Sapphire RISC-V SoC Hardware and Software User Guide

UG-RISCV-SAPPHIRE-v6.1
June 2024
www.efinixinc.com

Contents

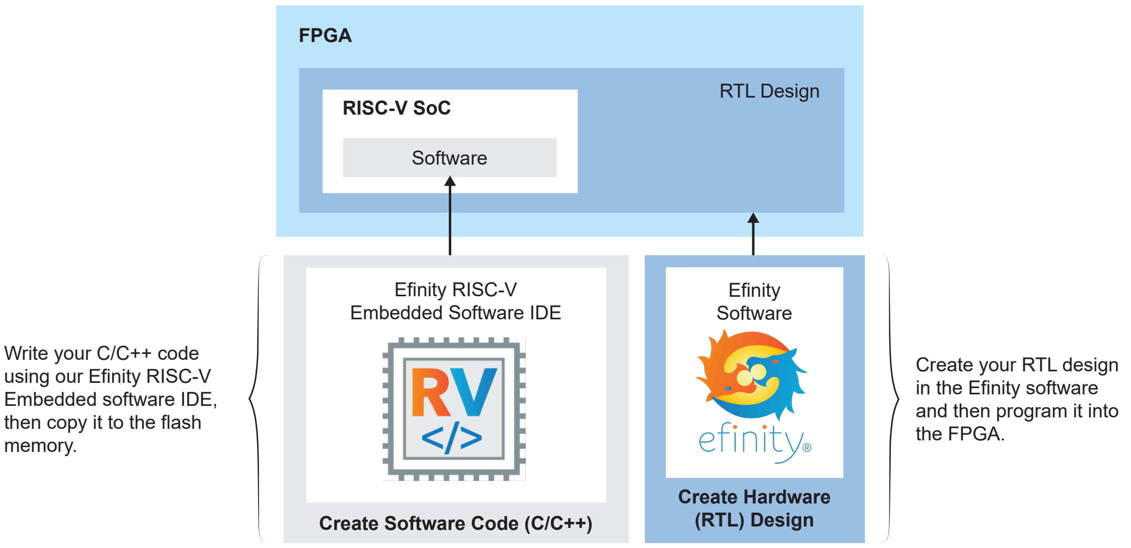
Introduction

Efinix provides a soft configurable RISC-V SoC, called Sapphire, that you can implement in Trion® or Titanium FPGAs. This user guide describes how to:

- Build RTL designs using the Sapphire RISC-V SoC using an example design targeting an Efinix® development board, and how to extend the example for your own application.
- Set up the software development environment using an example project, create your own software based on example projects, and use the API.

Note: The Sapphire SoC v2.0, released with the Efinity software v2021.2, has significant improvements from previous versions, and you cannot migrate an existing design to it automatically. Efinix recommends that you use v2.0 or higher for all new designs. You can continue to use previous versions with the Efinity software v2021.1. If you want to migrate an existing design to v2.0, refer to **Migrating to the Sapphire SoC v2.0 from a Previous Version** on page 116.

Figure 1: Designing Hardware and Software for the Sapphire RISC-V SoC


FPGA
RTL Design
RISC-V SoC
Software
Write your C/C++ code using our Efinity RISC-V Embedded software IDE, then copy it to the flash memory.
Efinity RISC-V Embedded Software IDE
RV </>
Create Software Code (C/C++)
Efinity Software
efinity
Create Hardware (RTL) Design
Create your RTL design in the Efinity software and then program it into the FPGA.

Learn more: Refer to the **Sapphire RISC-V SoC Data Sheet** for detailed specifications on the SoC.

VexRiscv RISC-V Core

The Sapphire SoC is based on the VexRiscv core created by Charles Papon. The VexRiscv core is a 32-bit CPU using the ISA RISCV32I with M, A, F, D, and C extensions, has six pipeline stages (fetch, injector, decode, execute, memory, and writeback), and a configurable feature set.

In the Sapphire SoC, the VexRiscv core is user configurable, and can support AXI4 and APB3 bus interfaces and instruction and data caches. The Sapphire SoC VexRiscv core uses Little-Endian for its memory storage.

The VexRiscv core won first place in the RISC-V SoftCPU contest in 2018.[1]

[1] https://www.businesswire.com/news/home/20181206005747/en/RISC-V-SoftCPU-Contest-Winners-Demonstrate-Cutting-Edge-RISC-V

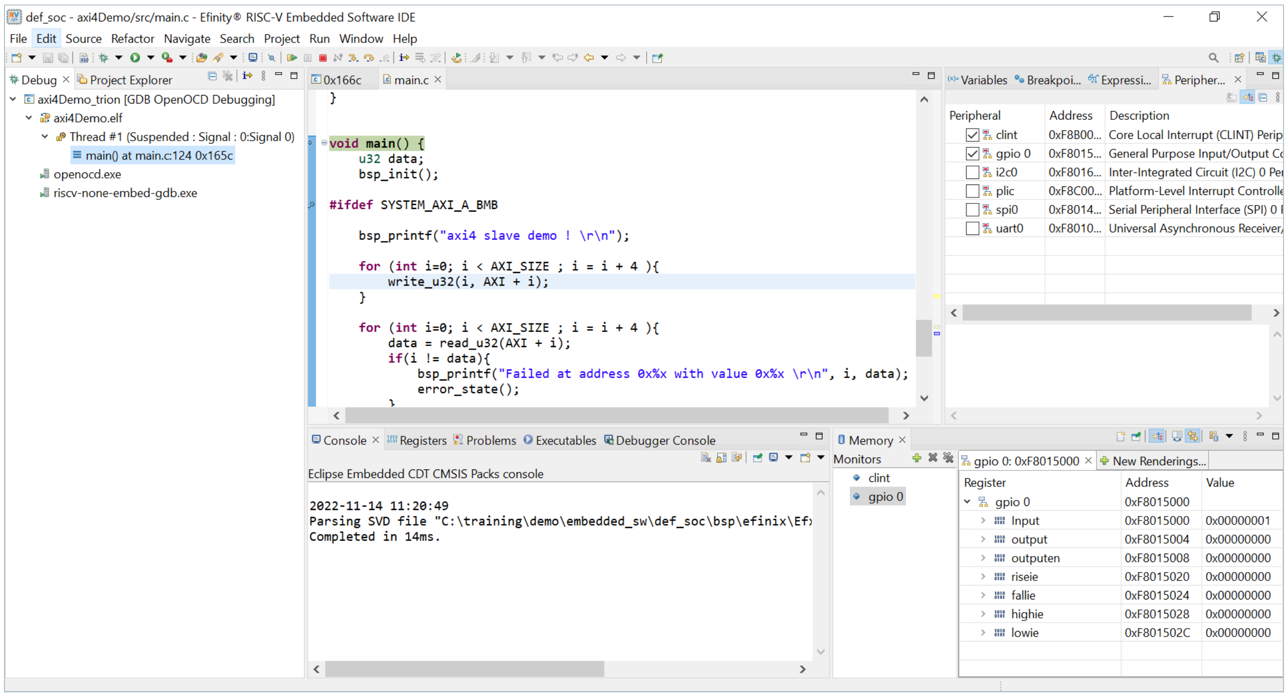
Efinity® RISC-V Embedded Software IDE

The Efinity® RISC-V Embedded Software IDE is an Eclipse-based Integrated Development Environment (IDE) powered by Ashling's ***RiscFree***™ IDE for Sapphire SoC. It provides a complete and seamless environment for RISC-V C and C++ software development.

Features include:

- Eclipse based IDE with full source project creation, edit, build, and debug
- QEMU emulator support for 32-bit RISC-V cores with out-of-box example design
- High-level Peripheral Register viewer
- Control and Status Register (CSR) viewer
- Integrated new project creation process with Board Support Package (BSP) generated in the Efinity software
- Integrated example program import process with Board Support Package (BSP) generated in the Efinity software
- Integrated serial terminal for viewing UART data
- FreeRTOS task and queue list debug view
- Debug support for all OpenOCD compliant probes

Figure 2: Efinity RISC-V Embedded Software IDE



```
axi4Demo_trion [GDB OpenOCD Debugging]
  axi4Demo.elf
    Thread #1 (Suspended : Signal : 0:Signal 0)
      main() at main.c:124 0x165c
  openocd.exe
  riscv-none-embed-gdb.exe
```

```
}


void main() {
    u32 data;
    bsp_init();

#ifdef SYSTEM_AXI_A_BMB

    bsp_printf("axi4 slave demo ! \r\n");

    for (int i=0; i < AXI_SIZE ; i = i + 4 ){
        write_u32(i, AXI + i);
    }

    for (int i=0; i < AXI_SIZE ; i = i + 4 ){
        data = read_u32(AXI + i);
        if(i != data){
            bsp_printf("Failed at address 0x%x with value 0x%x \r\n", i, data);
            error_state();
```

Peripheral	Address	Description
✓ clint	0xF8B00...	Core Local Interrupt (CLINT) Perip
✓ gpio 0	0xF8015...	General Purpose Input/Output Co
i2c0	0xF8016...	Inter-Integrated Circuit (I2C) 0 Per
plic	0xF8C00...	Platform-Level Interrupt Controlle
spi0	0xF8014...	Serial Peripheral Interface (SPI) 0
uart0	0xF8010...	Universal Asynchronous Receiver/

Eclipse Embedded CDT CMSIS Packs console

```
2022-11-14 11:20:49
Parsing SVD file "C:\training\demo\embedded_sw\def_soc\bsp\efinix\Ef
Completed in 14ms.
```

clint
gpio 0

Register	Address	Value
gpio 0	0xF8015000	
Input	0xF8015000	0x00000001
output	0xF8015004	0x00000000
outputen	0xF8015008	0x00000000
riseie	0xF8015020	0x00000000
fallie	0xF8015024	0x00000000
highie	0xF8015028	0x00000000
lowie	0xF801502C	0x00000000

Required Software

To write software for the Sapphire SoC, you need the following tools. The Efinity RISC-V Embedded Software IDE installer for Windows and Linux operating systems are available in the **Efinity software download page**.

Efinity® Software

Efinix® development environment for creating RTL designs targeting Trion® or Titanium FPGAs. The software provides a complete RTL-to-bitstream flow, simple, easy to use GUI interface, and command-line scripting support.

Version: 2021.1 or higher

Efinity RISC-V Embedded Software IDE

The Efinity RISC-V Embedded Software IDE is an Eclipse-based Integrated Development Environment (IDE) powered by Ashling's ***RiscFree***™ IDE for Sapphire SoC and provides a complete provides a complete, seamless environment for RISC-V C and C++ software development. The RISC-V IDE includes the following packages:

Disk space required: 2.4 GB (Windows), 2.5 GB (Linux)

xPack GNU RISC-V Embedded GCC—Open-source, prebuilt toolchain from the xPack Project.

Version: 8.3.0-2.3

Disk space required: 1.53 GB (Windows), 1.5 GB (Linux)

OpenOCD Debugger—The open-source Open On-Chip Debugger (OpenOCD) software includes configuration files for many debug adapters, chips, and boards. Many versions of OpenOCD are available. The Efinix RISC-V flow requires a custom version of OpenOCD that includes the VexRiscv 32-bit RISC-V processor.

Version: 0.11.0 (20240413)

Disk space required: 17.4 MB (Windows), 16.3 MB (Linux)

Note: Efinix recommends you use the latest version of Efinity RISC-V Embedded Software IDE to ensure compatibility with Efinity software.

Required Hardware

- Trion® T120 BGA324 Development Board, Titanium Ti60 F225 Development Board, or Titanium Ti180 J484 Development Board
- 5 or 12 V power cable
- Micro-USB cable
- Computer or laptop
- (Optional) USB to UART converter module for the [2]
- Trion® T120 BGA324 Development Board[3]
- (Optional) FTDI mini-module or FTDI chip cable, C232HM-DDHSL-0, if you want to use the OpenOCD debugger and Efinity® Debugger simultaneously

Note: Some of the software examples provided with the SoC use a UART terminal to display messages. See **Set Up a USB-to-UART Module (Trion)** on page 105 and **Using the On-board UART (Titanium)** on page 104 for more information.

[2] The Titanium Ti60 F225 Development Board has an on-board USB-to-UART converter and does not require a separate module.

[3] The Titanium Ti60 F225 Development Board and Titanium Ti180 J484 Development Board have an on-board USB-to-UART converter and do not require a separate module.

Chapter 1

Install Software and SoC

Contents:

Install the Efinity Software

If you have not already done so, download the Efinity software from the Support Center and install it. For installation instructions, refer to the **Efinity Software Installation User Guide.**

Warning: Do not use spaces or non-English characters in the Efinity path.

Install the Efinity RISC-V Embedded Software IDE

Download the installer file in **Efinity RISC-V Embedded Software IDE <version>** from the Support Center.

To install the Efinity RISC-V Embedded Software IDE:

Windows

1. Execute the installer file **efinity-riscv-ide-<version>-windows-x64.exe** to launch the installer.
2. Follow the steps in the setup process.
3. Install Efinity RISC-V IDE in a preferred directory or use the default directory **c:\Efinity\efinity-riscv-ide-<version>\**. Example, **c:\Efinity\efinity-riscv-ide-2022.2.3\**.

Linux

1. Execute the installer file **efinity-riscv-ide-<version>-linux-x64.run** or run the installer using **./<installer run file>.** Run the executable script with command:

```
chmod +x <installer run file>
```

2. Select either to install the RISC-V IDE for the current user or multiple users.
3. Follow the steps in the setup wizard.
4. Install Efinity RISC-V IDE in a preferred directory or use the default directory **/home/user/efinity/efinity-riscv-ide-<version>**. Example, **/home/user/efinity/efinity-riscv-ide-2022.2.3/**.

Note:

- **Efinix provides FREE licences for the Efinity software.** Alternatively, when you buy a development kit, you also get a software license and one year of upgrades. After the first year, you can request a free maintenance renewal. The Efinity software is available for download from the Support Center. To get your free license, create an account, login, and then go to the Efinity page to request your license.
- Efinix recommends you use the latest version of Efinity RISC-V Embedded Software IDE to ensure compatibility with Efinity software.

Chapter 2

IP Manager

Contents:

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinix® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinix development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.

The IP Manager consists of:

- *IP Catalog*—Provides a catalog of IP cores you can select. Open the IP Catalog using the toolbar button or using **Tools > Open IP Catalog**.
- *IP Configuration*—Wizard to customize IP core parameters, select IP core deliverables, review the IP core settings, and generate the custom variation.
- *IP Editor*—Helps you manage IP, add IP, and import IP into your project.

Generating Sapphire SoC with the IP Manager

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose an IP core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.

Note: You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the IP core's user guide or on-line help.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinix® development board and/or testbench. For SoCs, you can also optionally generate embedded software example code. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.

Note: You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.

9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated RTL Files

The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.
- **<module name>_tmpl.v**—Verilog HDL instantiation template.
- **<module name>_tmpl.vhd**—VHDL instantiation template.
- **<module name>.v**—IP source code.
- **settings.json**—Configuration file.
- **<kit name>_devkit**—Has generated RTL, example design, and Efinity® project targeting a specific development board.
- **Testbench**—Contains generated RTL and testbench files.

Note: For encrypted IP, the ModelSim software version of 2022.4 or later is required for successful simulation. For other simulators, the latest version is required.

Note: Refer to the IP Manager chapter of the Efinity Software User Guide for more information about the Efinity IP Manager.

Generated Software Code

If you choose to output embedded software, the IP Manager saves it into the *<project>***/embedded_sw/***<SoC module>* directory.

- **bsp**—Board specific package.
- **config**—Has the Eclipse project settings file and OpenOCD debug configuration settings files for Windows.
- **config_linux**—Has the Eclipse project settings file and OpenOCD debug configuration settings files for Linux.
- **software**—Software examples.
- **tool**—Helper scripts.
- **cpu<n>.yaml**—CPU file for debugging where <n> is the core number, up to 4 cores.

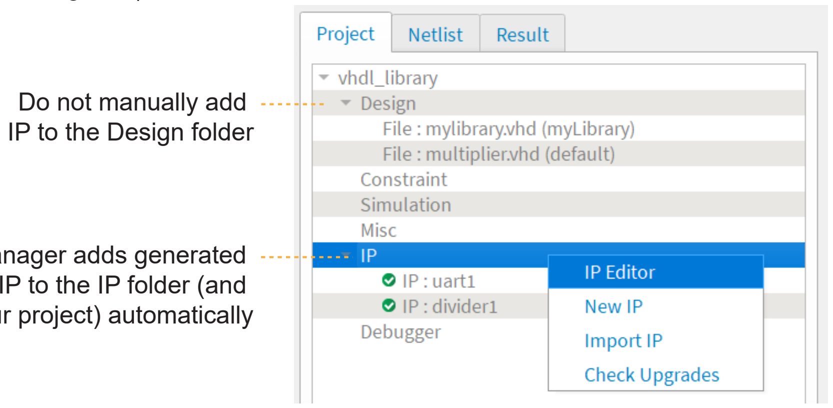
Instantiating the SoC

The IP Manager creates these template files in the *<project>***/ip/***<module name>* directory:

- *<module name>***.v_tmpl.v** is the Verilog HDL module.
- *<module name>***.v_tmpl.vhd** is the VHDL component declaration and instantiation template.

To use the IP, copy and paste the code from the template file into your design and update the signal names to instantiate the IP.

Important: When you generate the IP, the software automatically adds the module file (<*module name*>**.v**) to your project and lists it in the **IP** folder in the Project pane. Do not add the <*module name*>**.v** file manually (for example, by adding it using the Project Editor); otherwise the Efinity® software will issue errors during compilation.


Do not manually add IP to the Design folder
IP Manager adds generated IP to the IP folder (and your project) automatically
Project
Netlist
Result
vhdl_library
Design
File : mylibrary.vhd (myLibrary)
File : multiplier.vhd (default)
Constraint
Simulation
Misc
IP
IP : uart1
IP : divider1
Debugger
IP Editor
New IP
Import IP
Check Upgrades

Customizing the Sapphire SoC

There are two options available for the Sapphire SoC, which provides for different needs and applications:

- **Standard**—Best performance. This option utilizes more area of resources to achieve the best performance. Advanced features are only available in this option.
- **Lite**—Smallest area. This option utilizes a small area of resources by limiting the Sapphire SoC performance. Advanced features are not available in this option.

You customize the Sapphire SoC using the IP Configuration wizard. The parameters are arranged on tabs so you can click through them more easily.

There will be differences in the **SOC** and **Cache/Memory** tabs depending on the chosen option, either **Standard** or **Lite**, but all the other tabs are the same across both options.

Table 1: Sapphire SoC Tab Parameters

Parameter	Options	Description	Availability
Option	Standard, Lite	This option in the Sapphire SoC provides for different applications. Default: Standard	Standard and Lite
Core Number	1 - 4	Enter the number of CPU cores. Default: 1	Standard only
Frequency (MHz)	20 - 400	Enter the frequency in MHz. Default: 100	Standard and Lite
Peripheral Clock	On, off	Choose whether you want to run a dedicated clock for the APB3 slaves (SPI, I2C, GPIO, UART, and user timer) and AXI4 slave.	Standard and Lite
Peripheral Clock Frequency (MHz)	20 - 200	Enter the peripheral clock frequency in MHz.	Standard and Lite
Cache	On, off	Choose whether you want to include I$ and D$ caches.	Standard and Lite
Data Cache	On, off	Choose whether to include D$ cache. This parameter is only available when `Cache` parameter is turned on. You may choose to include I$ cache only or include both I$ and $D caches.	Lite only
Custom Instruction	On, off	Choose whether to enable the custom instruction interface.	Standard only
Linux Memory Management Unit	On, off	Choose whether to enable the Linux MMU.	Standard only
Floating-point Unit	On, off	Choose whether to enable the FPU.	Standard only
Atomic Extension	On, off	Choose whether to enable atomic extension instruction support. If you enable the Linux MMU, this option must be enabled and is turned on by default.	Standard only
Compressed Extension	On, off	Choose whether to enable compressed instruction support.	Standard only

Parameter	Options	Description	Availability
Multiplication and Division	On, off	Choose whether to enable multiplication and division, which is the RISC-V M extension. **Note:** This feature is turned on in Standard option.	Lite only
Barrel Shifter	On, off	Choose whether to include the barrel shifter, which is a module that can perform shift operations on any number of bits within a single clock cycle. **Note:** This feature is turned on in Standard option.	Lite only
CSR Optimization	On, off	Choose whether to minimize the number of RISC-V Control and Status Registers. **Note:** This feature is turned off when the RISC-V standard debug interface is enabled. This feature is also turned off in Standard option.	Lite only

Important: When running the SoC at high frequencies, Efinix recommends that you use the TIMING_1 place and route optimization. To set this option:

1. Open the Project Editor.
2. Click the **Place and Route** tab.
3. Double-click the **Value** cell for **--optimization_level**.
4. Choose **TIMING_1**.
5. Click **OK** and then compile.

Table 2: Sapphire Cache/Memory Tab Parameters

Parameter	Options	Description	Availability
Data Cache Way	1, 2, 4, 8	Choose the number of ways for the data cache. Default: 1	Standard and Lite
Cache Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB	Choose the size of the data cache. Default: 4 KB	Standard and Lite
Instruction Cache Way	1, 2, 4, 8	Choose the number of ways for the instruction cache. Default: 1	Standard and Lite
Cache Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB	Choose the size of the instruction cache. Default: 4 KB	Standard and Lite
External Memory Interface	On, off	On: By default. Instantiate the external memory interface. Off: Do not use the external memory interface.	Standard and Lite
AXI Interface Type	On, off	On: Use an AXI4 full duplex interface. Off: By default. Use an AXI3 half duplex interface.	Standard and Lite
AXI Interface Optimization	Optimize for area, Optimize for bandwidth	Optimize for area: Smaller area but lower bandwidth. Optimize for bandwidth: Full bandwidth but uses more resources.	Lite only
External Memory Clock Domain	Unified System Clock, Dedicated Memory Clock	Unified System Clock: The external memory interface will use the system clock (io_systemClk). This will utilize lesser resource as no CDC logic is required. **Note:** By sharing the system clock, the frequency of the system clock will be limited by the slowest domain in the system. Dedicated Memory Clock: The external memory interface will use the dedicated memory clock (io_memoryClk). This will utilize more resource.	Lite only
External Memory Data Width	32, 64, 128, 256, 512	Choose the data width for the AXI interface. Default: 128	Standard and Lite
External Memory AXI3 Address Size	4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, 0.5 GB, 1 GB, 1.5 GB, 2 GB, 2.5 GB, 3 GB, 3.5 GB	Choose the address size for the AXI interface. Default: 3.5 GB	Standard and Lite
On-Chip RAM Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 24 KB, 32 KB, 48 KB, 64 KB, 80 KB, 96 KB, 128 KB, 144 KB, 160 KB, 192 KB, 224 KB, 256 KB, 384 KB, 512 KB	Choose the size of the internal BRAM. Default: 4 KB	Standard and Lite

Parameter	Options	Description	Availability
Custom On-Chip RAM Application	On, off	On: Overwrite the default SPI flash bootloader with the user application. Off: By default. Use the default SPI flash bootloader.	Standard and Lite
User Application Path	–	Enter the path to your target user application. The file must be in **.hex** format.	Standard and Lite

Table 3: Sapphire Debug Tab Parameters

Parameter	Options	Description
Connection Type	Standalone, Chain	Choose whether you want to include the chain debug feature to the SoC. This allows the connection of multiple devices for JTAG debugging with a daisy-chain. Else, select as standalone. Standalone: By default. The debug feature is available for the standalone SoC only. Daisy-chain: The debug feature extends to multiple devices or SoC in the chain. Once enabled, you can debug multiple devices with a single debugger.
RISC-V Standard Debug	On, off	Choose whether to enable the RISC-V standard debug interface. On: Use the debug module that follows the ***RISC-V External Debug Support Version 0.13.*** **(Recommended)**[4][5] Off: Use debug module that is customized for the VexRiscv core.
Hardware Breakpoint	0 - 4	Number of hardware breakpoints. This hardware breakpoint is a program type breakpoint. Only applicable when the RISC-V Standard debug is turned on.
Additional Tap Devices (Max)	1 - 8	The maximum number of extra devices in the chain. This option is only applicable when you are using daisy-chain connection type. Default: 1
Soft Debug Tap	On, off	Choose whether you want to include a soft debug TAP for debugging. Off: By default. The SoC uses the JTAG User TAP interface block to communicate with the OpenOCD debugger. On: The SoC has a soft JTAG interface to communicate with the OpenOCD debugger. You need to use this setting if you want to use the soft JTAG interface instead of the JTAG User TAP.

(4) RISC-V standard debug is supported starting from Efinity 2023.1 or later. Debugging with RISC-V standard debug is only supported by Efinity RISC-V Embedded Software IDE version 2023.1 or later.
(5) The RISC-V standard debug requires connecting the hard `JTAG UPDATE` and `RESET` signals. Before Efinity 2023.1, these signals were unconnected. However, with Efinity 2023.1, the generated example designs automatically connect both signals.

Parameter	Options	Description
FPGA Tap Port	1, 2, 3, 4	Choose which Tap port you want to target with the OpenOCD debugger. This option is only applicable when you are using the JTAG User Tap interface block to communicate with the OpenOCD debugger.
Target Board/ Cable/Module	Trion T120 BGA324 Development Board Trion T120 BGA576 Development Board Trion T20 BGA256 Development Board Xyloni Titanium Ti60 F225 Development Board Titanium Ti180 J484 Development Board C232HM-DDHSL-0 (Soft debug) FTDI Module FT2232H (Soft debug) FTDI Module FT4232 (Soft debug) ISX-DLC_EF001 Programming Cable Custom	Choose which board you want to target with OpenOCD. Choose **Custom** to target your own board.
IDE Selection	Legacy Eclipse IDE (OpenOCD v0.10) Efinity RISC-V IDE (OpenOCD v0.11)	Choose which debug script format you want to generate. This selection allows you to roll back to target the Legacy Eclipse IDE. By default, Efinity RISC-V Embedded Software IDE is targeted.
Custom Target Board	-	Enter the name of your board.
Application Region Size	124KB, 252KB, 508KB, 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB, 256MB	Modify the linker script to outline the region for the user application. This option is only applicable for SoCs with external memory. For SoCs with internal memory, the region size is determined by the on-chip RAM size.
Application Stack Size	1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB, 1MB, 2MB, 4MB, 8MB, 16MB, 32MB, 64MB, 128MB	Modify the linker script to specify the application stack size. This option is only applicable for SoCs with external memory. For SoCs with internal memory, the region size is automatically set to 1/8 of the on-chip RAM size.
OpenOCD Debug Mode	Turn on by default Turn off by default	Choose whether you want software applications to run in debug mode by default or not. See **Debug with the OpenOCD Debugger** on page 51 for more details.

Table 4: Sapphire UART Tab Parameters

Where *n* is 0, 1, or 2

Parameter	Options	Description
UART *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
UART *n* Interrupt ID	1 - 36	Choose the interrupt ID for the UART. The IDs default to: UART 0: 1 UART 1: 2 UART 2: 3

Table 5: Sapphire SPI Tab Parameters

Where *n* is 0, 1, or 2.

Parameter	Options	Description
SPI *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
SPI *n* Interrupt ID	1 - 36	Choose the interrupt ID for the SPI. The IDs default to: SPI 0: 4 SPI 1: 5 SPI 2: 6
SPI *n* Data Width	8 - 16	Configure the data width for the SPI interface. **Note:** Only applicable for SPI 1 and SPI 2.
SPI *n* Chip Select Width	1 - 8	Choose the number of Chip select required for the SPI interface. **Note:** Only applicable for SP1 and SP2.

Table 6: Sapphire I2C Tab Parameters

Where *n* is 0, 1, or 2.

Parameter	Options	Description
I2C *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
I2C *n* Interrupt ID	1 - 36	Choose the interrupt ID for the I^{2}C. The IDs default to: I2C 0: 8 I2C 1: 9 I2C 2: 10

Table 7: Sapphire GPIO Tab Parameters

Where *n* is 0 or 1.

Parameter	Options	Description
GPIO *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
GPIO *n* Bit Width	1, 2, 4, 8, 16, 32	Choose the number of pins for the GPIO. Default: 4 (GPIO 0), 8 (GPIO 1)
GPIO *n* Interrupt ID 0	1 - 36	Choose the interrupt ID for the GPIO. The IDs default to: GPIO 0: 12 GPIO 1: 14
GPIO *n* Interrupt ID 1	1 - 36	Choose the interrupt ID for the GPIO. The IDs default to: GPIO 0: 13 GPIO 1: 15

Table 8: Sapphire APB3 Tab Parameters

Where *n* is 0, 1, 2, 3, or 4.

Parameter	Options	Description
APB Slave Address Size	4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB, 1MB	Choose the APB slave size. This setting applies to all APB slaves. Default: 64KB
APB3 Slave *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.

Table 9: Sapphire AXI4 Tab Parameters

Where *n* is 0 or 1.

Parameter	Options	Description
AXI Slave	On, off	On: Instantiate the interface. Off: Do not use the interface.
AXI Slave Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB	Choose the size of the AXI slave.
AXI Master *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
AXI Master *n* Data Width	32, 64, 128, 256, 512	Choose the width of the AXI master. Do not specify an AXI master width that is larger than the external memory data width.

Table 10: Sapphire User Interrupt Tab Parameters

Where *n* is A, B, C, D, E, F, G, or H.

Parameter	Options	Description
User *n* Interrupt	On, off	On: Instantiate the interface. Off: Do not use the interface.
User *n* Interrupt ID	1 - 36	Choose the interrupt ID. The defaults are: User A Interrupt: 16 User B Interrupt: 17 User C Interrupt: 22 User D Interrupt: 23 User E Interrupt: 24 User F Interrupt: 25 User G Interrupt: 26 User H Interrupt: 27

Table 11: Sapphire User Timer Tab Parameters

Where *n* is 0, 1, or 2.

Parameter	Options	Description
User Timer *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
User Timer *n* Counter Width	12, 16, 32	Choose the counter bit width. Default: 12
User Timer *n* Prescaler Width	8, 16	Choose the prescaler bit width. Default: 8
User Timer *n* Interrupt ID	1 - 36	Choose the interrupt ID. The defaults are: User Timer 0: 19 User Timer 1: 20 User Timer 2: 21

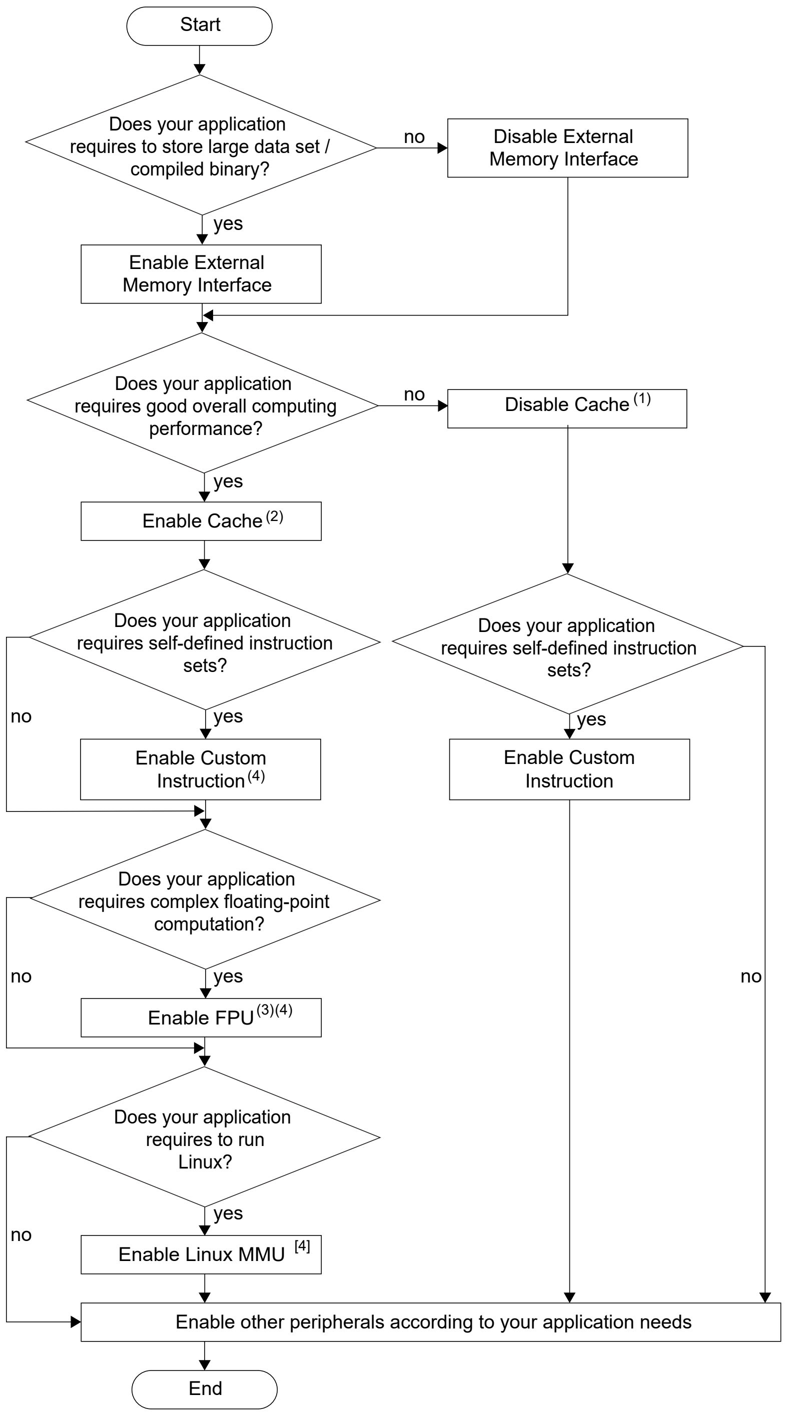
Table 12: Sapphire Base Address Tab Parameters

Parameter	Options	Description
Address Assignment Method	AUTO, MANUAL	AUTO: Automatically assign an address to the enabled peripherals. MANUAL: The user can assign addresses to the enabled peripherals.
External Memory Base Address	–	Displays the base address. You cannot change it.
AXI Slave Base Address	–	Displays the base address when the **Address Assignment Method** is set to AUTO. When the **Address Assignment Method** is **Manual**, enter the base address value. The wizard automatically rounds the value to 16 MB aligned addresses during IP generation. For example, 0x41234567 is rounded to 0x41000000.
Peripheral and IO Base Address	–	
UART*n* Address Offset	–	Displays the base address when the **Address Assignment Method** is set to AUTO. When the **Address Assignment Method** is **Manual**, enter base address value. The wizard automatically rounds the value to 4 KB aligned addresses during IP generation. For example, 0x41230 is rounded to 0x41000.
SPI*n* Address Offset	–	
I2C*n* Address Offset	–	
GPIO*n* Address Offset	–	
User Timer*n* Address Offset	–	
APB3 Slave *n* Address Offset	–	Displays the base address when the **Address Assignment Method** is set to AUTO. When the **Address Assignment Method** is **Manual**, enter base address value. The wizard automatically rounds the value to APB sized aligned addresses during IP generation. For example, if the APB size is 64 KB, 0x23456 is rounded to 0x20000.
On-Chip RAM Base Address	–	Displays the base address. You cannot change it.

SoC Configuration Guideline

Sapphire SoC is highly adaptive to different use cases. It is configurable to get the best balance between performance and resources. The following flow chart is a simple guideline to help you choose the configuration that suits your needs.

Figure 3: Sapphire Soc Configuration Guideline


Start
Does your application requires to store large data set / compiled binary?
no
Disable External Memory Interface
yes
Enable External Memory Interface
Does your application requires good overall computing performance?
no
Disable Cache (1)
yes
Enable Cache (2)
Does your application requires self-defined instruction sets?
no
yes
Enable Custom Instruction (4)
Does your application requires self-defined instruction sets?
yes
Enable Custom Instruction
no
Does your application requires complex floating-point computation?
no
yes
Enable FPU (3)(4)
Does your application requires to run Linux?
no
yes
Enable Linux MMU [4]
Enable other peripherals according to your application needs
End

Notes :

1. Using the SoC with an external memory interface but without a cache sharply impacts the SoC's overall performance.
2. Enabling the cache controller increases the efficiency of processing the instructions but consumes more RAM blocks.
3. The SoC calculates faster and more efficiently in floating-point computing if FPU is enabled, but it consumes more FPGA LUTs and RAM blocks.
4. When enabling custom instructions, the FPU and the Linux impact the f_{MAX} performance.

Modify the Bootloader

When you generate the Sapphire SoC, the IP Manager creates a pre-built bootloader **.bin** to target the on-chip RAM size you selected. If you assigned the peripheral addresses manually, you need to create a custom bootloader according to the following instructions.

Learn more: You need the embedded software example code to make these changes; if you have not already done so, generate it.

Note: The pre-build bootloader binaries only use a single data line SPI. To utilize dual or quad data line SPI, refer to **Modify the Bootloader Software to Enable Multi-Data Lines** on page 26.

Modify the Bootloader Software to Extend the External Memory Size

First you need to modify the bootloader code:

1. Open the **bootloaderConfig.h** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/EfxSapphireSoc/app** directory.
2. Change the `#define USER_SOFTWARE_SIZE` parameter for the new on-chip RAM size and save.
3. If you are using the MX25 flash device (e.g., Ti180J484 development kit), incorporate the following step into the bootloader application's makefile. Add **CFLAGS+=-DMX25_FLASH** before the line **LDSCRIPT?=${BSP_PATH}linker/bootloader.ld**

   Note: The addition of **CFLAGS+=-DMX25_FLASH** ensures that the necessary commands specific to the MX25 flash device are included in your build process.

4. In Efinity RISC-V Embedded Software IDE, import **standalone/bootloader** project. Build the project to generate new **bootloader.hex** file.

Second, you update and re-generate the SoC in the IP Manager to point to your new **bootloader.hex** and change the application region size. The default maximum size is 124 KB.

1. In the Sapphire IP wizard, go to the **Cache/Memory** tab.
2. Turn on the **Custom On-Chip RAM Application** option.
3. Click the **Browse** button for the to select the new **bootloader.hex** you created in the previous set of steps.
4. Generate the SoC.

Modify the Bootloader Software without External Memory Enabled

First, you need to modify the bootloader linker script:

1. Open the **bootloader.ld** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/EfxSapphireSoc/linker** directory.
2. Replace the `MEMORY` and `PHDRS` code with the following code. The <*bootloader_address*> should be 0xF9000000 + (<*memory size*>-1024), where <*memory size*> is your SoC's on-chip RAM size.

```
MEMORY
{
    start (wxai!r) : ORIGIN = 0xF9000000, LENGTH = 512
    ram   (wxai!r) : ORIGIN = <bootloader_address>, LENGTH = 1024
}

PHDRS
{
    start PT_LOAD;
    ram PT_LOAD;
}
```

Second you need to modify the bootloader code:

1. Open the **bootloaderConfig.h** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/EfxSapphireSoc/app** directory.
2. Change the `#define USER_SOFTWARE_SIZE` parameter for the new on-chip RAM size and save.
3. If you are using the MX25 flash device (e.g., Ti180J484 development kit), incorporate the following step into the bootloader application's makefile. Add **CFLAGS+=-DMX25_FLASH** before the line **LDSCRIPT?=${BSP_PATH}linker/bootloader.ld**

Note: The addition of **CFLAGS+=-DMX25_FLASH** ensures that the necessary commands specific to the MX25 flash device are included in your build process.

Note: If the new compiled bootloader does not fit into the allocated RAM, enable the following optimization in the makefile; `DEBUG?=no`, `BENCH?=yes`. Refer to Optimization Settings on page 40.

In Efinity RISC-V Embedded Software IDE, import **standalone/bootloader** project. Build the project to generate new **bootloader.hex** file.

Modify the Bootloader Software to Enable Multi-Data Lines

Before you can utilize the multi-data lines SPI in your bootloader, verify whether your board's flash drive supports Dual or Quad I/O modes.

In the Efinity RISC-V Embedded Software IDE example design, data ports 0 and 1 are exclusively connected. If you intend to use the Quad SPI for data transfer, you must establish connections for data ports 2 and 3. The following table shows the number of connected data lines interfacing with the respective FPGAs and flash devices.

Table 13: Multi-Data Lines Interface with FPGAs and Flash Devices

Development Kit	Flash Device	Number of Data Lines Connected
T8BGA81	W25Q80DLSNIG	2
T20BGA256	W25Q32JVSSIQ	2
T120BGA324	W25Q128JVSIQ	4
T120BGA576	W25Q128JVSIQ	4
Xyloni	W25Q128JVSIM	2
Ti60F225	W25Q64JWSSIQ	2
Ti180J484/ Ti180M484	MX25U25645GZ4I00	4

In the **bootloaderConfig.h** file, you can define the configurations by selecting from the various data line modes:

- SINGLE_SPI: Single data line
- DUAL_SPI: Dual data line
- QUAD_SPI: Quad data line

Note: If the flash device is MX25 (from Ti180J484 development kit), add **CFLAGS+=-DMX25_FLASH** before the **LDSCRIPT?=${BSP_PATH}linker/bootloader.ld** into the bootloader application's makefile. Defining the MX25 includes the required commands specific to the MX25 flash device.

Updating Bootloader with Efinity BRAM Initial Content Updater

The Efinity BRAM Initial Content Updater provides a convenient way to modify the default firmware (either bootloader or other application) within the Sapphire SoC on-chip RAM. This process enables you to update the on-chip RAM initial content without recompiling the entire project.

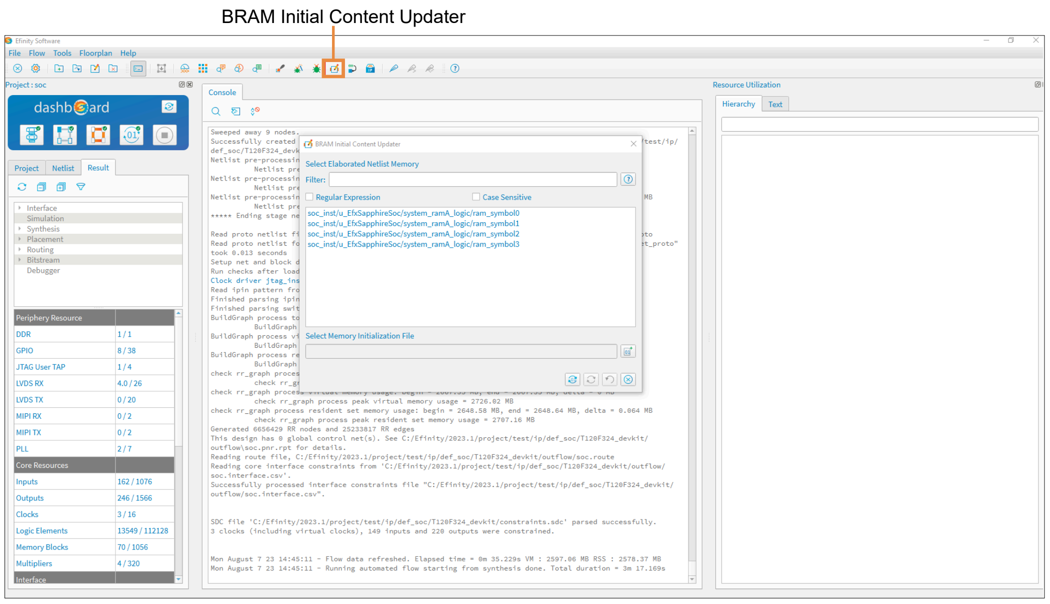
To update the on-chip RAM initial content, follow these steps:

1. *Compile and locate the **.hex** file:* Compile your new application in Efinity RISC-V Embedded Software IDE and locate the corresponding **.hex** file that contains the compiled code.
2. *Generate the Sapphire SoC with the new application:* By using the Sapphire SoC IP Configurator, update the default on-chip RAM application with your new application compiled in the provious step. You may refer to the Modify the Bootloader Software to Extend the External Memory Size on page 24 on how to use the **Custom On-Chip RAM Application** in the Sapphire SoC IP Configurator. and you are now ready for the updating process.

Note: You may opt to generate the binaries with the **binGen.py** helper script provided manually. Refer to Appendix: Re-Generate the Memory Initialization Files Manually on page 189.

3. *Locate the new binaries for your application:* After the Sapphire SoC is generated with your application, locate the new binaries, **EfxSapphireSoc.v_toplevel_system_ramA_logic_ram_symbol<n>.bin** where <n> is the range from 0 to 3 (up to 7 if FPU extension is enabled).
4. *Open the BRAM Initial Content Updater:* Click at the **BRAM Initial Content Updater** tab to launch the **BRAM Initial Content Updater**.

Figure 4: Open the BRAM Initial Content Updater



Periphery Resource	
DDR	1 / 1
GPIO	8 / 38
JTAG User TAP	1 / 4
LVDS RX	4.0 / 26
LVDS TX	0 / 20
MIPI RX	0 / 2
MIPI TX	0 / 2
PLL	2 / 7
Core Resources	
Inputs	162 / 1076
Outputs	246 / 1566
Clocks	3 / 16
Logic Elements	13549 / 112128
Memory Blocks	70 / 1056
Multipliers	4 / 320
Interface	

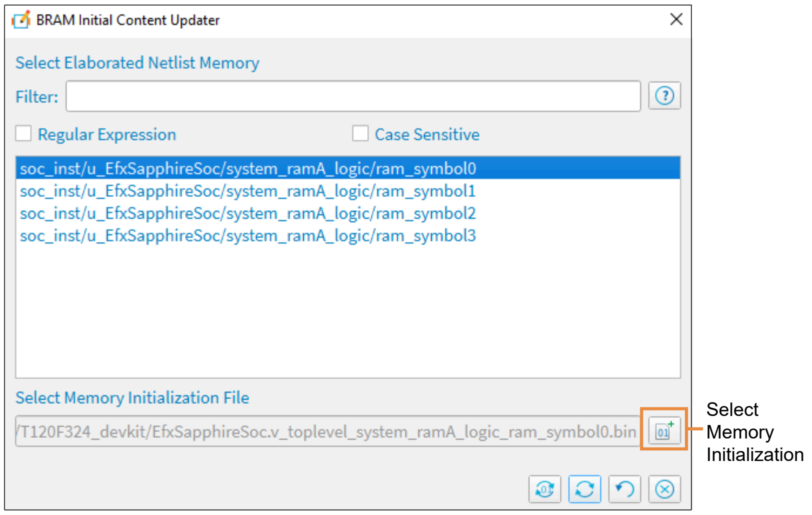
```
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol0
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol1
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol2
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol3
```

```
Sweeped away 9 nodes.
Successfully created                                                     /test/ip/
def_soc/T120F324_devk
Netlist pre-processin
         Netlist pre
Netlist pre-processin
         Netlist pre
Netlist pre-processin                                                    MB
         Netlist pre
***** Ending stage ne

Read proto netlist fi                                                    oto
Read proto netlist fo                                                    et_proto"
took 0.013 seconds
Setup net and block d
Run checks after load
Clock driver jtag_ins
Read ipin pattern fro
Finished parsing ipin
Finished parsing swit
BuildGraph process to
         BuildGraph
BuildGraph process vi
         BuildGraph
BuildGraph process re
         BuildGraph
check rr_graph proces
         check rr_gr
check rr_graph proces
         check rr_graph process peak virtual memory usage = 2726.02 MB
check rr_graph process resident set memory usage: begin = 2648.58 MB, end = 2648.64 MB, delta = 0.064 MB
         check rr_graph process peak resident set memory usage = 2707.16 MB
Generated 6656429 RR nodes and 25233817 RR edges
This design has 0 global control net(s). See C:/Efinity/2023.1/project/test/ip/def_soc/T120F324_devkit/
outflow\soc.pnr.rpt for details.
Reading route file, C:/Efinity/2023.1/project/test/ip/def_soc/T120F324_devkit/outflow/soc.route
Reading core interface constraints from 'C:/Efinity/2023.1/project/test/ip/def_soc/T120F324_devkit/outflow/
soc.interface.csv'.
Successfully processed interface constraints file "C:/Efinity/2023.1/project/test/ip/def_soc/T120F324_devkit/
outflow/soc.interface.csv".


SDC file 'C:/Efinity/2023.1/project/test/ip/def_soc/T120F324_devkit/constraints.sdc' parsed successfully.
3 clocks (including virtual clocks), 149 inputs and 220 outputs were constrained.


Mon August 7 23 14:45:11 - Flow data refreshed. Elapsed time = 0m 35.229s VM : 2597.06 MB RSS : 2578.37 MB
Mon August 7 23 14:45:11 - Running automated flow starting from synthesis done. Total duration = 3m 17.169s
```

5. *Select Memory Initialization File:* In the **BRAM Initial Content Updater** window, locate the Sapphire SoC BRAM that you would like to update and click on the ***_symbol0**. In the **Select Memory Initialization File** section, click the **Select Memory Initialization** tab. Browse to the updated

   EfxSapphireSoc.v_toplevel_system_ramA_logic_ram_symbol<n>.bin and click **Open**.

