AC492
Application Note
Running BareMetal User Applications on PolarFire SoC FPGA

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

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

51900492. 2.0 1/21

Contents

Figures

1 Revision History

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

1.1 Revision 2.0

Updated for Libero SoC v12.6.

1.2 Revision 1.0

The first publication of this document.

2 Running BareMetal User Applications on PolarFire SoC FPGA

Microchip's PolarFire® SoC FPGAs include industry's first RISC-V based Microprocessor Subsystem (MSS) and a fabric that inherits all the features of the PolarFire family. The PolarFire SoC MSS includes 5x 64-bit RISC-V processor cores, AXI Switch, DDR Controller, fabric Interface Controllers (FIC), and a rich set of peripherals. For more information about the PolarFire SoC MSS and its components, see *UG0880: PolarFire SoC FPGA MSS User Guide*. PolarFire SoC FPGAs are ideal for running BareMetal applications.

PolarFire SoC standalone MSS Configurator (no separate license required) facilitates independent configuration of MSS for use in Libero® SoC and SoftConsole. The standalone MSS configurator provides a seamless interface and can be used by Embedded Software developers and FPGA designers. For more information, see *PolarFire SoC Standalone MSS Configurator User Guide*. FPGA designers can add fabric components by using Libero SoC SmartDesign and IP Library.

Microchip's MPFS-ICICLE-KIT features the MPFS250T_ES SoC FPGA, a rich set of peripherals, ROM, and RAM options. For ROM requirement, the kit includes an 8 GB eMMC Flash memory. For external RAM requirement, the kit includes a 2 GB LPDDR4 device. The kit also features an SD Card slot. For more information, see *UG0882: PolarFire SoC FPGA ICICLE Kit User Guide*.

This application note describes how to run BareMetal user applications on MPFS-ICICLE-KIT.

2.1 Prerequisites

Before running the BareMetal user applications, the ICICLE kit must be programmed with the ICICLE reference design. Ensure to follow the documentation provided on *GitHub* and program the ICICLE kit reference design with one of the FlashPro Express programming files.

3 Running the BareMetal User Applications

Before you start, ensure to program the ICICLE KIT as described in Prerequisites. After successful programming, the following user applications can be run:

- System Services
- Single-Bit Error Detection and Correction for L2 LIM
- MicroPython

This document describes the steps to run the System Services application.

Note: For information about running other BareMetal user applications, see the *baremetal_applications GitHub* web page.

3.1 System Services

System Services SoftConsole project includes the following source files for running System Services:

- BareMetal MSS System Services driver which is available at the following location: *https://github.com/polarfire-soc/polarfire-soc-bare-metal-library/tree/master/src/platform/drivers/mss_sys_services*
- PolarFire SoC hardware abstraction layer source code (`mpfs_hal`). Using the `mpfs_hal` code, the user can access all of the PolarFire SoC MSS registers like E51 and U54 local interrupt registers, L2 cache, MPU, segmentation blocks, and others registers. This folder also includes the MSS peripherals base address file.

Note: Hardware parameters like MSS clocks, memory, and peripheral source files are generated in SoftConsole using the MSS XML configuration file. The MSS XML configuration file is available at *https://github.com/polarfire-soc/icicle-kit-reference-design/tree/master/XML.*

3.1.1 Running the System Service Application

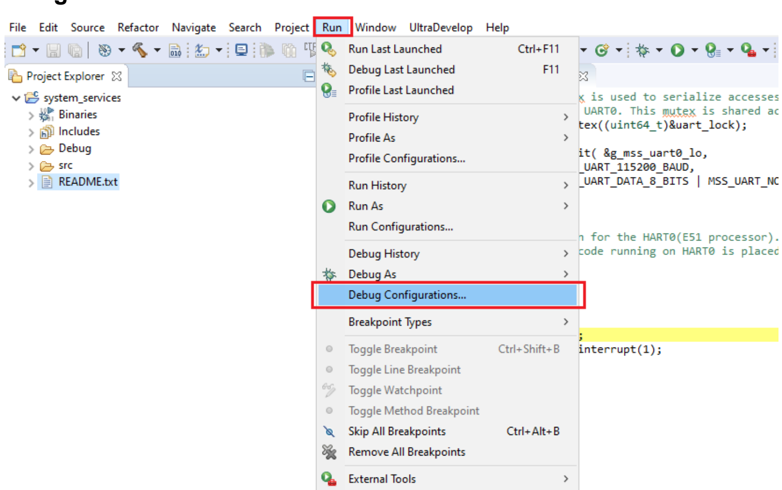
This section describes how to launch the System Services application in debug mode using SoftConsole and run the PolarFire SoC System Services via serial interface.

