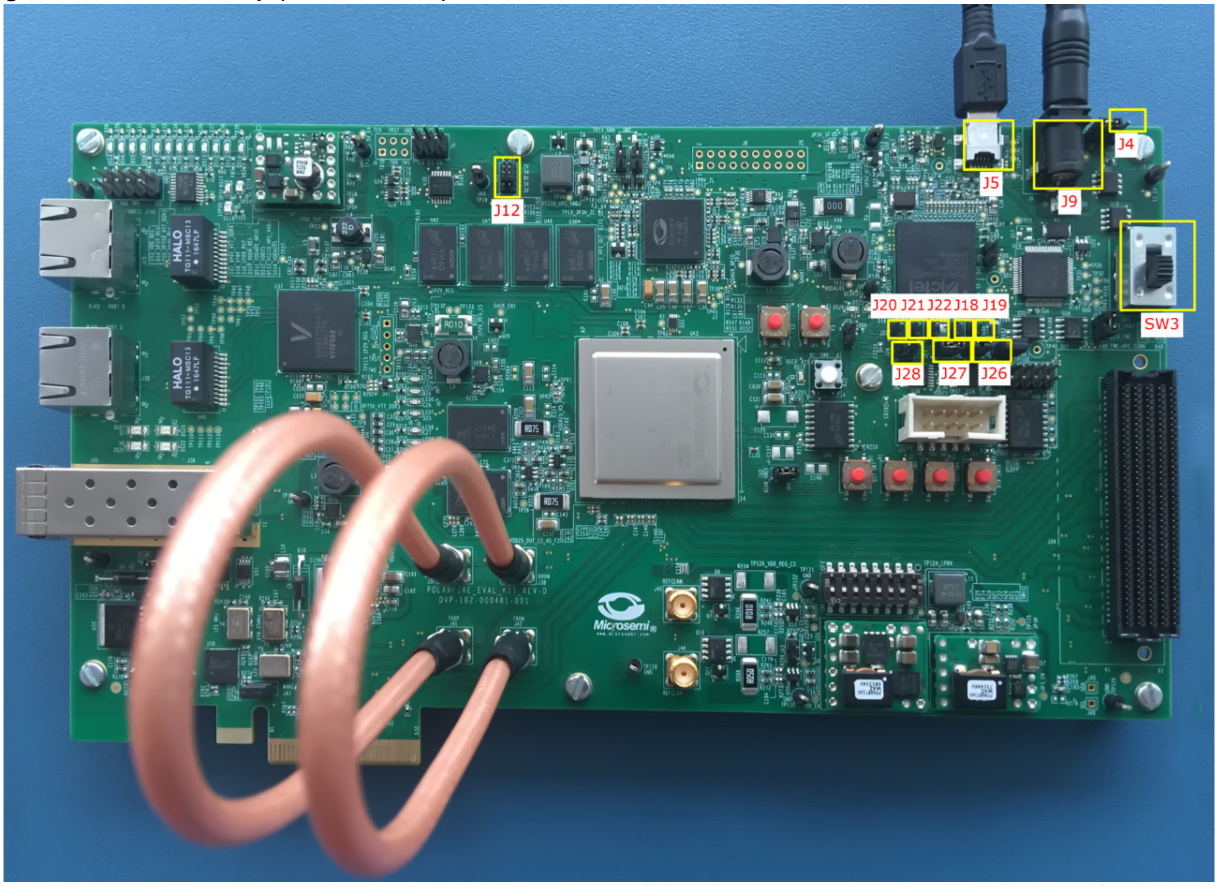
Table 2 • **Jumper Settings For Evaluation Kit**

Jumper	Description
J46	Short pin 1 and 2 for setting the Reference Clock to 125 MHz on-board oscillator
J18, J19, J20, J21, and J22	Short pin 2 and 3 for programming the PolarFire FPGA through FTDI
J28	Short pin 1 and 2 for programming through the on-board FlashPro5
J4	Short pin 1 and 2 for manual power switching using SW3
J17	Short pin 1 and 2
J12	Short pin 3 and 4 for 2.5 V

2. Connect the power supply cable to the **J9** connector on the board.
3. Connect the USB cable from the Host PC to the **J5** (FTDI port) on the board.
4. Power on the board using the **SW3** slide switch.

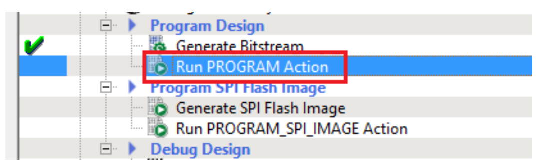
5. Switch OFF the DIP1 switch.
6. Connect **TXN** to **RXN** and **TXP** to **RXP** using 2 SMA to SMA cables as shown in Figure 12. The following figure shows the board setup.

Figure 12 • **Board Setup (Evaluation kit)**



7. In the **Design Flow** window, select **Run PROGRAM Action**, as shown in the following figure. This programs the design into the device.

Figure 13 • **Programming the Device**



2.8 Debugging Using SmartDebug

To debug the device using SmartDebug, follow these steps:

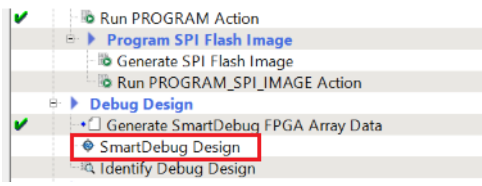
- Launch SmartDebug from Libero, page 11
- View Device Status, page 12
- Debug FPGA Array, page 12
- Using FHB, page 17
- Debug μPROM, page 23
- sNVM Debug, page 23
- Debug TRANSCEIVER, page 25
- Debug DDR IO Margin, page 36

2.8.1 Launch SmartDebug from Libero

On the **Design Flow** window:

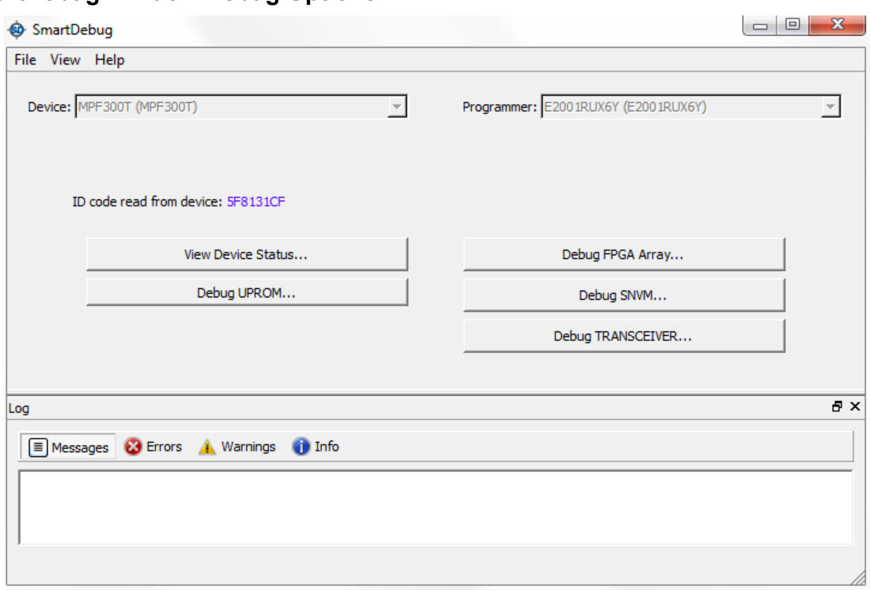
1. Select **Generate SmartDebug FPGA Array Data** to generate data for SmartDebug Design. Once the data is generated, a green tick mark is seen on the left side of the option indicating that the data generation is successful.
2. Open **SmartDebug Design**.

Figure 14 • **Launching SmartDebug Design**



The **SmartDebug** window is displayed, as shown in Figure 15.

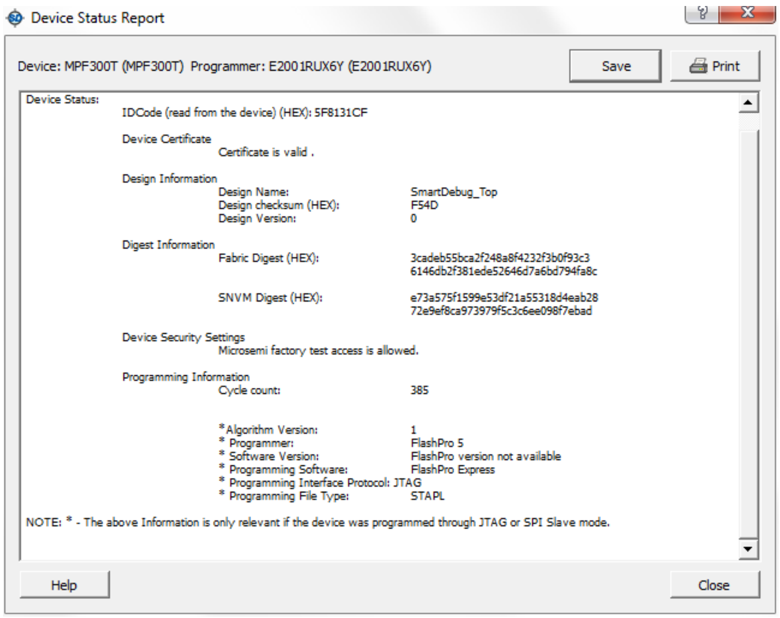
Figure 15 • **SmartDebug Window Debug Options**



2.8.2 View Device Status

The View Device Status option provides the device status report. It summarizes the device information, programmer information, design information, factory serial number, and security information, if any are set. To view the device status report, click **View Device Status** in the **SmartDebug** window. The following figure shows a sample of the device status information.

Figure 16 • **Device Status Report Sample**



2.8.3 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 DPSRAMs/URAMs. Probe insertion allows the 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:

- Live Probes, page 12
- Active Probes, page 13
- Memory Blocks, page 15
- Probe Insertion, page 17

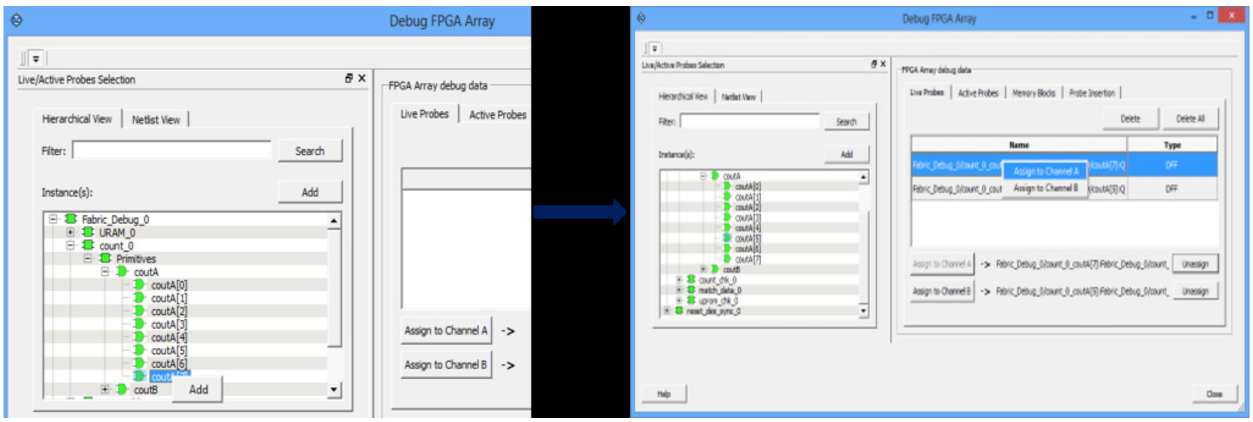
2.8.3.1 Live Probes

Live Probes enables the monitoring of two internal signals at a time in the design without having to repeat the place and route. PolarFire devices have two dedicated live probe channels (for example, pin H6 and G6 of PolarFire MPF300TS device). For more information about Live Probes, see *PolarFire SmartDebug User Guide*.

The following steps explain the procedure of adding probe point to a list:

1. Select the **Live 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 on the selected points and click **Add** to add them to the **Live 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.
4. Select any of the added probes and assign it to either Channel A or Channel B (by clicking on 'Assign to Channel A' or 'Assign to Channel B') as shown in Figure 17.
5. When the assignment is complete, the probe name appears to the right of the button for that channel, and SmartDebug configures the Channel A and Channel B I/Os to monitor the desired probe points.
6. Once the probe points are assigned, the probes can be monitored by connecting the probe points (for example, pin H6 and G6) to the oscilloscope.

Figure 17 • **Debug FPGA Array—Live Probes**

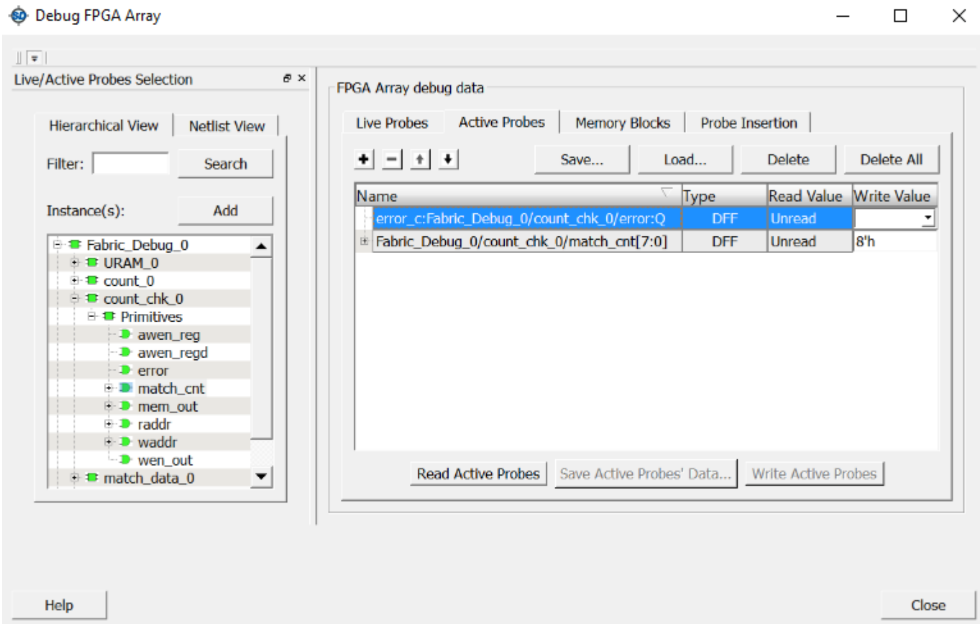


2.8.3.2 Active Probes

Active Probes enable reading or changing the values of probe points in a design through JTAG. Active Probes dynamically and asynchronously read or write to any logic element register bit. Active probes are useful for quick observation of an internal signal. For more information about Active Probes, see *PolarFire SmartDebug User Guide*. To add probe points to a list, perform the following steps:

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.
4. Click **Read Active Probes** to read the content of the registers added to the window.