Figure 5: Select Memory Initialization File


BRAM Initial Content Updater

Select Elaborated Netlist Memory

Filter:

Regular Expression   Case Sensitive

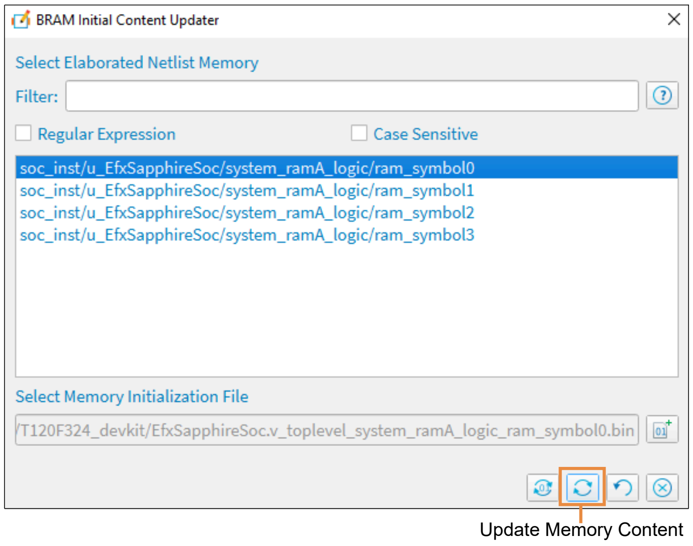
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol0
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol1
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol2
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol3

Select Memory Initialization File

/T120F324_devkit/EfxSapphireSoc.v_toplevel_system_ramA_logic_ram_symbol0.bin

6. *Update the BRAM:* Click on the **Update Memory Content** tab to update the **symbol0** BRAM with the new application **symbol0** binary.

Figure 6: Update the BRAM with New Application


BRAM Initial Content Updater

Select Elaborated Netlist Memory

Filter:

Regular Expression   Case Sensitive

soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol0
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol1
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol2
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol3

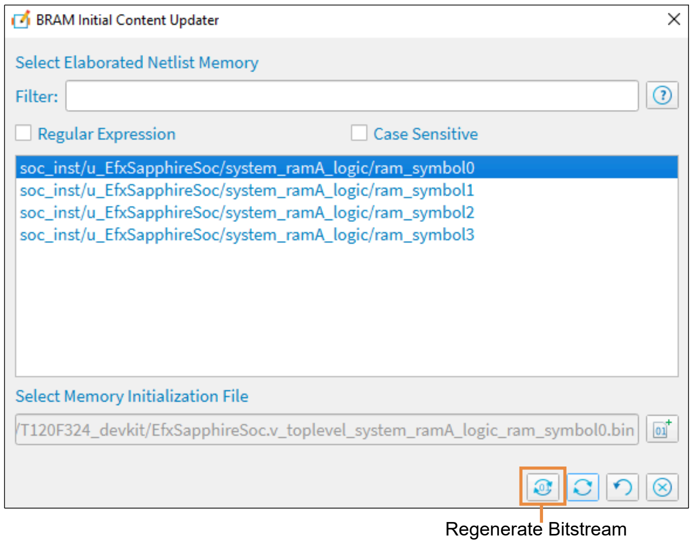
Select Memory Initialization File

/T120F324_devkit/EfxSapphireSoc.v_toplevel_system_ramA_logic_ram_symbol0.bin

7. *Update the remaining BRAM:* Repeat step 5 and 6 for all the available symbol files. Update the BRAM with the corresponding binary. For example, update ***_ram_symbol2** BRAM with **EfxSapphireSoc.v_toplevel_system_ramA_logic_ram_symbol2.bin** binary file.

8. *Generate the new bitstream:* Click on the **Regenerate Bitstream** icon to regenerate the bitstream. The generated new bitstreams are located in the outflow folder.

Figure 7: Generate the New Bitstream


BRAM Initial Content Updater

Select Elaborated Netlist Memory

Filter:

Regular Expression

Case Sensitive

soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol0
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol1
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol2
soc_inst/u_EfxSapphireSoc/system_ramA_logic/ram_symbol3

Select Memory Initialization File

/T120F324_devkit/EfxSapphireSoc.v_toplevel_system_ramA_logic_ram_symbol0.bin

Note: For more information on the Efinity BRAM Initial Content Updater and its application, see Efinity Software User Guide.

Chapter 3

Program the Board with the Sapphire RTL Design

Contents:

Before working with software code, Efinix recommends that you program your board with an RTL design that instantiates the Sapphire SoC. When you generate the Sapphire SoC with the IP Manager, you can optionally generate an example Efinity® project and bitstream file to get you started quickly.

About the Example Design

This example targets Trion and Titanium development boards:

- *Trion® T120 BGA324 Development Board*—The RTL design files are in the **T120F324_devkit** directory.
- *Titanium Ti60 F225 Development Board*—The RTL design files are in the **Ti60F225_devkit** directory.
- *Titanium Ti180 J484 Development Board*—The RTL design files are in the **Ti180J484_devkit** directory.

When you generate the IP core, the IP Manager creates the example design (PLL settings, SDC timing constraints, and I/O assignments) using the settings you chose in the wizard, with a few exceptions:

- For the Trion board, the example design only supports external memory widths of 128 and 256 bits because the DDR controller only supports these widths. Therefore, do not choose 32 or 64 bits for the external memory.
- The example design automatically connects UART0, SPI0, I2C0, GPIO0, the soft TAP pins, and the PLL source clock pins to top-level ports, and it assigns I/O pins to them (if they are enabled). If you add more of these peripherals, you need to connect them manually and create the I/O assignments for them.
- The example design uses PLL settings that look for the best effort multiplier and divider values.

Note: The following description is for the example design using the default settings.

This example writes to and reads from the development board's memory module using the AXI interface:

- For the Trion® T120 BGA324 Development Board, the design uses the board's LPDDR3 DRAM module.
- For the Titanium Ti60 F225 Development Board, the design uses the board's HyperRAM module.

- For the Titanium Ti180 J484 Development Board, the design uses the board's LPDDR4/ LPDDR4x DRAM module.

The Sapphire SoC is configured for:
- 100 MHz frequency
- External memory interface is enabled with a width of 128 and size of 3.5 GB
- Caches are enabled with both Data Cache and Instruction Cache set to one way with cachesize of 4 KB
- 4KB on-chip RAM size
- Soft Debug Tap is disabled
- UART 0 is enabled
- SPI 0 is enabled
- I2C 0 is enabled
- GPIO 0 is enabled
- APB3 0 is enabled
- AXI4 Slave is enabled
- AXI Master 0 is enabled
- User interrupt A is enabled

Figure 8: Example Design Block Diagram

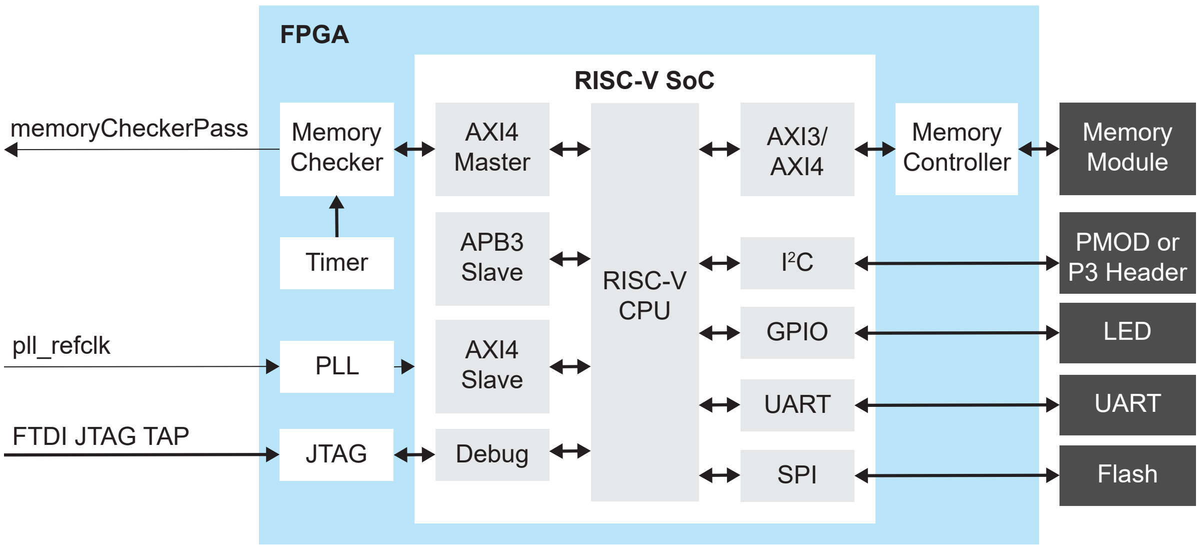


Table 14: Example Design Implementation

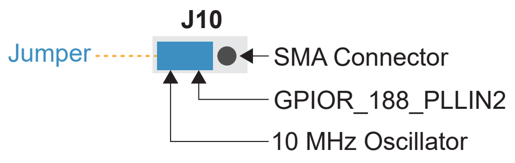
FPGA	Logic + Adders	Flipflops	Multipliers or DSP Blocks	Memory Blocks	f_{MAX} (MHz)	Language	Efinity Version
T120 BGA324 C4	8,830	8,919	4	70	107	Verilog HDL	2023.1
Ti60 F225 C4	11,178	9,973	4	82	180	Verilog HDL	2023.1
Ti180 J484 C4	12,213	15,866	4	100	146	Verilog HDL	2023.1

Note: All example designs are constrained with a 100 MHz system clock.

Enable the On-Board 10 MHz Oscillator (T120 BGA324 Board)

For the Trion® T120 BGA324 Development Board, the SoC design uses the on-board 10 MHz oscillator. To enable it, add a jumper to connect pins 2 and 3 on header J10.

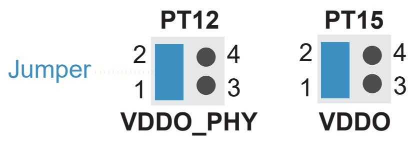
Figure 9: Connect Pins 2 and 3 on J10



Enable the LPDDR4x Memory (Ti180 J484 Board)

For the Titanium Ti180 J484 Development Board, the SoC design uses LPDDR4x settings to drive the external memory. To enable it, change the jumpers on PT12 and PT15 to connect pins 1 and 2 to provide 0.6 V to VDDQ and VDDQ_PHY.

Figure 10: Connect Pins 1 and 2 on PT12 and PT15



Installing USB Drivers

To program Trion® FPGAs using the Efinity® software and programming cables, you need to install drivers.

Efinix development boards have FTDI chips (FT232H, FT2232H, or FT4232H) to communicate with the USB port and other interfaces such as SPI, JTAG, or UART. Refer to the Efinix development kit user guide for details on installing drivers for the development board.

Note: If you are using more than one Efinix development board, you must manage drivers accordingly. Refer to AN 050: Managing Windows Drivers for more information.

Installing Drivers on Windows

On Windows, you use software from Zadig to install drivers. Download the Zadig software (version 2.7 or later) from zadig.akeo.ie. (You do not need to install it; simply run the downloaded executable.)

Install the driver for the interfaces listed in the following table.

Board	Interface to Install Driver
Trion® T120 BGA324 Development Board	Install drivers for all interfaces (0 and 1).
Titanium Ti60 F225 Development Board	Install drivers for interfaces 0 and 1 only. Windows automatically installs a driver for interfaces 2 and 3 when you connect the board to your computer.
Titanium Ti180 J484 Development Board	Install driver for interface 1 only.

To install the driver:

1. Connect the board to your computer with the appropriate cable and power it up.
2. Run the Zadig software.

   Note: To ensure that the USB driver is persistent across user sessions, run the Zadig software as administrator.

3. Choose **Options** > **List All Devices**.
4. Repeat the following steps for each interface. The interface names end with *(Interface N)*, where *N* is the channel number.
   - Select **libusb-win32** in the **Driver** drop-down list.
   - Click **Replace Driver**.
5. Close the Zadig software.

Note: This section describes the instruction to install the libusb-win32 driver for each interface separately. If you have previously installed a composite driver or installed using libusbK drivers, you do not need to update or reinstall the driver. They should continue to work correctly.

Installing Drivers on Linux

The following instructions explain how to install a USB driver for Linux operating systems.

1. Disconnect your board from your computer.
2. In a terminal, use these commands:

```
> sudo <installation directory>/bin/install_usb_driver.sh
> sudo udevadm control --reload-rules
```

Note: If your board was connected to your computer before you executed these commands, you need to disconnect and re-connect it.

Program the Development Board

When you generate the Sapphire SoC in the IP Manager, you can optionally generate an example design targeting an Efinix development board. Example designs include a bitstream file, **soc.hex**, so you can get started quickly without having to compile the design.

Table 15: Available Example Designs

Board	Location
Titanium Ti60 F225 Development Board	Ti60F225_devkit
Titanium Ti180 J484 Development Board	Ti180J484_devkit
Trion® T120 BGA324 Development Board	T120F324_devkit

Download the **.hex** file to the board using these steps:

Connect the board to your computer using a USB cable.

Learn more: Instructions on how to use the Efinity software and board documentation are available in the Support Center.

Chapter 4

Simulate

The IP Manager automatically generates a testbench and top-level file for simulation based on the settings you made in the wizard, including the top-level file generation, I/O connection to the testbench, simulation models, and stimulus such as clock and reset. The testbench bypasses the SPI flash data retrieval step to speed up simulation.

Note: If you manually assign addresses to the peripherals, the default simulation may not function correctly.

1. Open a terminal.
2. Change to the **Testbench** directory for your SoC.
3. Set up the Efinity environment:
   - Linux: `source /<path to Efinity>/bin/setup.sh`
   - Windows: `c:\<path to Efinity>\bin\setup.bat`
4. Run the simulation using the default application with the command `Python3 run.py`.

   Note: The default application requires **UART 0** to be turned on.

   Note: If you want to include the SPI flash retrieval step (requires **SPI 0** to be turned on), run the simulation with the command:

   ```
   Python3 run.py -f
   ```

A successful simulation returns the following messages

```
#     0 -----------------------------------------
#     0 [EFX_INFO]: Start executing helloWorld TEST
#     0 -----------------------------------------
#     51315 -----------------------------------------
#     51315 [EFX_INFO]: Receiving uart data from soc
#     51315 -----------------------------------------
#     2121065 -----------------------------------------
#     2121065 [EFX_INFO]: TEST PASSED
#     2121065 [EFX_INFO]: Hello World from Efinix!
#     2121065 -----------------------------------------
```

To simulate with a different application instead of the default, use the command:

```
Python3 run.py -b <path to application>/app.bin
```

When you use a non-default application, the testbench bypasses the default driver and monitor sequences and displays warning messages.

```
#   0 -----------------------------------------
#   0 [EFX_INFO]: Executing custom binary file...
#   0 [EFX_WARN]: Skipped testbench default driver and monitor sequences.
#   0 [EFX_INFO]: Running simulation...

#   0 -----------------------------------------
```

You need to develop your own sequence for your application.

The default simulator is Modelsim or Questasim. To run the simulation with Aldec Riviera simulator, look for `ALDEC_PATH` on the top part of `run.py`, uncomment the line, and set the path to your Aldec Riviera simulator installation path.

```
#ALDEC_PATH=Path('PUT', 'YOUR', 'OWN', 'ALDEC', 'PATH')
```

Then run the simulation command with “tool” argument:

```
Python3 run.py --tool aldec
```

Chapter 5

Launch Efinity RISC-V Embedded Software IDE

Contents:

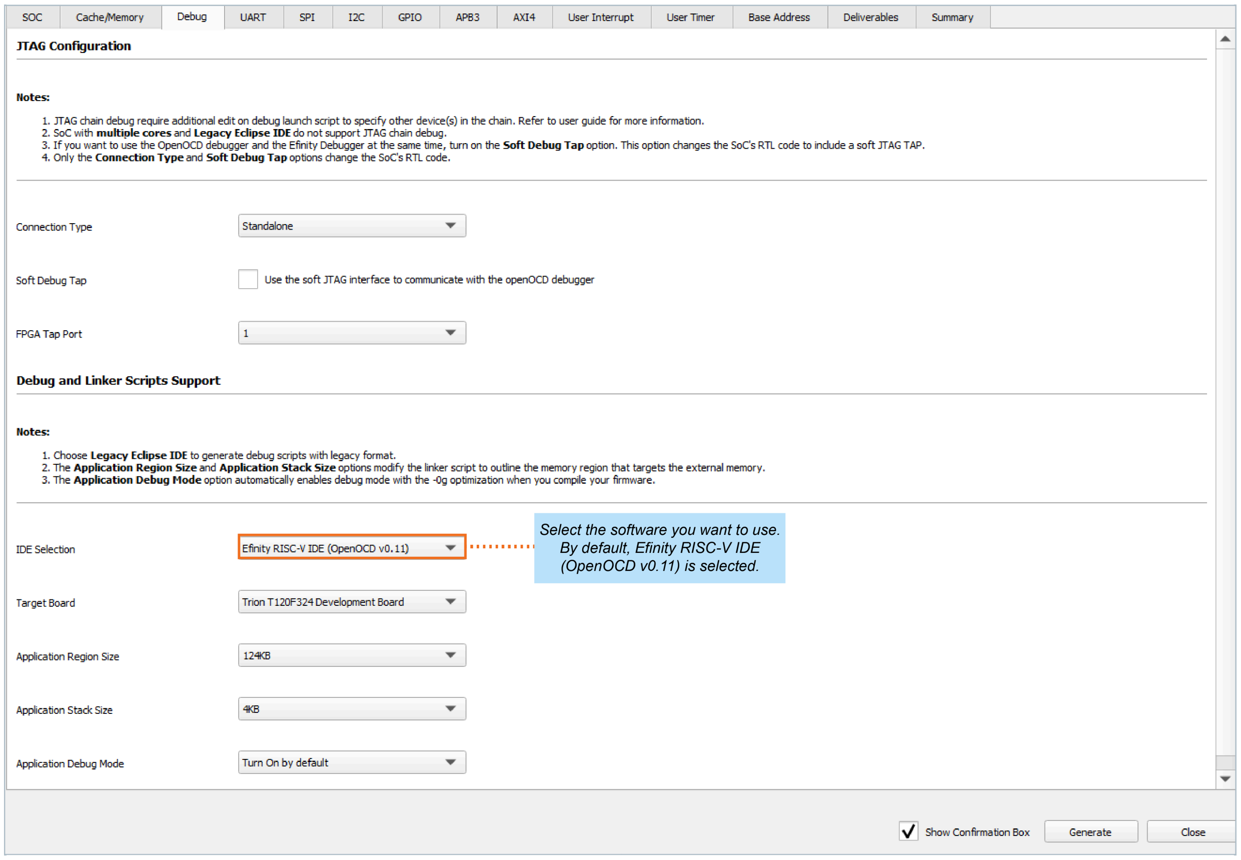
Sapphire SoC IDE Backward Compatibility

The Efinity software v2022.2 and higher includes the Efinity RISC-V Embedded software IDE for developing RISC-V software applications. Previously, you developed software using the open-source RISC-V SDK. The IDE provides an enhanced environment for developing embedded applications, and Efinix recommends that all users switch to it. The IDE has these features and advantages:

- Optimized process for importing projects
  - Eliminates redundant steps to import C/C++ Project Settings
  - Simplifies the sample projects import process with tick boxes
  - Automatically loads the correct C/C++ project settings for both standalone and FreeRTOS sample programs
- Ability to automate makefile generation for new project creation
- Integrated QEMU emulator for 32-bit RISC-V Core
  - Bundled with project examples
  - Allows you to debug without hardware
- Flexible workspace directory
  - RISC-V IDE allows you to point to your project's BSP and FreeRTOS Kernel
  - Eliminates the need to copy the FreeRTOS Kernel folder to each project's directory, ***embedded_sw/<SoC module>/software***
- Easier debug experience
  - CSR Register View
  - Peripheral Register View
  - FreeRTOS Queue and Task List View

When you configure your SoC in the Efinity IP Manager, the IDE Selection parameter is provided in the Debug tab. If you intend to use the open-source Eclipse software in the RISC-V SDK, select the **Legacy Eclipse IDE (OpenOCD v0.10)** option. By default, **Efinity RISC-V IDE (OpenOCD v0.11)** is selected.

Figure 11: IDE Selection Parameter in the Debug tab



JTAG Configuration

Notes:

1. JTAG chain debug require additional edit on debug launch script to specify other device(s) in the chain. Refer to user guide for more information.
2. SoC with **multiple cores** and **Legacy Eclipse IDE** do not support JTAG chain debug.
3. If you want to use the OpenOCD debugger and the Efinity Debugger at the same time, turn on the **Soft Debug Tap** option. This option changes the SoC's RTL code to include a soft JTAG TAP.
4. Only the **Connection Type** and **Soft Debug Tap** options change the SoC's RTL code.

Connection Type: Standalone

Soft Debug Tap: Use the soft JTAG interface to communicate with the openOCD debugger

FPGA Tap Port: 1

Debug and Linker Scripts Support

Notes:

1. Choose **Legacy Eclipse IDE** to generate debug scripts with legacy format.
2. The **Application Region Size** and **Application Stack Size** options modify the linker script to outline the memory region that targets the external memory.
3. The **Application Debug Mode** option automatically enables debug mode with the -0g optimization when you compile your firmware.

IDE Selection: Efinity RISC-V IDE (OpenOCD v0.11)

Select the software you want to use. By default, Efinity RISC-V IDE (OpenOCD v0.11) is selected.

Target Board: Trion T120F324 Development Board

Application Region Size: 124KB

Application Stack Size: 4KB

Application Debug Mode: Turn On by default

Launching the Efinity RISC-V Embedded Software IDE

Windows

Launch the Efinity RISC-V Embedded Software IDE by double-clicking on the Efinity RISC-V IDE shortcut available in the **efinity-riscv-ide-<version>** folder (example: efinity-riscv-ide-2022.2.3). For easy access, you may transfer the shortcut to the desktop. A new IDE window opens once the IDE is successfully invoked.

You need to select a workspace directory to store the IDE's preferences, configurations and temporary information. Follow these steps:

1. Click **Browse** and select your preffered location.
2. You may click the **Recent Workspaces** to select a previous workspace.
3. Click **Launch**.

Linux

Launch the Efinity RISC-V Embedded Software IDE in a Linux environment by double-clicking the **efinity-riscv-ide**. Alternatively, you may launch the **efinity-riscv-ide** in the terminal. A new IDE window opens once the IDE is successfully invoked.

You need to select a workspace directory to store the IDE's preferences, configurations and temporary information. Follow these steps:

1. Click **Browse** and select your preffered location.
2. You may click the **Recent Workspaces** to select a previous workspace.
3. Click **Launch**.

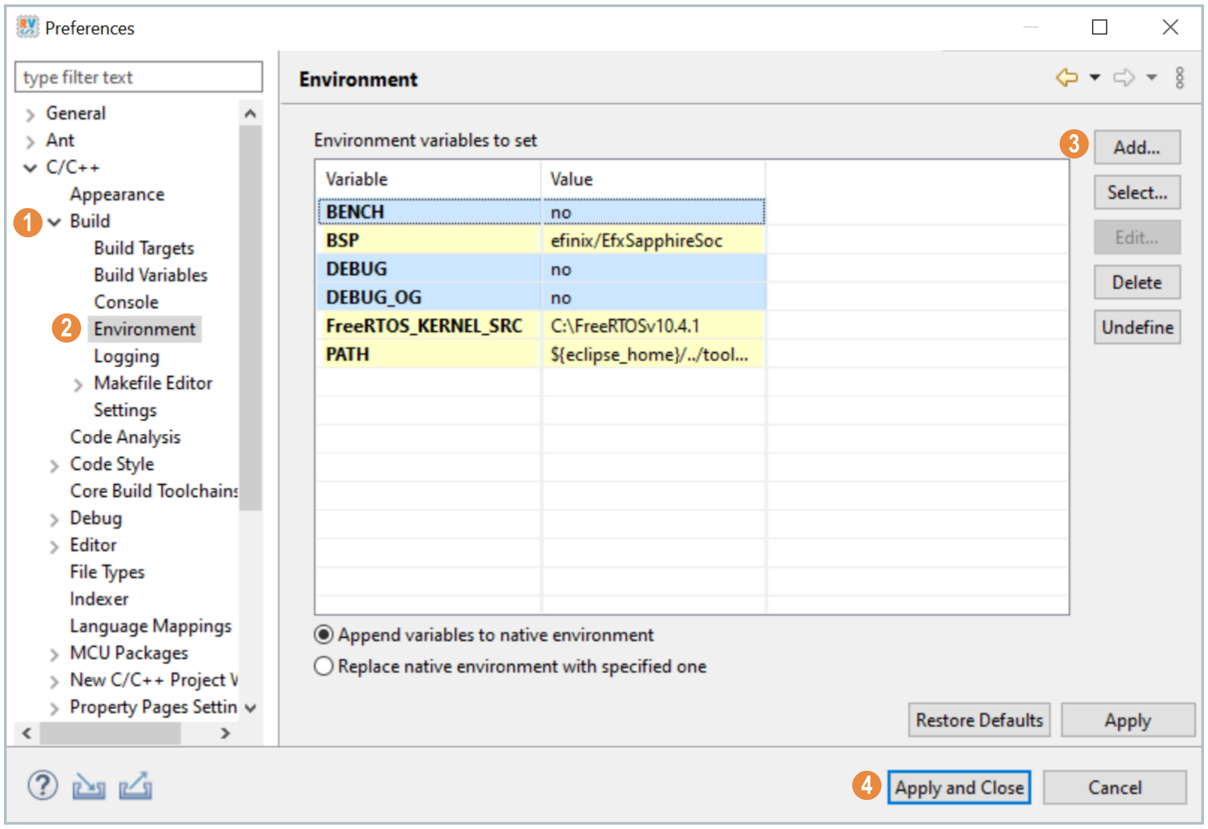
Note: You can choose any location for your workspace. If you have selected a folder that does not exist, the IDE automatically creates a folder for you.

Optimization Settings

OpenOCD uses three environment variables, `DEBUG`, `BENCH`, and `DEBUG_OG`. It is simplest to set them variables as global environment variables for all projects in your workspace. Then, you can adjust them as needed for individual projects.

Note: When you configure the SoC in the IP Manager, you can choose whether to turn on the debug mode by default or not. When you generate the SoC, the setting is saved in the **/embedded_sw/bsp/efinix/EfxSapphireSoc/include/soc.mk** file. If you want to change the debug mode, you can change the setting in the IP Configuration wizard and re-generate the SoC or use the following instructions to add the variables to your project and change them there.

Choose **Window > Preferences** to open the **Preferences** window and perform the following steps.



Variable	Value
BENCH	no
BSP	efinix/EfxSapphireSoc
DEBUG	no
DEBUG_OG	no
FreeRTOS_KERNEL_SRC	C:\FreeRTOSv10.4.1
PATH	${eclipse_home}/../tool...

[✓ Append variables to native environment]
Replace native environment with specified one

1. In the left navigation menu, expand **C/C++ > Build**.
2. Click **C/C++ > Build > Environment**.
3. Click **Add** to add the following environment variables:
4. Click **Apply and Close**.

Table 16: Environment Settings for Preferences Window

Variable	Value	Description
DEBUG	no	Enables or disables debug mode. no: Debugging is turned off yes: Debugging is enabled (-g3) **Note:** Setting the DEBUG to **no** prevents you from debugging step by step in the IDE but saves memory resources.
DEBUG_OG	no	Enables or disables optimization during debugging. Use an uppercase letter O not a zero. no: No optimization for debugging (-O0 setting) yes: Optimization for debugging (-Og setting)
BENCH	no	Modify the optimization level when DEBUG is set to **no**. no: Optimization for size (-Os) yes: Optimization for speed (-O3)

Alternatively, you may modify the variable through the projects's makefile similar to how it is done for coremark demo project.

```
PROJ_NAME=coremark
STANDALONE = ..
DEBUG=no
BENCH=yes
CFLAGS += -DITERATIONS=2000

SRCS =       $(wildcard src/*.c) \
             $(wildcard src/*.cpp) \
             $(wildcard src/*.S) \
             ${STANDALONE}/common/start.S

include ${STANDALONE}/common/bsp.mk
include ${STANDALONE}/common/riscv64-unknown-elf.mk
include ${STANDALONE}/common/standalone.mk
```

Note: For more information on the optimization settings, refer to https://gcc.gnu.org/onlinedocs/gcc-8.3.0/gcc/Optimize-Options.html

Chapter 6

Create, Import, and Build a Software Project

Contents:

After you set up your IDE workspace, you are ready to create a new or import an existing project and build it. These instructions walk you through the process using the new project wizard to create a project as well as using the import project wizard to import existing projects and build it.

Note: For more details on how to create a new project, refer to Sapphire RISC-V SoC Hardware and Software User Guide.

Create a New Project

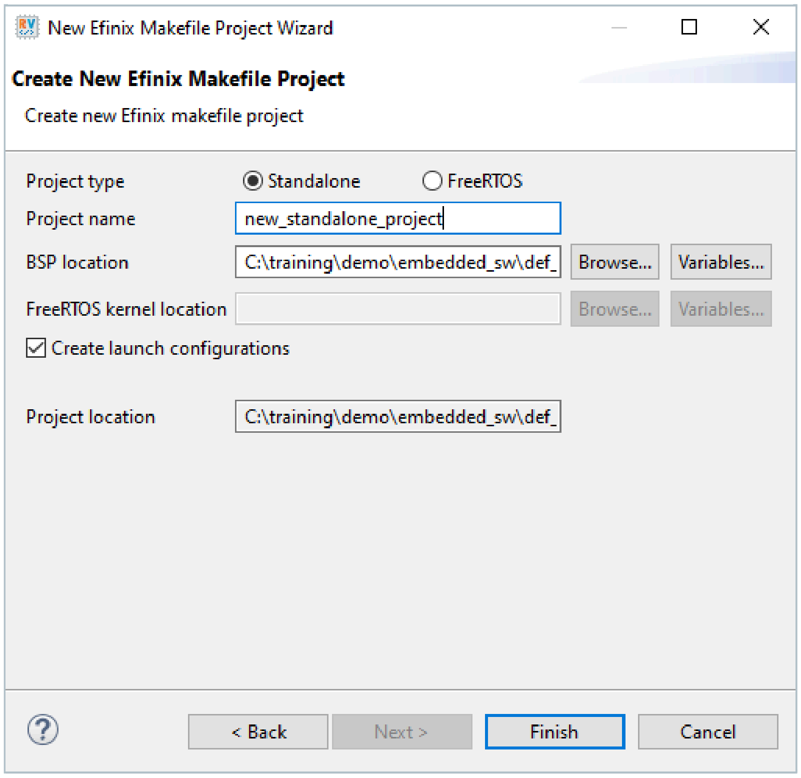
In the **Project Explorer**:

1. Click **Create a Project** to open the new project wizard.
2. Select the **Efinix Project** > **Efinix Makefile Project** > **Next .**

In the **New Efinix Makefile Project Wizard** window:

3. Select either **Standalone** or **FreeRTOS** project type. With this selection, the IDE imports the required header files.
4. Enter your project name. Whitespaces cause error and prevent you to complete the new project creation.
5. Click on **Browse...** > **Board Support Package (BSP)**. BSP location is generated by Efinity when you generate the Sapphire SoC with the IP Manager. Example BSP location: **C:/<project name>/embedded_sw/<ip name>/bsp/**.
6. Select **FreeRTOS**, browse to the **FreeRTOS** kernel location. By default, the kernel location is pointing to the FreeRTOS that comes with package.
7. The new project location shows up.
8. Click **Finish**.

Figure 12: Create New Efinix Makefile Project Wizard for Standalone Project


New Efinix Makefile Project Wizard

Create New Efinix Makefile Project

Create new Efinix makefile project

Project type [✓ Standalone] FreeRTOS

Project name: new_standalone_project

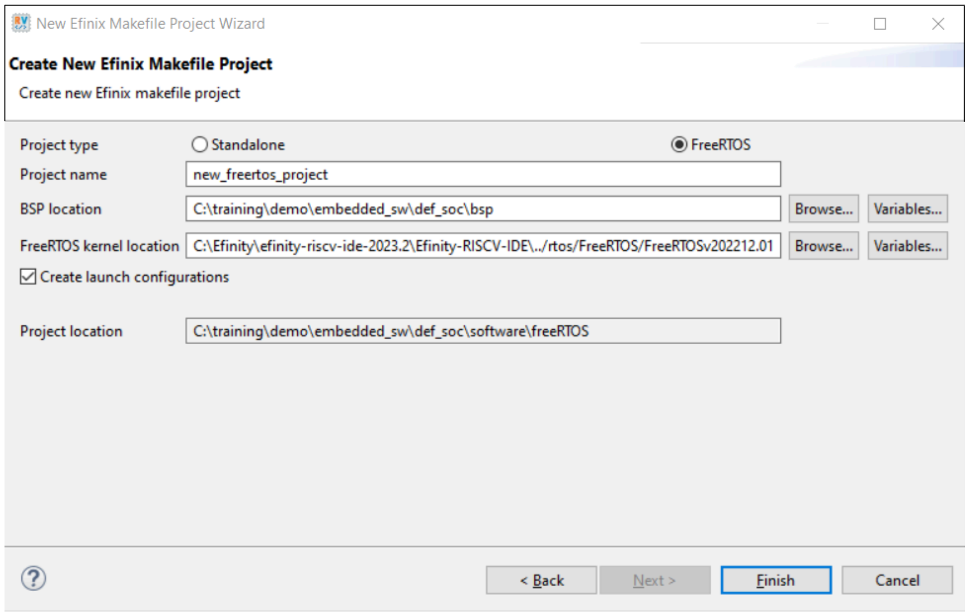
BSP location: C:\training\demo\embedded_sw\def_

FreeRTOS kernel location:

[✓] Create launch configurations

Project location: C:\training\demo\embedded_sw\def_

Figure 13: Create New Efinix Makefile Project Wizard for FreeRTOS Project


New Efinix Makefile Project Wizard

Create New Efinix Makefile Project

Create new Efinix makefile project

Project type Standalone [✓ FreeRTOS]

Project name: new_freertos_project

BSP location: C:\training\demo\embedded_sw\def_soc\bsp

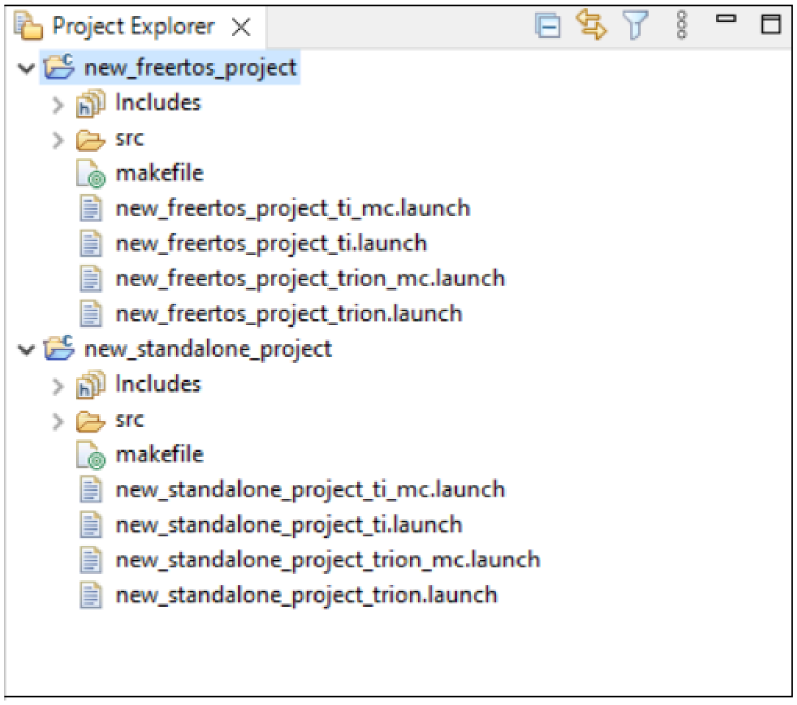
FreeRTOS kernel location: C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01

[✓] Create launch configurations

Project location: C:\training\demo\embedded_sw\def_soc\software\freeRTOS

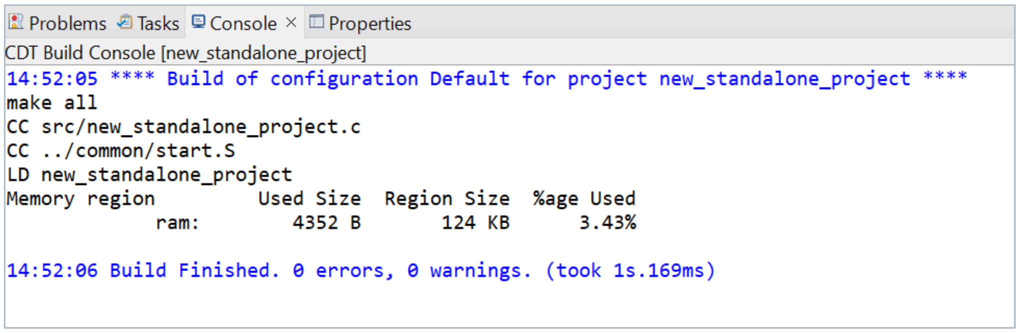
The new projects are updated in the **Project Explorer** pane. All required files are imported automatically. Launch scripts for **softTap** and **ti** configurations are generated automatically based on the debug configuration. **Trion.launch** and **ti.launch** are generated if the **hard TAP** option is selected for the Sapphire SoC, while **softTap.launch** is generated if the **soft TAP** option is selected. Additionally, the corresponding ***_mc.launch** files are generated for multi-core debugging.

Figure 14: Project Explorer Pane Showing All Projects Created


Project Explorer
new_freertos_project
Includes
src
makefile
new_freertos_project_ti_mc.launch
new_freertos_project_ti.launch
new_freertos_project_trion_mc.launch
new_freertos_project_trion.launch
new_standalone_project
Includes
src
makefile
new_standalone_project_ti_mc.launch
new_standalone_project_ti.launch
new_standalone_project_trion_mc.launch
new_standalone_project_trion.launch

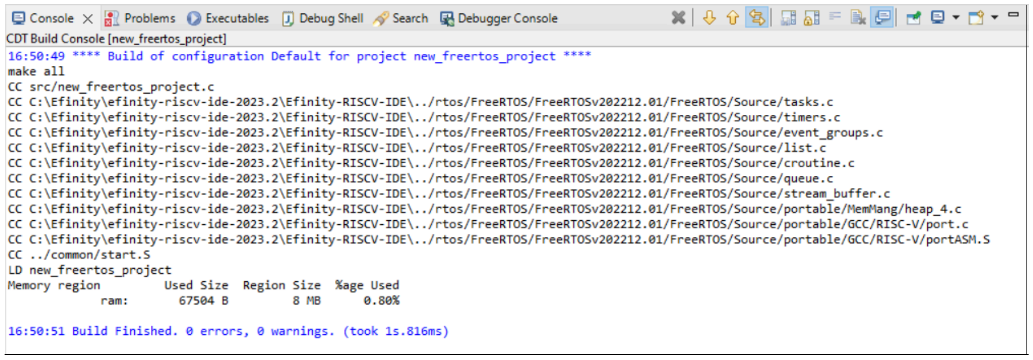
You can now browse the source files. To build the project, right-click on the project and select **Clean Project** > **Build Project**. The compilation output shows in the **Console** window.

Figure 15: Output Console Showing the Newly Generated Standalone Project Built Successfully



```
14:52:05 **** Build of configuration Default for project new_standalone_project ****
make all
CC src/new_standalone_project.c
CC ../common/start.S
LD new_standalone_project
Memory region         Used Size  Region Size  %age Used
             ram:        4352 B       124 KB      3.43%

14:52:06 Build Finished. 0 errors, 0 warnings. (took 1s.169ms)
```

Figure 16: Output Console Showing the Newly Generated FreeRTOS Project Built Successfully



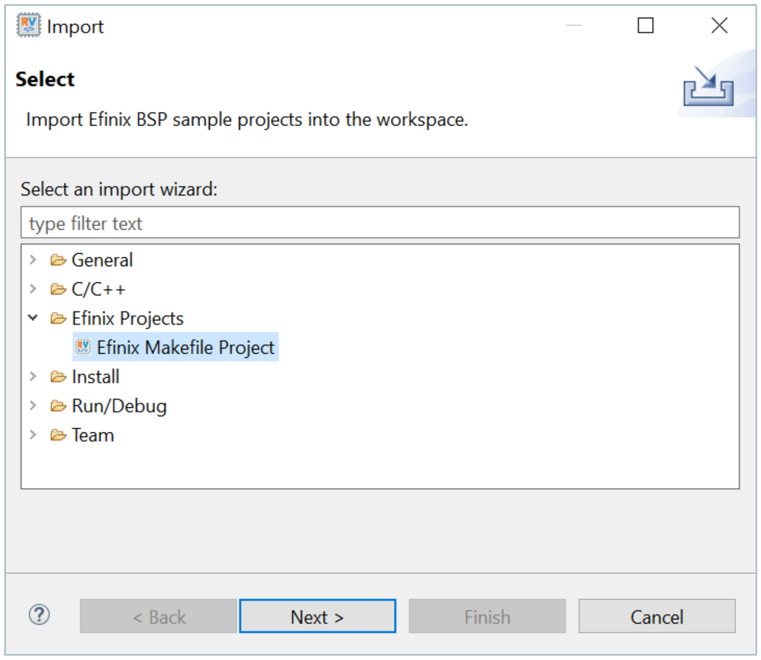
```
16:50:49 **** Build of configuration Default for project new_freertos_project ****
make all
CC src/new_freertos_project.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/tasks.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/timers.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/event_groups.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/list.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/croutine.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/queue.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/stream_buffer.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/portable/MemMang/heap_4.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/portable/GCC/RISC-V/port.c
CC C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/FreeRTOSv202212.01/FreeRTOS/Source/portable/GCC/RISC-V/portASM.S
CC ../common/start.S
LD new_freertos_project
Memory region         Used Size  Region Size  %age Used
             ram:       67504 B         8 MB      0.80%

16:50:51 Build Finished. 0 errors, 0 warnings. (took 1s.816ms)
```

Import Sample Projects

Efinix provides sample projects to help you get started with Sapphire SoC. The sample projects are generated with the Efinity software. The followings steps explain how to import existing projects into the IDE:

1. In the **Project Explorer**, click on **Import Projects...** to open the **Import** wizard.
2. In the **Import** wizard, select **Efinix Makefile Project** in **Efinix Projects** and click **Next**.

Figure 17: Import Wizard



Import

Select

Import Efinix BSP sample projects into the workspace.

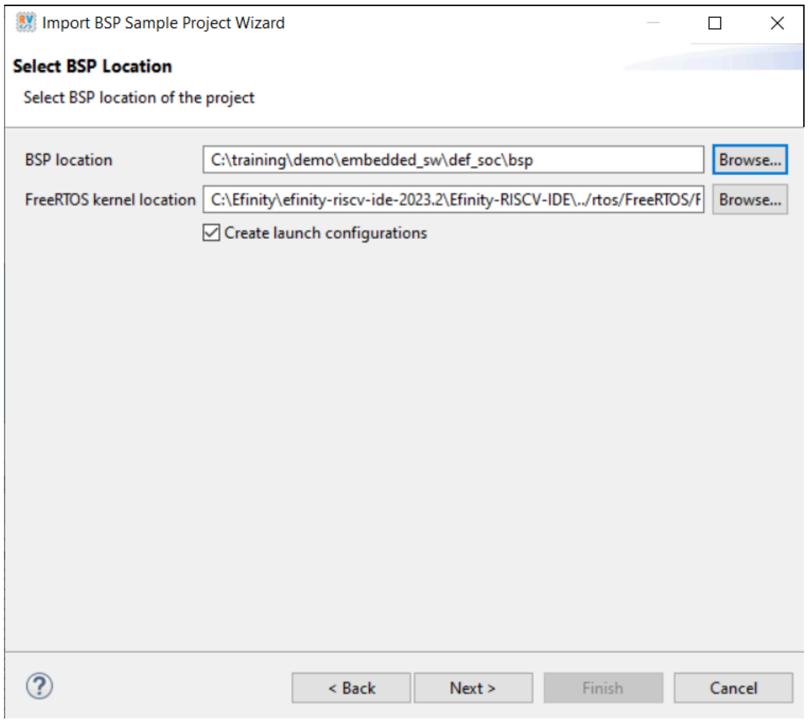
Select an import wizard:

type filter text

3. In the **Import BSP Sample Project Wizard**, click **Next** to browse to the next **BSP location** box.

4. If you would like to import the FreeRTOS sample projects, browse to the **FreeRTOS kernel location**. Turn on **Create launch configurations** and click **Next**.

Figure 18: Import BSP Sample Project Wizard



Import BSP Sample Project Wizard

Select BSP Location

Select BSP location of the project

BSP location: C:\training\demo\embedded_sw\def_soc\bsp

FreeRTOS kernel location: C:\Efinity\efinity-riscv-ide-2023.2\Efinity-RISCV-IDE\../rtos/FreeRTOS/F

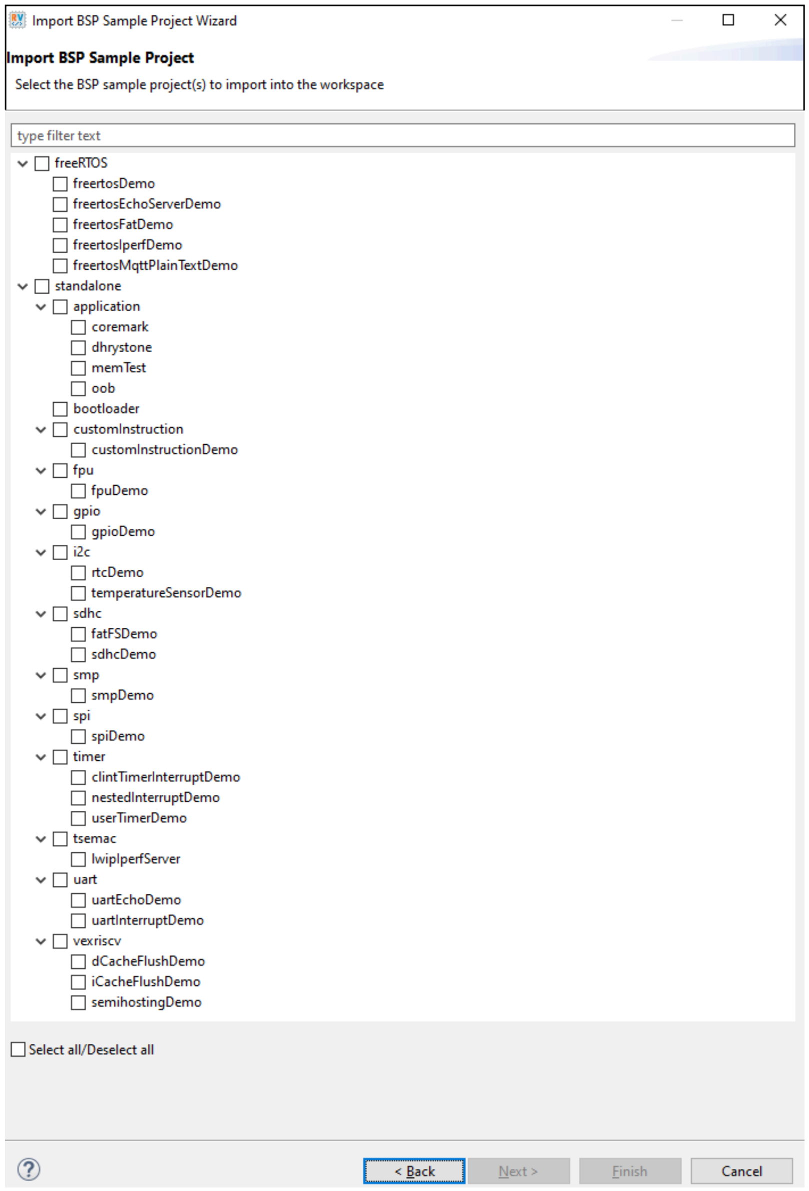
☑ Create launch configurations

Note: FreeRTOS projects is filtered if the FreeRTOS kernel location is not defined.

The next wizard page shows the **Import BSP Sample Project Wizard**, all sample projects are located in the **embedded_sw/<soc name>/software** are shown. Follow these steps:

1. Turn on the specific project to import that project.
2. You may turn on the sub-category, for example: Free RTOS, to import all the projects belonging to that particular sub-category.
3. Alternatively, you may click **Select all / Deselect all** to select or deselect all the projects available.
4. Click **Next**.

