

# PolarFire<sup>®</sup> SoC Software Development and Tool Flow User Guide

## Introduction

The PolarFire SoC tool flow has been constructed to allow embedded designers and FPGA designers to develop applications in the domain of their choice. Embedded designers and FPGA designers prefer using a tool flow they are familiar with. The recommended starting point when designing with PolarFire SoC FPGA is the PolarFire SoC MSS Configurator tool that graphically guides the user to define the initialization parameters for the microprocessor subsystem, MSS peripherals, DDR, and the interfaces between the processor subsystem and the FPGA fabric. The tool is also used to configure MSS I/O.

The following flow diagram illustrates the high-level tool flow.

#### Figure 1. High-Level Tool Flow



After the user completes a configuration, the MSS configurator exports the files needed for the embedded software development flow and FPGA developers. The XML file contains the system configuration information needed to

generate header files for Bare Metal system startup code included in the Hart Software Services (HSS) to configure the microprocessor subsystem to the expected state. The <project\_name>.cxz can be imported into Libero® SoC (v12.5 and above) and used by the FPGA designer to interface their design to the processor subsystem. The PolarFire SoC MSS Configurator tool is used to change the state of the microprocessor subsystem or any of the interfaces between the FPGA fabric and the microprocessor subsystem.

As the design progresses through the development process, different types of data are shared between the FPGA designer (Libero SoC) and the embedded designer (SoftConsole). The following are some of the examples.

- SoftConsole: outputs that can be part of the FPGA bitstream or programmed directly through SoftConsole using the FlashPro hardware.
  - Boot Mode configuration
  - Secure Boot Device Certificate
  - Embedded Non-Volatile Memory (eNVM) binary
  - Secure Non-Volatile Memory (sNVM) binary
- Libero SoC
  - FPGA memory map
  - FPGA design

The FPGA designer in collaboration with the embedded software designer defines and refines the MSS memory map within the FPGA. The files generated by the PolarFire SoC MSS Configurator must be shared with the embedded software developer for further development.

## References

- For information about the PolarFire SoC MSS, see PolarFire SoC MSS Technical Reference Manual.
- For information about MSS peripherals, see PolarFire SoC MSS Technical Reference Manual.
- For information about device power-up, see UG0890: PolarFire SoC FPGA Power-Up and Resets User Guide.
- For more information about Bare Metal, Yocto, and Buildroot applications, see GitHub.
- · For information about Yocto, see Yocto Project Reference Manual.
- For information about Buildroot, see Buildroot User Manual.
- For more information about PolarFire SoC MSS Configurator, see PolarFire SoC MSS Configurator.
- For more information about how to boot Linux on Icicle kit using eMMC, see GitHub.

# Acronyms

Acronym	Expanded
DTIM	Data Tightly Integrated Memory
DTS	Device Tree Source
eMMC	Embedded Multi-Media Controller
eNVM	Embedded Non-Volatile Memory/BootFlash
FSBL	First Stage Boot Loader
HAL	Hardware Abstraction Layer
Hart	Hardware Thread/Core/Processor Core
HSS	Hart Software Services
μPROM	Micro Programmable Read-Only Memory
LIM	Loosely Integrated Memory
MSS	Microprocessor Subsystem
MPFS	Microchip PolarFire SoC
OpenSBI	Open Source Supervisor Binary Interface
PMP	Physical Memory Protection
POR	Power-on Reset
PUF	Physically Unclonable Function
ROM	Read-only Memory
SBIC	Secure Boot Image Certificate
sNVM	Secure Non-volatile Memory
SRAM	Static Random-Access Memory
SSBL	Second Stage Boot Loader
WFI	Wait for Interrupt
ZSBL	Zero Stage Boot Loader

The following acronyms are used in this document.

# **Table of Contents**

Intro	oductio	n	1
Ref	erence	S	2
Acr	onyms.		3
1.	Devel	opment Tools	6
	1.1.	PolarFire SoC MSS Configurator	6
	1.2.	Libero® SoC	7
	1.3.	SoftConsole	7
	1.4.	FlashPro Express	26
	1.5.	RISC-V GCC Bare Metal	26
	1.6.	RISC-V Linux Toolchain	26
	1.7.	Yocto	.26
	1.8.	Buildroot	27
	1.9.	SmartDebug	27
	1.10.	Identify	28
2.	Softwa	are Stack	29
	2.1.	RISC-V Libraries	29
	2.2.	Hart Software Services (HSS)	30
	2.3.	Bare Metal Library	31
	2.4.	Linker Scripts	32
	2.5.	Linux Images	33
	2.6.	FreeRTOS <sup>™</sup>	34
	2.7.	Third Party Tools	35
3.	Applic	ation Development	36
	3.1.	Device Boot and Configuration Process	.36
	3.2.	Boot Mode 0-Idle Boot	.38
	3.3.	Boot Mode 1-Direct Boot from eNVM	38
	3.4.	Clock Management	42
	3.5.	Physical Memory Protection (PMP)	.42
	3.6.	Generating Boot Images	43
	3.7.	Bare Metal Development	48
	3.8.	Linux Application Development	53
4.	Apper	ndix	62
5.	Revisi	on History	63
The	Micro	shin Wehsite	64
			04
Pro	auct Cl	nange Notification Service	64
Cus	tomer	Support	64
Mic	rochip	Devices Code Protection Feature	64
Leg	al Noti	Ce	65

Trademarks	. 65
Quality Management System	. 66
Worldwide Sales and Service	67

# 1. Development Tools

PolarFire SoC comes with a suite of tools to help create a complete hardware and software solution. The following table lists the suite of tools available for creating the FPGA and embedded design targeted for PolarFire SoC.

ΤοοΙ	Description
PolarFire SoC MSS Configurator	A standalone tool to configure the MSS clock frequencies, peripherals, DDR, fabric interfaces, and MSS I/O configuration.
Libero SoC	Standard Microchip tool to configure the programmable section of the PolarFire SoC FPGA.
SoftConsole	Software development platform to develop and debug the Bare Metal and RTOS applications, which also includes debugging the software.
FlashPro Express	Available as a standalone tool or integrated as part of a Libero installation. Used for programming the MSS and programmable logic of the FPGA.
RISC-V GCC Bare Metal	The RISC-V GCC toolchain bundled with SoftConsole for Bare Metal development.
Yocto	An open source project to create $Linux^{^{(\!$
Buildroot	A tool to configure and generate embedded Linux distributions.
SmartDebug and Identify	Available as a standalone tool or integrated as part of Libero to debug the hardware in the MSS and programmable logic.
	Identify is the Embedded Logic Analyzer tool for Microchip FPGA devices offered as part of the Libero SoC software tool suite.

### 1.1 PolarFire SoC MSS Configurator

The PolarFire SoC MSS Configurator is a common tool to configure the PolarFire SoC MSS. It provides a seamless experience for the embedded software developers targeting the MSS and hardware engineers developing a solution using the MSS and the FPGA fabric. The PolarFire SoC MSS Configurator application is available in two options:

- As a standalone application
- As part of the Libero SoC Design Suite v12.5 and later

### 1.1.1 Installation

The PolarFire SoC MSS Configurator bundled with Libero is available at the following location in the Libero installation folder:

• Windows:

<\$Installation\_Directory>\Microsemi\Libero\_SoC\_vX.X\Designer\bin64\pfsoc\_mss.exe

• Linux: <\$Installation\_Directory>\Microsemi\Libero\_SoC\_vX.X\bin64\pfsoc\_mss

### Note: For Windows, a start menu entry is created for easy launching.

For more details on how to install Libero, see www.microsemi.com/product-directory/design-resources/1750-liberosoc#documents

The PolarFire SoC MSS configurator can also be installed as a standalone application. For more information, see www.microsemi.com/product-directory/design-resources/1750-libero-soc#documents.

### 1.1.2 Running the PolarFire SoC MSS Configurator

The Standalone MSS Configurator can run in one of the following modes.

#### **Batch Mode**

The PolarFire SoC MSS Configurator application can be executed in the Batch mode for scripted execution as follows:

· Windows:

```
<Libero SoC or Standalone MSS Configurator installation area>\bin64\pfsoc_mss.exe
-CONFIGURATION_FILE:<absolute path for configuration file name (.cfg)> -
OUTPUT DIR:<absolute path for output directory>
```

Linux:

```
<Libero SoC or Standalone MSS Configurator installation area>/bin64/pfsoc_mss
-CONFIGURATION_FILE:<absolute path for configuration file name (.cfg)> -
OUTPUT_DIR:<absolute path for output directory>
```

#### **Interactive Mode**

The Standalone MSS Configurator (pfsoc\_mss) can be launched from the Libero SoC installation directory (specified above) or from the Windows **Start Menu**.

For more details about Batch mode and Interactive mode usage, see www.microsemi.com/product-directory/soc-design-tools/5587-pfsoc-mss-configurator-tool#documents.

# 1.2 Libero<sup>®</sup> SoC

Libero System-on-Chip (SoC) design suite offers high productivity with its comprehensive, easy to learn, easy to adopt development tools for designing with Microsemi's power efficient Flash FPGAs, SoC FPGAs, and Rad-Tolerant FPGAs. The suite integrates industry standard Synopsys Synplify Pro<sup>®</sup> synthesis and Mentor Graphics ModelSim<sup>®</sup> simulation with best-in-class constraints management, debug capabilities, and secure production programming support.

For more details, see the Libero SoC Design Flow User Guide at:

www.microsemi.com/product-directory/design-resources/1750-libero-soc#documents

### 1.3 SoftConsole

SoftConsole is an Eclipse-based IDE facilitating the development and debug of Bare Metal and RTOS-based C/C++ applications for Microchip SoC based FPGAs. It provides development and debug support for all Microchip SoC FPGAs and 32-bit soft IP CPUs.

#### Figure 1-1. SoftConsole IDE

SC workspace.examples - mpfs-blink	ky/src/application/hart0/	e51.c - SoftConsole v6.4.0.410				-	o ×	
File Edit Source Refactor Naviga	ate Search Project Ru	n Window UltraDevelop Help						
🖆 🕶 🔛 🕲 💌 🐐 🖛 🔛 💷 💌 🖻	9 ( <b>b)</b> % % %   🔪   🕨 0	■ × 3. 0. 余 号 ② ▲ 10 平 台	• 🚳 • 🔮 • 🎯 • 🛛 👳	- 0 - 9 - 😕 🛷 - 🛃	9 🔟 e 9 🖛 👌 🖛 🗘 🖛 🔿	▲ □ ○ ○ ○ ○	ኒ 😰 🔝 🗟 🏘	-66
🗞 Project Explorer 😫 👘 🗖	@ e51.c #				" 🗖 😫 Ou	tline 😫 🖲 Build Target	s = 1	
fpga-cortex-m1-blinky     m1fpga-cortex-m1-blinky     miv-rv32im-interrupt-blinky     miv-rv32ims-systick-blinky     miv-rv32imaf-raytracer-uart-c     w fmfs-blinky     w; Binaries     s Includes     w src     * @ application     * @ hart0     * @ bart0	143 raise_sc 144 } 145 146 147*void e51_app 148 { 149 safe_MSS 150 151 while (1 152 { 153 // S 154 155 cons 155 voiz 155 rons 156 voiz 157 158 for	<pre>&gt;tt_interrupt(lu); &gt;lication(void) 5_UART0_polled_tx_string("Hel: 1) Stay in the infinite loop, new st uint64_t delay_loop_max = : itile uint64_t delay_loop_sum (uint64_t i = 0; i&lt; delay_loop</pre>	lo World from e51 ( ver return from mai 10000; = 0; pp_max; i++) {	hart 0).\r\n"); N	* * * * * * * * * * * *	e gpio0_bit2_or_gpio2 gpio0_non_direct_plik gpio1_non_direct_plik gpio2_non_direct_plik gpio2_non_direct_plik est_setup(void) : void est_setup(void) : void est_setup(	라고 아이지	~
> m bart1	160 }	delay_loop_sum = delay_loop_s	sum + 1;			1		_
	161	Expression	Туре	Value				>
> > modules	162 safe	delay_loop_sum	volatile uint64_t	9999000	)			
> 💩 platform	163						>	ř
> 👄 Debug								
mpfs-blinky hw Debug.lau	Problems Tasks						0 0 1 <b>1</b> 1 - 1	-
mpfs-blinky Renode Debu	<ul> <li>Second Second Sec</li></ul>	Name i delau leon sum						^
Renode Start-	I C:\Microchip\S	Details:99990000			<u>^</u>	`		
README.html	I riscv64-unkno	Default:99990000						
README.md	PolarFire-SoC-Rer	Decimal:99990000						
mpfs-freertos-lwip	I C:\Microchip\S	Hex:0x5f5b9f0	0111110000			1		
mpfs-mustein-julia	<ul> <li>Impfs-blinky Rend</li> </ul>	0.4.3.0575334760	0111110000		Y	1		
smartfusion-cortex-m3-blinky	<ul> <li> <sup>*</sup> <sup>®</sup> mpfs-blinky.el  </li> </ul>	1<			>			
smartfusion2-cortex-m3-blink	Thread #1	machine-0.e51[0]] 1 [core: 0] (Suspended	d : Breakpoint)					
	= e51_app	lication() at e51.c:156 0x8003c20						
	= e51() at	e51.c:186 0x8003cea						
	= main_ot	ner_narti) at system_startup.c:162 0x800	2082					
		st_nartij at system_startup.c:128 0x80020	030					
< >								¥
		Writ	table Sm	art Insert 153 :	36 : 4355 Launching n	npfs-blinky Red-deb	ug: (100%) 💻	-

For the latest SoftConsole Release Notes, see the SoftConsole webpage.

### 1.3.1 SoftConsole Presets

This section provides an outline of the default configurations for building projects for PolarFire SoC in SoftConsole v6.4 and later.

