易灵思

Sapphire High-Performance RISC-V SoC Hardware and Software User Guide

UG-SAPPHIREHPB-v2.0
December 2024
www.elitestek.com

Copyright © 2024. All rights reserved. Elitestek and the Elitestek logo are trademarks of Elitestek Co., Ltd.. All other trademarks and service marks are the property of their respective owners. All specifications subject to change without notice.

Contents

Introduction

Elitestek provides a hardened RISC-V SoC, called Sapphire High-Performance SoC, that you can implement in tandem with the soft logic block on the TJ-Series TJ375 FPGA.

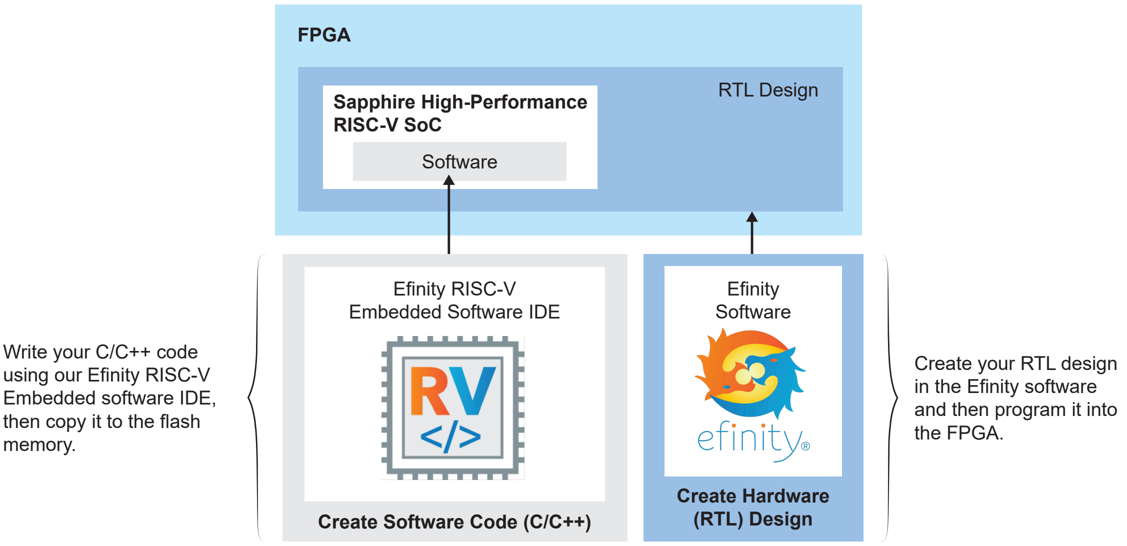
This hardened SoC features a 32-bit quad-core RISC-V processor based on the RISCV32I ISA[1] with M, A, C, F, and D extensions. It operates with six pipeline stages: fetch, injector, decode, execute, memory, and writeback.

Each CPU core includes a dedicated FPU and supports custom instructions. The processor follows the standard RISC-V debug specification and providing 8 hardware breakpoints. Additionally, it supports machine and supervisor privileged modes, along with Linux MMU SV32 page-based virtual memory.

This user guide describes how to:

- Build RTL designs using the Sapphire High-Performance RISC-V SoC using an example design targeting TJ-Series TJ375 N529 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.

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



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

[1] ISA: Instruction Set Architecture

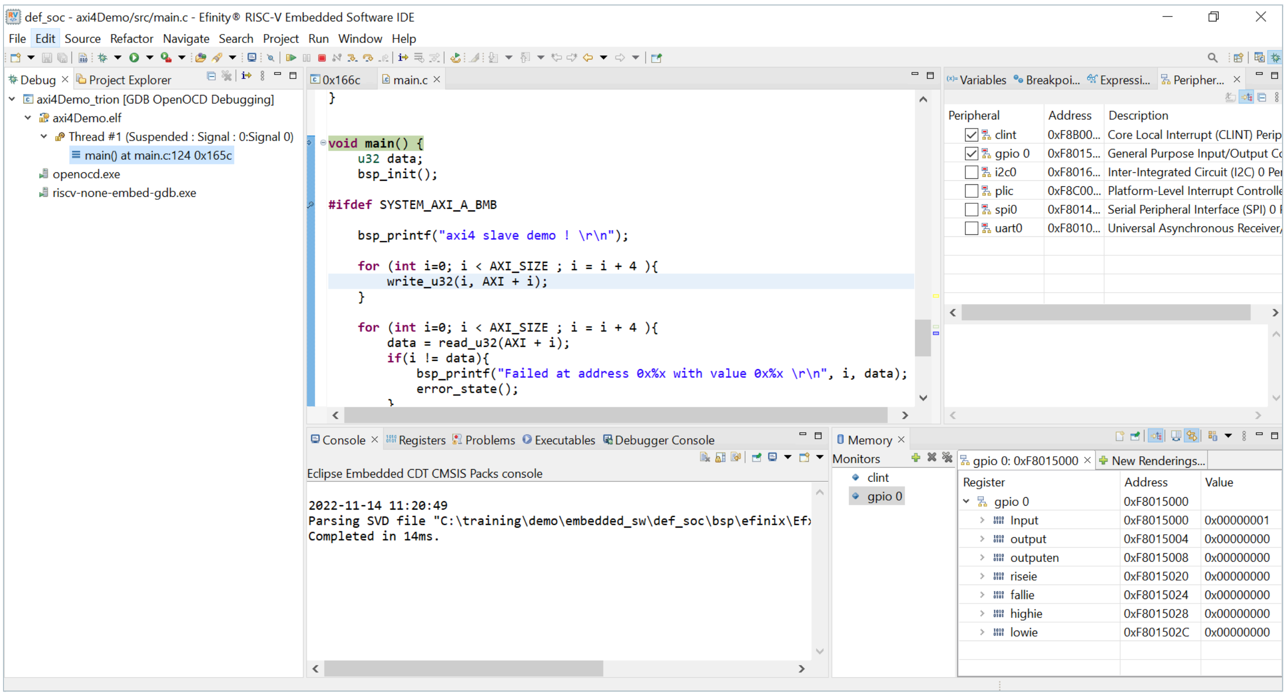
Efinity® RISC-V Embedded Software IDE

The Efinity® RISC-V Embedded Software IDE is an Eclipse-based Integrated Development Environment (IDE) powered by Ashling's ***RiscFree***™ IDE for Sapphire High-Performance 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



Required Software

To write software for the Sapphire High-Performance 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

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

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

Required Hardware

- TJ-Series TJ375 N529 Development Board
- 12 V power cable
- USB C-cable
- Computer or laptop
- FAT32 formatted SD card
- Cat 5e Ethernet cable and above

Comparison with Sapphire SoC

While the Sapphire High-Performance SoC architecture shares similarities with the Sapphire SoC, there are notable differences to consider:

Sapphire SoC

- User peripherals connected via internal bus.
- No data coherency between CPU and DMA via AXI Master port.
- Absence of a branch predictor.
- Uses a shared FPU configuration for multi-core setups.
- Supports up to 4 configurable hardware breakpoints.

Sapphire High-Performance SoC

- User peripherals are segmented into separate modules, with `EfxSapphireHpSoc_slb` connected via AXI slave interface. Note that the base address and AXI slave interface are dedicated to this module. If additional modules require AXI slave access, an AXI interconnect IP can facilitate connections to both `EfxSapphireHpSoc_slb` and your module.
- Ensure data coherency between CPU and DMA via AXI Master port, potentially eliminating the need for data cache flushing.
- Features an integrated static branch predictor.
- Dedicated FPU per core.
- Supports 8 hardware breakpoints for debug module.

Performance

The following table shows the overall performance of Sapphire High-Performance SoC.

Table 1: Key Performance of Sapphire High-Performance SoC

Test/Benchmark	Result	
Dhrystone Baremetal	1.2375 DMIPS/MHz	
Coremark Baremetal	2.345 Coremark/MHz	
Coremark Linux	2.222 Coremark/MHz	
Coremark Pro Linux	Multi Core	581.67
	Single Core	167.46
	Scaling	3.47

Configuration:

- System clock: 1.0 GHz
- Memory clock: 250 MHz
- DDR clock: 800 MHz (3.2 Gbps) @ x32 data lanes

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:

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

Chapter 2

IP Manager

Contents:

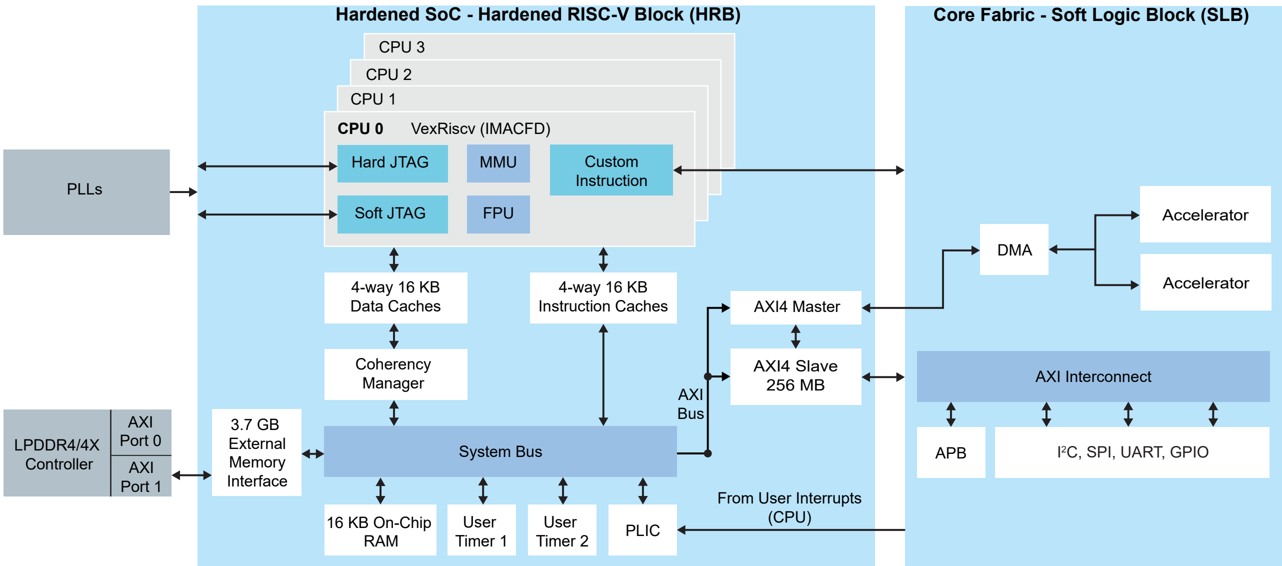
The Efinity® IP Manager is an interactive wizard that helps you customize and generate Elitestek® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Elitestek 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 High-Performance SoC with the IP Manager

Figure 3: Overall Block Diagram of Sapphire High-Performance SoC



The Sapphire High-Performance SoC consists of two (2) parts, the hardened RISC-V block (HRB) and the soft logic block (SLB). The HRB includes a quad-core CPU, caches, memory management, debug module, on-chip RAM, and data traffic management. In contrast, the SLB is formed by soft logic to exercise I/O control, custom ALU, and DMA. In relation, the IP Manager helps to configure the hardened blocks and instantiate the common-use controllers like SPI, I2C, GPIO, and UART. Additionally, the IP manager assists by configuring the required blocks like PLLs and LPDDR4 controllers.

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: The Sapphire High-Performance SoC soft logic block module name is fixed to **EfxSapphireHpSoc_slb**.

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 Elitestek® 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.

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/efx_hard_soc** directory.

- **bsp**—Board specific package.
- **software**—Software examples, includes FreeRTOS and baremetal demos.

Instantiating the Hardened RISC-V SoC

The IP manager helps to instantiate the hardened RISC-V block from the Efinity's Interface Designer which includes:

- Assigning top-level signal name to the block.
- Instantiating dedicated PLL to the hardened RISC-V block.
- Instantiating the required GPIO block.
- Assigning pre-defined pins to GPIO block.

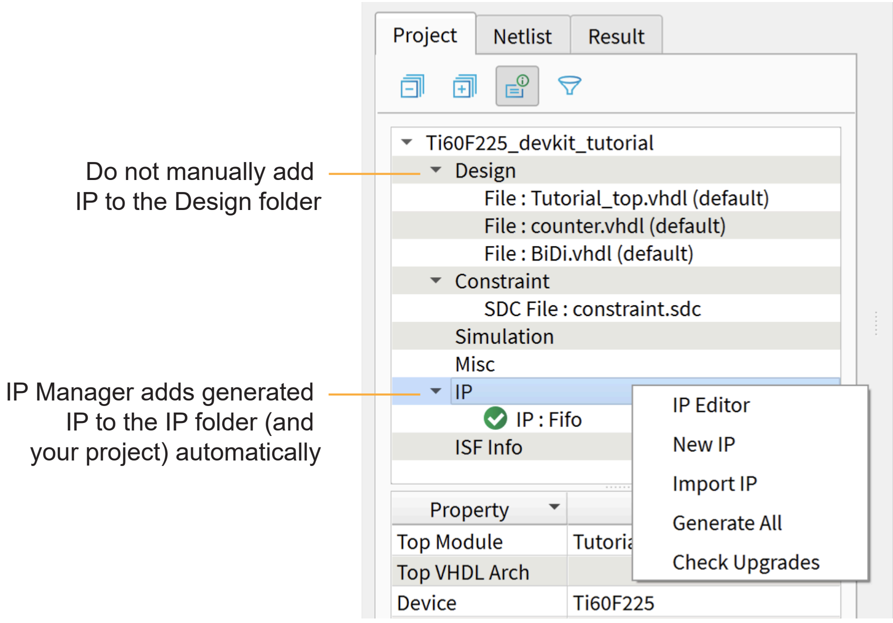
Instantiating the SoC Soft Logic Block

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.
- **EfxSapphireHpSoc_wrapper.v** is the wrapper file for soft logic block design.

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.



Customizing the Sapphire High-Performance SoC

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

Table 2: Sapphire High-Performance Hardened RISC-V Block Tab Parameters

Parameter	Options	Description
JTAG Debug Interface	FPGA User Tap, JTAG with GPIO	Choose whether to include a soft debug TAP for debugging. FPGA User Tap: The SoC uses the JTAG User Tap interface block to communicate with the OpenOCD debugger. JTAG with GPIO: The SoC has a soft JTAG interface to communicate with the OpenOCD debugger.
FPGA User Tap Port	JTAG_USER1, JTAG_USER2, JTAG_USER3, JTAG_USER4	Choose the tap port to target with the OpenOCD debugger. This option only applies when using the JTAG user tap interface block to communicate with the OpenOCD debugger.
AXI4 Master Interface	On, off	On: Instantiate the interface. Off: Do not use the interface.
AXI4 Slave Interface	On, off	On: Instantiate the interface. Off: Do not use the interface. This interface is forcibly enabled for peripheral interfacing.
CPU *n* Custom Instruction Interface	On, off	On: Instantiate the interface. Off: Do not use the interface.
User Interrupt Ports	0 - 24	0: Do not use interrupt port. 1 - 24: The number of interrupt port that turns on.
OCR Application	On, off	On: Overwrite the default SPI flash bootloader with the user application. Off: Initialize SoC without user application.
User Application	–	Enter the path to your target user application. The file must be in `.hex` format.