Figure 19: Import BSP Sample Project Wizard – List of Projects



Import BSP Sample Project

Select the BSP sample project(s) to import into the workspace

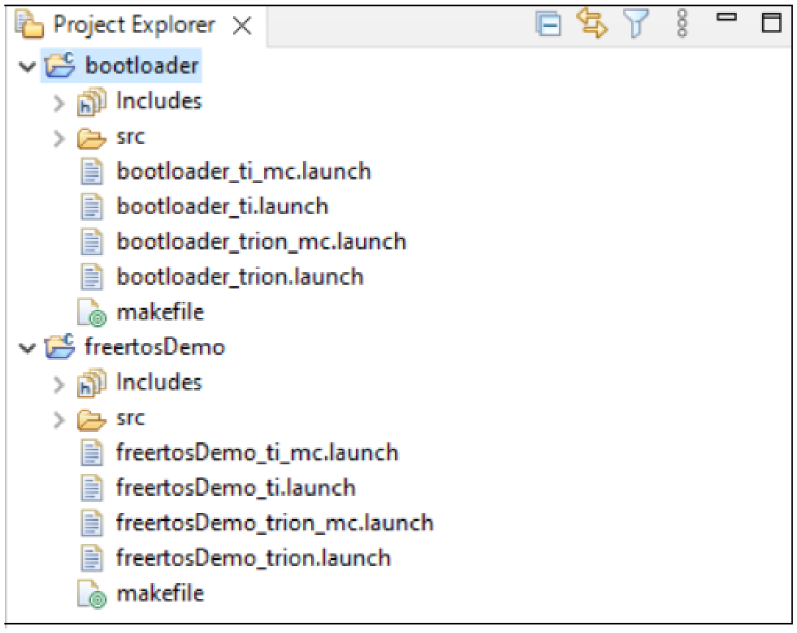
- freeRTOS
  - freertosDemo
  - freertosEchoServerDemo
  - freertosFatDemo
  - freertosIperfDemo
  - freertosMqttPlainTextDemo
- standalone
  - application
    - coremark
    - dhrystone
    - memTest
    - oob
  - bootloader
  - customInstruction
    - customInstructionDemo
  - fpu
    - fpuDemo
  - gpio
    - gpioDemo
  - i2c
    - rtcDemo
    - temperatureSensorDemo
  - sdhc
    - fatFSDemo
    - sdhcDemo
  - smp
    - smpDemo
  - spi
    - spiDemo
  - timer
    - clintTimerInterruptDemo
    - nestedInterruptDemo
    - userTimerDemo
  - tsemac
    - lwipIperfServer
  - uart
    - uartEchoDemo
    - uartInterruptDemo
  - vexriscv
    - dCacheFlushDemo
    - iCacheFlushDemo
    - semihostingDemo

Note:

- If you have custom programs that need to be exported to the IDE, you may either copy the programs into existing folders (FreeRTOS or Standalone) or you can create a folder at the same level as FreeRTOS and Standalone folders. Automatically, the IDE identifies the folder as a sub-category.
- IDE uses **makefile** to identify if the folder is considered a project. Ensure that you have a valid **makefile** for your custom project.

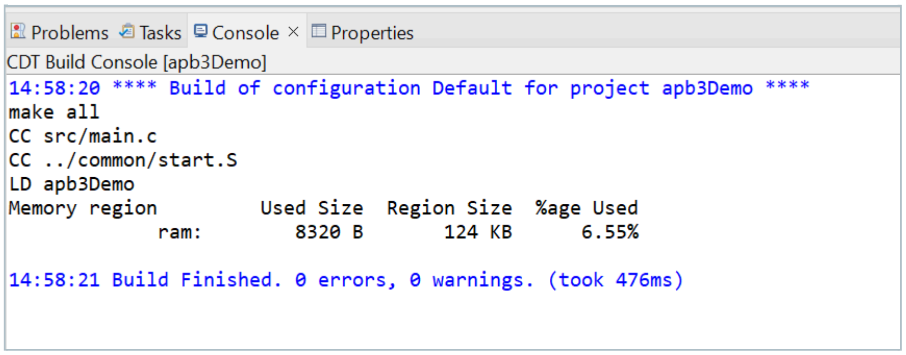
The selected sample projects are imported into the listed workspace in the **Project Explorer** pane.

Figure 20: Project Explorer Pane showing all the Imported Projects



You can now browse the source files. To build the project, right-click the project name and select **Clean Project** > **Build Project**. The compilation output shows up in the **Console** window.

Figure 21: Output Console showing the apb3Demo Standalone Project Built Successfully



```
14:58:20 **** Build of configuration Default for project apb3Demo ****
make all
CC src/main.c
CC ../common/start.S
LD apb3Demo
Memory region         Used Size  Region Size  %age Used
             ram:        8320 B       124 KB      6.55%

14:58:21 Build Finished. 0 errors, 0 warnings. (took 476ms)
```

Build

Choose **Project > Build Project** or click the Build Project toolbar button. Alternatively, right-click the project name in **Project Explorer > Build Project**.

Efinix recommends cleaning your project before building to ensure all files are compiled. To clean project, right-click on the project in **Project Explorer > Clean Project**.

The **makefile** builds the project and generates these files in the **build** directory:

- **<project name>.asm**—Assembly language file for the firmware.
- **<project name>.bin**—Firmware binary file. Download this file to the flash device on your board using OpenOCD. When you turn the board on, the SoC loads the application into the RISC-V processor and executes it.
- **<project name>.elf**—Executable and linkable format. Use this file when debugging with the OpenOCD debugger.
- **<project name>.hex**—Hex file for the firmware. (Do not use it to program the FPGA.)
- **<project name>.map**—Contains the SoC address map.

Chapter 7

Debug with the OpenOCD Debugger

Contents:

With the development board programmed and the software built, you are ready to configure the OpenOCD debugger and perform debugging. These instructions use the **axiDemo** example to explain the steps required.

Launch the Debug Script

With the Efinity software v2022.2 and higher, debugging scripts are available for each software example in the **/embedded_sw/<module>/software/** directory and are imported into your project when you create a new project or importing existing project into the workspace. You can use these scripts to launch the debug mode.

Table 17: Debug Configurations

Launch Script	Description
axiDemo_trion.launch	Debugging software on Trion® development boards.
axiDemo_ti.launch	Debugging software on Titanium development boards.
axiDemo_softTap.launch	Debugging software on Trion or Titanium development boards with the soft JTAG TAP interface. For example, you would need to use the soft TAP if you want to use the OpenOCD debugger and the Efinity® Debugger at the same time. (See **Using a Soft JTAG Core for Example Designs** on page 112.)
axiDemo_trion_mc.launch	Debugging software on Trion® development boards with multiple cores.
axiDemo_ti_mc.launch	Debugging software on Titanium development boards with multiple cores.
axiDemo_softTap_mc.launch	Debugging software on Trion or Titanium development boards with the soft JTAG TAP interface with multiple cores. For example, you would need to use the soft TAP if you want to use the OpenOCD debugger and the Efinity® Debugger at the same time. (See **Using a Soft JTAG Core for Example Designs** on page 112.)

Note:

- The `*_mc.launch` scripts for SMP debug are generated by the Efinity RISC-V Embedded Software IDE v2023.1 or later, with the number of cores configured is more than 1 core.
- You may receive TAP ID warnings in the Eclipse console when trying to debug a device with **softTap** enabled that is not the Trion T120F324 device. These warnings do not affect the debugging process. To remove these warnings, see **Unexpected CPUTAPID/JTAG Device ID** on page 134.

To debug the axiDemo project:

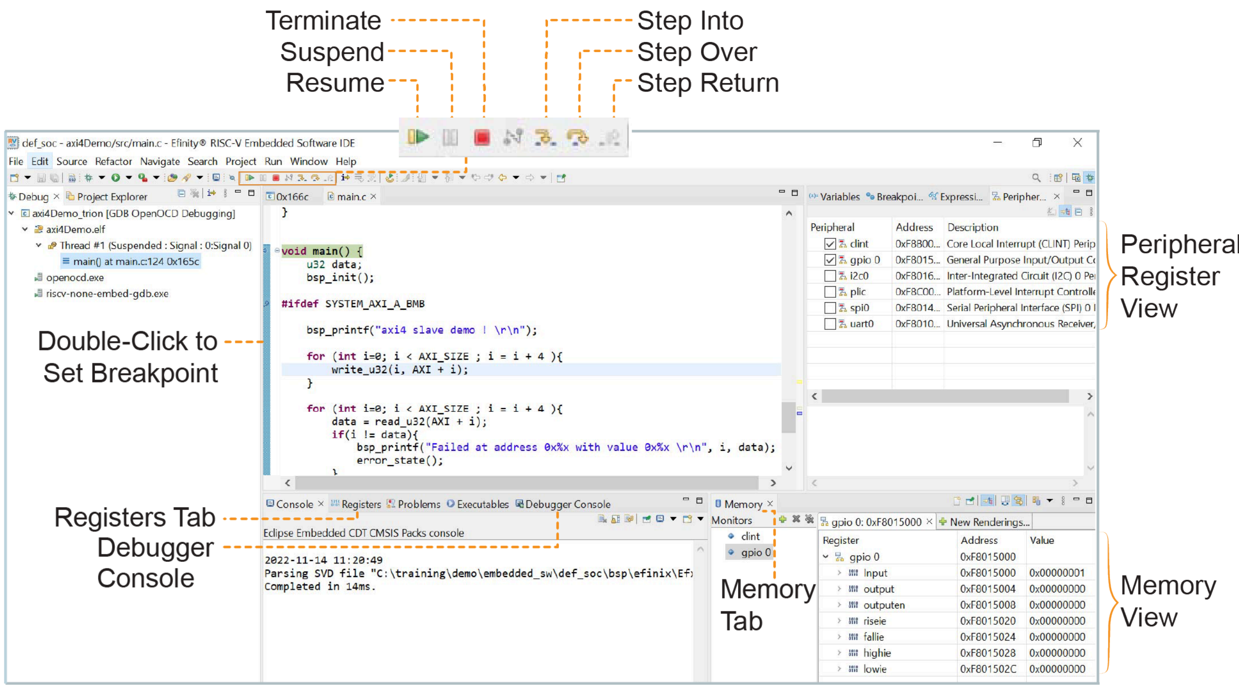
1. Right-click **axiDemo** > **axiDemo_<family>.launch**.
2. Choose **Debug As** > **axiDemo** > **axiDemo_<family>**. Efinity RISC-V Embedded Software IDE launches the OpenOCD debugger for the project.
3. Click **Debug**.
4. **Confirm Perspective Switch** window would prompt out. Click **Switch** to switch from **C/C++** perspective to **Debug** perspective to start the debug process.

Debug

After you click **Debug** in the Debug Configuration window, the OpenOCD server starts, connects to the target, starts the gdb client, downloads the application, and starts the debugging session. Messages and a list of VexRiscv registers display in the **Console**. The **main.c** file opens so you can debug each step.

1. Click the **Resume** button or press F8 to resume code operation.
2. Click **Step Over** (F6) to do a single step over one source instruction.
3. Click **Step Into** (F5) to do a single step into the next function called.
4. Click **Step Return** (F7) to do a single step out of the current function.
5. Double-click in the bar to the left of the source code to set a breakpoint. Double-click a breakpoint to remove it.
6. Click the **Registers** tab to inspect the processor's registers including the CSR registers.
7. Click the **Memory** tab to inspect the memory contents including the Peripheral register monitors.
8. Click the **Suspend** button to stop the code operation.
9. Turn on any peripheral in the Peripheral pane to add the peripheral to the Memory monitor.
10. When you finish debugging, click **Terminate** to disconnect the OpenOCD debugger.

Figure 22: Perform Debugging



```
}

void main() {
    u32 data;
    bsp_init();

#ifdef SYSTEM_AXI_A_BMB

    bsp_printf("axi4 slave demo ! \r\n");

    for (int i=0; i < AXI_SIZE ; i = i + 4 ){
        write_u32(i, AXI + i);
    }

    for (int i=0; i < AXI_SIZE ; i = i + 4 ){
        data = read_u32(AXI + i);
        if(i != data){
            bsp_printf("Failed at address 0x%x with value 0x%x \r\n", i, data);
            error_state();
```

Peripheral	Address	Description
✓ clint	0xF8B00...	Core Local Interrupt (CLINT) Perip
✓ gpio 0	0xF8015...	General Purpose Input/Output Co
i2c0	0xF8016...	Inter-Integrated Circuit (I2C) 0 Pe
plic	0xF8C00...	Platform-Level Interrupt Controlle
spi0	0xF8014...	Serial Peripheral Interface (SPI) 0 I
uart0	0xF8010...	Universal Asynchronous Receiver,

```
2022-11-14 11:20:49
Parsing SVD file "C:\training\demo\embedded_sw\def_soc\bsp\efinix\Ef
Completed in 14ms.
```

Register	Address	Value
gpio 0	0xF8015000	
Input	0xF8015000	0x00000001
output	0xF8015004	0x00000000
outputen	0xF8015008	0x00000000
riseie	0xF8015020	0x00000000
fallie	0xF8015024	0x00000000
highie	0xF8015028	0x00000000
lowie	0xF801502C	0x00000000

Learn more: For more information on debugging with Eclipse, refer to **Running and debugging projects** in the Eclipse documentation.

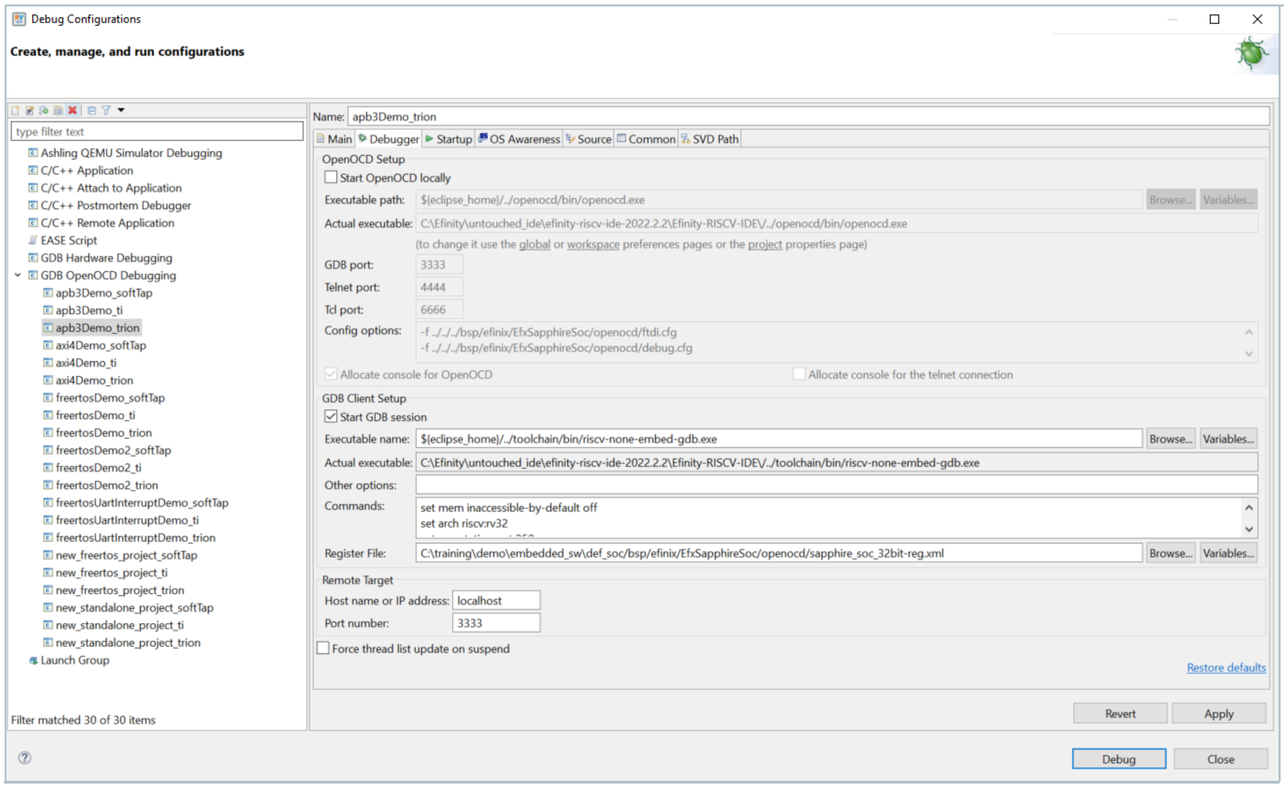
Debug - Multiple Cores

Debug - Single Core

By default, the OpenOCD debugger always targets the first core, core 0, when debugging. If your SoC has multiple cores, you can do standalone debugging with a core other than core 0. This debug method uses the openocdServer debug launch scripts, which are available in the **software/standalone/openocdServer** directory. The general procedure is:

1. Create an SoC with more than 1 core.
2. Import your software project in Efinity RISC-V Embedded Software IDE.
3. Import openocdServer project with **New** > **Makefile Project** with Existing Code.
4. Start the OpenOCD server.
    a. Right-click **openocdServer** > **openocdServer_<family>.launch**.
    b. Choose **Debug As** > **openocdServer_<family>**.
5. Modify the debug configuration for your application to use the OpenOCD server:
    a. Right-click **<project folder>** > **Debug As** > **Debug Configurations**.
    b. Choose **GDB OpenOCD Debugging** > **<launch script>** (e.g., **axiDemo_trion**).
    c. Click the Debugger tab.
    d. Turn off **Start OpenOCD locally**.
    e. Under **Remote Target**, change the **Port number** for the core you are using (the default is 3333 for core 0).
        - 3333: Core 0
        - 3334: Core 1
        - 3335: Core 2
        - 3336: Core 3
6. Click **Debug**. The RISC-V IDE enters into debug mode targeting the CPU that you specified with the port number.

Figure 23: Modify Debug Configuration for another Core



- Ashling QEMU Simulator Debugging
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB OpenOCD Debugging
  - apb3Demo_softTap
  - apb3Demo_ti
  - apb3Demo_trion
  - axi4Demo_softTap
  - axi4Demo_ti
  - axi4Demo_trion
  - freertosDemo_softTap
  - freertosDemo_ti
  - freertosDemo_trion
  - freertosDemo2_softTap
  - freertosDemo2_ti
  - freertosDemo2_trion
  - freertosUartInterruptDemo_softTap
  - freertosUartInterruptDemo_ti
  - freertosUartInterruptDemo_trion
  - new_freertos_project_softTap
  - new_freertos_project_ti
  - new_freertos_project_trion
  - new_standalone_project_softTap
  - new_standalone_project_ti
  - new_standalone_project_trion
- Launch Group

Filter matched 30 of 30 items

Name: apb3Demo_trion

OpenOCD Setup

- [ ] Start OpenOCD locally

Executable path: ${eclipse_home}/../openocd/bin/openocd.exe

Actual executable: C:\Efinity\untouched_ide\efinity-riscv-ide-2022.2.2\Efinity-RISCV-IDE\/../openocd/bin/openocd.exe

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 3333

Telnet port: 4444

Tcl port: 6666

Config options:
```
-f ../../../bsp/efinix/EfxSapphireSoc/openocd/ftdi.cfg
-f ../../../bsp/efinix/EfxSapphireSoc/openocd/debug.cfg
```

- [✓] Allocate console for OpenOCD
- [ ] Allocate console for the telnet connection

GDB Client Setup

- [✓] Start GDB session

Executable name: ${eclipse_home}/../toolchain/bin/riscv-none-embed-gdb.exe

Actual executable: C:\Efinity\untouched_ide\efinity-riscv-ide-2022.2.2\Efinity-RISCV-IDE\/../toolchain/bin/riscv-none-embed-gdb.exe

Other options:

Commands:
```
set mem inaccessible-by-default off
set arch riscv:rv32
```

Register File: C:\training\demo\embedded_sw\def_soc/bsp/efinix/EfxSapphireSoc/openocd/sapphire_soc_32bit-reg.xml

Remote Target

Host name or IP address: localhost

Port number: 3333

- [ ] Force thread list update on suspend

Debug - SMP

With Efinity software v2023.1 and higher, the multi-core Sapphire SoC can be debugged concurrently on all the available cores in a bare metal program. Import your project into Efinity RISC-V Embedded Software IDE to debug your SMP program. You will notice the following additional launch scripts that are generated:

- **smpDemo_softTap_mc.launch**
- **smpDemo_ti_mc.launch**
- **smpDemo_trion_mc.launch**

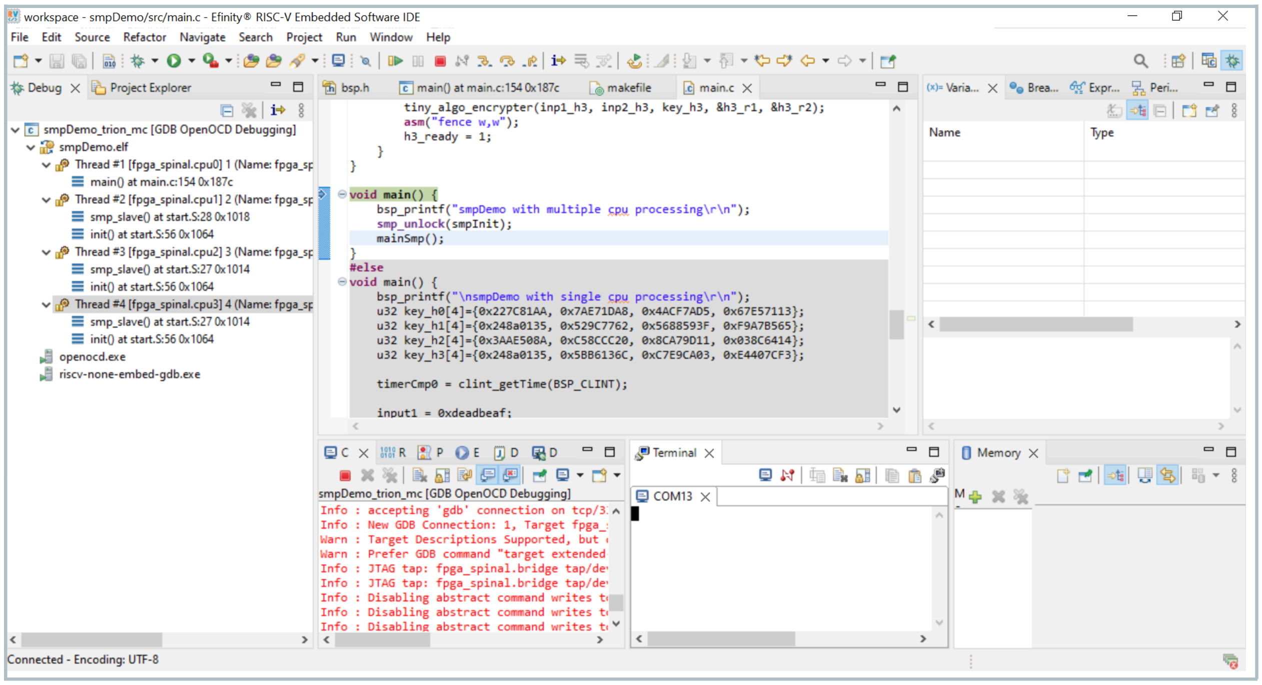
If the `*_mc.launch` scripts are not generated in your Efinity RISC-V Embedded Software IDE, it is advisable to check whether you have imported the correct Board Specific Package (BSP) specifically configured for multiple cores.

Launch `*_mc.launch` based on your hardware configuration; all the cores are shown as threads in the **Debug** pane.

The **Resume and suspend** selection affect all the cores while **Step Into**, **Step Over**, and **Step Return** selections affect only the core you have selected by clicking on the thread.

The **Breakpoint** selection breaks all the cores that go through the specific instruction. If the core does not run the instruction, then the core will not be halted by the breakpoint.

Figure 24: SMP Debug using smpDemo


workspace - smpDemo/src/main.c - Efinity® RISC-V Embedded Software IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Project Explorer
bsp.h
main() at main.c:154 0x187c
makefile
main.c
Varia...
Brea...
Expr...
Peri...
Name
Type
Terminal
COM13
Memory
Connected - Encoding: UTF-8

```
smpDemo_trion_mc [GDB OpenOCD Debugging]
  smpDemo.elf
    Thread #1 [fpga_spinal.cpu0] 1 (Name: fpga_sp
      main() at main.c:154 0x187c
    Thread #2 [fpga_spinal.cpu1] 2 (Name: fpga_sp
      smp_slave() at start.S:28 0x1018
      init() at start.S:56 0x1064
    Thread #3 [fpga_spinal.cpu2] 3 (Name: fpga_sp
      smp_slave() at start.S:27 0x1014
      init() at start.S:56 0x1064
    Thread #4 [fpga_spinal.cpu3] 4 (Name: fpga_sp
      smp_slave() at start.S:27 0x1014
      init() at start.S:56 0x1064
  openocd.exe
  riscv-none-embed-gdb.exe
```

```
            tiny_algo_encrypter(inp1_h3, inp2_h3, key_h3, &h3_r1, &h3_r2);
            asm("fence w,w");
            h3_ready = 1;
        }
}

void main() {
    bsp_printf("smpDemo with multiple cpu processing\r\n");
    smp_unlock(smpInit);
    mainSmp();
}
#else
void main() {
    bsp_printf("\nsmpDemo with single cpu processing\r\n");
    u32 key_h0[4]={0x227C81AA, 0x7AE71DA8, 0x4ACF7AD5, 0x67E57113};
    u32 key_h1[4]={0x248a0135, 0x529C7762, 0x5688593F, 0xF9A7B565};
    u32 key_h2[4]={0x3AAE508A, 0xC58CCC20, 0x8CA79D11, 0x038C6414};
    u32 key_h3[4]={0x248a0135, 0x5BB6136C, 0xC7E9CA03, 0xE4407CF3};

    timerCmp0 = clint_getTime(BSP_CLINT);

    input1 = 0xdeadbeaf;
```

```
smpDemo_trion_mc [GDB OpenOCD Debugging]
Info : accepting 'gdb' connection on tcp/3
Info : New GDB Connection: 1, Target fpga_
Warn : Target Descriptions Supported, but
Warn : Prefer GDB command "target extended
Info : JTAG tap: fpga_spinal.bridge tap/de
Info : JTAG tap: fpga_spinal.bridge tap/de
Info : Disabling abstract command writes t
Info : Disabling abstract command writes t
Info : Disabling abstract command writes t
```

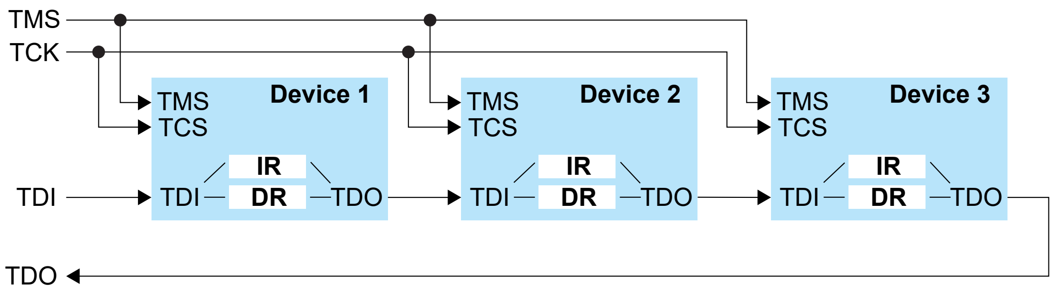
Note:

- To use the SMP debug, you must use both the Efinity RISC-V Embedded Software IDE v2023.1 and Standard debug interface.
- By default, the smpDemo sets the `DEBUG` to **NO**. Modify the `DEBUG` setting in the project **makefile** and then rebuild the project.

Debug - Daisy Chain

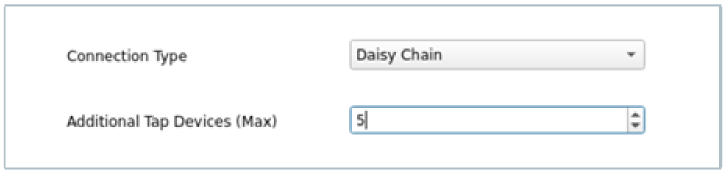
JTAG allows multiple devices to be connected to an interface in a daisy chain configuration. This allows user to use only one debugger to access multiple devices. In a daisy chain, only `TMS` and `TCK` signals are common signals which means these signals are required to be routed to all devices on the board. The first device's `TDI` is connected to the debugger and its `TDO` is connected to the `TDI` of the next device. The connection is shown in the following figure.

Figure 25: Daisy Chain Connection of Multiple Devices



In IP Manager, by default, the SoC has a **standalone** configuration that JTAG cannot access the device which, is outside of SoC. To fix this issue, you need to change the **Connection Type** to **Daisy Chain**. Then, specify the number of additional devices in the chain using the option **Additional Tap Devices (Max)**. The total number of devices in the chain should be **Additional Tap Devices (Max) + 1**. If you select 5 (five) as option for **Additional Tap Devices (Max)**, then you should have a total of up to 6 (six) devices in the chain.

Figure 26: Daisy Chain Parameter Input



Connection Type	Daisy Chain
Additional Tap Devices (Max)	5

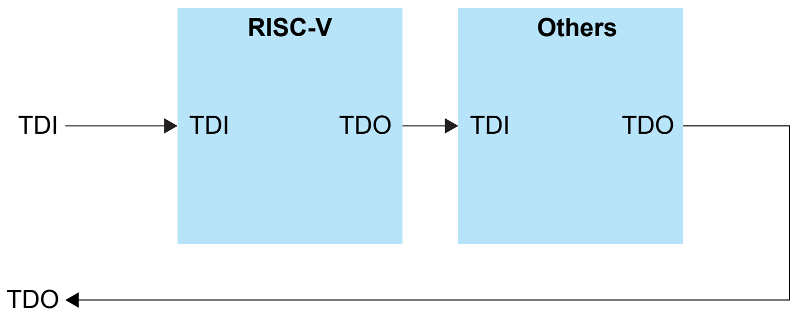
After generating the SoC, you need to manually edit the debug launch script to include the device information in the chain. The debug launch script is available in **embedded_sw/<soc>/bsp/efinix/EfxSapphireSoc/openocd/debug_<type>.cfg**. The debug launch script you select depends on the **tap type** that is being used on the first device, which is either **Trion hardened tap**, **Titanium hardened tap**, or **soft tap**.

You need to insert information as described:

1. Create a new tap using `jtag newtap` command to specify other devices in the chain. You need to provide the tap details like IR length, IR capture, and IR mask.

2. Create a new target if there is Sapphire SoC available with the tap.

Figure 27: Example 1: Daisy-Chain with 2 Devices



Example 1: Two devices in the daisy chain, and one of them is running on Sapphire SoC. The debug launch script is:

```
set _CHIPNAME fpga_spinal
set _CHIPNAME1 fpga_spinal1

#device closest to TDO
jtag newtap $_CHIPNAME bridge -expected-id $_CPUTAPID -irlen <unknown> -
ircapture <unknown> -irmask <unknown>
jtag newtap $_CHIPNAME1 bridge -expected-id $_CPUTAPID -irlen <unknown> -
ircapture <unknown> -irmask <unknown>
#device closest to TDI

target create $_CHIPNAME1.cpu0 vexriscv -endian $_ENDIAN -chain-position
 $_CHIPNAME1.bridge -coreid 0 -dbgbase 0x10B80000
vexriscv readWaitCycles 12
vexriscv cpuConfigFile $CPU0_YAML
vexriscv jtagMapping 8 8 0 1 2 2 1 1
```

3. Define a name for every device in the chain. Since, there are two devices, give it a name like *line 1* and *line 2*.
4. Create a new tap with the tap details. In the figure, the device closest to the `TDO` pin is the device that is without Sapphire SoC and the device closest to the `TDI` is the device with Sapphire SoC. Refer to the following table and fill the <unknown> with the correct tap details.

Table 18: Tap Details

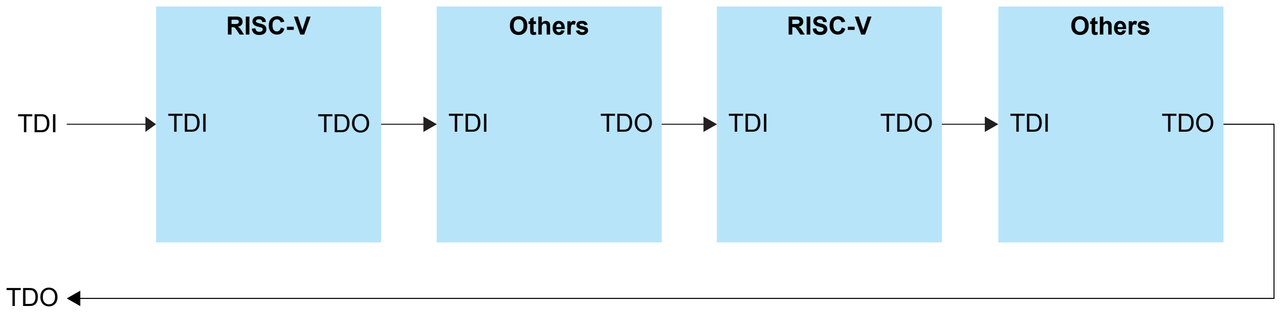
Device	IR Length	IR Capture	IR Mask
Soft Tap	4	0x1	0xF

5. Create a new target to specify the Sapphire SoC JTAG mapping details. Refer to the following table to insert the correct JTAG mapping.

Table 19: JTAG Mapping Details

Additional Tap Devices (Max)	JTAG Mapping
1	88012211
2	88012222
3	88012233
4	88012244
5	88012255
6	88012266
7	88012277
8	88012288

Figure 28: Example 2: Daisy Chain with 4 Devices



Example 2: Four devices in the daisy chain, and two of them are running on Sapphire SoC. The debug launch script is:

```
set _CHIPNAME other
set _CHIPNAME1 fpga_spinal1
set _CHIPNAME2 fpga_other2
set _CHIPNAME1 fpga_spinal3

#device closest to TDO
jtag newtap $_CHIPNAME bridge -irlen <unknown> -ircapture <unknown> -irmask <unknown>
jtag newtap $_CHIPNAME1 bridge -expected-id $_CPUTAPID -irlen <unknown> -ircapture <unknown> -
irmask <unknown>
jtag newtap $_CHIPNAME2 bridge -irlen <unknown> -ircapture <unknown> -irmask <unknown>
jtag newtap $_CHIPNAME3 bridge -expected-id $_CPUTAPID -irlen <unknown> -ircapture <unknown> -
irmask <unknown>
#device closest to TDI

target create $_CHIPNAME1.cpu0 vexriscv -endian $_ENDIAN -chain-position $_CHIPNAME1.bridge -
coreid 0 -dbgbase 0x10B80000
vexriscv readWaitCycles 12
vexriscv cpuConfigFile $CPU0_YAML
vexriscv jtagMapping 8 8 0 1 2 2 3 3

target create $_CHIPNAME3.cpu0 vexriscv -endian $_ENDIAN -chain-position $_CHIPNAME3.bridge -
coreid 0 -dbgbase 0x10B80000
vexriscv readWaitCycles 12
vexriscv cpuConfigFile $CPU0_YAML
vexriscv jtagMapping 8 8 0 1 2 2 3 3
```

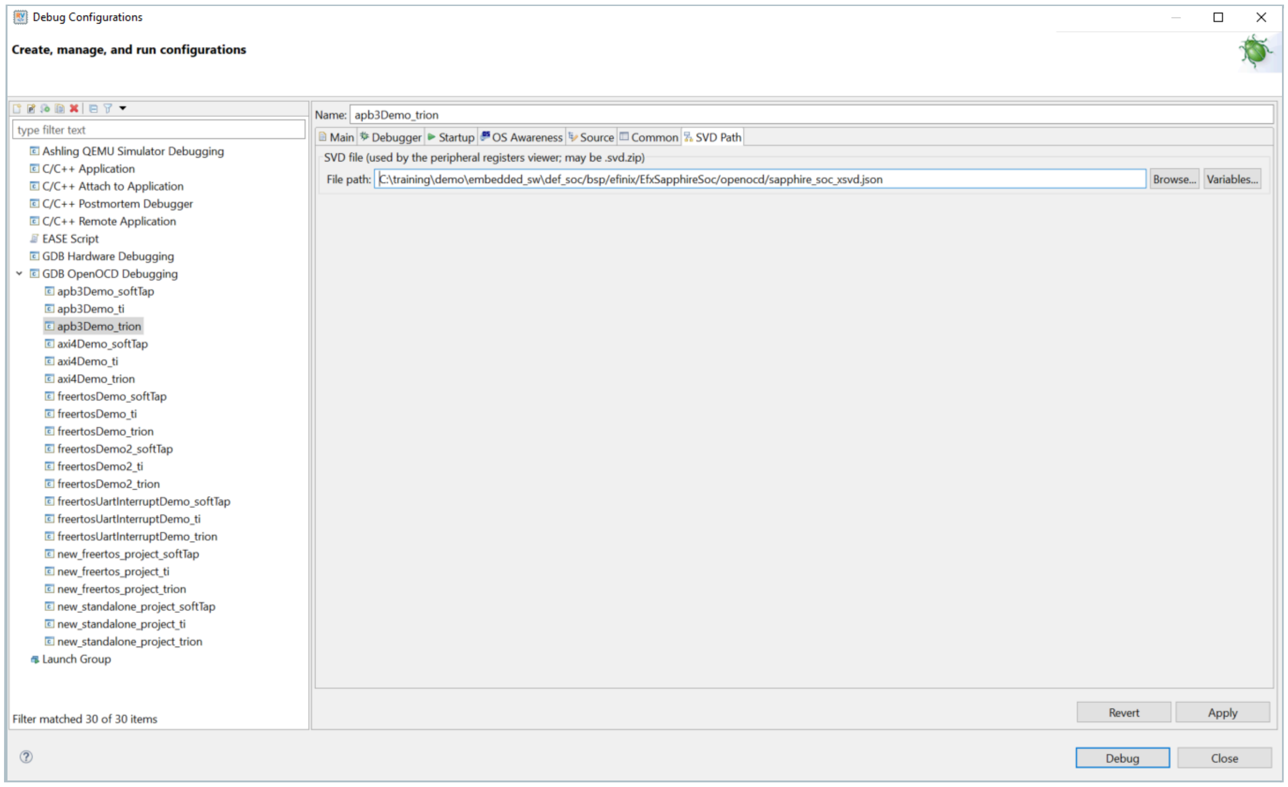
Peripheral Register View

With the Peripheral Register View, you can easily view the value of each register of the peripherals you have enabled for the Sapphire SoC. The view helps you with your debugging process without the need to view through memory addresses.

The IDE automatically points to the **.xsvd** file generated by the Efinity software. If you want to point to a different xsvd file, you may do it by going to **Debug > Debug Configurations > SVD Path** and browse to another **xsvd.json** file. The default generated **.xsvd** file is located in **/bsp/efinix/EfxSapphireSoc/openocd/sapphire_soc_xsvd.json**

Learn more: For more information on **xsvd** format, refer to the xPack SVD Definitions. This brings you to the website upon clicking.

Figure 29: SVD Path Tab in Debug Configuration


Debug Configurations

Create, manage, and run configurations

type filter text

- Ashling QEMU Simulator Debugging
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB OpenOCD Debugging
  - apb3Demo_softTap
  - apb3Demo_ti
  - apb3Demo_trion
  - axi4Demo_softTap
  - axi4Demo_ti
  - axi4Demo_trion
  - freertosDemo_softTap
  - freertosDemo_ti
  - freertosDemo_trion
  - freertosDemo2_softTap
  - freertosDemo2_ti
  - freertosDemo2_trion
  - freertosUartInterruptDemo_softTap
  - freertosUartInterruptDemo_ti
  - freertosUartInterruptDemo_trion
  - new_freertos_project_softTap
  - new_freertos_project_ti
  - new_freertos_project_trion
  - new_standalone_project_softTap
  - new_standalone_project_ti
  - new_standalone_project_trion
- Launch Group

Filter matched 30 of 30 items

Name: apb3Demo_trion

Main | Debugger | Startup | OS Awareness | Source | Common | SVD Path

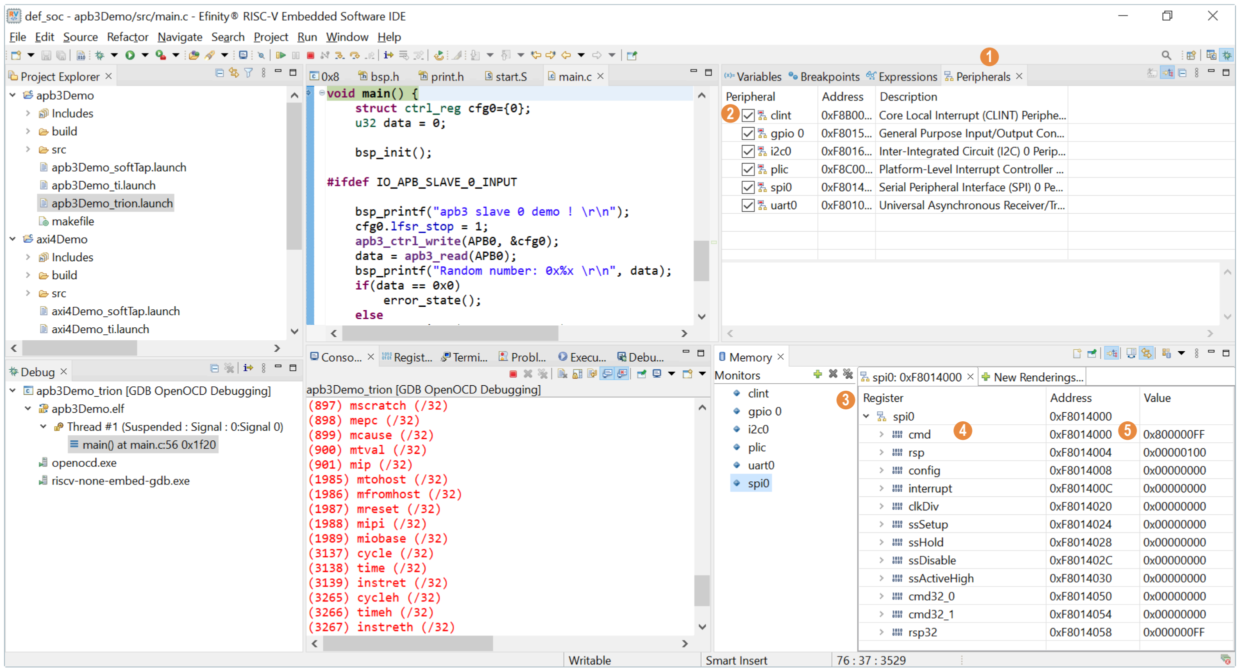
SVD file (used by the peripheral registers viewer; may be .svd.zip)

File path: C:\training\demo\embedded_sw\def_soc/bsp/efinix/EfxSapphireSoc/openocd/sapphire_soc_xsvd.json

When working with the Peripherals View, note that:

1. All available peripherals for the current Sapphire SoC are listed in the **Peripherals** tab.
2. To view the specific peripheral, turn on the preferred peripheral.
3. Once you have chosen the peripheral(s), the **Memory Monitor** shows up on the bottom right.
4. To view the register, just select the specific peripheral in the **Monitor**.
5. Each register has its own address and value in the memory. Hover your mouse over the register to view more information on each register.
6. The color on the register row changes if the current value is different from the previous value.

Figure 30: GUI with Peripherals View for all Available Peripherals



```
void main() {
    struct ctrl_reg cfg0={0};
    u32 data = 0;

    bsp_init();

#ifdef IO_APB_SLAVE_0_INPUT

    bsp_printf("apb3 slave 0 demo ! \r\n");
    cfg0.lfsr_stop = 1;
    apb3_ctrl_write(APB0, &cfg0);
    data = apb3_read(APB0);
    bsp_printf("Random number: 0x%x \r\n", data);
    if(data == 0x0)
        error_state();
    else
```

Peripheral	Address	Description
✓ clint	0xF8B00...	Core Local Interrupt (CLINT) Periphe...
✓ gpio 0	0xF8015...	General Purpose Input/Output Con...
✓ i2c0	0xF8016...	Inter-Integrated Circuit (I2C) 0 Perip...
✓ plic	0xF8C00...	Platform-Level Interrupt Controller ...
✓ spi0	0xF8014...	Serial Peripheral Interface (SPI) 0 Pe...
✓ uart0	0xF8010...	Universal Asynchronous Receiver/Tr...

```
apb3Demo_trion [GDB OpenOCD Debugging]
(897) mscratch (/32)
(898) mepc (/32)
(899) mcause (/32)
(900) mtval (/32)
(901) mip (/32)
(1985) mtohost (/32)
(1986) mfromhost (/32)
(1987) mreset (/32)
(1988) mipi (/32)
(1989) miobase (/32)
(3137) cycle (/32)
(3138) time (/32)
(3139) instret (/32)
(3265) cycleh (/32)
(3266) timeh (/32)
(3267) instreth (/32)
```

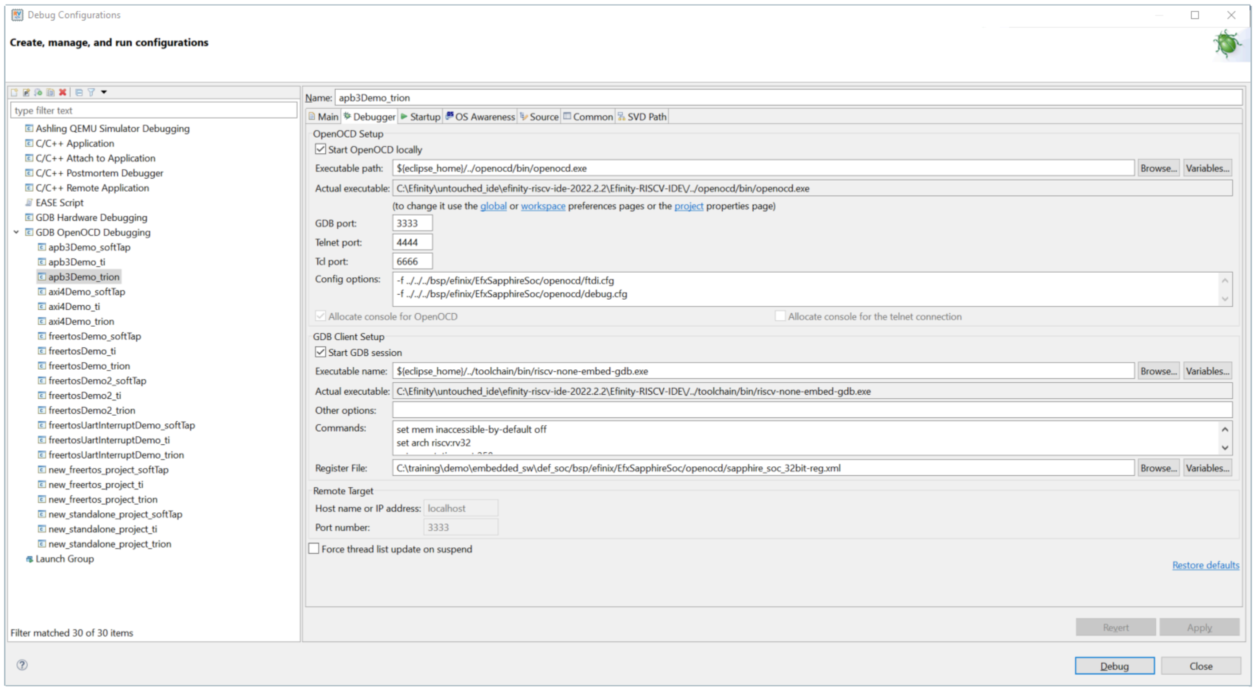
Register	Address	Value
spi0	0xF8014000	
cmd	0xF8014000	0x800000FF
rsp	0xF8014004	0x00000100
config	0xF8014008	0x00000000
interrupt	0xF801400C	0x00000000
clkDiv	0xF8014020	0x00000000
ssSetup	0xF8014024	0x00000000
ssHold	0xF8014028	0x00000000
ssDisable	0xF801402C	0x00000000
ssActiveHigh	0xF8014030	0x00000000
cmd32_0	0xF8014050	0x00000000
cmd32_1	0xF8014054	0x00000000
rsp32	0xF8014058	0x000000FF

CSR Register View

The CSR Register View displays all CSR values while you are debugging.

The IDE automatically points to the GDB Description file generated by the IP Manager when you generate the Sapphire SoC. If you want to point to a different **.xml** file, you may do it by going to **Debug** > **Debug Configurations** > **Debugger** > **GDB Client Setup** > **Register File** and browse to the new xml file. The default generated xml file is located in **/ bsp/efinix/EfxSapphireSoc/openocd/ sapphire_soc_32bit-reg.xml**.

