Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer’s responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided “as is, where is” and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

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.
# Contents

1 Revision History ......................................................... 1  
  1.1 Revision 4.0 .......................................................... 1  
  1.2 Revision 3.0 .......................................................... 1  
  1.3 Revision 2.0 .......................................................... 1  
  1.4 Revision 1.0 .......................................................... 1  

2 Debugging PolarFire FPGA Designs ................................... 2  
  2.1 Design Requirements ................................................ 3  
  2.2 Prerequisites .......................................................... 3  
  2.3 Demo Design ........................................................... 4  
  2.4 Clocking Structure .................................................. 5  
  2.5 Programming the Device .......................................... 6  
    2.5.1 Programming the Device on the Evaluation Kit .......... 7  
    2.5.2 Programming the Device on the SPLASH Kit ............ 8  
  2.6 Debugging Using SmartDebug ...................................... 10  
    2.6.1 Launch SmartDebug from Libero ......................... 10  
    2.6.2 View Device Status ......................................... 11  
    2.6.3 Debug FPGA Array ........................................... 11  
    2.6.4 Debug µPROM .................................................. 16  
    2.6.5 sNVM Debug .................................................... 16  
    2.6.6 Debug TRANSCEIVER ....................................... 19  
  2.7 Conclusion .......................................................... 25  

3 Appendix: Known Issues .............................................. 26  
  3.1 Probe Points Write Issue ....................................... 26  
  3.2 Data Traffic Errors on XCVR Lanes in CDR Mode ............. 27  

4 Appendix: Place and Route .......................................... 28  

5 Appendix: References ............................................... 31
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>SmartDebug Top-Level Blocks</td>
<td>4</td>
</tr>
<tr>
<td>Figure 2</td>
<td>XCVR_Debug Overall Design Blocks</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Fabric_Debug Overall Design Blocks</td>
<td>4</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Clocking Structure</td>
<td>5</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Board Setup (Evaluation kit)</td>
<td>6</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Programming the Device</td>
<td>7</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Board Setup (SPLASH kit)</td>
<td>8</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Launching SmartDebug Design</td>
<td>8</td>
</tr>
<tr>
<td>Figure 9</td>
<td>SmartDebug Window Debug Options</td>
<td>10</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Device Status Report Sample</td>
<td>10</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Debug FPGA Array—Live Probes</td>
<td>12</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Debug FPGA Array—Active Probes</td>
<td>13</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Pseudo-static Signal Polling</td>
<td>13</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Debug FPGA Array—Memory Blocks</td>
<td>14</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Memory Blocks—Read Block</td>
<td>14</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Memory Blocks—Write Block</td>
<td>15</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Debug FPGA Array—Probe Insertion</td>
<td>16</td>
</tr>
<tr>
<td>Figure 18</td>
<td>µPROM Debug</td>
<td>16</td>
</tr>
<tr>
<td>Figure 19</td>
<td>sNVM Debug</td>
<td>17</td>
</tr>
<tr>
<td>Figure 20</td>
<td>sNVM Debug—Client View</td>
<td>18</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Secured NVM Details</td>
<td>18</td>
</tr>
<tr>
<td>Figure 22</td>
<td>sNVM Debug—Page View</td>
<td>19</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Configuration Report</td>
<td>19</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Debug TRANSCEIVER—Smart BERT</td>
<td>20</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Smart BERT—Error Counter</td>
<td>20</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Debug TRANSCEIVER—Loopback Modes</td>
<td>21</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Static Pattern Transmit</td>
<td>22</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Recommended Settings for Eye Monitor</td>
<td>22</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Debug TRANSCEIVER—Eye Monitor</td>
<td>23</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Signal Integrity</td>
<td>23</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Optimize DFE</td>
<td>24</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Eye Diagram after using Optimize DFE</td>
<td>25</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Programming Connectivity and Interface</td>
<td>26</td>
</tr>
<tr>
<td>Figure 34</td>
<td>I/O Attributes Tab</td>
<td>26</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Editing with I/O Editor</td>
<td>28</td>
</tr>
<tr>
<td>Figure 36</td>
<td>I/O Editor</td>
<td>28</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Placing the TX_PLL, XCVR_REF_CLK, and PF_XCVR Components (Evaluation kit)</td>
<td>29</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Placing the TX_PLL, XCVR_REF_CLK, and PF_XCVR Components (SPLASH kit)</td>
<td>29</td>
</tr>
<tr>
<td>Figure 39</td>
<td>Constraint Files on the I/O Attributes Tab</td>
<td>30</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Place and Route</td>
<td>30</td>
</tr>
</tbody>
</table>
Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Design Requirements</td>
<td>3</td>
</tr>
<tr>
<td>Table 2</td>
<td>Jumper Settings For Evaluation Kit</td>
<td>7</td>
</tr>
<tr>
<td>Table 3</td>
<td>Jumper Settings For SPLASH Kit</td>
<td>8</td>
</tr>
</tbody>
</table>
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

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.