Existing SoftConsole projects can be downloaded from the PolarFire SoC Bare Metal Library: github.com/polarfire-soc/polarfire-soc-bare-metal-library. To import a project, follow these steps:

- 1. Click File > Import.
- 2. Select the Existing Projects into Workspace option.

#### Notes:

- 1. The downloaded projects are pre-configured with default settings and can be used as a base to build a new project.
- The sample XML is included with the Bare Metal example projects; XML for reference designs can be found in the kit design folder on Github. For example, the PolarFire SoC Icicle Kit Libero reference design can be found here: github.com/polarfire-soc/icicle-kit-reference-design and contains an XML folder with reference XML for eMMC and SD card targets.

#### 1.3.1.1 Build Options

To view the properties of a project, right click an open project in the workspace and select Properties.

To configure build options, select **C/C++ Build** followed by **Settings**. These options can be configured globally or for individual build configurations using the **Configuration** field as shown in the following figure.

### Figure 1-2. Project Properties—Build Configurations

type filter text	Settings					\$ <b>*</b> \$	≎ <b>*</b> 8
<ul> <li>Resource Builders</li> <li>C/C++ Build Build Variables Environment</li> </ul>	Configuration:	Debug [ Active ] Debug [ Active ] Release [ All configurations ]			Manage	Configura	tions
Settings Tool Chain Edi > C/C++ General > cppcheclipse	<ul> <li>Target P</li> <li>Optimiz</li> <li>Warning</li> <li>Debugg</li> </ul>	rocessor ation gs jing	Architecture Multiply extended Atomic extended	RV64G (-march=rv64g*) ension (RVM) nsion (RVA)			~

The default build configuration for the target processor section of a Bare Metal project are shown in the following figure.

### Figure 1-3. Configuration—Target Processor

filter text	Settings			$(\neg \bullet \circ \bullet)$
filter text source ilders C++ Build Build Variables Environment Logging Settings Tool Chain Edir C++ General poheclipse CU oject Natures oject Reference: n/Debug Settir	Settings Configuration: Debug [Active] Tool Settings Toolchains Devices P E Target Processor Optimization Warnings Debugging Set GNU RISC-V Cross Assembler Preprocessor Configuration Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Settings Miscellaneous Setting Miscellaneous	Build Steps          P Build /          Architecture          Multiply extension          Multiply extension        Atomic extension          Atomic extension        Atomic extension          Floating point          Y          Code model         Small data limit         Align          Force string op-          Other target flags	Artifact Binary Parsers Erro RV64G (-march=rv64g*) ion (RVM) on (RVA) None tension (RVC) LP64 (-mabi=lp64*) Double precision (d) Toolchain default Medium Low (-mcmodel=medle 8 Toolchain default (-mtune) /epilogue (-msave-restore) erations to call library functions (-	Manage Configurations          r Parsers         ··
	<ul> <li>General</li> <li>GNU RISC-V Cross Print Size</li> <li>General</li> </ul>		Derters D	- for the
>			Restore D	efaults Apply

The following figure shows the default **Includes** for the **GNU RISC-V Cross Assembler**. **Figure 1-4. Configuration—GNU RISC-V Cross Assembler—Includes** 

Resource Builders C/C++ Build Build Variables Environment Logging Settings Tool Chain Edi C/C++ General cppcheclipse MCU Project Natures Project Reference: Run/Debug Settir	Configuration: Debug [Active]  Tool Settings Toolchains Devices PBu Target Processor Optimization Warnings Debugging  Solut RISC-V Cross Assembler Preprocessor Niscellaneous  Solut RISC-V Cross C Compiler Preprocessor Niscellaneous Debug Riscellaneous Preprocessor Niscellaneous	Wanag uild Steps Pauld Artifact Binary Parsers Fror Parsers Include paths (-1) "\${workspace_loc:/\${ProjName}/src/application}" "\${workspace_loc:/\${ProjName}/src/modules}" "\${workspace_loc:/\${ProjName}/src/platform}"	e Configuration
	<ul> <li>S GNU RISC-V Cross C Linker</li> <li>General</li> <li>Libraries</li> <li>Miscellaneous</li> <li>S GNU RISC-V Cross Create Flash Image</li> <li>General</li> <li>S GNU RISC-V Cross Create Listing</li> </ul>	Include files (-include)	<ul> <li>副 副 副 引</li> </ul>
	<ul> <li>SGNU RISC-V Cross Print Size</li> <li>General</li> </ul>		

The default **Includes** for the **GNU RISC-V Cross C Compiler** are shown in the following figure.

Figure 1-5. Configuration—GNU RISC-V Cross C Compiler—Includes

sc Properties for mp	fs-gpio-interrupt		— 🗆 X
type filter text	Settings		⇔ ♥ ⇔ ♥ §
<ul> <li>Resource Builders</li> <li>C/C++ Build Build Variables Environment Logging Settings</li> </ul>	Configuration: Debug [Active]  Tool Settings Toolchains Devices P B  Target Processor	uild Steps 😤 Build Artifact 🗟 Binary Parsers 🧕 Erro Include paths (-1)	Manage Configurations r Parsers
Tool Chain Edi C/C++ General cppcheclipse MCU Project Natures Project Reference: Run/Debug Settir	<ul> <li>Optimization</li> <li>Warnings</li> <li>Debugging</li> <li>S GNU RISC-V Cross Assembler</li> <li>Preprocessor</li> <li>Includes</li> <li>Warnings</li> <li>Miscellaneous</li> </ul>	"\${workspace_loc:/\${ProjName}/src/application}" "\${workspace_loc:/\${ProjName}/src/modules}" "\${workspace_loc:/\${ProjName}/src/platform}"	
	<ul> <li>Preprocessor</li> <li>Includes</li> <li>Optimization</li> <li>Warnings</li> <li>Miscellaneous</li> <li>SOU RISC-V Cross C Linker</li> <li>General</li> <li>Libraries</li> <li>Miscellaneous</li> </ul>	Include system paths (-isystem)	ᡚ 🔊 ଲି କି। ଜୁ।
	<ul> <li>S GNU RISC-V Cross Create Flash Image</li> <li>General</li> <li>GNU RISC-V Cross Create Listing</li> <li>General</li> <li>GNU RISC-V Cross Print Size</li> <li>General</li> </ul>	Include files (-include)	<ul> <li>記 記 장! 요!</li> </ul>
< >		Restore D	efaults Apply
0		Apply and C	lose Cancel

The default Linker script used by the **GNU RISC-V Cross C Linker** targets the LIM as show in the following figure and several sample Linker scripts to target different memory sources are included in the sample projects.

sc Properties for mpf	s-gpio-interrupt		– 🗆 X
type filter text	Settings		⇔ ▼ ⇔ ▼ ≬
<ul> <li>kype filter text</li> <li>Resource Builders</li> <li>C/C++ Build Build Variables Environment Logging Settings Tool Chain Edit</li> <li>C/C++ General</li> <li>cppcheclipse</li> <li>MCU Project Natures Project Reference: Run/Debug Settir</li> </ul>	Settings Configuration: Debug [Active] Tool Settings Toolchains Devices P Target Processor Debugging Solu RISC-V Cross Assembler Preprocessor Continues Miscellaneous Solu RISC-V Cross C Compiler Preprocessor Miscellaneous Solu RISC-V Cross C Compiler Preprocessor Miscellaneous Solu RISC-V Cross C Linker Solu RISC-V Cross C Linker Solu RISC-V Cross C Linker Miscellaneous Solu RISC-V Cross C Linker Solu RISC-V Cross C Linker Debugging Miscellaneous Solu RISC-V Cross C Linker Solu RISC-V Cross C Linker Solu RISC-V Cross Create Flash Image Solu RISC-V Cross Create Listing	Build Steps  Build Artifact Binary Parsers  Script files (-T)  \$\text{Workspace_loc:/\$(ProjName)/src/platform/ }	<ul> <li> ♦ ★ ♦ ★ § </li> <li> Manage Configurations </li> <li> Error Parsers          &lt;</li></ul>
	<ul> <li>GNU RISC-V Cross Create Listing</li> <li>General</li> <li>GNU RISC-V Cross Print Size</li> <li>General</li> </ul>	Do not use standard start files (-nostartfiles) Do not use default libraries (-nodefaultlibs) No startup or default libs (-nostdlib) Remove unused sections (-Xlinkergc-sections) Print removed sections (-Xlinkerprint-gc-go-go-go-go-go-go-go-go-go-go-go-go-go-	s) tions) sections)
< >		Rest	ore Defaults Apply
1		Apply	and Close Cancel

#### Figure 1-6. Configuration—GNU RISC-V Cross C Linker—General

To change the Linker script, double click on the existing Linker script and select the **Workspace** option to select the Linker script from the project as shown in the following figure.

Figure 1-7. Ec	lit File Path			
<b>sc</b> Edit file path				×
File:				
"\${workspace_l	oc:/\${ProjName}/s	rc/platform/config,	/linker/mpfs-lim.ld}"	
	ОК	Cancel	Workspace	File system

The default output file format selected in GNU RISC-V Cross Create Flash Image for a Bare Metal project is Intel HEX. See the following figure.

Figure 1-8	Configuration-	-GNU RISC-V Cros	ss Create Flash Im	age—General
------------	----------------	------------------	--------------------	-------------

SC Properties for mpfs-gpio-interrupt	t
---------------------------------------	---

ilter text Settings		¢ ▼ ¢ ▼
ilter text     Settings       source     Iders       iders     Configuration: Debug       Build Variables     Tool Settings Tool Settings       Environment     Tool Settings Tool Chain Edir       Copectipse     Image: Settings       U     Image: Settings       U     Image: Settings       U     Image: Settings       Image: Settings     Image: Settings </th <th>[ Active ] Ichains Devices Build Steps Build Output file forma Section: -j.text Section: -j.dat Other sections ( Sections ( Section: -j.dat Other sections ( Section: -j.dat Section: -j.dat Section</th> <th>Manage Configuration Artifact Binary Parsers Error Parsers t (-O) Intel HEX t Intel HEX Motorola S-record Motorola S-record (symbols) -j) Raw binary</th>	[ Active ] Ichains Devices Build Steps Build Output file forma Section: -j.text Section: -j.dat Other sections ( Sections ( Section: -j.dat Other sections ( Section: -j.dat Section: -j.dat Section	Manage Configuration Artifact Binary Parsers Error Parsers t (-O) Intel HEX t Intel HEX Motorola S-record Motorola S-record (symbols) -j) Raw binary
<ul> <li>Beneral</li> <li>Libraries</li> <li>Miscellaneou</li> <li>GNU RISC-V Crowing General</li> <li>GNU RISC-V Crowing General</li> <li>GNU RISC-V Crowing General</li> <li>GNU RISC-V Crowing General</li> </ul>	s iss Create Flash Image iss Create Listing iss Print Size Other flags	Restore Defaults Apply

### 1.3.1.2 Debug Configurations

### **Debug Configuration Window**

All the example Bare Metal projects contain a Renode<sup>™</sup> debug configuration and a hardware debug configuration in the **GDB OpenOCD Debugging** section. The debug configuration also contains a **Launch Group** configuration option, which can be used to launch the Renode emulation platform and start the Renode debug configuration in one step as opposed to launching them independently. The following figure shows the **Debug Configuration** window.

Figure 1-9. Debug Configuration Window

sc Debug Configurations Create, manage, and run configurations	
Image: Second	Configure launch settings from this dialog: - Press the 'New Configuration' button to create a configuration of the selected type. - Press the 'New Prototype' button to create a launch configuration prototype of the selected type. - Press the 'Suport' button to export the selected configuration. - Press the 'Duplicate' button to copy the selected configuration. - Press the 'Duplicate' button to remove the selected configuration. - Press the 'Inter' button to configure filtering options. - Edit or view an existing configuration by selecting it. - Select launch configuration(s) and then select 'Unlink Prototype' menu item to link a prototype. - Select launch configuration(s) and then select 'Unlink Prototype Values' menu item to reset with prototype values. Configure launch perspective settings from the 'Perspectives' preference page.
()	Debug Close

### Debugger Tab

The default settings in the **Debugger** tab is shown in the following figure. **Figure 1-10. Debug Configurations—Debugger** 

2 3 1 × 1 7 ×		a Salaran a la colli la como de la com			
ype filter text	Name: mprs-mmua	rt-interrupt nw all-narts debug	am a d		
GDB OpenOCD Debugging     mpfs-mmuart-interrupt hw all-harts debug	OpenOCD Setup	r Startup 🦻 Source 🖾 Common 👔	SVD Path		
Launch Group	Executable path:	{openocd_path}/{{openocd_executab	e) B	rowse	Variables.
smpfs-mmuart-interrupt renode all-harts start-platforr	Actual executable:	C:\Microchip\scWindows-6.4.0.400-20	200811-162736\eclipse\//openocd/l	bin/opend	ocd.exe
S UltraDevelop Agent	GDB port:	(to change it use the <u>global</u> or <u>workspa</u> 3333	e preferences pages or the <u>project</u> pr	roperties p	oage)
	Telnet port:	4444			
	Tcl port:	6666			
	Config options:	command "set DEVICE MPFS" file board/microsemi-riscv.cfg			
	Allocate consol	e for OpenOCD	Allocate console for the telnet con	nection	
	GDB Client Setup				
	Start GDB sessi	on			
	Executable name:	{cross_prefix}gdb{{cross_suffix}	В	rowse	Variables
	Actual executable: Other options:	riscv64-unknown-elf-gdb			
	Commands:	set \$target_riscv=1 set mem inaccessible-by-default off			
	Remote Target				
	Host name or IP a	ddress: localhost			
	Port number:	3333			
	Force thread list	update on suspend			
				<u>Res</u>	tore defa
>					
ter matched 7 of 17 items			Revert		Apply

In the **Debugger** tab, under the **OpenOCD Setup > Config options** section, the default commands used are the following:

```
--command "set DEVICE MPFS"
--file board/microsemi-riscv.cfg
```

In the **Debugger** tab, under the **GDB Client Setup > Commands** section, the default commands used are the following:

```
set $target_riscv=1
set mem inaccessible-by-default off
file ${config_name:mpfs-mmuart-interrupt}/mpfs-mmuart-interrupt.elf
```

**Note:** The file command shown in the preceding section must match the name of the project being used. The {config\_name:mpfs-mmuart-interrupt}/ section selects the folder for build files used in the configuration (that is, Debug or Release) and the mpfs-mmuart-interrupt.elf is the name of the .elf file produced on a successful build.