Note: This SoftConsole project can also be built in release mode and run from eNVM. Select Run > External Tools > PolarFire SoC program non secure boot mode 1 option to program the eNVM with the application and execute it.

To run the System Services:

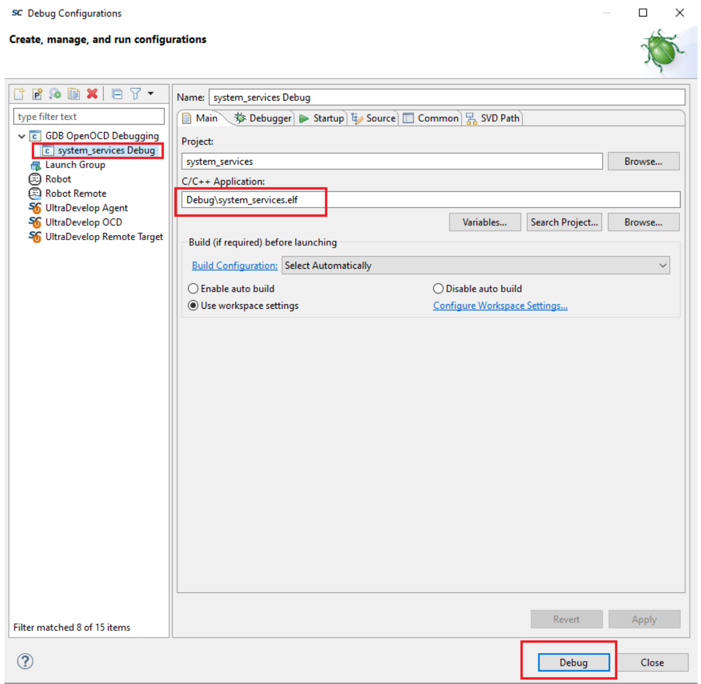
1. Launch SoftConsole with the provided System Services project.
2. Click **Run->Debug Configurations**.

Figure 1 • **Debug Configurations**



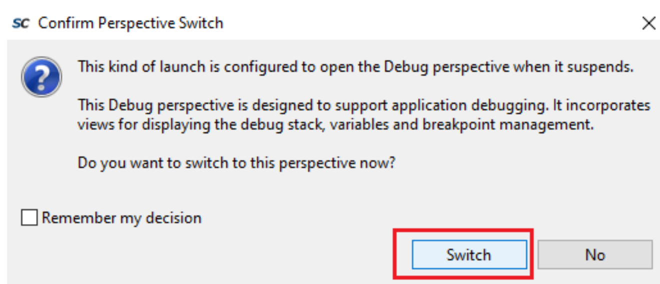
3. In the Debug Configurations window, select the debug project highlighted in Figure 2.
4. Select the **Debug** option in the dialog box for launching the application in debug mode.

Figure 2 • **Launching the Debugger**



5. In the **Confirm Perspective Switch** window, click **Switch** to go to the debug view.

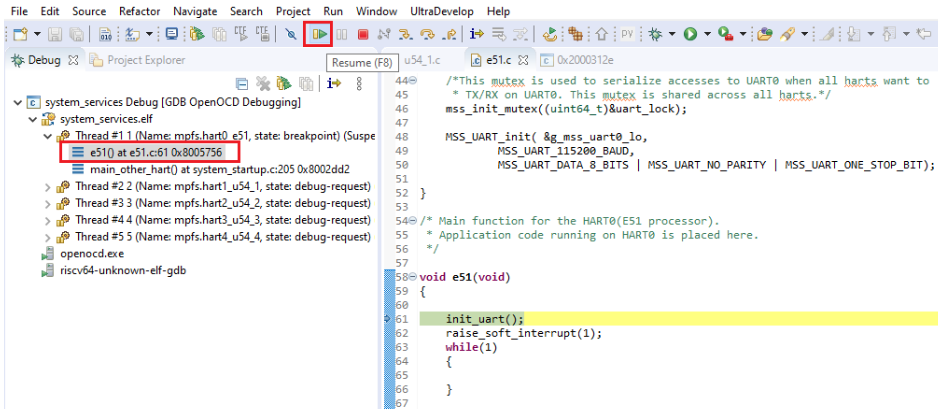
Figure 3 • **Confirm Perspective Switch Message**



