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 6.0 ............................................................ 1
  1.2 Revision 5.0 ............................................................ 1
  1.3 Revision 4.0 ............................................................ 1
  1.4 Revision 3.0 ............................................................ 1
  1.5 Revision 2.0 ............................................................ 1
  1.6 Revision 1.0 ............................................................ 1

2 Debugging PolarFire FPGA Designs ........................................... 2
  2.1 Design Requirements .................................................. 3
  2.2 Prerequisites .......................................................... 3
  2.3 Demo Design .......................................................... 4
     2.3.1 PF_CCC ....................................................... 4
     2.3.2 PF_INIT_MON ............................................... 4
     2.3.3 reset_des_sync ............................................... 4
     2.3.4 XCVR_Debug .................................................. 4
     2.3.5 Fabric_Debug .................................................. 5
  2.4 Clocking Structure .................................................... 6
  2.5 Reset Structure ....................................................... 6
  2.6 Enabling FHB ........................................................ 7
  2.7 Programming the Device ............................................. 7
     2.7.1 Programming the Device on the Evaluation Kit ............ 7
     2.7.2 Programming the Device on the SPLASH Kit .............. 9
  2.8 Debugging Using SmartDebug ......................................... 10
     2.8.1 Launch SmartDebug from Libero ............................. 10
     2.8.2 View Device Status ......................................... 11
     2.8.3 Debug FPGA Array .......................................... 11
     2.8.4 Use FHB ..................................................... 16
     2.8.5 Debug µPROM ................................................. 22
     2.8.6 sNVM Debug .................................................. 22
     2.8.7 Debug TRANSCEIVER ....................................... 24
  2.9 Conclusion .......................................................... 31

3 Appendix 1: Known Issues ................................................... 32
  3.1 Data Traffic Errors on XCVR Lanes in CDR Mode .................. 32

4 Appendix 2: Place and Route ................................................ 33

5 Appendix 3: References ..................................................... 35
Figures

<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>5</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Fabric_Debug Overall Design Blocks</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Clocking Structure</td>
<td>6</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Reset Structure</td>
<td>6</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Enabling FHB</td>
<td>7</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Board Setup (Evaluation kit)</td>
<td>8</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Programming the Device</td>
<td>8</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Board Setup (SPLASH kit)</td>
<td>9</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Launching SmartDebug Design</td>
<td>10</td>
</tr>
<tr>
<td>Figure 11</td>
<td>SmartDebug Window Debug Options</td>
<td>10</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Device Status Report Sample</td>
<td>11</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Debug FPGA Array—Live Probes</td>
<td>12</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Debug FPGA Array—Active Probes</td>
<td>13</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Pseudo-static Signal Polling</td>
<td>13</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Debug FPGA Array—Memory Blocks</td>
<td>14</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Memory Blocks—Read Block</td>
<td>15</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Memory Blocks—Write Block</td>
<td>15</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Debug FPGA Array—Probe Insertion</td>
<td>16</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Event Counter</td>
<td>17</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Frequency Monitor</td>
<td>18</td>
</tr>
<tr>
<td>Figure 22</td>
<td>User Clock Frequencies</td>
<td>18</td>
</tr>
<tr>
<td>Figure 23</td>
<td>Select an FHB</td>
<td>19</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Adding FHB Trigger to Active Probes</td>
<td>19</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Selecting Clock Domains</td>
<td>20</td>
</tr>
<tr>
<td>Figure 26</td>
<td>VCD to WLF Conversion</td>
<td>21</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Adding Signals</td>
<td>21</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Wave Window</td>
<td>21</td>
</tr>
<tr>
<td>Figure 29</td>
<td>µPROM Debug</td>
<td>22</td>
</tr>
<tr>
<td>Figure 30</td>
<td>sNVM Debug</td>
<td>23</td>
</tr>
<tr>
<td>Figure 31</td>
<td>sNVM Debug—Client View</td>
<td>23</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Secured NV Details</td>
<td>24</td>
</tr>
<tr>
<td>Figure 33</td>
<td>sNVM Debug—Page View</td>
<td>24</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Configuration Report</td>
<td>25</td>
</tr>
<tr>
<td>Figure 35</td>
<td>Debug TRANSCEIVER—Smart BERT</td>
<td>26</td>
</tr>
<tr>
<td>Figure 36</td>
<td>Smart BERT—Error Counter</td>
<td>26</td>
</tr>
<tr>
<td>Figure 37</td>
<td>Debug TRANSCEIVER—Loopback Modes</td>
<td>27</td>
</tr>
<tr>
<td>Figure 38</td>
<td>Static Pattern Transmit</td>
<td>27</td>
</tr>
<tr>
<td>Figure 39</td>
<td>Recommended Settings for Eye Monitor</td>
<td>28</td>
</tr>
<tr>
<td>Figure 40</td>
<td>Debug TRANSCEIVER—Eye Monitor</td>
<td>29</td>
</tr>
<tr>
<td>Figure 41</td>
<td>Signal Integrity</td>
<td>29</td>
</tr>
<tr>
<td>Figure 42</td>
<td>Optimize Receiver</td>
<td>30</td>
</tr>
<tr>
<td>Figure 43</td>
<td>Eye Diagram after using Optimize Receiver</td>
<td>31</td>
</tr>
<tr>
<td>Figure 44</td>
<td>I/O Attributes Tab</td>
<td>33</td>
</tr>
<tr>
<td>Figure 45</td>
<td>Editing with I/O Editor</td>
<td>33</td>
</tr>
<tr>
<td>Figure 46</td>
<td>I/O Editor</td>
<td>33</td>
</tr>
<tr>
<td>Figure 47</td>
<td>Placing the TX_PLL, XCVR_REF_CLK, and PF_XCVR Components (Evaluation kit)</td>
<td>34</td>
</tr>
<tr>
<td>Figure 48</td>
<td>Placing the TX_PLL, XCVR_REF_CLK, and PF_XCVR Components (SPLASH kit)</td>
<td>34</td>
</tr>
<tr>
<td>Figure 49</td>
<td>Constraint Files on the I/O Attributes Tab</td>
<td>34</td>
</tr>
<tr>
<td>Figure 50</td>
<td>Place and Route</td>
<td>34</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>9</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 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.