### Startup Tab

The default settings in the **Startup** tab is shown in the following figure. **Figure 1-11. Debug Configurations—Startup** 

8 & 10 ¥   10 7 ¥			
ma filter text	Name: mpfs-mmuart-interrupt hw all-harts debug		
GDB OpenOCD Debugging     mpfs-mmuart-interrupt hw all-harts debug     mpfs-mmuart-interrupt renode all-harts debug     Launch Group     mpfs-mmuart-interrupt renode all-harts start-platforr	Main & Debugger Startup & Source Common & SVD Path Initialization Commands Initial Reset. Type: init		
% UltraDevelop Agent	Enable ARM semihosting		
	Load Symbols and Executable Load symbols © Use project binary: mpfs-mmuart-interrupt.elf		
	Symbols offset (hex):         Load executable         Use project binary:         mpfs-mmuart-interrupt.elf	Workspace	hie System
	Use file: Executable offset (hex):	Workspace	File System
	Runtime Options		
	Run/Restart Commands           Pre-run/Restart reset         Type: (always executed at Restart)		
	thread apply all set \$pc=_start		
	Set program counter at (hex):  Set breakpoint at:  Continue		
			Restore defa
>			

In the Startup tab, under the Run/Restart Commands section, the default commands used are the following:

thread apply all set \$pc=\_start

### 1.3.2 Debugging using SoftConsole

See the SoftConsole section for information on configuring builds and setting up debug configurations using SoftConsole.

### 1.3.2.1 Launching a Debug Configuration

Follow these steps to launch the debug configuration.

1. To launch a debug session, click the drop down arrow beside the debug icon and then, click the **Debug Configurations** option.

#### Figure 1-12. Debug Configurations

sc test - mpfs-mmuart-interrupt/src/application/hart0/e51.c - SoftConsole v6.4.0.400

File Edit Source Refactor Navigate Se	arch <u>P</u> roject <u>G</u> it <u>R</u> un <u>W</u> indow <u>H</u> elp
🕼   🐔 🔻   🗶   🕪 💷 📾 🖉 🖉 . e 📲 🚺	> A A = 40 + 0 + 0 + 0 + 0 A + 0 A + 0 A + 0 + 0
陷 Project Explorer 🕴 🖻 🕵 🖓 🕴 🖱 🗖	l e51.c ∞ 5 1 PolarFire SoC UltraSoC Trace Connection
<ul> <li>mpfs-gpio-interrupt</li> <li>mpfs-i2c-master-slave</li> <li>mpfs-mmuart-interrupt</li> </ul>	1=/**     Debug As     ************************************
<ul> <li>Includes</li> <li>src</li> <li>application</li> <li>hart0</li> <li>6 e51.c</li> <li>hart1</li> <li>hart2</li> </ul>	<pre>5 * 6 * Application code running on E51 7 * 8 * Example project demonstrating the use of polled and interrupt driven 9 * transmission and reception over MMUART. Please refer README.txt in th 10 * folder of this example project 11 */ 12</pre>
<ul> <li>&gt; Is nart2</li> <li>&gt; Is hart3</li> <li>&gt; Is hart4</li> <li>&gt; Is inc</li> <li>&gt; Is boards</li> <li>&gt; Is platform</li> <li>Impfs-mmuart-interrupt hw all-hit</li> <li>Impfs-mmuart-interrupt renode a</li> </ul>	<pre>12 13 #include <stdio.h> 14 #include <string.h> 15 #include "mpfs_hal/mss_hal.h" 16 #include "drivers/mss_mmuart/mss_uart.h" 17 18e/************************************</string.h></stdio.h></pre>

2. Select the debug session to be launched and click **Debug**.

Figure 1-13. Debug Configurations

nmon & SVD Path	Search Project	Browse Browse	
Uart-interrupt.elf Variables O Disable auto buil Configure Workspar	Search Project	Browse Browse	
uart-interrupt.elf Variables O Disable auto buil <u>Configure Workspa</u>	Search Project	Browse	
Uart-interrupt.elf Variables O Disable auto buil Configure Workspar	Search Project	Browse	
Uart-interrupt.elf Variables O Disable auto buil Configure Workspar	Search Project	Browse	
Variables O Disable auto buil	Search Project	Browse	
O Disable auto bui	Variables Search Project Browse		
O Disable auto bui Configure Workspa			
O Disable auto build Configure Workspace Settings			
		Dennet	

The debug session is launched and connects to the target.

### 1.3.2.2 Perspectives

There are several perspectives to choose from, each has a different layout optimized for different tasks. Perspectives can be chosen using icons at the top-right corner of the SoftConsole window. The following figure and table show icons for different perspectives and their description.

#### Figure 1-14. Choosing Different Perspectives



The description for each number is provided in the following table.

1	View all available perspectives
2	Develop and debug perspective
3	C/C++ perspective
4	Debug perspective
5	UltraDevelop perspective

To add windows or tools to the perspective, select **Window > Show View** and choose the required window. In the following figure, the **Debug** window is chosen.

#### Figure 1-15. Adding Windows or Tools for Perspectives



### 1.3.2.3 Debugging a Hart

Each hart in the system appears as a different thread in the **Debug** window. This window is automatically shown in the **Debug** perspective and can be added to an active perspective. All the threads appear under the project name as shown in the following figure.

#### Figure 1-16. Debug Window Showing Harts as Threads

🐌 🗓 🖻 🧏 🖬 🖇 🗖 🗖 \* Debug 🛱 陷 Project Explorer Y 5 mpfs-mmuart-interrupt renode all-harts start-platform-and-debug [Launch Group] C:\Microchip\scWindows-6.4.0.400-20200811-162736\renode\bin\Renode.exe riscv64-unknown-elf-gdb PolarFire-SoC-Icicle-Renode-emulation-platform [Program] C:\Microchip\scWindows-6.4.0.400-20200811-162736\renode\bin\Renode.exe Impfs-mmuart-interrupt renode all-harts debug [GDB OpenOCD Debugging] mpfs-mmuart-interrupt.elf [cores: 0,1,2,3,4] ✓ P Thread #1 [machine-0.e51[0]] 1 [core: 0] (Suspended : Breakpoint) = e51() at e51.c:85 0x800531e main\_other\_hart() at system\_startup.c:205 0x8002eb0 ■ u54\_1() at u54\_1.c:50 0x80050ac main\_other\_hart() at system\_startup.c:211 0x8002ec8 Phread #3 [machine-0.u54\_2[2]] 3 [core: 2] (Suspended : Container) u54\_2() at u54\_2.c:50 0x8004eba main\_other\_hart() at system\_startup.c:217 0x8002ee0 Phread #4 [machine-0.u54\_3[3]] 4 [core: 3] (Suspended : Container) ≡ u54\_3() at u54\_3.c:50 0x8004cc8 main\_other\_hart() at system\_startup.c:223 0x8002ef8 Phread #5 [machine-0.u54\_4[4]] 5 [core: 4] (Suspended : Container) ≡ u54\_4() at u54\_4.c:49 0x8004ad6 main\_other\_hart() at system\_startup.c:229 0x8002f10 Riscv64-unknown-elf-gdb

Each thread in the preceding figure represents a hart, and the function listed below each thread is the code being executed.

The following figure describes how to interpret the current function that is executed on a hart. **Figure 1-17. Current Function Executing on a Hart** 



- mpfs-mmuart-interrupt renode all-harts start-platform-and-debug [Launch Group]
   C:\Microchip\scWindows-6.4.0.400-20200811-162736\renode\bin\Renode.exe
  - 📕 riscv64-unknown-elf-gdb
- PolarFire-SoC-Icicle-Renode-emulation-platform [Program]
  - C:\Microchip\scWindows-6.4.0.400-20200811-162736\renode\bin\Renode.exe
- - - ✓ Inread #1 [machine-0.e51[0]] 1 [core: 0] (Suspended : Breakpoint)



The description for each number is provided in the following table.

1	Shows the function that is currently being executed.
2	Shows the file in which function is found.
3	Shows the line of the file where the code is currently being executed
4	Memory address of the code that is executed

### 1.3.2.4 Debug Session Controls

The following figure shows the buttons that are used to control the execution of the debug session. **Figure 1-18. Buttons to Control Debug Session** 



1. Disable breakpoints 2. Resume 3. Halt 4. Stop 5. Step into 6. Step over 7. Step return 8. Instruction Stepping Mode

### 1.3.2.5 Setting Breakpoints

To add Breakpoints, right click beside the line number where the Breakpoint is required and select **Toggle Breakpoint**. Alternately, the same can be achieved by double clicking on the same location.

Figure 1-19. Toggle Breakpoint

100

	p.c 🛛 u54_1.c 🔹 mss	_uart.c
76 77°/* Main function 78 * Application 79 * MMUART0 local 80 * In the respen- 81 * are enabled. 82° void e51 (void) 83 { 84 int8_t info 85 uint64_t mc	n for the hart0(E51 code running on hart l interrupt is enabl ctive U54 harts, loc e.g. in U54_1.c loc _string[100]; ycle_start = 0U;	processor). t0 is placed here. led on hart0. cal interrupts of the corresponding MMUART cal interrupt of MMUART1 is enabled. */
86 uint64_t mc	ycle_end = 0U;	
Toggle Breakpoint	Ctrl+Shift+B	tid);
Add Breakpoint Add Dynamic Printf Disable Breakpoint Breakpoint Properties.	Ctrl+Double Click Shift+Double Click Ctrl+Double Click	Reset */ << 5u)   (1u << 6u)   << 7u)   (1u << 8u)   (1u << 9u));
Build Selected File(s) Clean Selected File(s)	,	fffff;
Go to Annotation cppcheck	Ctrl+1	lock);
Add Bookmark Add Task		MSS_UART_NO_PARITY   MSS_UART_ONE_STOP_BIT);
<ul> <li>Show Quick Diff</li> <li>Show Line Numbers</li> <li>Folding</li> </ul>	Ctrl+Shift+Q	<pre>now. Please see uart0_rx_handler() for uart0_lo, x_handler,</pre>
Preferences		T_FIFO_SINGLE_BYTE);
111 MSS_UART_ena 112 113 /* Demonstr	able_local_irq(&g_ms ating polled MMUART	<pre>ss_uart1_lo); transmission */</pre>

These breakpoints are set for all harts.

If a breakpoint is required only for a single hart and shared code is being run, the breakpoint can be filtered using the following steps:

- 1. Click the **Breakpoints** window.
- 2. Right click the breakpoint to be filtered and select **Breakpoint Properties**.

#### ९ 🛛 😰 🚼 😼 🎋 🏀 🗶 🍇 🧶 🕙 🗙 🛤 🔜 🕀 🖃 🕾 💲 😑 🗖 🗱 Variables 🔹 Breakpoints 🛛 🎋 Expressions 🗟 Peripherals 🗹 🔮 [function: e51] [type: 😔 Go to File e51.c [line: 91] ☑ Enable mss\_uart.c [line: 1346 Disable X Remove Remove All Triggers 🕷 Remove All Select All Ctrl+A Сору Ctrl+C Paste Ctrl+V Import Breakpoints... Se Export Breakpoints... Breakpoint Properties...

Select the Filter option and enable the breakpoint for the hart(s) required.
 Figure 1-21. Filter Breakpoints

Figure 1-20. Breakpoint Properties

sc Properties for C/C	++ Function Breakpoint			$\times$
type filter text	Filter		⇔ ▼ <	> <b>•</b> 8
Common Actions Filter	Restrict to Selected Processes and Threads:			
?	Apply and Clos	se	Cancel	

### 1.3.2.6 Setting Watchpoints

Watchpoints can be set on variables while running a debug session. To set a Watchpoint, open the **Variables** window, right click on the variable and select **Add Watchpoint (C/C++)...**.

### Figure 1-22. Add Watchpoint

				_	Ø	$\times$
⊧ Variables ≋ <mark>●</mark> Break	points & Expressions 🚡	Peripherals	Value	Q 2010	:œ  <b>∷</b>  C3 ⊄	8 - C
<ul> <li>info_string</li> </ul>	int8_t [100]		0x800a350			
⇔ mcycle Salact	All	Ctrl+A	0			
M mcycle	Variables	Ctrl+C	0			
⇔ delta_	valiables	Cuite	0			
we hartid     *0 Cast T     ×11 Displa     View 1	o Type ay As Array Memory	,	0			
Find	Find Ctrl+F					
🐐 Add V	Watchpoint (C/C++)					
💥 Watch	1					

This Watchpoint can be configured using the **Properties for C/C++ Watchpoint** window. **Figure 1-23. Watchpoint Properties Window** 

<b>SC</b> Properties for C/C	++ Watchpoint					$\times$
Common	Common				⇔ ▼ ⇒	• 00
	Class:	C/C++ Watchpoi	nt			
	Expression to watch:	mcycle_start				
	🗹 Range:	8				
	Read					
	✓ Write					
	✓ Enabled	r				
	Condition:					
	Ignore count:	0				
?			Apply and Close	9	Cancel	

The added Watchpoint appears in the **Breakpoints** window as an expression.