The debugger stops at the `void e51 (void)` startup function in the `e51.c` file. This function initializes the UART interface and sends a software interrupt to the U54_1 processor core and releases it from the wait for interrupt mode (WFI). The `u54_1` hart starts executing the System Services functions.

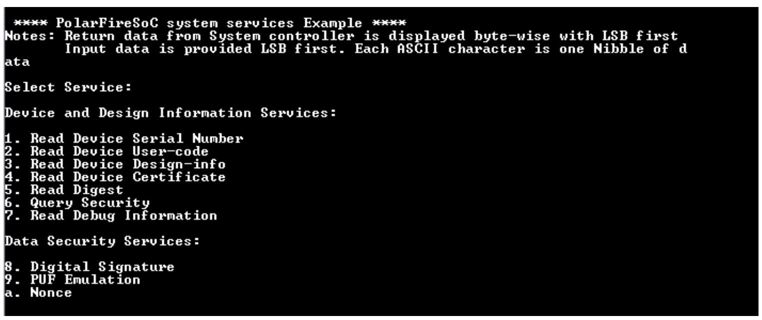
6. Click Resume to start running System Services application as shown in Figure 4.

Figure 4 • Resume Debugging



7. System Services options are displayed on the terminal, as shown in Figure 5. For information about each System Service and output bit fields, see *UG0905: PolarFire SoC FPGA System Services User Guide*.

Figure 5 • System Services Options



```
**** PolarFireSoC system services Example ****
Notes: Return data from System controller is displayed byte-wise with LSB first
       Input data is provided LSB first. Each ASCII character is one Nibble of d
ata

Select Service:

Device and Design Information Services:

1. Read Device Serial Number
2. Read Device User-code
3. Read Device Design-info
4. Read Device Certificate
5. Read Digest
6. Query Security
7. Read Debug Information

Data Security Services:

8. Digital Signature
9. PUF Emulation
a. Nonce
```

8. Enter 1 to select **Read Device Serial Number**.
   The 128-bit Read Device Serial Number (DSN) is displayed, as shown in Figure 6.

Figure 6 • Read Device Serial Number

```
--------------------------------------------------------------------------------
Device serial number:  E24BABFEB7CA08B639F6FDB7D0E9EB45
--------------------------------------------------------------------------------
```

Each PolarFire SoC FPGA device has a unique, publicly readable, 128-bit DSN. The DSN can be used in cryptographic protocols to uniquely identify the device. For more information, see *UG0918: PolarFire SoC FPGA Security User Guide*.

9. Enter 2 to select **Read Device User-code**.
   This can be configured in the Libero SoC project from Design flow->Program Design->Configure Programming Options.

   The 32-bit device USERCODE/Silicon signature is displayed, as shown in Figure 7.

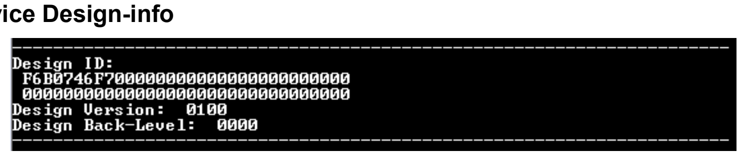
Figure 7 • Read Device User-code

```
--------------------------------------------------------------------------------
32bit USERCODE/Silicon signature (MSB first):  ABCDEF01
--------------------------------------------------------------------------------
```

Note: In the Libero project, this USERCODE/silicon signature can be configured from Design flow->Program Design->Configure Programming Options. If the values are not entered, you can see zero.