Figure 31: Debug Configurations with Register File



Debug Configurations

Create, manage, and run configurations

type filter text

- Ashling QEMU Simulator Debugging
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- EASE Script
- GDB Hardware Debugging
- GDB OpenOCD Debugging
  - apb3Demo_softTap
  - apb3Demo_ti
  - apb3Demo_trion
  - axi4Demo_softTap
  - axi4Demo_ti
  - axi4Demo_trion
  - freertosDemo_softTap
  - freertosDemo_ti
  - freertosDemo_trion
  - freertosDemo2_softTap
  - freertosDemo2_ti
  - freertosDemo2_trion
  - freertosUartInterruptDemo_softTap
  - freertosUartInterruptDemo_ti
  - freertosUartInterruptDemo_trion
  - new_freertos_project_softTap
  - new_freertos_project_ti
  - new_freertos_project_trion
  - new_standalone_project_softTap
  - new_standalone_project_ti
  - new_standalone_project_trion
- Launch Group

Filter matched 30 of 30 items

Name: apb3Demo_trion

Main | Debugger | Startup | OS Awareness | Source | Common | SVD Path

OpenOCD Setup

[✓] Start OpenOCD locally

Executable path: ${eclipse_home}/../openocd/bin/openocd.exe

Actual executable: C:\Efinity\untouched_ide\efinity-riscv-ide-2022.2.2\Efinity-RISCV-IDE\../openocd/bin/openocd.exe

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 3333

Telnet port: 4444

Tcl port: 6666

Config options:
```
-f ../../../bsp/efinix/EfxSapphireSoc/openocd/ftdi.cfg
-f ../../../bsp/efinix/EfxSapphireSoc/openocd/debug.cfg
```

[✓] Allocate console for OpenOCD   [ ] Allocate console for the telnet connection

GDB Client Setup

[✓] Start GDB session

Executable name: ${eclipse_home}/../toolchain/bin/riscv-none-embed-gdb.exe

Actual executable: C:\Efinity\untouched_ide\efinity-riscv-ide-2022.2.2\Efinity-RISCV-IDE\../toolchain/bin/riscv-none-embed-gdb.exe

Other options:

Commands:
```
set mem inaccessible-by-default off
set arch riscv:rv32
```

Register File: C:\training\demo\embedded_sw\def_soc/bsp/efinix/EfxSapphireSoc/openocd/sapphire_soc_32bit-reg.xml

Remote Target

Host name or IP address: localhost

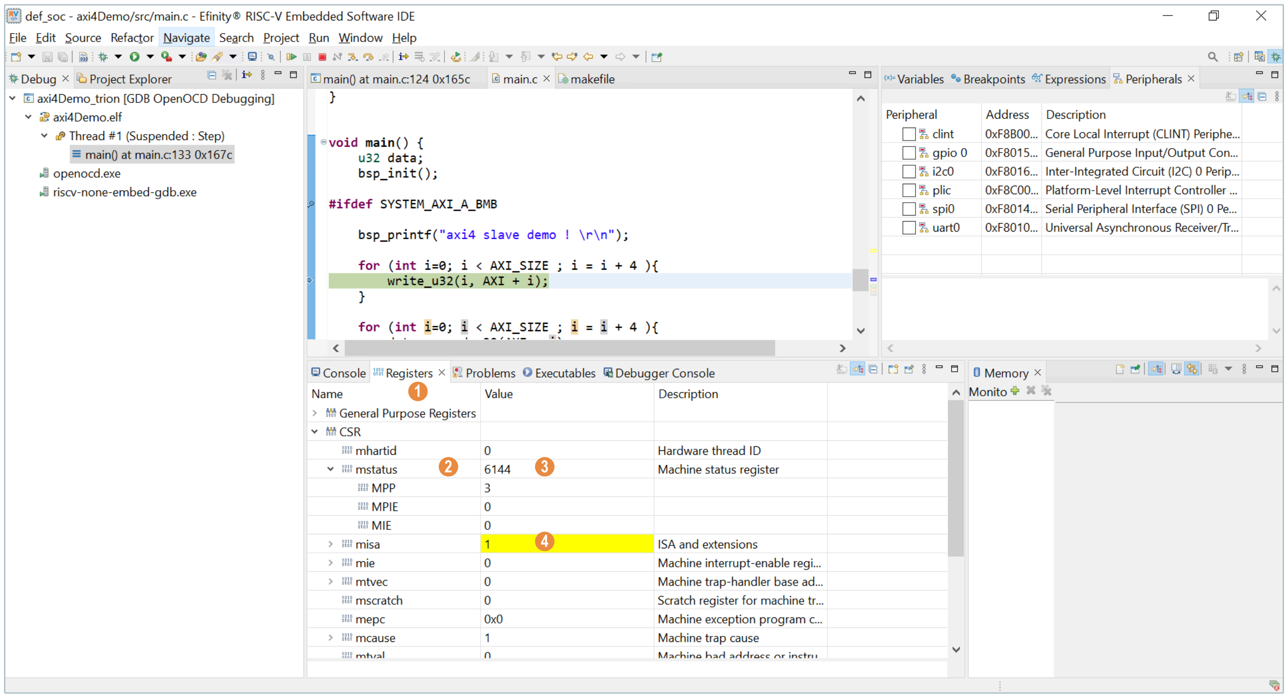
Port number: 3333

[ ] Force thread list update on suspend

When working with the CSR Register View, note that:

1. The CSR View is in the **Registers** tab.
2. All supported RISC-V CSRs are listed in the registers depending on the Sapphire SoC configuration (example: FPU enabled, MMU enabled).
3. Each CSR has its own value and description. CSRs are represented in bits and show up in drop-down menu.
4. The cell value is highlighted when the current value is different from the previous value.

Figure 32: Registers View


def_soc - axi4Demo/src/main.c - Efinity® RISC-V Embedded Software IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Project Explorer
axi4Demo_trion [GDB OpenOCD Debugging]
axi4Demo.elf
Thread #1 (Suspended : Step)
main() at main.c:133 0x167c
openocd.exe
riscv-none-embed-gdb.exe
main() at main.c:124 0x165c
main.c
makefile
Variables
Breakpoints
Expressions
Peripherals
Console
Registers
Problems
Executables
Debugger Console
Memory
Monito
1
2
3
4

```
}


void main() {
    u32 data;
    bsp_init();

#ifdef SYSTEM_AXI_A_BMB

    bsp_printf("axi4 slave demo ! \r\n");

    for (int i=0; i < AXI_SIZE ; i = i + 4 ){
        write_u32(i, AXI + i);
    }

    for (int i=0; i < AXI_SIZE ; i = i + 4 ){
```

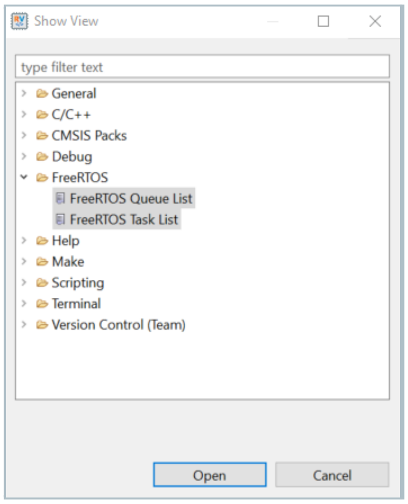
Peripheral	Address	Description
☐ clint	0xF8B00...	Core Local Interrupt (CLINT) Periphe...
☐ gpio 0	0xF8015...	General Purpose Input/Output Con...
☐ i2c0	0xF8016...	Inter-Integrated Circuit (I2C) 0 Perip...
☐ plic	0xF8C00...	Platform-Level Interrupt Controller ...
☐ spi0	0xF8014...	Serial Peripheral Interface (SPI) 0 Pe...
☐ uart0	0xF8010...	Universal Asynchronous Receiver/Tr...

Name	Value	Description
General Purpose Registers		
CSR		
mhartid	0	Hardware thread ID
mstatus	6144	Machine status register
MPP	3	
MPIE	0	
MIE	0	
misa	1	ISA and extensions
mie	0	Machine interrupt-enable regi...
mtvec	0	Machine trap-handler base ad...
mscratch	0	Scratch register for machine tr...
mepc	0x0	Machine exception program c...
mcause	1	Machine trap cause
mtval	0	Machine bad address or instru

FreeRTOS View

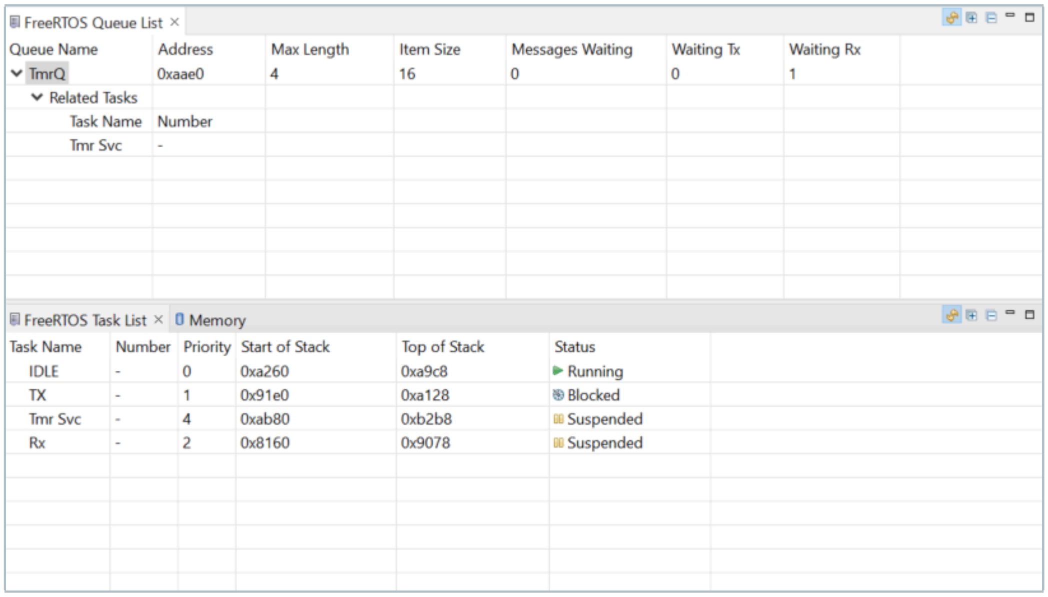
The FreeRTOS View includes **Queue** and **Task List**. These views help during your debugging; you can view the available tasks and their priority and status. It also allows you to view the maximum queue length, messages waiting, etc.

Figure 33: Show View Window



FreeRTOS **Queue** and **Task List** are not automatically instantiated during the debug session. To view it go to **Window > Show View > Others... > FreeRTOS > FreeRTOS Queue List/Free RTOS Task List** and click **Open**.

Figure 34: FreeRTOS Queue and Task List View



Queue Name	Address	Max Length	Item Size	Messages Waiting	Waiting Tx	Waiting Rx
TmrQ	0xaae0	4	16	0	0	1
Related Tasks						
Task Name	Number					
Tmr Svc	-					

Task Name	Number	Priority	Start of Stack	Top of Stack	Status
IDLE	-	0	0xa260	0xa9c8	Running
TX	-	1	0x91e0	0xa128	Blocked
Tmr Svc	-	4	0xab80	0xb2b8	Suspended
Rx	-	2	0x8160	0x9078	Suspended

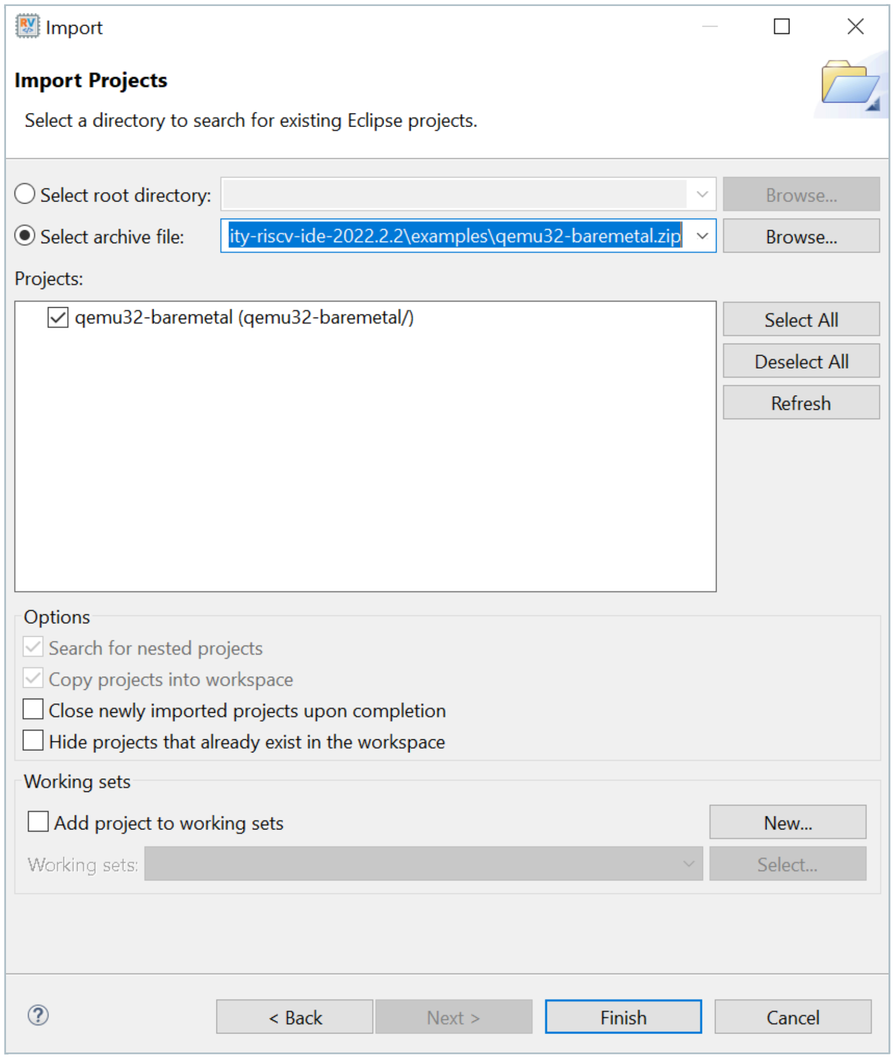
QEMU Emulator

The QEMU Emulator lets you try out your program without hardware. This feature is helpful for emulating your program before the hardware is ready.

To get started with the QEMU emulator, follow these steps:

1. Select **Import Projects...** in the **Project Explorer**.
2. In the **Import Projects** window, select **General > Existing Projects into Workspace > Next**.
3. Choose the **Select archive file > Browse**. Browse to the **<Efinity IDE Installation Path>/efinity-riscv-ide-2022.2.2/examples/qemu32-baremetal.zip**. Click **Open**.
4. Turn on for qemu32-baremetal project.

Figure 35: Importing QEMU Project



Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory:
[● Select archive file:] ity-riscv-ide-2022.2.2\examples\qemu32-baremetal.zip

Projects:
☑ qemu32-baremetal (qemu32-baremetal/)

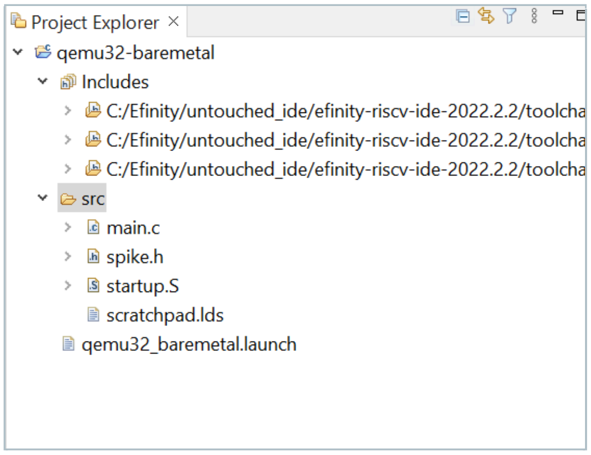
Options
☑ Search for nested projects
☑ Copy projects into workspace
☐ Close newly imported projects upon completion
☐ Hide projects that already exist in the workspace

Working sets
☐ Add project to working sets
Working sets:

5. Click **Finish**.

6. You can now browse through all source files in the project.

Figure 36: Project Explorer Pane showing qemu32-baremetal Project



Project Explorer
- qemu32-baremetal
  - Includes
    - C:/Efinity/untouched_ide/efinity-riscv-ide-2022.2.2/toolcha
    - C:/Efinity/untouched_ide/efinity-riscv-ide-2022.2.2/toolcha
    - C:/Efinity/untouched_ide/efinity-riscv-ide-2022.2.2/toolcha
  - src
    - main.c
    - spike.h
    - startup.S
    - scratchpad.lds
  - qemu32_baremetal.launch

7. To clean the project, right-click the project name and select **Clean Project**. Select **Build Project** to start building the program.
8. To start debugging the QEMU, right-click on the QEMU project and select **Debug As > Debug Configurations....**
9. In the **Debug Configurations**, select **qemu32_baremetal** in **Ashling_QEMU Simulator Debugging**.
10. Click **Debug** to start the debugging process.

Note: Windows Security Alert might prompt you to ask for permission to allow the QEMU machine emulator to run. Click **Allow access**.

Chapter 8

Boot Sequence

Contents:

When the SoC loads and runs your software application, there are several boot sequence scenarios, depending on where the application is stored. With a *bootloader*, the embedded program loads the user binary from secondary memory to primary memory during boot up. If your software application is small enough (less than 4 KB), you can embed it in the on-chip RAM. It is recommended to follow the procedure in Modify the Bootloader for building an embedded user application.

Figure 37: Boot Sequence Flow Chart

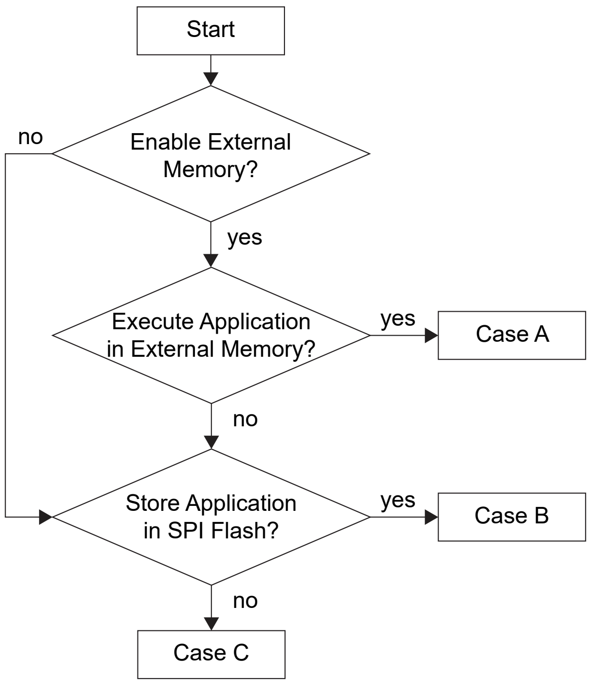

Start
Enable External Memory?
no
yes
Execute Application in External Memory?
yes
Case A
no
Store Application in SPI Flash?
yes
Case B
no
Case C

Table 20: User Application

Item	Case A	Case B	Case C
Bootloader needed?	Yes	Yes	No
Application storage	SPI flash	SPI flash	On-chip RAM
Execute location	External memory	On-chip RAM	On-chip RAM

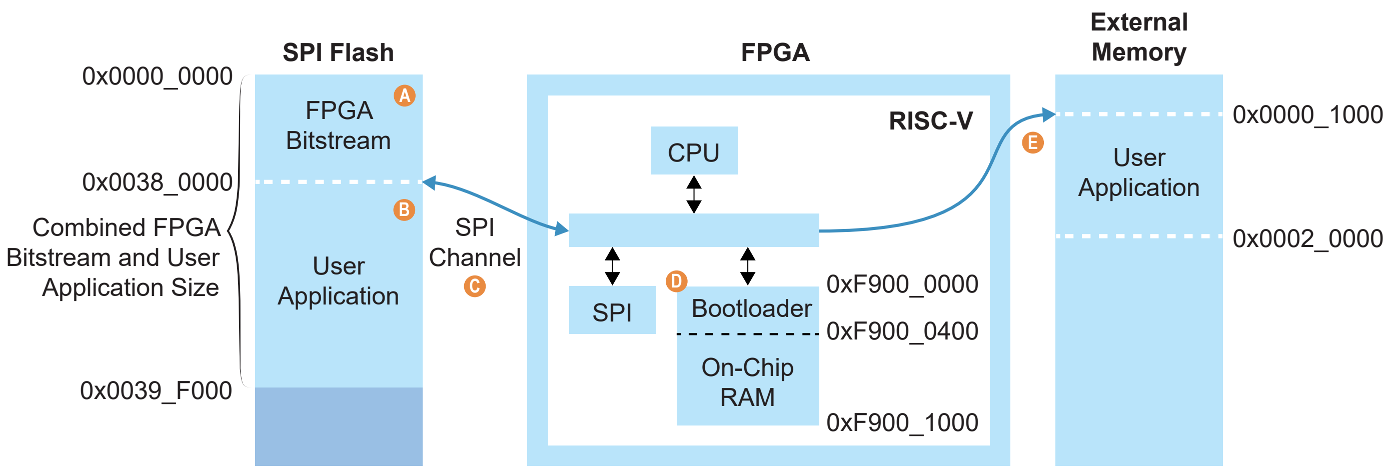
The following sections describe these cases in more detail.

The Sapphire SoC supports multiple cores; Booting Multiple Cores on page 70 describes the programming sequence.

Boot Sequence: Case A

The following figure shows the interaction of the FPGA, SPI flash, and external memory during booting.

Figure 38: Boot Sequence Diagram



Notes:

A. The bitstream has a default start address of 0x0000_0000 in the Efinity Programmer.
B. The application has a start address of 0x0038_0000.
C. The bootloader reads the SPI flash data from 0x0038_0000.
D. The CPU starts at 0xF900_0000. The default On-Chip RAM size is 4 KB.
E. The bootloader copies the SPI flash data to external memory address 0x0000_1000 and redirects the address to 0x0000_1000 for execution.

The system starts from the PC's `0xF900_0000`, which is the starting address of the on-chip RAM. The bootloader, which reads a larger user application from the SPI flash, is embedded by default.

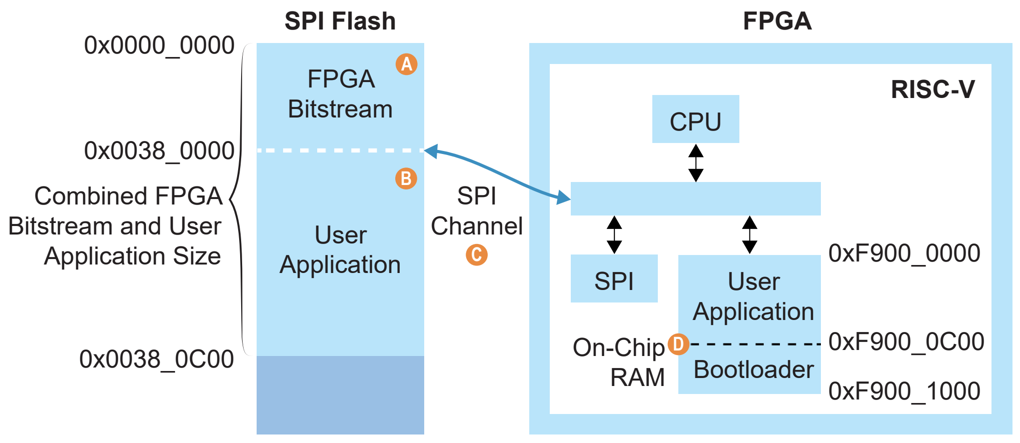
1. The PC starts at the system address `0xF900_0000` of the on-chip RAM.
2. The bootloader starts reading the SPI Flash address `0x38_0000` for the user application.
3. The bootloader writes the user application to external memory starting from system address `0x0000_1000`.
4. The bootloader finishes reading the user application from the SPI flash.
5. The PC jumps to system address `0x0000_1000` and starts to execute the user application.
6. All accesses remain in the external memory space, which is malloc() by default (unless you specify the on-chip RAM space in the software code)

Note: For RISC-V SoC booting from a flash device, the GPIOs for the SPI signals (`system_spi_*`) should have the **Register Option** > **register** set in the Interface Designer. Refer to the IP Manager generated example design to see how you should set up the SPI channel.

Boot Sequence: Case B

The following figure shows the interaction of the FPGA and SPI flash during booting.

Figure 39: Boot Sequence Diagram



Notes:
A. The bitstream has a default start address of 0x0000_0000 in the Efinity Programmer.
B. The application has a start address of 0x0038_0000.
C. The bootloader reads the SPI flash data from 0x0038_0000.
D. The bootloader copies the SPI flash data to the On-Chip RAM 0xF900_0000 and redirects the address to 0xF900_0C00 for execution.

- The last 1 KB of On-Chip RAM is reserved for the bootloader.
- The user application should not exceed the size 0xC00 which breaks the bootloader that is stored at 0xF900_0C00.

The boot sequence is:

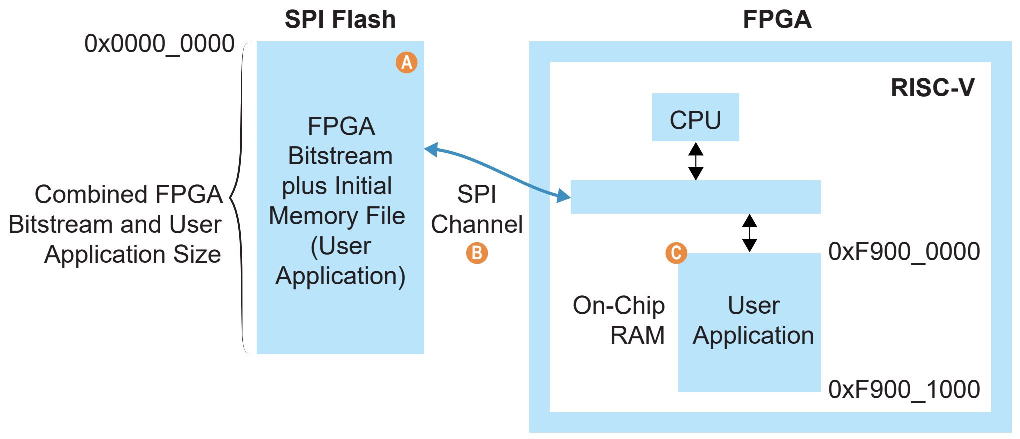
1. The PC starts at the system address `0xF900_0000` of the on-chip RAM and the PC jumps to `0xF900_0C00` for bootloader execution.
2. The bootloader starts reading the SPI Flash address `0x0038_0000`.
3. The bootloader writes the user application to On-Chip RAM starting from system address `0xF900_0000`.
4. The bootloader finishes reading the user application from the SPI flash.
5. The PC jumps to system address `0xF900_0000` and starts to execute the user application.

Note: For RISC-V SoC booting from a flash device, the GPIOs for the SPI signals (`system_spi_*`) should have the **Register Option** > **register** set in the Interface Designer. Refer to the IP Manager generated example design to see how you should set up the SPI channel.

Boot Sequence: Case C

The following figure shows the interaction of the FPGA and SPI flash during booting.

Figure 40: Boot Sequence Diagram


SPI Flash
FPGA
0x0000_0000
A
FPGA Bitstream plus Initial Memory File (User Application)
Combined FPGA Bitstream and User Application Size
SPI Channel
B
RISC-V
CPU
C
0xF900_0000
On-Chip RAM
User Application
0xF900_1000

Notes:

A. The bitstream has a default start address of 0x0000_0000 in the Efinity Programmer.
B. The application initial memory file is synthesized with the FPGA bitstream with address 0xF900_0000 for the RISC-V application.
C. The CPU starts at 0xF900_0000.

The boot sequence is:

1. The system starts from the PC's `0xF900_0000`, which is the starting address of the On-Chip RAM.
2. The user application is already compiled with the bitstream. It starts executing automatically from the FPGA's BRAM.

Booting Multiple Cores

If you configure multiple cores, the Sapphire SoC has two or more identical processors that share a common main memory and the same set of hardware I/Os. The processors can execute programs simultaneously; one processor can access the processed data or result from other processors because they are connected in a shared backplane.

With symmetric multi-processing (SMP), you can share the workload across all of the processors, resulting in less time to get a result compared to using a single-core processor. Thus, SMP helps improve overall system throughput and performance. The following flow chart explains how to do multi-core programming in a baremetal environment.

Figure 41: Boot Sequence for Multiple Cores

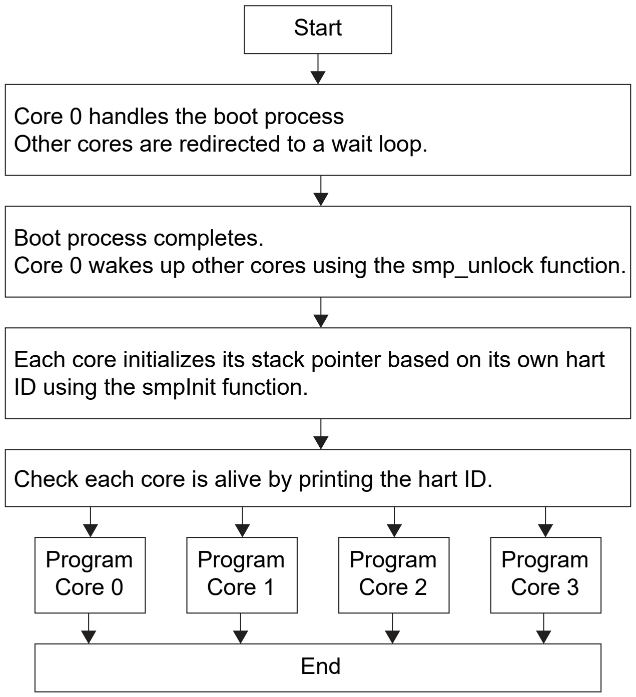

Start
Core 0 handles the boot process
Other cores are redirected to a wait loop.
Boot process completes.
Core 0 wakes up other cores using the smp_unlock function.
Each core initializes its stack pointer based on its own hart ID using the smpInit function.
Check each core is alive by printing the hart ID.
Program Core 0
Program Core 1
Program Core 2
Program Core 3
End

Table 21: SMP Helper Functions

File	Description
start.S	Functions to lock and unlock additional cores directory. To enable these functions, you should include following flag in your makefile: `CFLAGS+=-DSMP`
smpInit.S	Function to initialize the core.

These files are located in the **embedded_sw/standalone/common/** directory.

Each core has a dedicated interrupt ID for the PLIC to determine which core serves the external interrupts. Refer to **bsp/efinix/EfxSapphireSoc/include/soc.h** for the interrupt ID definitions for each core:

```
#define SYSTEM_PLIC_SYSTEM_CORES_0_EXTERNAL_INTERRUPT 0
#define SYSTEM_PLIC_SYSTEM_CORES_1_EXTERNAL_INTERRUPT 1
#define SYSTEM_PLIC_SYSTEM_CORES_2_EXTERNAL_INTERRUPT 2
#define SYSTEM_PLIC_SYSTEM_CORES_3_EXTERNAL_INTERRUPT 3
```

For the Clint timer interrupt, each core has a dedicated `MTIMECMP` register that you can use to set the trigger. You should provide the hart ID to the API to determine which core receives the interrupt from the Clint timer. For example:

```
clint_setCmp(BSP_CLINT, TriggerValue, HartID);
```

Each core has a dedicated floating-point unit, Linux memory management unit, and custom instruction interface, if these features are enabled in IP Manager.

Chapter 9

Create Your Own RTL Design

Contents:

After you have explored the Sapphire using the included example Efinity® project, you can use these tips to modify the design for your own use.

Note: Efinix recommends that you use the provided example design project as a starting point instead of creating a new project.

Target another FPGA

To change the design to target a different FPGA:

1. Edit the project to change the FPGA, package, and speed grade.
2. Update the interface design.
   a. Open the Interface Designer. The software prompts you that a device change was detected. Click **Update Design**. The Interface Designer opens and shows invalid assignments in the Message Viewer.
   b. Open the Resource Assigner.
   c. Click the instance name in the Message Viewer. The software jumps to that assignment in the Resource Assigner. Pick a new resource and press enter.
   d. Continue re-assigning pins until all assignments are valid.
   e. Generate a constraint file and close the Interface Designer.
3. Compile your modified design.

Target another Efinix Board

The Sapphire SoC BSP includes FTDI configuration files that specify the FTDI device VID and PID and board description for Efinix development boards.

When you configure the SoC, you can choose which Efinix board to target with the **Debug tab > Target OpenOCD** option. To target another board, change this option and re-generate the SoC files.

Table 22: Provided FTDI Configuration Files

File	Use for
ftdi.cfg	Trion development board
ftdi_ti.cfg	Titanium development board

If you do not want to re-generate the SoC, you can also change the target Efinix board manually by editing the **.cfg** file. However, if you want to target your own board, refer to Target Your Own Board on page 74 because the Efinix drivers specifically target the FTDI chips used on Efinix boards and your board will probably not have that chip.

To target a different Efinix development board manually, follow these steps with the development board attached to the computer:

1. Open the Efinity® Programmer.
2. Click the **Refresh USB Targets** button to display the board name in the **USB Target** drop-down list.
3. Make note of the board name.
4. In a text editor, open the **ftdi.cfg** or **ftdi_ti.cfg** file in the **embedded_sw/<SoC module>/bsp/efinix/EfxSapphireSoC/openocd** directory.
5. Change the `ftdi_device_desc` setting to match the board name. For example, use this code to change the name from Trion T120F324 Development Board to Trion T120F576 Development Board:

```
interface ftdi
ftdi_device_desc "Trion T120F324 Development Board"
#ftdi_device_desc "Trion T120F576 Development Board"
ftdi_vid_pid 0x0403 0x6010
```

6. Save the file.
7. Debug as usual in OpenOCD.

Target Your Own Board

For your own board, you generally use an FTDI cable or another JTAG cable or module. You can also use an FTDI chip on your board.

Using the FTDI Module or FTDI C232HM-DDHSL-0 JTAG cable

The Sapphire SoC also includes a configuration file for the FTDI Module or FTDI C232HM-DDHSL-0 JTAG cable (**external.cfg**), which bridges between your computer's USB connector and the JTAG signals on the FPGA. If you use external JTAG cable to connect your board to your computer, you can simply use this configuration file instead of **ftdi.cfg** or **ftdi_ti.cfg**. You may select your preffered external JTAG cable in **Debug and Linker Scripts Support** under the Debug tab in the IP Manager.

Note: Efinix does not recommend the FTDI Chip C232HM-DDHSL-0 programming cable due to the possibility of the FPGA not being recognized or the potential for programming failures. You are encourage to use FTDI chip FT2232H or FT4232H mini-module.

Note: Refer to Connect the FTDI Mini-Module on page 113 for instructions on using the cable.

Updating OpenOCD Configuration for External FTDI Cable

If you are using a custom FTDI cable to debug your board, you need to update the OpenOCD configuration file for external FTDI cable, **external.cfg** before launching the OpenOCD debugger.

Table 23: OpenOCD Confuguration File Setting for External FTDI Cable

Setting	Description
ftdi device_desc	FTDI device descriptor. The default setting is based on your selection of the debug cable during SoC configuration. You may find your cable description in the Device Manager (Windows) or lsusb (Linux) easily, i.e., ftdi device_desc "C232HM-DDHSL-0".
ftdi vid_pid	FTDI device vendor ID and product ID. The first hexadecimal represents the FTDI vendor ID while the second hexadecimal represents the FTDI product ID, i.e., ftdi vid_pid 0x403 0x6014.
ftdi layout_init	Initial values of the FTDI GPIO data and direction registers. The first hexadecimal represents data register while the second hexadecimal represents direction register. The values are based on the schematics of the adapter, i.e., ftdi_layout_init 0x0008 0x000b.
ftdi channel	FTDI device channel usage. Selects the channel of the FTDI device for operations, i.e., ftdi channel 1. The default is channel 0. **Note:** You can ignore this configuration if your FTDI device is single channel or uses channel 0.

Launching OpenOCD for Your Own Board

The standard launch scripts only support the following:

- ***_softTap**: External FTDI Cable + SoC soft JTAG Port
- ***_ti**: Standard Titanium FTDI + SoC hard JTAG Port
- ***_trion**: Standard Trion FTDI + SoC hard JTAG Port

To use an external FTDI Cable (i.e., C232HM-DDHSL-0 Programming Cable) with SoC hard JTAG Port (using device TAP Controller), you are required to modify the debug configuration to use the **external.cfg** to target the external FTDI cable and **ftdi.cfg** for Trion device or **ftdi_ti.cfg** for Titanium device.

The following steps guide you to adapt the existing gpioDemo launch configuration to utilize the external FTDI cable + SoC hard JTAG Port:

1. Select the preferred external JTAG Cable in the IP Manager when configuring the Sapphire SoC.
2. Import your desired project (i.e., gpioDemo) in the Efinity RISC-V Embedded Software IDE.
3. Right-click on the **gpioDemo_trion.launch** file in the Project Explorer pane to open the **Debug Configuration** setting.
4. Click on either **gpioDemo_ti** or **gpioDemo_trion** for either Titanium or Trion device.
5. In the **Debugger** tab, browse to the **OpenOCD Setup** section. There, you would see the **Config options** text box. Replace either the **ftdi_ti.cfg** or **ftdi.cfg** file depending on the launch scripts you have selected with **external.cfg**. Use your own configuration filename if you are using a different configuration file.
6. Click **Apply** and **Debug** to launch your application.

Note: Unexpected tap/device errors may occur in the console. You can remove the error by updating the CPUTAPID in the external **.cfg** file.

Using another JTAG Cable or Module

Generally, when debugging your own board you use a JTAG cable to connect your computer and the board. Therefore, you need to use the OpenOCD driver for that cable when debugging. OpenOCD includes a number of configuration files for standard hardware products. These files are located in the following directory:

openocd/build-win64/share/openocd/scripts/interface (Windows)

openocd/build-x86_64/share/openocd/scripts/interface (Linux)

You can also write your own configuration file if desired.

Follow these instructions when debugging with your own board:

1. Connect your JTAG cable to the board and to your computer.
2. Copy the OpenOCD configuration file for your cable to the **bsp/efinix/EfxSapphireSoc/openocd** directory.
3. Follow the instructions for debugging, except target your configuration file instead of the **ftdi.cfg** (Trion) or **ftdi_ti.cfg** (Titanium) file.

```
-f <path>/bsp/efinix/EfxSapphireSoc/openocd/<my cable>.cfg
```

Using an FTDI Chip on your Board

When you configure the Sapphire SoC in the IP Configuration wizard, choose **Target OpenOCD > Custom**. Then, specify your board name. When you generate the SoC, the **ftdi.cfg** file is populated with your board name. Edit the file for your board's VID and PID.

Create a Custom AXI4 Slave Peripheral

When you generate an example design for the Sapphire SoC, the IP Manager creates an example AXI4 peripheral and software code that you can use as a template to create your own peripheral. This example uses the simple dual-port RAM design to write to and read from the CPU through the AXI4 interface.

- Refer to the `axi4_slave` module in **design_modules.v** in the **T120F324_devkit**, **Ti60F225_devkit**, or **Ti180J484_devkit** directory for the RTL design.
- Refer to **main.c** in the **embedded_sw/<SoC module>/software/standalone/axiDemo/src** directory for the C code.

Create a Custom APB3 Peripheral

When you generate an example design for the Sapphire SoC, the IP Manager creates an APB3 peripheral and software code that you can use as a template to create your own peripheral. This simple example shows how to implement an APB3 slave wrapper.

- Refer tothe `apb3_slave` module in **design_modules.v** in the **T120F324_devkit**, **Ti60F225_devkit**, or **Ti180J484_devkit** directory for the RTL design.
- Refer to **main.c** in the **embedded_sw/<SoC module>/software/standalone/apb3Demo/src** directory for the C code.

Use another DDR DRAM Module (Trion Only)

The Trion® T120 BGA324 Development Board has an LPDDR3 DRAM module with 256 Mbits x 16 bits supporting up to 4 Gb. If you want to target a different module, you need to update the DDR block in the Interface Designer to reflect the specifications for your module.

Note: Refer to the Trion DDR DRAM Block User Guide for more information on changing the DDR block.

Use the I²C Interface for DDR Calibration

You can use the I²C interface to calibrate and reset the DDR DRAM interface on the Trion® T120 BGA324 Development Board or Trion® T120 BGA576 Development Board. If you want to use calibration:

1. In the Efinity Interface Designer, select the DDR block and turn on **Enable Control** in the Block Editor's **Control** tab. Save.
2. In your RTL design, connect the I²C interface to the DDR block's I²C interface. See the following example code:

```
// top level port
output  ddr_inst1_CFG_SCL_IN,
output  ddr_inst1_CFG_SDA_IN,
input   ddr_inst1_CFG_SDA_OEN,

// assignment
assign ddr_inst1_CFG_SDA_OEN_workaround = ddr_inst1_CFG_SDA_OEN;
assign ddr_inst1_CFG_SDA_IN = system_i2c_2_io_sda_write &&
 ddr_inst1_CFG_SDA_OEN_workaround;
assign ddr_inst1_CFG_SCL_IN = system_i2c_2_io_scl_write;

assign system_i2c_2_io_sda_read  = system_i2c_2_io_sda_write &&
 ddr_inst1_CFG_SDA_OEN_workaround;
assign system_i2c_2_io_scl_read  = system_i2c_2_io_scl_write;

// SoC connection
.system_i2c_2_io_sda_write          (system_i2c_2_io_sda_write),
.system_i2c_2_io_sda_read           (system_i2c_2_io_sda_read),
.system_i2c_2_io_scl_write          (system_i2c_2_io_scl_write),
.system_i2c_2_io_scl_read           (system_i2c_2_io_scl_read),
```

3. Connect the DDR control pins in the Interface Designer's DDR Block Editor.

Remove Unused Peripherals from the RTL Design

The Sapphire SoC includes a variety of peripherals. if you do not want to use a peripheral, simply remove the signal name from within the parentheses () in the SapphireSoc SapphireSoc_inst definition in the top-level Verilog HDL file. For example, the SoC instantiation has these signals:

```
.system_i2c_0_io_sda_write          (system_i2c_0_io_sda_write),
.system_i2c_0_io_sda_read           (system_i2c_0_io_sda_read),
.system_i2c_0_io_scl_write          (system_i2c_0_io_scl_write),
.system_i2c_0_io_scl_read           (system_i2c_0_io_scl_read),
```

To disable I²C `0`, remove the signal name in () as shown below:

```
.system_i2c_0_io_sda_write         (),
.system_i2c_0_io_sda_read          (),
.system_i2c_0_io_scl_write         (),
.system_i2c_0_io_scl_read          (),
```

Chapter 10

Create Your Own Software

Contents:

Now that you have explored the methodology for designing with the Sapphire SoC, you can develop your own software applications.

Deploying an Application Binary

During normal operation, your user binary application file (**.bin**) is stored in a SPI flash device. When the FPGA powers up, the Sapphire SoC copies your binary file from the SPI flash device to the DDR DRAM module, and then begins execution.

For debugging, you can load the user binary (**.elf**) directly into the Sapphire SoC using the OpenOCD Debugger. After loading, the binary executes immediately.

Note: The settings in the linker prevent user access to the address. This setting allows the embedded bootloader to work properly during a system reset after the user binary is executed but the FPGA is not reconfigured.

Boot from a Flash Device

When the FPGA boots up, the Sapphire SoC copies your binary application file from a SPI flash device to the external memory module, and then begins execution. The SPI flash binary address starts at 0x0038_0000.

To boot from a SPI flash device:

1. Power up your board. The FPGA loads the configuration image from the on-board flash device.
2. When configuration completes, the bootloader begins cloning a 124 KByte user binary file from the flash device at physical address 0x0038_0000 to an off-chip DRAM logical address of 0x0000_1000.

   Note: It takes ~300 ms to clone a 124 KByte user binary (this is the default size).
3. The Sapphire SoC jumps to logical address 0x0000_1000 to execute the user binary.

Note: Refer to Boot Sequence on page 67 for other possible boot scenarios.

Boot from the OpenOCD Debugger

To boot from the OpenOCD debugger:

1. Power up your board. The FPGA loads the configuration image from the on-board flash device.
2. Launch Efinity RISC-V Embedded Software IDE.
3. The user binary is suspended on boot up. Click the Resume button to start the program.

Note: Refer to **Debug with the OpenOCD Debugger** on page 51 for complete instructions.

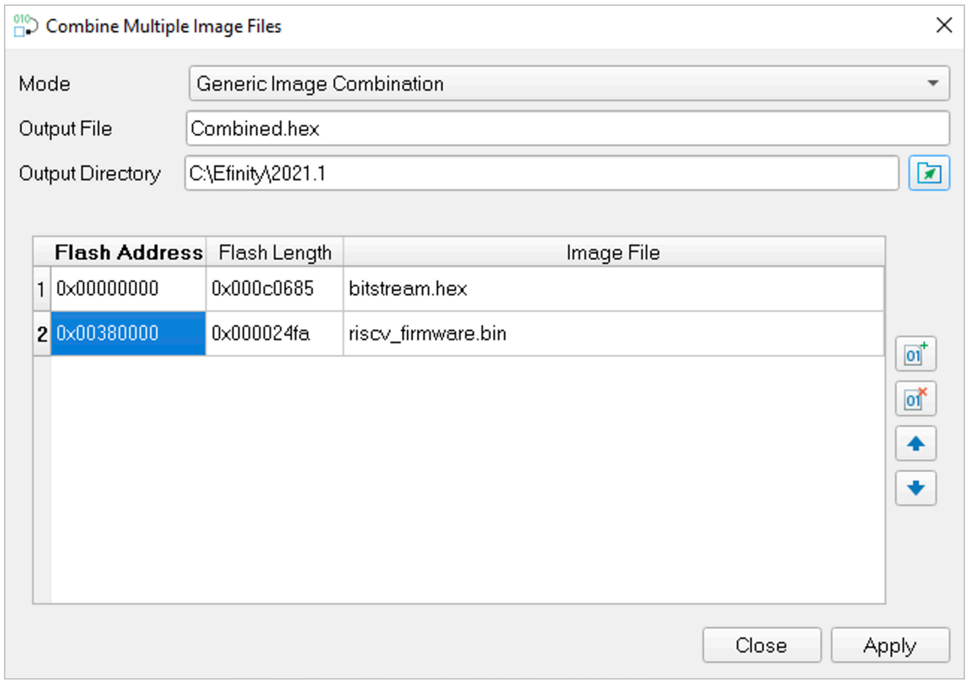
Copy a User Binary to Flash (Efinity Programmer)

To boot from a flash device, you need to copy the application binary to the flash. If you want to store the binary in the same flash device that holds the FPGA bitstream, you can simply combine the two files and download the combined file to the flash device with the Efinity Programmer.

1. Open the Efinity Programmer.
2. Click the **Combine Multiple Image Files** button.
3. Choose **Mode** > **Generic Image Combination**.
4. Enter a name for the combined file in **Output File**.
5. Click the Add Image button. The **Open Image File** dialog box opens.
6. Browse to the bitstream **.hex** file, select it, and click **Open**.
7. Click the Add Image button a second time.
8. Browse to the RISC-V application binary **.bin** file, select it, and click **Open**.
9. Specify the **Flash Address** as follows:

File	Address
Bitstream	0x00000000
RISC-V application binary	0x00380000

Figure 42: Combining a Bitstream and RISC-V Application Binary



Mode: Generic Image Combination
Output File: Combined.hex
Output Directory: C:\Efinity\2021.1

	Flash Address	Flash Length	Image File
1	0x00000000	0x000c0685	bitstream.hex
2	0x00380000	0x000024fa	riscv_firmware.bin

10. Click **Apply**. The software creates the combined **.hex** file in the specified **Output Directory** (the default is the project **outflow** directory).
11. Program the flash with the **.hex** file using **Programming Mode > SPI Active**.
12. Reset the FPGA or power cycle the board.

Note: You can also use two terminals to copy the application binary to flash. Refer to Appendix: Copy a User Binary to the Flash Device (2 Terminals) on page 192.

About the Board Specific Package

The board specific package (BSP) defines the address map and aligns with the Sapphire SoC hardware address map. The BSP files are located in the **bsp/efinix/EfxSapphireSoC** subdirectory.

Table 24: BSP Files

File or Directory	Description
app	Files used by the example software and bootloader.
include\soc.mk	Supported instruction set.
include\soc.h	Defines the system frequency and address map.
linker\default.ld	Linker script for the main memory address and size.
linker\default_i.ld	Linker script for the internal memory address and size.
linker\bootloader.ld	Linker script for the bootloader address and size.
openocd	OpenOCD configuration files.

Address Map

Because the address range might be updated, Efinix recommends that you always refer to the parameter name when referencing an address in firmware, not by the actual address. The parameter names and address mappings are defined in **/embedded_sw/**<module>**/bsp/efinix/EfxSapphireSoc/include/soc.h**.

Note: If you need to update the address map, use the IP Configuration wizard to change the addressing and then re-generate the SoC. Using this method keeps the software **soc.h** and FPGA netlist definitions aligned.

Table 25: Default Address Map, Interrupt ID, and Cached Channels

The AXI user slave channel is in a cacheless region (I/O) for compatibility with AXI-Lite.