Figure 1-24. Breakpoints Window with Watchpoint

(x	<sup>)</sup> ⁼ Variables	<b>⁰•</b> Breakpoints ⊠	🛫 Expressions	🗟 Peripherals
	🗸 🔮 [fu	unction: e51] [type	: Temporary]	
	🗸 🔎 e5	1.c [line: 91]		
	🗸 🖉 ms	s_uart.c [line: 1346	5]	
	🖌 🖉 [e:	xpression: 'mcycle	_start'] [units: 8]	

### 1.3.3 Renode<sup>™</sup>

Renode is an open-source software development framework with commercial support from Antmicro that lets you develop, debug and test multi-node device systems reliably, scalably, and effectively.

For more information, see: https://renode.io/ and https://github.com/renode/renode.

### 1.4 FlashPro Express

FlashPro Express is the software tool for programming PolarFire SoC using the FlashPro Programmer hardware.

For the latest version of FlashPro Express User Guide, see www.microsemi.com/product-directory/design-resources/ 1750-libero-soc#documents.

### 1.5 RISC-V GCC Bare Metal

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project supporting various programming languages. GCC is a key component of the GNU toolchain and the standard compiler for most projects related to GNU and Linux, including the Linux kernel. The Free Software Foundation (FSF) distributes GCC under the GNU General Public License (GNU GPL). GCC has played an important role in the growth of free software, as both a tool and an example.

SoftConsole is shipped with Bare Metal riscv-gcc-toolchain with newlib and newlib.nano for 40 abi/arch multilib combinations that allow a single toolchain to target various different target architectures (see SoftConsole Release Notes for more details).

For GCC documentation, see Using the GNU Compiler Collection.

For GCC RISC-V specific options, see **RISC-V Options.** 

For more details about march, mabi, and mtune arguments, see www.sifive.com/blog/all-aboard-part-1-compiler-args and The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2.

### 1.6 RISC-V Linux Toolchain

The RISC-V GNU Linux-ELF/glibc toolchain is used by the Linux build tools (Yocto and Buildroot) to build a Linux image.

The source used for the toolchain is available on GitHub at the following location: github.com/riscv/riscv-gnu-toolchain.

### 1.7 Yocto

Yocto is an opensource development build environment for Linux. It can be used to customize a Linux image for embedded or IoT application deployment. The Yocto framework is modular in nature and designed as a software stack with layers managing different tasks and functions. The BSP layer provides machine configurations. The

distro layer includes the top level polices for a distribution. The OpenEmbedded Build system (referred to as "the build system") is the build system used by Yocto based on "Poky". BitBake is used by the build system for image generation.

The Microchip Yocto BSP can be found at the following location: github.com/polarfire-soc/meta-polarfire-soc-yoctobsp. It contains predefined recipes to build different PolarFire SoC targets and build images with different bundles of tools. For a full list of available builds, see the readme or the console output after successfully configuring a build.



**Important:** Microchip peripheral drivers are currently added as patches with the Microchip PCIe driver being up streamed to the Linux kernel version 5.8, it is planned to upstream remaining drivers for other system peripherals in later releases.

Yocto builds supported on Linux and Yocto are not currently supported by the Windows Subsystem for Linux.

More information about Yocto is available on the Yocto project website.

#### **Related Links**

https://www.yoctoproject.org/ https://www.yoctoproject.org/docs/?section=reference-manuals

### 1.8 Buildroot

Buildroot is a tool that simplifies and automates the process of building a complete Linux system for an embedded system, using cross-compilation. It is able to generate a cross-compilation toolchain, a root filesystem, a Linux kernel image, and a bootloader for the target.

The Microchip PolarFire SoC Buildroot SDK is available at the following location: github.com/polarfire-soc/polarfiresoc-buildroot-sdk. It contains configured builds for different PolarFire SoC targets. For a full list of available builds, see the readme.



**Important:** Microchip peripheral drivers are currently added as patches with the Microchip PCIe driver being up streamed as to the Linux kernel version 5.8, it is planned to upstream remaining drivers for other system peripherals in later releases.

Buildroot is supported on Linux and supported by the Windows Subsystem for Linux.

More information about Buildroot is available on the Buildroot website.

#### **Related Links**

https://buildroot.org/ https://buildroot.org/downloads/manual/ manual.html#:~:text=Buildroot%20is%20a%20tool%20that,a%20bootloader%20for%20your%20target

### 1.9 SmartDebug

SmartDebug is a tool that enables verification and troubleshooting at the hardware level. It provides access to sNVM, SRAM, transceiver, uPROM, and fabric probe capabilities.

SmartDebug accesses the built-in probe points through the Active Probe and Live Probe features that enable designers to check the state of inputs and outputs in real-time without modification of the design.

SmartDebug can be run in the following modes:

- · Integrated mode from the Libero Design Flow
- · Standalone mode

• Demo mode (without target hardware connected)

For the latest version of SmartDebug User Guides, see www.microsemi.com/product-directory/design-resources/ 1750-libero-soc#documents.

### 1.10 Identify

Identify is a tool to find and correct functional design bugs by probing internal signals of the design directly from the FPGA at the system speed.

For more information, see Identify ME webpage.

## 2. Software Stack

RISC-V is a large ecosystem with a variety of compilers, software libraries, examples, and tools available for application development. This section outlines the open source RISC-V libraries available, along with Microchip applications and examples to aid development for PolarFire SoC.

### 2.1 **RISC-V** Libraries

Standard libraries provide generic type definitions, functions, and macros for tasks that will be undertaken on a system. This section provides information on the Newlib C standard library implementation and the GNU Binary Utilities toolset bundled with SoftConsole.

### 2.1.1 Newlib

Newlib is a C standard library implementation intended for use on embedded systems. It is a conglomeration of several library parts, all under free software licenses that make them easily usable for embedded products.

SoftConsole's RISC-V GCC toolchain and its multilibs come with pre-compiled and ready-to-use "newlib" and "newlibnano" C library.

Microchip's toolchains come with generic basic Newlib implementation.

More information about Newlib and FAQ is available on the Sourceware website.

#### **Related Links**

https://www.sourceware.org/newlib/faq.html https://www.sourceware.org/newlib/docs.html

### 2.1.2 Binutils

The GNU Binary Utilities, or binutils, are a set of programming tools for creating and managing binary programs, object files, libraries, profile data, and assembly source code.

SoftConsole comes bundled with a ready-to-use Bare Metal binutils. The SoftConsole Release Notes show some use cases of how to use nm and objcopy; however, most of the tools are used and invoked automatically by the IDE. These tools are generic and can be used to target the 32-bit Mi-V RISC-V cores and the 64-bit PolarFire SoC targets.

Tool Name	What it does	Documentation
as	Assembler	sourceware.org/binutils/docs/as/
ld	Linker	sourceware.org/binutils/docs/ld/
gprof	Profiler	sourceware.org/binutils/docs/gprof/
addr2line	Convert address to file and line	sourceware.org/binutils/docs/binutils/addr2line.html
ar	Create, modify, and extract from archives	sourceware.org/binutils/docs/binutils/ar.html
c++filt	Demangling filter for C++ symbols	sourceware.org/binutils/docs/binutils/ c_002b_002bfilt.html
nm	List symbols in object files	sourceware.org/binutils/docs/binutils/nm.html
objcopy	Copy object files, possibly making changes	sourceware.org/binutils/docs/binutils/objcopy.html
objdump	Dump information about object files	sourceware.org/binutils/docs/binutils/objdump.html
ranlib	Generate indices for archives (for compatibility, same as ar -s)	sourceware.org/binutils/docs/binutils/ranlib.html
readelf	Display content of ELF files	sourceware.org/binutils/docs/binutils/readelf.html

continued						
Tool Name	What it does	Documentation				
size	List total and section sizes	sourceware.org/binutils/docs/binutils/size.html				
strings	List printable strings	sourceware.org/binutils/docs/binutils/strings.html				
strip	Remove symbols from an object file	sourceware.org/binutils/docs/binutils/strip.html				

### 2.2 Hart Software Services (HSS)

Hart Software Services, commonly referred to as "HSS", is a collection of services that run on the E51 monitor core. HSS is used for the following:

- Program memory using USB mass storage or YMODEM transfer.
- Copy a program (Linux or Bare Metal) from a non-volatile storage (for example, eMMC or SD card) to the LIM or DDR.
- Create a payload containing multiple applications to be booted and run.
- Pass messages between cores in the MSS.

#### Operation

The HSS uses Bare Metal drivers to initialize the system, which are found in the PolarFire SoC Bare Metal Library. It also relies on XML generated by the PolarFire SoC MSS Configurator to configure the system on boot.

The HSS comprises of the following:

- A superloop monitor running on the E51 processor, which receives requests from the individual U54 application processors to perform certain services on their behalf.
- A Machine-Mode software interrupt trap handler, which allows the E51 to send messages to the U54s, and requests them to perform certain functions for it related to rebooting the U54.

#### HSS as a ZSBL

The HSS can function as a Zero Stage Boot Loader (ZSBL) to boot Linux. In this case, the HSS loads U-Boot acting as a ZSBL with U-Boot subsequently loading an OS. U-Boot is a First Stage Boot Loader (FSBL) and a Second Stage Boot Loader (SSBL).

#### HSS as a FSBL

The HSS can be used to boot Linux directly like the Berkeley Boot Loader (BBL) acting as an FSBL and SSBL.

#### Licenses

This software is released under an MIT license. It also uses other open source tools. RISC-V OpenSBI is released under a BSD-2-Clause and FastLZ compression is released under an MIT license. More information on licensing can be found at: github.com/polarfire-soc/hart-software-services/blob/master/LICENSE.md.

#### Building

The HSS can be built as a standalone image. The source is published on GitHub and build instructions for different targets can be found in its readme: github.com/polarfire-soc/hart-software-services.

#### Releases

The HSS GitHub repository is the most up-to-date location to retrieve the source files and build instructions for the HSS.

## 2.3 Bare Metal Library

The PolarFire SoC Bare Metal Library contains the most recent version of the PolarFire SoC HAL source code with a pre-populated platform folder for a PolarFire SoC Bare Metal project with all drivers. It also contains Bare Metal examples for each driver available for PolarFire SoC. These examples show how to use different functions available in the drivers and how they are configured for PolarFire SoC.

To use the Bare Metal Library examples, follow the instructions in the <code>polarfire-soc-bare-metal-library/examples</code> readme.md file available at: github.com/polarfire-soc/polarfire-soc-bare-metal-library/tree/master/examples

To use the pre-populated platform folder in a Bare Metal project, follow these steps:

- 1. Download the Bare Metal Library repository and extract it.
- 2. Delete the existing platform folder in the project (back up any changes such as, Linker script updates). Figure 2-1. Removing Platform Folder

## SC workspace.examples - SoftConsole v6.4.0.410

Filo	Edit Sou	.c.nu	Refactor Navigato S	loard	h D	roject Rup Wi	ndow UltraDovolop
riie		urce		m III	F CTE	lo lo m	
					- 8		- 200 - 200
P	roject Expl	orer					
	fpga-co	rtex-	m1-blinky				
	hart-sof	twar	e-services				
	mitpga	-con	tex-m1-blinky				
	miv-rv3	21m- 2:	interrupt-blinky				
	miv-rv3	2im- 2:	systick-blinky				
	miv-rvs/	2ima 2ima	fractional and				
1	miv-rvs.	inla	ir-raytracer-uart-cpp				
1	mpis-bi	inky	x-huip				
~ 1	s mpfs-m	mua	ort-interrunt				
		dos	it-interrupt				
		acs	New			,	
	> 🕞 ai		Golato				
	> 😑 b		do into				-
	> 🕞 p		Open in New Window				
	🖹 mpf:		Show In			Alt+Shift+W	Console 🛛 🗆 Pro
	🖹 mpf:		Show in Local Termina			2	> at this time.
	🗎 mpf:		Сору			Ctrl+C	
;	🕨 🖹 REAI	ß	Paste			Ctrl+V	
1	mpfs-m	×	Delete	Dele	ete	Delete	
1	🔋 smartfu		Source	_	_	3	>
1	🔋 smartfu		Move				
			Rename			F2	
		2	Import				
			Export				
		٢	Robot Framework			;	>
			Build Project			Alt+B	

3. Copy the platform folder extracted from the Bare Metal Library repository into the SoftConsole project and re-implement any changes that were made.

### Figure 2-2. Adding Updated Platform Folder



The only folder that might be modified by the user is the platform/config folder. The drivers, hal, and mpfs\_hal do not need user modification.

mpfs\_hal contains the part of the HAL specific to PolarFire SoC. It contains startup code, MSS register descriptions, and performs DDR training. The content of this folder is not intended to be modified. It also contains the code for interrupt and exception handling, and hardware access methods.

### 2.4 Linker Scripts

The main purpose of the Linker script is to describe how the sections in the input files must be mapped into the output file, and to control the memory layout of the output file.