Table 3: Sapphire High-Performance Soft Logic Block Tab Parameters

Parameter	Options	Description
Peripheral Interconnect	On, off	On: Instantiate an interconnect with peripherals attached to it. Off: Do not use the interconnect generated by the IP Manager.
Pin Resource Assignment	On, off	On: Update project peri.xml to instantiate required GPIO blocks for enabled peripherals. Off: Do not update project peri.xml
Pin Assignment	On, off	On: Update project peri.xml to assign pre-defined pins to GPIO blocks. Off: Do not update project peri.xml.
Required SoC Interrupt Ports	–	Show required interrupts for enabled peripherals.
Uart Controller *n*	On, off	On: Instantiate the controller. Off: Do not use the controller.
SPI Controller *n*	On, off	On: Instantiate the controller. Off: Do not use the controller.
I2C Controller *n*	On, off	On: Instantiate the controller. Off: Do not use the controller.
GPIO Controller *n*	On, off	On: Instantiate the controller. Off: Do not use the controller.
Specify GPIO *n* pin width	4, 8, 16, 24, 32	Specify the number of pins to be enabled for the GPIO controller.
Watchdog Timer	On, off	On: Instantiate the watchdog timer. Off: Do not use the watchdog timer.
APB3 interface *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
Specify APB3 *n* size	4 KB, 16 KB, 64 KB, 256 KB, 1 MB	Specify the size of the APB interface.
SD Host Controller	On, off	Instantiate SD host controller ip and integrate into soc wrapper file the interface. To be supported in the upcoming 2024.1 patch release.
Triple-speed Ethernet MAC	On, off	Instantiate ethernet controller ip and integrate into soc wrapper file. To be supported in the upcoming 2024.1 patch release.

Table 4: Sapphire High-Performance PLL Configuration Tab Parameters

Parameter	Options	Description
System Clock PLL		
Pin Assignment	On, off	On: Update project peri.xml to include this PLL. Off: Do not update project peri.xml
Instance Name	Fixed string	The PLL instance name will be configured later in the Interface Designer.
PLL Resource	PLL_BL0, PLL_BL1, PLL_BL2	Choose which PLL resource you want to utilize in Interface Designer.
PLL External Clock Source	Clock 0, Clock 1	Specify which external clock source as reference clock to PLL.
Reference Clock Frequency	Input value in MHz	Specify reference clock frequency.
System Clock Frequency	250 - 1000 MHz	Specify the system clock frequency that drives most of the logic of the hardened RISC-V block including CPU, FPU, MMU, caches, on-chip RAM, etc.
Memory Clock Frequency	25 - 250 MHz	Specify the memory clock frequency that drives the AXI traffic to external memory.
DDR Clock Frequency	200 - 900 MHz	Specify the DDR clock frequency that is input to the DDR controller.
Peripheral Clock PLL		
Pin Assignment	On, off	On: Update project peri.xml to include this PLL. Off: Do not update project peri.xml
Instance Name	Fixed string	The PLL instance name that will be configured later in the Interface Designer.
PLL Resource	PLL_BLn, PLL_BRn PLL_TLn PLL_TRn,	Choose which PLL resource you want to utilize in Interface Designer. The PLL resource cannot be the same as system clock PLL.
PLL External Clock Source	Clock 0 Clock 1	Specify which external clock source as reference clock to PLL.
Reference Clock Frequency	Input value in MHz	Specify reference clock frequency.
Peripheral Clock	25 - 250 MHz	Specify peripheral clock frequency that drives soft logic block logic.
AXI4 Master Clock	25 - 250 MHz	Specify AXI4 master clock.
Custom Instruction Clock	25 - 250 MHz	Specify custom instruction clock frequency.

Table 5: Sapphire High-Performance LPDDR4 Configuration Tab Parameters

Parameter	Options	Description
Device Setting		
LPDDR4 Controller Assignment	On, off	On: Update project peri.xml to include this configuration. Off: Do not update project peri.xml
Instance Name	Fixed string	The DDR instance name will be configured later in the Interface Designer.
Memory Data Width	16, 32	The DDR device data width.
Memory Density	2 GB, 3 GB, 4 GB, 6 GB, 8 GB, 12 GB, 16 GB	The DDR device memory density.
Memory Type	LPDDR4, LPDDR4x	The DDR device memory type.
Memory Physical Rank	1, 2	The DDR device memory physical rank.

Table 6: Sapphire High-Performance Embedded Software Configuration

Parameter	Options	Description
FTDI Type	Single Channel Dual Channel Quad Channel	Specify the number of channels available for the FTDI device use.
FTDI Debug Channel	Channel 0,Channel 1, Channel 2,Channel 3	Specify which channel the JTAG connected.
Application Size	124 KB, 252 KB, 324 KB, 508 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, Custom	Specify the size allocated for application in linker scripts.
Application Size (KB)	–	Specify the custom size allocated for application in linker scripts.
Stack Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, Custom	Specify the size allocated for application stack in linker scripts.
Stack Size (KB)	–	Specify the custom size allocated for application stack in linker scripts.

Modify the Bootloader

When you generate the Sapphire High-Performance SoC, the IP Manager does not include any pre-built firmware to target the on-chip RAM size you selected. You can compile SPI flash bootloader software codes in the **embedded_sw/efx_hard_soc/software/standalone/bootloader**

Learn more: You need to install Efinity RISC-V Embedded Software IDE to compile the bootloader or other software.

Note: By default, the bootloader uses only a single data line SPI. To use dual or quad data line SPI, refer to **Modify the Bootloader Software to Enable Multi-Data Lines** on page 18.

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/efxh_hard_soc/software/standalone/bootloader** directory.
2. Change the `#define USER_SOFTWARE_SIZE` parameter for the new on-chip RAM size and save.
3. 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 324 KB.

1. In the Sapphire High-Performance IP wizard, go to the **HRB** tab.
2. Turn on the **OCR Application** option.
3. Click the **Browse** button to select the new **bootloader.hex** you created in the previous set of steps.
4. Generate the SoC.

Modify the Bootloader Software to Enable Multi-Data Lines

Before utilizing the multi-data lines SPI in your bootloader, ensure your board's flash device supports Dual or Quad I/O modes.

In the example design, data ports 0 and 1 are currently connected. If you intend to use the Quad SPI for data transfer, you must establish connections for data ports 2 and 3.

In the **bootloaderConfig.h** file, you can define your preferred SPI lane configuration by selecting from the following data line modes:

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

```
#define SINGLE_SPI 1 //define DUAL_SPI for dual data SPI or QUAD_SPI for quad data SPI

void bsp Main() {
#ifndef SIM
        spiFlash_init(SPI, SPI_CS);
        spiFlash_wake(SPI, SPI_CS);
        spiFlash_exit4ByteAddr(SPI, SPI_CS);
#ifdef SINGLE_SPI
        spiFlash_f2m(SPI, SPI_CS, USER_SOFTWARE_FLASH, USER_SOFTWARE_MEMORY,
        USER_SOFTWARE_SIZE);
#elif DUAL_SPI
        spiFlash_f2m_dual(SPI, SPI_CS, USER_SOFTWARE_FLASH, USER_SOFTWARE_MEMORY,
        USER_SOFTWARE_SIZE); //dual data line half duplex
#elif QUAD_SPI
        spiFlash_f2m_quad(SPI, SPI_CS, USER_SOFTWARE_FLASH, USER_SOFTWARE_MEMORY,
        USER_SOFTWARE_SIZE); //quad data line full duplex
#else
        #error "You must either define SINGLE_SPI to use single data line SPI, DUAL_SPI to use
        dual data line SPI or QUAD_SPI to use quad data line SPI."
#endif
#endif
        void (*userMain)() = (void (*)())USER_SOFTWARE_MEMORY;
    #ifdef SMP
        smp_unlock(userMain);
    #endif
        userMain();
}
```

Chapter 3

Recommended Design Practice

Instantiate the Sapphire High-Performance SoC and Soft Logic Block Using the IP Manager

Before you integrate your design with the Sapphire High-Performance SoC, you need to generate the RISC-V block and soft logic block together, with the PLL and GPIO resource allocation, and pin assignment. This first step allows you to bring up a working design and having a reference for your next-step debugging. You are free to change the configuration, pins, or block resources later with the interface designer as you can bypass the interface designer configuration update shall you require to re-generate the soft logic block design files with the custom interface designer configuration.

PLL Utilization

The Sapphire High-Performance SoC requires at least 3 crucial clocks:

- *System clock*—Drives most of the logic within the SoC including CPUs, FPU, MMU, caches, on-chip RAM, etc.
- *Memory clock*—Drives the AXI path to communicate with the DDR controller.
- *DDR controller clock*—Input clock for the DDR controller.

Note: Elitestek recommends you use the same PLL to drive the clocks to save the utilization of PLL. Only three PLLs are supported to deliver the clock path to RISC-V block. If you have doubts about the PLL assignment, you may generate the working design with IP Manager.

The following table shows the assignment of the clock.

Table 7: Clock Assignment

PLL Resource	Output Clock	Functionality
PLL_BL0 and PLL_BL2	Clock 1	System clock
	Clock 2	Memory clock
	Clock 3	DDR clock
PLL_BL1	Clock 1	Memory clock
	Clock 2	System clock
	Clock 3	DDR clock

Soft Logic Block Design Files

The IP Manager generates the design files for soft logic block when you generate the Sapphire High-Performance SoC block. You can see the output files, **EfxSapphireHpSoc_slb.v** and **EfxSapphireHpSoc_wrapper.v**.

The **EfxSapphireHpSoc_slb.v** is the design file that does the following:

- Handles master reset control
- LPDDR4 controller reset and calibration control
- Connects with the selected peripherals in the IP Manager
- Establish a connection between peripherals and user interrupt ports
- Establish a connection between the SoC debug module and the FPGA user tap or GPIO

The **EfxSapphireHpSoc_wrapper.v** is the example top file for you to refer to and is optional to be included in the project for compilation. You can open and copy the **EfxSapphireHpSoc_wrapper.v**file to your own top file in directory **ip/EfxSapphireHpSoc_slb/**. Including the wrapper file in the project compilation list is not recommended. The file can revert to the default design whenever you regenerate the block using IP Manager.

Handling SoC Interfaces

The Sapphire High-Performance SoC provides the following:

- Custom instruction interface
- AXI master and slave interface
- 24 user interrupt ports to interact with soft logic from FPGA core fabric

You can use a custom instruction interface to deliver the custom ALU for CPU, an AXI master interface for direct memory access, an AXI slave for peripheral communication, and interrupt ports for triggering to access priority routine. However, this can block the SoC from out of reset state if you enable the interfaces but left them unconnected in a design. Hence, you must connect the pins below to a known state even if your design is not ready in the first place.

No further action is required if you disable the interface in the IP Manager.

Table 8: Clock Assignment

Interface	Pin	State
Custom Instruction	cpuX_customInstruction_cmd_ready	High
AXI Master	io_ddrMasters_0_b_ready	High
	io_ddrMasters_0_r_ready	High

Chapter 4

Example Design

Contents:

Before working with software code, Elitestek recommends that you program your board with an RTL design that instantiates the Sapphire High-Performance 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 TJ-Series development boards:

- *TJ-Series TJ375 N529 Development Board*—The RTL design files are in the **Ti375C529_devkit** directory.

When you generate the IP core, the IP Manager creates the example design (PLL settings, SDC timing constraints, and I/O assignments).

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

- For the TJ-Series TJ375 N529 Development Board, the design uses the board's LPDDR4/LPDDR4x DRAM module.

The Sapphire High-Performance SoC is configured for:

- 1000 MHz system clock frequency
- 250 MHz memory clock frequency
- 800 MHz DDR controller clock frequency
- 200 MHz peripheral clock frequency
- 250 MHz AXI master clock frequency
- 125 MHz custom instruction clock frequency
- Used FPGA user tap 1 for debugging
- Custom instruction for each CPU is enabled
- UART 0 is enabled
- SPI 0 is enabled
- I2C 0 is enabled
- GPIO 0 is enabled
- AXI4 Slave is enabled
- AXI Master 0 is enabled
- 8 User interrupts are enabled

Additional soft IPs like AXI interconnect, SD host controller and ethernet controller are included in the example design.

Figure 4: Example Design Block Diagram

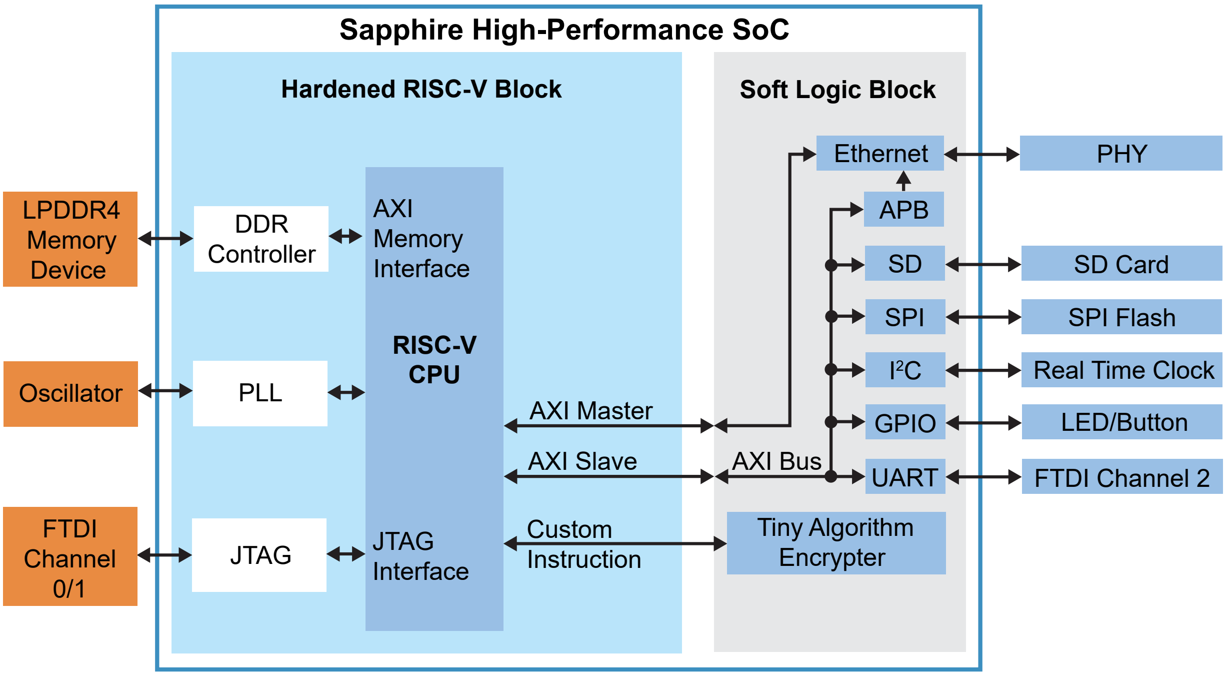


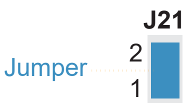
Table 9: Example Design Implementation

FPGA	Logic + Adders	Flipflops	Multipliers or DSP Blocks	Memory Blocks	f_{MAX} (MHz)	Language	Efinity Version
TJ375 BGA529 C4	14,987	11,858	0	76	233	Verilog HDL	2024.1

Enable the LPDDR4x Memory (TJ375 N529 Board)

For the TJ-Series TJ375 N529 Development Board, by default, the SoC design uses LPDDR4x settings to drive the external memory. To enable LPDDR4 setting, change the jumpers on J21 to connect pins 1 and 2 to provide 1.2 V to VDDQ and VDDQ_PHY.