Device	Parameter	Size	Interrupt ID	Region
Off-chip memory	SYSTEM_DDR_BMB	4 MB to 3.5 GB	-	Cache
GPIO 0	SYSTEM_GPIO_0_IO_CTRL	4 K	[0]: 12 [1]: 13	I/O
GPIO 1	SYSTEM_GPIO_1_IO_CTRL	4 K	[0]: 14 [1]: 15	I/O
I^2C 0	SYSTEM_I2C_0_IO_CTRL	4 K	8	I/O
I^2C 1	SYSTEM_I2C_1_IO_CTRL	4 K	9	I/O
I^2C 2	SYSTEM_I2C_2_IO_CTRL	4 K	10	I/O
Core timer	SYSTEM_CLINT_CTRL	4 K	-	I/O
PLIC	SYSTEM_PLIC_CTRL	4 K	-	I/O
SPI master 0	SYSTEM_SPI_0_IO_CTRL	4 K	4	I/O
SPI master 1	SYSTEM_SPI_1_IO_CTRL	4 K	5	I/O
SPI master 2	SYSTEM_SPI_2_IO_CTRL	4 K	6	I/O
UART 0	SYSTEM_UART_0_IO_CTRL	4 K	1	I/O
UART 1	SYSTEM_UART_1_IO_CTRL	4 K	2	I/O
UART 2	SYSTEM_UART_2_IO_CTRL	4 K	3	I/O
User timer 0	SYSTEM_USER_TIMER_0_CTRL	4 K	19	I/O
User timer 1	SYSTEM_USER_TIMER_1_CTRL	4 K	20	I/O
User timer 2	SYSTEM_USER_TIMER_2_CTRL	4 K	21	I/O
User peripheral 0	IO_APB_SLAVE_0_CTRL	4 K to 1 MB	-	I/O
User peripheral 1	IO_APB_SLAVE_1_CTRL	4 K to 1 MB	-	I/O
User peripheral 2	IO_APB_SLAVE_2_CTRL	4 K to 1 MB	-	I/O
User peripheral 3	IO_APB_SLAVE_3_CTRL	4 K to 1 MB	-	I/O
User peripheral 4	IO_APB_SLAVE_4_CTRL	4 K to 1 MB	-	I/O
On-chip BRAM	SYSTEM_RAM_A_BMB	1 - 512 KB	-	Cache
AXI user slave	SYSTEM_AXI_A_BMB	1 K to 256 MB	-	I/O

Device	Parameter	Size	Interrupt ID	Region
External interrupt	–	–	[0]: 16 [1]: 17 [2]: 22 [3]: 23 [4]: 24 [5]: 25 [6]: 26 [7]: 27	I/O

When accessing the addresses in the I/O region, type casting the pointer with the keyword **volatile**. The compiler recognizes this as a memory-mapped I/O register without optimizing the read/write access. An example of the casting is shown by the following command:

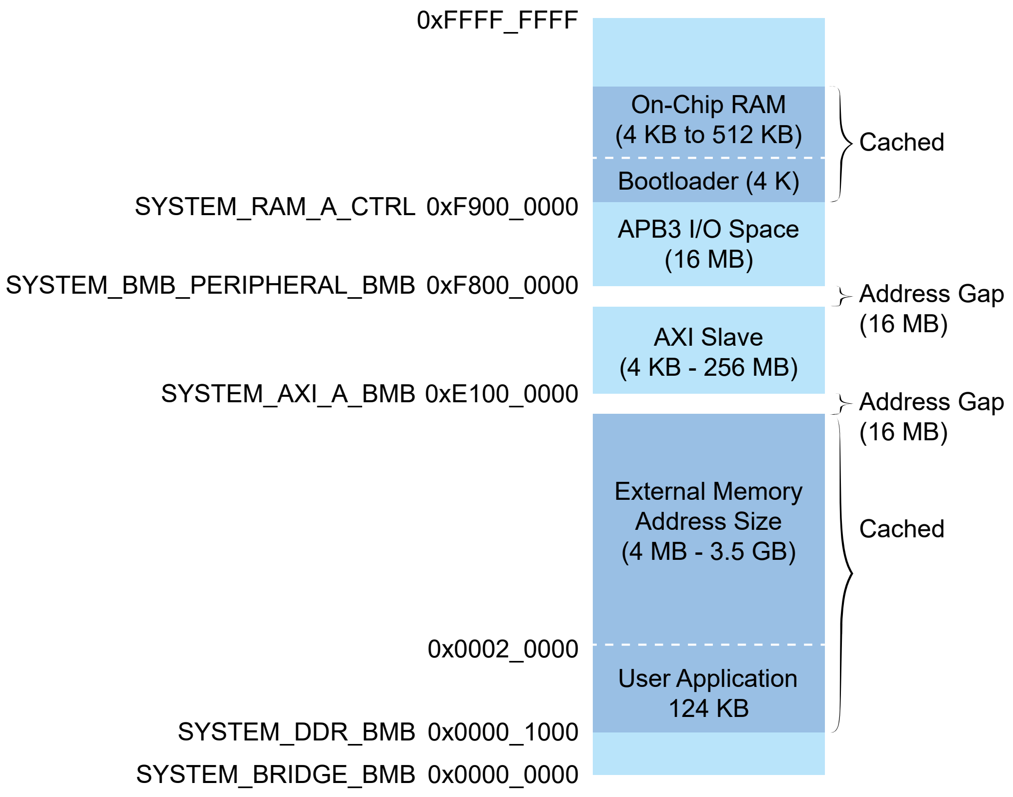
```
*((volatile u32*) address);
```

For the cached regions, the burst length is equivalent to an AXI burst length of 8. For the I/O region, the burst length is equivalent to an AXI burst length of 1. The AXI user slave is compatible with AXI-Lite by disconnecting unused outputs and driving a constant 1 to the input port.

Note: The RISC-V GCC compiler does not support user address spaces starting at 0x0000_0000.

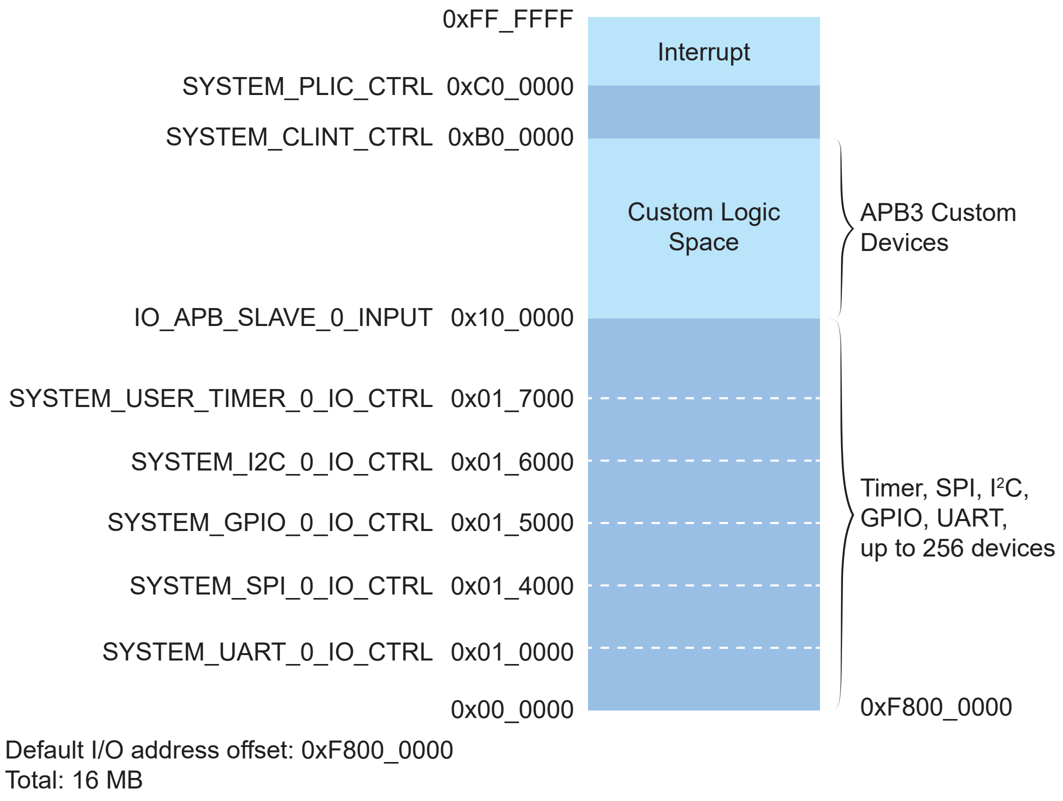
The following figure shows the default address map and the corresponding software parameters for modules in the memory space.

Figure 43: Sapphire Memory Space


0xFFFF_FFFF
On-Chip RAM (4 KB to 512 KB)
Bootloader (4 K)
Cached
SYSTEM_RAM_A_CTRL 0xF900_0000
APB3 I/O Space (16 MB)
SYSTEM_BMB_PERIPHERAL_BMB 0xF800_0000
Address Gap (16 MB)
AXI Slave (4 KB - 256 MB)
SYSTEM_AXI_A_BMB 0xE100_0000
Address Gap (16 MB)
External Memory Address Size (4 MB - 3.5 GB)
Cached
0x0002_0000
User Application 124 KB
SYSTEM_DDR_BMB 0x0000_1000
SYSTEM_BRIDGE_BMB 0x0000_0000

The following figure shows the default address map and the corresponding software parameters for I/O.

Figure 44: Sapphire I/O Space


0xFF_FFFF
Interrupt
SYSTEM_PLIC_CTRL 0xC0_0000
SYSTEM_CLINT_CTRL 0xB0_0000
Custom Logic Space
APB3 Custom Devices
IO_APB_SLAVE_0_INPUT 0x10_0000
SYSTEM_USER_TIMER_0_IO_CTRL 0x01_7000
SYSTEM_I2C_0_IO_CTRL 0x01_6000
SYSTEM_GPIO_0_IO_CTRL 0x01_5000
SYSTEM_SPI_0_IO_CTRL 0x01_4000
SYSTEM_UART_0_IO_CTRL 0x01_0000
0x00_0000
Timer, SPI, I²C, GPIO, UART, up to 256 devices
0xF800_0000
Default I/O address offset: 0xF800_0000
Total: 16 MB

Example Software

To help you get started writing software for the Sapphire, Efinix provides a variety of example software code that performs functions such as communicating through the UART, controlling GPIO interrupts, performing Dhrystone benchmarking, etc. Each example includes a **makefile** and **src** directory that contains the source code.

Note: Many of these examples display messages on a UART. Refer to the following topics for information on attaching a UART module and connecting to it in a terminal:

Learn how to attach a UART module.

Learn how to open serial terminal in Efinity RISC-V Embedded Software IDE and connect to the UART module.

Table 26: Example Software Code

Directory	Description
apb3Demo	This example shows how to implement an ABP3 slave.
Axi4Demo	This example illustrates how to implement a user AXI4 slave.
bootloader	This software is the bootloader for the system.
common	Provides linking for the makefiles.
compatibilityDemo	This example shows how to migrate a software application from the Sapphire SoC v1.1 to v2.0 or higher.
coreTimerInterruptDemo	This example shows how to use the clint timer with interrupt.
coremark	This example is a synthetic computing benchmark program.
customInstructionDemo	This example illustrates how to implement a custom instruction.
dCacheFlushDemo	This example illustrates how to invalidate the data cache.
dhrystone	This example is a synthetic computing benchmark program.
driver	This directory contains the system drivers for the peripherals (I^2C, UART, SPI, etc.). Refer to API Reference on page 135 for details.
FreeRTOS	This example shows the example software projects targeting the RTOS.
freertosUartInterruptDemo	This example illustrates the operation that uses UART interrupt with software execution done in the Free RTOS software framework.
fpuDemo	This example shows how to use the floating-point unit.
gpioDemo	This example shows how to control the GPIO and its interrupt.
iCacheFlushDemo	This example illustrates how to invalidate the instruction cache.
inlineASMDemo	This example illustrates utilizing the inline assembly feature.
i2cDemo	This example shows how to connect to an MCP4725 digital-to-analog converter (DAC) using an I^2C peripheral.
i2CEepromDemo	This example illustrates how to use I^2C driver to communicate with the on-board EEPROM device, AT24C01 on either the T120F324 or T120F576 Development Kit.
i2cMasterDemo	This example illustrates how to effectively utilize the Sapphire SoC as an I^2C master.

Directory	Description
i2cSlaveDemo	This example illustrates how to effectively utilize the Sapphire SoC as an I^2C slave.
memTest	This code performs a memory address and data test.
nestedInterruptDemo	This example shows how to set a higher priority to an interrupt routine, which allows the CPU to prioritize the task execution instead of other interrupts.
openocdServer	This folder provides launch scripts for the openOCD server. Refer to Debug - Multiple Cores on page 54 for details.
semihostingDemo	This examples shows how to use write and read debug messages through semihosting.
smpDemo	This example illustrates how to use multiple cores to execute the Tiny encryption algorithm in parallel.
spiDemo	This code reads the device ID and JEDEC ID of a SPI flash device and echoes the characters on a UART.
spiReadFlash	This example shows how to read from a SPI flash device.
spiWriteFlash	This example shows how to write to a SPI flash device.
uartEchoDemo	This example shows how to use the UART.
uartInterruptDemo	This exmple shows how to use a UART interrupt.
userInterruptDemo	This example demonstrates user interrupts with UART messages.
userTimerDemo	This example shows how to use the user timer with interrupt.

Axi4Demo Design

This example (**axi4Demo** directory) performs a write and read test for the internal BRAM that is attached to an AXI interface. First, the software writes to the internal BRAM through the AXI interface. Next, it reads back the data and compares it to the expected value. If the data is correct, the software writes `Passed` to a UART terminal

The AXI bus interrupt pin triggers a software interrupt when write data to the AXI bus is 0xABCD. The design displays these messages in a UART terminal:

```
axi4 slave demo !
Passed!
axi4 slave interrupt demo !
Entered AXI Interrupt Routine, Passed!
```

apb3Demo

This simple software design illustrates how to use an APB3 slave peripheral.

The APB3 slave is attached to a pseudorandom number generator. When you run the application, the Sapphire SoC programs the APB3 slave to stop generating a new random number and reads the last random number generated. The test passes if the returned data is a non-zero value.

```
apb3 slave 0 demo !
Random number:
0xE1ECA84A
Passed!
```

compatibilityDemo

This example (**compatibilityDemo** directory) shows how to migrate an application from the Sapphire Soc v1.1 to v2.0. To run your previously developed software applications on v2.0, include **compability.h** and **bsp.h** at the top of your code. These files map the v2.0 code and driver changes to the old SoC version.

This demo runs a machine timer with interrupt, which was available in v1.1. See **Migrating to the Sapphire SoC v2.0 from a Previous Version** on page 116 for additional migration details.

```
Hello world
BSP_MACHINE_TIMER 0
BSP_MACHINE_TIMER 1
BSP_MACHINE_TIMER 2
BSP_MACHINE_TIMER 3
```

coreTimerInterruptDemo

This demo (**coreTimerInterruptDemo** directory) shows how to use the core timer and its interrupt function. This demo configures the core timer to generate an interrupt every 1 second. It prints messages on a terminal when the SoC is interrupted by the core timer.

```
core timer interrupt demo !
core timer interrupt 0
core timer interrupt 1
core timer interrupt 2
core timer interrupt 3
core timer interrupt 4
core timer interrupt 5
core timer interrupt 6
core timer interrupt 7
core timer interrupt 8
core timer interrupt 9
```

coremark

This code (**coremark** directory) is a benchmark application to measure CPU performance. The final score is calculated based on the result of algorithm processing (e.g., list processing, matrix manipulation, state machine, and CRC). This application is configured to run 2,000 iterations with a runtime of approximately 20s.

When you run the application, it displays information similar to the following in a terminal:

```
coremark app is running, please wait...
2K performance run parameters for coremark.

CoreMark Size    : 666

Total ticks      : 1117963326

Total time (secs): 11.179633

Iterations/Sec   : 178.896745

Iterations       : 2000

Compiler version : GCC8.3.0

Compiler flags   : -o3

Memory location  : STACK

seedcrc          : 0xe9f5

[0]crclist       : 0xe714

[0]crcmatrix     : 0x1fd7

[0]crcstate      : 0x8e3a

[0]crcfinal      : 0x4983

Correct operation validated. See README.md for run and reporting rules.

CoreMark 1.0 : 178.896745 / GCC8.3.0 / -o3
 / STACK
```

customInstructionDemo

This demo (**customInstructionDemo** directory) shows how to use a custom instruction to accelerate the processing time of an algorithm. It demonstrates how performing an algorithm in hardware can provide significant acceleration vs, using software only. This demo uses the Tiny encryption algorithm to encrypt two 32-bit unsigned integers with a 128-bit key. The encryption is 1,024 cycles.

The demo first processes the algorithm with a custom instruction, and then processes the same algorithm in software. Timestamps indicate how many clock cycles are needed to output results. If both methods output the same results, `Passed!` prints on a terminal. Otherwise, it prints `Failed`.

```
custom instruction demo !
please enable custom instruction plugin to run this demo

custom instruction processing clock cycles:1093
software processing clock cycles:36126

Passed!
```

dCacheFlushDemo

This example (**dCacheFlushDemo** directory) illustrates how to invalidate the data cache. The data cache invalidation is critical to ensure the coherency between the cache and the main memory, ensuring that the CPU fetches the most up-to-date data. In the example, a value of 0xaa550000 is directly written into the memory using a pointer at address 0x00100000. The data increments by 1 for each address increment of 4. The writing process continues until address 0x0010001C.

Next, the data is overwritten by the memory_checker module, triggered by the apb3 module. The overwritten data ranges from 0xaa001100 to 0xaa001107. When printing the data from addresses 0x00100000 to 0x0010001C, it can be observed that the data remains unchanged. This is because the data is still being fetched from the cache memory and not from the main memory. To address this, the data cache invalidation is performed before reading the data again. This ensures that the data is updated.

By following this process, the CPU fetches the most up-to-date data from the main memory and maintains coherency with the cache. The design displays these messages in a UART terminal:

```
d-cache clearing demo !
Value at address 0x00100000 with value of 0xaa550000
Value at address 0x00100004 with value of 0xaa550001
Value at address 0x00100008 with value of 0xaa550002
Value at address 0x0010000c with value of 0xaa550003
Value at address 0x00100010 with value of 0xaa550004
Value at address 0x00100014 with value of 0xaa550005
Value at address 0x00100018 with value of 0xaa550006
Value at address 0x0010001c with value of 0xaa550007

Overwrite values using custom logic..
Done!!

Read some addresses again
Value at address 0x00100000 with value of 0xaa550000
Value at address 0x00100004 with value of 0xaa550001
Value at address 0x00100008 with value of 0xaa550002
Value at address 0x0010000c with value of 0xaa550003
Value at address 0x00100010 with value of 0xaa550004
Value at address 0x00100014 with value of 0xaa550005
Value at address 0x00100018 with value of 0xaa550006
Value at address 0x0010001c with value of 0xaa550007
Values are still same, CPU took them from cache!

This time clear the cache before read the same address
You can clear single line cache or flush the whole D-cache

Value at address 0x00100000 with value of 0xaa001100
Value at address 0x00100004 with value of 0xaa001101
Value at address 0x00100008 with value of 0xaa001102
Value at address 0x0010000c with value of 0xaa001103
Value at address 0x00100010 with value of 0xaa001104
Value at address 0x00100014 with value of 0xaa001105
Value at address 0x00100018 with value of 0xaa001106
Value at address 0x0010001c with value of 0xaa001107
Now you have updated new values from external memory!!
```

By default, the example design performs a full data cache invalidation. However, if you want to use the line cache invalidation, you can make the following changes:

Comment out the line:

```
data_cache_invalidate_all();
```

Uncomment the lines:

```
//    for(j-0; j<NUM; j=j+1){
//      data cache_invalidate_address(LOC_MEM+);
//    }
```

dhrystone Example

The Dhrystone example (**dhrystone** directory) is a classic benchmark for testing CPU performance. When you run this application, it performs dhrystone benchmark testing and displays messages and results on a UART terminal.

The following code shows example results:

```
Dhrystone Benchmark, Version C, Version 2.2
   Program compiled without 'register' attribute
   Using time(), HZ=12000000
   Trying 500 runs through Dhrystone:
   Final values of the variables used in the benchmark:
   Int_Glob:    5
   should be:   5
   Bool_Glob:   1
   should be:   1
....
   Enum_Loc:    1
   should be:   1
   Str_1_Loc:   DHRYSTONE PROGRAM, 1'ST STRING
   should be:   DHRYSTONE PROGRAM, 1'ST STRING
   Str_2_Loc:   DHRYSTONE PROGRAM, 2'ND STRING
   should be:   DHRYSTONE PROGRAM, 2'ND STRING

   Microseconds for one run through Dhrystone: 40
   Dhrystones per Second:                      24472
   User_Time : 245176
   Number_Of_Runs : 500
   HZ : 12000000
   DMIPS per Mhz:                              1.16
```

FreeRTOS Examples

The Sapphire SoC supports the popular FreeRTOS real-time operating system, and includes example software projects targeting the RTOS. For more details on using FreeRTOS, go to their web site at **https://www.freertos.org**.

Download the FreeRTOS

The freeRTOS examples require you to download FreeRTOS.

1. Download the FreeRTOS zip file from **https://www.freertos.org**. Efinix recommends using FreeRTOS v10.4.1.
2. Unzip the folder to any directory.
3. Point to the folder when importing existing project or creating new project.

After you have downloaded the FreeRTOS, you use the software projects in the same manner as the other example software.

freertosDemo

This example shows how the FreeRTOS schedular handles two program executions using task and queue allocation. Generally, the FreeRTOS queue is used as a thread FIFO buffer and for intertask communication. This example creates two tasks and one queue; the queue sends and receives traffic. The receive traffic (or receive queue) blocks the program execution until it receives a matching value from the send traffic (or send queue).

Tasks in the send queue sit in a loop that blocks execution for 1,000 miliseconds before sending the value `100` to the receive queue. Once the value is sent, the task loops, i.e., blocks for another 1,000 miliseconds.

When the receive queue receives the value `100`, it begins executing its task, which sends the message `Blink` to the UART peripheral and toggles an LED on the development board.

```
Hello world, this is FreeRTOS
Blink
Blink
Blink
```

freertosDemo2

This example shows how FreeRTOS schedular handles two program executions using a binary semaphore. The semaphore holds the hardware resource until one of the tasks execute, which then releases it to the next task. If the hardware resource is running a task, no other task can use that resource. In this example, two tasks use the same UART peripheral to print messages. By using a semaphore, the two tasks have alternate access to the UART peripheral.

```
Hello world, this is FreeRTOS
Inside uart task 1 loop
Inside uart task 2 loop
Inside uart task 1 loop
Inside uart task 2 loop
Inside uart task 1 loop
Inside uart task 2 loop
```

freertosUartInterruptDemo Example

This demo illustrates the same operation as the uartInterruptDemo, but it executes using the FreeRTOS software framework. The tasks and queues are allocated to an interrupt routine so that the FreeRTOS scheduler can control the execution with the given priority.

The application displays messages on a UART terminal:

```
Hello world
RX FIFO not empty interrupt
RX FIFO not empty interrupt
RX FIFO not empty interrupt
```

fpuDemo

This example (**fpuDemo** directory) shows how to use the floating-point unit to perform various mathematical operations such as calculating sine, cosine, tangent, square root, and division. The demo records the number of clock cycles needed to complete each calculation. You can turn off the floating-point unit in the SoC's IP Configuration wizard to compare the FPU results with those obtained using the base I-extension.

The processing time to obtain the results are faster and the binary size is smaller when using the F/D-extension with floating-point unit.

```
fpu math demo !
rv32i (base-extension) is capable to perform floating-point calculation but
 rv32i requires
more processing time and instruction to calculate the result enable fpu with
 d-extension
will sharply improve processing time and decrease app binary size

sine processing clock cycles:879

cosine processing clock cycles:864

tangent processing clock cycles:1148

square root processing clock cycles:2171

division processing clock cycles:377


Input i (in rad): 0.5820
Sine result: 0.5497
Cosine result: 0.8353
Tangent result: 0.6581

Input x: 3828.1234
Square root result: 61.8718
Divsion result: 1040.5619
```

gpioDemo

This example(**gpioDemo** directory) shows how to use the GPIO and its interrupt function. LED(s) on the development board blink for about 5 seconds and then the application goes into interrupt mode. Toggle `system_gpio_0[0]` to let the GPIO go into the interrupt routine.

```
gpio 0 demo !
onboard LEDs blinking
gpio 0 interrupt demo !
Ti60 press and release onboard button sw6
T120 press and release onboard button sw7
gpio 0 interrupt routine
```

iCacheFlushDemo

This example (**iCacheFlushDemo** directory) illustrates how to invalidate the instruction cache. The instruction cache invalidation is critical to ensure the coherency between the cache and the main memory, ensuring that the CPU fetches the most up-to-date instructions. Firstly, the string `funcA` is copied into an array that is printed out in this example. The `funcA` can be seen as the output. Next, the string `funcB` is copied into the same array that is printed out again. Even though `funcB` is stored in the array, the `funcA` is seen as the output because the instruction cache has not yet been flushed.

To address this, the instruction cache invalidation is called upon. Once the instruction cache is invalidated, the `funcB` can be expected to be printed out in the UART console. Additionally, the most up-to-date instructions are fetched from the main memory.

By following this process, you can ensure that the CPU fetches the most recent instructions from the main memory and maintains coherency with the instruction cache. The design displays these messages in a UART terminal:

```
Expected 'funcA', Obtained : funcA
Expected 'funcA', Obtained : funcA
Expected 'funcB', Obtained : funcB
Test Complete
```

inlineAsmDemo

This example (**inlineAsmDemo** directory) illustrates utilizing the inline assembly feature. The inline assembly feature allows you to embed assembly language code into your high-level code such as C and C++ whenever you need to implement low-level operations or improve the performance.

The following are demonstrations of **inlineAsmDemo** applications

- Integer arithmetic operations
- Looping implementation
- if-else implementation
- Memory access
- Proper use of general-purpose register (x0 – x31)
- Exchange of values between the inline assembly and C

This example provides both C and assembly language for the same implementation for easier understanding and further includes the following definition to use the C language implementation.

```
#define C_IMPLEMENTATION 1
```

This application increments the LEDs until all LEDs are enabled and waits for the UART input character 'r'. Once received, the LEDs will be reset for increment again.

The UART terminal prints these messages when C_IMPLEMENTATION is defined.

```
Inline Assembly Demo

Demonstrating C implementation

Reset the LEDs by pressing 'r'
```

The UART terminal prints these messages when C_IMPLEMENTATION is not defined and inline assembly is used.

```
Inline Assembly Demo

Reset the LEDs by pressing 'r'
```

Refer to Inline Assembly to understand more about inline assembly and its application.

i2cDemo Example

The I^2C interrupt example (**i2cDemo** directory) provides example code for an I^2C master writing data to and reading data from an off-chip MCP4725 device with interrupt. The Microchip MCP4725 device is a single channel, 12-bit, voltage output digital-to-analog converter (DAC) with an I^2C interface.

The MCP4725 device is available on breakout boards from vendors such as Adafruit and SparkFun. You can connect the breakout board's SDA and SCL pins to a development board.

The code assumes that the I^2C block is the only master on the bus, and it sends frames in blocks. When you run it, the application connects to the MCP4725 device and increases the DAC value. It also prints the message `Start` on a UART terminal.

In this example:
- `void trap()` traps entries on exceptions and interrupt events
- `void externalInterrupt()` triggers an interrupt event

i2cEepromDemo

This example (**i2cEepromDemo** directory) demonstrates the usage of the I^2C driver to establish communication with the on-board EEPROM device, specifically the AT24C01 that is used in the Trion T120F324 and T120F576 development kit. The UART console serves as an interactive terminal that allows you to select the available operations, specify the address and number of bytes to read/write, and provide the data to be written.

This example shows you on the interaction and configuration with the onboard EEPROM using the I^2C driver that facilitates the data transfer and its manipulation through the UART console.

```
T120F324/T120F576 Dev Kit on-board EEPROM, AT24C01 i2c-demo !
Please make sure you are using the T120F324/T120F576 Dev Kit to run this demo!
Please choose the feature you would like to run (key in the selection and press enter):
1: Write a byte to EEPROM
2: Read a byte from EEPROM
3: Current Address Read (Last accessed address incremented by 1)
4: Read multiple byte from EEPROM
5: Write multiple byte to EEPROM
```

To write a byte of data to the EEPROM, follow these steps:

1. Type `1` on the console and press enter.
2. Enter the desired address in hexadecimal format. For example, if you want to write to the address '0000', enter `0000` and press enter.
3. If you enter an invalid address, the message "invalid address input" is displayed. Re-enter a valid address in hexadecimal format.
4. Upon entering a valid address, you will be prompted to enter a byte of data to be written into the EEPROM. For instance, if you want to write the hexadecimal value of '55', type the value `55` and press enter.

By following these steps, you will be able to write a byte of data to the EEPROM using the provided interface. You must follow the instructions and input the required values accurately to ensure successful data writing.

```
Please choose the feature you would like to run (Key in the selection and press Enter):
1: Write a byte to EEPROM
2: Read a byte from EEPROM
3: Current Address Read (Last accessed address incremented by 1)
4: Read multiple byte from EEPROM
5: Write multiple byte to EEPROM
1
Write operation selected, please enter the location in hex with 16-bit size
00000
Invalid address input, please key in correct address. I.e. 1024 which is in hexadecimal.
0000
Valid address input, please wait while the operation process
address in hex = 0x0000000000000000
Enter the byte of data to write into the eeprom in hexadecimal
55
Inputted number of byte of data to write
Single Byte Write operation started
Write operation successful
```

To read a byte of data from the EEPROM, follow these steps:

1. Type `2` on the console and press enter to select the read operation.
2. Enter the desired address in hexadecimal format. For example, if you want to read from the address '0000', enter `0000` and press enter.

The system reads the data from the specified address in the EEPROM and prints out the read-back data on the console.

```
Please choose the feature you would like to run (Key in the selection and press Enter):
1: Write a byte to EEPROM
2: Read a byte from EEPROM
3: Current Address Read (Last accessed address incremented by 1)
4: Read multiple byte from EEPROM
5: Write multiple byte to EEPROM
2
Read operation selected, please enter the location in hex with 16-bit size
0000
Valid address input, please wait while the operation process
address in hex = 0x0000000000000000
Read operation started
Read operation successful.
Read data = 00000055
```

To write multiple bytes of data to the EEPROM, follow these steps:

1. Type `5` on the console and press enter to select the multiple bytes to write operation.
2. Enter the desired address in hexadecimal format. For example, type `0000` if you want to start writing at address '0000', and press enter.
3. Enter the number of bytes of data you want to write into the EEPROM, in hexadecimal format. For example, type `05` if you want to write 5 bytes and press enter.
4. Enter the bytes of data to be written into the EEPROM, in hexadecimal format. The data must not have any spacing in between and in an ascending pattern. For example, type `0102030405` to write the bytes 01, 02, 03, 04, and 05, and press enter.

By following these steps, you will be able to write multiple bytes of data to the EEPROM using the provided interface. You must enter the correct values in hexadecimal format in an ascending order data pattern without spacing between the bytes.

```
Please choose the feature you would like to run (Key in the selection and press Enter):
1: Write a byte to EEPROM
2: Read a byte from EEPROM
3: Current Address Read (Last accessed address incremented by 1)
4: Read multiple byte from EEPROM
5: Write multiple byte to EEPROM
5
Write Multi-Byte operation selected, please enter the location in hex with 16-bit size
0000
Valid address input, please wait while the operation process
address in hex = 0x0000000000000000
Enter the number of byte of data to write/read into/from the eeprom in hexadecimal (Maximum:
 255 Bytes)
05
Number of bytes: 00000005
Enter the byte of data to write intothe eeprom in hexadecimal (without spacing in between)
0102030405
Inputted number of byte of data to write
Multi Byte Write operation started
Write operation successful
```

To read multiple bytes of data from the EEPROM, follow these steps:

1. Type `4` on the console and press enter to select the multiple bytes read operation.
2. Enter the desired address in hexadecimal format. For example, if you want to start reading from the address '0000', enter `0000` and press enter.
3. Enter the number of bytes of data to be read from the EEPROM, in hexadecimal format. For example, type `5` if you want to read 5 bytes, and press enter.

The system reads the specified number of bytes of data from the EEPROM, starting from the specified address, and prints out the read-back data on the console.

By following these steps, you will be able to read multiple bytes of data from the EEPROM using the provided interface. You must enter the correct operation code of the desired starting address and the number of bytes to read, in hexadecimal format, to retrieve accurate data from the EEPROM.

```
Please choose the feature you would like to run (Key in the selection and press Enter):
1: Write a byte to EEPROM
2: Read a byte from EEPROM
3: Current Address Read (Last accessed address incremented by 1)
4: Read multiple byte from EEPROM
5: Write multiple byte to EEPROM
4
Read Multi-Byte operation selected, please enter the location in hex with 16-bit size
0000
Valid address input, please wait while the operation process
address in hex = 0x0000000000000000
Enter the number of byte of data to write/read into/from the eeprom in hexadecimal (Maximum:
 255 Bytes)
05
Number of bytes to read: 00000005
Read operation successful.
0x00000001 0x00000002 0x00000003 0x00000004 0x00000005
```

Note:
- This example can only be used either for Trion T120F324 or T120F576 Development Kit.
- The input to the terminal is in hexadecimal number. You are not require to add “0x” to your input.

i2cMasterDemo Design

This example illustrates how to utilize the Sapphire SoC as an I^2C master. The program demonstrates the transmission and reception of data, initially with a single byte, and subsequently with a larger chunk of 20 bytes.

By default, the configuration assumes the slave device is set to transmit a 1-byte register address. For 2-byte register addresses, you need to modify the definition of `WORD_REG_ADDR` to 1.

The design displays these messages in a UART terminal:

```
i2c Master Demo!
Please ensure you 've either connect to a compatible I2C Slave or running the
 i2CSlaveDemo
with I2C ports connected.
TEST STARTED!
I2C Master Demo completed.
TEST PASSED!
```

Note: In the event that the I^2C slave is not connected to the I2C Master, the terminal displays up to `TEST STARTED` only.

i2cSlaveDemo Design

This example illustrates how to utilize the Sapphire SoC as an I^{2}C slave, offering the functionality of an 8-bit by 256-bit memory module. The provided i2cMasterDemo application can control the i2cSlaveDemo application as described in this section.

Upon running the program, you will have the information on the I^{2}C configurations, including the slave address, timeout settings, and various timing configurations.

The UART console acts as an interactive terminal, facilitating the monitoring of current memory values by simply pressing the I key.

By default, the slave is configured for 1-byte register addresses. For 2-byte register addresses, you need to modify the definition of `WORD_REG_ADDR` to 1.

The design displays these messages in a UART terminal:

```
i2c 0 slave demo!
i2c 0 init done
This device will asct as I2C Slave with 8 bit x 256 bit memory
Configurations:
Slave Address = 0x67
Timeout setting = 0x4c4b40
Tsu = 166
tLow = 250
tHigh = 250
tBuf = 500

    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
10: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
20: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
30: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
40: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
50: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
60: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
70: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
80: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
90: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
a0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
b0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
c0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
d0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff
e0: ff ff ff ff ff ff ff ff ff ff ff ff ff ff

press i to show the memory content of I2C slave
>>
```

memTest Example

The memory test example (**memTest** directory) provides example code that performs a memory test on the external memory module and reports the results on a UART terminal. A successful test prints:

```
Memory test
Passed
```

If the memory test fails, the application prints `Failed at address` <*address*>.

nestedInterruptDemo

This demonstration (**nestedInterruptDemo** directory) illustrates how to escalate from an interrupt routine and to execute higher priority routine. The program returns to the lower priority routine after the higher priority routine finished executing. This demo instantiates two user timers; timer `0` has higher priority than timer 1. Timer `0` interrupts the CPU multiple times. The CPU then executes the timer `0` interrupt routine in the middle of executing the timer 1 interrupt routine.

The demo outputs the following messages to a terminal:

```
T1S
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T0S-HP
T0E-HP
T1E
```

openocdServer

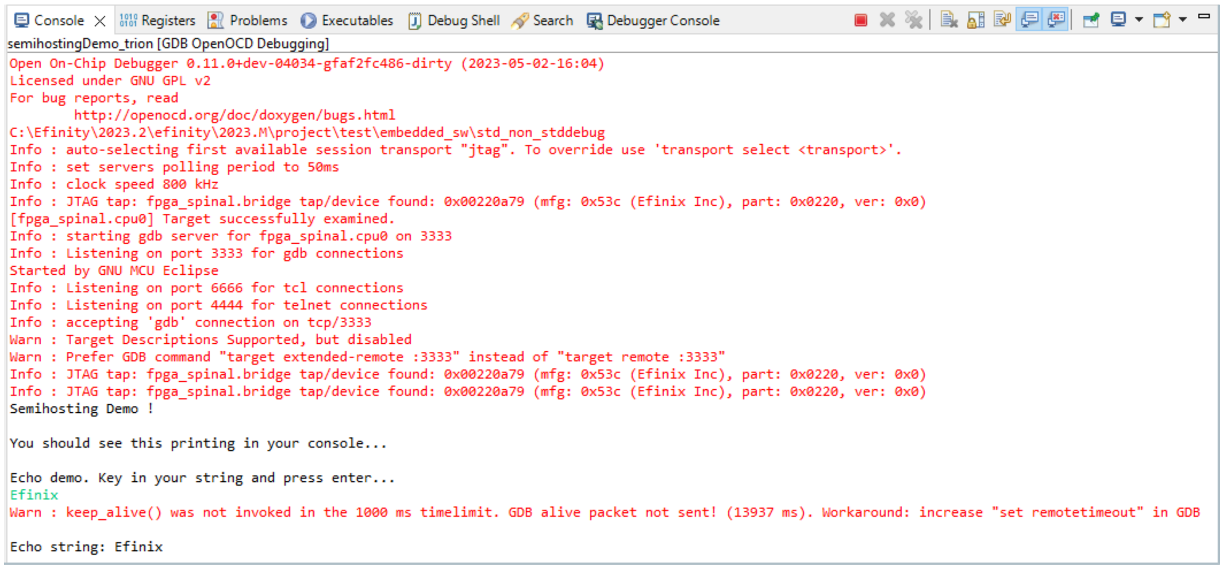
This code (**openocdServer** directory) contains OpenOCD debug scripts to launch the OpenOCD server without the debugger. This script is intended for multi-core debug in standalone environment. Refer to **Debug - Multiple Cores** on page 54 for more details.

semihostingDemo

The semihosting facilitates communication between the host machine and the targeted embedded system through a debugger. This feature is useful during the development and debugging phases, as it allows you to print debug messages without needing a UART peripheral enabled. Also, this is practically advantageous when you want to omit the UART peripheral in resource-constrained designs.

The semihostingDemo example design clearly illustrates how to leverage semihosting in the Sapphire SoC. To activate semihosting, ensure that the **ENABLE_SEMIHOSTING_PRINT** define is set to **1** in the **bsp.h** header file. This enables the seamless output of debug messages. All UART printing calls, e.g., bsp_print, bsp_printf, and other printing APIs, that are available in the **bsp.h** file is directed to the Efinity RISC-V Embedded Software IDE console. No modifications are required for your embedded software design.

This demonstration showcases the capability of the Efinity RISC-V Embedded Software IDE in printing debug messages and reading them from the console itself.



semihostingDemo_trion [GDB OpenOCD Debugging]

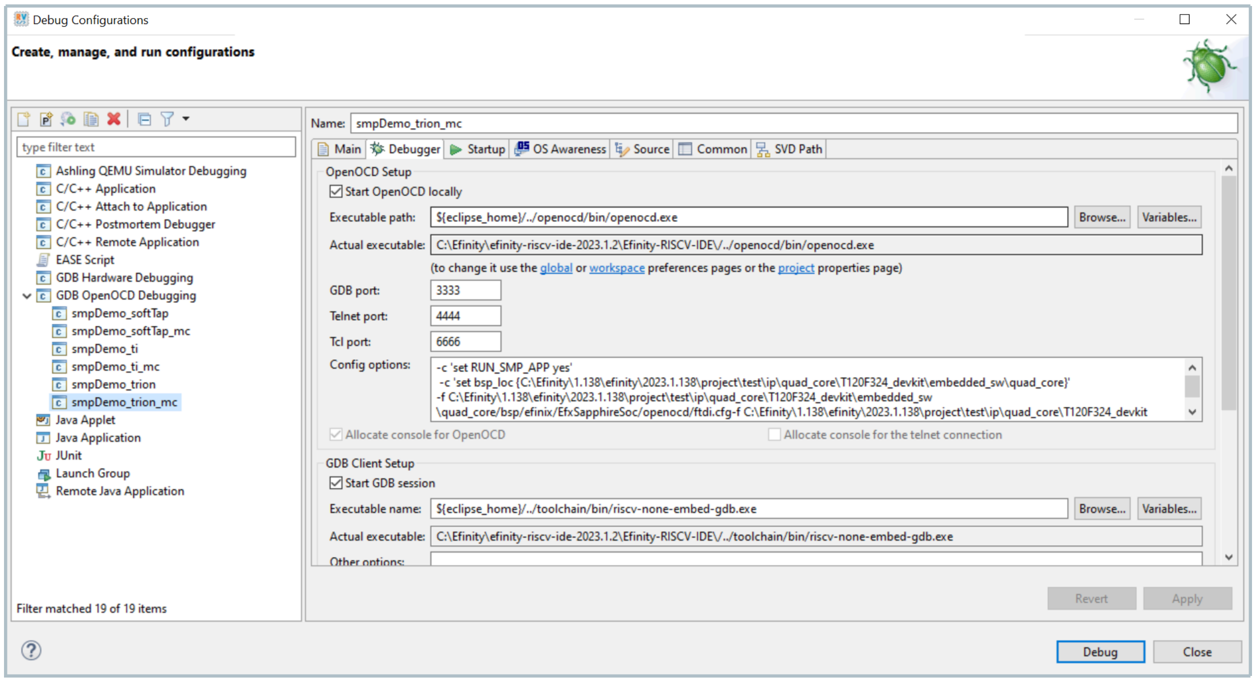
```
Open On-Chip Debugger 0.11.0+dev-04034-gfaf2fc486-dirty (2023-05-02-16:04)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
C:\Efinity\2023.2\efinity\2023.M\project\test\embedded_sw\std_non_stddebug
Info : auto-selecting first available session transport "jtag". To override use 'transport select <transport>'.
Info : set servers polling period to 50ms
Info : clock speed 800 kHz
Info : JTAG tap: fpga_spinal.bridge tap/device found: 0x00220a79 (mfg: 0x53c (Efinix Inc), part: 0x0220, ver: 0x0)
[fpga_spinal.cpu0] Target successfully examined.
Info : starting gdb server for fpga_spinal.cpu0 on 3333
Info : Listening on port 3333 for gdb connections
Started by GNU MCU Eclipse
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : accepting 'gdb' connection on tcp/3333
Warn : Target Descriptions Supported, but disabled
Warn : Prefer GDB command "target extended-remote :3333" instead of "target remote :3333"
Info : JTAG tap: fpga_spinal.bridge tap/device found: 0x00220a79 (mfg: 0x53c (Efinix Inc), part: 0x0220, ver: 0x0)
Info : JTAG tap: fpga_spinal.bridge tap/device found: 0x00220a79 (mfg: 0x53c (Efinix Inc), part: 0x0220, ver: 0x0)
Semihosting Demo !

You should see this printing in your console...

Echo demo. Key in your string and press enter...
Efinix
Warn : keep_alive() was not invoked in the 1000 ms timelimit. GDB alive packet not sent! (13937 ms). Workaround: increase "set remotetimeout" in GDB

Echo string: Efinix
```

Note: While running the application, you may observe a warning in the console indicating `keep_alive()` is not invoked. This warning arises from the blocking nature of the semihosting reading, which can potentially delay the debugger from sending the `keep_alive()` signal on time. This warning does not impact the functionality of the application. It is simply a notification related to the timing of the `keep_alive()` signal. Therefore, it should not be a cause of alarm regarding the overall performance or expected behavior of the system.

smpDemo

This demo (**smpDemo** directory) illustrates how to use multiple cores to process multiple encryption pat the same time in parallel. Each core is assigned an encryption algorithm with an input keys (each core has a different key). Core `0` prints the final encrypted values after the other cores complete the encryption. If a single core performed the encryption, it would take four times more clock cycles to complete the process.

To run the smpDemo correctly, ensure your Sapphire SoC is configured with more than 1 core, or else you may encounter a build error. If your Sapphire SoC is configured as a multi-core, the `*_mc.launch` scripts are generated by the Efinity RISC-V Embedded Software IDE. Launch the `*_mc.launch` based on your configuration.



Name: smpDemo_trion_mc

OpenOCD Setup

[✓] Start OpenOCD locally

Executable path: ${eclipse_home}/../openocd/bin/openocd.exe

Actual executable: C:\Efinity\efinity-riscv-ide-2023.1.2\Efinity-RISCV-IDE\/../openocd/bin/openocd.exe

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 3333

Telnet port: 4444

Tcl port: 6666

Config options:

```
-c 'set RUN_SMP_APP yes'
 -c 'set bsp_loc {C:\Efinity\1.138\efinity\2023.1.138\project\test\ip\quad_core\T120F324_devkit\embedded_sw\quad_core}'
-f C:\Efinity\1.138\efinity\2023.1.138\project\test\ip\quad_core\T120F324_devkit\embedded_sw
\quad_core/bsp/efinix/EfxSapphireSoc/openocd/ftdi.cfg-f C:\Efinity\1.138\efinity\2023.1.138\project\test\ip\quad_core\T120F324_devkit
```

[✓] Allocate console for OpenOCD   [ ] Allocate console for the telnet connection

GDB Client Setup

[✓] Start GDB session

Executable name: ${eclipse_home}/../toolchain/bin/riscv-none-embed-gdb.exe

Actual executable: C:\Efinity\efinity-riscv-ide-2023.1.2\Efinity-RISCV-IDE\/../toolchain/bin/riscv-none-embed-gdb.exe

Other options:

Click **Apply** and start the debugging by clicking **Debug**.

Note: You must enable the Standard debug interface in Sapphire SoC Configuration to debug the multi-core.

The demo outputs the following messages to a terminal:

```
smpDemo with multiple cpu processing
synced!
processing clock cycles:24353

hart 0 encrypted output A:167C6CC6
hart 0 encrypted output B:465E6781
hart 1 encrypted output A:E39A3A87
hart 1 encrypted output B:70CF21D1
hart 2 encrypted output A:CBA365FF
hart 2 encrypted output B:003FDFA8
hart 3 encrypted output A:93D5278B
hart 3 encrypted output B:62F40A6F
```

spiDemo Example

The SPI example (**spiDemo** directory) provides example code for reading the device ID and JEDEC ID of the SPI flash device on the development board.

- The default base address map of the SPI flash master is 0xF801_4000.
- The default `SCK` frequency is half of the SoC system clock frequency.
- The default base address of the UART is 0xF801_0000 with a default baud rate of 115200.

The application displays the results on a UART terminal. It continues to print to the terminal until you suspend or stop the application.

```
spi 0 demo !
Device ID : 17
CMD 0x9F : EF4018
CMD 0x9F : EF4018
...
```

spiReadFlashDemo Example

The read flash example (**spiReadFlashDemo** directory) shows how to read data from the SPI flash device on the development board. The software reads 124K of data starting at address 0x380000, which is the default location of the user binary in the flash device. The application displays messages on a UART terminal:

```
Read Flash Start
Addr 00380000 : =FF
Addr 00380001 : =FF
Addr 00380002 : =FF
...
Addr 0039EFFE : =FF
Addr 0039EFFF : =FF
Read Flash End
```

spiWriteFlashDemo Example

The read flash example (**spiWriteFlashDemo** directory) shows how to write data to the SPI flash device on the development board. The software writes data starting at address 0x380000, which is the default location of the user binary in the flash device. The application displays address and data messages on a UART terminal:

```
Write Flash Start
WR Addr 00380000 : =00
WR Addr 00380001 : =01
WR Addr 00380002 : =02
...
WR Addr 003800FD : =FD
WR Addr 003800FE : =FE
WR Addr 003800FF : =FF
Write Flash End
```

uartEchoDemo

This demo (**uartEchoDemo** directory) shows how to use the UART to print messages on a terminal. The characters you type on a keyboard are echoed back to the terminal from the SoC and printed on the terminal.

```
uart echo demo !
start typing on terminal to send character...
echo character:l
echo character:k
echo character:m
```

UartInterruptDemo Example

The UartInterruptDemo example shows how to use a UART interrupt to indicate task completion when sending or receiving data over a UART. The UART can trigger a interrupt when data is available in the UART receiver FIFO or when the UART transmitter FIFO is empty. In this example, when you type a character in a UART terminal, the data goes to the UART receiver and fills up FIFO buffer. This action interrupts the processor and forces the processor to execute an interrupt/priority routine that allows the UART to read from the buffer and send a message back to the terminal.

The application displays messages on a UART terminal:

```
RX FIFO not empty interrupt
RX FIFO not empty interrupt
RX FIFO not empty interrupt
```

userInterruptDemo Example

This demo (**userInterruptDemo** directory) shows how to handle a user interrupt that accepts an interrupt signal from user logic. In this demo, ten seconds after the Sapphire SoC comes out of reset, the user interrupt gets a trigger from the external module. Operation jumps from the main routine to the interrupt routine. When the interrupt code finishes executing, it jumps back to the main routine.