Each SoftConsole project comes with at least one Linker script. Specify the memory location where the application will be deployed. The sample Linker scripts provided include:

- mpfs-ddr-e51
- mpfs-dtim
- mpfs-envm
- mpfs-lim
- mpfs-lim-lma-scratchpad-vma

To switch between them, open Settings and navigate to the Script files:

Project's Properties > C/C++ Build > Settings > Tool Settings > GNU RISC-V Cross C Linker > General > Script files

#### Figure 2-3. Tool Settings 😎 Properties for mpfs-blinky type filter text ← → ⇒ % Settings > Resource Builders Configuration: Debug [Active] Manage Configurations... V C/C++ Build **Build Variables** Environment 🛞 Tool Settings 🛞 Toolchains 🔳 Devices 🎤 Build Steps 😤 Build Artifact 📓 Binary Parsers 😣 Error Parsers Logging Settings 🖄 Target Processor **Tool Chain Editor** Script files (-T) 🗿 🌒 🚳 🖗 🚱 🖄 Optimization > C/C++ General ker/mpfs-lim.ld}" "\$workspace\_loc:/\$ProjName}/src/platfo 🖄 Warnings > cppcheclipse Debugging > MCU **Project Natures** GNU RISC-V Cross Assembler Preprocessor **Project References** Run/Debug Settings 🖄 Includes 🖄 Warnings 🖄 Miscellaneous V 🛞 GNU RISC-V Cross C Compiler Preprocessor Includes 🖄 Optimization 🖄 Warnings Miscellaneous ✓ SGNU RISC-V Cross C Linker 🖄 General 🖄 Libraries Miscellaneous 🗸 🥘 GNU RISC-V Cross Create Flash Image 🖄 General GNU RISC-V Cross Create Listing 🖄 General 🗸 🥘 GNU RISC-V Cross Print Size Do not use standard start files (-nostartfiles) 🖄 General Do not use default libraries (-nodefaultlibs) No startup or default libs (-nostdlib) Remove unused sections (-Xlinker --gc-sections) Print removed sections (-Xlinker --print-gc-sections) Omit all symbol information (-s) **Restore Defaults** Apply ? Apply and Close Cancel

Microchip recommends users to use the supplied Linker scripts (located in the src/platform/config/linker folder) and use these as a base script for their custom Linker scripts (when the supplied Linker scripts are not sufficient).

For the Linker script manual, see: sourceware.org/binutils/docs/ld/Scripts.html

### **Related Links**

https://www.sifive.com/blog/all-aboard-part-2-relocations https://www.sifive.com/blog/all-aboard-part-3-linker-relaxation-in-riscv-toolchain

### 2.5 Linux Images

The current Linux kernel version (at the time of publishing this document) is 5.6.16. The next update is for moving to Linux kernel version 5.8. Beyond version 5.8, it is planned to use the latest long-term support kernel in all builds. Information on long-term support for kernels is available on: www.kernel.org/.

In Yocto, the kernel version used in the build is specified in the recipes-kernel/linux/\*.bb file as shown in the following figure.

#### Figure 2-4. Example of Yocto Linux Kernel Version

2 master - meta-polarfire-soc-yocto-bsp / recipes-kernel / linux / mpfs-linux_5.%.bb					Got	o file	JĿ
C Ihanlyu Sorting out patches for the Aloevera and Aloevera lite.			Latest comn	nit 5097cab	on Jul	1 3	) Histor
Ril contributor							
74 lines (64 sloc) 2.94 KB	Raw	Blame	Copy File	2.94 KB	φ	Ð	0 t
1 require recipes-kernel/linux/mpfs-linux-common.inc							
LINUX_VERSION ?= "5.6.x"							
<pre>4 KERNEL_VERSION_SANITY_SKIP="1"</pre>							
BRANCH = "linux-5.6.y"							
SRCREV = VS.6.16							
= ont_uni = 1 oit://oit.kornol.org/nuh/scm/linux/kornol/oit/stable/linux.oit:hranch=\$/BRANCH1 \							
Presti Presses nerros Pi hant ren transforment Pres 200 rel Transforment de la constante de la							

For Buildroot, the specific commit of the kernel used can be seen in the top-level folder specified as **linux @ [ commit number ]** as shown in the following figure.

#### Figure 2-5. Example of Buildroot Linux Kernel Version

r	master 👻 양 2 branche	s 🛇 2 tags	Go to file	Add file	••	🛓 Code 🗸
	ConchuOD doc: readme re	phrasing	6514a11	14 days a	go ;	257 commits
	bsp/env/freedom-u500	adding openocd.cfg file for freedom unleashed board.				2 years ago
	buildroot @ 497e3df	icicle: add initial support				last month
	conf	hss: use xml flow for emmc design				28 days ago
Đ	fsbl @ 54bfc90	fsbl: Build FSBL not U-Boot M-Mode		103 Bytes	Ŷ	6 months ago
Đ	hardware-config-genera	icicle: bump hwcfg gen script		1.23 KB	ф	19 days ago
Đ	hart-software-services	icicle: add sd card support		61 Bytes	φ	last month
٥	linux @ 960a4cc	icicle: add initial support		21.37 KB	Ŷ	last month
63	opensbi @ ac5e821	opensbi: bump to v0.6		10.87 KB	P	6 months ago

### 2.6 FreeRTOS<sup>™</sup>

FreeRTOS is a real-time operating system kernel for embedded devices that has been ported to many microcontroller platforms. It is distributed under the MIT license.

SoftConsole is shipped with a bundled FreeRTOS example, see the mpfs-freertos-lwip example. The example can target the bundled Renode emulation and users can access the webserver running on it (See the *mpfs-freertos-lwip*'s readme).

The following figure shows a webserver running on top of FreeRTOS and lwip.



The latest version of FreeRTOS targeting PolarFire SoC hardware is available from the Bare Metal Library at this location: github.com/polarfire-soc/polarfire-soc-bare-metal-library/tree/master/examples/mss-ethernet-mac

More information is available on the FreeRTOS website.

#### **Related Links**

https://www.freertos.org/a00104.html#getting-started https://www.freertos.org/Documentation/RTOS\_book.html

### 2.7 Third Party Tools

The following third party tool is used for emulating RISC-V subsystem:

Renode

# 3. Application Development

PolarFire SoC supports Bare Metal, Linux, and RTOS. This section describes the device boot process, boot modes, and development flow to build user applications for these types of embedded systems. For example, the following can be executed on the application cores.

- Bare Metal applications
- Linux user applications
- RTOS
- Combination of the above (AMP)

Bare Metal applications for PolarFire SoC devices can be developed using SoftConsole. The PolarFire SoC Bare Metal firmware drivers and source files for Linux user application development are available on the PolarFire SoC GitHub.

PolarFire SoC MSS comprises of one E51 monitor core and four U54 application cores. The E51 core executes the Hart System Services (HSS), which configures the MSS and responds to runtime events. The U54 cores execute any of the following:

- Bare Metal user applications
- Operating Systems

### 3.1 Device Boot and Configuration Process

The boot-up sequence starts when the PolarFire SoC FPGA is powered-up or the device is reset. It ends when the processor is ready to execute user applications. The booting sequence runs through several stages before it begins execution of user application code. A set of operations are performed during the boot-up process that includes Power-on Reset of the hardware, peripheral initialization, memory initialization, and loading a user-defined application from non-volatile memory to volatile memory for execution.

The following figure shows the different phases of the boot-up sequence.

Figure 3-1. Boot-Up Sequence



For more information about the booting process, see UG0890: PolarFire SoC FPGA Power-Up and Resets User Guide.

PolarFire SoC MSS supports the following boot modes:

- Boot Mode 0—Wait for Interrupt Mode/Idle Boot Mode (Used for debugging)
- Boot Mode 1—Non-Secure User Boot Mode (Direct boot from eNVM)
- Boot Mode 2—Secure User Boot Mode
- Boot Mode 3 Factory-Secure Boot: MSS boots using this protocol



Important: Boot Mode 2 and Boot Mode 3 are currently not supported.

These boot modes are performed by the E51 core and are configured using SoftConsole.

### 3.2 Boot Mode 0-Idle Boot

Boot Mode 0 is set by default. If the MSS is not configured (for example, blank device), it executes a fixed, pre-configured boot ROM program, which holds all the processors in an infinite loop until a debugger connects to the target. The boot vector registers maintain their value until a new Boot mode configuration is programmed. This mode is typically used for initial software development and debug phase where the application code is loaded using the debugging through SoftConsole.

### 3.3 Boot Mode 1-Direct Boot from eNVM

In this mode, the MSS executes from a specified eNVM address without authentication. It is the fastest boot option, but there is no authentication of the code image. Boot Mode 1 includes the following steps:

- 1. The user application image needs to be programmed into eNVM using SoftConsole and the Boot mode is set.
- 2. If the eNVM content is a boot loader, it fetches the final user application from non-volatile storage and loads it to the desired memory location specific by the application, the harts then execute the application.

Boot vector addresses for all five processors are absolute addresses in eNVM.

### 3.3.1 Programming the eNVM

Launch SoftConsole and create an application project. Ensure that the SoftConsole application project contains latest **mpfs\_hal** and firmware drivers from GitHub. For illustration purposes, **mpfs\_blinky** is used as an example application project. To configure the project's build tool settings, follow these steps:

1. Right click mpfs\_blinky and select Properties as shown in the following figure.



### Figure 3-2. Properties for mpfs-blinky

The **Properties** window appears.

- 2. Expand C/C++ Build, and select Settings.
- 3. Set the **Configuration** to **Release**.
- 4. Expand **GNU RISC-V Cross C Linker**, select **General** and perform the following actions to select the appropriate Linker script:
  - 4.1. Click **Add...**.
  - 4.2. Select **Workspace** on the **Add File path** window.
- 5. In the File Selection tab, expand the mpfs-blinky and browse to: mpfs-blinky > src > platform >Config > linker > examples > mpfs-envm.ld file. Then, click OK.

The Linker is a script file which provides the information about the memory from where the code must be executed and how that memory must be used for heap and stack.

For Release mode, the Linker script mpfs-envm.ld is selected to build the application that executes the code from eNVM with the stack and heap in LIM. Other Linker script files are also available to execute the code out of LIM (mpfs-lim.ld), DDR memory (mpfs-ddr-e51.ld), and Data Tightly Integrated Memory (mpfs-dtim.ld).

#### Figure 3-3. mpfs-blinky Release Window



- 6. Click Apply and Close.
- 7. Select **Project > Clean** to clean the project.

Now, the project build settings are completed and ready for building.

8. Select **Project > Build All**.

The project is built successfully as shown in the following figure.

#### Figure 3-4. Project Successful Message



9. The mpfs-blinky.elf file is generated in the Release folder. This ELF file is programmed to the eNVM storage using SoftConsole so that at device power-up the MSS executes the application from eNVM.

To program the Boot mode settings and eNVM using SoftConsole, select the desired project and click **Run > External Tools > PolarFire SoC program non-secure boot mode 1** as shown in the following figure. **Note:** This step requires Libero SoC or the Program/Debug tool to be installed on the host PC. It also requires pre-configuration in SoftConsole. For information on pre-configuration and installation, see the readme.txt file in (SoftConsole install)/extras/mpfs.

#### Figure 3-5. PolarFire SoC Boot Mode 1

File Edit Source Refactor Navigate Search Project Git	Run Window Help		
ରା ବି •ି t t t t t t t t t t t t t t t t t t	Instruction Stepping Mode           ≫ Move to Line (C/C++)           ™ Resume at Line (C/C++)           ™ Resume           ™ Suppend           ■ Terminate           ≫ Step Into           ™ Step Over           ≪ Step Return           ≪ Step Return           ≪ Step Return		
	<ul><li>Run Last Launched</li><li>Debug Last Launched</li></ul>	Ctrl+F11 F11	
	Run History Run As Run Configurations	>	
	Debug History Debug As Debug Configurations	>	
	Breakpoint Types	>	
P Debug 22 ® Breakpoints ーロ 日後 卧 日 14 第 No consoles to c	<ul> <li>Toggle Breakpoint</li> <li>Toggle Line Breakpoint</li> <li>Toggle Watchpoint</li> <li>Toggle Method Breakpoint</li> <li>Skip All Breakpoints</li> <li>Remove All Breakpoints</li> </ul>	Ctrl+Shift+B Ctrl+Alt+B	bugger Console
	External Tools	>	1 PolarFire SoC program idle boot mode 0
			Q         2 Polarfire SoC program non-secure boot mode 1           3         Mi-V-Renode-emulation-platform           4         Polarfire-SoC-cicle-Renode-emulation-platform           5         Polarfire-SoC-Renode-emulation-platform

This sets the Boot mode to 1 and programs the application to eNVM as shown in the following figure. After powercycling the board, the application gets executed from eNVM.

#### Figure 3-6. Programming the Application



### 3.4 Clock Management

The clock configuration made in the **Clocks** tab of the PolarFire SoC MSS Configurator is stored in the XML file. The SoftConsole build process converts the XML file into hardware configuration files that include the clock configuration as shown in the following figure. If any changes to the clock configuration are required, these changes need to be made in the PolarFire SoC MSS Configurator and the updated XML file needs to be imported into the SoftConsole project. The system start-up code uses these configuration files to configure the PLLs and clocking related system registers.

### Figure 3-7. Clocks Folder



### 3.5 Physical Memory Protection (PMP)

### 3.5.1 Using the PMPs in Bare Metal

To support secure execution of application code, it is required to limit the physical addresses accessible by the software running on a Hardware Thread (Hart). The access to physical addresses can be restricted using the PMP unit in each hart. The PMP defines a finite number of regions that can be individually configured by setting the PMP registers using the user application code. It is applied on harts to allow physical memory access privileges (read, write, and execute). A PMP is configured to achieve the following:

- Security of the system is improved as there is no possibility of code injection attacks. The memory of one hart is not accessible to other harts.
- If there is any overflow in the stack usage, PMP detects it.
- It is less expensive to get safety certification for the product.

### 3.5.2 Using the PMPs in Linux

When booting Linux using HSS, the PMPs are automatically configured by the HSS for the system configuration based on the payload created.



**Important:** Configuring the PMPs using Libero is currently not supported and will be available in the next release.

### 3.6 Generating Boot Images

This section outlines the software configuration for Bare Metal project(s) that targets different harts for user applications. The steps to configure a Bare Metal application as the sole application to run on the system are provided along with how to configure multiple independent Bare Metal applications. This section also outlines the steps required to configure an application to be executed directly from eNVM and how to configure an application to be stored in the external memory and load it using the HSS. The process of programming the eNVM is also introduced.

### 3.6.1 Targeting Harts

The MPFS HAL (PolarFire SoC HAL) can be used to target multiple or single cores in a Bare Metal application. In the Bare Metal application in **src > platform > config > software > mpfs\_hal the mss\_sw\_config.h** file can be used to target harts.