10. Enter 3 to select **Read Device Design-info**.

Figure 8 • Read Device Design-info



```
--------------------------------------------------------------------------------
Design ID:
 F6B0746F700000000000000000000000
 00000000000000000000000000000000
Design Version:  0100
Design Back-Level:  0000
--------------------------------------------------------------------------------
```

The device design information consists of:

- 256-bit user-defined Design ID.
- 16-bit design version: This can be configured from Design flow->Program Design->Configure Programming Options. In auto update programming, the current design version is compared with the available images in external SPI flash to initiate the auto update on power up.
- 16-bit design back-level: This can be configured from Design flow->Program Design->Configure Security. When back level protection is enabled, the device can only be programmed if the target design version is more than the back-level value.

11. Enter 4 to select **Read Device certificate**.
    The device supply chain assurance certificate is displayed, as shown in Figure 9.

Figure 9 • **Read Device Certificate**

```
Device Certificate:
 308203CC30820352A003020102021240
 3636A5D6BD9C40DB2D3CB1750944F993
 01300A06082A8648CE3D040303304931
 0B3009060355040613025553310C300A
 060355040B130350524F310D300B0603
 55040A13044D434850311D301B060355
 0403131434613633376165653135616 4
 63313939336435623020170D31353130
 31343030303734375A180F3939393931
 32333132333535393535395A304E314C300B
 060355042C1304303030303013060355
 04040C0C506F6C61724669726553 6F43
 3028060355042A0C214D504653323530
 54532020202020202020202020202020
 20202020202020202020203076301006 07
 2A8648CE3D020106052B810400220362
 0004F084E4D0366BD91251EEB056E0A5
 B92143B025683243749EF045E0BBF946
 90544C35E1238E233E03FF31D3007EF6
 B62AC6B311B96B92E291E7568F11F162
 133B5FEEE09C75CF2AAC832D32C412B7
 06BEE71A55977E7831548ED33C86561D
 D11E81090040364A662911B600821100
 D6334BB5A2A35A4CD771E9C50602BDBB
 A38201D6308201D23012060A2B060104
 0182BD64010104040000000030818F06
 0A2B0601040182BD640102048180FFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFF003012
 060A2B0601040182BD64010304040100
 000030818606 0A2B0601040182BD6401
 04047800000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 000000000000000000000030818C060A
 2B0601040182BD640105047E00000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000300A06082A86
 48CE3D0403030368003065023 06FC8C1
 B9C88E1DF613F30923C707A3236C67C3
 CB4A012B57967F99D5B6BB48D3F54791
 B85A9C2EAD762DC8BBF1B27905023100
 B9BFAE5381A930338844A97B3B7BBAE3
 F4B861AA1BEF217FC11A86418416B36D
 861FA107AB93A02466241880B60BEA42
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
```

The Device Certificate is a 1024 bytes Microsemi-signed X-509 certificate programmed during manufacturing. The certificate is used to guarantee the authenticity of a device and its characteristics.

12. Enter 5 to select **Read Digest**.

    The 576 bytes Digest contains the fabric digest, sNVM digest, and user key digests. The digest protects data integrity. Figure 10 shows the 576 bytes digest.

Figure 10 • **Read Digest**

```
Read Digest:
 98DDC8BD2892E8427A5A2FC92DE86C14
 BE721C795FCBE6FFAB9A4BBFE165169F
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 3880781015D4B19CC8CBAB0EDD682120
 C6BB7851335F7C410312397F01C88F50
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 00000000000000000000000000000000
 2EA9AB9198D1638007400CD2C3BEF1CC
 745B864B76011A0E1BC52180AC6452D4
 F5A5FD42D16A20302798EF6ED309979B
 43003D2320D9F0E8EA9831A92759FB4B
 00000000000000000000000000000000
 00000000000000000000000000000000
 2EA9AB9198D1638007400CD2C3BEF1CC
 745B864B76011A0E1BC52180AC6452D4
 F5A5FD42D16A20302798EF6ED309979B
 43003D2320D9F0E8EA9831A92759FB4B
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 0243D7630DD1F1CA4C5B93988B4B1D2A
 4F7676103B9D88F34DCF228E044F4BA0
 F5A5FD42D16A20302798EF6ED309979B
 43003D2320D9F0E8EA9831A92759FB4B
 00000000000000000000000000000000
 00000000000000000000000000000000
 EC5E6321EAD8D054484CD9F6F98DCC28
 9F3C58913D36EDE2ED95174E15EB1204
 B5F0B8CB89DAD4031B7B8B57ED90F2AE
 CC5AF9197C26BE8387F4C0C73A8D5348
```

13. Enter 6 to select **Query Security**.

    The non-volatile states of user security lock is displayed, as shown in Figure 11.