1.2 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 Use FHB, page 16.
• Updated the supported Eye Scan Modes in Eye Monitor, page 28.

1.3 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.

1.4 Revision 3.0
The document was updated for Libero SoC PolarFire v2.2.

1.5 Revision 2.0
The document was updated for Libero SoC PolarFire v2.1.

1.6 Revision 1.0
The first publication of this document.
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, 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 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.

   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 writes 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.
   - **FPGA Hardware Breakpoint (FHB):** This enables force halting or halting the design by controlling the clock domains and enables control over the debugging cycle. You can step on the clock domains and halt them either selectively or all at once.
   - **Debug Memory:** This enables dynamic and asynchronous read from 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 the debugging of high-speed serial designs easy. The JTAG interface extends access to control, configure, 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 for Transceiver and debugging.
2.1 Design Requirements

The following table lists the hardware and software 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>Hardware</td>
<td></td>
</tr>
<tr>
<td>PolarFire Evaluation Kit (MPF300T-1FCG1152I) Rev D or later</td>
<td></td>
</tr>
<tr>
<td>Or</td>
<td></td>
</tr>
<tr>
<td>PolarFire Splash Kit (MPF300T_ES-1FCG484I) Rev 2 or later</td>
<td></td>
</tr>
<tr>
<td>2 SMA-to-SMA cables with 5 Gbps support</td>
<td>Only for Evaluation Kit</td>
</tr>
<tr>
<td>Software</td>
<td></td>
</tr>
<tr>
<td>Libero® SoC</td>
<td></td>
</tr>
</tbody>
</table>

**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. For demo design files download link:
   • For Evaluation kit: http://soc.microsemi.com/download/rsc/?f=mpf_ac479_eval_df
   • For Splash kit: http://soc.microsemi.com/download/rsc/?f=mpf_ac479_spl_df
2.3 Demo Design

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

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 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 4
5. Fabric_Debug, page 5

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.
2.3.5 Fabric_Debug

Figure 3 shows the IP blocks inside the Fabric_Debug block.

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 LSRAM instance (DPSRAM). The data value of the LSRAM block is the same as the address value of the block. On the read side of the LSRAM, 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 LSRAM 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.