Project Explorer 🕮	🖻 😘 🖓 🕴 🗖 🖻 mss_sw_config.h 🖾
<ul> <li>Sompfs-gpio-interrupt</li> <li>Sompfs-gpio-interrupt</li> <li>Sompfs_hal</li> <li>Sompfs_hal</li> <li>Sompfs_hal</li> </ul>	<pre>34</pre>
> 🖻 mss_sw_config.h > 📄 readme.txt	49 #enalt 50 51⊕ /*

#### Figure 3-8. Targeting Harts

The MPFS\_HAL\_FIRST\_HART define selects the hart that will boot up and configure the system. The MPFS\_HAL\_LAST\_HART define selects the hart that will be the last to start up. In the preceding image, the e51 is the hart selected to start up first and will be used to wake all of the U54 harts from WFI (Wait For Interrupt) mode.

If the system was configured as shown in the following figure, the e51 is still the main hart but only the U54\_1 and U54\_2 are taken out of the WFI mode (the e51 is hart 0).

#### Figure 3-9. MPFS\_HAL\_LAST\_HART Configuration



If the system is configured as shown in the following figure, the e51 is held in WFI mode and the U54\_1 becomes the main hart to wake the other U54 harts from WFI mode, in this case only the U54 2 is taken out of the WFI mode.

#### Figure 3-10. MPFS\_HAL\_FIRST\_HART and MPFS\_HAL\_LAST\_HART Configuration

Project Explorer 🔤	El state st
<ul> <li>✓ Includes</li> </ul>	<pre>34 PEFFS_HAL_FIRST_HART on a value other than 0 if you do not want your code</pre>
<ul> <li>✓ is src</li> <li>&gt; is application</li> <li>&gt; ≥ modules</li> <li>✓ is platform</li> <li>✓ is config</li> </ul>	<ul> <li>37 * to start and execute code on the E51 hart.</li> <li>38 * Set MPFS_HAL_LAST_HART to a value smaller than 4 if you do not wish to use</li> <li>39 * all U54 harts.</li> <li>40 * Harts that are not started will remain in an infinite WFI loop unless used</li> <li>41 * through some other method</li> <li>42 */</li> </ul>
> e> hardware > e> linker > e> software	43= #ifndef MPFS_HAL_FIRST_HART 44 #define MPFS_HAL_FIRST_HART 1 45 #endif 46
<ul> <li>&gt; iso drivers</li> <li>&gt; iso mpfs_hal</li> <li>&gt; iso mss_sw_config.h</li> <li>&gt; iso readme.txt</li> </ul>	47=#ifndef MPFS_HAL_LAST_HART 48 #define MPFS_HAL_LAST_HART 2 49 #endif 50 51=/*-

The projects must be built targeting the memory they will be executed from (for example, DDR) and not in the location they will be stored in (for example, eMMC). When built in SoftConsole, the resulting files created using the HSS tools are called a payload. The HSS readme contains information on the steps for creating a payload, and the tools required are stored in tools folder of the repository. This payload can then be programmed to the intended memory (for example, SD card or eMMC) using the HSS. The HSS then boots on the e51 and unpacks the payload into the relevant memory locations they are targeted for and wakes the harts they will run on.

#### 3.6.2 Storing a Single Bare Metal Application in an eNVM

If a single Bare Metal application targeting one or more harts is to be used and no other application is running on the system, the application can be stored directly in the eNVM, provided it is less than 128 kB, and executed from memory. The e51 can be used as the main hart of the system (MPFS\_HAL\_FIRST\_HART 0) and wakes any of the required U54 harts from WFI.

### 3.6.3 Storing Bare Metal Application(s) to an External Memory

If a single Bare Metal application or multiple independent applications are being used in the system and cannot be stored in the eNVM (for example, they are greater than 128 kB), then the HSS must be used to program the external non-volatile storage with the applications and copy their code to the relevant memory location on boot.

The HSS must be programmed into the eNVM to be executed on boot, it uses a small portion of the LIM for stack and heap when running on the E51. The Bare Metal application(s) can then be developed—they must not target the e51 as this will be running the HSS and must not overlap with the LIM memory locations used by the HSS. The area of LIM used by the HSS can be identified using the .map file generated when the HSS is built.

### 3.6.3.1 Single Bare Metal Application

If a single Bare Metal application is created, the first hart must target one of the U54s (for example, MPFS\_HAL\_FIRST\_HART 1) that wakes the other U54 harts in the system. The project must be built in the memory it will be executed from (for example, DDR) and not in the location it will be stored in (for example, eMMC). When built in SoftConsole, the resulting files created using the HSS tools are called payload. The HSS readme contains information on steps for creating a payload, and the tools required are stored in tools folder of the repository. This payload can then be programmed to the memory intended to be used for storing the payload (for example, SD card or eMMC) using the HSS. The HSS then boots on the e51 and unpacks the payload into the relevant memory location it is targeted for and wakes the harts it will run on.

### 3.6.3.2 Multiple Bare Metal Applications

If multiple Bare Metal applications are created, the first hart for each project must target separate U54s and the last hart in the project (that is, the final U54 this project will run on) must not overlap with the first hart of a subsequent project.

See the following example:

Project 1:

MPFS\_HAL\_FIRST\_HART 1

MPFS\_HAL\_LAST\_HART 2

This project runs on U54\_1 and U54\_2.

Project 2:

MPFS\_HAL\_FIRST\_HART 3

MPFS\_HAL\_LAST\_HART 4

This project runs on U54\_3 and U54\_4.

For more information, see the 3.6.1 Targeting Harts section.

### 3.6.4 Programming the eNVM

SoftConsole is capable of programming the eNVM and setting the boot modes for the harts used in the system. If a single Bare Metal project is used, it can be programmed directly into the eNVM. If multiple projects are used (that is, as a payload), the HSS must be programmed into the eNVM. This programming is achieved using an external tool configuration provided with SoftConsole.

#### Figure 3-11. External Tools Setting Boot Modes



To use this tool, you must have the project to be programmed selected in the project explorer and the build configuration is used to locate programming files. This implies, if a project is built in a Debug configuration, SoftConsole programs it using the files from the Debug folder in the project, whereas if the project is built in a Release configuration, SoftConsole uses the files from the Release folder.

The external tool generates a bitstream containing only the eNVM programming files (that is, it does not overwrite FPGA programming) and sets the Boot mode accordingly.

### 3.6.5 Unused Harts

If harts are not going to be used (for example, only U54\_1 and U54\_2 are used), no main function(s) (for example, u54\_3.c) need to be provided for unused harts.

#### Figure 3-12. Unused Harts



The mpfs\_hal contains weakly linked functions that are used in place of strongly linked functions if no main function is found, which implies, if no u54\_3() function is found in the project, the function shown in the following figure "system\_startup.c" file is used.

The mpfs\_hal contains weakly linked functions for all main functions available for each hart. A weakly linked function is used as a fallback or default function if a strongly linked function is not defined. By default, all functions are strongly linked without any modification and if present overrides the weakly linked functions. The mpfs\_hal weakly linked functions can be found in platform/mpfs\_hal/system\_startup.c.

A function is defined as weakly linked by adding the following attribute to its definition: \_\_attribute\_\_((weak)).

Consider the following examples for U54\_1:

- Weakly linked function: attribute ((weak)) void u54 1(void)
- Strongly linked function: void u54\_1 (void)



**Attention:** If a strongly linked function and a weakly linked function are defined, only the strongly linked function will be included in the build. If two strongly linked functions are defined with the same name a symbol link error occurs at build time.

#### Figure 3-13. system\_startup.c File Showing Weakly Linked Functions

A Project Explorer 12	8881.0	■ system	_startup.c <sup>33</sup>
> 🗁 modules	^	291	{
Y 😕 platform		292	<pre>volatile static uint64_t counter = 0U;</pre>
> 🗁 config		293	/# Addid come and an debinance bases if in these dates and in #/
> 👄 drivers		294	/* Added some code as debugger hangs it in loop doing nothing */
> 🖻 hal		296	}
Y in mpfs hal		297 }	,
Atomic h		298	
> D bitch		2998/*	
B encoding h		300	' Third U54. I Ta shaara of an application function of this range with strong linkage this
<ul> <li>Reantry S</li> </ul>		302	function will get linked
> in enuy.s		303	This default implementation is for illustration purpose only. If you need to
<ul> <li>Minimum distribution</li> </ul>		304	modify this function, create your own one in an application directory space.
> B mss_clint.h		305 *	/
> E mss_coreplex.h		306*	_attribute((weak)) void u54_3(void)
> @ mss_h2t.c		307	stated a backlet and and backlets
> 🖻 mss_h2f.h		308	uint64_t hartid = read_csr(mhartid);
> 🖻 mss_hal.h		3100	/*Clear pending software interrupt in case there was any.
> in mss_hart_ints.h		311	Enable only the software interrupt so that other core can bring this core
> 🗟 mss_ints.h		312	out of WFI by raising a software interrupt.
> iii mss_mpu.c		313	Note that any other interrupt can also be used to bring CPU out of WFI*/
> 🗟 mss_mpu.h		314	<pre>clear_soft_interrupt();</pre>
> I mss_mutex.S		315	<pre>set_csr(mie, MIP_MSIP);</pre>
> is mss_peripheral_base_add.h		317	/*out this hart into WFI.*/
> is mss_plic.h		318	do
> mss_prci.h		319	{
> 🗟 mss seg.h		320	asm("wfi");
> @ mss stubs.c		321	}while(0 == (read_csr(mip) & MIP_MSIP));
> 🖻 mss sysreq.h		322	/*The hant is out of WET, clean the SW internunt. Here onwards Application
> @ mss util.c		324	can enable and use any interrupts as required*/
> 🕞 mss util h		325	clear_soft_interrupt();
> i mtran c		326	
> R mtrap.c		327	<pre>enable_irq();</pre>
<ul> <li>R nowlin stubs c</li> </ul>		328	
> Revetere startup s		329	white(1)
> B system_startup.c		331	volatile static uint64 t counter = 0U:
<ul> <li>a system_startup.n</li> </ul>		332	
mprs-gpio-example nw-emulation all-narts debug.launce	n	333	/* Added some code as debugger hangs if in loop doing nothing */
mpts-gpio-example hw-emulation all-harts release.laund	n	334	counter = counter + 1;
mpts-gpio-example renode attach-to-hart0.launch		335	}
mpts-gpio-example renode attach-to-hart1.launch		337	
mpfs-gpio-example renode start-platform-and-debug.la	unch	220 /	

This function is entered if the hart is taken out of WFI (if the **MPFS\_HAL\_LAST\_HART** value still includes this hart, it is included in the system) and simply puts the hart back into WFI as no code is found for it to run.

### 3.7 Bare Metal Development

The firmware drivers and associated platform specific files are available on GitHub Bare Metal Library. The Bare Metal application(s) can be executed from one of the memories—LIM, eNVM, DDR memory and so on. Linker script files to execute applications from corresponding memories are also available on GitHub Bare Metal Library. If the application size is more than eNVM size (128 kB), it is recommended to store the application in an external flash memory. The HSS that runs on E51 fetches the application from the external memory and buffers it in DDR, it then copies to the address in LIM or DDR that the application runs from and executes the application. The E51 core releases the U54 cores from WFI depending upon the application requirements. The SoftConsole tool provides an environment to develop Bare Metal applications.

### 3.7.1 Single U54

The Bare Metal start-up code (mpfs\_hal) initializes the system clocks and external memory. For single U54 Bare Metal development, the E51 wakes the U54 hart from WFI mode by raising the software interrupt. The U54 executes the application tasks. The remaining harts are kept in WFI mode by the default weakly linked functions defined in the start-up code.

### 3.7.2 Multiple U54s

For multiple U54 Bare Metal development, the E51 wakes any combination of U54 harts from WFI mode by raising the software interrupt on each hart. The unused harts are held in the WFI mode. All of the harts run the same start-up code, read their hart ID, and enter WFI. When the U54s receive a software interrupt from E51, they exit the WFI mode and execute their application(s). The U54 harts can execute the same application or different individual applications depending upon application requirements. The HSS provides the necessary functions for the E51 hart to communicate with individual U54 harts to perform certain services on their behalf.

### 3.7.3 Initializing the Application Execution Space (LIM or DDR)

The E51 core runs the HSS to fetch a payload containing an application or applications from an external memory. It buffers the application to DDR and then copies the application to the destination memory and releases the U54(s) from WFI. The U54 cores run the application from the destination memory. Linker script files to execute applications from different memory locations are available from the PolarFire SoC Bare Metal Library .

The HSS uses a portion of the LIM while running. The Bare Metal application start addresses must be greater than the end address of memory used by HSS. When built a .map file is produced for the HSS outlining memory locations used for functions, variables, and so on. The following image shows a output.map file.

#### Figure 3-14. HSS output.map File

Kconfig
 Kconfig.build
 Kconfig.general
 LICENSE.md
 Makefile
 output.map
 README.md
 rules.mk
 targets.mk

This file can be opened in the SoftConsole text editor or an external editor. The last region of memory used by HSS is the stack, searching the map file for  $\__stack\_top\_h4\$$  shows the last address used for data in the stack, as shown in the following figure.