Figure 5: Connect Pins 1 and 2 on J21



Installing USB Drivers

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

Elitestek development board has FT4232H FTDI chip to communicate with the USB port and other interfaces such as SPI, JTAG, or UART. Refer to the Elitestek development kit user guide for details on installing drivers for the development board.

Note: If you are using more than one Elitestek 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
TJ-Series Ti375 C529 Development Board	Install drivers for interfaces (0 and 1)

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
> sudo udevadm trigger
```

Note: If your board was connected to your computer before you executed these commands, you need to disconnect it, then 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 Elitestek development board. You need to compile example design to get the bitstream file.

Table 10: Available Example Designs

Board	Location
TJ-Series Ti375 C529 Development Board	Ti375C529_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 5

Launch Efinity RISC-V Embedded Software IDE

Contents:

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-2023.2). 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:

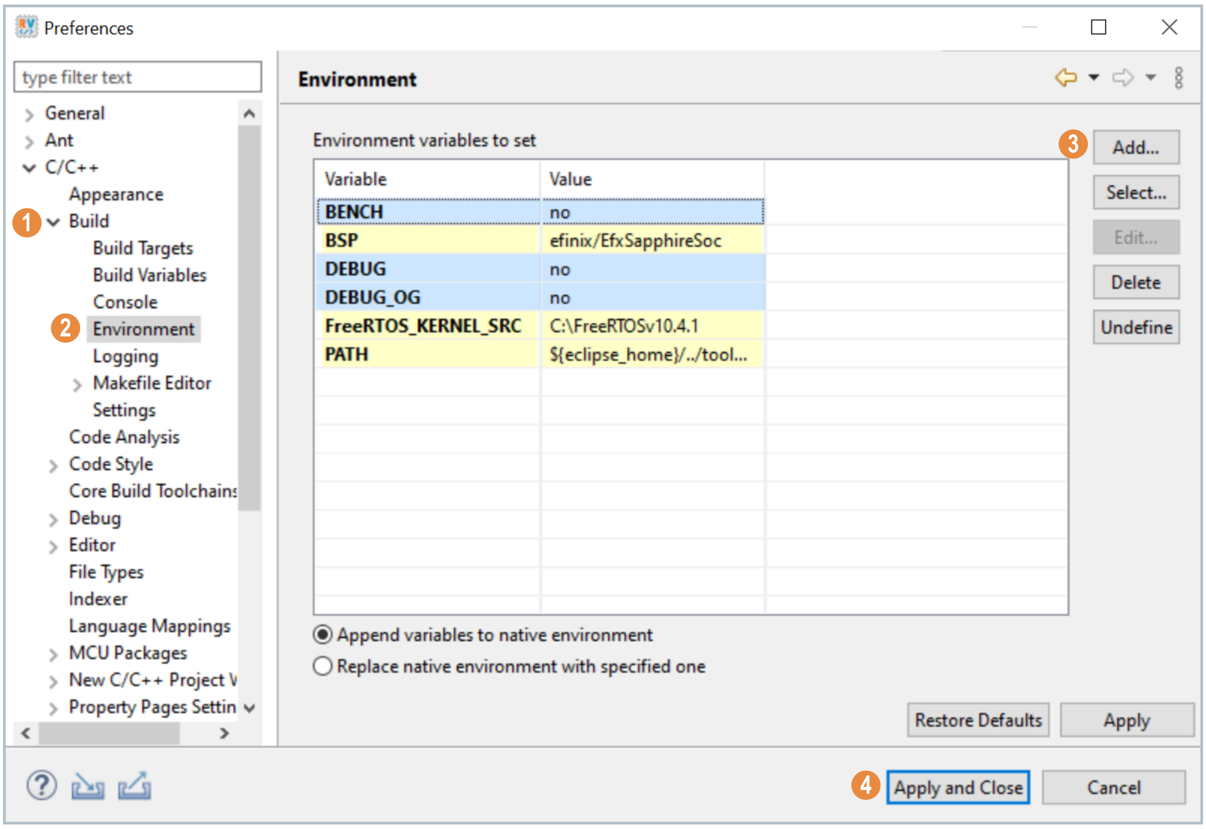
- You can choose any location for your workspace. If you have selected a folder that does not exist, the IDE automatically creates a folder for you.
- The Sapphire High-Performance SoC is supported by Efinity RISC-V Embedded Software IDE v2023.2.5 and above only.

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.



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 11: 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.

Create a New Project

In the **Project Explorer**:

1. Click **Create a Project** to open the new project wizard.
2. Select the **Elitestek 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 High-Performance 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 6: Create New Elitestek Makefile Project Wizard for Standalone Project

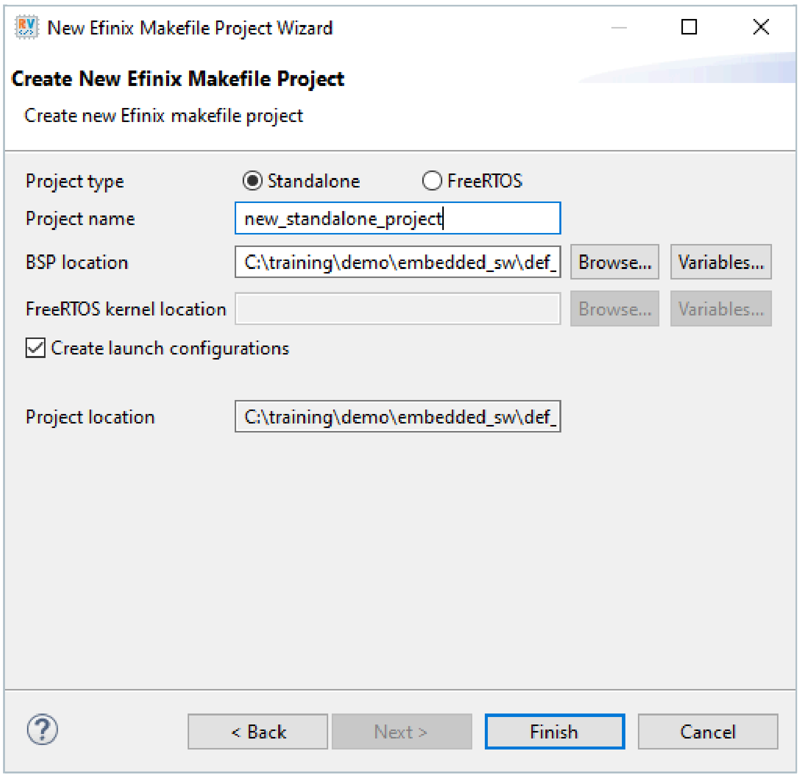
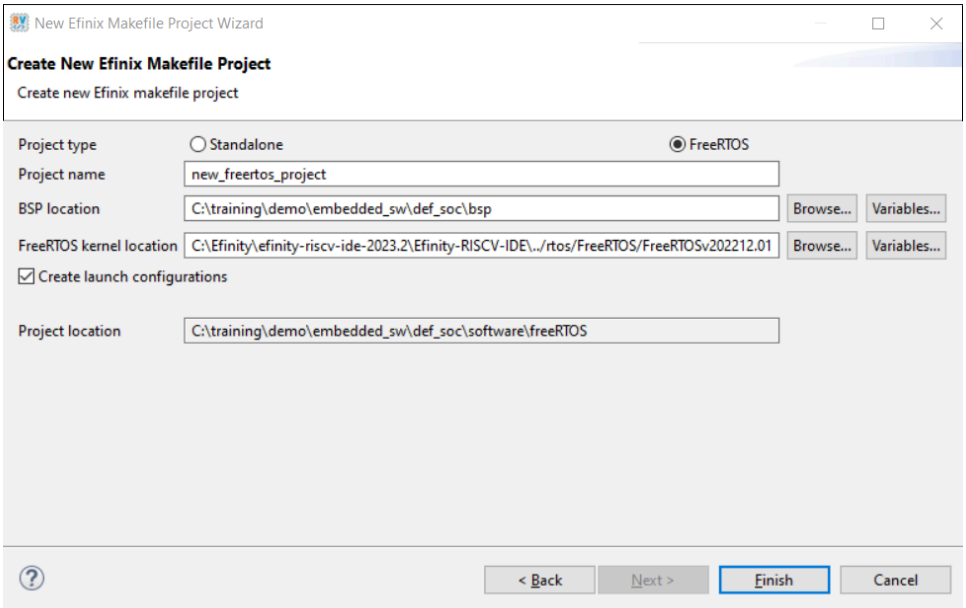
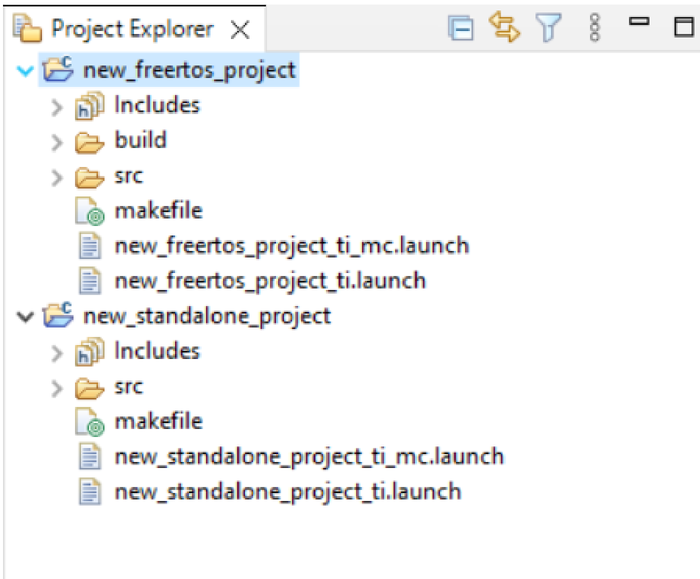


Figure 7: Create New Elitestek Makefile Project Wizard for FreeRTOS Project



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. Both **ti.launch** and **ti_mc.launch** is generated if hard TAP is selected for the Sapphire High-Performance SoC. While **softTap.launch** and **softTap_mc.launch** is generated if soft Tap is selected.

Figure 8: Project Explorer Pane Showing All Projects Created



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 9: Output Console Showing the Newly Generated Standalone Project Built Successfully

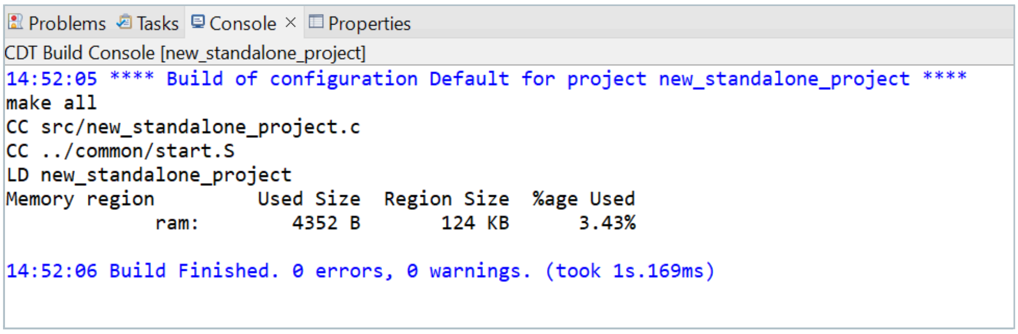
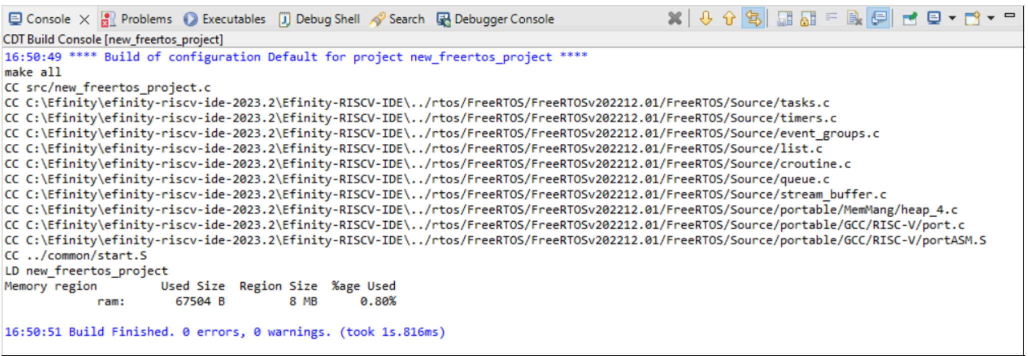


Figure 10: Output Console Showing the Newly Generated FreeRTOS Project Built Successfully

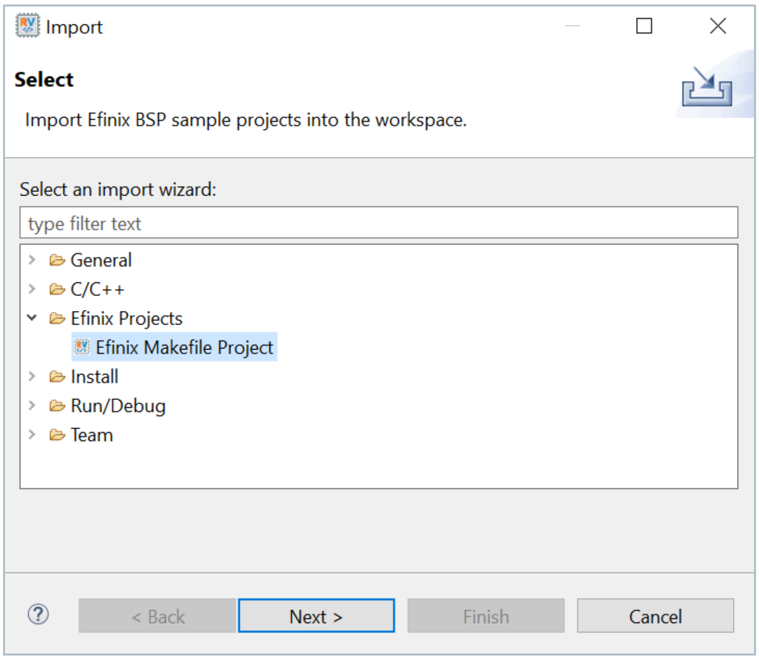


Import Sample Projects

Elitestek provides sample projects to help you get started with Sapphire High-Performance 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 **Elitestek Makefile Project** in **Elitestek Projects** and click **Next**.

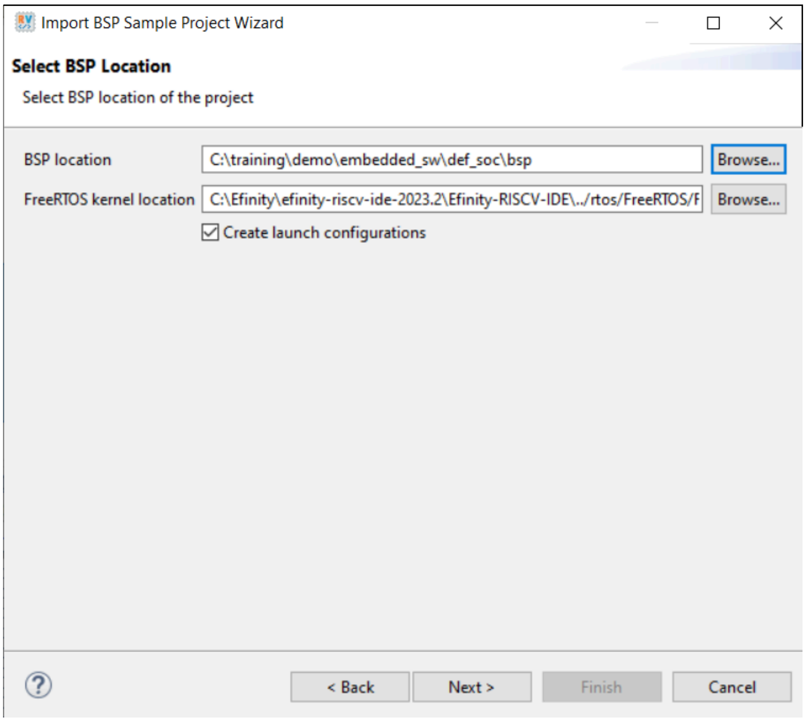
Figure 11: Import Wizard



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 12: Import BSP Sample Project Wizard

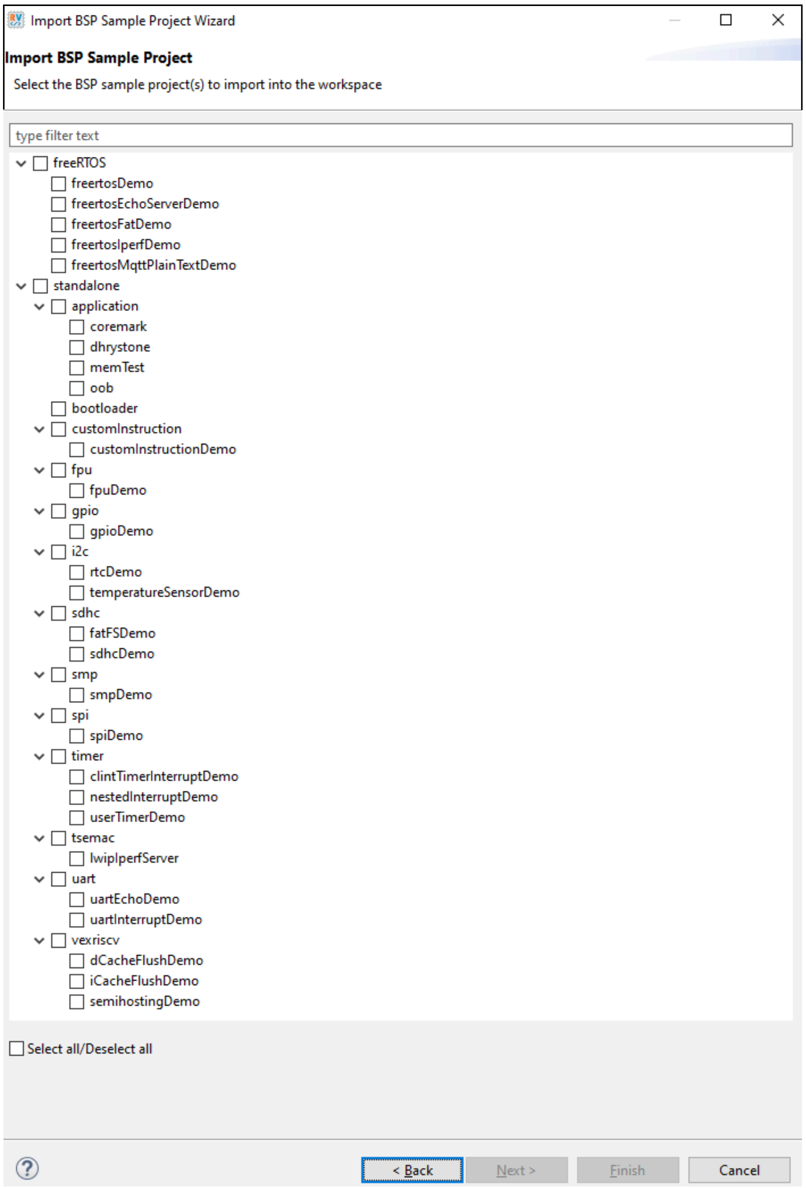


Note: FreeRTOS projects is filtered if the FreeRTOS kernel location is not defined.

The next wizard page shows the **Import BSP Sample Project Wizard**, all sample projects are located in the **embedded_sw/efx_hard_soc/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 13: Import BSP Sample Project Wizard – List of Projects

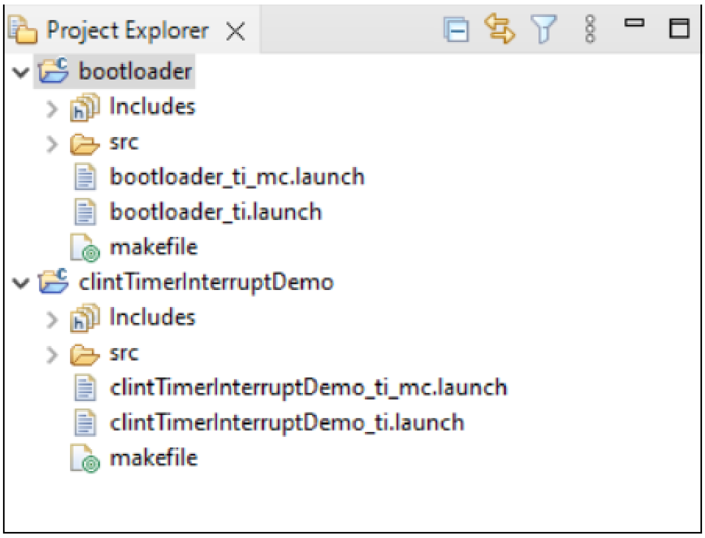


Note:

- 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.

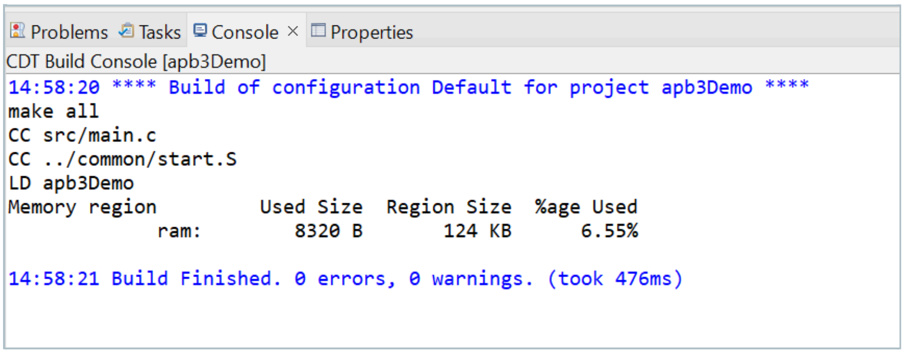
The selected sample projects are imported into the listed workspace in the **Project Explorer** pane.

Figure 14: 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 15: Output Console showing the apb3Demo Standalone Project Built Successfully



Build

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

Elitestek 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 **gpioDemo** 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/efx_hard_soc/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 12: Debug Configurations

Launch Script	Description
<app>_ti.launch	Debugging software on TJ-Series development boards.
<app>_softTap.launch	Debugging software on Trion or TJ-Series 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 95.)
<app>_ti_mc.launch	Debugging software on TJ-Series development boards with multiple cores.
<app>_softTap_mc.launch	Debugging software on Trion or TJ-Series 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 95.)

To debug the gpioDemo project:

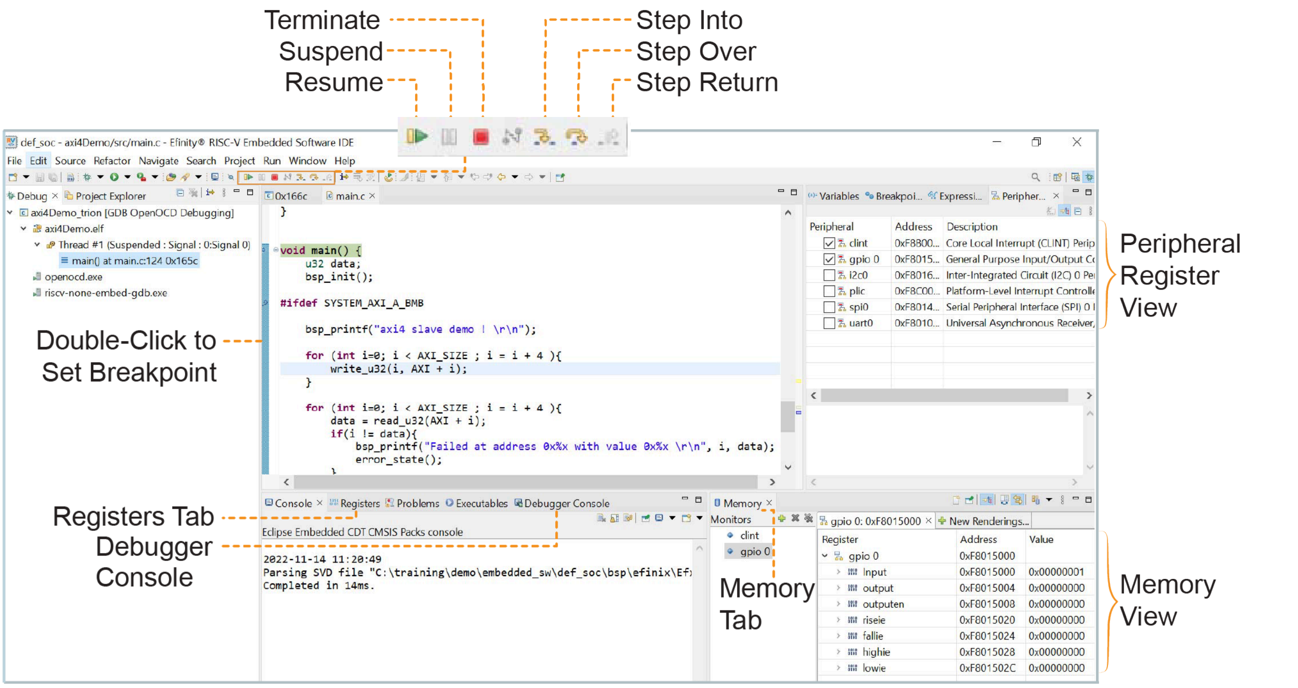
1. Right-click **gpioDemo** > **gpioDemo_<family>.launch**.
2. Choose **Debug As** > **gpioDemo** > **gpioDemo_<family>**. Efinity RISC-V Embedded Software IDE launches the OpenOCD debugger for the project.
3. Click **Debug**.