    The design does not include any user security settings for device security. Security locks can be configured from Design flow->Program and Debug Design->Configure Security.

Figure 11 • **Security Locks**

```
Security locks:  000000000000000000357AB801000000
 FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 00
```

14. Enter 7 to select **Read Debug Information**.
    The debug information is displayed, as shown in Figure 12. In Figure 12, the highlighted 4 bytes E1000000 (LSB first) indicates the number of times the device was programmed (programming cycles).

Figure 12 • **Read Debug Information**

```
Debug info:
 000000CE280021002300CF09C409CB09
 0A07B902CC1900000000000000000000
 C101C44C0171198AFF00FFFF00000000
 0003FFFFCA0000000440400000E1000000
 0080781015D4B19C0000000000010010
 FF3E00007E00030003000000000000
```

15. Enter 8 to select **Digital Signature**.

The digital signature in both Raw and DER formats is displayed, as shown in Figure 13.

Figure 13 • **Digital Signature**

```
----------------------------------------
Digital Signature service:
48 byte hash value:
 1B50C2118989595748080084B47DD499D
 1CD656DC4C18A187F4CE58A6265F689D
 1B50C2118989595748080084B47DD499D
Raw format:
Digital Signature service successful.
Output Digital Signature - Raw format:
 FCFACE438477C71DC46E37A95C996FD4
 223BD44547B1C914B5800647FFFC1D13
 865DB760968604BC61975E8936725A45
 3287F42FEDDAC4F6452F14151BD0C754
 BF18EA57D677578D1F6EF4D73E9E40BA
 7EEF45F2D33F562A99D5243B2ECA61A8
DER format:

Digital Signature service successful.
Output Digital Signature - DER format:
 3066023100A83885070B274618CAF277
 33B889CD6A8B870D4B0D51966BB9B2DC
 EE5A86AB58EFBB580A4EC51EE1D22CE1
 540251979A023100B9DD245769CE911E
 00C2125DFC2E777A0352BB5AE5E77959
 6418BB7005F2B289963A2E862643ACEA
 EA6BB2DB9863DB76
```

The digital signature service takes a user supplied SHA384 hash and signs it with the devices private key. The application randomly generates the SHA384 hash value. The Digital Signature service sends the hash value to the System Controller. The System Controller Cryptoprocessor runs the Elliptic Curve Digital Signature Algorithm (ECDSA) using the hash and the device private key to generate the signature.

16. Enter 9 to select PUF Emulation service.

    The PUF emulation service provides a mechanism for authenticating a device, or for generating a pseudo-random bit strings that can be used for different purposes. When this service is selected, the service by default accepts a 128-bit challenge and an 8-bit optype, and returns a 256-bit response unique to the challenge and the optype as shown in Figure 14.

Figure 14 • **PUF Emulation Service**

```
------------------------------------------------------
The challenge OPTYPE range(0x0 to 0xFF):  C8
16 byte challenge:
 C8B02F4FB7F702B1A08AD1FECBACCC63
PUF emulation service successful.Generated Response:
 23C8C11D424C6837C1DFCCBFAFE60D05
 59C96927D179026E187E1B1FDE996DB6
------------------------------------------------------
```

17. Enter “a” to select **Nonce service**.

The 32 bytes nonce value is generated by the device as shown in Figure 15.

Figure 15 • **Nonce Value**

```
Generated Nonce:
 897DF0D3A3A6F64963A616995A7602FA
 D8441714A2AFE3964897F6BD524E7C71
----------------------------------------
```

This concludes the running System Services application and the SoftConsole debug session can be terminated.