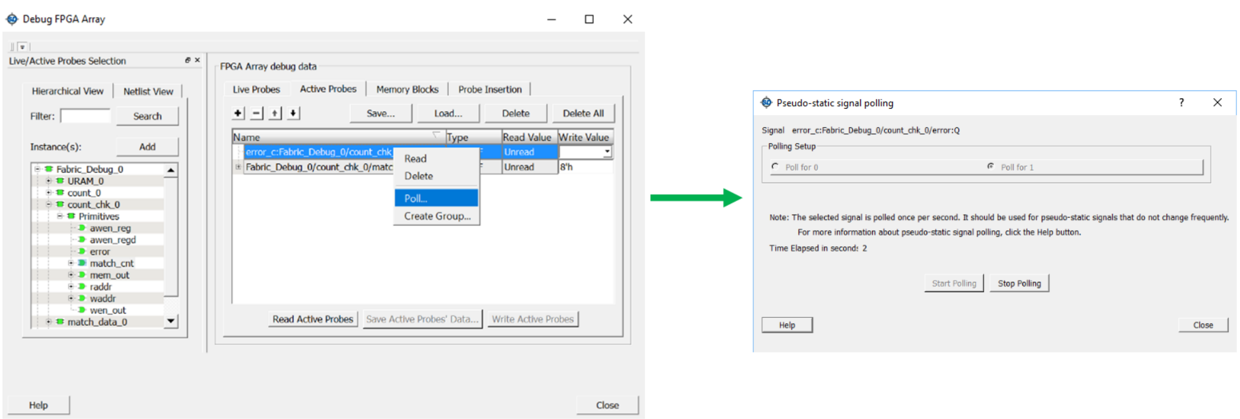
Figure 18 • **Debug FPGA Array—Active Probes**



5. To use pseudo static signal polling, on the **Active Probes** tab, right-click any probe point and select **Poll**, as shown in the following figure.

Static signal polling is used to check whether the logical bit value is changed to expected polled value.

Figure 19 • **Pseudo-static Signal Polling**



2.8.3.3 Memory Blocks

SmartDebug provides the **Memory Blocks** tab to dynamically and asynchronously read from and write to a selected FPGA fabric SRAM block. For more information about the Memory Blocks tab, see *PolarFire SmartDebug User Guide*.Using the **Memory Blocks** tab, user can select the required memory block for performing the following:

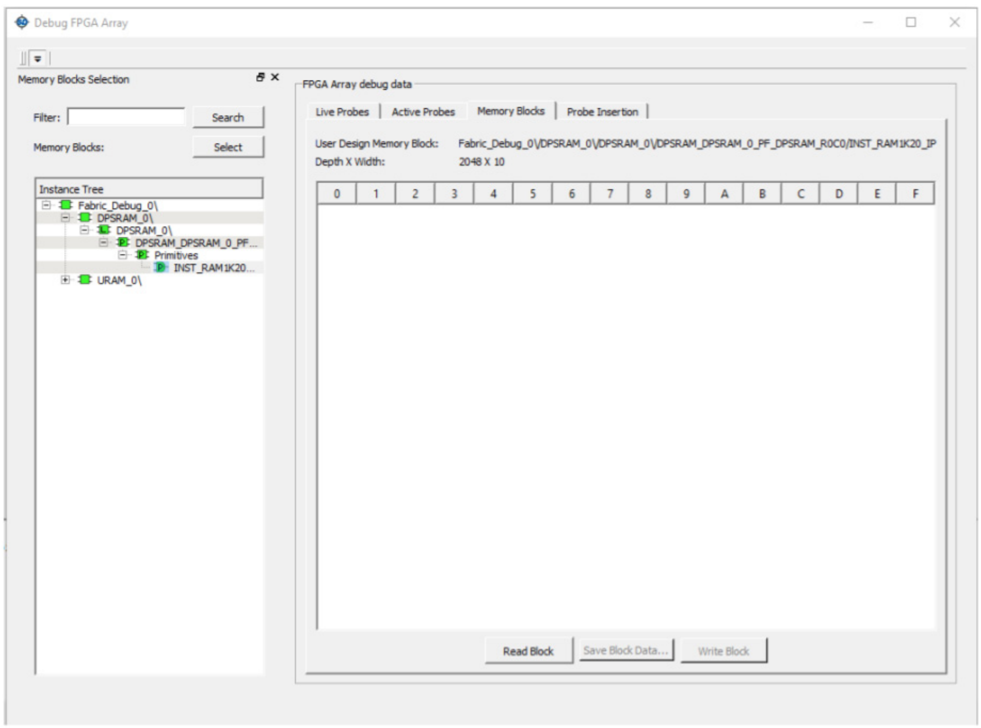
- Reading
- Capturing a snapshot of the memory
- Modifying memory values, and then write the values back to that block

To read and write memory blocks, follow these steps:

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**.

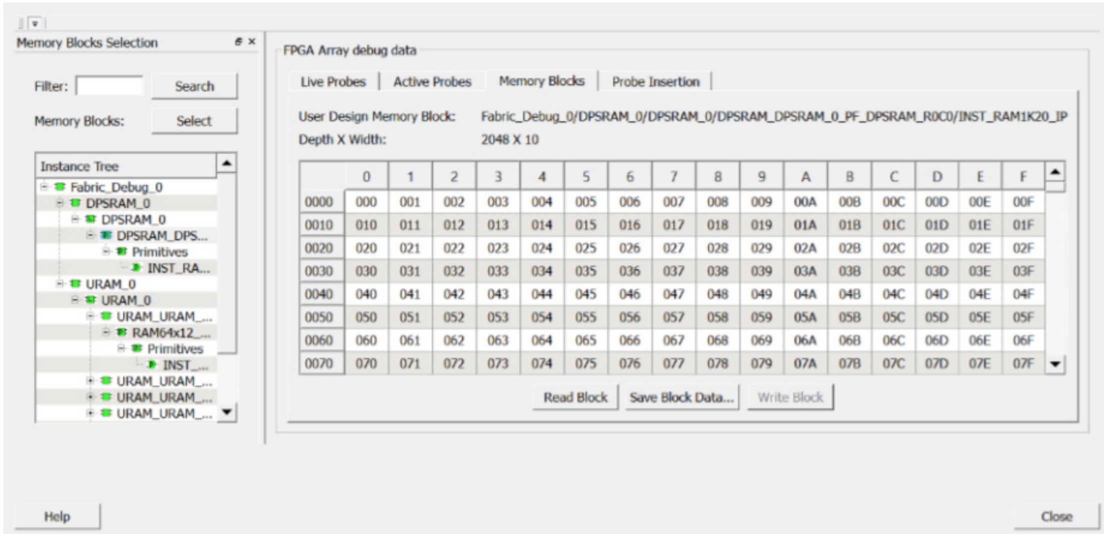
The following figure shows the **Memory Blocks** tab in **Debug FPGA Array** window.

Figure 20 • **Debug FPGA Array—Memory Blocks**



5. Click **Read Block**. The specified memory block is read as shown in the following figure.

Figure 21 • **Memory Blocks—Read Block**

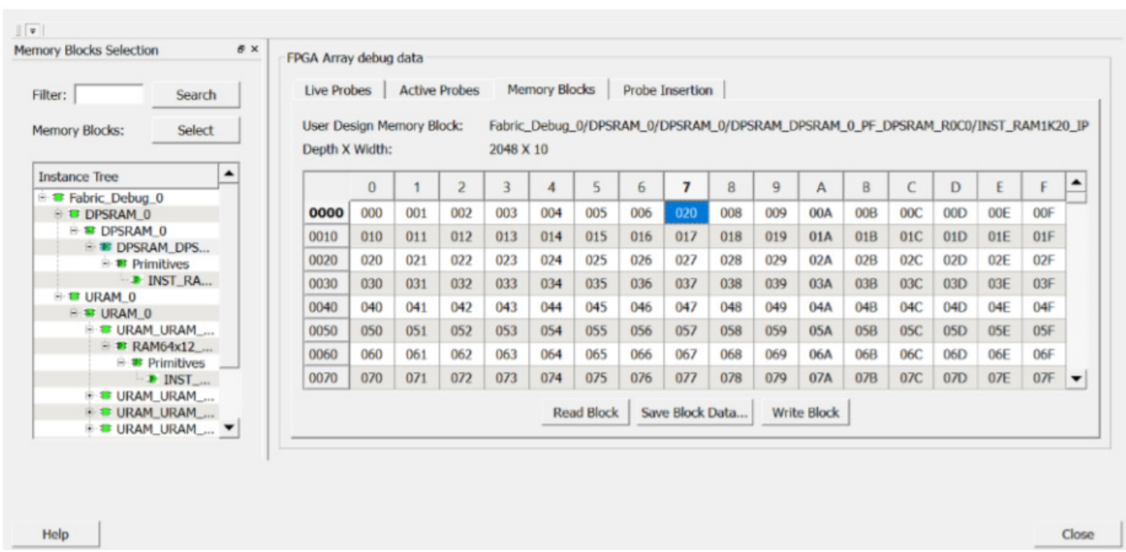


6. Enter a hexadecimal value in the memory block locations and click **Write Block** to write content into memory.

Note: The counter logic writes to SRAM constantly. Before you write to SRAM using SmartDebug, ensure that the A_WEN signal (DIP1 of SW11) is low. This prevents SRAM being overwritten by the counter logic.

7. Switch On DIP1, enter a hexadecimal value in the memory block location(s) and click **Write Block** to write the modified value to the SRAM, as shown in Figure 22.

Figure 22 • **Memory Blocks—Write Block**



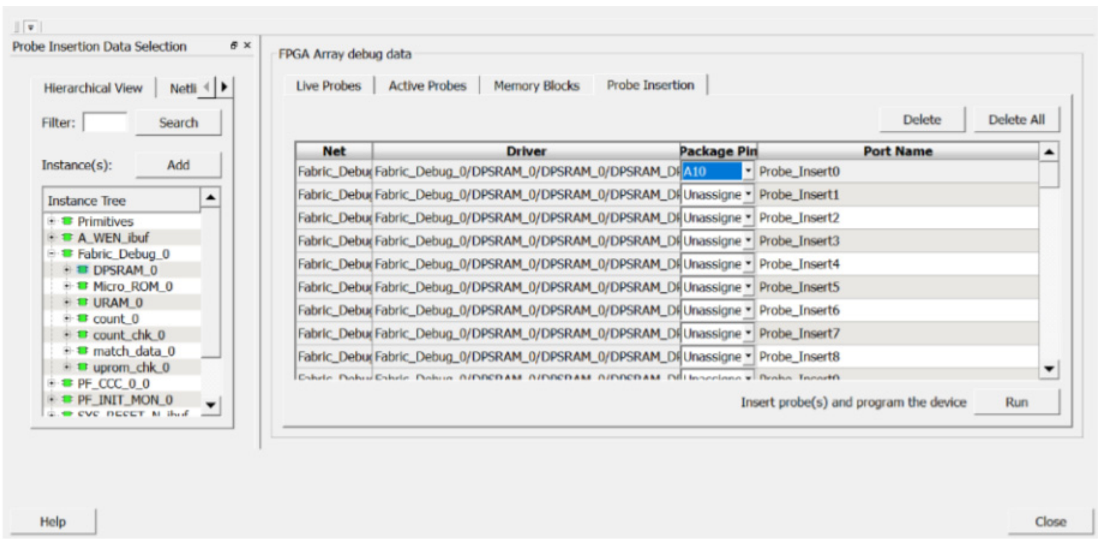
8. The error LED(F22) light turns on, indicating an error in the counting pattern.
9. Go to **Active Probes** tab, read the value of **error** signal, it should show '1'. To use static signal polling, right-click **error_c:Fabric_Debug_0/count_chk_0/error:Q** and select **Poll** (**Poll for 0)**, as shown in Figure 19.

10. Move DIP1 to off state to resume the write operation from the counter to the SRAM. This overwrites the error that was injected into the SRAM. Check the status of LED, it must turn off. Hit the **Poll for 0**, **User value match** message should appear on the polling window. Close the **Pseudo-static signal polling** window.
11. The content of the SRAM can be rechecked by clicking **Read Block** in the **Memory Blocks** tab.

2.8.3.4 Probe Insertion

Probe insertion is a post-layout debug process that enables internal nets in the FPGA design to be routed to unused or used I/Os. Nets are selected and assigned to probes using the **Probe Insertion** tab in SmartDebug. The rerouted design is reprogrammed automatically by Libero into the FPGA, where an external logic analyzer or oscilloscope can be used to view the activity of the probed signal. Figure 23 shows the **Probe Insertion** tab in the **Debug FPGA Array** window.