4. **Confirm Perspective Switch** window would prompt out. Click **Switch** 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 16: Perform Debugging



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

Debug - Multiple Cores

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**

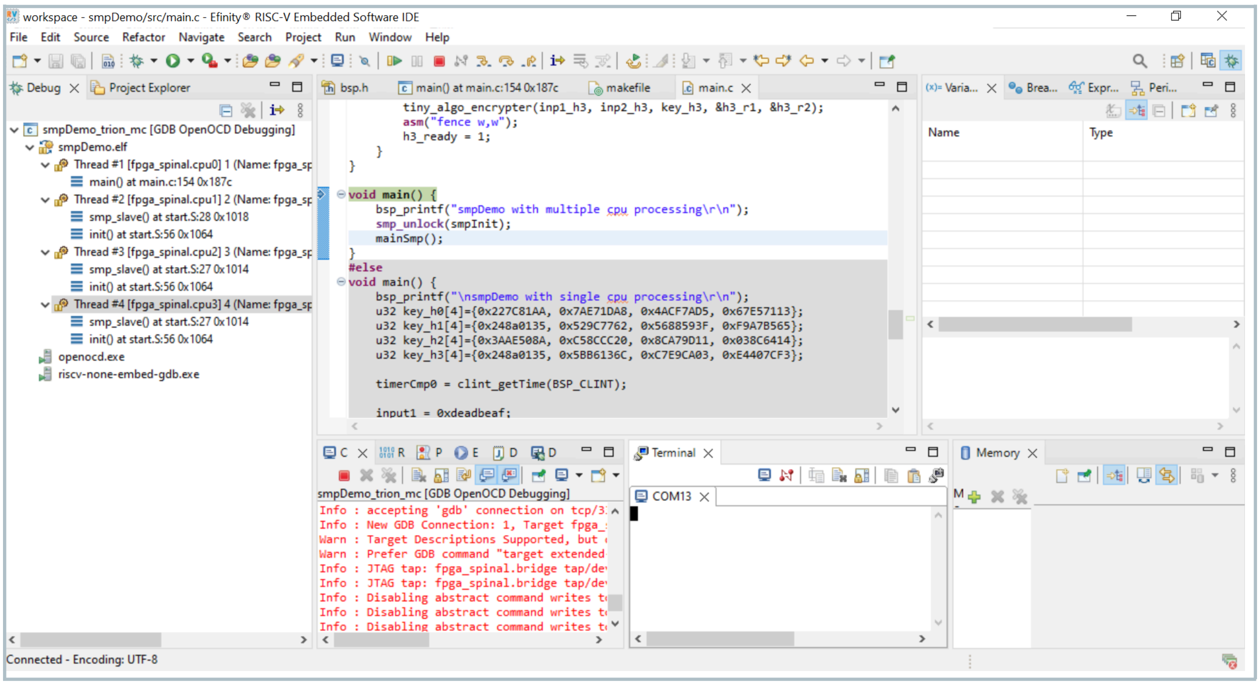
If the `*_mc.launch` scripts are not generated in your Efinity RISC-V Embedded Software IDE, it is advisable to check whether you have imported the correct Board Specific Package (BSP) specifically configured for multiple cores.

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 17: SMP Debug using smpDemo



> **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.

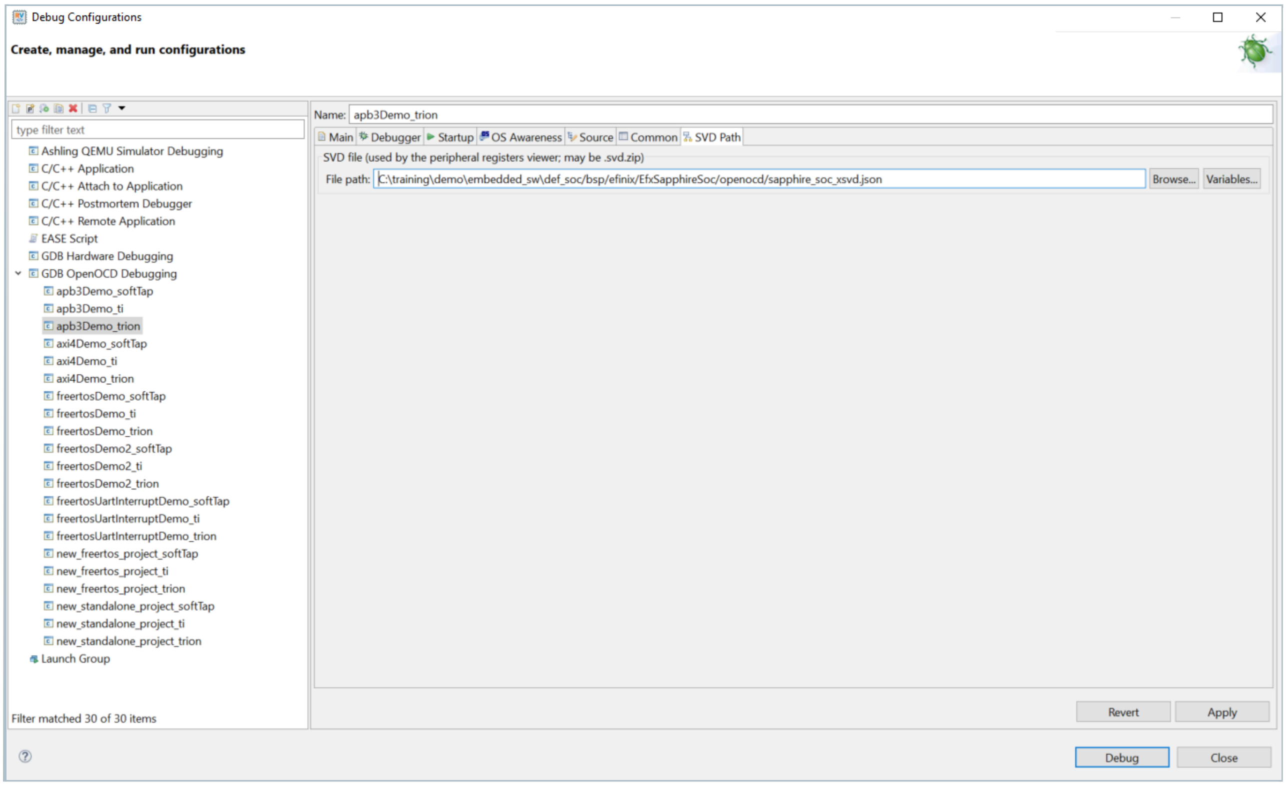
Peripheral Register View

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.

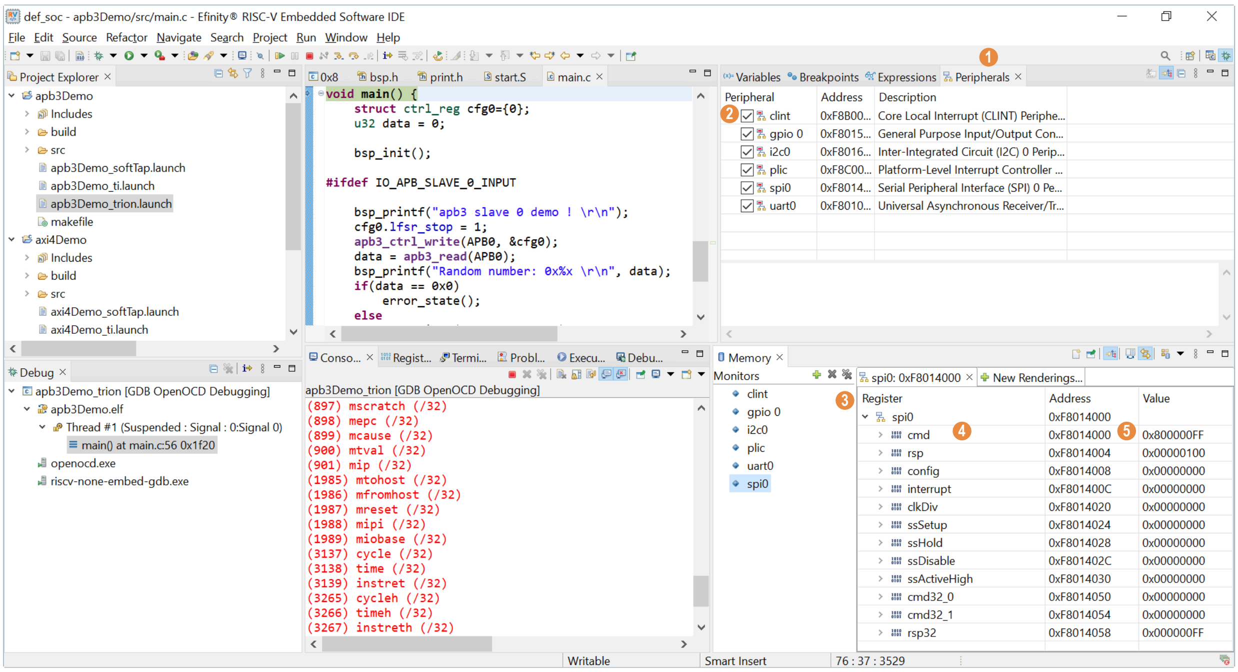
Figure 18: SVD Path Tab in Debug Configuration



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 19: GUI with Peripherals View for all Available Peripherals

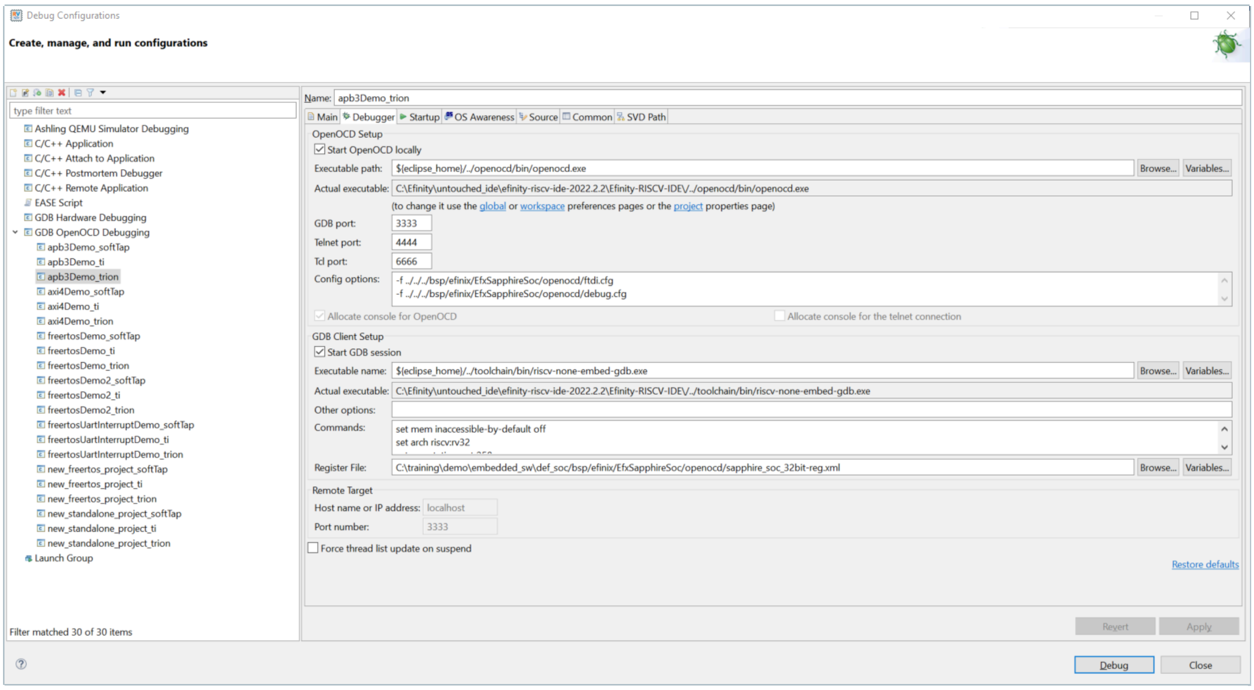


CSR Register View

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 High-Performance 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**.

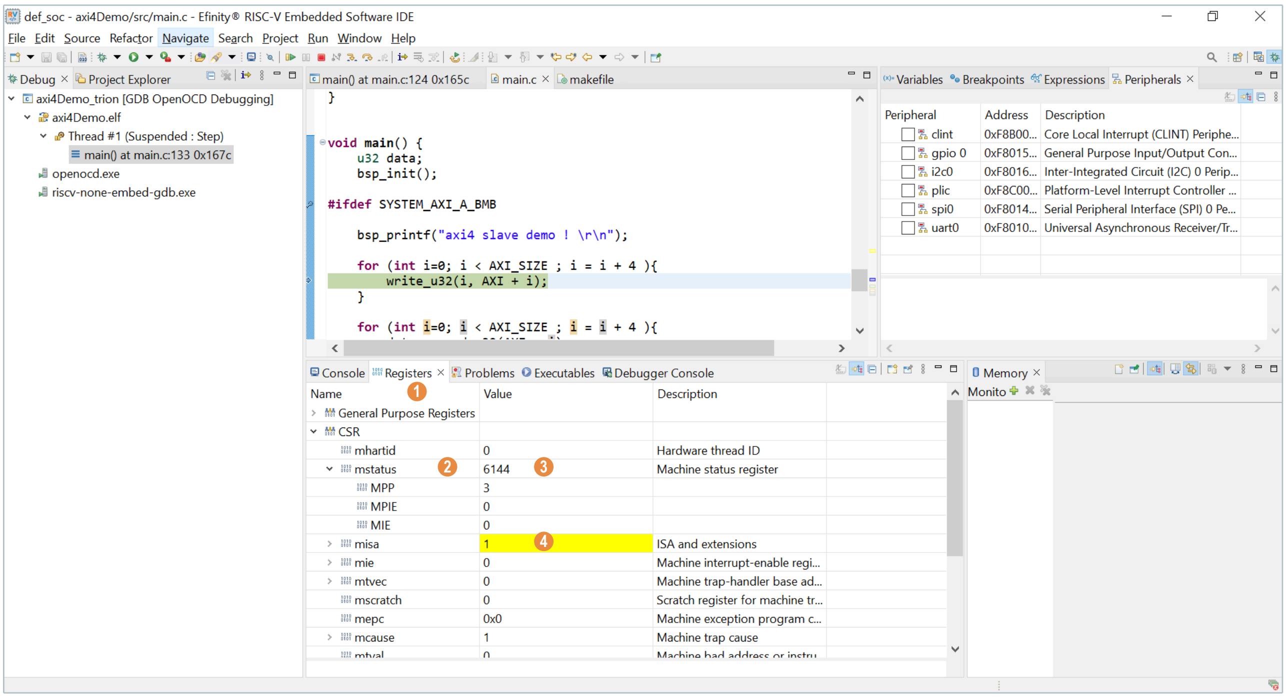
Figure 20: Debug Configurations with Register File



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 High-Performance 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.

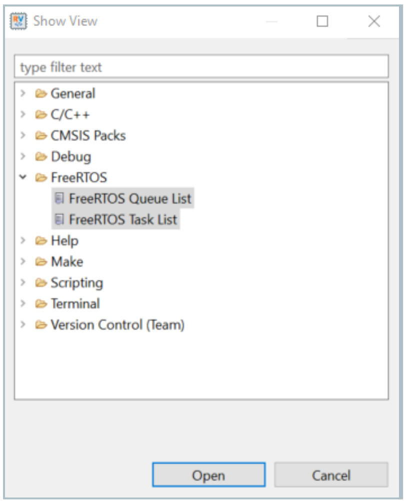
Figure 21: Registers View



FreeRTOS View

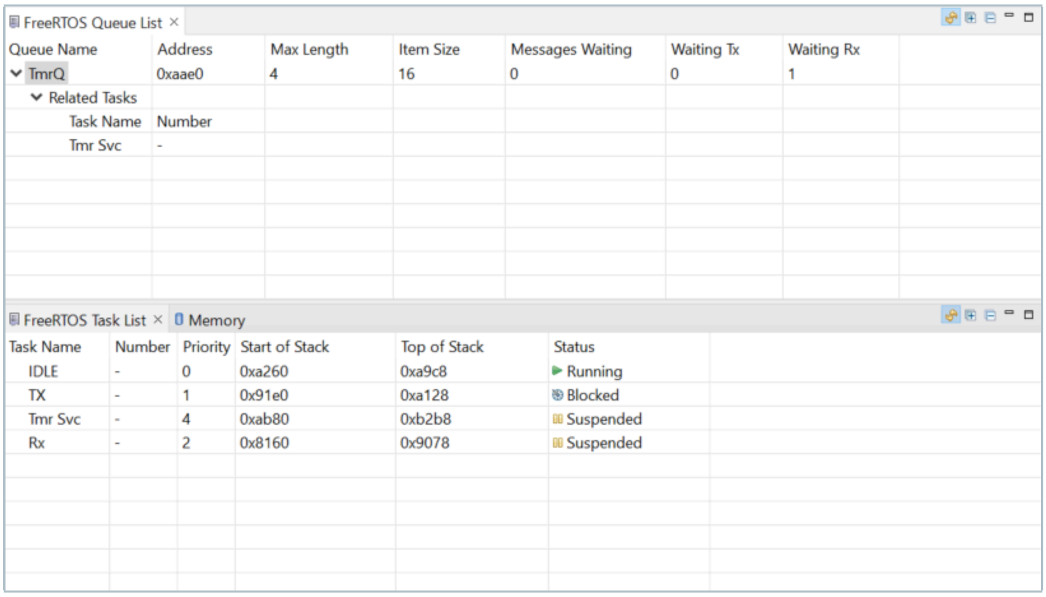
The FreeRTOS View includes **Queue** and **Task List**. These views help during your debugging; you can view the available tasks and their priority and status. It also allows you to view the maximum queue length, messages waiting, etc.

Figure 22: 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 23: FreeRTOS Queue and Task List View



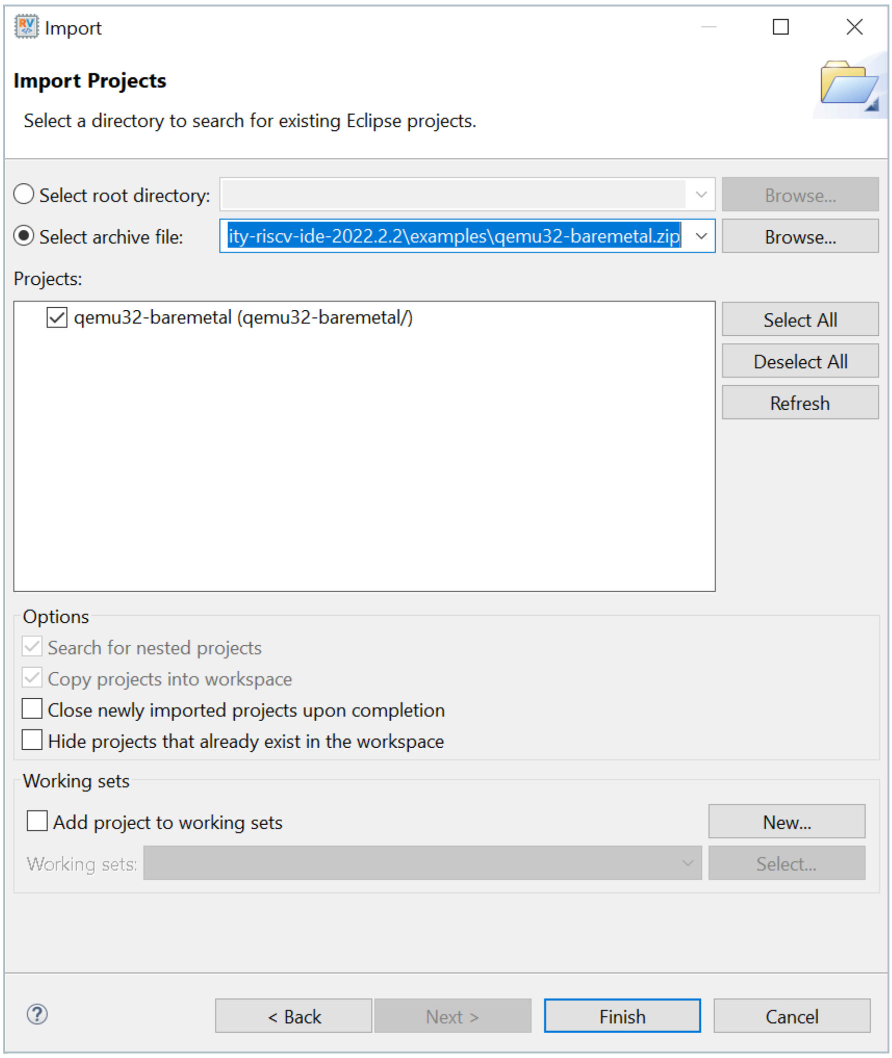
QEMU Emulator

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.

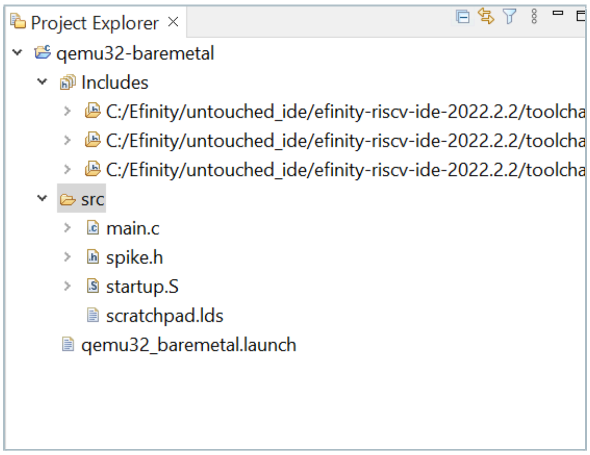
Figure 24: Importing QEMU Project



5. Click **Finish**.

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

Figure 25: Project Explorer Pane showing qemu32-baremetal Project



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 16 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 26: Boot Sequence Flow Chart

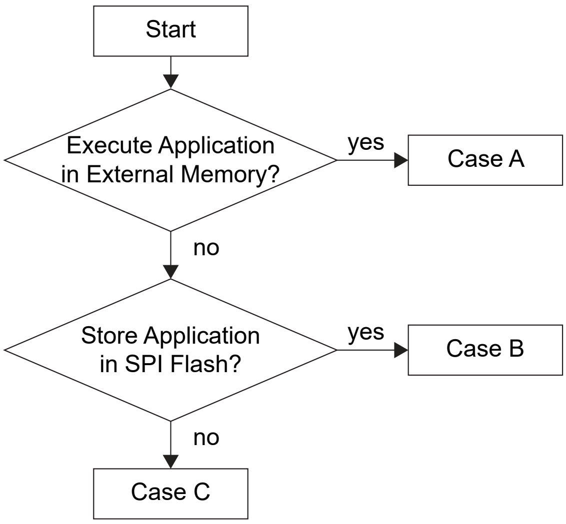


Table 13: User Application

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

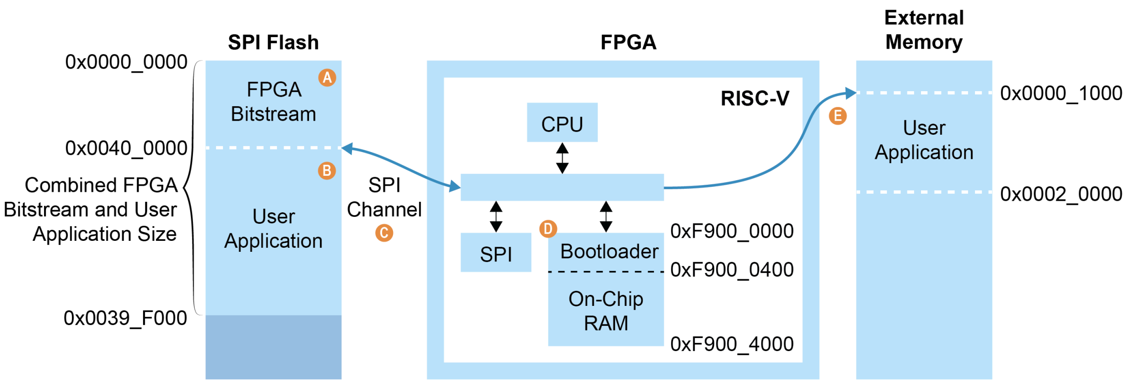
The following sections describe these cases in more detail.

The Sapphire High-Performance SoC supports multiple cores; **Booting Multiple Cores** on page 49 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 27: 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 0x0040_0000.
C. The bootloader reads the SPI flash data from 0x0040_0000.
D. The CPU starts at 0xF900_0000. The On-Chip RAM size is 16 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.

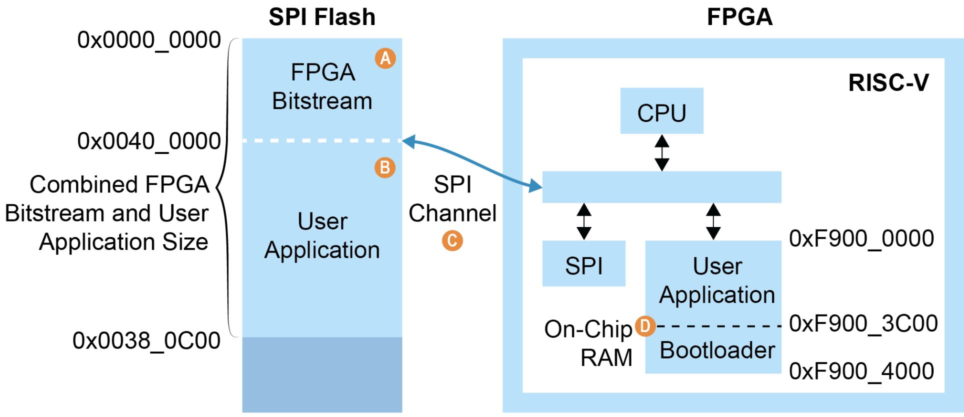
1. The PC starts at the system address 0xF900_0000 of the on-chip RAM.
2. The bootloader starts reading the SPI Flash address 0x40_0000 for the user application.
3. The bootloader writes the user application to external memory starting from system address 0x0000_1000.
4. The bootloader finishes reading 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 28: 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 0x0040_0000.
C. The bootloader reads the SPI flash data from 0x0040_0000.
D. The bootloader copies the SPI flash data to the On-Chip RAM 0xF900_0000 and redirects the address to 0xF900_0000 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_3C00.

The boot sequence is:

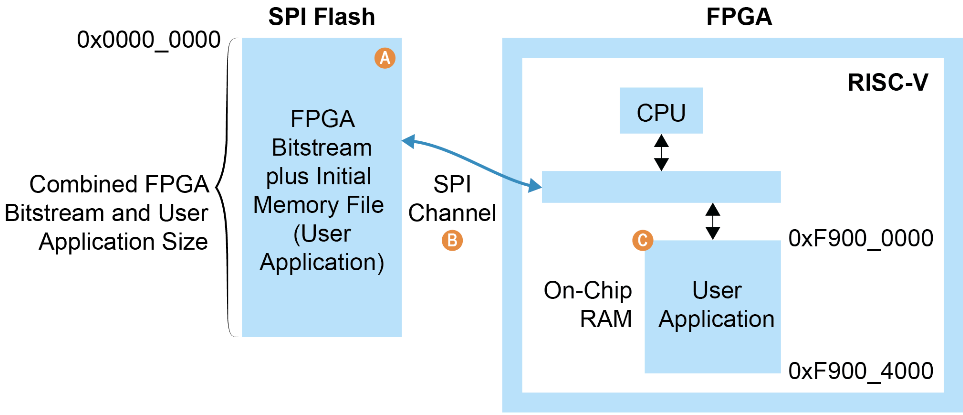
1. The PC starts at the system address 0xF900_0000 of the on-chip RAM and the PC jumps to 0xF900_0C00 for bootloader execution.
2. The bootloader starts reading the SPI Flash address 0x0040_0000.
3. The bootloader writes the user application to On-Chip RAM starting from system address 0xF900_0000.
4. 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 29: Boot Sequence Diagram



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

The Sapphire High-Performance SoC has four 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 30: Boot Sequence for Multiple Cores

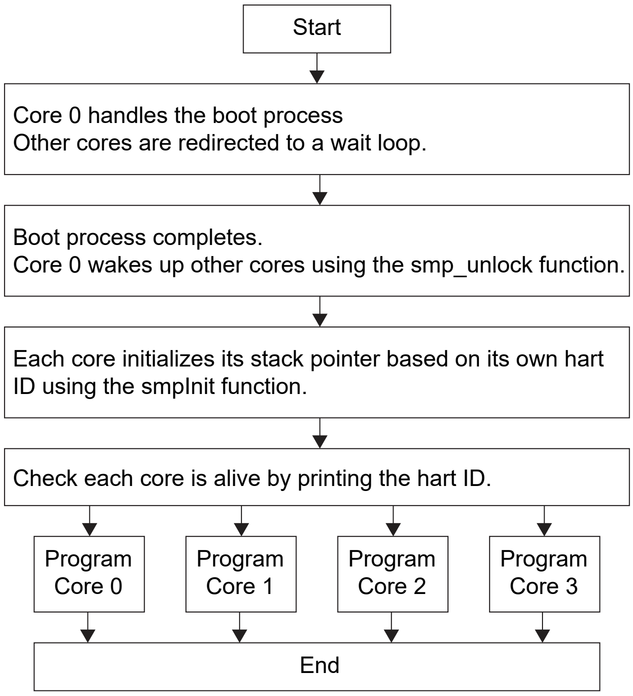


Table 14: SMP Helper Functions

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 High-Performance using the included example Efinity® project, you can use these tips to modify the design for your own use.

Note: Elitestek 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 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 High-Performance 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**.

Note: Elitestek 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** 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 15: 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 TJ-Series 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_ti.cfg** for TJ-Series 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 High-Performance SoC.
2. Import your desired project (i.e., gpioDemo) in the Efinity RISC-V Embedded Software IDE.