Figure 3-15.	Stack Location	HSS	output.map	File
--------------	----------------	-----	------------	------

output	.map 🛙						
11297 11298	.bss *(COMMON)	0x0000000080116e0	0×200	thirdparty/opensbi/build/lib/libsbi.a(	sbi_ipi.o)		
11299		0x0000000080118e0		. = ALIGN (0x10)	CC C. LUD. L.		
11300		0x00000000080118e0		bss end = .	SC Find/Replac	e –	
11301							
11302	stack	0x000000008011900	0x14000	load address 0x000000002023bd80	Eind:	stack_te	op_h4\$ ~
11303		0x000000008011900		stack bottom = .	Dankara with		
11304		0x000000008011900		stack bottom h0\$ = .	Replace with:		~
11305		0x000000008015900		. = (. + STACK SIZE PER HART)	Direction	C.	
11306	*fill*	0x000000008011900	0x4000		Direction	30	cope
11307		0x000000008015900		stack top h0\$ = .	• Forward	۲	) All
11308		0x000000008015900		stack bottom h1\$ = .	Backward	C	Selected lines
11309		0x000000008019900		. = (. + STACK SIZE PER HART)			
11310	*fill*	0x000000008015900	0x4000		Options		
11311		0x000000008019900		stack top h1\$ = .	Case sensit	tive 🗹 W	/rap search
11312		0x000000008019900		stack_bottom_h2\$ = .	Whole wor	d 🗌 In	cremental
11313		0x00000000801d900		. = (. + STACK_SIZE_PER_HART)			
11314	*fill*	0x000000008019900	0x4000		Kegular exp	pressions	
11315		0x00000000801d900		stack_top_h2\$ = .	0.00		
11316		0x00000000801d900		stack bottom h3\$ = .	Find		Replace/Find
11317		0x000000008021900		. = (. + STACK_SIZE_PER_HART)			
11318	*fill*	0x00000000801d900	0x4000		<u>R</u> epla	ce	Replace All
11319		0x000000008021900		stack_top_h3\$ = .			
11320		0x000000008021900		stack_bottom_h4\$ = .			Close
11321		0x000000008025900		. = (. + STACK_SIZE_PER_HART)	-		
11322	*fill*	0x000000008021900	0x4000				
11323		0x000000008025900		stack_top_h4\$ = .			
11324		0x000000008025900		stack_top = .			
11325		0x000000008025900		_end = .			
113260	OUTPUT(hss.el	<pre>lf elf64-littleriscv)</pre>					

In the preceding example, the last address used in the HSS is 0x8025900 and the target Bare Metal application must use an address greater than this as its start address.

The standard Bare Metal Library applications use E51 as the main hart of the system. When the E51 hart runs the HSS, a different hart, one of the U54s, must be used to wake harts in use from the WFI mode. The main hart defined in the  $mss_sw_config.h$  file in <code>platform/mpfs\_hal\_config</code> must be changed to reflect that the U54 core is the main hart.

The following figure shows a standard <code>mss\_sw\_config.h</code> configuration.

Figure 3-16. Standard mss\_sw\_config.h File

```
h mss_sw_config.h 🖾
 31 #ifndef USER_CONFIG_MSS_USER_CONFIG_H_
 32 #define USER_CONFIG_MSS_USER_CONFIG_H_
 33
  340 /*----
     * MPFS HAL FIRST HART and MPFS HAL LAST HART defines used to specify which
 35
    * harts to actually start.
  36
  37 * Set MPFS HAL FIRST HART to a value other than 0 if you do not want your code
  38
     * to start and execute code on the E51 hart.
     * Set MPFS HAL LAST HART to a value smaller than 4 if you do not wish to use
  39
     * all U54 harts.
 40
 41
      * Harts that are not started will remain in an infinite WFI loop unless used
 42 * through some other method
 43 */
 440 #ifndef MPFS HAL FIRST HART
 45 #define MPFS HAL FIRST HART
                                   0
 46 #endif
 47
 48⊖ #ifndef MPFS HAL LAST HART
 49 #define MPFS HAL LAST HART
                                   4
 50 #endif
 51
 500 /*
   The MPFS HAL FIRST HART is set to 0 and the MPFS HAL LAST HART is set to 4. The hart values are as
   follows:
```

E51: Hart 0

U54\_1: Hart 1

U54\_2: Hart 2

U54\_3: Hart 3

U54\_4: Hart 4

Changing the MPFS\_HAL\_FIRST\_HART value to 1 sets U54\_1 as the main hart of the system.

```
Figure 3-17. mss_sw_config.h File Using U54_1 as the Main Hart
```

```
h *mss_sw_config.h ⊠
 310 #ifndef USER CONFIG MSS_USER_CONFIG_H
 32 #define USER_CONFIG_MSS_USER_CONFIG_H_
 33
 340 /*-----
                              * MPFS HAL FIRST HART and MPFS HAL LAST HART defines used to specify which
 35
 36
     * harts to actually start.
     * Set MPFS HAL FIRST HART to a value other than 0 if you do not want your code
 37
     * to start and execute code on the E51 hart.
 38
     * Set MPFS HAL LAST HART to a value smaller than 4 if you do not wish to use
 39
 40
     * all U54 harts.
     * Harts that are not started will remain in an infinite WFI loop unless used
 41
 42
     * through some other method
 43
     */
 44⊖ #ifndef MPFS HAL FIRST HART
 45 #define MPFS HAL FIRST HART
                                 1
 46 #endif
 47
 48 #ifndef MPFS_HAL_LAST_HART
 49 #define MPFS HAL LAST HART
                                 4
 50 #endif
 51
 ED0 /*
```

This implies that on system start up U54\_1 wakes the other three U54 harts from WFI. If less harts are used by this application (for example, U54\_1 and U54\_2 only), then the last hart value can be changed.

Figure 3-18. mss\_sw\_config.h File

```
h *mss_sw_config.h ⊠
 319 #ifndef USER CONFIG MSS USER CONFIG H
    #define USER CONFIG MSS USER CONFIG H
 32
 33
 34 /*-----
                        _____
 35
     * MPFS HAL FIRST HART and MPFS HAL LAST HART defines used to specify which
 36
      * harts to actually start.
 37
     * Set MPFS HAL FIRST HART to a value other than 0 if you do not want your code
 38
     * to start and execute code on the E51 hart.
     * Set MPFS HAL LAST HART to a value smaller than 4 if you do not wish to use
 39
 40
     * all U54 harts.
 41
     * Harts that are not started will remain in an infinite WFI loop unless used
     * through some other method
 42
     */
 43
 44⊕ #ifndef MPFS HAL FIRST HART
    #define MPFS HAL FIRST HART
 45
                                1
 46
    #endif
 47
 48⊕ #ifndef MPFS HAL LAST HART
 49 #define MPFS HAL LAST HART
                                2
 50
    #endif
 51
 52 /*-----
```

This uses U54\_1 to wake U54\_2 and then both harts continue on to the application.

### 3.7.4 Merging Multiple Bare Metal Applications

To merge Bare Metal applications, see the HSS readme in its GitHub repository.

### 3.8 Linux Application Development

A typical boot process consists of multiple stages.

- The HSS is executed from the eNVM. The HSS acts as a ZSBL that includes the Open Source Supervisor Binary Interface (OpenSBI). OpenSBI is a platform-specific firmware running in M-mode. It acts as an interface between the HSS and U-boot. The HSS loads the first stage boot loader (U-Boot) from a boot device to an external RAM. The HSS uses the OpenSBI functions to switch the execution mode from M-mode to S-mode when transferring execution control to U-Boot.
- 2. U-Boot initializes the peripherals and loads the kernel. The boot device can either be an embedded memory microcontroller (eMMC) or an SD card. U-Boot loads the Linux kernel from the boot device to DDR.
- 3. In the next stage, Linux is executed (from DDR).
- 4. Init is the first process executed by the Linux kernel and it is the parent of all processes.
- 5. In the final stage, user applications are executed in Linux.

The following figure shows the boot process flow.

### Figure 3-19. Linux Boot Process Flow



### 3.8.1 Building Linux Images

Linux images can be built using the Yocto or Buildroot build systems. Both of these systems come with a readme that lists the required packages and build steps.

The Microchip Yocto BSP can be found at: github.com/polarfire-soc/meta-polarfire-soc-yocto-bsp

It supports several board targets and build configurations for different images (for example, minimal/development tools).

The Microchip Buildroot SDK can be found at: github.com/polarfire-soc/polarfire-soc-buildroot-sdk

It supports several board targets and is pre-configured to generate a minimal image.

### 3.8.2 Integrating Linux Applications in Yocto

### 3.8.2.1 Existing Linux Applications

To integrate an existing Linux application (package) into Yocto, ensure that the package is a part of the Yocto source and add the package to the final image. An example on integrating Linux package using apache2 is shown in the following steps.

1. Find the package .bb file in the Yocto repository apache2 (web server), which is shown as an example in the following code snippet.

```
microchip@microchip-OptiPlex-9010:~/riscv/icicle/yocto-dev$
microchip@microchip-OptiPlex-9010:~/riscv/icicle/yocto-dev$ find ./meta-* -name apache2
./meta-openembedded/meta-webserver/recipes-httpd/apache2
./meta-openembedded/meta-webserver/recipes-httpd/apache2/apache2
```

2. Ensure that the package meta layer directory is present in the bblayers.conf file.

```
microchip@microchip-OptiPlex-9010:~/riscv/icicle/yocto-dev$
microchip@microchip-OptiPlex-9010:~/riscv/icicle/yocto-dev$ ls build/conf/bblayers.conf
build/conf/bblayers.conf
```

3. If the package meta layer directory is not part of bblayers.conf file, add the directory path as highlighted in the following code snippet.

4. The following code snippet shows the mpfs-dev-cli.bb files folder in Yocto source for PolarFire SoC, open the mpfs-dev-cli.bb file.

```
microchip@microchip-OptiPlex-9010:~/riscv/icicle/yocto-dev$
microchip@microchip-OptiPlex-9010:~/riscv/icicle/yocto-dev$ ls meta-polarfire-soc-yocto-
bsp/recipes-core/images/
mpfs-dev-cli.bb riscv-initramfs-image.bb
```

5. The mpfs-dev-cli.bb files show a list of packages added in the PolarFire SoC device. Add apache2 package in the existing list as shown in the following code snippet.

```
DESCRIPTION = "Microchip MPFS Development CLI Linux image"
inherit image-buildinfo core-image extrausers
EXTRA USERS PARAMS = "usermod -P microchip root;"
IMAGE FEATURES += " ssh-server-openssh \
                    tools-debug tools-sdk debug-tweaks \
                    dev-pkgs dbg-pkgs \
IMAGE INSTALL = "\setminus
   packagegroup-core-boot \
    packagegroup-core-full-cmdline \
    perl-modules \
    alsa-utils \
    i2c-tools \
    apache2 \
    screen \
    apps \
    vim vim-vimrc \
    dhcp-client \
    nbd-client \
    mpfr-dev \
    gmp-dev \
    libmpc-dev \
    zlib-dev \
    flex \setminus
    bison \
    dejagnu \
    gettext \
    texinfo \
    procps \
    glibc-dev \
    elfutils \
    elfutils-dev \
    pciutils \setminus
    usbutils \
    mtd-utils \
    sysfsutils \
    htop \setminus
    iw \
    python3 \
    git \
    swig \
    boost \
    orc \
    libudev \
    glib-2.0 \
    evtest devmem2 iperf3 memtester 1mbench \
    tcpdump \
    iw \
    libudev \
    nano \
    nfs-utils-client \
    cifs-utils \setminus
    openssh-sftp \
    openssh-sftp-server \setminus
    procps \
    protobuf \
    ntp ntpdate ntp-utils \
    linux-firmware \
    libsodium \
    sqlite3
    tar \
    wget \
    zip \
    unzip \
    rsync \
    kernel-modules kernel-devsrc kernel-dev \
    ${CORE IMAGE EXTRA INSTALL} \
```

Upon successful addition of the apache2 package as shown in the preceding steps, Yocto source can be built.

### 3.8.2.2 Custom Linux Applications

To integrate an existing Linux application into Yocto, ensure that the particular package is part of the Yocto source and add the package to the final image. An example of integrating Linux application using apache2 is shown in the following steps.

Note: The user package recipe must be placed under the apps folder to be part of the Yocto build process.

```
The following code snippet shows a sample application .bb file.
```

```
#
# This file was derived from the 'Hello World!' example recipe in the
# Yocto Project Development Manual.
#
DESCRIPTION = "Simple application to blink LEDs"
SECTION = "examples"
LICENSE = "MIT"
LIC FILES CHKSUM = "file://${COMMON LICENSE DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
PR = "r0"
SRC_URI = "file://led_blinky.c \
S = "${WORKDIR}/"
do compile() {
                  ${CC} led blinky.c ${LDFLAGS} -o led blinky
}
FILES ${PN} += "/microchip-apps"
do install() {
               install -d ${D}/microchip-apps
              install -m 0755 led blinky.c ${D}/microchip-apps
install -m 0755 led_blinky ${D}/microchip-apps
}
```

The mpfs-dev-cli.bb file shows the list of packages added in the PolarFire SoC device. Add apps package in the existing list as shown in the following code snippet.

```
DESCRIPTION = "Microchip MPFS Development CLI Linux image"
inherit image-buildinfo core-image extrausers
EXTRA USERS PARAMS = "usermod -P microchip root;"
IMAGE FEATURES += " ssh-server-openssh \
                    tools-debug tools-sdk debug-tweaks \
                    dev-pkgs dbg-pkgs \
IMAGE INSTALL = "\setminus
   packagegroup-core-boot \
    packagegroup-core-full-cmdline \
    perl-modules \
    alsa-utils \
    i2c-tools \
    screen \
    apps \
    vim vim-vimrc \
    dhcp-client \
    nbd-client \
    mpfr-dev \
    gmp-dev \
    libmpc-dev \
    zlib-dev \
    flex \
    bison ∖
    dejagnu \
    gettext \
    texinfo \
    procps \
    glibc-dev \
    elfutils \
    elfutils-dev \
    pciutils \
    usbutils \
    mtd-utils \
    sysfsutils \
    htop ∖
    iw \
    python3 \
    git \
    swig \
    boost
    orc \
    libudev \
    glib-2.0 \
    evtest devmem2 iperf3 memtester lmbench \
    tcpdump \
    iw \
    libudev \
    nano \
    nfs-utils-client \setminus
    cifs-utils \setminus
    openssh-sftp \
    openssh-sftp-server \setminus
    procps \
    protobuf \
    ntp ntpdate ntp-utils \
    linux-firmware \
    libsodium \
    sqlite3 \
    tar \
    wget \
    zip \
    unzip \
    rsync \
    kernel-modules kernel-devsrc kernel-dev \
    {CORE IMAGE EXTRA INSTALL} \setminus
```

Upon successful addition of the apps package as shown in the preceding code snippet, Yocto source can be built.