Figure 23 • **Debug FPGA Array—Probe Insertion**



2.8.4 Using FHB

When FHB is enabled, an FHB instance is created on each clock domain of the design. Each FHB instance gates its associated clock domain. You can add a trigger signal (**countB[0]**) to a live probe and halt the design on the positive edge of the trigger.

When FHB is enabled the following options are enabled on the Live Probes tab:

- **Event Counter**—counts the transition of signals that are assigned to Channel A or Channel B through the Live Probe feature. This feature tracks events from the board. When the Event Counter is activated, and a signal is assigned to Channel A, the counter starts counting the rising edge transitions. The counter must be stopped to get the final signal transition count. During the count, you cannot assign another signal to Channel A/Channel B or go to any other tab on the window.
- **Frequency Monitor**—calculates the frequency of any signal in the design that can be assigned to Live Probe Channel A or Channel B. You can enter the duration of monitoring the signal. The accuracy of results increases as the monitor time increases. The unit of measurement is displayed in MegaHertz (MHz). During the run, progress is displayed.
- **User Clock Frequencies**—shows the clock frequencies from the CCC block.

This section describes the following procedures:

- Use Event Counter, page 18
- Use Frequency Monitor, page 18
- Use User Clock Frequency, page 19
- Select FHB, page 19

2.8.4.1 Use Event Counter

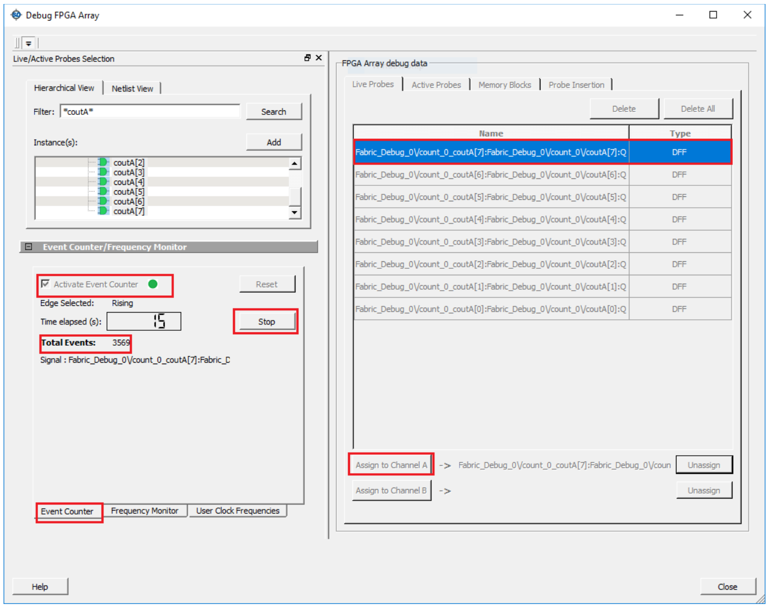
This section describes the procedure to add a trigger signal (**coutA[7]**) to Live Probe, activate the event counter to count the rising edge transitions of that signal.

Follow these steps:

1. Go to the Live Probes tab, and enter ***coutA*** in the filter box, then click **Search**.
2. Select **coutA [7]** to **Cout[0]** and click **Add**, as shown in Figure 24.
   coutA [7] is used as the hardware break point trigger.
3. Select **coutA [7]: Q**, then click **Assign to Channel A**.
4. Click **Activate Event Counter**.
5. The count is updated every second, and is displayed as **Total Events**.
6. Click **Stop** button to stop counting as shown in Figure 24.

Note: When Event Counter is running, only the Stop button is enabled.

Figure 24 • **Event Counter**

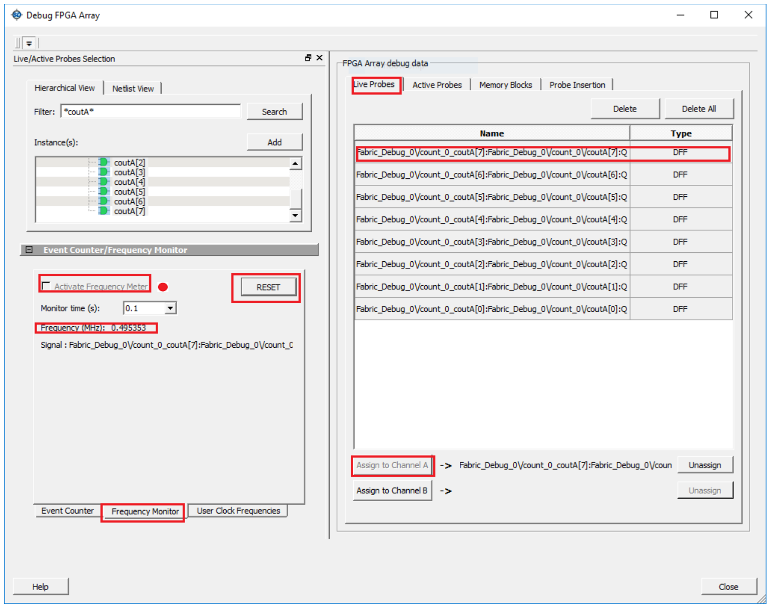


2.8.4.2 Use Frequency Monitor

To use Frequency Monitor, perform the following steps:

1. Click **Live Probe** tab and assign a signal to Channel A, and then click **Frequency Monitor** tab.
2. Set 0.1 as Monitor Time(s) and select the Activate Frequency Monitor check box.
3. The Frequency Monitor stops when the specified monitor time is over. The result is displayed as Frequency (MHz). The window and the tabs on the control panel are enabled. The Reset button is also enabled to reset the Frequency to 0 to start over the next iteration. The progress bar is hidden when the Frequency Monitor stops.

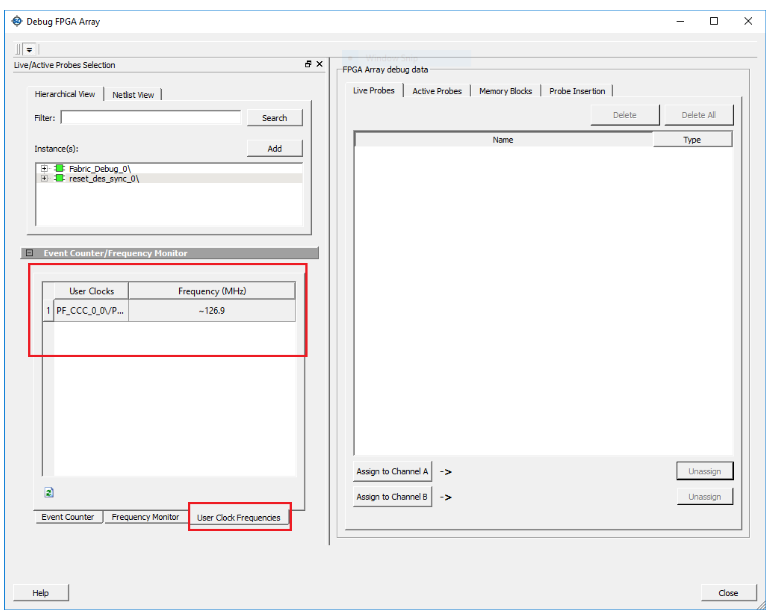
**Figure 25 •	Frequency Monitor**



2.8.4.3 Use User Clock Frequency

All of the CCC clock frequencies are calculated by selecting the User Clock Frequencies tab as shown in Figure 26. Live probes are temporarily unavailable till all the user clock frequencies are calculated and displayed.

**Figure 26 •	User Clock Frequencies**



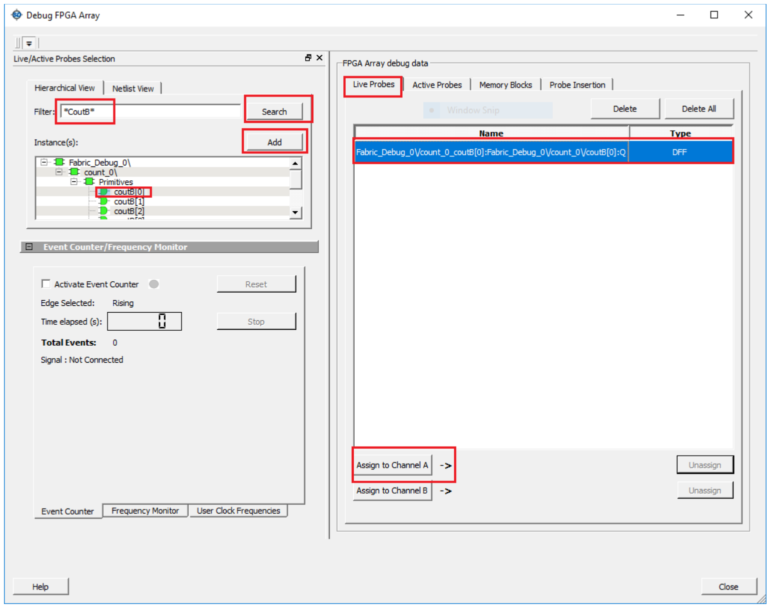
Note: The design includes one clock frequencies from PF_CCC component.

2.8.4.4 Select FHB

To select an FHB, perform the following steps:

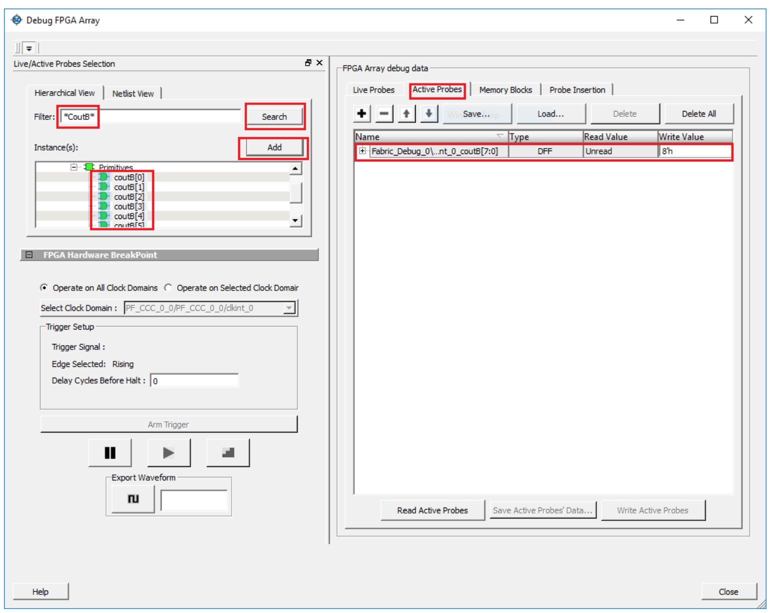
1. Go to the Live Probes tab and enter ***coutB*** in the filter box, then click **Search**.
2. Select **coutB [0]** and click **Add** as shown in Figure 27.
   coutB [0] is used as the hardware break point trigger.
3. Select **coutB [0]: Q**, and click **Assign to Channel A** as shown in Figure 27.

Figure 27 • **Select an FHB**



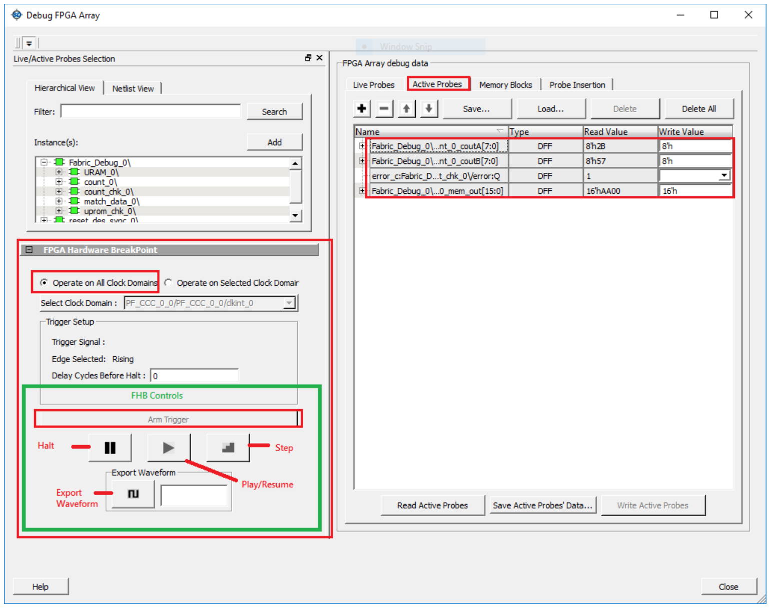
4. Select the Active Probes tab, and search for ***coutB***.
5. Select **coutB [0]** to **coutB [7]** by holding the Shift key and click **Add** as shown in Figure 28.

Figure 28 • **Adding FHB Trigger to Active Probes**



6. Select **Operate on All Clock Domains** as shown in the following figure.

Figure 29 • **Selecting Clock Domains**



7. Click **Arm Trigger** as shown in Figure 29. The counter halts on the next positive edge that occurs on the signal connected to Channel A (**coutB [0]**) in Live Probes.

Note: If you require a certain number of clock cycles before halting the clock domain after triggering, a value between 0 and 255 must be entered in **Delay Cycles Before Halt** before you click **Arm Trigger**. This sets the FHBs to trigger after the specified delay from the rising edge trigger.