1.2 Revision 3.0

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

1.3 Revision 2.0

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

1.4 Revision 1.0

The first publication of this document.
Debugging PolarFire FPGA Designs

Design debug is a critical phase of the FPGA design flow. Microsemi PolarFire® devices support the following debugging methods.

1. Debugging using Identify
   The Identify debug tool integrated into Libero SoC PolarFire, enables FPGA debugging using an embedded logic analyzer. For more information, see TU0780: Using Identify ME with Libero SoC Tutorial.

   **Note:** The Identify tool uses the UJTAG IP for hardware debugging. For designs that require hardware and software (Identify and SoftConsole) debugging, ensure to connect the JTAG I/Os of the Mi-V IP to GPIOs and not use the CoreJTAGDebug IP.

2. Debugging Processor-based designs using SoftConsole
   SoftConsole enables debugging of processor-based designs (Mi-V or Cortex-M1 based designs). For more information, see TU0775: PolarFire FPGA: Building a Mi-V Processor Subsystem Tutorial and TU0778: PolarFire FPGA Building a Cortex-M1 Processor Subsystem Tutorial.

3. Debugging using SmartDebug
   SmartDebug enables 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.

   The built-in probe points of the PolarFire device and the probe capabilities of SmartDebug enable real-time debug features.

   SmartDebug offers the following capabilities:
   
   - **Live probes:** Two dedicated probes can be configured to observe a probe point. The probe point may be any output of a register. After selecting the probe points, the probe data can be sent to two dedicated pins (PROBE_A and PROBE_B). You can connect an oscilloscope to the probe pins and monitor the signal status.
   - **Active probes:** This enables dynamic asynchronous read and write to a flip-flop or probe point for quickly observing the output of the logic internally, or for quickly experimenting on how the logic is affected by writing to a probe point.
   - **Debug memory:** This provides the Memory Blocks tab to dynamically and asynchronously read and write to a selected FPGA fabric SRAM block.
   - **sNVM debug:** This enables reading each page or multiple pages from sNVM.
   - **Probe insertion:** This is a post-layout process that enables you to insert probes into the design and get the signals out to the FPGA package pins to evaluate and debug the design.
   - **Transceiver debug:** This feature makes debugging of high-speed serial designs easy. The JTAG interface extends access to configure, control, and observe transceiver operations and is accessible in every transceiver design. The designs are implemented using the Libero System Builder to incorporate the transceiver block enabling transceiver access from SmartDebug. The Debug TRANSCEIVER window displays real-time system and the lane status information. Transceiver configurations are supported with TCL scripting, allowing access to the entire transceiver register map for real-time customized tuning.

This application note provides a demo design to demonstrate SmartDebug's capabilities, which are used to perform real-time signal integrity testing and debugging.
2.1 **Design Requirements**

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

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>64-bit Windows 7 or 10</td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td></td>
</tr>
<tr>
<td>PolarFire Evaluation Kit (MPF300-EVAL-KIT)</td>
<td>Rev D or later</td>
</tr>
<tr>
<td>Or PolarFire Splash Kit (MPF300TS-1FCG484EES)</td>
<td>Rev 2 or later</td>
</tr>
<tr>
<td>2 SMA-to-SMA cables with 5 Gbps support.</td>
<td>Only for Evaluation Kit</td>
</tr>
<tr>
<td><strong>Software</strong></td>
<td></td>
</tr>
<tr>
<td>Libero SoC PolarFire</td>
<td>v2.3</td>
</tr>
</tbody>
</table>

2.2 **Prerequisites**

Before you start:

1. Download and install Libero SoC PolarFire v2.3 from the following location:
   
https://www.microsemi.com/products/fpga-soc/design-resources/design-software/libero-soc-polarfire#downloads