2.4  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 4**  Clocking Structure

1. On SPLASH kit, 125 MHz crystal oscillator is used.
2. On SPLASH kit, Clock Domain 1 is 125 MHz.

2.5  Reset Structure

**Figure 5** shows the reset structure used in the design.

**Figure 5**  Reset Structure

1. On SPLASH kit, SYS_RESET_N is mapped to N4.
2.6 Enabling FHB

To enable FHB, perform the following steps:

1. Go to Project > Project Settings. The Project settings dialog box is displayed.
2. Select Design flow in the left pane. The design flow settings are displayed in the right pane.
3. Under Root SmartDebug_Top, select Enable FPGA Hardware Breakpoint Auto Instantiation check box as shown in Figure 6.

Figure 6 • Enabling FHB

4. Click Save to save the settings and then click Close to close Project Settings.

Note: FHB is already enabled in the provided design. For more information about using FHB, see Use FHB, page 16.

2.7 Programming the Device

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 9

2.7.1 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, and 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>J17</td>
<td>Short pin 1 and 2</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 7. The following figure shows the board setup.

*Figure 7 • 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 8 • Programming the Device*
2.7.2 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, and 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.

**Figure 9 • Board Setup (SPLASH kit)**

5. In the Design Flow window, select Run PROGRAM Action, as shown in Figure 8. This programs the design into the device.
2.8 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
• Use FHB, page 16
• Debug µPROM, page 22
• sNVM Debug, page 22
• Debug TRANSCEIVER, page 24

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 10* • Launching SmartDebug Design

The SmartDebug window is displayed, as shown in *Figure 11.*

*Figure 11* • 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.

![Device Status Report Sample](image)

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 LSRAMs/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 11
- Active Probes, page 12
- Memory Blocks, page 14
- Probe Insertion, page 16

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).

To use Live Probes, reserve pins using Reserve Pins for Probes under Constraints Manager in Libero SoC. 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 the 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 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 13.
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 13**  •  **Debug FPGA Array—Live Probes**

### 2.8.3.2 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 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, 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.
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 15 • 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. 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, perform the following 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.

![Debug FPGA Array—Memory Blocks](image)

5. Click Read Block. The specified memory block is read as shown in the following figure.
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 18.

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 15.
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 19 shows the Probe Insertion tab in the Debug FPGA Array window.

Figure 19 • Debug FPGA Array—Probe Insertion

2.8.4 Use 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 17
- Use Frequency Monitor, page 17
- Use User Clock Frequency, page 18
- Select FHB, page 18
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 20. 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 20.

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

Figure 20 • 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.
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 22. Live probes are temporarily unavailable till all the user clock frequencies are calculated and displayed.

Figure 22 • 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 23. 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 23.
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 24.

Figure 24 • Adding FHB Trigger to Active Probes

6. Select Operate on All Clock Domains as shown in the following figure.
7. Click Arm Trigger as shown in Figure 25. 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 25, 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` command as shown in Figure 26.

Figure 26 • VCD to WLF Conversion

```
<file1.vcd> <file2.wlf>
```

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 27.

Figure 27 • Adding Signals

5. Open the Wave window. Observe that the error signal is not asserted indicating data written and data read from DPSRAM is same as shown in Figure 28.

Figure 28 • 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 29.
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 29 to verify the content.

**Figure 29 • µPROM Debug**

![µPROM Debug](image)

**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 30 • sNVM Debug

![sNVM Debug](Image)

Figure 30 shows the Client View window.

Figure 31 • sNVM Debug—Client View

![sNVM Debug—Client View](Image)

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 32 • 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 33.

Figure 33 • 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.

![Configuration Report](image)

**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.
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.

### 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.
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 38 shows the Static Pattern Transmit tab.
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 39 • 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 40 shows the Eye Monitor tab.*
2.8.7.6 **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.

**Figure 41 • Signal Integrity**
2.8.7.6.1 **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.

2.8.7.6.2 **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.

2.8.7.7 **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 42 • **Optimize Receiver**

![Optimize Receiver Window](image)
2.8.7.7.1 Eye Monitor after Optimizing Receiver

After Optimizing Receiver, follow the steps mentioned in section Eye Monitor, page 28 for plotting the Eye Diagram.

**Figure 43 • Eye Diagram after using Optimize Receiver**

![Eye Diagram after using Optimize Receiver](image)

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.
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.
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 44, page 33. 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 45, page 33.

3. The I/O Editor will open as shown in Figure 46, page 33.

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 47, page 34 for Evaluation kit or Figure 48, page 34 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 50, page 34.
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 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.