3. Right-click on the **gpioDemo_ti** file in the Project Explorer pane to open the **Debug Configuration** setting.
4. In the **Debugger** tab, browse to the **OpenOCD Setup** section. There, you would see the **Config options** text box. Replace the **ftdi_ti.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.
5. 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_ti.cfg** (TJ-Series) file.

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

Chapter 10

Create Your Own Software

Contents:

Now that you have explored the methodology for designing with the Sapphire High-Performance 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 High-Performance 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 High-Performance 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 0x0040_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 0x0040_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 High-Performance SoC jumps to logical address 0x0000_1000 to execute the user binary.

Note: Refer to **Boot Sequence** on page 46 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 36 for complete instructions.

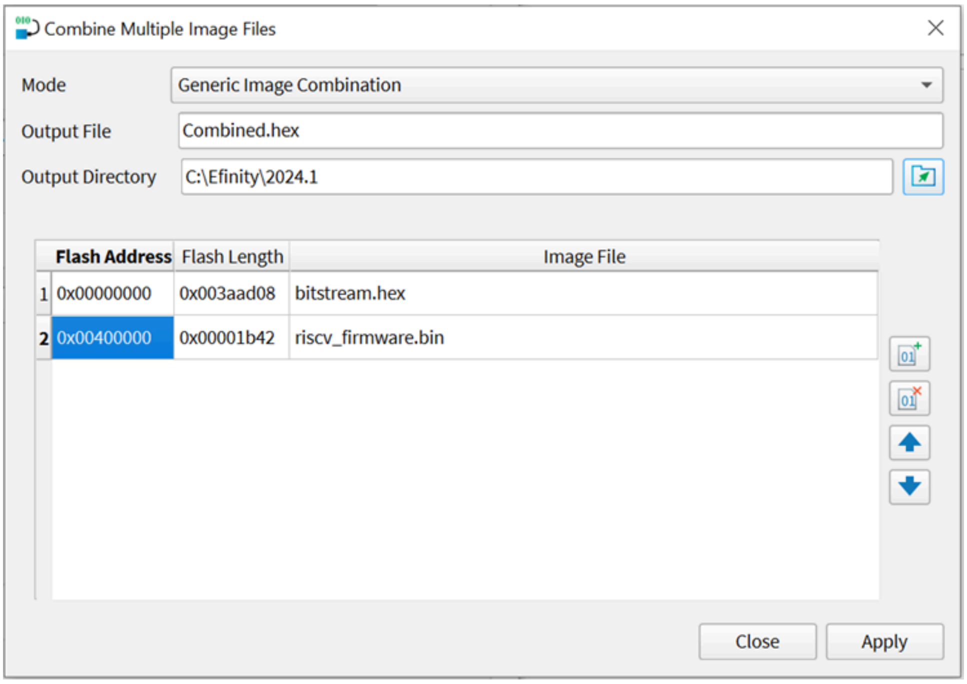
Copy a User Binary to Flash (Efinity Programmer)

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	0x00400000

Figure 31: Combining a Bitstream and RISC-V Application Binary



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 using JTAG Bridge (new)**.

12. Reset the FPGA or power cycle the board.

About the Board Specific Package

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

Table 16: BSP Files

File or Directory	Description
app	Third-party application libraries, i.e. FatFS.
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.
linker\freertos.ld	Linker script for the FreeRTOS application running on main memory address and size.
linker\freertos_i.ld	Linker script for the FreeRTOS application running on internal memory address and size.

Address Map

Because the address range might be updated, Elitestek 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 17: 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	3.7 GB	–	Cache
AXI user slave	SYSTEM_AXI_A_BMB	256 MB	–	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
CLINT timer	SYSTEM_CLINT_CTRL	4 K	–	I/O
PLIC	SYSTEM_PLIC_CTRL	4 MB	–	I/O
On-chip BRAM	SYSTEM_RAM_A_BMB	16 KB	–	Cache
External interrupt	–	–	[A]: 1 [B]: 2 [C]: 3 [D]: 4 [E]: 5 [F]: 6 [G]: 7 [H]: 8 [I]: 9 [J]: 10 [K]: 11 [L]: 12 [M]: 13 [N]: 14 [O]: 15 [P]: 16 [Q]: 17 [R]: 18 [S]: 19 [T]: 20 [U]: 21 [V]: 22 [W]: 23 [X]: 24	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:

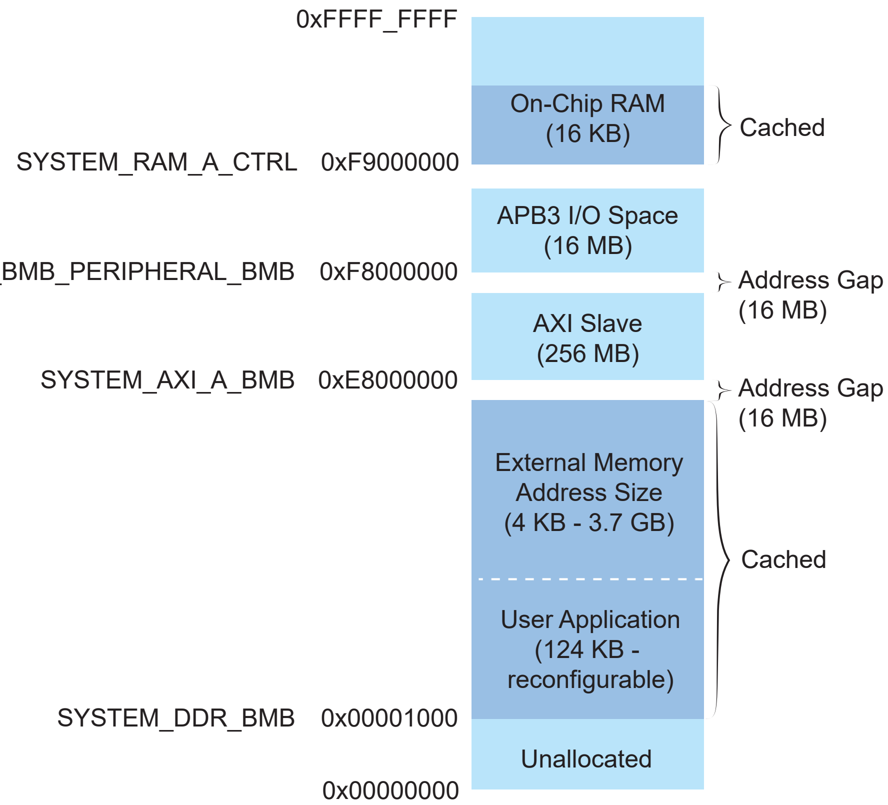
```
*((volatile u32*) address);
```

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 32: Sapphire High-Performance Memory Space



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

Figure 33: Sapphire High-Performance APB3 I/O Space (Offset 0xF8000000 – 16 MB)

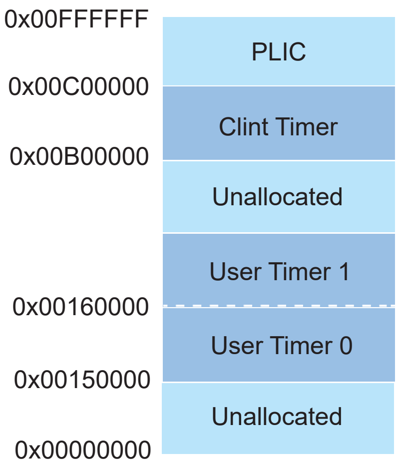
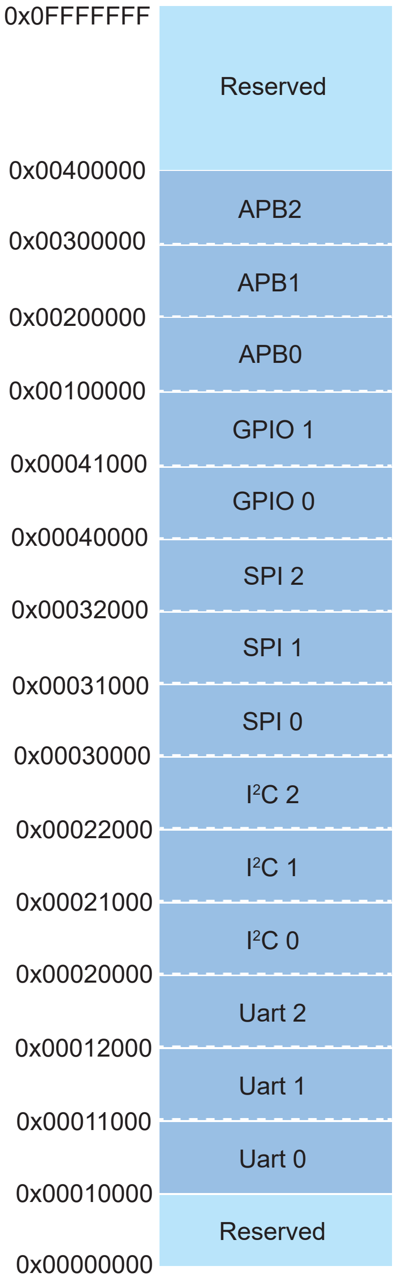


Figure 34: Sapphire High-Performance AXI4 I/O Space (Offset 0xE8000000 – 256 MB)



Example Software

To help you get started writing software for the Sapphire, Elitestek 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 18: Example Software Code

Directory	Description
bootloader	This software is the bootloader for the system.
common	Provides linking for the makefiles.
clintTimerInterruptDemo	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 Example	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 99 for details.
fatFSDemo	This example demonstrates the implementation of the FatFS File System with a Command Line Interface (CLI) for interaction.
FreeRTOS Examples	This example shows the example software projects targeting the RTOS.
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.
lwipIperfServer	This example illustrates how to use the LWIP software stack to enable the Sapphire high-performance RISC-V SoC as an Iperf server.
memTest Example	This example provides example code that performs a memory test on the external memory module and reports the results on a UART terminal.
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.
oob Example	The out-of-box example provides example code that performs multi core operation where Core 0 controls the LED(s) blinking while Core 1 controls the printing of a rotating donut.
rtcDemo	This example shows how to use the on-board PCF8523 RTC module on the Ti375C529 FPGA.
sdhcDemo	This example evaluates the throughput performance of the SD Host Controller (SDHC) by reading and writing a specific amount of data to and from the SD card.
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.
temperatureSensorDemo	This example shows how to communicate with the on-board EMC1413 temperature module on Ti375 C529 Development Board.
uartEchoDemo	This example shows how to use the UART.
UartInterruptDemo	This exmple shows how to use a UART interrupt.

Directory	Description
userTimerDemo	This example shows how to use the user timer with interrupt.

clintTimerInterruptDemo

This demo (**clintTimerInterruptDemo** 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.