2. Download the design files from the following link:
   
- For Evaluation kit:  
  http://soc.microsemi.com/download/rsc/?f=mpf_ac479_eval_liberoscopolarfirev2p3_df
- For SPLASH kit:
  http://soc.microsemi.com/download/rsc/?f=mpf_ac479_spl_liberoscopolarfirev2p3_df
2.3 Demo Design

Figure 1 • SmartDebug Top-Level Blocks

Note: In the SPLASH kit design, the reference clock to PF_CCC is routed through a CLKINT buffer. The SPLASH and Evaluation kit designs are the same except for their floorplan and I/O constraints. In the evaluation kit design, the data is looped back from the on-board TX SMA ports to RX SMA ports. In the SPLASH kit design, the data is on-board looped back to the FPGA.

The design consists of five main blocks: the XCVR debug block (XCVR_Debug), the fabric debug block (Fabric_Debug), PF_INIT_MON block, PF_CCC, and Reset_des_sync_0 as shown in Figure 1, page 4.

Figure 2 • XCVR_Debug Overall Design Blocks

XCVR_Debug: 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 consists of a CoreSmartBERT core along with TX_PLL and XCVR_REF_CLK macros. It implements the PolarFire transceiver in PMA mode.

CoreSmartBERT: SmartBERT includes the PolarFire Transceiver, which interfaces with 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.
2.4 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.
2.5 Programming the Device

Before programming the device, SmartBERT probe related constraints need to be generated. The SmartDebug reads and writes to probe points associated with the SmartBERT IP for debugging. JTAG write to some probe points are not working as expected. This is a known issue. A software workaround is provided to determine the working probe points. The constraints need to be updated by following the steps mentioned in Appendix: Known Issues, page 26. The section Probe Points Write Issue, page 26 in "Appendix: Place and Route" chapter on page 28, describes a method to generate PDC file that locates SmartBERT IP registers in locations where the probe points work. This work around consists of:

- Programming the PolarFire FPGA with a programming file provided in the Design File package using SmartDebug Stand-Alone.
- Running a Tcl script (from SmartDebug stand-alone) to generate a template PDC file that contains the quads and lanes in the User’s design.
- Editing the template PDC file generated by running the Tcl script to match the names in the User’s design.

To program the device, see any of the following sections based on the kit used.

- Programming the Device on the Evaluation Kit, page 7
- Programming the Device on the SPLASH Kit, page 8
2.5.1 Programming the Device on the Evaluation Kit

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**

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

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 5, page 7. The following figure shows the board setup.

   **Figure 5 • 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.

![Programming the Device](image)

### 2.5.2 Programming the Device on the SPLASH Kit

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

1. Ensure that the following jumper settings are followed.

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

   **Table 3 • Jumper Settings For SPLASH Kit**

<table>
<thead>
<tr>
<th>Jumper</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J11</td>
<td>Close pin 1 and 2 for programming through FTDI chip</td>
</tr>
<tr>
<td>J5, J6, J7, J8, J9</td>
<td>Close pin 2 and 3 for programming the PolarFire FPGA through FTDI</td>
</tr>
<tr>
<td>J10</td>
<td>Open pin 1 and 2 for programming through the FTDI SPI</td>
</tr>
<tr>
<td>J4</td>
<td>Short pin 1 and 2 for manual power switching using SW1</td>
</tr>
<tr>
<td>J3</td>
<td>Open pin 1 and 2 for 1.0 V</td>
</tr>
</tbody>
</table>

2. Connect the power supply cable to the **J2** connector on the board.
3. Connect the USB cable from the Host PC to the **J1** (FTDI port) on the board.
4. Power on the board using the **SW1** slide switch.

![Board Setup (SPLASH kit)](image)
5. In the **Design Flow** window, select **Run PROGRAM Action**, as shown in **Figure 6**, page 8. This programs the design into the device.
2.6 Debugging Using SmartDebug

To debug the device using SmartDebug, follow these steps:

- Launch SmartDebug from Libero, page 10
- View Device Status, page 11
- Debug FPGA Array, page 11
- Debug μPROM, page 16
- sNVM Debug, page 16
- Debug TRANSCEIVER, page 19

2.6.1 Launch SmartDebug from Libero

On the Design Flow window:

1. Double-click on 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. Double-click SmartDebug Design.

Figure 8 • Launching SmartDebug Design

The SmartDebug window is displayed, as shown in Figure 9, page 10.