The FHB controls are highlighted in Figure 29, the following actions can be performed using them:

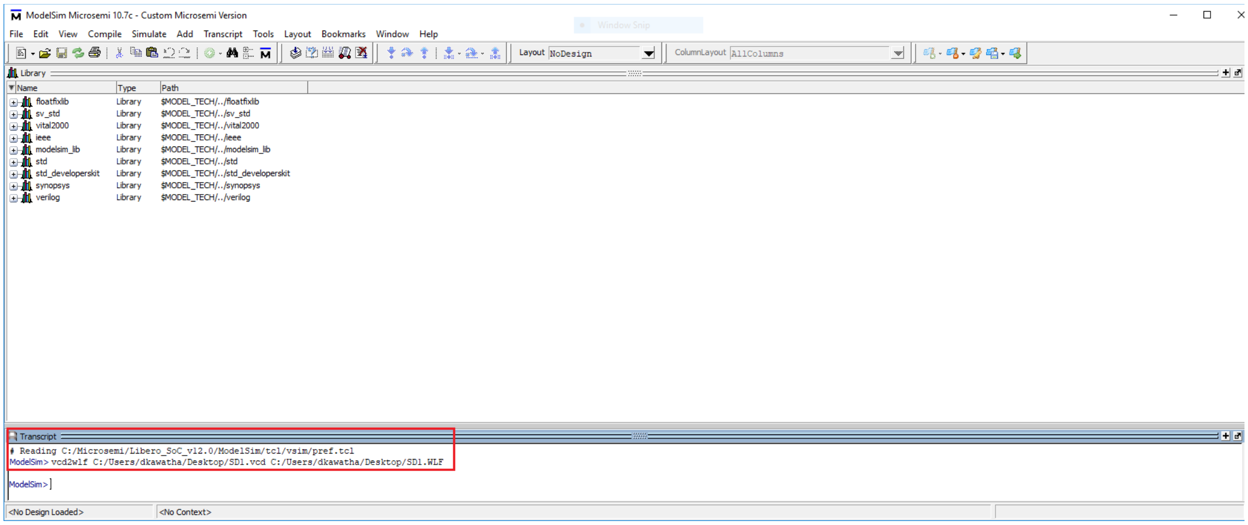
1. Provide custom delay cycles before the halt.
2. Force a selected clock domain or all clock domains to halt without waiting for a trigger from a live probe signal, by clicking the Halt button.
3. When the clock domain is in the halted state (live probe halt or force halt), resume the clock domain by clicking the Play/Resume button.
4. When the clock domain is in the halted state (live probe halt or force halt), advance the clock domain by one clock cycle and hold the state of the clock domain by clicking the Step button.
5. Save the waveform view of the selected active probes by specifying the number of clock cycles to capture in Export Waveform text box, and then clicking the Capture Waveform button. The waveform is saved as a vcd file.
6. View the saved waveforms by importing the vcd file. The waveform file can be viewed in a waveform viewer that supports the vcd format.
7. Click Close to close the Debug FPGA Array window. Click No when prompted for saving the active probes to a file.

2.8.4.5 Opening VCD File in ModelSim

To view the signals which are exported by SmartDebug in the vcd file use the Modelsim Waveform window:

1. Open ModelSim.
2. Go to the Transcript window and convert VCD to WLF format using the following the `vcd2wlf <file1.vcd> <file2.wlf>` command as shown in Figure 30.

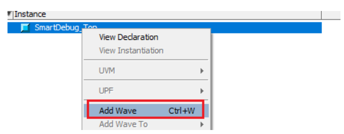
Figure 30 • **VCD to WLF Conversion**



Note: Conversion failures are mostly caused by non-existing instance path. Ensure that the specified instance paths are correct.

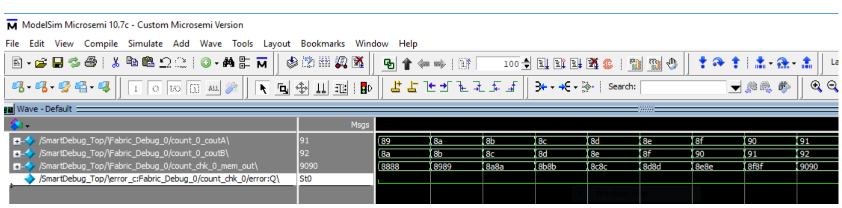
3. Open the WLF file created using File menu -> Open -> file2.wlf.
4. Select window with wlf file name and add signals to the Waveform window as shown in Figure 31.

Figure 31 • **Adding Signals**



5. Open the Wave window. Observe that the error signal is not asserted indicating data written and date read from DPSRAM is same as shown in Figure 32.

Figure 32 • **Wave Window**



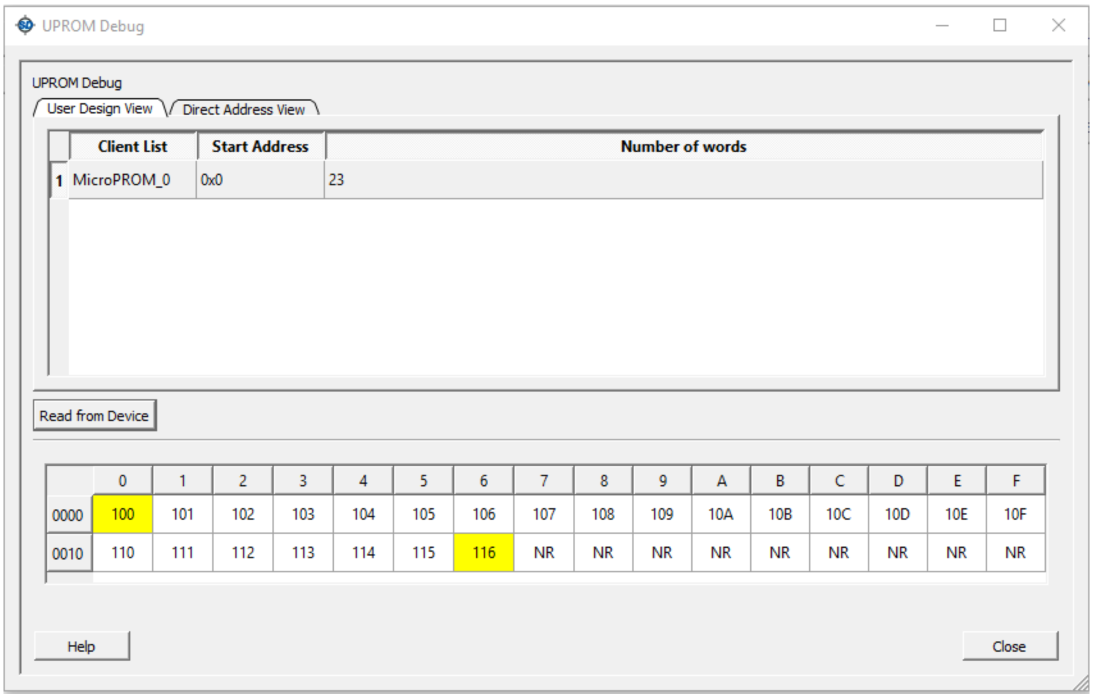
Note: The Fabric_Debug block includes the match_out signal, which must always be high indicating that the data expected from DPSRAM, and the data read from DPSRAM matches. But the provided design has a bug due to which the match_out signal always toggles. Debug the match_data.v logic using the FHB feature and find the route cause. Add match_out and mem_out of count_chk_0 block to Active probe and export the signals. Observe when the match_out signal is asserted to 1'b0.

2.8.5 Debug µPROM

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

1. Click **Debug µPROM** in the **SmartDebug** window. The **µPROM Debug** window is shown in the Figure 33.
2. Select **MicroPROM_0** in the **User Design View** tab and then click **Read from Device** to read the µPROM content. Check whether the content provided in uprom.mem file (part of design stimulus files) matches with the data read from µPROM. You can check the highlighted locations 100 and 116 in Figure 33 to verify the content.

Figure 33 • **µPROM Debug**



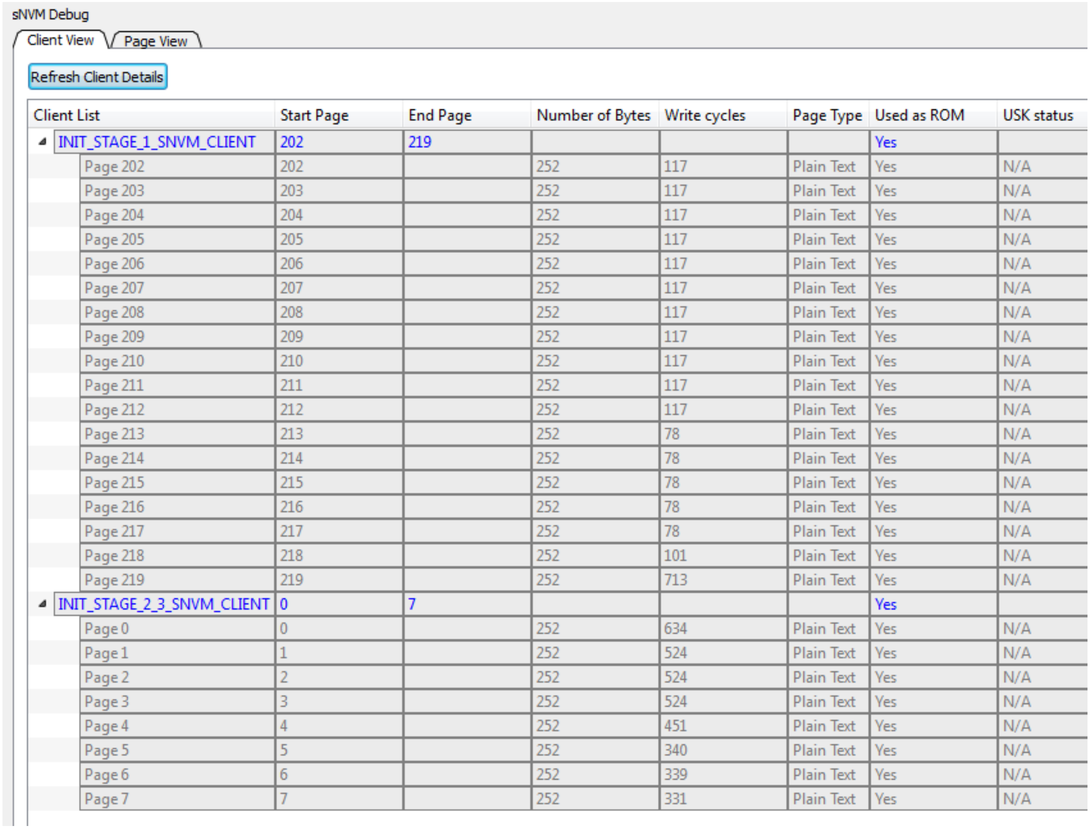
Note: PolarFire devices have a single user programmable read only memory (µPROM) row located at the bottom of the fabric, providing up to 459 Kb of non-volatile, read-only memory. The address bus is 16 bits wide, and the read data bus is 9-bit wide. µPROM is used to store the configuration data, which is used by Fabric logic to process.

2.8.6 sNVM Debug

sNVM Debug feature enables reading from the sNVM during debug. Debug Pass Key is required to carry out SNVM_DEBUG instruction. This feature supports debugging of non-authenticated plain text, authenticated plain text, and clients cipher authenticated.

1. Click **Debug SNVM** in the **SmartDebug** window.
2. Click **Client View** tab. The client view details are listed—Client Names, Start Page, Number of Bytes, Write Cycles, Page Type, Used as ROM, and USK Status.
3. Select a client from the list in the **Client View** and click **Read from Device** as shown in the following figure.

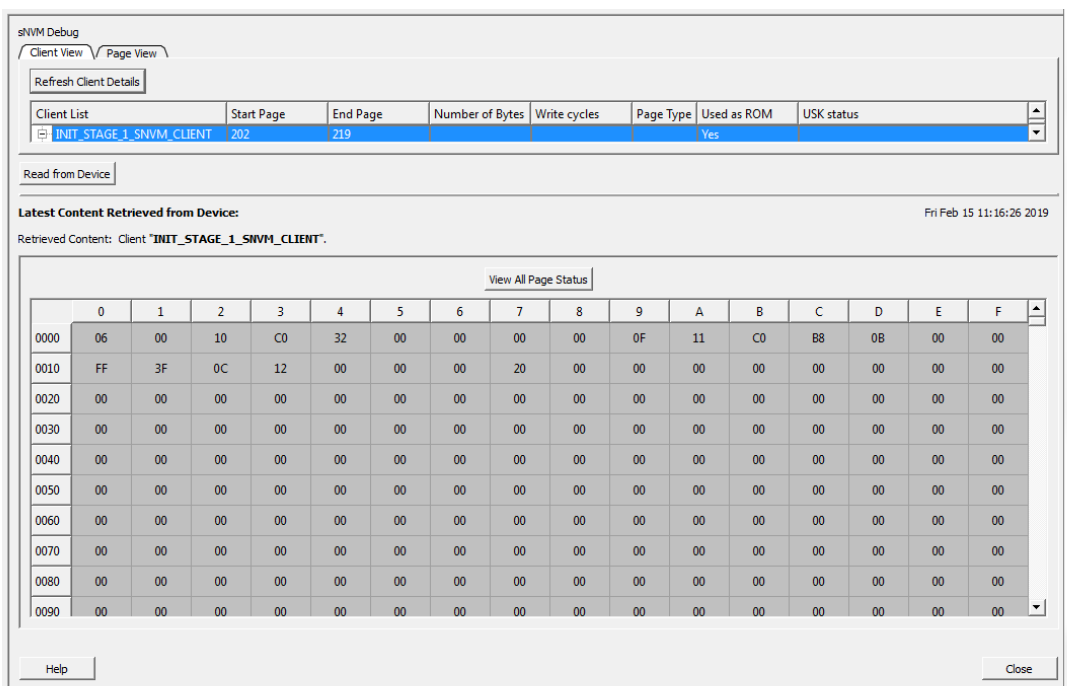
Figure 34 • **sNVM Debug**



Client List	Start Page	End Page	Number of Bytes	Write cycles	Page Type	Used as ROM	USK status
INIT_STAGE_1_SNVM_CLIENT	202	219				Yes	
Page 202	202		252	117	Plain Text	Yes	N/A
Page 203	203		252	117	Plain Text	Yes	N/A
Page 204	204		252	117	Plain Text	Yes	N/A
Page 205	205		252	117	Plain Text	Yes	N/A
Page 206	206		252	117	Plain Text	Yes	N/A
Page 207	207		252	117	Plain Text	Yes	N/A
Page 208	208		252	117	Plain Text	Yes	N/A
Page 209	209		252	117	Plain Text	Yes	N/A
Page 210	210		252	117	Plain Text	Yes	N/A
Page 211	211		252	117	Plain Text	Yes	N/A
Page 212	212		252	117	Plain Text	Yes	N/A
Page 213	213		252	78	Plain Text	Yes	N/A
Page 214	214		252	78	Plain Text	Yes	N/A
Page 215	215		252	78	Plain Text	Yes	N/A
Page 216	216		252	78	Plain Text	Yes	N/A
Page 217	217		252	78	Plain Text	Yes	N/A
Page 218	218		252	101	Plain Text	Yes	N/A
Page 219	219		252	713	Plain Text	Yes	N/A
INIT_STAGE_2_3_SNVM_CLIENT	0	7				Yes	
Page 0	0		252	634	Plain Text	Yes	N/A
Page 1	1		252	524	Plain Text	Yes	N/A
Page 2	2		252	524	Plain Text	Yes	N/A
Page 3	3		252	524	Plain Text	Yes	N/A
Page 4	4		252	451	Plain Text	Yes	N/A
Page 5	5		252	340	Plain Text	Yes	N/A
Page 6	6		252	339	Plain Text	Yes	N/A
Page 7	7		252	331	Plain Text	Yes	N/A

Figure 35 shows the **Client View** window.