```
***Starting Clint Timer Interrupt Demo***
Entering clint timer interrupt routine ..
Count:0 .. Done
Entering clint timer interrupt routine ..
Count:1 .. Done
Entering clint timer interrupt routine ..
Count:2 .. Done
Entering clint timer interrupt routine ..
Count:3 .. Done
Entering clint timer interrupt routine ..
Count:4 .. Done
Entering clint timer interrupt routine ..
Count:5 .. Done
Entering clint timer interrupt routine ..
Count:6 .. Done
Entering clint timer interrupt routine ..
Count:7 .. Done
Entering clint timer interrupt routine ..
Count:8 .. Done
```

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 50,000 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      : 4281210528

Total time (secs): 21.406053

Iterations/Sec   : 2335.787959

Iterations       : 50000

Compiler version : GCC8.3.0

Compiler flags   : -o3

Memory location  : STACK

seedcrc          : 0xe9f5

[0]crclist       : 0xe714

[0]crcmatrix     : 0x1fd7

[0]crcstate      : 0x8e3a

[0]crcfinal      : 0xa14c

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

CoreMark 1.0 : 2335.787959 / 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`.

```
***Starting Custom Instruction Demo***
Custom instruction method processing clock cycles: 1791
Software method processing clock cycles: 6667
Custom instruction and software output results are matched ..
***Succesfully Ran Demo***
```

dCacheFlushDemo

This example (**dCacheFlushDemo** directory) illustrates how to invalidate the data cache by using API.