Figure 9 • SmartDebug Window Debug Options
2.6.2 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 10 • Device Status Report Sample

2.6.3 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 LSRAMs/URAMs. 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:

• Live Probes
• Active Probes
• Memory Blocks
• Probe Insertion

2.6.3.1 Live Probes

Live Probes enables the monitoring of two internal signals at a time in the design without having to repeat 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 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. 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.

Note: Live Probe feedback macro support is not enabled yet in Libero Flow.
The following steps explains 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 below image.
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 oscilloscope.

**Figure 11 • Debug FPGA Array—Live Probes**

2.6.3.2 **Active Probes**

Active Probes enables to read or change the values of probe points in a design through JTAG. 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 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 Probes** tab.

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

### 2.6.3.3 Memory Blocks

SmartDebug provides the Memory Blocks tab to dynamically and asynchronously read from and write to a selected FPGA fabric SRAM block. 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**

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

**Figure 14 • Debug FPGA Array—Memory Blocks**

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

**Figure 15 • 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 writes to the SRAM constantly. To prevent the overwrite of the changes that are forced into the SRAM, the writing is stopped by forcing A_WEN signal value to low through DIP1 (first switch of SW11). This drives a SELECT of a MUX that selects between high and low inputs. When DIP1 is asserted, A_WEN becomes low, which prevents any write from the counter to the SRAM block.

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 16, page 15.
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 13, page 13.
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.

2.6.3.4 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 17, page 16 shows the Probe Insertion tab in the Debug FPGA Array window.
2.6.4 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 18, page 16.
2. Select **Initialization** 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.

**Figure 18 • 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.

2.6.5 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 the **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 19** • sNVM Debug

![Client View Window](image)

**Figure 20**, page 18 shows the **Client View** window.
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 20 • sNVM Debug—Client View**

5. Click the **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 page number that you want to read in the **Start Page** and **Number of Bytes** in the respective boxes.

7. Click **Check Page Status**. The page status information is displayed as shown in **Figure 22**, page 19.
2.6.6 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
- SmartBERT
- Loopback Modes
- Static Pattern Transmit
- Eye Monitor

### 2.6.6.1 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.

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

**Note**: As SmartBERT is configured in CDR mode, user need to send data to get the PLL Locked (RX PLL and RX CDR PLL status will be red initially until data is sent).
2.6.6.2 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.

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

When a SmartBERT IP lane is added, the Error Injection column is displayed in the 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.

The following figure shows the Smart BERT tab and status of the TXPLL, RXPLL, Lock to Data, Data rate, and the BER.
2.6.6.3 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, follow the below mentioned 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 prbs pattern and click on **Start**.
4. Check the status of TX PLL, RX PLL, Lock to Data.
5. Click on **Stop** to stop the pattern transmission for the selected lane.

![Figure 26](https://example.com/image26.png)

2.6.6.4 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 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:

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 27](https://example.com/image27.png)

Figure 27, page 22 shows the **Static Pattern Transmit** tab.
2.6.6.5 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. 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.

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

Figure 28 • Recommended Settings for Eye Monitor

For plotting the Eye Diagram, follow the procedure enlisted below:

1. Go to EYE Monitor tab and Select LANE0.
2. Click on Power on Eye Monitor.
3. Go to SmartBERT tab, select LANE0 and choose any prbs pattern and click on 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 29, page 23 shows the Eye Monitor tab.
2.6.6.6 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.
2.6.6.6.1 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. Design default parameter options are applied to the device and updated in Modified Constraints.

2.6.6.6.2 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.

2.6.6.7 Optimize DFE
SmartDebug uses the DFE coefficients to optimize the settings for the overall signal integrity at the receiver. On-demand, the SmartDebug utility runs the adaptive algorithm of the DFE filter to resolve the TAP values of the DFE coefficients. This can be applied on any XCVR design greater than 5G.

To run ‘Optimize DFE’, follow the below steps:
1. In Debug Transceiver, go to Signal Integrity and click on Optimize DFE.
2. In Optimize DFE window the following settings
   • Select Lanes to Optimize DFE: Lane0
   • DFE Algorithm: Software Based
3. Click on Optimize DFE on selected Lanes.
Software based Optimizing DFE process is successful on all selected lanes.