Figure 35 • **sNVM Debug—Client View**

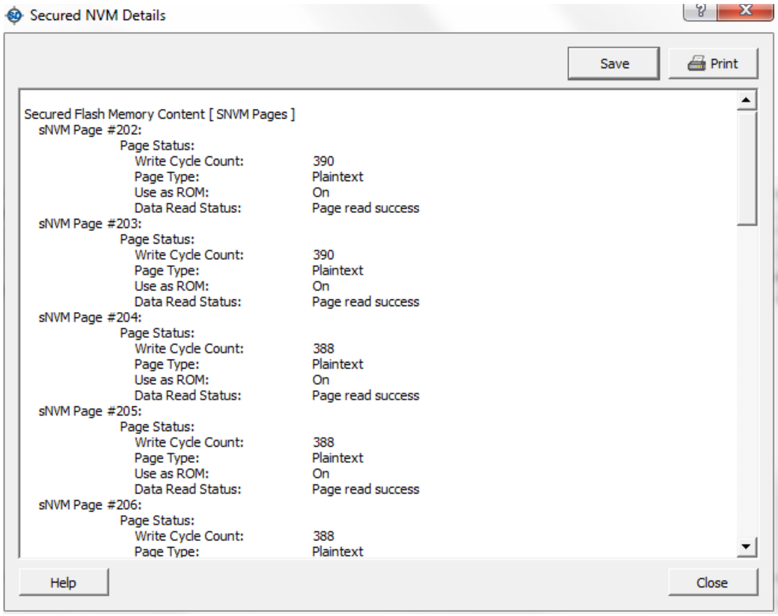


Client List	Start Page	End Page	Number of Bytes	Write cycles	Page Type	Used as ROM	USK status
INIT_STAGE_1_SNVM_CLIENT	202	219				Yes	

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	06	00	10	C0	32	00	00	00	00	0F	11	C0	B8	0B	00	00
0010	FF	3F	0C	12	00	00	00	20	00	00	00	00	00	00	00	00
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

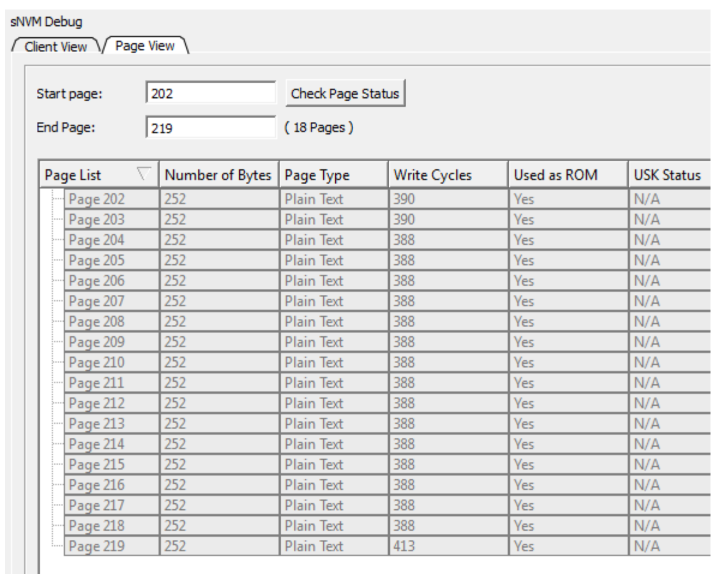
4. Click **View All Page Status** to view the page status such as Write Cycle Count, Page Type, Use as ROM, and Data Read Status as shown in the following figure.

Figure 36 • Secured NVM Details



5. Click **Page View** tab in the **sNVM Debug** window, Page view displays the client details of the required pages. You can read pages from 0-220 in the page view.
6. Enter the **Start page** and **End page** in the respective boxes.
7. Click **Check Page Status**. The page status information is displayed as shown in Figure 37.

Figure 37 • sNVM Debug—Page View



2.8.7 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
- SmartBERT
- Loopback Modes
- Static Pattern Transmit
- Eye Monitor

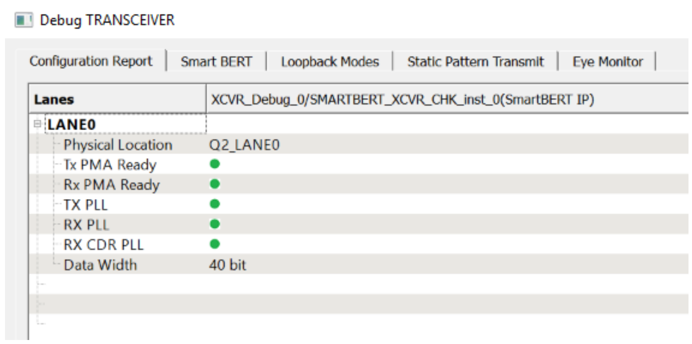
2.8.7.1 Configuration Report

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.

The following figure shows **Configuration Report** tab.

Figure 38 • **Configuration Report**



Note: The initial status of RX PLL and RX CDR PLL status is inactive, the status changes to active when the data is sent. SmartBERT is configured in CDR mode so the data must be sent to get the PLL Locked. Go to the Smart BERT tab, select the XCVR instance and start the data transmission using the default PRBS pattern. Then, the status changes to active.

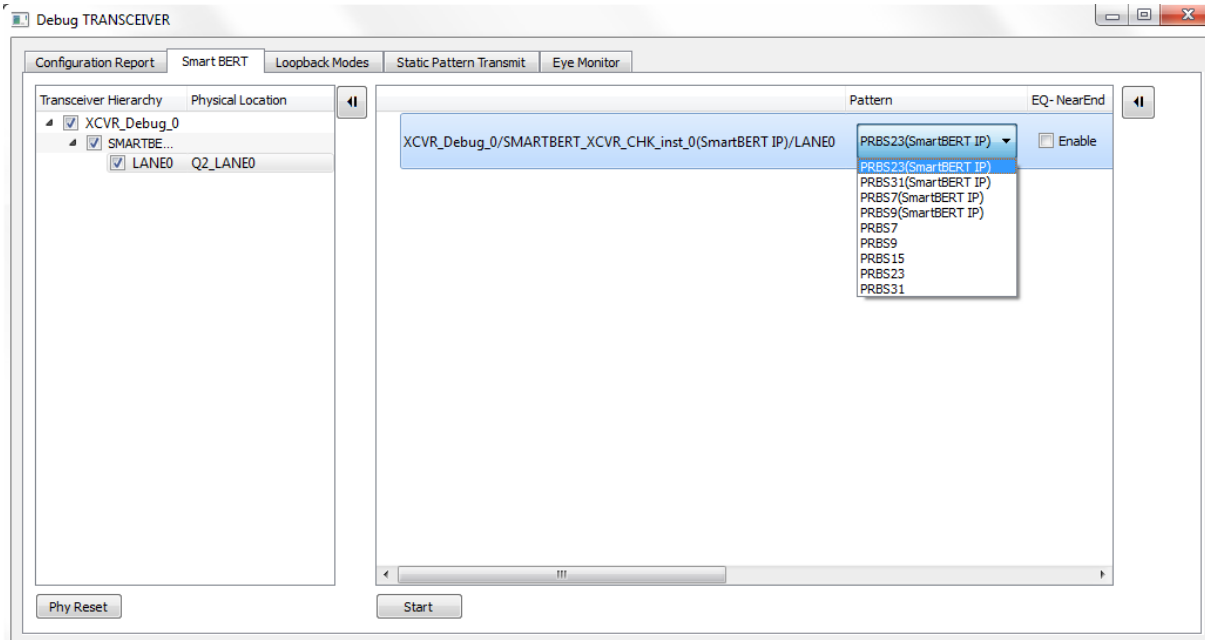
2.8.7.2 SmartBERT

SmartBERT enables you to run diagnostic tests on the transceiver lanes. 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(SmartBERT IP), PRBS9(SmartBERT IP), PRBS15(SmartBERT IP), PRBS23(SmartBERT IP), and PRBS31(SmartBERT IP). Near-end loopback can be performed using one of these PRBS patterns.

To run SmartBERT in Debug TRANSCEIVER, follow these steps:

1. Select the **SmartBERT** tab in the **Debug TRANSCEIVER** window.
2. Select **LANE** in the left pane.
3. Select the **Pattern** from the drop-down list.
4. Select the **EQ-NearEnd** check box to enable internal loop back, (this step can be ignored if external loop back is enabled).
5. Click **Start**. It enables both transmitter and the receiver for a particular lane and for a particular PRBS pattern. The following figure shows the Debug TRANSCEIVER window and the PRBS pattern options for SmartBERT.

Figure 39 • **Debug TRANSCEIVER—Smart BERT**

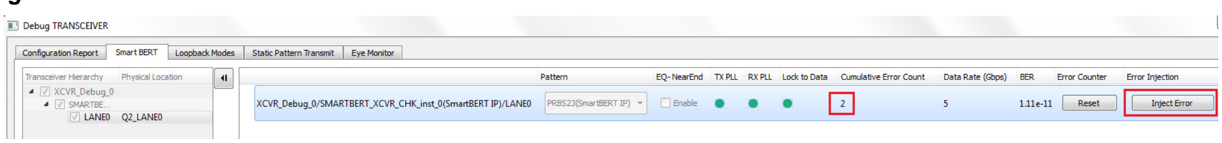


When a SmartBERT IP lane is added, the **Error Injection** column is displayed in the right pane. The error injection feature is provided to inject an error while running a PRBS pattern. This feature is unavailable if regular lanes are added. Also, this feature is enabled only for SmartBERT IP supported PRBS patterns.

6. Select **Reset** to clear the error count under **Cumulative Error Counter**. Error Count is displayed when the lane is added.

The following figure shows the **Smart BERT** tab and status of the TXPLL, RXPLL, Lock to Data, Data rate, and the BER.

Figure 40 • **Smart BERT—Error Counter**



2.8.7.3 Loopback Modes

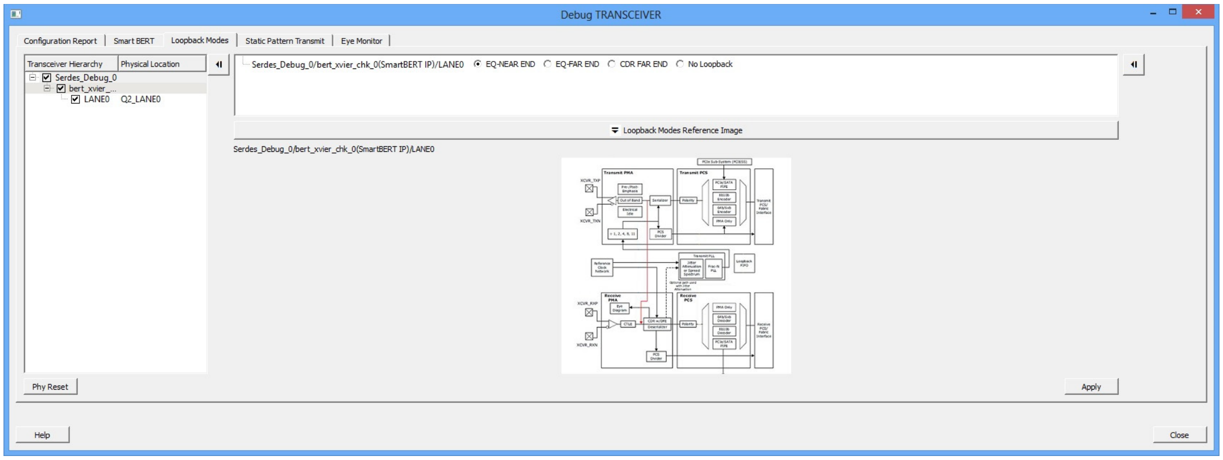
Loopback modes perform the following types of loopback tests:

- EQ-Near End Loopback: 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 loopback supports data transmission rates of up to 10.315 Gbps.
- EQ-Far End Loopback: Serialized data from Rx is looped back to Tx in PMA. This is called far-end serial loopback. EQ-Far End loopback supports data transmission rates of up to 1.25 Gbps.
- CDR-Far End Loopback: De-serialized data from PCS Rx channel is looped back to Tx.
- No Loopback: Data is not looped internally.