The application displays the messages on a UART terminal:

```
User Interrupt Demo, waiting for user interrupt...
Entered User Interrupt A Routine
```

userTimerDemo

This demo (**userTimerDemo** directory) shows how to use the user timer and its interrupt function. This demo configures the user timer and its prescaler setting, which you use to further scale down the frequency used by the timer's counter. When the timer's counter reaches the targeted tick value, it generates an interrupt signal to interrupt the controller to let the SoC jump from the main routine to the interrupt routine.

```
user timer 0 demo !
user timer 0 interrupt routine
user timer 0 interrupt routine
user timer 0 interrupt routine
user timer 0 interrupt routine
```

Chapter 11

Third-party Debugger

With the RISC-V standard debug enabled, you can debug using other customized debuggers compliant with the standard. Therefore, Efinix has included sample debug scripts for some external debuggers tested working with Sapphire SoC.

The debug scripts are in the **embedded_sw/<SoC module>/bsp/efinix/EfxSapphireSoc/lauterbach_trace32** directory. The directory contains debug scripts for the Lauterbach's TRACE32 debugger.

Note: The Lauterbach demo supports soft JTAG only.

Chapter 12

Using a UART Module

Contents:

A number of the software examples display messages on a UART terminal. If you are using a Titanium development board, you can simply connect a USB cable to the board and to your computer. For Trion development boards, you need to use a USB-to-UART converter.

Using the On-board UART (Titanium)

The Titanium Ti60 F225 Development Board has a USB-to-UART converter connected to the Ti60's GPIOL_01 and GPIOL_02 pins. The Titanium Ti180 J484 Development Board has a USB-to-UART converter connected to the Ti180's GPIOR_67 and GPIOR_68 pins. To use the UART, simply connect a USB cable to the FTDI USB connector on the targeted development board and to your computer.

Note: The board has an FTDI chip to bridge communication from the USB connector. FTDI interface 2 on Ti60 and FTDI interface 0 on Ti180 communicate with the on-board UART. You do not need to install a driver for this interface because when you connect the Titanium Ti60 F225 Development Board or Titanium Ti180 J484 Development Board to your computer, Windows automatically installs a driver for it.

Finding the COM Port (Windows)

1. Type Device Manager in the Windows search box.
2. Expand **Ports (COM & LPT)** to find out which COM port Windows assigned to the UART module. You should see 2 devices listed as USB Serial Port (COM*n*) where *n* is the assigned port number. Note the COM number for the first device; that is the UART.

Finding the COM Port (Linux)

In a terminal, type the command:

```
ls /dev/ttyUSB*
```

The terminal displays a list of attached devices.

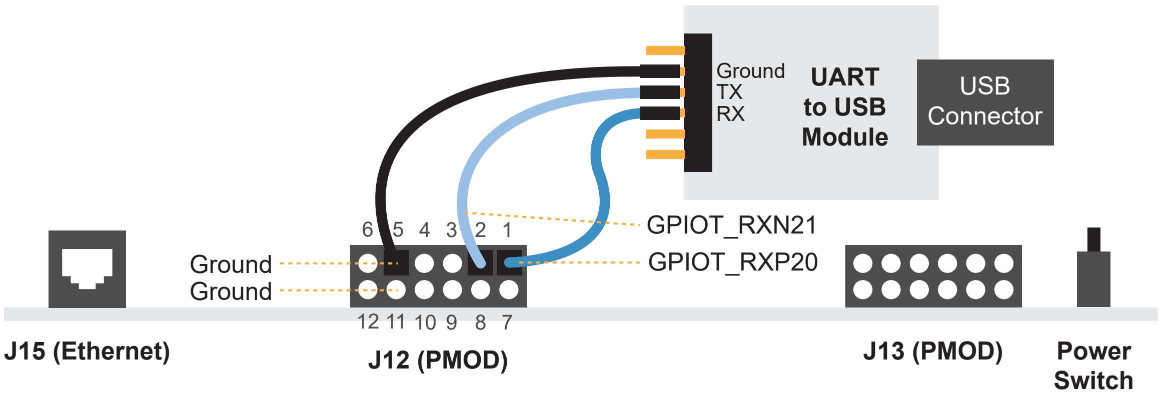
```
/dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyUSB2 /dev/ttyUSB3
```

The UART is `/dev/ttyUSB2`.

Set Up a USB-to-UART Module (Trion)

The Trion® T120 BGA324 Development Board does not have a USB-to-UART converter, therefore, you need to use a separate USB-to-UART converter module. A number of modules are available from various vendors; any USB-to-UART module should work.

Figure 45: Connect the UART Module to PMOD Connector J12



1. Connect the UART module to the PMOD port J12
   - *RX*—GPIOT_RXP20, which is pin 1 on PMOD J12
   - *TX*—GPIOT_RXN21, which is pin 2 on PMOD J12
   - *Ground*—Use ground pin 5 or 11 on PMOD J12.
2. Plug the UART module into a USB port on your computer. The driver should install automatically if needed.

Finding the COM Port (Windows)

1. Type Device Manager in the Windows search box.
2. Expand **Ports (COM & LPT)** to find out which COM port Windows assigned to the UART module; it is listed as USB Serial Port (COM*n*) where *n* is the assigned port number. Note the COM number.

Finding the COM Port (Linux)

In a terminal, type the command:

```
dmesg | grep ttyUSB
```

The terminal displays a series of messages about the attached devices.

```
usb <number>: <adapter> now attached to ttyUSB<number>
```

There are many USB-to-UART converter modules on the market. Some use an FTDI chip which displays a message similar to:

```
usb 3-3: FTDI USB Serial Device converter now attached to ttyUSB0
```

However, the Trion® T120 BGA324 Development Board also has an FTDI chip and gives the same message. So if you have both the UART module and the board attached at the same time, you may receive three messages similar to:

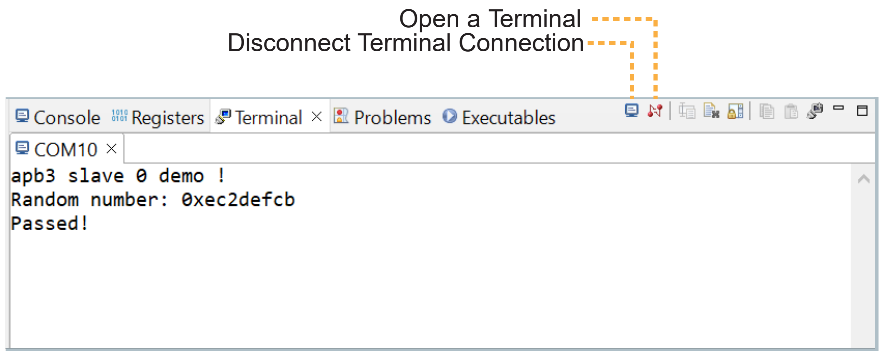
```
usb 3-3: FTDI USB Serial Device converter now attached to ttyUSB0
usb 3-2: FTDI USB Serial Device converter now attached to ttyUSB1
usb 3-2: FTDI USB Serial Device converter now attached to ttyUSB2
```

In this case, the second 2 lines (marked by `usb 3-2`) are the development board and the first line (`usb 3-3`) is the UART module.

Open a Terminal

You can use any terminal program, such as Putty, termite, or the built-in terminal in the Efinity RISC-V Embedded Software IDE, to connect to the UART. These instructions explain how to use the built-in terminal while the others are similar.

1. In Efinity RISC-V Embedded Software IDE, choose **Window** > **Show View** > **Terminal**. The Terminal tab opens.



```
apb3 slave 0 demo !
Random number: 0xec2defcb
Passed!
```

2. Click the **Open a Terminal** button.
3. In the **Launch Terminal** dialog box, enter these settings:

Option	Setting
Choose terminal	Serial Terminal
Serial port	COM*n* (Windows) or ttyUSB*n* (Linux) where *n* is the port number for your UART module.
Baud rate	115200
Data size	8
Parity	None
Stop bits	1
Encoding	Default (ISO-8859-1)

4. Click **OK**. The terminal opens a connection to the UART.
5. Run your application. Messages are printed in the terminal.
6. When you are finished using the application, click the **Disconnect Terminal Connection** button.

Enable Telnet on Windows

Windows does not have telnet turned on by default. Follow these instructions to enable it:

1. Type `telnet` in the Windows search box.
2. Click **Turn Windows features on or off (Control panel)**. The **Windows Features** dialog box opens.
3. Scroll down to **Telnet Client** and click the checkbox.
4. Click **OK**. Windows enables telnet.
5. Click **Close**.

Chapter 13

Unified Printf

Contents:

Prior to Efinity 2022.2, you need specific functions provided in the bsp.h to print various kinds of data such as bsp_printHex, bsp_print, and bsp_printHexDigit. In Efinity 2022.2 or later, Efinix introduces unified printf implementation that enables printf implementation that resembles GNU C library, printf function. Unified printf also supports the legacy bsp_print functions for backward compatibility.

Starting from Efinity 2022.2 onwards, there are 3 print or printf versions that are available for users to print characters to the UART terminal:

- Bsp_print
- Bsp_printf
- Bsp_printf_full

Bsp_print

Bsp_print is the legacy function that consists of various bsp_print* functions as listed:

- bsp_printHex—Print 4-byte Hexadecimal characters (example: 0 x 12345678)
- bsp_print—Print string with newline at the end
- bsp_printHexDigit —Print 1 digit of Hexadecimal value (example: 0 x A)
- bsp_printHexByte—Print 2 digit of Hexadecimal value (example: 0 x AB)
- bsp_printReg—Print string followed by 4-byte Hexadecimal characters
- bsp_putString—Print string without newline at the end
- bsp_putChar—Print an 8-bit character

Bsp_printf

Bsp_printf is a lite version of bsp_printf_full where it only supports a minimum number of specifiers. Bsp_printf is located in *bsp/efinix/EfxSapphireSoc/app/print.h*. Bsp_printf is enabled by default. An example of calling bsp_printf to print out a hex value of 0 x 10 is as follows:

```
bsp_printf("Printing 0x10: %x", 0x10)
```

It supports the following type:

1. Character (%c)
2. String (%s)
3. Decimal (%d)
4. Hexadecimal (%x)
5. Float (%f)

Note: You need to switch the **Enable_Floating_Point_Support** to **1** in the **bsp.h** to enable the floating point supports. The **Enable_Floating_Point_Support** follows the FPU setting where it would be enabled by default if the FPU is included in the SoC.

Bsp_printf_full

Bsp_printf_full is based on open-source Tiny Printf implementation. This printf function supports most of the specifiers. Bsp_print_full is disabled by default. Bsp_printf_full can be enabled by setting the **ENABLE_BSP_PRINTF_FULL** to **1** in the **bsp.h** file. An example of calling bsp_printf_full to print out hex value of 0 x 10 is as follows:

```
bsp_printf_full("Printing 0x10: %x", 0x10)
```

The bsp_printf_full follows the following prototype:

```
%[flags][width][.precision][length]type
```

Note: By enabling **ENABLE_BRIDGE_FULL_TO_LITE** in the **bsp.h** file and the bsp_printf is disabled, bsp_printf_full can be called with bsp_printf instead. This would be beneficial if your program is already using the bsp_printf but requires additional specifiers support that is supported only in bsp_printf_full function.

Table 27: Supported Fomat Types

Type	Description
d or i	Signed decimal integer
u	Unsigned decimal integer
b	Unsigned binary
o	Unsigned octal
x	Unsigned hexadecimal integer (lowercase)
X	Unsigned hexadecimal integer (uppercase)
f or F	Decimal floating point
e or E	Scientific-notation (exponential) floating point
g or G	Scientific or decimal floating point
c	Single character
s	String of characters
P	Pointer address
%	A % followed by another % character output a single %

Table 28: Supported Flags

Flag	Description
-	Left-justify within the given field width; Right justification is the default.
+	Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, b, x or X specifiers; the value is preceeded by 0, 0b, 0x or 0X respectively for values other than zero.
0	Left-pad fills the number with zeros (0) instead of space when padding is specified (see width sub-specifier).

Table 29: Supported Width

Width	Description
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, then the result is padded with blank spaces. The value is not truncated even if the result is larger.
*	The width is not specified in the string format, but as an additional integer value argument preceding the argument that has to be formatted.

Table 30: Supported Precision

Pecision	Description
.number	For integer specifiers (d, i, o , u, x, X): Precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of zero (0) means that no character is written for the value zero (0). For f and F specifiers: This is the number of digits to be printed after the decimal point. By default, the **minimum is 6 (six) and the maximum is 9 (nine)**.
.*	The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.

Table 31: Supported Length

Length	%d, %i	%u, %o, %x, %X
(none)	int	unsigned int
hh	char	unsigned char
h	short int	unsigned short int
l	long int	unsigned long int
ll	long long int	unsigned long long int (if Printf_Support_Long_Long is defined)
j	intmax_t	uintmax_t
z	size_t	size_t
t	ptrdiff_t	ptrdiff_t (if Printf_Support_Ptrdiff_T is defined)

Semihosting Printing

Semihosting is a powerful feature that enhances the development and debugging experience when designing embedded software for your Sapphire SoC. Semihosting acts as a bridge between your host machine and the Sapphire SoC. With semihosting, printing debug messages is achievable without the need for additional peripherals like UART. This is beneficial for designs with limited resources where the debug capabilities are not compromised.

Efinix integrates the semihosting ability to the bsp_print* APIs. By enabling the **ENABLE_SEMIHOSTING_PRINT** in **bsp.h** file, all printing APIs such as bsp_print, bsp_printf, and bsp_printf_full is routed to the semihosting printing where the printout appears in the Efinity RISC-V Embedded Software IDE console instead. No modifications are required for your design source code.

Efinix provides an example design illustrating how to write and read through the semihosting in semihostingDemo.

Preprocessor Directives

Unified printf implementation uses preprocessor directives/switches located in the **bsp.h** to allow customization of the printf function suited to your needs.

Table 32: Preprocessor Directives

Switch	Description	Default
ENABLE_BSP_PRINT	Enable legacy bsp_print functions.	Enabled
ENABLE_BSP_PRINTF	Enable bsp_printf function.	Enabled
ENABLE_BSP_PRINTF_FULL	Enable bsp_printf_full function.	Disabled
ENABLE_SEMIHOSTING_PRINT	Enable semihosting printing. All print functions is routed to the console printout if enabled.	Disabled
ENABLE_FLOATING_POINT_ SUPPORT	Enable floating point printout support.	Follows FPU setting
ENABLE_FP_EXPONENTIAL_ SUPPORT	Enable floating point exponential printout support.	Disabled
ENABLE_PTRDIFF_SUPPORT	Enable pointer difference datatype support.	Disabled
ENABLE_LONG_LONG_SUPPORT	Enable long long datatype support.	Disabled
ENABLE_BRIDGE_FULL_TO_LITE	When enabled and bsp_printf is disabled, the bsp_printf_full can be called using bsp_printf.	Enabled
ENABLE_PRINTF_WARNING	When enabled, warning is printed out when the specifier type is not supported.	Enabled

Chapter 14

Using a Soft JTAG Core for Example Designs

Contents:

The Efinity® Debugger uses the hard JTAG TAP interface. Out of the box, the Sapphire SoC example design also uses the hard JTAG TAP interface for OpenOCD. If you try to use the same USB connection to the development board for both applications at the same time, they will conflict. To solve this problem, you use a soft JTAG block to handle the OpenOCD JTAG communication. With this method, you use an FTDI chip cable to connect the board to your computer (the Efinity® Debugger uses the USB cable).

The simplest way to implement a soft JTAG interface is to use the IP Manager to output an example design that enables the soft JTAG interface. The IP Manager automatically connects the soft JTAG pins to PMOD J12 when you turn on the Soft Debug Tap option.

Note: Efinix does not recommend the FTDI Chip C232HM-DDHSL-0 programming cable due to the possibility of the FPGA not being recognized or the potential for programming failures. You are encourage to use FTDI chip FT2232H or FT4232H mini-module.

Connect the FTDI Mini-Module

When you turn on the Enable Soft JTAG TAP option in the IP Configuration wizard, the example design assigns the JTAG pins to resources in the interface design. Use the following figures to connect the table to the JTAG pins. By default, the C232HM-DDHSL-0 JTAG is targeted. Select your preferred JTAG cable in **Debug and Linker Scripts Support** under the Debug tab in the IP Manager

Note: If you have not already done so, install the driver for the FTDI mini-module as described in Installing USB Drivers on page 32.

Connecting to the Titanium Ti60 F225 Development Board

Figure 46: Connecting the FTDI Module or C232HM-DDHSL-0 Cable

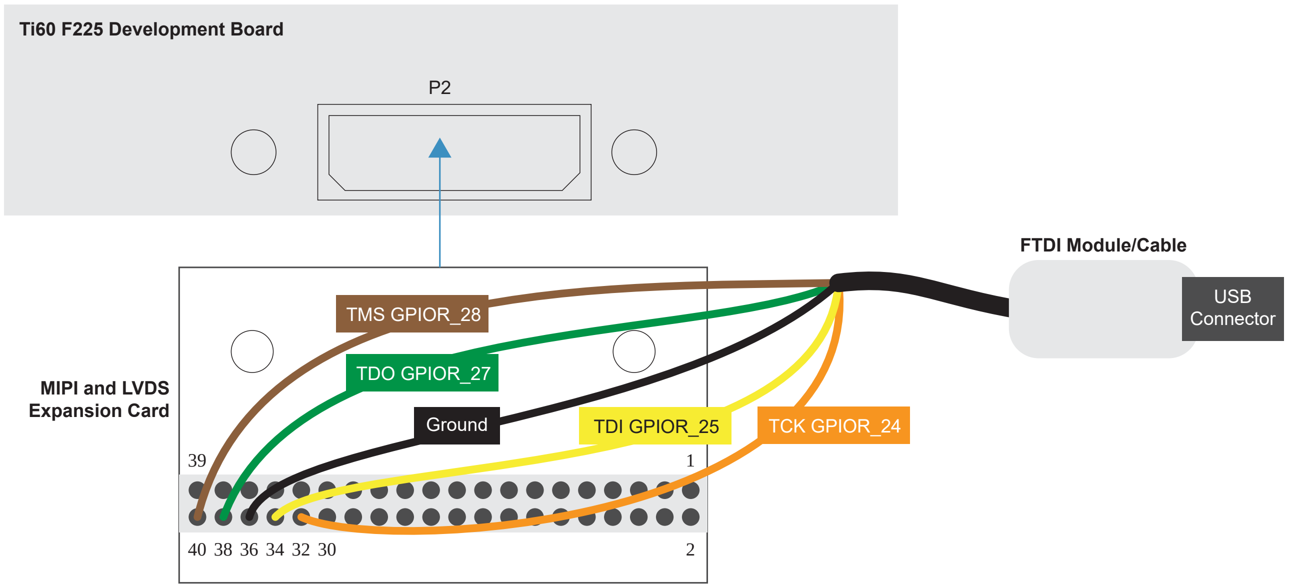


Table 33: FTDI to Daughter Card Connections

Port	Resource	MIPI and LVDS Expansion Daughter Card (P2) Pin
TCK	GPIOR_24	32
TDI	GPIOR_25	34
TDO	GPIOR_27	38
TMS	GPIOR_28	40
GND	-	36

Connecting to the Titanium Ti180 J484 Development Board

Figure 47: Connecting the FTDI Module or C232HM-DDHSL-0 Cable

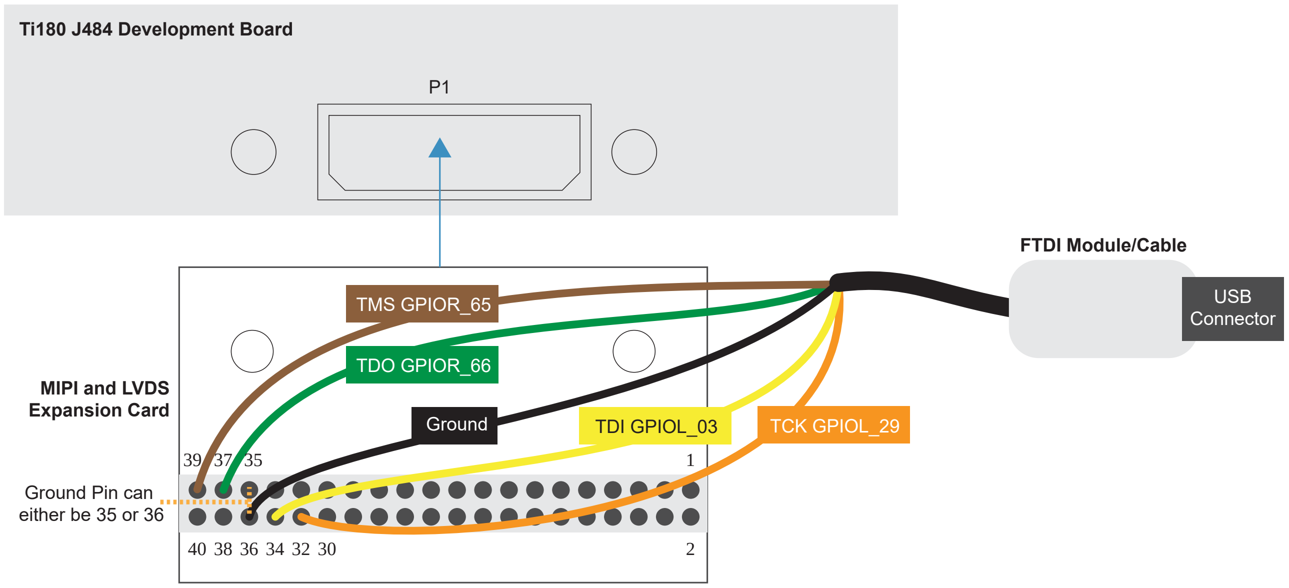


Table 34: FTDI to Daughter Card Connections

Port	Resource	MIPI and LVDS Expansion Daughter Card (P1) Pin
TCK	GPIOL_29	32
TDI	GPIOL_03	34
TDO	GPIOR_66	37
TMS	GPIOR_65	39
GND	-	35/36

Connecting to the Trion® T120 BGA324 Development Board

Figure 48: Connecting the FTDI Module or C232HM-DDHSL-0 Cable

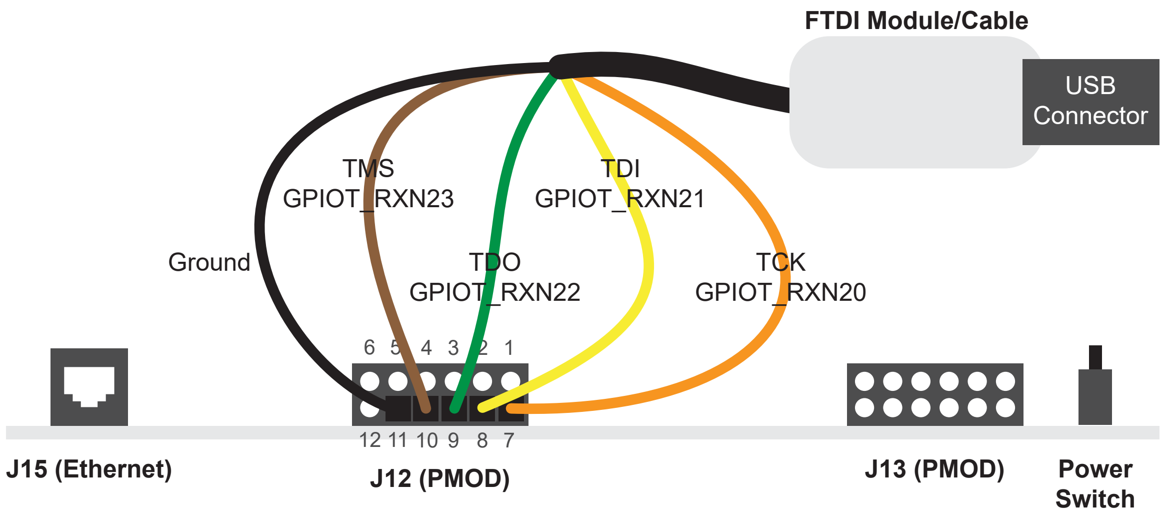


Table 35: FTDI to PMOD Connections

Port	Resource	PMOD (J12) Pin
TCK	GPIOT_RXN20	7
TDI	GPIOT_RXN21	8
TDO	GPIOT_RXN22	9
TMS	GPIOT_RXN23	10
GND	-	5 or 11

Debugging in Efinity RISC-V Embedded Software IDE

1. Open your Efinity RISC-V Embedded Software IDE project.
2. Run or debug the software with the OpenOCD debugger using the **default_softTap** to launch the configuration.
3. Refer to Debug with the OpenOCD Debugger on page 51 for complete instructions.
4. Open the Debugger to perform hardware debugging.

Chapter 15

Migrating to the Sapphire SoC

Contents:

Migrating to the Sapphire SoC v2.0 from a Previous Version

The Sapphire SoC v2.0 available in the Efinity software v2021.2 has many new features compared to previous versions, and the **IP Configuration** wizard and drivers are updated to reflect these new features. Therefore, you cannot automatically migrate an existing design to v2.0. If you want to migrate to v2.0, the following sections provide some guidelines.

Note: Efinix recommends that you use v2.0 for all new designs.

IP Configuration Wizard

The configuration options for the Sapphire SoC v2.0 support new features such as more configurable caching, FPU, MMU, and a peripheral clock. Use the following settings to create a v2.0 SoC that is similar to previous versions.

Table 36: IP Configuration Settings

Tab	Option	Setting	Notes
SOC	Peripheral Clock	DISABLE	In v1.*x*, the APB3 peripherals are driven by the system clock. In v2.0, set this option to DISABLE.
	Custom Instruction	DISABLE	Not supported in v1.*x*
	Linux Memory Management Unit	DISABLE	
	Floating-point Unit	DISABLE	
	Atomic extension	DISABLE	
Cache/ Memory	Data Cache Way	1	In v1.0, the SoC has a fixed I$ and D$ cache way (1 way) and size (4 KB). In v1.1, the wizard supports 1 ways and 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, or 32 KB
	Data Cache Size	4 KB (v1.0) 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, or 32 KB (v1.1)	
	Instruction Cache Way	1	
	Instruction Cache Size	4 KB (v1.0) 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, or 32 KB (v1.1)	

Tab	Option	Setting	Notes
	External Memory AXI3 Interface	DISABLE (v1.0) ENABLE or DISABLE (v1.1)	In v1.*x*, an external memory interface is not supported with a cacheless CPU
	On-Chip RAM Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, or 512 KB	The v1.*x* SoC supports fewer sizes for On-Chip RAM. Choose one of these options in v2.0 for compatibility.
Debug	Target OpenOCD	See v2.0 options	This option is not supported in v1.0. This option is the same in v1.1 and v2.0.
	Custom Target OpenOCD	See v2.0 options	This option is not supported in v1.0. This option is the same in v1.1 and v2.0.
	OpenOCD Debug Mode	Any	This option is not supported in v1.x. However, you can choose either option because it sets IDE environment variables and does not affect the SoC.

Debug Configuration

The **default_softTap** debug configuration file is updated in v2.0. Therefore, you cannot use the **default_softTap** generated with v1.1 with v2.0. If you are using v2.0, you need to remove the old **default_softTap** debug configuration from your Eclipse project and import the v2.0 one. See Appendix: Import the Debug Configuration on page 190 for instructions. Importing the Debug Configuration is not applicable if you are using Efinity RISC-V Embedded Software IDE as the IDE generates the debug configuration during the import project wizard.

Application Software

In v2.0, the there are several changes to the generated embedded software:

- **SoC device names and definitions**—The device names and definitions in the **soc.h** file are updated. The v2.0 embedded software includes the file **compability.h**, which converts the naming from v1.*x* to v2.0. Include **compability.h** at the top of your software application code to convert the names. You can also reference the example **compabilityDemo** in the **/embedded_sw/**<*module name*>**/software/standalone** folder.
- **Core timer driver**—The machine timer is replaced with the Clint timer, which is a native CPU timer. The software driver code is slightly different than the code for the machine timer. To convert from the machine timer function to the Clint timer function, include the **compability.h** and **bsp.h** at the top of your software application code.

Note: compatibilityDemo on page 86 provides an example of how to use **compability.h** and **bsp.h**.

Migrating Ruby, Jade, and Opal to the Sapphire SoC

The Ruby, Jade, and Opal SoCs are end of life in the Efinity software v2022.1. The following sections provide the parameters you should set in the Sapphire SoC IP Configuration wizard to get the same functionality as Ruby, Jade, or Opal.

Ruby Configuration

Parameter		Setting	Address
SOC	Core Number	1	
	Frequency	Configurable	
	Peripheral Clock	No	
	Cache	Yes	
	Custom Instruction	No	
	Linux Memory Management Unit	No	
	Floating-point unit	No	
	Atomic Extension	No	
Cache/Memory	Data Cache Way	1	
	Data Cache Size	4KB	
	Instruction Cache Way	1	
	Instruction Cache Size	4KB	
	External Memory Interface	Yes	
	AXI Interface Type	AXI3	
	External Memory Data Width	128	
	External Memory Address Size	3.5GB	
	On-Chip RAM Size	Configurable	0xf9000000
	Custom On-Chip RAM Application	No	
Debug	Soft Debug Tap	Configurable	
	FPGA Tap Port	Configurable	
	Target Board	Configurable	
	Application Region Size	-	
	Application Stack Size	-	
	Application Debug Mode	-	
UART	UART0	Yes	0xf8010000
	UART0 Interrupt ID	1	

Parameter		Setting	Address
	UART1	Yes	0xf8011000
	UART1 Interrupt ID	2	
	UART2	No	
	UART2 Interrupt ID	-	
SPI	SPI0	Yes	0xf8014000
	SPI0 Interrupt ID	4	
	SPI1	Yes	0xf8015000
	SPI1 Interrupt ID	5	
	SPI2	Yes	0xf8016000
	SPI2 Interrupt ID	6	
I2C	I2C0	Yes	0xf8018000
	I2C0 Interrupt ID	8	
	I2C1	Yes	0xf8019000
	I2C1 Interrupt ID	9	
	I2C2	Yes	0xf801A0000
	I2C2 Interrupt ID	10	
GPIO	GPIO0	Yes	0xf8000000
	GPIO0 Width	16	
	GPIO0 Interrupt ID	12, 13	
	GPIO1	No	
	GPIO1 Width	-	
	GPIO1 Interrupt ID	-	
APB3	APB3 Slave Size	64KB	
	APB0	Yes	0xf8800000
	APB1	Yes	0xf8810000
	APB2	No	
	APB3	No	
	APB4	No	
AXI4	AXI Slave	Yes	0xfa000000
	AXI Slave Size	16MB	
	AXI Master 0	Yes	
	AXI Master 0 Data Width	32	
	AXI Master 1	-	
	AXI Master 1 Data Width	-	
User Interrupt	User Interrupt A	Yes	
	User Interrupt A ID	25	
	User Interrupt B	No	

Parameter		Setting	Address
	User Interrupt B ID	-	
	User Interrupt C	No	
	User Interrupt C ID	-	
	User Interrupt D	No	
	User Interrupt D ID	-	
	User Interrupt E	No	
	User Interrupt E ID	-	
	User Interrupt F	No	
	User Interrupt F ID	-	
	User Interrupt G	No	
	User Interrupt G ID	-	
	User Interrupt H	No	
	User Interrupt H ID	-	
User Timer	User Timer 0	No	
	User Timer 0 Counter Width	-	
	User Timer 0 Prescaler Width	-	
	User Timer 0 Interrupt ID	-	
	User Timer 1	No	
	User Timer 1 Counter Width	-	
	User Timer 1 Prescaler Width	-	
	User Timer 1 Interrupt ID	-	
	User Timer 2	No	
	User Timer 2 Counter Width	-	
	User Timer 2 Prescaler Width	-	
	User Timer 2 Interrupt ID	-	

Jade Configuration

Parameter		Setting	Address
SOC	Core Number	1	
	Frequency	Configurable	
	Peripheral Clock	No	
	Cache	Yes	
	Custom Instruction	No	
	Linux Memory Management Unit	No	
	Floating-point unit	No	
	Atomic Extension	No	
Cache/Memory	Data Cache Way	1	
	Data Cache Size	4KB	
	Instruction Cache Way	1	
	Instruction Cache Size	4KB	
	External Memory Interface	No	
	AXI Interface Type	-	
	External Memory Data Width	-	
	External Memory Address Size	-	
	On-Chip RAM Size	Configurable	0xf9000000
	Custom On-Chip RAM Application	No	
Debug	Soft Debug Tap	Configurable	
	FPGA Tap Port	Configurable	
	Target Board	Configurable	
	Application Region Size	-	
	Application Stack Size	-	
	Application Debug Mode	-	
UART	UART0	Yes	0xf8010000
	UART0 Interrupt ID	1	
	UART1	No	
	UART1 Interrupt ID	-	
	UART2	No	
	UART2 Interrupt ID	-	
SPI	SPI0	Yes	0xf8014000
	SPI0 Interrupt ID	4	

Parameter		Setting	Address
	SPI1	Yes	0xf8015000
	SPI1 Interrupt ID	5	
	SPI2	-	
	SPI2 Interrupt ID	-	
I2C	I2C0	Yes	0xf8018000
	I2C0 Interrupt ID	8	
	I2C1	Yes	0xf8019000
	I2C1 Interrupt ID	9	
	I2C2	No	
	I2C2 Interrupt ID	-	
GPIO	GPIO0	Yes	0xf8000000
	GPIO0 Width	16	
	GPIO0 Interrupt ID	12, 13	
	GPIO1	No	
	GPIO1 Width	-	
	GPIO1 Interrupt ID	-	
APB3	APB3 Slave Size	64KB	
	APB0	Yes	0xf8800000
	APB1	No	
	APB2	No	
	APB3	No	
	APB4	No	
AXI4	AXI Slave	No	
	AXI Slave Size	-	
	AXI Master 0	-	
	AXI Master 0 Data Width	-	
	AXI Master 1	-	
	AXI Master 1 Data Width	-	
User Interrupt	User Interrupt A	Yes	
	User Interrupt A ID	25	
	User Interrupt B	No	
	User Interrupt B ID	-	
	User Interrupt C	No	
	User Interrupt C ID	-	
	User Interrupt D	No	
	User Interrupt D ID	-	
	User Interrupt E	No	

Parameter		Setting	Address
	User Interrupt E ID	-	
	User Interrupt F	No	
	User Interrupt F ID	-	
	User Interrupt G	No	
	User Interrupt G ID	-	
	User Interrupt H	No	
	User Interrupt H ID	-	
User Timer	User Timer 0	No	
	User Timer 0 Counter Width	-	
	User Timer 0 Prescaler Width	-	
	User Timer 0 Interrupt ID	-	
	User Timer 1	No	
	User Timer 1 Counter Width	-	
	User Timer 1 Prescaler Width	-	
	User Timer 1 Interrupt ID	-	
	User Timer 2	No	
	User Timer 2 Counter Width	-	
	User Timer 2 Prescaler Width	-	
	User Timer 2 Interrupt ID	-	

Opal Configuration

Parameter		Setting	Address
SOC	Core Number	1	
	Frequency	Configurable	
	Peripheral Clock	No	
	Cache	No	
	Custom Instruction	-	
	Linux Memory Management Unit	-	
	Floating-point unit	-	
	Atomic Extension	-	
Cache/Memory	Data Cache Way	-	
	Data Cache Size	-	

Parameter		Setting	Address
	Instruction Cache Way	-	
	Instruction Cache Size	-	
	External Memory Interface	No	
	AXI Interface Type	-	
	External Memory Data Width	-	
	External Memory Address Size	-	
	On-Chip RAM Size	Configurable	0xf9000000
	Custom On-Chip RAM Application	No	
Debug	Soft Debug Tap	Configurable	
	FPGA Tap Port	Configurable	
	Target Board	Configurable	
	Application Region Size	-	
	Application Stack Size	-	
	Application Debug Mode	-	
UART	UART0	Yes	0xf8010000
	UART0 Interrupt ID	1	
	UART1	No	
	UART1 Interrupt ID	-	
	UART2	No	
	UART2 Interrupt ID	-	
SPI	SPI0	Yes	0xf8014000
	SPI0 Interrupt ID	4	
	SPI1	No	
	SPI1 Interrupt ID	-	
	SPI2	-	
	SPI2 Interrupt ID	-	
I2C	I2C0	Yes	0xf8018000
	I2C0 Interrupt ID	8	
	I2C1	No	
	I2C1 Interrupt ID	-	
	I2C2	No	
	I2C2 Interrupt ID	-	
GPIO	GPIO0	Yes	0xf8000000
	GPIO0 Width	8	

Parameter		Setting	Address
	GPIO0 Interrupt ID	12, 13	
	GPIO1	No	
	GPIO1 Width	-	
	GPIO1 Interrupt ID	-	
APB3	APB3 Slave Size	64KB	
	APB0	Yes	0xf8800000
	APB1	No	
	APB2	No	
	APB3	No	
	APB4	No	
AXI4	AXI Slave	No	
	AXI Slave Size	-	
	AXI Master 0	-	
	AXI Master 0 Data Width	-	
	AXI Master 1	-	
	AXI Master 1 Data Width	-	
User Interrupt	User Interrupt A	Yes	
	User Interrupt A ID	25	
	User Interrupt B	No	
	User Interrupt B ID	-	
	User Interrupt C	No	
	User Interrupt C ID	-	
	User Interrupt D	No	
	User Interrupt D ID	-	
	User Interrupt E	No	
	User Interrupt E ID	-	
	User Interrupt F	No	
	User Interrupt F ID	-	
	User Interrupt G	No	
	User Interrupt G ID	-	
	User Interrupt H	No	
	User Interrupt H ID	-	
User Timer	User Timer 0	No	
	User Timer 0 Counter Width	-	
	User Timer 0 Prescaler Width	-	

Parameter		Setting	Address
	User Timer 0 Interrupt ID	-	
	User Timer 1	No	
	User Timer 1 Counter Width	-	
	User Timer 1 Prescaler Width	-	
	User Timer 1 Interrupt ID	-	
	User Timer 2	No	
	User Timer 2 Counter Width	-	
	User Timer 2 Prescaler Width	-	
	User Timer 2 Interrupt ID	-	

Chapter 16

Troubleshooting

Contents:

Error 0x80010135: Path too long (Windows)

When you unzip the legacy RISC-V SDK on Windows, you may get the error message:

```
An unuexpected error is keeping you from copying the file. If you continue
to receive this error, you can use the error code to search for help with
this problem.

Error 0x80010135: Path too long
```

This error occurs if you try to unzip the SDK files into a deep folder hierarchy instead of one that is close to the root level. Instead unzip to **c:\riscv-sdk**.

OpenOCD Error: timed out while waiting for target halted

The OpenOCD debugger console may display this error when:
- There is a bad contact between the FPGA header pins and the programming cable.
- The FPGA is not configured with a Sapphire SoC design.
- You may not have the correct PLL settings to work with the Sapphire SoC.
- Your computer does not have enough memory to run the program.
- You may use the wrong launch scripts to launch the debug.

To solve this problem:
- Make sure that all of the cables are securely connected to the board and your computer.
- Check the JTAG connection.

Memory Test

Your user binary may not boot correctly if there is a memory corruption problem (that is, the communication between the DDR hard controller and memory module is not functioning). This issue can appear when booting using the SPI flash or OpenOCD debugger. The following instructions provide a debugging flow to determine whether you system has this problem. You use two command prompts or shells to perform the test:

- The first terminal opens an OpenOCD connection to the SoC.
- The second connects to the first terminal for performing the test.

Important: If you are using the OpenOCD debugger in Efinity RISC-V Embedded Software IDE, terminate any debug processes before performing this test.

Set Up Terminal 1

To set up terminal 1, the flow varies on your IDE selection during the Sapphire SoC generation.

Efinity RISC-V Embedded Software IDE Selected

1. Open a Windows command prompt or Linux shell.
2. Change the directory to any of the example designs in your selected bsp location. The default location for **<efinity-riscv-ide installation path>** would be **C:\Efinity \efinity-riscv-ide-2022.2.3 for windows and home/<user>/efinity/efinity-riscv-ide-2022.2.3** for Linux.

   Note: The 2022.2.3 in the installation path may be different based on your IDE versions.

   Windows:

```
<efinity-risc-v-ide installation path>\openocd\bin\openocd.exe -f ..\..\..
\bsp\efinix\EfxSapphireSoc\openocd\ftdi.cfg
-c "set CPU0_YAML ..\..\..\cpu0.yaml"
-f ..\..\..\bsp\efinix\EfxSapphireSoc\openocd\flash.cfg
```

   Linux:

```
<efinity-risc-v-ide installation path>/openocd/bin/openocd -f ../../../bsp/
efinix/EfxSapphireSoc/openocd/ftdi.cfg
-c "set CPU0_YAML ../../../cpu0.yaml"
-f ../../../bsp/efinix/EfxSapphireSoc/openocd/flash.cfg
```

   The OpenOCD server connects and begins listening on port 4444.

Legacy Eclipse IDE Selected

1. Open a Windows command prompt or Linux shell.
2. Change to **SDK_Windows** or **SDK_Ubuntu**.
3. Execute the **setup.bat** (Windows) or **setup.sh** (Linux) script.
4. Change to the directory that has the **cpu0.yaml** file.
5. Type the following commands to set up the OpenOCD server:

   Windows:

   ```
   openocd.exe -f bsp\efinix\EfxSapphireSoc\openocd\ftdi.cfg
     -c "set CPU0_YAML cpu0.yaml"
     -f bsp\efinix\EfxSapphireSoc\openocd\flash.cfg
   ```

   Linux:

   ```
   openocd -f bsp/efinix/EfxSapphireSoc/openocd/ftdi.cfg
     -c "set CPU0_YAML cpu0.yaml"
     -f bsp/efinix/EfxSapphireSoc/openocd/flash.cfg
   ```

   The OpenOCD server connects and begins listening on port 4444.

Set Up Terminal 2

1. Open a second command prompt or shell.
2. Enable telnet if it is not turned on. Turn on telnet (Windows)
3. Open a telnet host on port 4444 with the command `telnet localhost 4444`.
4. To test the on-chip RAM, use the `mdw` command to get the bootloader binary. Type the command `mdw` <*address*> <*number of 32-bit words*> to display the content of the memory space. For example: `mdw 0xF900_0000 32`.
5. To test the DRAM:
   - Use the `mww` command to write to the memory space: `mww` <*address*> <*data*>. For example: `mww 0x00001000 16`.
   - Then, use the `mdw` command to write to the memory space: `mdw` <*address*> <*data*>. For example: `mdw 0x00001000 16`. If the memory space has collapsed, the console shows all `0`s.

Close Terminals

When you finish:
- Type `exit` in terminal 2 to close the telnet session.
- Type Ctrl+C in terminal 1 to close the OpenOCD session.

Important: OpenOCD cannot be running in Efinity RISC-V Embedded Software IDE when you are using it in a terminal. If you try to run both at the same time, the application will crash or hang. Always close the terminals when you are done flashing the binary.

Reset the FPGA

Press the reset button on your development board:
- *Trion® T120 BGA324 Development Board*—SW2
- *Titanium Ti60 F225 Development Board*—SW3
- *Titanium Ti180 J484 Development Board*—SW1

OpenOCD error code (-1073741515)

The OpenOCD debugger may fail with error code -1073741515 if your system does not have the **libusb0.dll** installed. To fix this problem, install the DLL. This issue only affects Windows systems.

OpenOCD Error: no device found

The FTDI driver included with the Sapphire SoC specifies the FTDI device VID and PID, and board description. In some cases, an early revision of the Efinix development board may have a different name than the one given in the driver file. If the board name does not match the name in the driver, OpenOCD fails with an error similar to the following:

```
Error: no device found
Error: unable to open ftdi device with vid 0403, pid 6010, description 'Trion T20 Development
   Board', serial '*' at bus location '*'
```

To fix this problem, follow these steps with the development board attached to the computer:

1. Open the Efinity Programmer.
2. Click the **Refresh USB Targets** button to display the board name in the **USB Target** drop-down list.
3. Make note of the board name.
4. In a text editor, open the **ftdi.cfg** (Trion) or **ftdi_ti.cfg** (Titanium) file in the **/bsp/efinix/EFXSapphireSoC/openocd** directory.
5. Change the `ftdi_device_desc` setting to match your board name. For example, use this code to change the name from Trion T20 Development Board to Trion T20 Developer Board:

```
interface ftdi
ftdi_device_desc "Trion T20 Developer Board"
#ftdi_device_desc "Trion T20 Development Board"
ftdi_vid_pid 0x0403 0x6010
```

6. Save the file.
7. Debug as usual in OpenOCD.

OpenOCD Error: failed to reset FTDI device: LIBUSB_ERROR_IO

This error is typically caused because you have the wrong Windows USB driver for the development board. If you have the wrong driver, you get an error similar to:

```
Error: failed to reset FTDI device: LIBUSB_ERROR_IO
Error: unable to open ftdi device with vid 0403, pid 6010, description
'Trion T20 Development Board', serial '*' at bus location '*'
```

OpenOCD Error: target 'fpga_spinal.cpu0' init failed

You may receive this error when trying to debug after creating your OpenOCD debug configuration. The RISC-V IDE Console gives an error message similar to:

```
Error cpuConfigFile C:RiscVsoc_Jadesoc_jade_swcpu0.yaml not found
Error: target 'fpga_spinal.cpu0' init failed
```

This error occurs because the path to the **cpu0.yaml** file is incorrect, specifically the slashes for the directory separators. You should use:

- a single forward slash (/)
- 2 backslashes (\\)

For example, either of the following are good:

```
C:\\RiscV\\soc_Jade\\soc_jade_sw\\cpu0.yaml
C:/RiscV/soc_Jade/soc_jade_sw/cpu0.yaml
```

Eclipse Fails to Launch with Exit Code 13

The Legacy Eclipse software requires a 64-bit version of the Java JRE. When you launch Eclipse using a 32-bit version, you get an error that Java quits with exit code `13`.

If you are downloading the JRE using a web browser from www.java.com, it defaults to getting the 32-bit version. Instead, go to https://www.java.com/en/download/manual.jsp to download the 64-bit version.

The Efinity RISC-V Embedded Software IDE does not require you to install Java JRE as it contains its own Java Executable within its folder. This reduces the chances of failure caused by the Java JRE versioning.

Efinity Debugger Crashes when using OpenOCD

The Efinity® Debugger crashes if you try to use it for debugging while also using OpenOCD. Both applications use the same USB connection to the development board, and conflict if you use them at the same time. To avoid this issue:

- Do not use the two debuggers at the same time.
- Use an FTDI cable and a soft JTAG core for OpenOCD debugging. See Using a Soft JTAG Core for Example Designs for details.