```
***Starting Invalidate Data Cache Demo***
Invalidate 3 cache lines ..
Invalidate all cache line ..
***Succesfully Ran Demo***
```

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 rdcycle(), HZ=200000000
   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
   Ch_1_Glob:         A
   should be:         A
   Ch_2_Glob:         B
   should be:         B
   Arr_1_Glob[8]:     7
   should be:         7
   Arr_2_Glob[8][7]: 510
   should be:         Number_Of_Runs + 10
   Ptr_Glob->
   Ptr_Comp:          23088
   should be:         (implementation dependent)
   Discr:             0
   should be:         0
   Enum_Comp:         2
   should be:         2
   Int_Comp:          17
   should be:         17
   Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
   should be:         DHRYSTONE PROGRAM, SOME STRING
   Next_Ptr_Glob->
   Ptr_Comp:          23088
   should be:         (implementation dependent), same as above
   Discr:             0
   should be:         0
   Enum_Comp:         1
   should be:         1
   Int_Comp:          18
   should be:         18
   Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
   should be:         DHRYSTONE PROGRAM, SOME STRING
   Int_1_Loc:         5
   should be:         5
   Int_2_Loc:         13
   should be:         13
   Int_3_Loc:         7
   should be:         7
   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: 2
   Dhrystones per Second:                      9292
   User_Time : 231101
   Number_Of_Runs : 500
   HZ : 200000000
   DMIPS per Mhz:                              1.23
```

fatFSDemo

This example (**fatFSDemo** directory) demonstrates the implementation of the FatFS File System with a Command Line Interface (CLI) for interaction. The disk IO layer is ported to the SD Host Controller.

Upon execution, the example initializes the SD Host Controller and the FAT File System automatically. Additionally, the FatFSDemo is integrated with the Real-Time Clock (RTC) available on board. You can configure the RTC using the **rtcDemo** provided within the BSP.

```
***FatFs File System Demo***
Initialize...
Filesystem found in SD card ..
[Buffer controls]
 bd <ofs> - Dump working buffer
 be <ofs> [<data>] ... - Edit working buffer
 br <pd#> <lba> [<count>] - Read disk into working buffer
 bw <pd#> <lba> [<count>] - Write working buffer into disk
 bf <val> - Fill working buffer
[File system controls]
 fi <ld#> [<mount>]- Force initialized the volume
 fs [<path>] - Show volume status
 fl [<path>] - Show a directory
 fo <mode> <file> - Open a file
    mode 0 => Open existing file
    mode 1 => Open as read file
    mode 2 => Open as write file
    mode 4 => Create new file
    mode 8 => Create new file always
    mode 16 => Open a file always
    mode 48 => Open a file append
 fc - Close the file
 fe <ofs> - Move fp in normal seek
 fd <len> - Read and dump the file
 fr <len> - Read the file
 fw <len> <val> - Write to the file
 fn <org.name> <new.name> - Rename an object
 fu <name> - Unlink an object
 fv - Truncate the file at current fp
 fk <name> - Create a directory
 fa <atrr> <mask> <object name> - Change attribute of an object
 ft <year> <month> <day> <hour> <min> <sec> <name> - Change timestamp of an object
 fx <src.file> <dst.file> - Copy a file
 fg <path> - Change current directory
 fq - Show current directory
 fb <name> - Set volume label
 fm <ld#> <type> <csize> - Create file system
 fz [<len>] - Change/Show R/W length for fr/fw/fx command
[Misc commands]
 md[b|h|w] <addr> [<count>] - Dump memory
 mf <addr> <value> <count> - Fill memory
 me[b|h|w] <addr> [<value> ...] - Edit memory
 t [<year> <mon> <mday> <hour> <min> <sec>] - Set/Show RTC
```

The following table lists the common features used for file systems along with the corresponding commands and examples to use them:

Table 19: List of Commands and Examples

Feature	Command	Example
Initialize file system	fi <ld#> [<mount>]	fi 0 1
Show volume status	fs [<path>]	fs
Show directory	fl [<path>]	fl
Create directory	fk <name>	fk test
Unlink/ Delete file or directory	fu <name>	fu test
Open a new file	fo 4 <file>	fo 4 test.txt
Open a file to write	fo 2 <file>	fo 2 test.txt
Open a file to read	fo 1 <file>	fo 1 test.txt
Write a number of data to file	fw <len> <value>	Write 55 times of decimal 55(ASCII:7) fw 55 55
Dump a number of data from opened file	fd <len>	Dump 100 data from opened file fd 100
Close the opened file	fc	fc
Create file system	fm <type>	Format as FAT32 fm 2
Change the timestamp of an object	ft <year> <month> <day> <hour> <min> <sec> <name>	Change test folder time and date to 2024/05/06 10:33:10 ft 2024 05 06 10 33 10 test
Change current path	fg <path>	fg test

The SD Host Controller (SDHC) example supports both PIO and ADMA accesses. To enable ADMA mode, uncomment the `#define DMA_MODE` preprocessor directive in the **userDef.h** file. Otherwise, comment it out to use the PIO mode.

To enable debug messages, set the `DEBUG_PRINTF_EN` directive to 1. This is beneficial for debugging during the development stage.

FreeRTOS Examples

The Sapphire High-Performance 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**.

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
```

freertosEchoServerDemo

This example demonstrates an Ethernet server that echoes the data packets received from the client, which in this case would be your machine. The echoed data will be printed out via the UART terminal. The design utilizes the *FreeRTOS-Plus-TCP* library.

Note: Before running this design, you need to set up your Ethernet adapter similar to the **lwiplperfServer** example.

To run this design, you need to download the **echoTool** and run it in PowerShell with the following command:

```
.\echotool.exe "192.168.31.55" /p tcp /r 10000 /d strawberry /n 1
.\echotool.exe "192.168.31.55" /p tcp /r 10000 /d apple /n 1
```

The application displays messages on a UART terminal:

```
***Hello world, this is FreeRTOS Echo Server***
Linked Up
Link Partner Full duplex 1000 Mbps

Received bytes: 10, Received data strawberry
Received bytes: 5, Received data apple
```

freertosFatDemo

This example demonstrates how to use the *FreeRTOS-Plus-FAT* library to initialize the SD card as well as to write a text file, **freertos.txt** into the SD Card. This example prints out the SD Card information.

Note: An SD card formatted to FAT32 is required.

The application displays messages on a UART terminal:

```
***Hello world, this is FreeRTOS FAT demo***

--- FreeRTOS Demo Start ---

Initialize...FF_Part: no partitions, try as PBR
****** FreeRTOS+FAT initialized 60432384 sectors
Reading FAT and calculating Free Space
Partition Nr   0
Type           12 (FAT32)
VolLabel       'NO NAME    '
TotalSectors   60432384
SecsPerCluster 32
Size           29492 MB
FreeSize       29491 MB ( 100 percent free )
FF_SDDiskInit: Mounted SD-card as root "/sd0"
Reading FAT and calculating Free Space
Partition Nr   0
Type           12 (FAT32)
VolLabel       'NO NAME    '
TotalSectors   60432384
SecsPerCluster 32
Size           29492 MB
FreeSize       29491 MB ( 100 percent free )

--- FreeRTOS Demo Finish ---
```

freertosIperfDemo

This example demonstrates how to use the *FreeRTOS-Plus-TCP* library to enable the Sapphire High-Performance SoC as an Iperf server on FreeRTOS OS. Iperf is a simple performance measuring tool used to check Ethernet bandwidth. You can use iPerf3 on a PC as the client. Before running the **iPerf3** tool, establish connection between your device and your computer.

Note:

- Before running this design, you need to set up your Ethernet adapter similar to the **lwipIperfServer** example.
- You need to use iPerf3 version 3.1.3 to test. This demo is validated with this version.