To select Loopback mode, perform the following steps:

1. Select **LANE** in the left pane.
2. Select Loopback Mode and click on **Apply** to apply the loopback.
3. Go to SmartBERT tab, select **LANE** and choose any probes pattern and click **Start**.
4. Check the status of TX PLL, RX PLL, Lock to Data.
5. Click **Stop** to stop the pattern transmission for the selected lane.

Figure 41 • **Debug TRANSCEIVER—Loopback Modes**



2.8.7.4 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

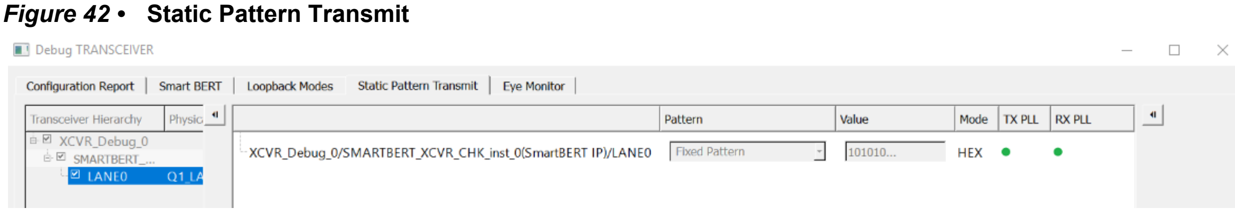
The user pattern is defined in the value column. It must be hex numbers and not greater than the configured data width.

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, perform the following steps:

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. Select a pattern from the **Pattern** drop-down list.
3. Click **Start**. The static pattern for the selected lanes is transmitted.
4. The static pattern for the selected lanes is transmitted. Status of TX PLL and RX PLL should be green.
5. Click **Stop**. The static pattern transmission is stopped for the selected lanes. Figure 42 shows the **Static Pattern Transmit** tab.

Figure 42 • **Static Pattern Transmit**



2.8.7.5 Eye Monitor

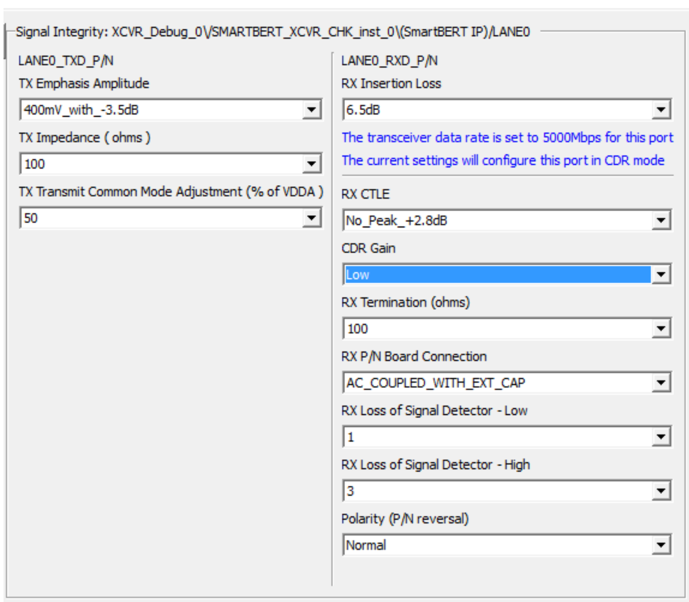
Eye Monitor enables visualizing the eye diagram present within the receiver. This feature plots the receive eye after the CTLE and Receiver functions. The diagram representation provides vertical and horizontal measurements of the eye and BER performance measurements. Whenever PRBS/static pattern transmission is in progress, click the **Eye Monitor** tab in the **Debug TRANSCEIVER** window to see the eye monitor representation within the receiver.

In Libero SoC, the following types of Eye Scan modes are supported:

- **Normal mode—**in this mode, Eye Monitor performs a single eye scan and displays the Eye Diagram on the Eye Monitor plot.
- **Infinite Persistent Mode**— in this mode, the Plot Eye button changes to Start Plot Eye. Select **Start Plot Eye** to start infinite persistent eye monitoring. The Start Plot Eye button changes to Stop Plot Eye and the infinite scanning and accumulation process begins. In every iteration, the eye is cumulated with all previous eyes to make a single cumulative eye. This cumulative eye is displayed with a color scheme on the Eye Monitor plot. The completed iteration number and the cumulative BER is updated and displayed after every iteration, along with the cumulative eye. To stop cumulative eye monitoring, click the Stop Plot Eye button. The process halts after the current iteration completes.
- **Design Initiated Eye Plots**—in this mode, the Select Eye Output drop-down is enabled when an Eye Plot log file is browsed and loaded in the Eye Monitor page. Click Browse File to load the Eye Plot output files. If the loaded Design Initiated Eye Plot log file does not contain any eye output, it is disabled. After selecting Eye output from the Select Eye Output drop-down, click Plot Eye to start eye monitoring for the lane. Then the Eye diagram displays for the selected log file.

The following figure shows the recommended SI settings for the demo design. These settings are for short reach and less lossy cables.

Figure 43 • **Recommended Settings for Eye Monitor**



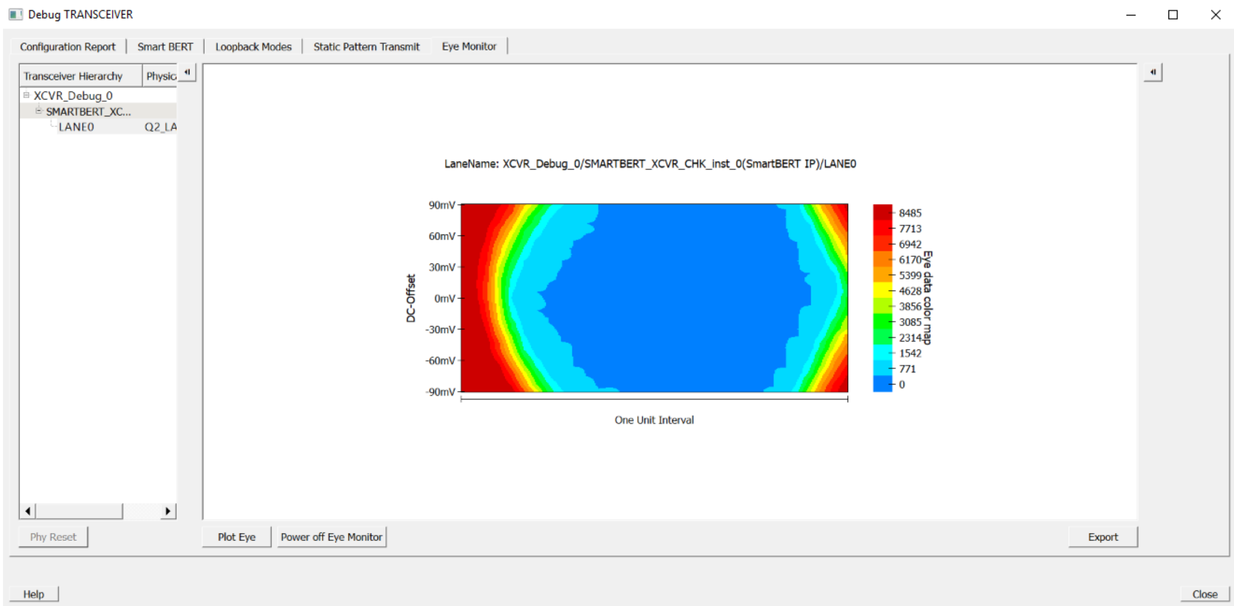
For plotting the Eye Diagram, perform the following steps:

1. Go to **EYE Monitor** tab and Select **LANE0**.
2. Click **Power on Eye Monitor**.
3. Go to **SmartBERT** tab, select **LANE0** and choose any probes pattern and click **Start**.
4. Go back to **Eye Monitor** tab and click **Plot Eye** to plot the eye.

The Eye Plot of the Signal in **LANE0** is plotted.

Figure 44 shows the **Eye Monitor** tab.

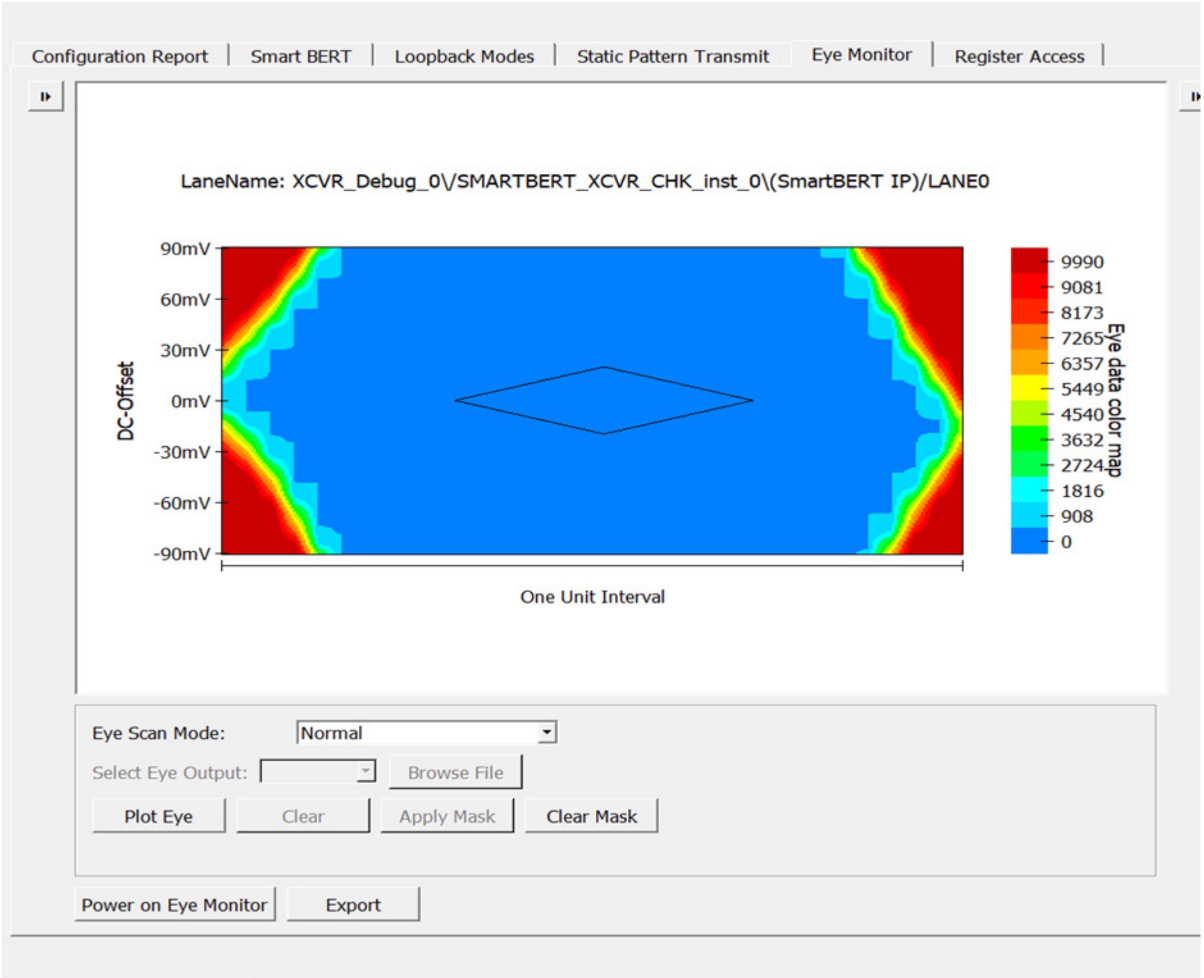
Figure 44 • **Debug TRANSCEIVER—Eye Monitor**



After plotting the Eye diagram, user can use the **Apply Mask** option to know the best eye opening with least errors. Both the **Apply Mask** and the **Clear Mask** options are disabled in the Default View. Click Plot Eye to enable the **Apply Mask** option.

5. After selecting the **Apply Mask** option, the Clear Mask option is enabled and the Eye Mask for the Eye Plot appears as shown in Figure 45.

Figure 45 • **Eye Monitor GUI After Applying the Mask Using Apply Mask Button**



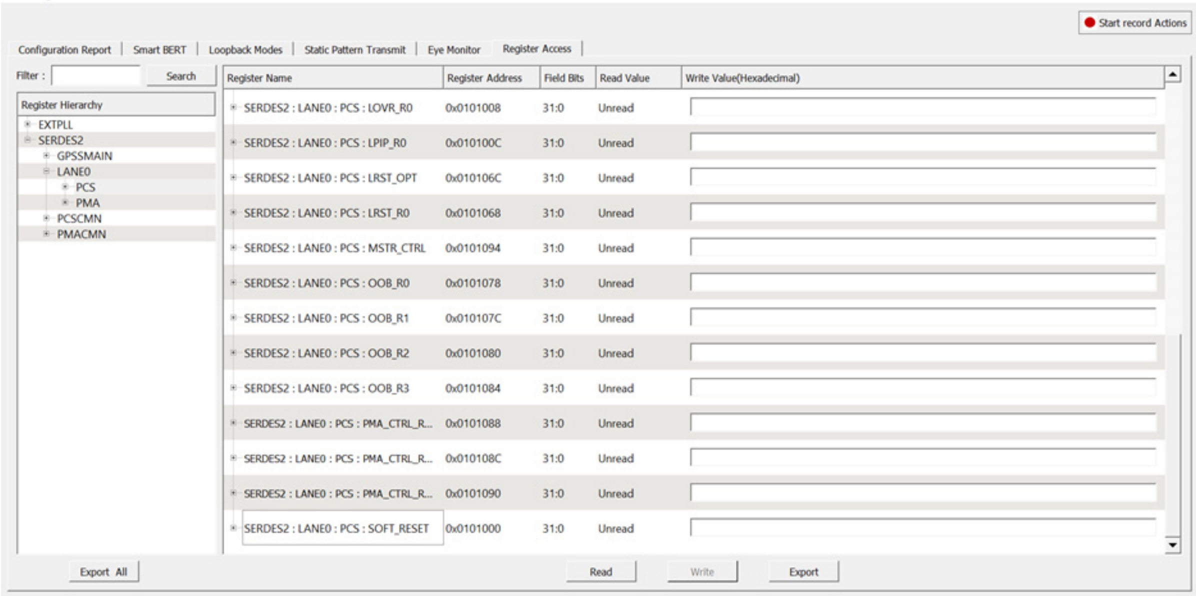
2.8.7.6 Register Access

The **Register Access** tab enables the following operations:

- Register read and write
- Export all register operations. The exported register details are saved to a `.csv` file.
- Register hide

Figure 46 shows the register access tab.