For more information about Yocto, see Yocto Reference Manual.

Microchip offers a complete Yocto source image for PolarFire SoC Icicle Kit that can be used for customized Linux images including application development. See GitHub page for more information about the directory structure, source files, and documentation.

#### 3.8.3 Integrating Linux Application in Buildroot

#### 3.8.3.1 Existing Linux Applications

To integrate existing Linux application (package) into Buildroot, go to the PolarFire SoC Buildroot SDK path and execute the following commands.

• To configure an eMMC target, execute the command:

```
microchip@microchip-OptiPlex-9010:~/riscv/icicle/polarfire-soc-buildroot-sdk$ make
DEVKIT=icicle-kit-es buildroot_initramfs-menuconfig
```

• To configure an SD card target, execute the command:

```
microchip@microchip-OptiPlex-9010:~/riscv/icicle/polarfire-soc-buildroot-sdk$ make
DEVKIT=icicle-kit-es-sd buildroot_rootfs-menuconfig
```

When the make command is successfully executed, the Config menu appears as shown in the following figure. Search for the package that needs to be added. **Select** the package and **Save**. Apache is used as an example package in the following figure.

#### Figure 3-20. Apache Buildroot

/home/microchip/riscv/icicle/polarfire-soc-buildroot-sdk/work/buildroot_rootfs/.config - Buildroot 2020.05-983-gfc3d6a3ed0 Configuration								
Artige: packages > Networking applications     Aetworking applications								
Arrow keys navigate the menu. <enter> selects submenus&gt; (or empty submenus). Highlighted letters are hotkeys. Pressing <y> selects a feature, while <n> excludes a feature. Press <esc><esc> to exit, <? > for Help,  for Search. Legend: [*] feature is selected [ ] feature is excluded</esc></esc></n></y></enter>								
[] aircrack-ng [] aoetools								
[ <mark>] apache</mark> Multi-Processing Module (MPM) (worker)>								
[] argus								
[] arr-scan [] arrbales-legacy								
[] asterisk								
[] attp [] avai								
[] axel								
[] babeld								
[ ] bardwidtid								
[] bcusdk								
[] bind								
[] bluez-utils 5.x								
[ ] brcm-patchram-plus								
[] bridge-utils								
[] C-icap								
[] can-utils								
[] cannettoni								
[] civetweb								
<pre><select> &lt; Exit &gt; &lt; Help &gt; &lt; Save &gt; &lt; Load &gt;</select></pre>								

Upon successful addition of the apache package as shown in the preceding image, the Buildroot source can be built.

#### 3.8.3.2 Custom Linux Applications

To integrate a custom Linux application (package) into Buildroot, follow these steps:

 Create a new directory in the package folder of Buildroot source. For example, microchip-apps package folder is used as an example, which consists of two files—Config.in and microchip-apps.mk. If you have any C source files, create a directory files to store the source files. The following code snippet shows the microchip-apps folder.

```
microchip@microchip-OptiPlex-9020:~/work/icicle/polarfire-soc-buildroot-sdk/buildroot/
package/microchip-apps$ ls
Config.in files microchip-apps.mk
```

microchip@microchip-OptiPlex-9020:~/work/icicle/polarfire-soc-buildroot-sdk/buildroot/ package/microchip-apps\$

2. Create a Config.in file with the following code snippet.

3. Create a microchip-apps.mk file with the following code snippet.

4. Edit the parent package config file (package/Config.in) to include the microchip-apps package.

```
source "package/microchip-apps/Config.in"
```

5. Add the microchip-apps package to the buildroot\_initramfs\_config file to build the microchipapps package as part of the eMMC Linux image.

```
BR2_TOOLCHAIN_EXTERNAL_CUSTOM_GLIBC=y
BR2_PACKAGE_GDB_NEEDS_CXX11=y
BR2_TOOLCHAIN_EXTERNAL_CXX=y
BR2_TOOLCHAIN_EXTERNAL_INET_RPC=y
BR2_ROOTFS_DEVICE_CREATION_DYNAMIC_MDEV=y
BR2_TARGET_GENERIC_ROOT_PASSWD="microchip"
BR2_SYSTEM_DHCP="eth0"
BR2_PACKAGE_DHRYSTONE=y
BR2_PACKAGE_DHRYSTONE=y
BR2_PACKAGE_LMBENCH=y
BR2_PACKAGE_MTD=y
BR2_PACKAGE_MTD=y
BR2_PACKAGE_MTD=y
BR2_PACKAGE_MTD_NANDDUMP is not set
# BR2_PACKAGE_MTD_NANDDWFITE is not set
# BR2_PACKAGE_MTD_NANDWRITE is not set
# BR2_PACKAGE_MTD_NANDWRITE is not set
# BR2_PACKAGE_MTD_NANDWRITE is not set
```

Upon successful addition of the apps package as shown in the preceding steps, Buildroot source can be built.

### 3.8.4 Different Sources of Booting

The PolarFire SoC target hardware runs the HSS from eNVM to load the Linux image either from eMMC or SD card depending on its configuration when built as shown in the following figure.

### Figure 3-21. Different Sources of Booting



When the customized Linux images are available, USB, UART or Ethernet can be used to transfer the image to eMMC or SD card.

To boot Linux on Icicle kit using eMMC, see GitHub.

### 3.8.5 Device Tree Source (DTS)

A Device Tree is a data structure for describing the hierarchy of hardware subsystems within a hardware platform, or an add-on peripheral to that platform. It is used to select and configure the device drivers for Embedded Linux platform during the boot process. It can be represented in different formats, such as

- Device Tree Source format (.dts)
- Compiled binary Device Tree binary format (.dtb)
- For example, icicle-kit-es-a000-microchip.dts and icicle-kit-es-a000-microchip.dtb

Additional peripherals like GPIO, LSRAM, and so on, can be added to a kernel by including them in the DTS.

### 3.8.5.1 Adding a Sample Device Node for GPIO

The following example device node can be added to a Device Tree.

```
gpio@20122000 {
   compatible = "microsemi,ms-pf-mss-gpio";
   interrupt-parent = <&Ll>;
   interrupts = <53 53 53 53 53 53 53 53>;
   gpio-controller;
   reg = <0x0 0x20122000 0x0 0x1000>;
   reg-names = "control";
   status = "okay";
   };
```

### 3.8.5.2 Adding a Sample Device node for LSRAM (UIO Framework)

The following example device node can be added to device tree (icicle-kit-es-a000-microchip.dts).

```
uio_lsram@0x60000000 {
    compatible = "generic-uio";
    reg = < 0x0 0x60000000 0x0 0x00010000 // LSRAM0 Memory
    0x0 0x60010000 0x0 0x00010000 >; //LSRAM1 Memory
    status = "okay";
};
```

Hardware that is ideally suited for an UIO driver fulfills all the following.

- · The device has memory that can be mapped. The device can be controlled completely by writing to this memory.
- The device usually generates interrupts.
- The device does not fit into one of the standard kernel subsystems.

# 4. Appendix

### HSS

The HSS is a ZSBL that is stored in the eNVM of PolarFire SoC. It can be used to program memories and boot applications running on different harts in the system. It runs in a super loop executing on the E51 core and provides a machine mode trap handler to pass messages between the U54 harts.

The HSS can be found on GitHub here: github.com/polarfire-soc/hart-software-services

### HAL

The MPFS HAL provides the initial boot code, interrupt handling, hardware access methods for the PolarFire SoC MSS and DDR training code. The terms PolarFire-SoC HAL and MPFS HAL are used interchangeably but the term MPFS HAL is preferred. The MPFS HAL is a combination of C and assembly source code.

Location of the repository: github.com/polarfire-soc/polarfire-soc-bare-metal-library/tree/master/src/platform/mpfs\_hal

### Peripheral Driver Library

The PolarFire SoC Bare Metal Library includes:

- Source code for start-up code and HAL the PolarFire SoC MSS.
- Source code for the PolarFire SoC MSS peripheral drivers.
- Documentation for the HAL and peripheral drivers.
- SoftConsole example projects demonstrating the use of the various PolarFire SoC peripherals.

Location of the repository: github.com/polarfire-soc/polarfire-soc-bare-metal-library

Presentation recording: microchip.webex.com/microchip/lsr.php?RCID=ed4bf40309bd37afa46d9723270bb192

Presentation slides: www.microsemi.com/document-portal/doc\_download/1244894-polarfire-soc-renode-webinar-10

# 5. 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.

Revision	Date	Description
В	05/2021	Updated the References section
A	09/2020	Initial Revision

# The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

# **Product Change Notification Service**

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

# Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- · Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

## **Microchip Devices Code Protection Feature**

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- · Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code
  protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly
  evolving. We at Microchip are committed to continuously improving the code protection features of our products.
  Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act.
  If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue
  for relief under that Act.

## Legal Notice

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

# Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

<sup>©</sup> 2021, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-8247-5

# Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.



# **Worldwide Sales and Service**

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office	Australia - Sydney	India - Bangalore	Austria - Wels
2355 West Chandler Blvd.	Tel: 61-2-9868-6733	Tel: 91-80-3090-4444	Tel: 43-7242-2244-39
Chandler, AZ 85224-6199	China - Beiiing	India - New Delhi	Fax: 43-7242-2244-393
Tel: 480-792-7200	Tel: 86-10-8569-7000	Tel: 91-11-4160-8631	Denmark - Copenhagen
Fax: 480-792-7277	China - Chengdu	India - Pune	Tel: 45-4485-5910
Technical Support:	Tel: 86-28-8665-5511	Tel: 91-20-4121-0141	Fax: 45-4485-2829
www.microchip.com/support	China - Chongging	Japan - Osaka	Finland - Espoo
Web Address:	Tel: 86-23-8980-9588	Tel: 81-6-6152-7160	Tel: 358-9-4520-820
www.microchip.com	China - Dongguan	Japan - Tokvo	France - Paris
Atlanta	Tel: 86-769-8702-9880	Tel: 81-3-6880- 3770	Tel: 33-1-69-53-63-20
Duluth, GA	China - Guangzhou	Korea - Daegu	Fax: 33-1-69-30-90-79
Tel: 678-957-9614	Tel: 86-20-8755-8029	Tel: 82-53-744-4301	Germany - Garching
Fax: 678-957-1455	China - Hangzhou	Korea - Seoul	Tel: 49-8931-9700
Austin. TX	Tel: 86-571-8792-8115	Tel: 82-2-554-7200	Germany - Haan
Tel: 512-257-3370	China - Hong Kong SAR	Malaysia - Kuala Lumpur	Tel: 49-2129-3766400
Boston	Tel: 852-2943-5100	Tel: 60-3-7651-7906	Germany - Heilbronn
Westborough, MA	China - Naniing	Malavsia - Penang	Tel: 49-7131-72400
Tel: 774-760-0087	Tel: 86-25-8473-2460	Tel: 60-4-227-8870	Germany - Karlsruhe
Eax: 774-760-0088	China - Qingdao	Philippines - Manila	Tel: 49-721-625370
Chicago	Tel: 86-532-8502-7355	Tel: 63-2-631-9065	Germany - Munich
	China - Shanghai	Singanore	Tel: 40-80-627-144-0
Tel: 630-285-0071	Tel: 86-21-3326-8000	Tel: 65-6334-8870	Fax: 40-80-627-144-0
Fax: 630-285-0075	China - Shenyang	Taiwan - Hsin Chu	Germany - Rosenheim
Dallas	Tel: 86-24-2334-2820	Tel: 886-3-577-8366	Tel: 10-8031-351-560
Addison TY	China Shonzhon	Taiwan Kaobsiung	
Tal: 072 818 7423	Tol: 86 755 8864 2200	Tal: 886 7 213 7830	Tal: $072$ 0 744 7705
Fex: 072 919 2024	China Suzbou	Teiwan Teinei	Itely Miler
Detroit	Tol: 86 186 6233 1526	Tal: 886 2 2508 8600	Tal: 30 0331 7/2611
Nevi MI	China Wuhan	Theiland Bangkok	Eax: 30 0331 466781
Tol: 248 848 4000	Tal: 96 27 5090 5200		Italy Badaya
Heuston TY	China Vian	Vietnem He Chi Minh	
Tol: 281 804 5083			Nothorlands Drupon
Indianapolis	China Viamon	Tel: 04-20-3440-2100	Tol: 31 416 600300
Noblosvillo, IN	Tol: 86 502 2388138		Eax: 31 416 600340
	China Zhubai		Nervey Trendheim
Fex: 217 772 5452			Tal: 47 72994299
1 al. 311-113-3433	161. 00-700-3210040		Boland Warsaw
			Tal. 49.00.0205727
Los Aligeles			Pomania Rucharaat
101. 343-402-3023			101.40-21-407-07-00 Spain Madrid
Fax. 343-402-3000			Spail - Wauriu
101. 901-213-1000 Releigh NC			Eav: 24.01.709.02.04
<b>Railey II, NG</b>			Fax. 34-91-100-00-91
New TOFK, NT			1et: 40-31-704-00-40
San Jose, CA			1ei: 46-8-5090-4654
Iei: 408-436-4270			Iei: 44-118-921-5800
Canada - Toronto			⊢ax: 44-118-921-5820
Tel: 905-695-1980			
⊦ax: 905-695-2078			