2.6.6.7.1 Eye Monitor after Optimizing DFE
After Optimizing DFE, follow the steps mentioned in section Eye Monitor, page 22 for plotting the Eye Diagram.
2.7 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.
This chapter lists known issues related to SmartDebug hardware design debug and provides workarounds for each of the issues.

### 3.1 Probe Points Write Issue

The SmartDebug reads and writes to the probe points associated with the SmartBERT IP for debugging. There is a known issue where JTAG writes to some probe points do not work. The following procedure provides a workaround to ensure that this issue does not impact the functionality of the SmartBERT IP. The workaround involves generating a constraint file that ensures the design is placed only in probe points that work.

**Note:** This procedure be followed before running Place and Route in the Libero design flow.

The following files are provided in a [Active_Probes_Constraints_SB_IP](#) folder.

- DDC file ([Full_Fabric_FF.ddc](#) – a test design)
- TCL script ([Execute_probes.tcl](#) – execute from SmartDebug)
- Input file ([Input_File.txt](#) – edit before executing TCL)
- Reference_Files folder (internal use), which contains the following files:
  - sd.reference.pdc
  - SmartBERT_IP_Quad0.fp.pdc
  - SmartBERT_IP_Quad1.fp.pdc
  - SmartBERT_IP_Quad2.fp.pdc
  - SmartBERT_IP_Quad3.fp.pdc

1. Go to [Active_Probes_Constraints_SB_IP](#) folder
2. Open the Standalone SmartDebug.
3. Click **Project** > **New Project** to create a new project.
4. Import DDC Full_Fabric_FF.ddc file. Click **OK**.
5. Program the design using the **Programming Connectivity and Interface** window as shown in the following figure. Close the window when the design has been programmed.

**Figure 33 • Programming Connectivity and Interface**

6. Open [Input_Files.txt](#) in text editor. Enter Quad and number of lanes that are configured for CoreSmartBERT IP in the design.
7. Go back to the SmartDebug window and click **Project** > **Execute script** and enter the TCL script file path ([Execute_probes.tcl](#)).
8. Click **Run** to execute. This may take approximately three minutes to complete.
9. Close the Standalone SmartDebug.

The output of the TCL execution is a PDC (*.pdc) file/files that can be used in the Libero flow.
10. Go to the current directory and locate the Output_PDC_Files folder.
    Generated PDC file contains a list of registers used in CoreSmartBERT IP.
11. Import the PDC files into the design.
12. Replace the top-level name for each constraint mentioned in the PDC file with the hierarchy name of
the IP in the design.
13. If multiple hierarchy levels are present, include all levels in the space specified in the PDC file.
   For example, if design has CoreSmartBERT IP with component name SmartBERT_IP_0, replace
   "<Enter_module_path_here>" with "SmartBERT_IP_0" for each location constraint in the PDC file.

   **Note:** For this demo design, replace `<Enter_module_path_here>` with
   XCVR_Debug_0/SmartBert_xcvr_chk_0. Use the updated pdc file instead of the file provided in the
design (user.pdc).

### 3.2 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 DFE and EM blocks are turned
off during normal operation to save power. This issue does not impact the 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: 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 above mentioned steps & to Place and Route the design, follow the below mentioned steps:

1. On the I/O Attributes tab, select the check box next to the io_cons.pdc file, as shown in Figure 34, page 28. The io_cons.pdc file contains the I/O assignment for reference clock, switches and XCVR Lanes.

2. From the Edit drop-down list, select Edit with I/O Editor, as shown in Figure 35, page 28.

3. The I/O Editor will open as shown in Figure 36, page 28.

4. Select the XCVR View in the I/O Attribute editor. This view allows the transceiver components to be placed.

5. Place TX_PLL, XCVR_REF_CLK, and XCVR_Debug as shown in Figure 37, page 29 for Evaluation kit or Figure 38, page 29 for 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.
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 40**, page 30.
5 Appendix: 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 UG0773: PolarFire SmartDebug User Guide.
- For more information about PolarFire transceiver blocks, see UG0677: PolarFire FPGA Transceiver User Guide.
- For more information about PF_CCC, see UG0684: PolarFire FPGA Clocking Resources User Guide.
- For more information about Libero, 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 PF_UPROM, PF_URAM, and PF_DPSRAM, see Libero catalog.
- For more information about Identify RTL, see Synopsis Identify RTL User Guide.