Figure 46 • **Register Access Tab**

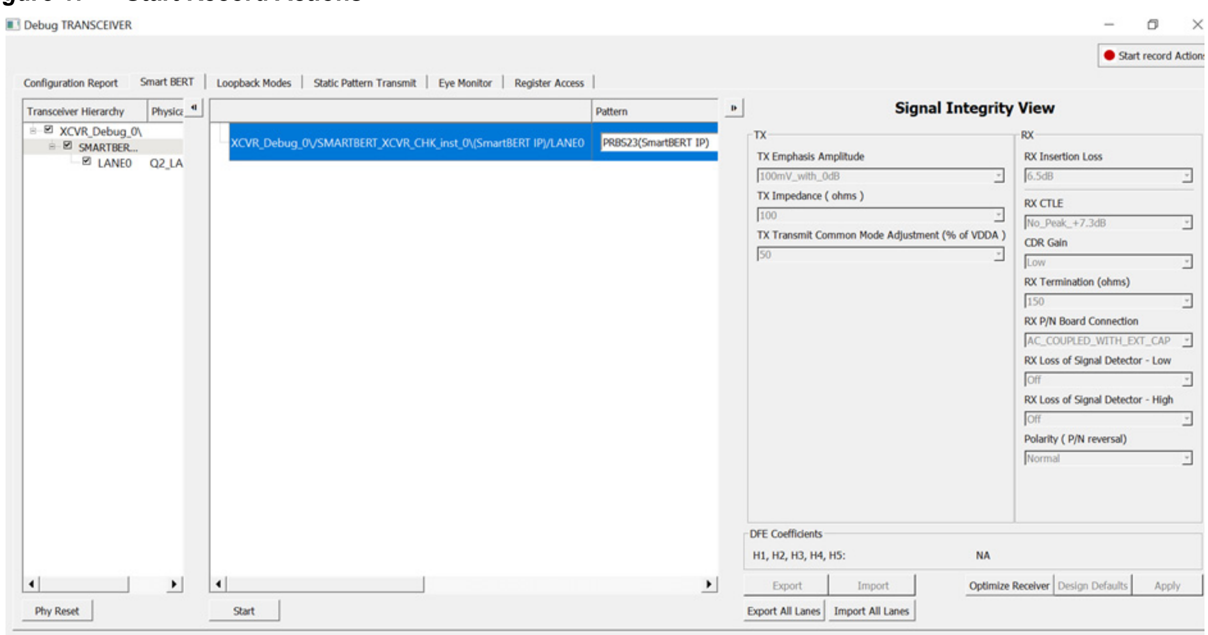


For detailed information about Register Access tab, see *PolarFire SmartDebug User Guide*.

2.8.7.7 Start Record Actions

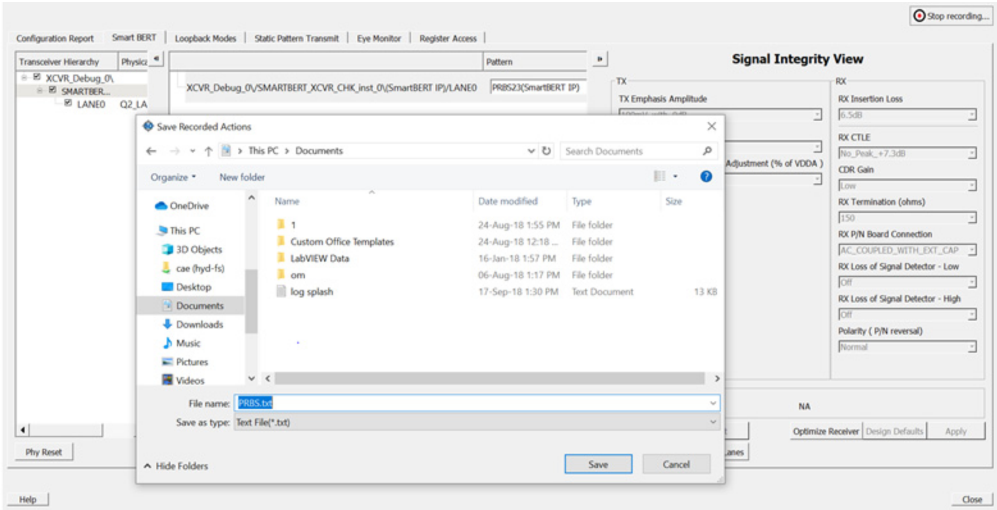
The **Start Record Actions** option is used to record the register sequence of XCVR operations into a file. This option is available at the top-right on the Debug Transceiver window as shown in Figure 47.

Figure 47 • **Start Record Actions**



This option is hidden in the Demo Mode. When this option is selected, the recording starts and the option changes to **Stop recording** for stopping the recording. When **Stop recording** is selected, a window pop-up prompts the user for the output to be saved to a `.txt` file as shown in. After saving the file, the Debug TRANSCEIVER window goes to default state.

Figure 48 • Saving the Recording



For more information about this option, see *PolarFire SmartDebug User Guide*.

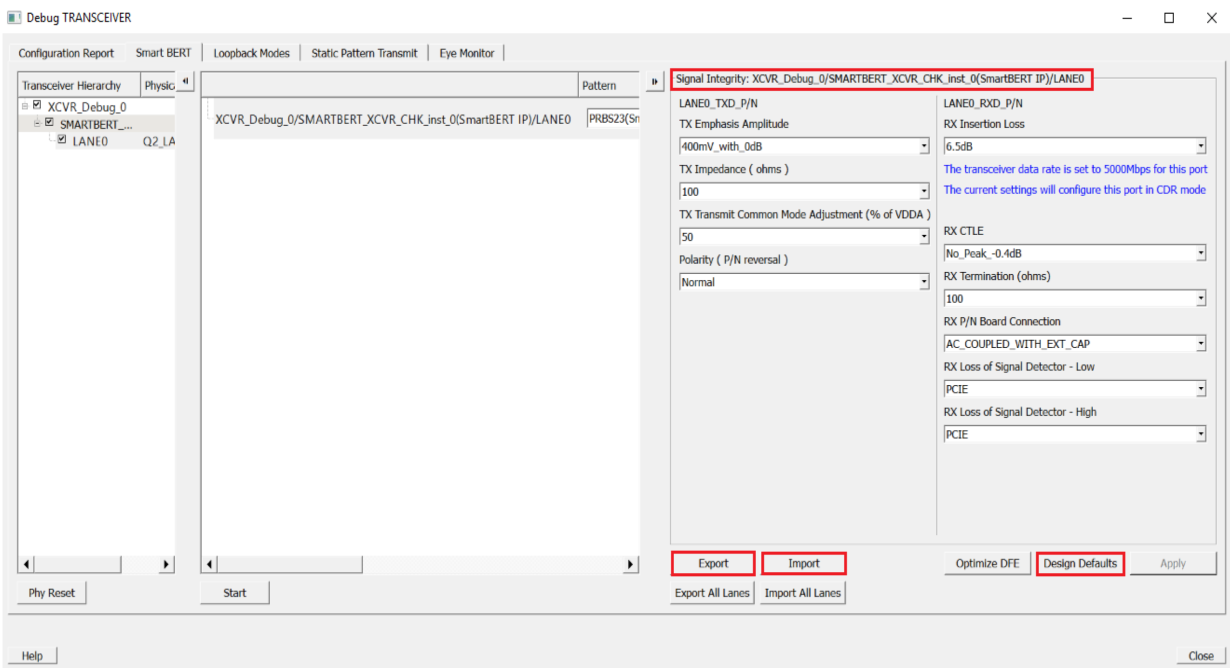
2.8.7.8 Signal Integrity

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 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, as shown in the following figure.

Figure 49 • **Signal Integrity**



2.8.7.8.1 Design Defaults

Click **Design Defaults** to load the signal integrity parameter options for the selected lane instance. These are the signal integrity settings selected in the Libero design flow and reside in the STAPL file.

2.8.7.8.2 Export

Click **Export** to export the selected parameter options and other physical information to an external PDC file. A popup box prompts to choose the location where you want the PDC file to be exported.

The exported content is in two set_io commands form—TXP and RXP ports of the selected lane instance.

2.8.7.9 Optimize Receiver

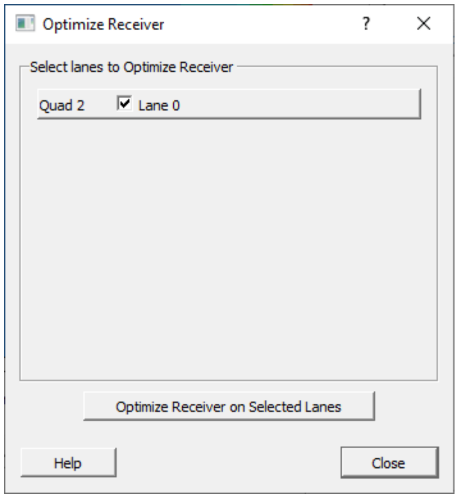
SmartDebug uses the Receiver coefficients to optimize the settings for the overall signal integrity at the receiver.

To run 'Optimize Receiver', perform the following steps:

1. In Debug Transceiver, go to Signal Integrity and click on Optimize Receiver.
2. In Optimize Receiver window the following settings
   - Select Lanes to Optimize Receiver: Lane0
3. Click **Optimize Receiver on selected Lanes**.

Software based Optimizing Receiver process is successful on all selected lanes.

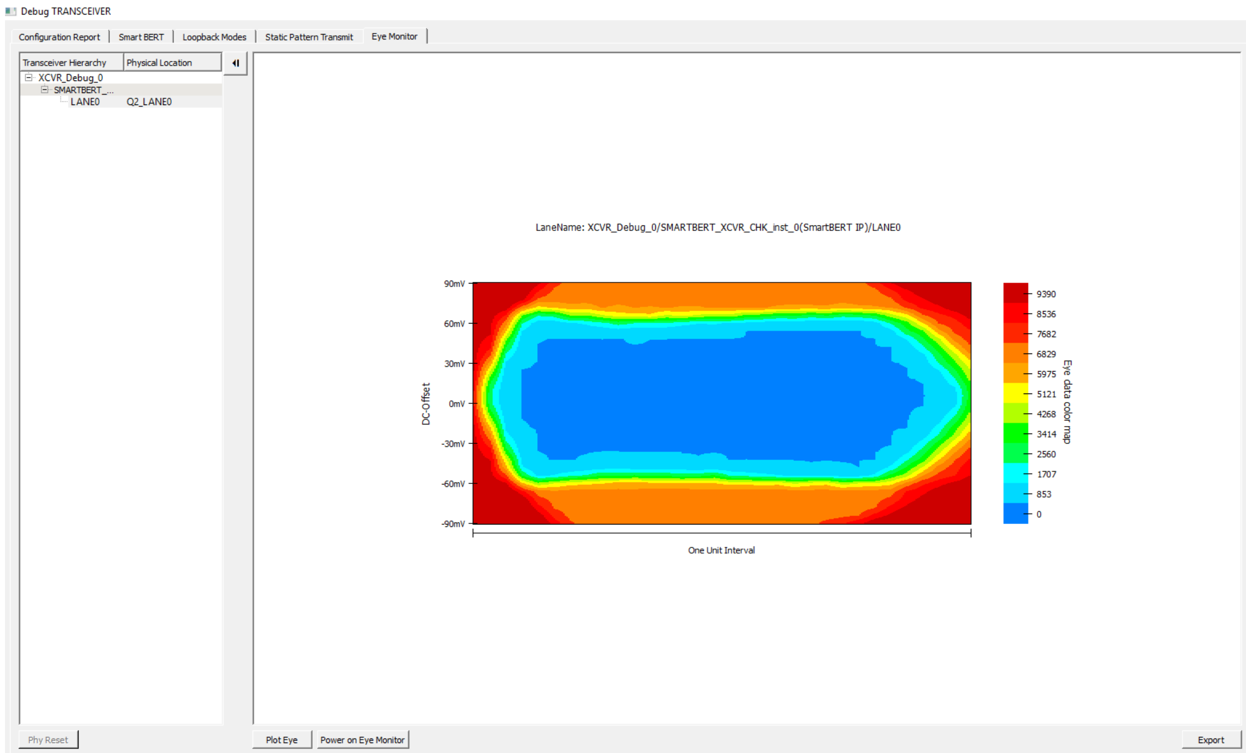
Figure 50 • **Optimize Receiver**



2.8.7.9.1 Eye Monitor after Optimizing Receiver

After Optimizing Receiver, follow the steps mentioned in section Eye Monitor, page 29 for plotting the Eye Diagram.