Exception in thread "main"

When you generate the SoC with a custom user application, you may receive messages similar to the following when you compile your software application:

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 29361152 out of bounds for
 length 1024
at spinal.lib.misc.HexTools$$anonfun$initRam$1.apply$mcVII
$sp(HexTools.scala:53)
```

This can happen when you have an SoC with external memory interface. The default linker script targets the external memory region during application compilation. You should compile your application to target on-chip RAM instead by following these steps:

1. Open the file <*project*>**/embedded_sw/**<*module*>**/software/standalone/common/bsp.mk**.
2. Change line 7 from

   ```
   LDSCRIPT ?= ${BSP_PATH}/linker/default.ld
   ```

   to

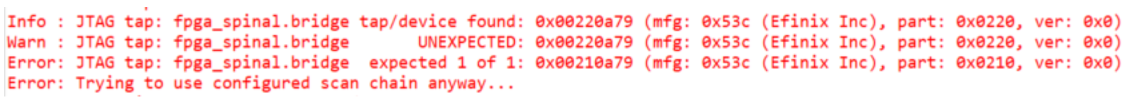
   ```
   LDSCRIPT ?= ${BSP_PATH}/linker/default_i.ld
   ```
3. Recompile the application.

If these steps do not solve the issue, contact the Efinix support team via our forum in the Support Center.

Unexpected CPUTAPID/JTAG Device ID

You may receive the following warnings in the Efinity RISC-V Embedded Software IDE console when trying to debug a device other than the following development kits or devices:

- Trion T20 BGA256 Development Kit (T20BGA256)
- Trion® T120 BGA324 Development Kit (T120BGA324)
- Trion T120 BGA576 Development Kit (T120F576)
- Xyloni (T8BGA81)
- Titanium Ti60 F225 Development Kit (Ti60F225)
- Titanium Ti180 M484 Development Kit (Ti180M484)
- Titanium Ti180 J484 Development Kit (Ti180J484)



```
Info : JTAG tap: fpga_spinal.bridge tap/device found: 0x00220a79 (mfg: 0x53c (Efinix Inc), part: 0x0220, ver: 0x0)
Warn : JTAG tap: fpga_spinal.bridge       UNEXPECTED: 0x00220a79 (mfg: 0x53c (Efinix Inc), part: 0x0220, ver: 0x0)
Error: JTAG tap: fpga_spinal.bridge  expected 1 of 1: 0x00210a79 (mfg: 0x53c (Efinix Inc), part: 0x0210, ver: 0x0)
Error: Trying to use configured scan chain anyway...
```

You will receive the warning if you have selected the wrong development kit. These warnings do not cause any issues to launch the debugging process.

To resolve the unwanted warnings, follow these steps:

1. Go to **Debug Configuration** > **Debugger** > **OpenOCD Setup** > **Config Options**.
2. Type the following command line:

```
-c 'set CPUTAPID 0x<ID>'
```

where <ID> is the correct TAP ID of the connected device in Hexadecimal format.

Note: You may find your device's JTAG device ID in the "JTAG Programming" topic in the Efinity Software User Guide.

Chapter 17

API Reference

Contents:

The following sections describe the API for the code in the **driver** directory.

Control and Status Registers

Note: Refer to Sapphire RISC-V SoC Data Sheet for the available Control and Status Registers (CSR).

csr_clear()

Usage	`csr_clear(csr, val)`
Parameters	[IN] `csr` CSR register [IN] `val` CSR bit to clear. Set 1 on bit to clear.
Include	**driver/riscv.h**
Description	Clear a CSR.

Example	`csr_clear(mie, MIE_MTIE | MIE_MEIE);`
	`// Clear MTIE and MEIE bit in mie CSR`

csr_read()

Usage	`csr_read(csr)`
Parameters	[IN] `csr` CSR register
Returns	[OUT] 32-bit CSR register data
Include	**driver/riscv.h**
Description	Read from a CSR.

Example	`u32 mie = csr_read(mie);`
	`// Read MIE CSR register data in mie variable`

csr_read_clear()

Usage	`csr_read_clear(csr, val)`
Parameters	[IN] `csr` CSR register [IN] `val` CSR bit to clear. Set 1 on bit to clear.
Returns	[OUT] 32-bit CSR register data
Include	**driver/riscv.h**
Description	Read the entire CSR register and clear the specified bits indicated by the argument, `val`.

csr_read_set()

Usage	`csr_read_set(csr, val)`
Parameters	[IN] `csr` CSR register [IN] `val` CSR bit to set. Set 1 on bit to set.
Returns	[OUT] 32-bit CSR register data
Include	**driver/riscv.h**
Description	Read the entire CSR register and set the specified bits indicated by the argument, `val`.

csr_set()

Usage	`csr_set(csr, val)`
Parameters	[IN] `csr` CSR register [IN] `val` CSR bit to set. Set 1 on bit to set.
Include	**driver/riscv.h**
Description	Set the specified bits indicated by the argument, `val` to the CSR.

csr_swap()

Usage	`csr_swap(csr, val)`
Parameters	[IN] `csr` CSR register [IN] `val` Value to swap into CSR register.
Returns	[OUT] 32-bit CSR register data swapped out
Include	**driver/riscv.h**
Description	Swaps values in the CSR.

Example	`u32 val = csr_swap(mtvec, 0x120);`
	`// mtvec CSR will be set to 0 x 120 while the original mtval`
	`// CSR value will be returned as val.`

csr_write()

Usage	`csr_write(csr, val)`
Parameters	[IN] `csr` CSR register [IN] `val` Value to write into CSR register.
Include	**driver/riscv.h**
Description	Write to a CSR.

Example	`csr_write(mtvec, 0x100);`
	`// Write 0 x 100 to mtvec CSR register`

opcode_R()

Usage	`opcode_R(opcode, func3, func7, rs1, rs2)`
Include	**driver/riscv.h**
Description	Define an opcode for the custom instruction.

Example	`#define tea_l(rs1, rs2);`
	`opcode_R(CUSTOM0, 0x00, 0x00, rs1, rs2);`

GPIO API Calls

gpio_getFilteringHit()

Usage	`gpio_getFilteringHit(reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Read the 32-bit I^{2}C register filter hit with a call back function.
Example	`if(gpio_getFilteringHit(I2C_CTRL) == 1);` `// Check filter hit value, bit [7] from slave address,` `// read ='1' write ='0'`

Note: gpio_getFilteringHit() is deprecated, use i2C_getFilteringHit() instead.

gpio_getFilteringStatus()

Usage	`gpio_getFilteringStatus(reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Read the 32-bit I^{2}C register filter status with a call back function.
Example	`if(gpio_getFilteringStatus (I2C_CTRL) == 1);` `// Check filter hit status, bit [7] from slave address,` `// read ='1' write ='0`

Note: gpio_getFilteringStatus() is deprecated, use i2C_getFilteringStatus() instead.

gpio_getInput()

Usage	`gpio_getInput(reg)`
Parameters	[IN] `reg` base address of specific GPIO
Returns	[OUT] 32-bit GPIO input state
Include	**driver/gpio.h**
Description	Get input from a GPIO.

gpio_getInterruptFlag()

Usage	`gpio_getInterruptFlag(reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Returns	[OUT] 32-bit I^{2}C register interrupt flag
Include	**driver/i2c.h**
Description	Read the 32-bit I^{2}C register interrupt flag with a call back function.

Example	`Int flag = gpio_getInterruptFlag(I2C_CTRL) & I2C_INTERRUPT_DROP;`
	`// Get Drop interrupt flag from Interrupt register`
	`//[2] I2C_INTERRUPT_TX_DATA`
	`//[3] I2C_INTERRUPT_TX_ACK`
	`//[7] I2C_INTERRUPT_DROP`
	`//[16] I2C_INTERRUPT_CLOCK_GEN_BUSY`
	`//[17] I2C_INTERRUPT_FILTER`

Note: gpio_getInterruptFlag() is deprecated, use i2C_getInterruptFlag() instead.

gpio_getMasterStatus()

Usage	`gpio_getMasterStatus(reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Returns	[OUT] 32-bit I^{2}C register master status
Include	**driver/i2c.h**
Description	Read the 32-bit I^{2}C register master status with a call back function.

Example	`int status = gpio_getMasterStatus(I2C_CTRL) & I2C_MASTER_BUSY;`
	`// Get master busy status from status register`
	`[0]I2C_MASTER_BUSY`
	`[4]I2C_MASTER_START`
	`[5]I2C_MASTER_STOP`
	`[6]I2C_MASTER_DROP`

Note: gpio_getMasterStatus() is deprecated, use i2C_getMasterStatus() instead.

gpio_getOutput()

Usage	`gpio_getOutput(reg)`
Parameters	[IN] `reg` base address of specific GPIO
Returns	[OUT] 32-bit GPIO output state
Include	**driver/gpio.h**
Description	Read the output pin.

gpio_getOutputEnable()

Usage	`gpio_getOutputEnable(reg)`
Parameters	[IN] `reg` base address of specific GPIO
Returns	[OUT] 32-bit GPIO output enable setting
Include	**driver/gpio.h**
Description	Read GPIO output enable.

gpio_setOutput()

Usage	`gpio_setOutput(reg, value)`
Parameters	[IN] `reg` base address of specific GPIO [IN] `value` GPIO pin bitwise
Include	**driver/gpio.h**
Description	Set GPIO as 1 or 0.

gpio_setOutputEnable()

Usage	`gpio_setOutputEnable(reg, value)`
Parameters	[IN] `reg` base address of specific GPIO [IN] `value` GPIO pin bitwise
Include	**driver/gpio.h**
Description	Set 1 to set GPIO bit as output. Set 0 to set GPIO bit as input.

gpio_setInterruptRiseEnable()

Usage	`gpio_setInterruptRiseEnable(reg, value)`
Parameters	[IN] `reg` base address of specific GPIO [IN] `value` GPIO Rise Interrupt Enable bitwise
Include	**driver/gpio.h**
Description	Set 1 to set GPIO bit to interrupt when a rising edge is detected.

gpio_setInterruptFallEnable()

Usage	`gpio_setInterruptFallEnable(reg, value)`
Parameters	[IN] `reg` base address of specific GPIO [IN] `value` GPIO Fall Interrupt Enable bitwise
Include	**driver/gpio.h**
Description	Set 1 to set GPIO bit to interrupt when a falling edge is detected.

gpio_setInterruptHighEnable()

Usage	`gpio_setInterruptHighEnable(reg, value)`
Parameters	[IN] `reg` base address of specific GPIO [IN] `value` GPIO High Interrupt Enable bitwise
Include	**driver/gpio.h**
Description	Set 1 to set GPIO bit to interrupt when a high state is detected.

gpio_setInterruptLowEnable()

Usage	`gpio_setInterruptLowEnable(reg, value)`
Parameters	[IN] `reg` base address of specific GPIO [IN] `value` GPIO Low Interrupt Enable bitwise
Include	**driver/gpio.h**
Description	Set 1 to set GPIO bit to interrupt when a low state is detected.

I^{2}C API Calls

i2c Config Struct

```
typedef struct{
       //Master/Slave mode
       //Number of cycle - 1 between each SDA/SCL sample
   u32 samplingClockDivider;
       //Number of cycle - 1 after which an inactive frame is considered dropped.
   u32 timeout;
       //Number of cycle - 1 SCL should be keept low (clock stretching)
       //after having feed the data to the SDA to ensure a correct
       //propagation to other devices
   u32 tsuDat;
       //Master mode
       //SCL low (cycle count -1)
   u32 tLow;
       //SCL high (cycle count -1)
   u32 tHigh;
       //Minimum time between the Stop/Drop -> Start transition
   u32 tBuf;
   } I2c_Config;
```

i2c_getFilteringHit()

Usage	`I2c_getFilteringHit(reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Returns	[OUT] 2-bit output: [0] indicates address hit for address setting 0. [1] indicates address hit for address setting 1.
Description	Read the 32-bit I^{2}C register filter hit to register filter hit with a call back function. Return 1 on a specific bit if the filter address is enabled and the address received from the master is tallied with the target address settings for target address 0 (0 x 88) and target address 1 (0 x 8C). Used for slave mode.

Example	`if(i2c_getFilteringHit(I2C_CTRL) == 1);`
	`// Check if address 0 received is the expected address from master.`

i2c_getFilteringStatus()

Usage	`I2c_getFilteringStatus(reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Returns	[OUT] 1-bit output indicates the operation requested from master: Return 1 indicates read operation requested. Return 0 indicates write operation requested.
Description	Read the operation requested from master. Used in slave mode.

Example	`if(i2c_getFilteringStatus(I2C_CTRL) == 1);`
	`// Check filter hit value, bit [7] from slave address,`
	`// read ='1' write ='0'`

i2c_getInterruptFlag()

Usage	`I2c_getInterruptFlag(reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Returns	[OUT] 32-bit interrupt flags: [4] Start flag [5] Restart flag [6] End flag [7] Drop flag [15] Clock generation exit flag [16] Clock generation enter flag [17] Filter generation flag
Description	Read the 32-bit I^{2}C register interrupt flag.

Example	`Int flag = i2c_getInterruptFlag(I2C_CTRL) & I2C_INTERRUPT_DROP;`
	`// Get Drop interrupt flag from Interrupt register`

i2c_getMasterStatus()

Usage	`I2c_getMasterStatus(reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Returns	[OUT] 32-bit current master status: [0] I^{2}C controller busy [4] Start sequence in progress/requested [5] Stop sequence in progress/requested [6] Drop sequence in progress/requested [7] Recover sequence in progress/requested [9] Sequence dropped when executing start sequence [10] Sequence dropped when executing stop sequence [11] Sequence dropped when executing recover sequence
Description	Read the 32-bit I^{2}C register current master status.

Example	`int status = i2c_getMasterStatus(I2C_CTRL) & I2C_MASTER_BUSY;`
	`// Get master busy status from status register`

i2c_getSlaveStatus()

Usage	`I2c_getSlaveStatus(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Returns	[OUT] 32-bit current slave status: [0] Indicates the slave is in frame. Start sequence executed. Required stop or drop sequence to exit from frame. [1] Current state of SDA bus [2] Current state of SCL bus
Description	Read the I^{2}C bus status. This function allows the software to obtain the current state of the SDA and SCL bus.

i2c_getSlaveOverride()

Usage	`I2c_getSlaveOverride(u32 reg, u32 value)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `value` I^{2}C slave override value
Include	**driver/i2c.h**
Returns	[OUT] 32-bit slave override setting: [1] SDA bus override setting [2] SCL bus override setting
Description	Manually controls the state of SDA and SCL. Setting of zero will forcefully pull the bus low while setting of one will release the bus as the I^{2}C bus is always in pull-up condition.

i2c_applyConfig()

Usage	`void i2c_applyConfig(u32 reg, I2c_Config *config)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `config` struct of I^{2}C configuration
Include	**driver/i2c.h**
Description	Apply I^{2}C configuration to register or for initial configuration.

i2c_clearInterruptFlag()

Usage	`void i2c_clearInterruptFlag(u32 reg, u32 value)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `value` I^{2}C interrupt flag to reset **Note:** Refer to "Interrupt Clears Register: 0x0000_0024" in **Sapphire RISC-V SoC Data Sheet**.
Include	**driver/i2c.h**
Description	Clear the I^{2}C interrupt flag by setting the interrupt bit to 1.

i2c_disableInterrupt()

Usage	`void i2c_disableInterrupt(u32 reg, u32 value)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `value` I^{2}C interrupt register: [0] I2C_INTERRUPT_RX_DATA [1] I2C_INTERRUPT_RX_ACK [2] I2C_INTERRUPT_TX_DATA [3] I2C_INTERRUPT_TX_ACK [4] I2C_INTERRUPT_START [5] I2C_INTERRUPT_RESTART [6] I2C_INTERRUPT_END [7] I2C_INTERRUPT_DROP [15] I2C_INTERRUPT_CLOCK_GEN_EXIT [16] I2C_INTERRUPT_CLOCK_GEN_ENTER [17] I2C_INTERRUPT_FILTER
Include	**driver/i2c.h**
Description	Disable I^{2}C interrupt.

Example	`i2c_disableInterrupt(I2C_CTRL, I2C_INTERRUPT_TX_ACK);`
	`// Enable I2C interrupt  with interrupt TX Ack`

i2c_enableInterrupt()

Usage	`void i2c_enableInterrupt(u32 reg, u32 value)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `value` I^{2}C interrupt register: [0] I2C_INTERRUPT_RX_DATA [1] I2C_INTERRUPT_RX_ACK [2] I2C_INTERRUPT_TX_DATA [3] I2C_INTERRUPT_TX_ACK [4] I2C_INTERRUPT_START [5] I2C_INTERRUPT_RESTART [6] I2C_INTERRUPT_END [7] I2C_INTERRUPT_DROP [15] I2C_INTERRUPT_CLOCK_GEN_EXIT [16] I2C_INTERRUPT_CLOCK_GEN_ENTER [17] I2C_INTERRUPT_FILTER
Include	**driver/i2c.h**
Description	Enable I^{2}C interrupt.

Example	`i2c_enableInterrupt(I2C_CTRL, I2C_INTERRUPT_FILTER |`
	` I2C_INTERRUPT_DROP);`
	`// Enable I2C interrupt with interrupt filter and drop`

i2c_filterEnable()

Usage	`void i2c_filterEnable(u32 reg, u32 filterId, u32 config)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `filterID` filter configuration ID number [IN] `config` struct of I^{2}C configuration: • [0] Filter address 0 • [1] Filter address 1
Include	**driver/i2c.h**
Description	Enable the filter configuration.

i2c_listenAck()

Usage	`void i2c_listenAck(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Listen acknowledge from the slave.

i2c_masterBusy()

Usage	`int i2c_masterBusy(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Returns	[OUT] Integer master busy status (1-bit): Returns 0 indicates Master is available Returns 1 indicates Master is busy/in progress
Description	Get the I^{2}C busy status.

i2c_masterStatus()

Usage	`int i2c_masterStatus(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Returns	[OUT] 32-bit current master status: [0] I^{2}C controller busy [4] Start sequence in progress/requested [5] Stop sequence in progress/requested [6] Drop sequence in progress/requested [7] Recover sequence in progress/requested [9] Sequence dropped when executing start sequence [10] Sequence dropped when executing stop sequence [11] Sequence dropped when executing recover sequence
Description	Get the I^{2}C status.

i2c_masterDrop()

Usage	`void i2c_masterDrop(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Change the I^{2}C master to the drop state.
Example	`i2c_masterDrop(I2C_CTRL);`

i2c_masterStart()

Usage	`void i2c_masterStart(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Assert start condition.

i2c_masterRestart()

Usage	`void i2c_masterRestart(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Restart the I^{2}C master by sending a start condition.

i2c_masterStartBlocking()

Usage	`void i2c_masterStartBlocking(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Asserts a start condition and wait for the master to start the process.

i2c_masterRestartBlocking()

Usage	`void i2c_masterRestartBlocking(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Restart the I^{2}C master by sending a start condition. Wait for the master to start the process.

i2c_masterStop()

Usage	`void i2c_masterStop(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Asserts a stop condition.

i2c_masterStopBlocking()

Usage	`void i2c_masterStartBlocking(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Asserts a stop condition and waits for the master to start the process.

i2c_masterStopWait()

Usage	`void i2c_masterStopWait(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Waits for the master to be available.

i2c_masterRecoverBlocking()

Usage	`void i2c_masterRecoverBlocking(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	To recover the slave, toggle the SCL bus until the slave releases the SDA bus,except for a timeout. This function will retry 3 times. This function may be used as a backup plan to ensure that the slave can be recovered if a transaction fails in between.

i2c_setFilterConfig()

Usage	`void i2c_setFilterConfig(u32 reg, u32 filterId, u32 value)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `filterID` filter configuration ID number [IN] `value` filter configuration register: • [0] Filter address 0 • [1] Filter address 1 • [9:0] I2C slave address • [14] I2C_FILTER_10BITS • [15] I2C_FILTER_ENABLE
Include	**driver/i2c.h**
Description	Set the filter configuration for selected filter ID.

Example	`i2c_setFilterConfig(I2C_CTRL, 0, 0x30 | I2C_FILTER_ENABLE);`
	`// Enable filter with ID=0 slave addr = 0x30 default 7 bit filter`

i2c_txAck()

Usage	`void i2c_txAck(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Transmit acknowledge.

i2c_txAckBlocking()

Usage	`void i2c_txAckBlocking(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Transmit knowledge and wait for it to complete.

i2c_txAckWait()

Usage	`void i2c_txAckWait(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Wait for an acknowledge to transmit.

i2c_txByte()

Usage	`void i2c_txByte(u32 reg, u8 byte)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `byte` 8 bits data to send out
Include	**driver/i2c.h**
Description	Transfers one byte to the I^{2}C slave.

i2c_txByteRepeat()

Usage	`void i2c_txByteRepeat(u32 reg, u8 byte)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `byte` 8 bits data to send out
Include	**driver/i2c.h**
Description	Send a byte in repeat mode.

i2c_txNack()

Usage	`void i2c_txNack(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Transfers a NACK.

i2c_txNackRepeat()

Usage	`void i2c_txNackRepeat(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Send a NACK in repeat mode.

i2c_txNackBlocking()

Usage	`void i2c_ txNackBlocking(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Include	**driver/i2c.h**
Description	Transfer a NACK and wait for the completion.

i2c_rxAck()

Usage	`int i2c_rxAck(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Returns	[OUT] 1 bit acknowledge
Include	**driver/i2c.h**
Description	Receive an acknowledge from the I^{2}C slave.

i2c_rxData()

Usage	`u32 i2c_rxData(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Returns	[OUT] 1 byte data from I^{2}C slave
Include	**driver/i2c.h**
Description	Receive one byte data from I^{2}C slave.

i2c_rxNack()

Usage	`int i2c_rxNack(u32 reg)`
Parameters	[IN] `reg` base address of specific I^{2}C
Returns	[OUT] 1 bit no acknowledge. Return 1 if NACK is received.
Include	**driver/i2c.h**
Description	Receive no acknowledge from the I^{2}C slave.

i2c_writeData_b()

Usage	`void i2c_writeData_b(u32 reg, u8 slaveAddr, u8 regAddr, u8 *data, u32 length)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `slaveAddr` 8-bit slave address (left shift 1-bit) [IN] `regAddr` 8-bit register address [IN] `data` 8-bit write data pointer [IN] `length` number of byte of data to be transmitted
Include	**driver/i2c.h**
Description	Write a number of data with 8-bit register address.

i2c_writeData_w()

Usage	`void i2c_writeData_w(u32 reg, u8 slaveAddr, u16 regAddr, u8 *data, u32 length)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `slaveAddr` 8-bit slave address (left shift 1-bit) [IN] `regAddr` 8-bit register address [IN] `data` 8-bit write data pointer [IN] `length` number of byte of data to be transmitted
Include	**driver/i2c.h**
Description	Write a number of data with 16-bit register address.

i2c_readData_b()

Usage	`void i2c_readData_b(u32 reg, u8 slaveAddr, u8 regAddr, u8 *data, u32 length)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `slaveAddr` 8-bit slave address (left shift 1-bit) [IN] `regAddr` 8-bit register address [IN] `data` 8-bit read data pointer [IN] `length` number of byte of data to be transmitted
Include	**driver/i2c.h**
Description	Read a number of data with 8-bit register address.

i2c_readData_w()

Usage	`void i2c_readData_w(u32 reg, u8 slaveAddr, u16 regAddr, u8 *data, u32 length)`
Parameters	[IN] `reg` base address of specific I^{2}C [IN] `slaveAddr` 8-bit slave address (left shift 1-bit) [IN] `regAddr` 16-bit register address [IN] `data` 8-bit read data pointer [IN] `length` number of byte of data to be transmitted
Include	**driver/i2c.h**
Description	Read a number of data with 16-bit register address.

I/O API Calls

read_u8()

Usage	`u8 read_u8(u32 address)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address
Returns	[OUT] 8-bit data
Description	Read 8-bit data from the specified address.

read_u16()

Usage	`u16 read_u16(u32 address)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address
Returns	[OUT] 16-bit data
Description	Read 16-bit data from the specified address.

read_u32()

Usage	`u32 read_u32(u32 address)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address
Returns	[OUT] 32-bit data
Description	Read 32-bit data from the specified address.

write_u8()

Usage	`void write_u8(u8 data, u32 address)`
Include	**driver/io.h**
Parameters	[IN] `data` SoC address data [IN] `address` SoC address
Description	Write 8 bits unsigned data to the specified address.

write_u16()

Usage	`void write_u16(u16 data, u32 address)`
Include	**driver/io.h**
Parameters	[IN] `data` SoC address data [IN] `address` SoC address
Description	Write 16 bits unsigned data to the specified address.

write_u32()

Usage	`void write_u32(u32 data, u32 address)`
Include	**driver/io.h**
Parameters	[IN] `data` SoC address data [IN] `address` SoC address
Description	Write 32 bits unsigned data to the specified address.

write_u32_ad()

Usage	`void write_u32_ad(u32 address, u32 data)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address [IN] `data` SoC address data
Description	Write 32 bits unsigned data to the specified address.

Core Local Interrupt Timer API Calls

clint_setCmp()

Usage	`void clint_setCmp(u32 p, u64 cmp, u32 hart_id)`
Include	**driver/clint.h**
Parameters	[IN] `p` CLINT base address [IN] `cmp` timer compare register [IN] `hart_id` HART ID, 0 to 3
Description	Set a timer value to trigger an interrupt when the value is reached.

clint_getTime()

Usage	`u64 clint_getTime(u32 p)`
Include	**driver/clint.h**
Parameters	[IN] `p` CLINT base address
Returns	[OUT] Current core timer value
Description	Gets the timer value.

clint_uDelay()

Usage	`u64 clint_uDelay(u32 usec, u32 hz, u32 reg)`
Include	**driver/clint.h**
Parameters	[IN] `usec` microseconds [IN] `hz` core frequency [IN] `reg` CLINT base address
Description	Delay for certain duration in microsecond with CLINT.

Example	`#define bsp_uDelay(usec);`
	`clint_uDelay(usec, SYSTEM_CLINT_HZ, SYSTEM_CLINT_CTRL);`

User Timer API Calls

prescaler_setValue()

Usage	`void prescaler_setValue(u32 reg, u32 value)`
Include	**driver/prescaler.h**
Parameters	[IN] `reg` user timer base address [IN] value `prescaler` value
Description	Set the user timer prescaler value.

timer_setConfig()

Usage	`void timer_setConfig(u32 reg, u32 value)`
Include	**driver/timer.h**
Parameters	[IN] `reg` user timer base address [IN] `value` user timer configuration value: [0] Set timer to run without prescaler [1] Set timer to run with prescaler [16] Set if timer need to restart after timer limit reach
Description	Set the user timer configuration.

timer_setLimit()

Usage	`void timer_setLimit(u32 reg, u32 value)`
Include	**driver/timer.h**
Parameters	[IN] `reg` user timer base address [IN] `value` user timer configuration value
Description	Set the limit value for the timer to generate an interrupt.

timer_getValue()

Usage	`u32 timer_getValue(u32 reg)`
Include	**driver/timer.h**
Parameters	[IN] `reg` user timer base address
Returns	[OUT] 32-bit Timer value
Description	Get the timer value.

timer_clearValue()

Usage	`void timer_clearValue(u32 reg)`
Include	**driver/timer.h**
Parameters	[IN] `reg` user timer base address
Description	Clear the timer value by setting it to 0.

PLIC API Calls

plic_set_priority()

Usage	`void plic_set_priority(u32 plic, u32 gateway, u32 priority)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC base address [IN] `gateway` interrupt type. Gateway is the interrupt number for a particular peripheral when configuring the Sapphire SoC. The gateway for all peripherals are available in **soc.h**, i.e., SYSTEM_PLIC_TIMER_INTERRUPTS_0. [IN] `priority` interrupt priority. Priority can be set within a range of 0 to 3.
Description	Set the interrupt priority.

plic_get_priority()

Usage	`u32 plic_get_priority(u32 plic, u32 gateway)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC base address [IN] `gateway` interrupt type
Returns	[OUT] 32-bit priority
Description	Get the interrupt priority.

plic_set_enable()

Usage	`void plic_set_enable(u32 plic, u32 target, u32 gateway, u32 enable)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC base address [IN] `target` HART number [IN] `gateway` interrupt type [IN] `enable` Enable interrupt for the particular gateway on the selected target.
Description	Set the interrupt enable.

plic_set_threshold()

Usage	`void plic_set_threshold(u32 plic, u32 target, u32 threshold)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC base address [IN] `target` HART number [IN] `threshold` HART interrupt threshold
Description	Set the threshold of a particular HART to accept interrupt source.

Example	`plic_set_threshold(BSP_PLIC, BSP_PLIC_CPU_0, 0);`
	`// cpu 0 accept all interrupts with priority above 0`

plic_claim()

Usage	`u32 plic_claim(u32 plic, u32 target)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC base address [IN] `target` HART number
Description	Claim the PLIC interrupt for specific HART.

plic_release()

Usage	`void plic_release(u32 plic, u32 target, u32 gateway)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC base address [IN] `target` HART number [IN] `gateway` interrupt type
Description	Release the PLIC interrupt for specific HART.

SPI API Calls

SPI Config Struct

```
typedef struct{
        u32 cpol; // Clock polarity during idle state setting
        u32 cpha; // Clock phase setting
        u32 mode; // SPI Mode setting
        u32 clkDivider; // Clock divider setting on SCL generation
        u32 ssSetup; // Clock cycle between activated chip-select and first rising-edge of SCLK
        u32 ssHold; // Clock cycle between last falling-edge and deactivated chip-select is
                     //activated.
        u32 ssDisable; // Clock cycle delay before the next chip select can be activated
        } Spi_Config;
```

spi_applyConfig()

Usage	`void spi_applyConfig(u32 reg, Spi_Config *config)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address [IN] `config` struct of the SPI configuration
Description	Applies the SPI configuration to a register for initial configuration.

spi_cmdAvailability()

Usage	`u32 spi_cmdAvailability(u32 reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address
Returns	[OUT] SPI TX FIFO availability (16 bits)
Description	Reads the number of bytes for TX FIFO availability (up to 256 bytes).

spi_diselect()

Usage	`void spi_diselect(u32 reg, u32 slaveId)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address [IN] `slaveId` ID for the slave
Description	De-asserts the selected SPI (SS) pin based on the slaveId. SlaveId range from 0 up to (SPI Chip Select Width) -1. SPI 0 only have 1 chip select.

spi_read()

Usage	`u8 spi_read(u32 reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address
Returns	[OUT] One byte of data
Description	Receives one byte from the SPI slave.

spi_read32()

Usage	`u32 spi_read32(u32 reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address
Returns	[OUT] Data (up to 16 bits)
Description	Receives up to 16 bits of data from the SPI slave.

spi_rspOccupancy()

Usage	`u32 spi_rspOccupancy(u32 reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address
Returns	[OUT] SPI RX FIFO occupancy (16 bits)
Description	Read the number of bytes for RX FIFO occupancy.

spi_select()

Usage	`void spi_select(u32 reg, u32slaveId)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address [IN] `slaveId` ID for the slave
Description	Asserts the SPI select (SS) pin on the selected slave.

spi_write()

Usage	`void spi_write(u32reg, u8 data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address [IN] `data` 8 bits of data to send out
Description	Transfers one byte to the SPI slave.

spi_write32()

Usage	`void spi_write32(u32 reg, u32 data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address [IN] `data` up to 16 bits of data to send out
Description	Transfers up to 16 bits to the SPI slave.

spi_writeRead()

Usage	`u8 spi_writeRead(u32 reg, u8 data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address [IN] `data` 8 bits of data to send out
Returns	[OUT] One byte of data
Description	Transfers one byte to the SPI slave and receives one byte from the SPI slave.

spi_writeRead32()

Usage	`u32 spi_writeRead32(u32 reg, u32 data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` SPI base address [IN] `data` up to 16 bits of data to send out
Returns	[OUT] Up to 16 bits of data
Description	Transfers up to 16 bits of data to the SPI slave and receives up to 16 bits of data from the SPI slave.

SPI Flash Memory API Calls

spiFlash_f2m_()

Usage	`void spiFlash_f2m_(u32 spi, u32 flashAddress, u32 memoryAddress, u32 size)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `flashAddress` flash device start address [IN] `memoryAddress` RAM memory start address [IN] `size` programming address size
Description	Copy data from the flash device to memory.

spiFlash_f2m()

Usage	`void spiFlash_f2m(u32 spi, u32 cs, u32 flashAddress, u32 memoryAddress, u32 size)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID [IN] `flashAddress` flash device start address [IN] `memoryAddress` RAM memory start address
Description	Copy data from the flash device to memory with chip select control.

spiFlash_f2m_withGpioCs()

Usage	`void spiFlash_f2m_withGpioCs(u32 spi, Gpio_Reg *gpio, u32 cs, u32 flashAddress, u32 memoryAddress, u32 size)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `gpio` GPIO base address [IN] `cs` chip select/slaveID [IN] `flashAddress` flash device start address [IN] `memoryAddress` RAM memory start address [IN] `size` programming address size
Description	Flash device from the SPI master with GPIO chip select.

spiFlash_diselect()

Usage	`void spiFlash_diselect(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID
Description	De-asserts the SPI flash device from the master chip select.

spiFlash_diselect_withGpioCs()

Usage	`void spiFlash_diselect_withGpioCs(u32 gpio, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `gpio` GPIO base address [IN] `cs` chip select/slaveID
Description	De-asserts the SPI flash device from the master with the GPIO chip select.

spiFlash_init_()

Usage	`void spiFlash_init_(u32 spi)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address

Description	Initialize the SPI reg struct with the following default settings:
	spiCfg.cpol = 0;
	spiCfg.cpha = 0;
	spiCfg.mode = 0;
	spiCfg.clkDivider = 2;
	spiCfg.ssSetup = 2;
	spiCfg.ssHold = 2;
	spiCfg.ssDisable = 2;

spiFlash_init()

Usage	`void spiFlash_init(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID

Description	Initialize the SPI reg struct with chip select de-asserted with the following default settings:
	spiCfg.cpol = 0;
	spiCfg.cpha = 0;
	spiCfg.mode = 0;
	spiCfg.clkDivider = 2;
	spiCfg.ssSetup = 2;
	spiCfg.ssHold = 2;
	spiCfg.ssDisable = 2;

spiFlash_init_withGpioCs()

Usage	`void spiFlash_init_withGpioCs(u32 spi, u32 gpio, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `gpio` GPIO base address [IN] `cs` chip select/slaveID

Description	Initialize the SPI reg struct with GPIO chip select de-asserted with the following default settings:
	spiCfg.cpol = 0;
	spiCfg.cpha = 0;
	spiCfg.mode = 0;
	spiCfg.clkDivider = 2;
	spiCfg.ssSetup = 2;
	spiCfg.ssHold = 2;
	spiCfg.ssDisable = 2;

spiFlash_read_id_()

Usage	`u8 spiFlash_read_id_(u32 spi)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address
Returns	[OUT] 8-bit SPI flash ID
Description	Read the ID from the flash.

spiFlash_read_id()

Usage	`u8 spiFlash_read_id(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID
Returns	[OUT] 8-bit SPI flash ID
Description	Read the ID from the flash with chip select.

spiFlash_select()

Usage	`void spiFlash_select(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID
Description	Select the SPI flash device with chip select.

spiFlash_select_withGpioCs()

Usage	`spiFlash_select_withGpioCs(u32 gpio, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `gpio` GPIO base address [IN] `cs` chip select/slaveID
Description	Select the SPI flash device with the GPIO chip select.

spiFlash_software_reset()

Usage	`void spiFlash_software_reset(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID
Description	Reset the SPI flash with chip select.

spiFlash_wake_()

Usage	`void spiFlash_wake_(u32 spi)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address
Description	Release power down from the SPI master.

spiFlash_wake()

Usage	`void spiFlash_wake(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID
Description	Release power down from the SPI master with chip select.

spiFlash_wake_withGpioCs()

Usage	`void spiFlash_wake_withGpioCs(u32 spi, u32 gpio, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `gpio` GPIO base address [IN] `cs` chip select/slaveID
Description	Release power down from the SPI master with the GPIO chip select.

UART API Calls

UART Config Struct

```
typedef struct{
enum UartDataLength dataLength;
enum UartParity parity;
enum UartStop stop;
u32 clockDivider;
} Uart_Config;
```

uart_applyConfig()

Usage	`void uart_applyConfig(u32 reg, Uart_Config *config)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address [IN] `config` struct of the UART configuration
Description	Applies the UART configuration to to a register for initial configuration.

uart_TX_emptyInterruptEna()

Usage	`void uart_TX_emptyInterruptEna(u32 reg, char Ena)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address [IN] `ena` Enable interrupt
Description	Enable the TX FIFO empty interrupt.

uart_RX_NotemptyInterruptEna()

Usage	`void uart_RX_NotemptyInterruptEna(u32 reg, char Ena)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address [IN] `ena` Enable interrupt
Description	Enable the RX FIFO not empty interrupt.

uart_read()

Usage	`char uart_read(u32reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address
Returns	[OUT] `reg` character that is read
Description	Reads a character from the UART slave.

uart_readOccupancy()

Usage	`u32 uart_readOccupancy(u32reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address
Returns	[OUT] `reg` FIFO occupancy
Description	Read the number of bytes in the RX FIFO up to 128 bytes.

uart_status_read()

Usage	`u32 uart_status_read(u32 reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address
Returns	[OUT] 32-bit status register from the UART
Description	Refers to UART Status Register: 0x0000_0004 in the Sapphire Datasheet.

uart_status_write()

Usage	`void uart_status_write(u32 reg, char data)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address [IN] `data` input data for the UART status.
Description	Write the UART status. Only TXInterruptEnable and RXInterruptEnable are writable.

uart_write()

Usage	`void uart_write(u32 reg, char data)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address [IN] `data` write a character
Description	Write a character to the UART.

uart_writeHex()

Usage	`void uart_writeHex(u32 reg, int value)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address [IN] `value` number to send as UART character
Description	Convert a number to a character and send it to the UART in hexadecimal.

uart_writeStr()

Usage	`void uart_writeStr(u32 reg, const char* str)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address [IN] `str` string to write
Description	Write a string to the UART.

uart_writeAvailability()

Usage	`u32 uart_writeAvailability(u32 reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` UART base address
Returns	[OUT] `reg` FIFO availability
Description	Read the number of bytes in the TX FIFO up to 128 bytes.

RISC-V API Calls

data_cache_invalidate_all()

Usage	`void data_cache_invalidate_all(void)`
Include	**driver/vexriscv.h**
Description	Invalidate whole data cache. Critical to ensure the data coherency between the cache and the main memory.

data_cache_invalidate_address()

Usage	`void data_cache_invalidate_address(address)`
Include	**driver/vexriscv.h**
Description	Invalidate the address data cache. Critical to ensure the data coherency between the cache and the main memory.

instruction_cache_invalidate()

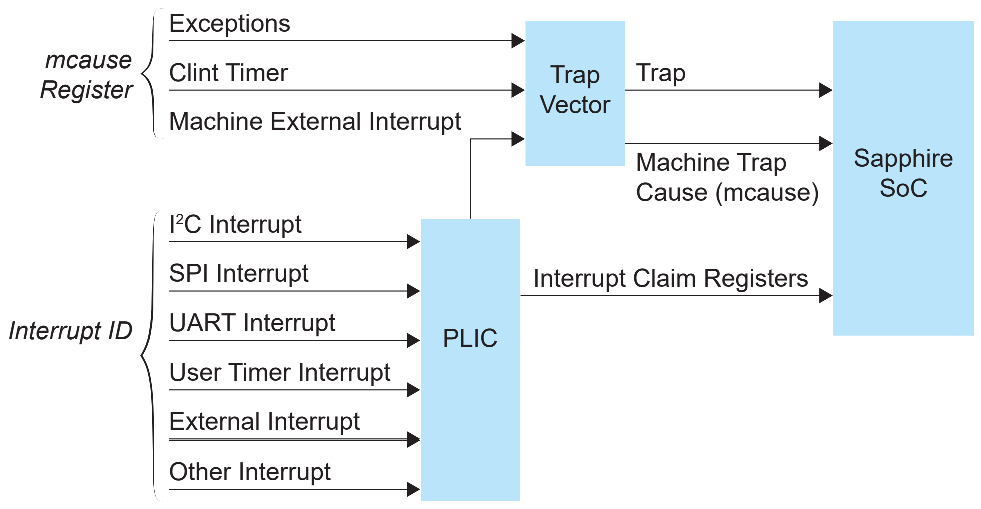
Usage	`void instruction_cache_invalidate(void)`
Include	**driver/vexriscv.h**
Description	Invalidate the whole instruction cache. Critical to ensure the instruction coherency between the cache and the main memory.

Note: For more information on the usage of the cache invalidation API, see iCacheFlushDemo and dCacheFlushDemo.

Handling Interrupts

There are two kinds of interrupts, trap vectors and PLIC interrupts, and you handle them using different methods.

Figure 49: Types of Interrupts

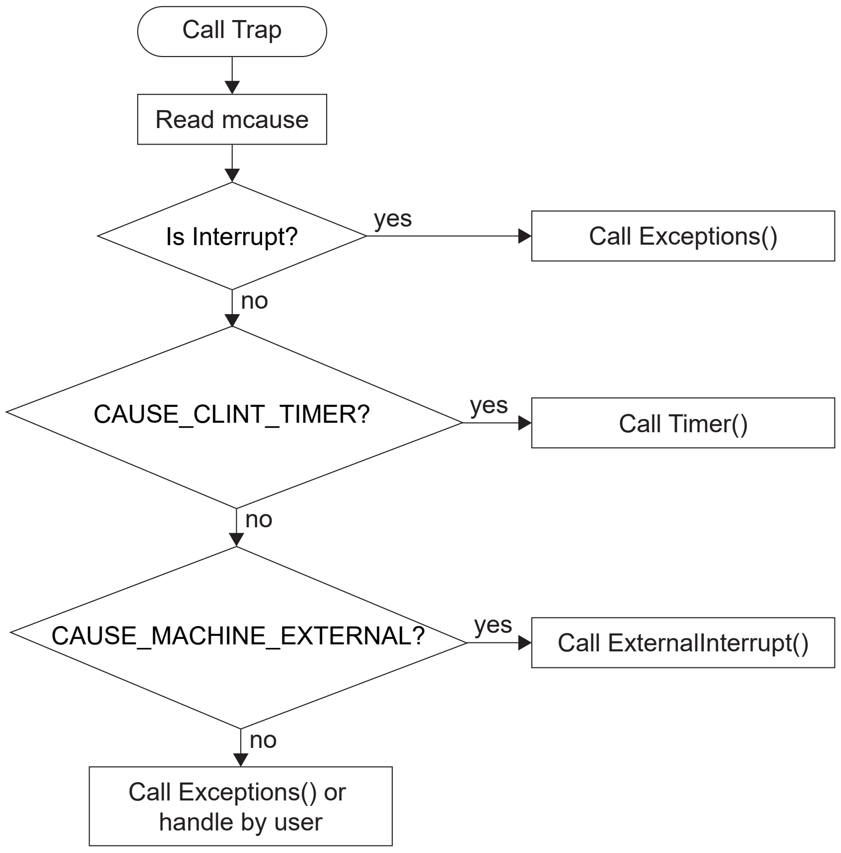


Trap Vectors

Trap vectors trap interrupts or exceptions from the system. Read the Machine Cause Register (`mcause`) to identify which type of interrupt or exception fthe system is generating. Refer to "Machine Cause Register (mcause): 0x342" in the data sheet for your SoC for a list of the exceptions and interrupts used for trap vectors. The following flow chart explains how to handle trap vectors.

For CAUSE_MACHINE_EXTERNAL, it will call the subroutine to process the PLIC level interrupts.

Figure 50: Handling Trap Vectors



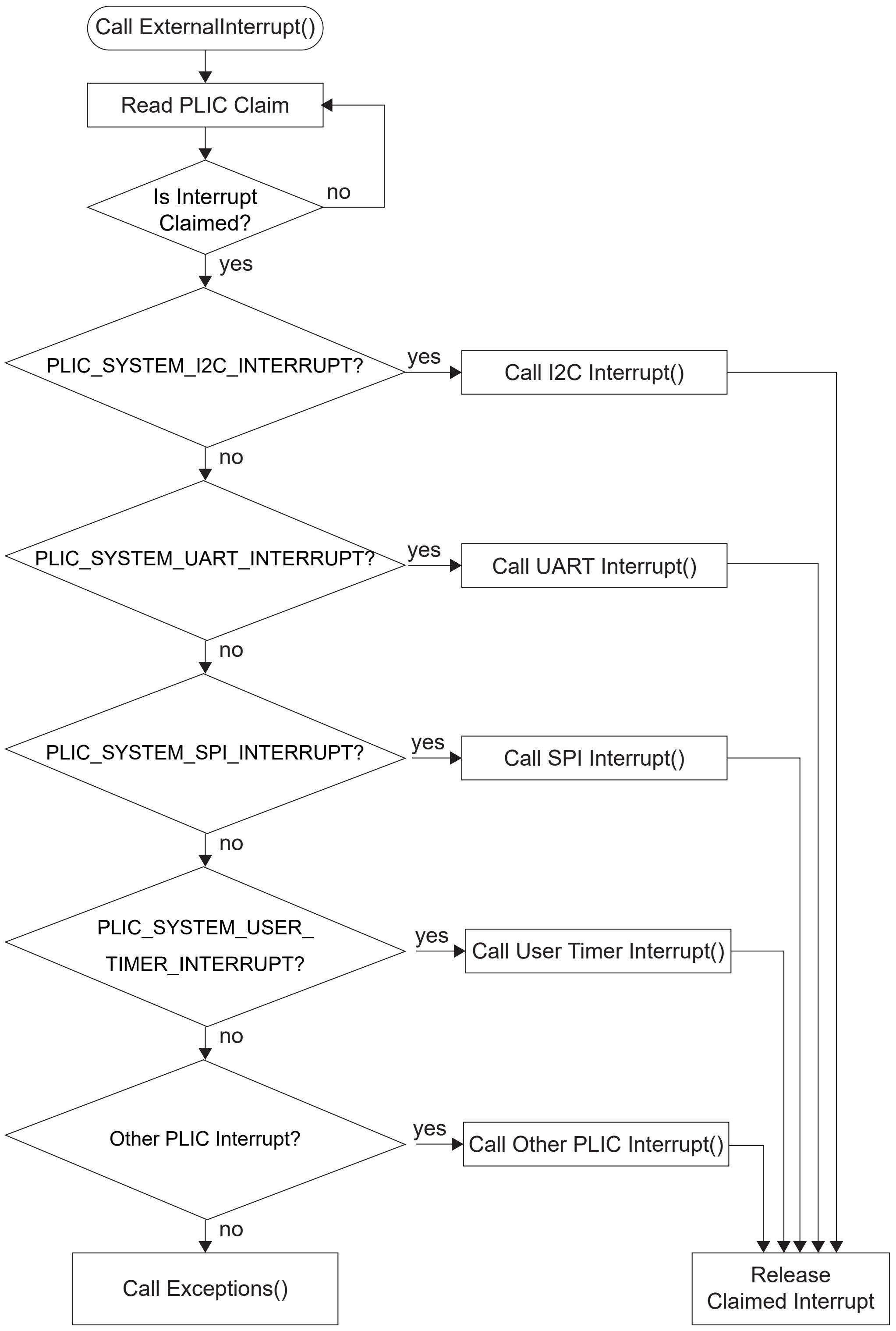
PLIC Interrupts

The PLIC collects external interrupts and is also used for CAUSE_MACHINE_EXTERNAL cases. Read the interrupt claim registers (PLIC claim) to identify the source of the external interrupt. Refer to Address Map on page 81 for a list of the interrupt IDs.

Note: For the Sapphire SoC, the interrupt IDs are user configurable. Refer to the interrupt IDs that you set in the IP Manager for each peripheral. The Address Map shows the default values.

The following flow chart shows how the PLIC handles interrupts.The PLIC identifies the interrupt ID and processes the corresponding interrupts.

Figure 51: Handling PLIC Interrupts


Call ExternalInterrupt()
Read PLIC Claim
Is Interrupt Claimed?
no
yes
PLIC_SYSTEM_I2C_INTERRUPT?
yes
Call I2C Interrupt()
no
PLIC_SYSTEM_UART_INTERRUPT?
yes
Call UART Interrupt()
no
PLIC_SYSTEM_SPI_INTERRUPT?
yes
Call SPI Interrupt()
no
PLIC_SYSTEM_USER_ TIMER_INTERRUPT?
yes
Call User Timer Interrupt()
no
Other PLIC Interrupt?
yes
Call Other PLIC Interrupt()
no
Call Exceptions()
Release Claimed Interrupt

Chapter 18

Inline Assembly

Contents:

Introduction

The inline assembly is a feature in programming languages like C and C++ that allows you to embed assembly language code directly within your high-level code. This feature allows you to write your assembly instructions in line with your C or C++ code, instead of having to write and compile the assembly language file separately. This is useful in situations that need fine-grained control over hardware resources or performing low-level operations that are not easily expressed in higher-level languages. Typically, inline assembly can be useful if you need to:

- *Access hardware resources*—Inline assembly allows you access to hardware resources that is unaccessible or does not have suitable intrinsic function available in high-level language.
- *Performance optimization*—You may use inline assembly to design sections of code that are time-critical and more optimized than high-level language.

CAUTION: Inline assembly is a powerful tool for low-level operations and optimization. However, inline assembly can make your design harder to maintain. Therefore, you need to use it with caution and sparingly.

Note: All inline assembly syntax explained in this user guide is based on GNU GCC v8.3.0, which is the out-of-box toolchain used by Efinity RISC-V Embedded Software IDE. Refer to **GNU GCC Online Documentation** for more information.

Inline Assembly Syntax

The inline assembler has the following syntax:

```
_asm_<asm-qualifiers>
(
"assembly_instructions_string"
:"output_operand_list"
:"input_opearand_list"
:"clobbered_resource_list"
);
```

Table 37: Inline Assembly Syntax

Syntax	Description
asm	Indicates the start of the inline assembly block.
asm-qualifiers	Optional qualifiers that you can use to specify various attributes of the inline assembly, such as constraints, options, or flags, e.g., _volatile_

"assembly_instructions_string"	Specify the actual assembly code as a string separated by /n. Each operation can be a valid assembler instruction, or a data definition assembler directive prefixed by an optional label. There can be no whitespace before the label, and it must be followed by ":". For example:
	`_asm_ _volatile_`
	`(`
	`"label:"`
	`"nop/n"`
	`"j label"`
	`);`
	Note:
	• The labels you define in the inline assembler statement is categorized as local with reference to the respective statement.
	• Use this to implement loops or conditional code.

:"output_operand_list"	Defines the output operands of the assembly code. Output operands are used to pass values from the assembly code back to the C/C++ code. They are specified as a comma-separated list. The "output_operand_list" typically consists of variables or registers where the results of the assembly instructions will be stored.
:"input_operand_list"	Defines the input operands of the assembly code. Input operands are used to pass values from the C/C++ code to the assembly code. Like the output operands, the "input_operand_list" is a comma-separated list of variables or registers used as inputs to the assembly instructions.
:"clobbered_resource_list"	Specifies clobbered resources, which are registers or memory locations that may be modified by the assembly code but are not explicitly listed as input or output operands. The "clobbered_resource_list" is also a comma-separated list, and it informs the compiler that is should not rely on the values of these resources after the inline assembly block. This is an optional part, and if there are no clobbered resources, it can be left empty.

Operands

An inline assembler statement can have one input and one output comma-separated list of operands. Each operand consists of an optional symbolic name in brackets, a quoted constraint, followed by a C expression parentheses.

Operand Syntax

The representation of an operand syntax is as follows:

[<symbolic-name>] "<modifiers> <constraints>" (expr)

Example 1:

```
int Add (int term1, int term2)
{
     int sum;
     _asm_ _volatile_
     (
     "add %0, %1, %2"
     : "=r" (sum)
     : "r" (term1), "r" (term2)
     );
     return sum;
}
```

Table 38: Explanation of Example 1

C Function Implementation	Description
`Add()`	This function uses inline assembly to perform an addition operation. Inputs two integer parameters, `term1` and `term2`, and returns the result as a `sum`.
`add %0, %1, %2`	This is the assembly instruction. It adds two integer parameters, `term1` and `term2`, and stores the result in the output operand `%0` (which corresponds to `sum` in this case). `%1` and `%2` are placeholders for input operands, which are `term1` and `term2` respectively.
`"=r" (sum)`	This is an output operand constraint. It tells the compiler that the assembly instruction modifies the `sum` variable and should be stored in a general-purpose register (`r`).

`"=r" (term1),`
`"=r" (term2)`
	These are input operand constraints. They specify that `term1` and `term2` should be stored in registers (`r`) and are used as input to the assembly instruction.

You can omit any C function implementation by leaving it empty as shown by the following example.

Example 2:

```
int matrix [M][N];
void MatrixPreloadNow (int row)
{
     _asm_ _volatile_
     (
     "lw t0, 0(%0)"
     :                //empty//
     : "r" (%matrix [row] [0])
     );
}
```

The code in Example 2 loads the `%0` data into temporary register, `t0`. The assembly only provides the input constraint and provides nothing to the output constraint. The pointer uses the data from `&matrix[row][0]`.

Operand References

The placeholders, `%0`, `%1`, etc., are known as operand references or substitution operands. These placeholders represent input and output operands within the inline asembly code. The numbers inside the placeholders correspond to the sequence of operands specified in the constraints. The following is the example of its usage.

Example 3:

```
int Add (int term1, int term2)
{
     int sum;
     _asm_ _volatile_
     (
     "add %0, %1, %2"
     : "=r" (sum)
     : "r" (term1), "r" (term2)
     );
     return sum;
}
```

In the Add function from Example 3, `%0` is used to represent the output operands, which is the integer, `sum`. The `%1` represents the input operand, `term1` while `%2` represents the input operand, `term2`.

Input Operands

The characteristics of input operands are as follows:

The input operands cannot have any constraint modifiers, but they can have any valid C expression if the type of the expression fits the register.

The C expression is evaluated just before any of the assembler instructions in the inline assembler statement and assigned to the constraint, for example a register.

Output Operands

The characteristics of output operands are as follows:

- Output operands must have "=" as a constraint modifier and the C expression must be a l-value and specify writable location. For example, "`=r`" for a write-only general-purpose register.
- The constraint is assigned to the evaluated C expression (as a l-value) immediately after the last assembler instruction in the inline assembler statement.
- Input operands are assumed to be consumed before output is produced.
- The compiler may use the same register for an input and output operand.
- To prohibit this, prefix the output constraint with "`&`" to make it an early clobber resource. For example, "`=&r`".

The above characteristics ensure that the output operand is allocated to a different register from the input operands.

Input/Output Operands

The characteristics of input/output operands are as follows:

- An operand that should be used both for input and output must be listed as an output operand and have the "+" modifier.
- The C expression must be a l-value and specify a writable location.
- The location is read immediately before any assembler instructions, and is written right after the last assembler instruction.

Example of using a read-write operand:

Example 4:

```
int Double (int value)
{
     _asm_ _volatile_
     (
     "add %0, %0, %0"
     : "+r" (value)
     );
     return value;
}
```

In Example 4, the input `value` is placed in a general-purpose register. After the assembler statement, the result from the `add` instruction is placed in the same register and return the result.

Operand Constraints

A constraint is a string full of letters, each of which describes one kind of operand that is permitted.

Table 39: Inline Assembler Operand Constraints

Constraint Syntax	Description
A	An address that is held in a general-purpose register.
m	Memory.
r	Uses a general-purpose integer register for the expression: *x1-x31*
i	A 32-bit integer.
I	An I-type 12-bit signed integer.
J	The constant integer zero.
K	A 5-bit unsigned integer for CSR instructions.
f	Uses a general-purpose floating-point register.
register_name	Uses this specific register for the expression.
digit	• The input must be in the same location as the output operand *digit*. • If a digit is used together with letters within the same alternative, then the digit should come last.

Note: For the full lists of operand constraints, refer to the GNU GCC documentation.

Operand Constraint Modifiers

The constraint modifiers can be used together with a constraint to modify its meaning. The modifier should put in the first character of the constraint string. The following table lists the supported constraint modifiers:

Table 40: Supported Constraint Modifiers

Modifier Syntax	Description
+	Read-write operand.
=	Write-only operand: the previous value is discarded and replaced by new data.
&	This operand is an earlyclobber operand, which is written to before the instruction has processed all the input operands.

Note: The compiler can only handle one commutative (constraint) pair in an assembly. The compiler may fail if you use more than one commutative pair.

Clobbered Resources

The characteristics of clobbered resources are as follows:

- An inline assembler statement can contain a list of clobbered resources.
- The clobbered registers that can be thrashed need to be specified in the assembly statement.
- By optimizing the GCC, you can specify or check for the clobbered registers.
- Any value that resides in a clobbered resource and that is needed after the inline assembly statement is reloaded.

Note: Clobbered resources is used as input or output operands.

Example of using clobbered resources:

Example 5:

```
int Add0x10000 (int term)
{
     int sum;
     __asm__ __volatile__
     (
     "lui s0, 0x10\n"
     "add %0, %1, s0"
     : "=r" (sum)
     : "r" (term)
     : "s0"
     );
     return sum;
}
```

The following table lists the valid clobbered resources:

Table 41: Lists of Valid Clobbered Resources

Clobber	Description
x1-x3, a0-a7, s0-s11, t0-t6	General-purpose integer registers.
f0-f31, fa0-fa7, fs0-fs11, ft0-ft11	General-purpose floating-point registers.
Memory	To be used if the instructions modify any memory. This avoids keeping memory values cached in registers across the inline assembler statement.

Example of using clobbered memory:

Example 6:

```
void Store (unsigned long*location, unsigned long value)
{
     _asm_ _volatile_
     (
     "sw %1, 0(%0)"
     :
     : "=r" (location), "r" (value)
     : "memory"
     );
}
```

RISC-V Registers

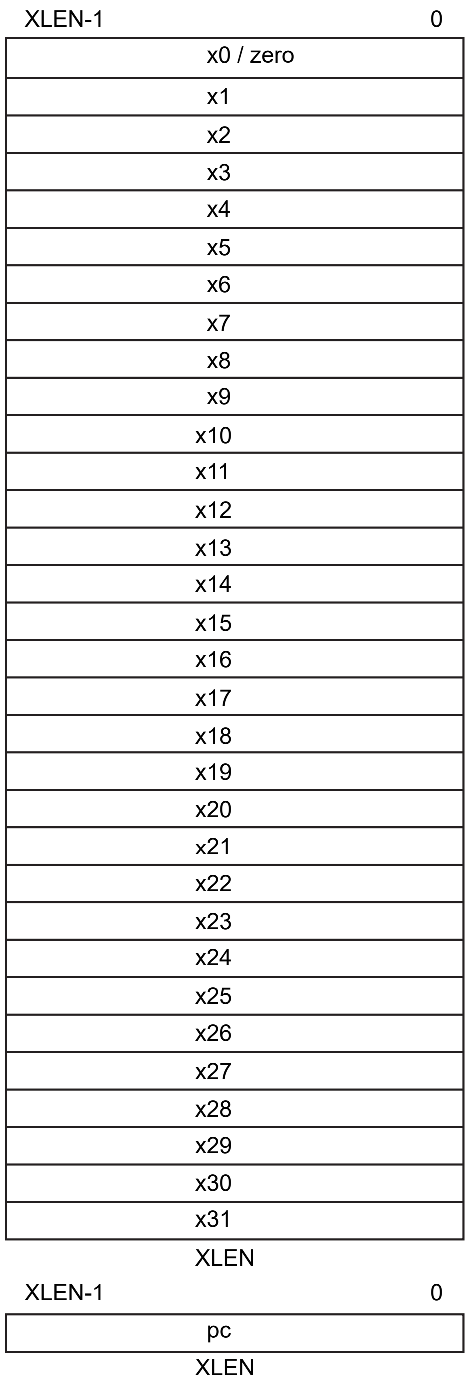
RISC-V has the following 32-bit registers:
- 32 general-purpose registers
- A program counter (PC)

A 32 general-purpose registers have the following assigned functions:
- x0 is hard-wired to 0 and can be used as a target register for any instructions where the result must be discarded.
- x0 can also be used as a source of zero (0) if needed.
- x1-x31 are general-purpose registers. The 32-bit integers they hold are interpreted, depending on the instruction.

A PC has the following assigned functions and characteristics:
- PC points to the next instruction to be executed.
- The PC cannot be written or read using load/store instructions.

The following figure shows the 32 general-purpose registers in a RISC-V ISA[6] CPU.

Figure 52: RISC-V Base Unprivileged Integer Register State



[6] ISA: Instruction Set Architecture

Calling Convention for RISC-V Registers

The symbolic name in the table is the name used by the RISC-V register when applying the inline assembly in the design.

Table 42: Symbolic Names in RISC-V General Purpose Registers

Register Name	Symbolic Name	Description
x0	Zero	Hardwired zero register, always read as zero (0), and writes are ignored.
x1	Ra	Return address register, used to store the return address.
x2	Sp	Stack pointer register, used to point to the top of the call stack.
x3	Gp	Global pointer register, used to addressing global data.
x4	Tp	Thread pointer, used for addressing thread-local data.
x5	t0	Temporary register/alternate link register, used for general temporary storage.
x6-x7	t1-t2	Temporary registers, used for general temporary storage.
x8	s0/fp	Saved register/frame pointer, often used to establish and maintain stack frames.
x9	s1	Saved register, used for saving and restoring values across function calls.
x10-x11	a0-a1	Function argument registers/return value register.
x12-x17	a2-a7	Function argument registers.
x18-x27	s2-s11	Saved registers, used for saving and restoring values across function calls.
x28-x31	t3-t6	Temporary registers, often used for general temporary storage.

Note: Ensure correct registers are used when designing your program to avoid any data corruption.

Efinix provides an example design that focuses on the implementation of these inline assembly features for RISC-V Sapphire SoC core. You can refer to the **InlineAsmDemo** example design which is generated alongside with the Sapphire SoC core.

Appendix: Required Software for Eclipse (RISC-V SDK)

These instructions are for reference if you are using open-source Eclipse IDE provided with RISC-V SDK.

RISC-V SDK

Eclipse MCU—Open-source Java-based development environment that uses plug-ins to extend and customize its functionality. The GNU MCU Eclipse plug-in lets you develop applications for ARM and RISC-V cores.

Version: 2020-09 (4.17.0)

Disk space required: 433 MB (Windows), 433 MB (Linux)

xPack GNU RISC-V Embedded GCC—Open-source, prebuilt toolchain from the xPack Project.

Version: 8.3.0-2.3

Disk space required: 1.53 GB (Windows), 1.5 GB (Linux)

OpenOCD Debugger—The open-source Open On-Chip Debugger (OpenOCD) software includes configuration files for many debug adapters, chips, and boards. Many versions of OpenOCD are available. The Efinix RISC-V flow requires a custom version of OpenOCD that includes the VexRiscv 32-bit RISC-V processor.

Version: 20200421

Disk space required: 9.4 MB (Windows), 7.4 MB (Linux)

GNU MCU Eclipse Windows Build Tool (Windows Only)—This open-source Windows-specific package helps to manage build projects and includes GNU make.

Version: 4.2.1-2-win32-x64

Disk space required: 4.99 MB

Java JRE

Open-source Java 64-bit runtime environment; required for Eclipse.

Version: 8 Update 241

https://www.java.com/en/download/manual.jsp (Java 8 official release)

https://developers.redhat.com/products/openjdk/download (OpenJDK 8 or 11)

http://jdk.java.net/16/ (OpenJDK 16)

Install the RISC-V SDK

To install the SDK:

1. Download the file **riscv_sdk_windows-v<version>.zip** or **riscv_sdk_ubuntu-v<version>.zip** from the Support Center.
2. Create a directory for the SDK, such as **c:\riscv-sdk** (Windows) or **home/my_name/riscv-sdk** (Linux).
3. Unzip the file into the directory you created. The complete SDK is distributed as compressed files. You do not need to run an installer.

Windows directory structure:

- **SDK_Windows**
  - **eclipse**—Eclipse application.
  - **GNU MCU Eclipse**—Windows build tools.
  - **openocd**—OpenOCD debugger.
  - **riscv-xpack-toolchain_8.3.0-2.3_windows**—GCC compiler.
  - **run_eclipse.bat**—Batch file that sets variables and launches Eclipse.
  - **setup.bat**—Batch file to set variables for running OpenOCD on the command line to flash the binary.

Ubuntu directory structure:

- **SDK_Ubuntu<version>**
  - **eclipse**—Eclipse application.
  - **openocd**—OpenOCD debugger.
  - **riscv-xpack-toolchain_8.3.0-2.3_linux**—GCC compiler.
  - **run_eclipse.sh**—Shell file that sets variables and launches Eclipse.
  - **setup.sh**—Shell file to set variables for running OpenOCD on the command line to flash the binary.

Install the Java JRE

To install the JRE:

1. Download the 64-bit version of the JRE or JDK for your operating system from
   https://www.java.com/en/download/manual.jsp (Java 8 official release)
   https://developers.redhat.com/products/openjdk/download (OpenJDK 8 or 11)
   http://jdk.java.net/16/ (OpenJDK 16)
2. Follow the installation instructions on the web site to install the JRE.

Note: You need a 64-bit version of the Java JRE. When you launch Eclipse using a 32-bit version, you get an error that Java quits with exit code `13`.

Appendix: Launch Eclipse (RISC-V SDK)

These instructions are for reference if you are using open-source Eclipse IDE provided with RISC-V SDK.

The RISC-V SDK includes the **run_eclipse.bat** file (Windows) or **run_eclipse.sh** file (Linux) that adds executables to your path, sets up envonment variables for the Sapphire BSP, and launches Eclipse. Always use this executable to launch Eclipse; do not launch Eclipse directly.

When you first start working with the Sapphire SoC, you need to configure your Eclipse workspace and environment. Setting up a global development environment for your workspace means you can store all of your Sapphire software code in the same place and you can set global environment variables that apply to all software projects in your workspace.

You should use a unique workspace for your Sapphire SoC projects. Efinix recommends using the **embedded_sw/<SoC module>** directory as the workspace directory.

Note: With IP Manager, you can generate multiple SoCs with different options. Using the **embedded_sw/<SoC module>** directory as your workspace means that you can explore more than one SoC by simply switching workspaces.

Follow these steps to launch Eclipse and set up your workspace:

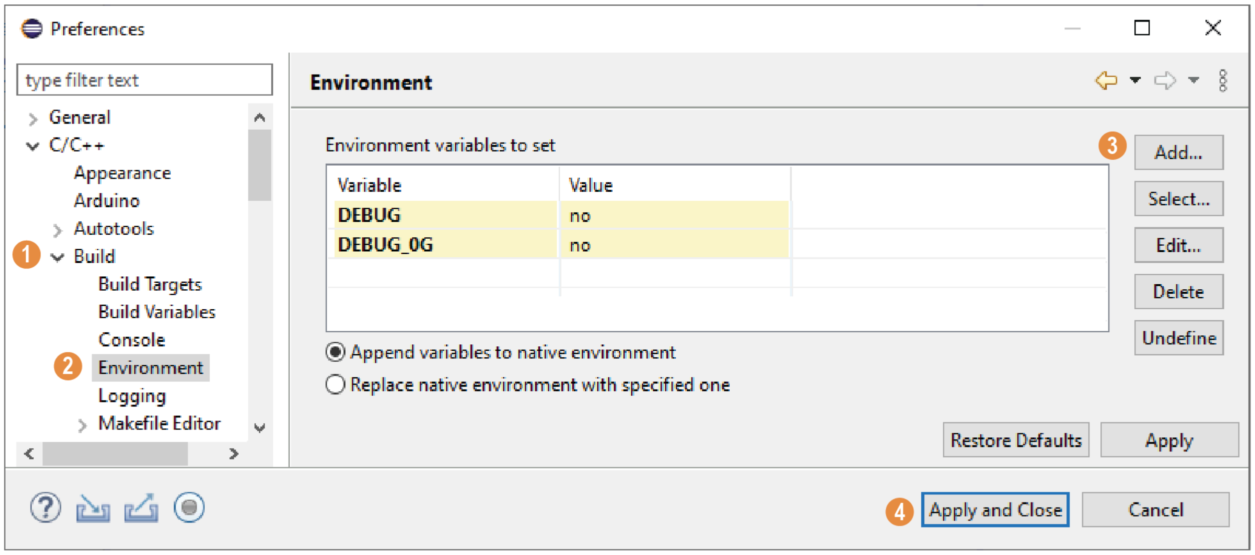
1. Launch Eclipse using the **run_eclipse.bat** file (Windows) or **run_eclipse.sh** file.
2. If this is the first time you are running Eclipse, create a new workspace that points to the **embedded_sw/<SoC module>** directory. Otherwise, choose **File > Switch Workspace > Other** to choose an existing workspace directory and click **Launch**.

Set Global Environment Variables

OpenOCD uses two environment variables, `DEBUG` and `DEBUG_OG`. It is simplest to set them as global environment variables for all projects in your workspace. Then, you can adjust them as needed for individual projects.

Note: When you configure the SoC in the IP Manager, you can choose whether to turn on debug mode by default or not. When you generate the SoC, the setting is saved in the **/embedded_sw/bsp/efinix/EfxSapphireSoc/include/soc.mk** file. If you want to change the debug mode, you can change the setting in the IP Configuration wizard and re-generate the SoC, or use the following instructions to add the variables to your project and change them there.

Choose **Window** > **Preferences** to open the **Preferences** window and perform the following steps.



Environment variables to set

Variable	Value
DEBUG	no
DEBUG_OG	no

(●) Append variables to native environment
( ) Replace native environment with specified one

1. In the left navigation menu, expand **C/C++** > **Build**.
2. Click **C/C++** > **Build** > **Environment**.
3. Click **Add** and add the following environment variables:

Variable	Value	Description
DEBUG	no	Enables or disables debug mode. no: Debugging is turned off yes: Debugging is enabled
DEBUG_OG	no	Enables or disables optimization during debugging. Use an uppercase letter O not a zero.

4. Click **Apply and Close**.

Appendix: Create and Build a Software Project (RISC-V SDK)

After you set up your Eclipse workspace, you are ready to create a new project and build it. These instructions walk you through the process using the **axiDemo** example project from the **software** directory.

Create a New Project

In this step you create a new project from the **axiDemo** code example.

1. Launch Eclipse.
2. Select the Sapphire workspace if it is not open by default.
3. Make sure you are in the C/C++ perspective.

Import the **axiDemo** example:

4. Choose **File** > **New** > **Makefile Project with Existing Code**.
5. Click **Browse** next to **Existing Code Location**.
6. Browse to the **software/standalone/axiDemo** directory and click **Select Folder**.
7. Select **<none>** in the **Toolchain for Indexer Settings** box.
8. Click **Finish**.

The **axiDemo** project folder displays in the Project Explorer. The folder has the makefile and **main.c** source code as well as launch scripts for the OpenOCD Debugger.

Import Project Settings (Optional)

Efinix provides a C/C++ project settings file that defines the include paths and symbols for the C code. Importing these settings into your project lets you explore and jump through the code easily.

Note: You are not required to import the project settings to build. These settings simply make it easier for you to write and debug code.

To import the settings:

1. Choose **File** > **Import** to open the **Import** wizard.
2. Expand **C/C++**.
3. Choose **C/C++** > **C/C++ Project Settings**.
4. Click **Next**.
5. Click **Browse** next to the **Settings file** box.
6. Browse to one of the following files and click **Open**:

Option	Description
Windows	embedded_sw\<SoC module>\config\project_settings_soc.xml
Linux	embedded_sw/<SoC module>/config_linux/project_settings_soc.xml

7. In the **Select Project** box, select the project name(s) for which you want to import the settings.

8. Click **Finish**.

Eclipse creates a new folder in your project named **Includes**, which contains all of the files the project uses.

After you import the settings, clean your project (**Project > Clean**) and then build (**Project > Build Project**). The build process indexes all of the files so they are linked in your project.

Enable Debugging

If you chose **OpenOCD Debug Mode > Turn On by default** when you configured the SoC, debugging is turned on and you can skip the instructions in this topic.

If you chose **OpenOCD Debug Mode > Turn Off by default** when you configured the SoC, debugging is turned off. Add the environment variables as described in Set Global Environment Variables on page 182 and then change the variables as needed.

- To run the program for normal operation, keep **DEBUG** set to **no**.
- To debug with the OpenOCD debugger, set **DEBUG** to **yes**.

In debug mode, the program suspends operation after loading so that you can set breakpoints or perform debug tasks.

To change the debug settings for your project, right-click the project name **axiDemo** in the Project Explorer and choose **Properties** from the pop-up menu.

1. Expand **C/C++ Build**.
2. Click **C/C++ Build > Environment**.
3. Click the **Debug** variable.
4. Click **Edit**.
5. Change the **Value** to `yes`.
6. Click **OK**.
7. Click **Apply and Close**.

Important: When you change the debug value for a project you previously built, you must clean the project (**Project** > **Clean**) before building again. Otherwise, Eclipse gives a message in the Console that there is `Nothing to be done for 'all'`

Build

Choose **Project > Build Project** or click the Build Project toolbar button.

The **makefile** builds the project and generates these files in the **build** directory:

- **axiDemo.asm**—Assembly language file for the firmware.
- **axiDemo.bin**—Download this file to the flash device on your board using OpenOCD. When you turn the board on, the SoC loads the application into the RISC-V processor and executes it.
- **axiDemo.elf**—Use this file when debugging with the OpenOCD debugger.
- **axiDemo.hex**—Hex file for the firmware. (Do not use it to program the FPGA.)
- **axiDemo.map**—Contains the SoC address map.

Appendix: Debug with the OpenOCD Debugger (RISC-V SDK)

These instructions are for reference if you are using an earlier software version. With the development board programmed and the software built, you are ready to configure the OpenOCD debugger and perform debugging. These instructions use the **axiDemo** example to explain the steps required.

Launch the Debug Script

With the Efinity software v2022.1 and higher, debugging scripts are available for each software example in the **/embedded_sw/<module>/software/standalone/** directory and are imported into your project when you create a new project from an existing makefile. You can use these scripts to launch debug mode.

Table 43: Debug Configurations

Launch Script	Description
axiDemo_trion.launch	Debugging software on Trion® development boards.
axiDemo_ti.launch	Debugging software on Titanium development boards
axiDemo_softTap.launch	Debugging software on Trion or Titanium development boards with the soft JTAG TAP interface. For example, you would need to use the soft TAP if you want to use the OpenOCD debugger and the Efinity® Debugger at the same time. (See Using a Soft JTAG Core for Example Designs on page 112.)

To debug the axiDemo project:

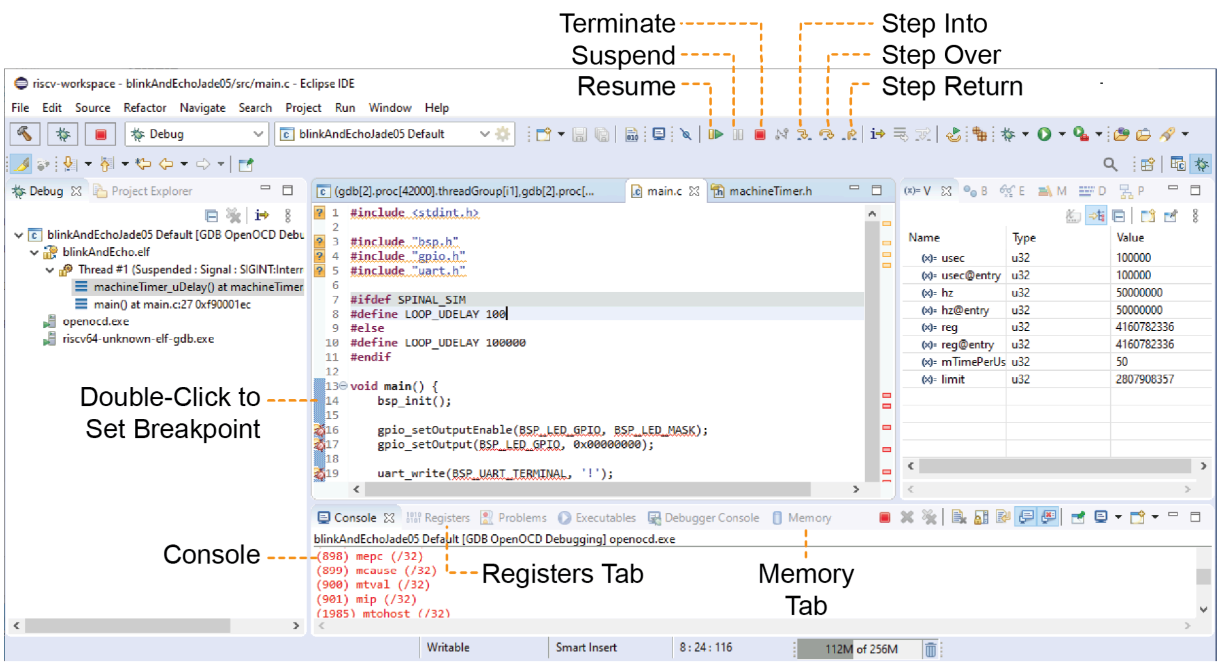
1. Right-click **axiDemo** > **axiDemo_<family>.launch**.
2. Choose **Debug As** > > **axiDemo_<family>**. Eclipse launches the OpenOCD debugger for the project.
3. Click **Debug**.

Debug

After you click **Debug** in the Debug Configuration window, the OpenOCD server starts, connects to the target, starts the gdb client, downloads the application, and starts the debugging session. Messages and a list of VexRiscv registers display in the **Console**. The **main.c** file opens so you can debug each step.

1. Click the **Resume** button or press F8 to resume code operation. All of the LEDs on the board blink continuously in unison.
2. Click **Step Over** (F6) to do a single step over one source instruction.
3. Click **Step Into** (F5) to do a single step into the next function called.
4. Click **Step Return** (F7) to do a single step out of the current function.
5. Double-click in the bar to the left of the source code to set a breakpoint. Double-click a breakpoint to remove it.
6. Click the **Registers** tab to inspect the processor's registers.
7. Click the **Memory** tab to inspect the memory contents.
8. Click the **Suspend** button to stop the code operation.
9. When you finish debugging, click **Terminate** to disconnect the OpenOCD debugger.

Figure 53: Perform Debugging


Terminate
Suspend
Resume
Step Into
Step Over
Step Return
riscv-workspace - blinkAndEchoJade05/src/main.c - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
blinkAndEchoJade05 Default
Debug
Project Explorer
blinkAndEchoJade05 Default [GDB OpenOCD Debu
blinkAndEcho.elf
Thread #1 (Suspended : Signal : SIGINT:Interr
machineTimer_uDelay() at machineTimer
main() at main.c:27 0xf90001ec
openocd.exe
riscv64-unknown-elf-gdb.exe
(gdb[2].proc[42000].threadGroup[i1],gdb[2].proc[...
main.c
machineTimer.h
Double-Click to Set Breakpoint
Console
Registers
Problems
Executables
Debugger Console
Memory
Registers Tab
Memory Tab
Writable
Smart Insert
8 : 24 : 116
112M of 256M

```
#include <stdint.h>

#include "bsp.h"
#include "gpio.h"
#include "uart.h"

#ifdef SPINAL_SIM
#define LOOP_UDELAY 100
#else
#define LOOP_UDELAY 100000
#endif

void main() {
    bsp_init();

    gpio_setOutputEnable(BSP_LED_GPIO, BSP_LED_MASK);
    gpio_setOutput(BSP_LED_GPIO, 0x00000000);

    uart_write(BSP_UART_TERMINAL, '!');
```

Name	Type	Value
usec	u32	100000
usec@entry	u32	100000
hz	u32	50000000
hz@entry	u32	50000000
reg	u32	4160782336
reg@entry	u32	4160782336
mTimePerUs	u32	50
limit	u32	2807908357

```
blinkAndEchoJade05 Default [GDB OpenOCD Debugging] openocd.exe
(898) mepc (/32)
(899) mcause (/32)
(900) mtval (/32)
(901) mip (/32)
(1985) mtohost (/32)
```

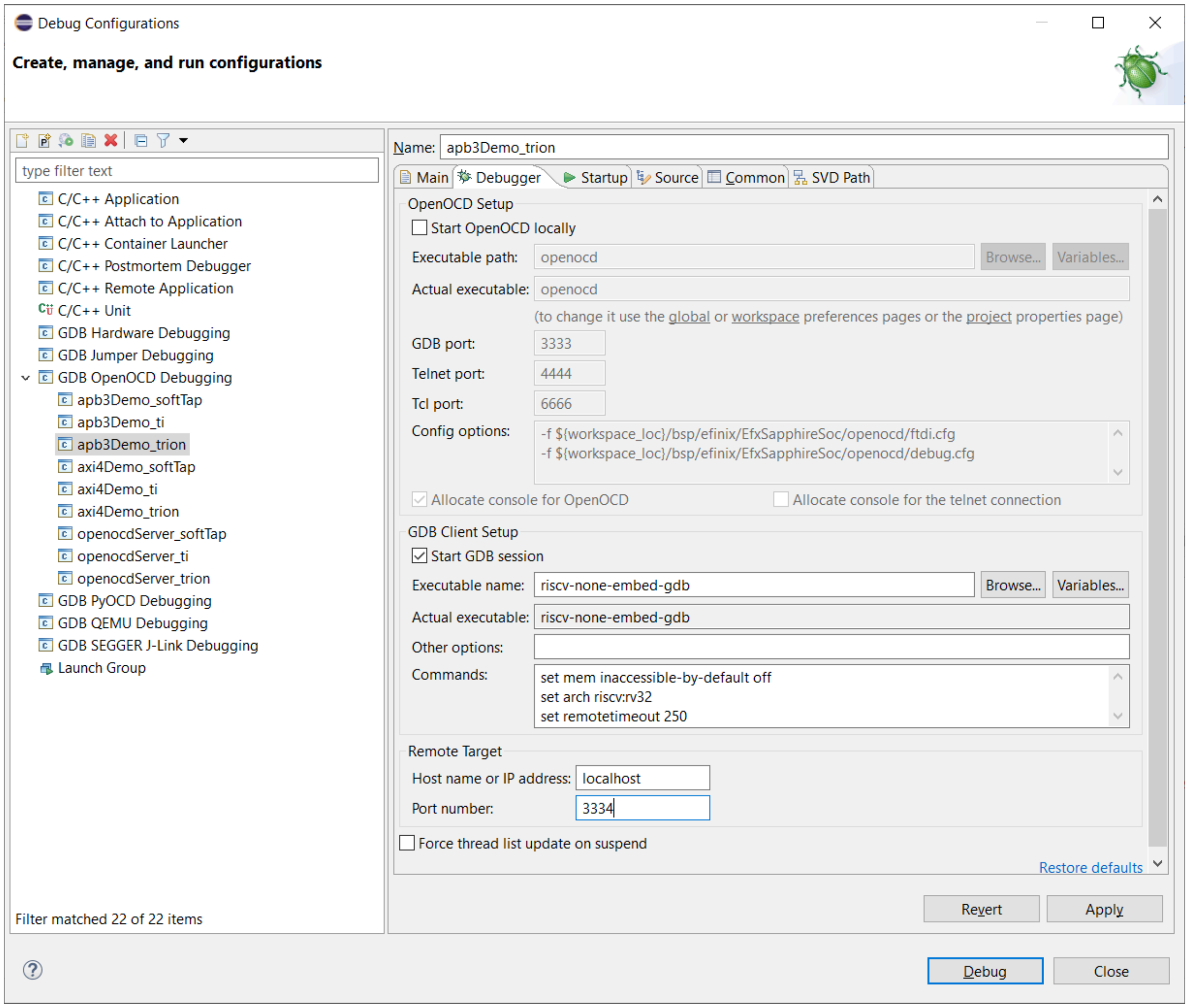
Learn more: For more information on debugging with Eclipse, refer to Running and debugging projects in the Eclipse documentation.

Debug - Multiple Cores

By default, the OpenOCD debugger always targets the first core, core `0`, when debugging. If your SoC has multiple cores, you can do standalone debugging with a core other than core `0`. This debug method uses the openocdServer debug launch scripts, which are available in the **software/standalone/openocdServer** directory. The general procedure is:

1. Create an SoC with more than 1 core.
2. Create a new project in Eclipse for your software code.
3. Create a new project for the openocdServer files.
4. Start the OpenOCD server.
   a. Right-click **openocdServer** > **openocdServer_<family>.launch**.
   b. Choose **Debug As** > > **openocdServer_<family>**.
5. Modify the debug configuration for your application to use the OpenOCD server:
   a. Right-click **<project folder>** > **Debug As** > **Debug Configurations**.
   b. Choose **GDB OpenOCD Debugging** > **<launch script>** (e.g., **axiDemo_trion**).
   c. Click the Debugger tab.
   d. Turn off **Start OpenOCD locally**.
   e. Under **Remote Target**, change the **Port number** for the core yiou are using (the default is 3333 for core `0`).
      - 3333: Core `0`
      - 3334: Core 1
      - 3335: Core 2
      - 3336: Core 3
6. Click **Debug**. Eclipse enters debug mode targeting the CPU that you specified with the port number.

Figure 54: Modify Debug Configuration for another Core


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Container Launcher
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- C/C++ Unit
- GDB Hardware Debugging
- GDB Jumper Debugging
- GDB OpenOCD Debugging
  - apb3Demo_softTap
  - apb3Demo_ti
  - apb3Demo_trion
  - axi4Demo_softTap
  - axi4Demo_ti
  - axi4Demo_trion
  - openocdServer_softTap
  - openocdServer_ti
  - openocdServer_trion
- GDB PyOCD Debugging
- GDB QEMU Debugging
- GDB SEGGER J-Link Debugging
- Launch Group

Filter matched 22 of 22 items

Name: apb3Demo_trion

Main | Debugger | Startup | Source | Common | SVD Path

OpenOCD Setup

- [ ] Start OpenOCD locally

Executable path: openocd Browse... Variables...

Actual executable: openocd

(to change it use the global or workspace preferences pages or the project properties page)

GDB port: 3333

Telnet port: 4444

Tcl port: 6666

Config options:
```
-f ${workspace_loc}/bsp/efinix/EfxSapphireSoc/openocd/ftdi.cfg
-f ${workspace_loc}/bsp/efinix/EfxSapphireSoc/openocd/debug.cfg
```

- [✓] Allocate console for OpenOCD
- [ ] Allocate console for the telnet connection

GDB Client Setup

- [✓] Start GDB session

Executable name: riscv-none-embed-gdb Browse... Variables...

Actual executable: riscv-none-embed-gdb

Other options:

Commands:
```
set mem inaccessible-by-default off
set arch riscv:rv32
set remotetimeout 250
```

Remote Target

Host name or IP address: localhost

Port number: 3334

- [ ] Force thread list update on suspend

Restore defaults

Revert Apply

Debug Close

Appendix: Re-Generate the Memory Initialization Files Manually

With the Efinity software v2022.1 and higher, you do not need to re-generate these files manually. These instructions are for reference if you are using an earlier software version.

To re-generate the memory initialization files manually using the **binGen.py** helper script. You find this script in the <*project*>**/embedded_sw/**<*SoC module*>**/tool** directory.

Windows:

Open a command prompt and type these commands:

```
${EFINITY_HOME}/bin/setup.bat
python3 binGen.py -b bootloader.bin -s <RAM size> -f <FPU>
```

Linux:

Open a terminal and type these commands:

```
source ${EFINITY_HOME}/bin/setup.sh
python3 binGen.py -b bootloader.bin -s <RAM size> -f <FPU>
```

where:
- <*RAM size*> is the on-chip RAM size you want to use.
- <*FPU*> indicates whether the floating-point unit is enabled for the SoC. 1: floating-point is enabled, 0: disabled.

This command generates the new memory initialization files. Copy these files into the same directory as your project **.xml** file, replacing the existing files.

Compile your design.

Appendix: Import the Debug Configuration

With the Efinity software v2022.1 and higher, you do not need to import the debug configuration. These instructions are for reference if you are using an earlier software version with legacy Eclipse IDE.

To simplify the debugging steps, the Sapphire SoC includes debug configurations that you import. There are several configuration files, depending on which board you use.

Table 44: Debug Configurations

Debug Configuration	Use for
default	Debugging software on Trion® development boards.
default_ti	Debugging software on Titanium development boards.
default_softTap	Debugging software on Trion or Titanium development boards with the soft JTAG TAP interface. For example, you would need to use the soft TAP if you want to use the OpenOCD debugger and the Efinity® Debugger at the same time. (See **Using a Soft JTAG Core for Example Designs** on page 112.)

To import a debug configuration and use it to launch a debug session:

1. Launch Eclipse by running the **run_eclipse.bat** file (Windows) or **run_eclipse.sh** (Linux).
2. Select a workspace (if you have not set one as a default).
3. Open the **axiDemo** project or select it under **C/C++ Projects**.
4. Right-click the **axiDemo** project name and choose **Import**.
5. In the Import dialog box, choose **Run/Debug > Launch Configurations**.
6. Click **Next**. The Import Launch Configurations dialog box opens.
7. Browse to the following directory and click **OK**:

Option	Description
Windows	**embedded_sw\<SoC module>\config**
Linux	**embedded_sw/<SoC module>/config_linux**

8. Check the box next to **config** (Windows) or **config_linux** (Linux).
9. Click **Finish**.
10. Right-click the **axiDemo** project name and choose **Debug As > Debug Configurations**.
11. Enter `axiDemo` in the **Project** box.
12. Enter `build\axiDemo.elf` in the **C/C++ Application** box.
13. *Windows only:* you need to change the path to the **cpu0.yaml** file:

    a. Click the **Debugger** tab.

    b. In the **Config options** box, change `${workspace_loc}` to the full path to the **<SoC module>** directory.

    Note: For the **cpu0.yaml** path, make sure to use \\ as the directory separator because the first slash escapes the second one. For example, use:

    c:\\Efinity\\2021.2\\project\\<project name>\\embedded_sw\\<SoC module>\\cpu0.yaml

14. Click **Debug**.

Note: If Eclipse prompts you to switch to the Debug Perspective, click **Switch**.

Appendix: Copy a User Binary to the Flash Device (2 Terminals)

To boot from a flash device, you need to copy the binary to the device. These instructions describe how to use two command prompts or terminals to flash the user binary file.

Note: If you want to store the binary in the same flash device that holds the FPGA bitstream, refer to Copy a User Binary to Flash (Efinity Programmer) on page 79 instead.

You use two command prompts or terminals:

- The first terminal opens an OpenOCD connection to the SoC.
- The second connects to the first terminal to write to the flash.

Important: If you are using the OpenOCD debugger in Eclipse, terminate any debug processes before attempting to flash the memory.

Set Up Terminal 1

To set up terminal 1, the flow varies on your IDE selection during the Sapphire SoC generation.

Efinity RISC-V Embedded Software IDE Selected

1. Open a Windows command prompt or Linux shell.
2. Change the directory to any of the example designs in your selected bsp location. The default location for **<efinity-riscv-ide installation path>** would be **C:\Efinity \efinity-riscv-ide-2022.2.3 for windows and home/<user>/efinity/efinity-riscv-ide-2022.2.3** for Linux.

   Note: The 2022.2.3 in the installation path may be different based on your IDE versions.

   Windows:

```
<efinity-risc-v-ide installation path>\openocd\bin\openocd.exe -f ..\..\..
\bsp\efinix\EfxSapphireSoc\openocd\ftdi.cfg
-c "set CPU0_YAML ..\..\..\cpu0.yaml"
-f ..\..\..\bsp\efinix\EfxSapphireSoc\openocd\flash.cfg
```

   Linux:

```
<efinity-risc-v-ide installation path>/openocd/bin/openocd -f ../../../bsp/
efinix/EfxSapphireSoc/openocd/ftdi.cfg
-c "set CPU0_YAML ../../../cpu0.yaml"
-f ../../../bsp/efinix/EfxSapphireSoc/openocd/flash.cfg
```

   The OpenOCD server connects and begins listening on port 4444.

Legacy Eclipse IDE Selected

1. Open a Windows command prompt or Linux shell.
2. Change to **SDK_Windows** or **SDK_Ubuntu**.
3. Execute the **setup.bat** (Windows) or **setup.sh** (Linux) script.
4. Change to the directory that has the **cpu0.yaml** file.
5. Type the following commands to set up the OpenOCD server:

   Windows:

```
openocd.exe -f bsp\efinix\EfxSapphireSoc\openocd\ftdi.cfg
  -c "set CPU0_YAML cpu0.yaml"
  -f bsp\efinix\EfxSapphireSoc\openocd\flash.cfg
```

   Linux:

```
openocd -f bsp/efinix/EfxSapphireSoc/openocd/ftdi.cfg
  -c "set CPU0_YAML cpu0.yaml"
  -f bsp/efinix/EfxSapphireSoc/openocd/flash.cfg
```

   The OpenOCD server connects and begins listening on port 4444.

Set Up Terminal 2

1. Open a second command prompt or shell.
2. Enable telnet if it is not turned on. Turn on telnet (Windows)
3. Open a telnet local host on port 4444 with the command `telnet localhost 4444`.
4. In the OpenOCD shell or command prompt, use the following command to flash the user binary file:

```
flash write_image erase unlock <path>/<filename>.bin 0x380000
```

   Where <*path*> is the full, absolute path to the **.bin** file.

   Note: For Windows, use \\ as the directory separators.

Close Terminals

When you finish:
- Type `exit` in terminal 2 to close the telnet session.
- Type Ctrl+C in terminal 1 to close the OpenOCD session.

Important: OpenOCD cannot be running in Efinity RISC-V Embedded Software IDE when you are using it in a terminal. If you try to run both at the same time, the application will crash or hang. Always close the terminals when you are done flashing the binary.

Reset the FPGA

Press the reset button on your development board:
- *Trion® T120 BGA324 Development Board*—SW2
- *Titanium Ti60 F225 Development Board*—SW3
- *Titanium Ti180 J484 Development Board*—SW1

Revision History

Table 45: Revision History

Date	Version	Description
June 2024	6.1	Added in D-Cache in table Sapphire SoC Tab Parameters. (DOC-1790) Added in option 32 for GPIO n Bit Width in table Sapphire GPIO Tab Parameters. Updated Simulate chapter. Updated Create a New Project and Import Sample Projects topic.
December 2023	6.0	Added in Lite option in Customizing the Sapphire SoC. (DOC-1533) Added semihostngDemo in Example Software. Added in Semihosting Printing section in Unified Printf. Added ENABLE_SEMIHOSTING_PRINT in Preprocessor Directives.
October 2023	5.4	Added steps and notes in Modify the Bootloader Software to Extend the External Memory Size, Modify the Bootloader Software without External Memory Enabled. Created new section Modify the Bootloader Software to Enable Multi-Data Lines. (DOC-1471) Added new example software inlineASMDemo. Added new topic Inline Assembly. Corrected wording of images with prints/messages in software examples. (DOC-1419)
August 2023	5.3	Updated and added new API Reference: (DOC-1379) –Control and Status Registers –GPIO API Calls –I^{2}C API Calls –Core Local Interrupt Timer API Calls –User Timer API Calls –PLIC API Calls –SPI API Calls –SPI Flash Memory API Calls –UART API Calls –RISC-V API Calls (new) Added footnote in Sapphire Debug Tab Parameters table. Added new section: Launching OpenOCD for Your Own Board and Updating OpenOCD Configuration for External FTDI Cable in Target Your Own Board topic. Added new topic: Updating Bootloader with Efinity BRAM Initial Content Updater under Modify the Bootloader topic. Added new section: i2cMasterDemo Design. Replaced new content in i2cSlaveDemo Design section. Replaced the section Warning when Debug with softTap with Unexpected CPUTAPID/JTAG Device ID.

Date	Version	Description
June 2023	5.2	Updated address from 0xF900_0000 to 0xF900_0C00 in line 4 and add a new line in Notes of Boot Sequence Case B. (DOC-1181) Added new paragraph after Default Address Map, Interrupt ID, and Cached Channels table in Address Map topic. (DOC-1199) Updated the following sections: (DOC-1253) –Customizing the Sapphire SoC –Example Design Implementation table –Launch the Debug Script –Debug Daisy Chain –Example Software –API Reference Added new topic: Other Customize Debugger Added in new sub-topics: –Debug - Single Core and Debug - SMP –dCacheFlushDemo, iCacheFlushDemo, and 12CEepromDemo –i2C_getSlaveStatus(), i2C_getSlaveOverride(), and i2C_masterRecoverBlocking() Change Ti180M484 to Ti180J484.
January 2023	5.1	Updated On-Chip RAM of 512KB in the figure titled Sapphire Memory Space. (DOC-

Date	Version	Description
December 2022	5.0	Moved topics Required Software for Eclipse, Launch Eclipse, Create and Build a Software Project, and Debug with the OpenOCD Debugger to Appendix. (DOC-981) Changed topic title, *Connect the FTDI Cable → Connect the FTDI Mini-Module* and FTDI cable → FTDI mini-module Added new main topics: –Launch Efinity RISC-V Embedded Software IDE –Create and Build a Software Project –Debug with the OpenOCD Debugger –Unified Printf Added new sub-topics: –Efinity RISC-V Embedded Software IDE –Install the Efinity RISC-V Embedded Software IDE –SoC Configuration Guideline –Sapphire SoC IDE Backward Compatibility –Launching the Efinity RISC-V Embedded Software IDE. –Optimization Settings –Import Sample Projects –Debug - Daisy Chain –Peripheral Register View –CSR Register View –FreeRTOS View –QEMU Emulator –Bsp_print –Bsp_printf –Bsp_printf_full –Preprocessor Directives –Warning when Debug with softTap Updated on sub-topics Open a Terminal, Connect the FTDI Cable, OpenOCD Error: timed out while waiting for target halted, Memory Test, Eclipse Fails to Launch with Exit Code 13, and API Reference. Updated Appendix: Copy a User Binary to the Flash Device (2 Terminals) and Import the Debug Configuration.
November 2022	4.2	Corrected boot sequence cases A and B. (DOC-932)
September 2022	4.1	Updates for the Ti180 M484 development board.
September 2022	4.0	Updated the instructions for debugging with OpenOCD. You now use launch scripts. Added information on the possible boot sequence scenarios. Enhanced the information on the address map. Added description for debugging with multiple cores. Added new SPI API functions. Added instructions on migrating from Ruby, Jade, and Opal to Sapphire. Updated IP Manager configuration options. Updated instructions on launching Eclipse. Updated Installing USB drivers topics.
June 2022	3.2	When finding the COM port in Windows, look for the first COM port listed under **Ports (COM & LPT)**. (DOC-811) The VexRiscv core used in the Sapphire SoC has six pipeline stages.

Date	Version	Description
March 2022	3.1	Fixed typo in Connect the FTDI Cable topic. (DOC-731)
December 2021	3.0	Updated the SDK version numbers. Updated the IP Manager Configuration Wizard description for new configuration options. Added instructions for using the Ti60 F225 Development Board and example design. Updated instructions for Eclipse global environment variables. Explained new Efinity Programmer feature for programming a flash device with a combined user bitstream and application binary. Updated register map. Updated the API Reference for new driver support.
October 2021	2.1	Corrected incomplete instructions for copying a user binary to flash. (DOC-576)
October 2021	2.0	IP Manager options changed for the updated Sapphire wizard. (DOC-544) Updated the address map. (DOC-544) Updated the example design description for the new features in the design. (DOC-544) New simulation instructions. (DOC-544) New instructions for changing the bootloader RAM size. (DOC-544) Changed the EfxApb3Example, EfxAxi4Example, and userInterruptDemo example descriptions. (DOC-544) Changed the TX pin number for the instructions on setting up a USB-to-UART module. (DOC-544) When using the Soft Debug Tap option, the IP Manager connects the pins for you. (DOC-544) Described the pins needed to connect an FTDI cable to the Trion® T120 BGA324 Development Board when using the Soft Debug Tap option. (DOC-544)
August 2021	1.1	Corrected typo in example design name in topics describing Eclipse and OpenOCD (EfxAxi4Example instead of EfxAxiExample). (DOC-517)
July 2021	1.0	Initial release.