Figure 51 • Eye Diagram after using Optimize Receiver

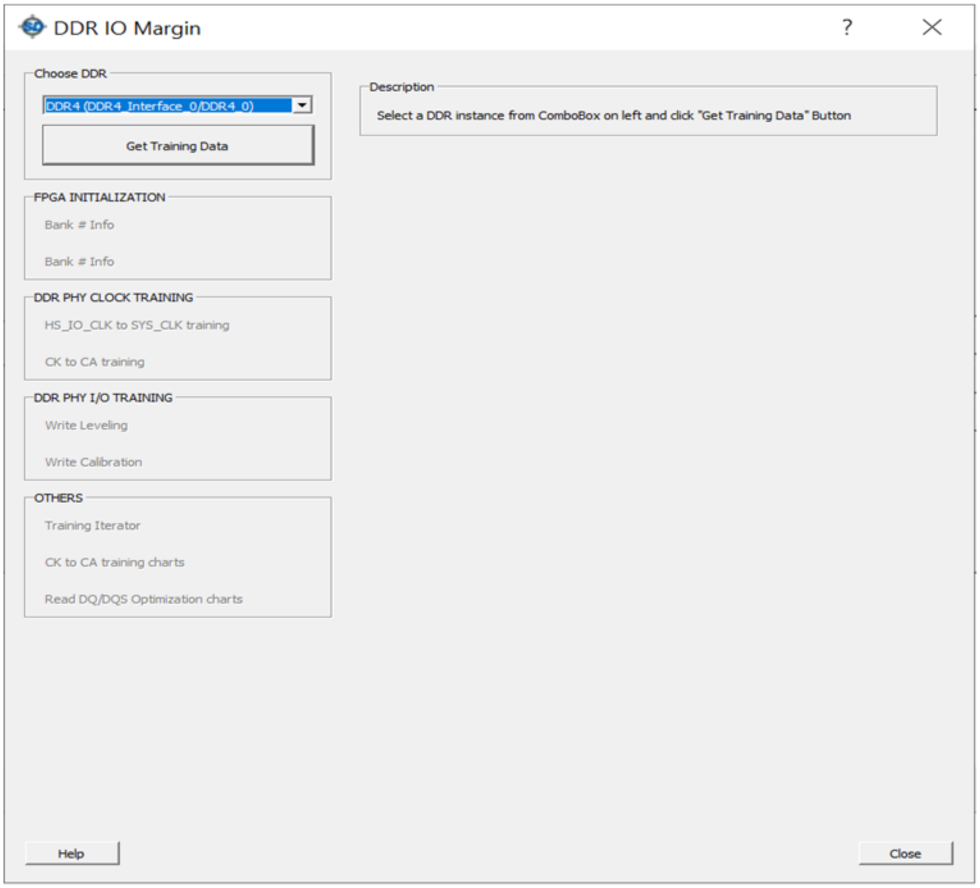


2.8.8 Debug DDR IO Margin

To access the Debug DDR IO Margin feature, select **Debug DDR Memory** from the main SmartDebug window. This option is available only for DDR3/DDR4/LPDDR3 memory configurations. This option is not visible when DDR memory is not used in the design. The default view of the **DDR IO Margin** window.

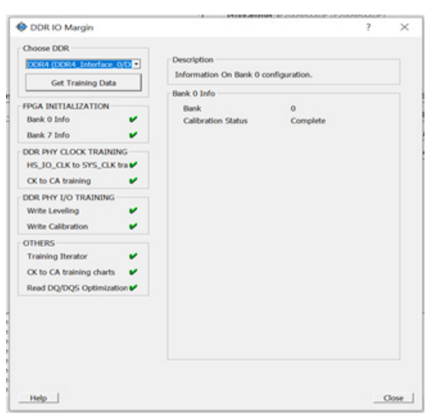
Note: After programming the device, the LED11 glows, which indicates the completion of DDR transactions.

Figure 52 • DDR IO Margin Window



Initially, all options in the DDR IO Margin GUI are disabled. Select the required DDR instances and click **Get Training Data**. After this, a script is run for fetching the training data. After around two minute the fetched information of the selected DDR Instance is displayed as shown in Figure 53.

Figure 53 • Training Data



2.9 Conclusion

This application note demonstrated capabilities of SmartDebug to observe and analyze many embedded device features. Live probes give a real-time access to device test points, and internal logic states can be accessed using active probes. The SmartDebug TRANSCEIVER utility assists FPGA and board designers to validate signal integrity of high-speed serial links in a system and improve board bring-up time. This can be done in real-time without any design modifications. The PMA analog settings can be tuned to optimize link performance and to match the design to the system.

3 Appendix 1: Known Issues

This chapter lists known issues related to SmartDebug hardware design debug and provides workarounds for each of the issues.

3.1 Data Traffic Errors on XCVR Lanes in CDR Mode

While plotting the eye using eye monitor, errors are introduced in data traffic on transceiver lanes configured to use the CDR receiver path. The errors are introduced when Receiver and EM blocks are turned off during normal operation to save power. This issue does not impact functionality. The cumulative error count and BER values can be ignored when plotting the eye. A software update will be provided in future Libero releases to fix the issue.

4 Appendix 2: Place and Route

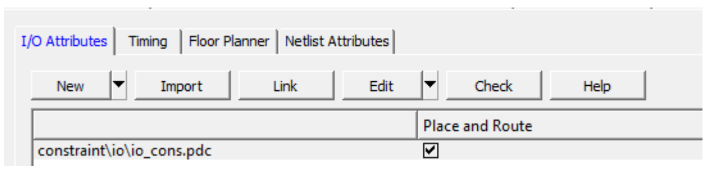
The place and route process requires the following steps to be completed:

- Selecting the already imported `io_cons.pdc` file.
- Placing the XCVR_Debug_0 block using the I/O Editor.
- Ensuring all the I/Os are locked.

To complete the place and route process, follow these steps:

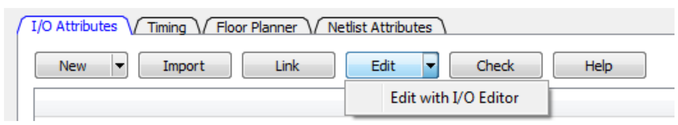
1. On the I/O Attributes tab, select the check box next to the `io_cons.pdc` file, as shown in Figure 54, page 40. The `io_cons.pdc` file contains the I/O assignment for reference clock, switches and XCVR Lanes.

Figure 54 • I/O Attributes Tab



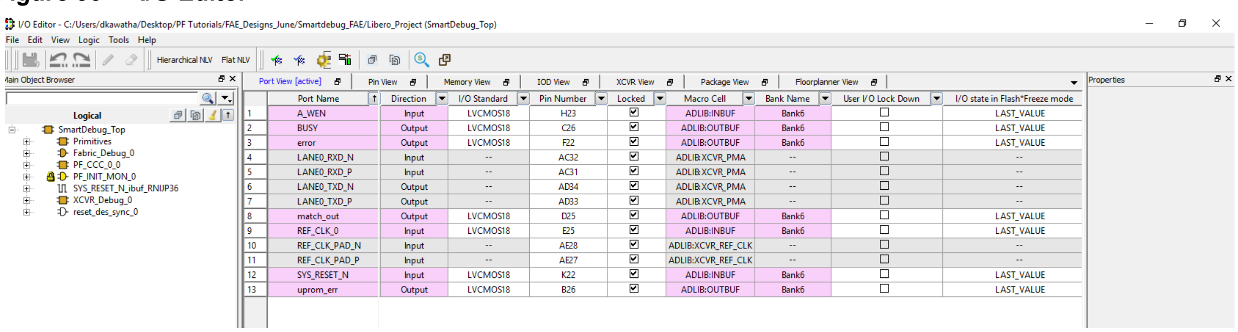
2. From the Edit drop-down list, select Edit with I/O Editor, as shown in Figure 55, page 40.

Figure 55 • Editing with I/O Editor



3. The I/O Editor will open as shown in Figure 56, page 40.

Figure 56 • I/O Editor



4. Select the XCVR View in the I/O Attribute editor. This view allows you to assign IO locations to XCVR and reference clock.
5. Place TX_PLL, XCVR_REF_CLK, and XCVR_Debug as shown in Figure 57, page 41 for Evaluation kit or Figure 58, page 41 for SPLASH kit.

Figure 57 • **Placing the TX_PLL, XCVR_REF_CLK, and PF_XCVR Components (Evaluation kit)**

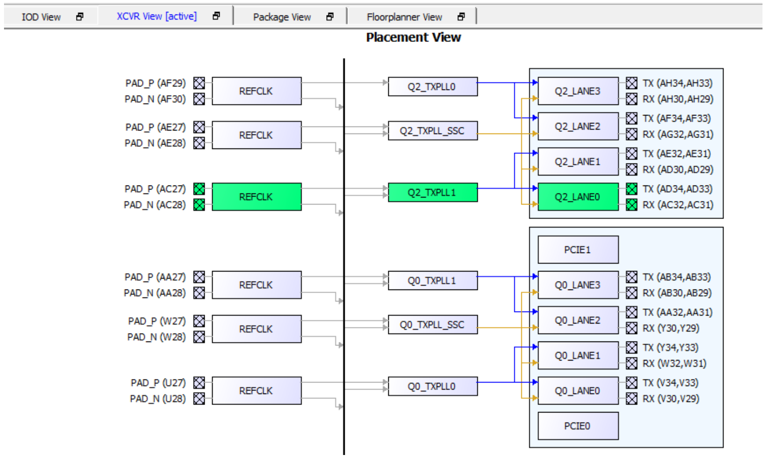
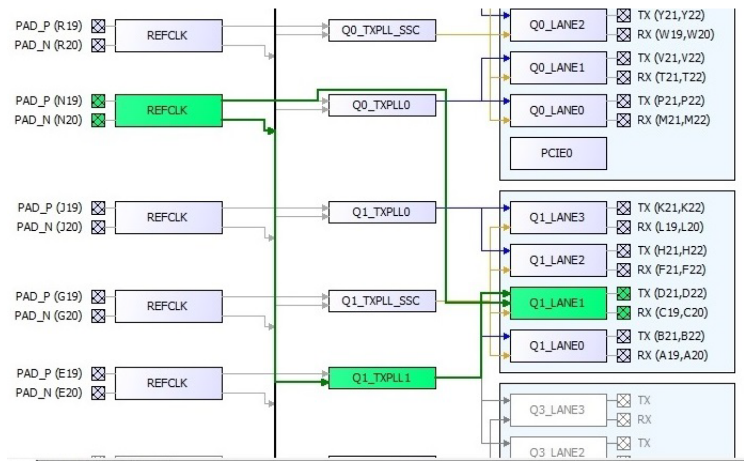
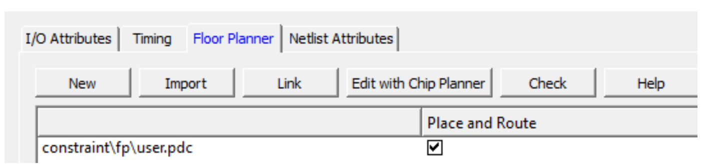


Figure 58 • **Placing the TX_PLL, XCVR_REF_CLK, and PF_XCVR Components (SPLASH kit)**



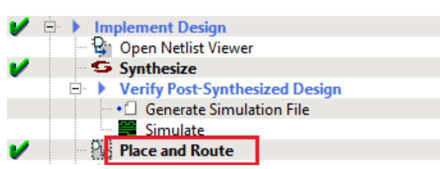
6. Select **File > Commit** to save the placement the close the I/O Editor (**File > Exit**).
7. Select the Constraint Manager Floor Planner. The constraint file (`user.pdc`) should be visible. Ensure the file is checked to be used for Place and Route.

Figure 59 • **Constraint Files on the I/O Attributes Tab**



8. Double-click **Place and Route** from the **Design Flow** tab. When place and route is successful, a green tick mark appears next to **Place and Route** as shown in Figure 60, page 41.

Figure 60 • **Place and Route**



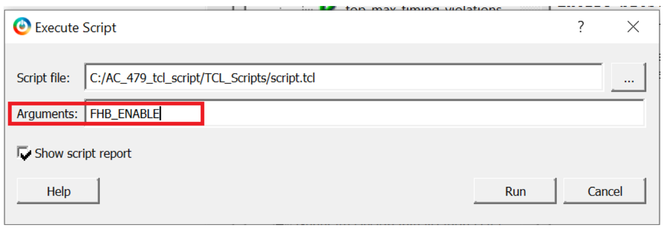
5 Appendix 3: Running the TCL Script

TCL scripts are provided in the design files folder under directory TCL_Scripts. If required, the design flow can be reproduced from Design Implementation till generation of job file.

To run the TCL, follow the steps below:

1. Launch the Libero SoC.
2. Select **Project > Execute Script....**
3. Click **Browse** and select `script.tcl` from the downloaded `TCL_Scripts` directory.
4. In order to enable FHB provide argument `FHB_ENABLE` in the **Arguments** tab.

Figure 61 • **Enabling FHB**



5. Select **Run**.
   After successful execution of TCL script, Libero project is created within TCL_Scripts directory.

For more information about TCL scripts, see the following readme:

- **mpf_ac479_eval_df/TCL_Scripts/readme.txt**

Refer to *Libero® SoC TCL Command Reference Guide* for more details on TCL commands. Contact Technical Support for any queries encountered when running the TCL script.

6 Appendix 4: References

This section lists documents that provide more information about the SmartDebug and IP cores used in the reference design.

- For more information about SmartDebug, see *PolarFire SmartDebug User Guide*.
- For more information about PolarFire transceiver blocks, see *PolarFire FPGA and PolarFire SoC FPGA Transceiver User Guide*.
- Fore more information about PF_CCC, see *PolarFire FPGA and PolarFire SoC FPGA Clocking Resources User Guide*.
- For more information about Libero SoC, see the *Microsemi Libero SoC PolarFire* web page.
- For more information about PolarFire FPGA Evaluation Kit, see *UG0747: PolarFire FPGA Evaluation Kit User Guide*.
- For more information about the Splash kit, see *UG0786: PolarFire FPGA Splash Kit User Guide*.
- For more information about PF_UPROM, PF_USRAM, and PF_DPSRAM, see *Libero* catalog.
- For more information about Identify RTL, see *Synopsys Identify RTL User Guide*.