```
***Hello world, this is FreeRTOS iPerf demo***
Linked Up
Link Partner Full duplex 1000 Mbps

vIPerfTask: created TCP server socket 210832 port 5001: 0 listen 0
vIPerfTask: created UDP server socket 211328 port 5001: 0
Use for example:
FreeRTOS receive: iperf3 -c 192.168.31.55 -p 5001 -n 100M
FreeRTOS send:    iperf3 -c 192.168.31.55 -p 5001 -n 100M -R
vIPerfTask: Received a connection from 192.168.31.222:1292
TCP[ port 1292 ] recv[ 0 ] 37
Got Control Socket: rc -1: exp: '' got: 'DESKTOP-MI0I69O.1717996238.242000.71'
TCP[ port 1292 ] recv[ 1 ] 4
TCP skipcount 88 xRecvResult 4
TCP[ port 1292 ] recv[ 2 ] 88
Control string:
 {"tcp":true,"omit":0,"num":104857600,"parallel":1,"len":131072,"client_version":"3.1.3"}
vIPerfTask: Received a connection from 192.168.31.222:1293
TCP[ port 1293 ] recv[ 0 ] 37
Got expected client: rc 0: 'DESKTOP-MI0I69O.1717996238.242000.71'
TCP[ port 1292 ] recv[ 3 ] 1
TCP[ port 1292 ] recv 1 bytes: 0x00000004
TCP[ port 1292 ] recv[ 4 ] 4
TCP skipcount 196 xRecvResult 4
TCP[ port 1292 ] recv[ 5 ] 196
vIPerfTCPClose: Closing server socket 192.168.31.222:1293 after 104743357 bytes
vIPerfTCPClose: Closing server socket 192.168.31.222:1292 after 331 bytes
vIPerfTask: Received a connection from 192.168.31.222:5471
TCP[ port 5471 ] recv[ 0 ] 37
Got Control Socket: rc -1: exp: '' got: 'DESKTOP-MI0I69O.1717996257.895331.2b'
TCP[ port 5471 ] recv[ 1 ] 4
TCP skipcount 103 xRecvResult 4
TCP[ port 5471 ] recv[ 2 ] 103
Control string: {"tcp":true,"omit":0,"num":104857600,"parallel":1,"reverse":true,"len":131072,
"client_version":"3.1.3"}
Reverse 1 send 104857600 bytes timed 0: 0
vIPerfTask: Received a connection from 192.168.31.222:5472
TCP[ port 5472 ] recv[ 0 ] 37
Got expected client: rc 0: 'DESKTOP-MI0I69O.1717996257.895331.2b'
TCP[ port 5471 ] recv[ 3 ] 1
TCP[ port 5471 ] recv 1 bytes: 0x00000004
Shutdown connection
TCP[ port 5471 ] recv[ 4 ] 4
TCP skipcount 197 xRecvResult 4
TCP[ port 5471 ] recv[ 5 ] 197
vIPerfTCPClose: Closing server socket 192.168.31.222:5472 after 37 bytes
vIPerfTCPClose: Closing server socket 192.168.31.222:5471 after 347 bytes
```

freertosMqttPlainTextDemo

This demo illustrates how to use the *FreeRTOS-Plus coreMQTT* library. It connects to a local broker, mosquitto, subscribes to a topic, publishes a message to the broker, reads back the message from the broker, and finally unsubscribes from the broker.

The plaintext MQTT demo means that the message transactions are not encrypted.

To get started, you are required to do the following steps:

1. Download **Mosquitto** to act as a local broker.
2. Edit the **mosquitto.conf** file by adding the following lines:

```
Listener 1883 192.168.31.222
allow_anonymous true
```

3. Execute the mosquitto executable in PowerShell with the following command:

```
.\mosquitto.exe -v -c .\mosquitto.conf
```

There would be plenty of printout during an MQTT transaction. The application would display the following messages on a UART terminal indicating that the MQTT transaction has been completed:

```
[INFO] [MQTTDemo] [prvMQTTUnsubscribeFromTopics:870] Unsubscribing from topic testClient09:10:51/
example/topic0.

[INFO] [MQTTDemo] [prvMQTTUnsubscribeFromTopics:870] Unsubscribing from topic testClient09:10:51/
example/topic1.

[INFO] [MQTTDemo] [prvMQTTUnsubscribeFromTopics:870] Unsubscribing from topic testClient09:10:51/
example/topic2.

[INFO] [MQTTDemo] [prvMQTTProcessResponse:947] PUBREL received for packet id PUBCOMP received for
 packet id
[INFO] [MQTTDemo] [prvMQTTProcessResponse:924] UNSUBACK received for packet ID PINGRESP should not
 be handled by the application callback when using MQTT_ProcessLoop.

[INFO] [MQTTDemo] [prvMQTTDemoTask:553] Disconnecting the MQTT connection with 192.168.31.222.

[INFO] [MQTTDemo] [prvMQTTDemoTask:568] prvMQTTDemoTask() completed an iteration successfully.
 Total free heap is Demo completed successfully.

[INFO] [MQTTDemo] [prvMQTTDemoTask:569] Demo completed successfully.

[INFO] [MQTTDemo] [prvMQTTDemoTask:570] -------DEMO FINISHED-------

[INFO] [MQTTDemo] [prvMQTTDemoTask:571] Short delay before starting the next iteration....
```

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.

```
***Starting FPU Demo***
Input 1 (in rad): -0.8414
Sine result: -0.7456
Cosine result: 0.6663
Tangent result: -1.1189
Input 2: 0.4161
Square root result: 0.6450
Divsion result: 0.1131
***Succesfully Ran Demo***
```

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.

```
***Starting GPIO Demo***
Configure GPIOs to blink ..
***Starting GPIO Interrupt Demo***
Press and release onboard button sw4 ..
gpio 0 interrupt routine
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:

```
***Starting Flush Instruction Cache Demo***
Memcpy funcA into array ..
Flush the instruction cache once to avoid preloaded data ..
Expected 'funcA', Obtained : funcA
Memcpy funcB into array ..
Expected 'funcA', Obtained : funcA
Still get the FuncA as there is no cache flush ..
Flush the instruction cache now ..
Expected 'funcB', Obtained : funcB
***Successfully Ran Demo***
```

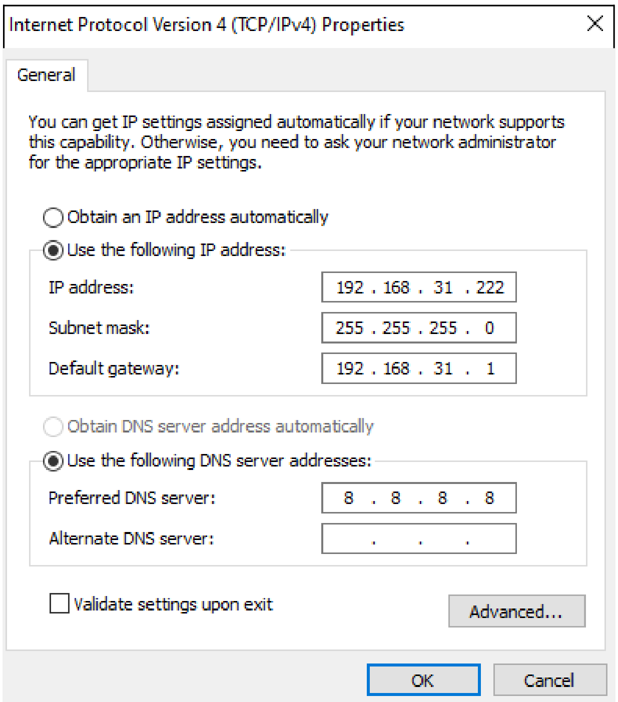
lwipIperfServer

This demonstration (**lwipIperfServer** directory) illustrates how to use the LWIP software stack to enable the Sapphire high-performance RISC-V SoC as an Iperf server. Iperf is a simple performance measuring tool used to check Ethernet bandwidth. You can use iPerf2 on a PC as the client. Before running the iPerf2 tool, ensure that you can ping your device from your computer to establish a connection.

Before running this design, you need to set up your Ethernet adapter as follows:

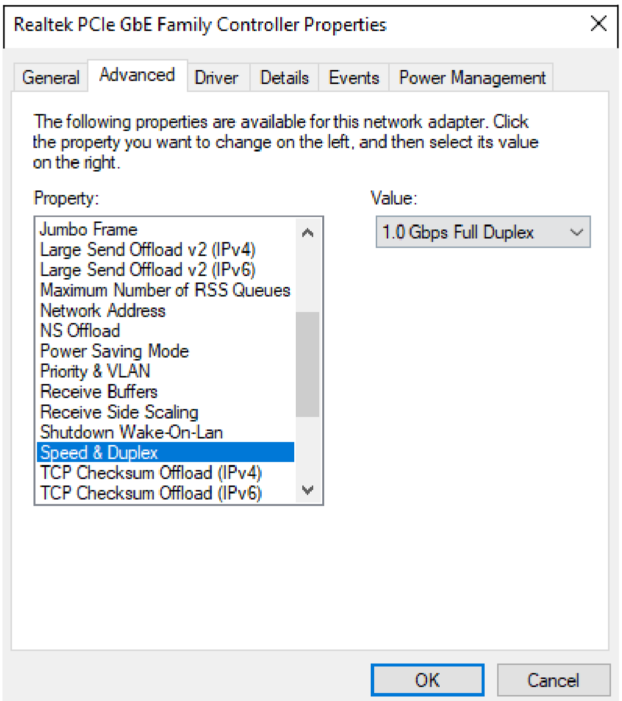
1. Set the IPv4 and netmask as follows:

Figure 35: Setting the IPv4 and Netmask



2. Set the network adapter speed according to your RTL design. The default setting is 1.0 Gbps full duplex.

Figure 36: Setting the Network Adapter Speed



The demo outputs the following messages to a terminal:

```
***Starting TSEMAC Demo***
Phy Init ..
Waiting Link Up ..
Linked Up
Link Partner Full duplex 1000 Mbps

iperf server Up

=========================================
======Lwip Raw Mode Iperf TCP Server ====
=========================================
======IP:                192.168.31.55
======Netmask:           255.255.255.0
======GateWay:           192.168.31.1
======link Speed:        1000 Mbps
=========================================
```

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:

```
***Starting Memory Test***
Data matched .. Test PASSED
***Succesfully Ran Demo***
```

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.

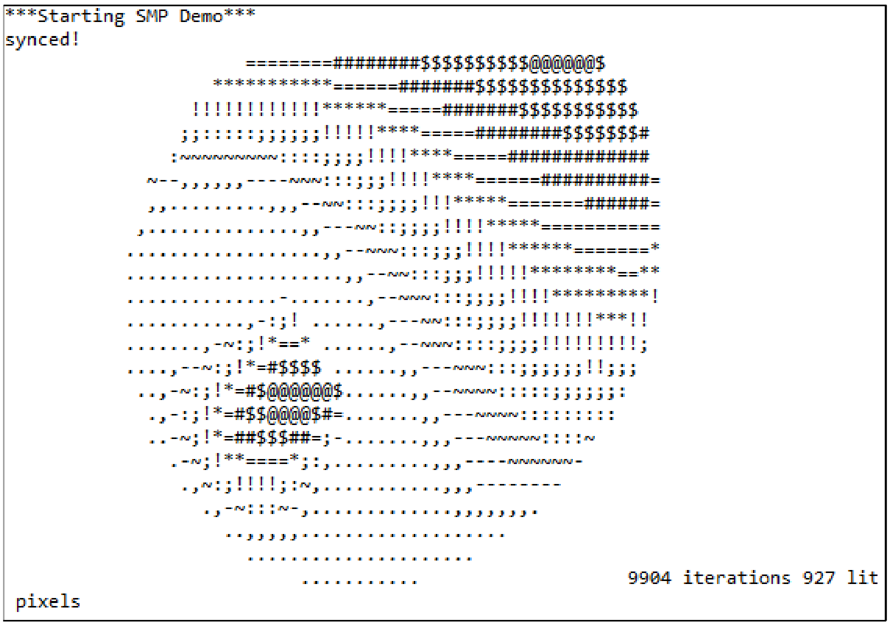
The demo outputs the following messages to a terminal:

```
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
```

oob Example

The out-of-box example (**oob** directory) provides example code that performs multi core operation where Core 0 controls the LED(s) blinking while Core 1 controls the printing of a rotating donut. When you run this example on your development board, you will observe incremental blinking of the LED(s).A successful test prints:

Figure 37: OOB Printout



rtcDemo

This example (**rtcDemo** directory) shows how to use the on-board PCF8523 RTC module on the Ti375C529 FPGA. The demo allows the user to change various configurations such as real-time data with a convertible 12/24hr time system, alarm, and battery setting of the PCF8523 module when the main menu feature is enabled, else it will print the real-time data every few seconds.

```
Welcome to RTC Demo for Ti375C529

************START OF SYSTEM INITIALIZATION************

Checking CR information now !
RTC CR1 readback: 00000002
RTC CR2 readback: 00000000
RTC CR3 readback: 00000008

Enable battery switch-over!
Enable battery low detection function!

Checking battery information now !
Battery status (LOW/OK): OK
Battery switch-over: ENABLED
Battery low detection: ENABLED
Checking completed !

*************END OF SYSTEM INITIALIZATION*************


************Rtc Demo Main Menu***************
Please key in the selection and press enter:
1: Check Time 2: Check Alarm 3: Configure Time
4: Set Alarm  5: Disable/Reset Alarm
Welcome to RTC Demo for Ti375C529

************START OF SYSTEM INITIALIZATION************

Checking CR information now !
RTC CR1 readback: 00000002
RTC CR2 readback: 00000000
RTC CR3 readback: 00000008

Enable battery switch-over!
Enable battery low detection function!

Checking battery information now !
Battery status (LOW/OK): OK
Battery switch-over: ENABLED
Battery low detection: ENABLED
Checking completed !

*************END OF SYSTEM INITIALIZATION*************


************Rtc Demo Main Menu***************
Please key in the selection and press enter:
1: Check Time 2: Check Alarm 3: Configure Time
4: Set Alarm  5: Disable/Reset Alarm
6: Change TimeSystem (12/24hrs)
7: Change Battery mode
8: Check Battery information
9: Soft Reset on RTC module
*****************************************

Showing current time now...
2/5/2024
Thursday,2nd May 2024
Current Time: 14:29:44
```

sdhcDemo

This example (located in the **sdhcDemo** directory) evaluates the throughput performance of the SD Host Controller (SDHC) by reading and writing a specific amount of data to and from the SD card.

```
*** Starting SDHC Demo ***

Initialize ..Respose: 0x40ff8000
Respose: 0xc0ff8000
Done
**************START SPEED TEST*******************
**SD CLOCK SPEED = 50000
**CARD SPEED = 25000 kHz
**CARD SIZE = 29508 Mbyte Total BLOCK = 60432384
**SD BUS WIDTH = 4
**BLOCK SIZE = 512 BUFFER OF BLOCK = 256
**TEST SIZE = 128 kbyte
*************************************************

!!!!Warning it will crash the SD card data!!!!

      ###Push Any Key to Continue###

Tested Block 0/60432384   Write s=16388 KByte/s   Read s=20535 KByte/s
```

Note: Running this design setting would overwrite the data within the SD card causing the File System to be corrupted. You are required to re-format the SD card again after running this demo.

The SD Host Controller (SDHC) example supports both PIO and ADMA accesses. To enable ADMA mode, uncomment the `#define DMA_MODE 1` preprocessor directive in the **userDef.h** file. Otherwise, comment it out to use the PIO mode.

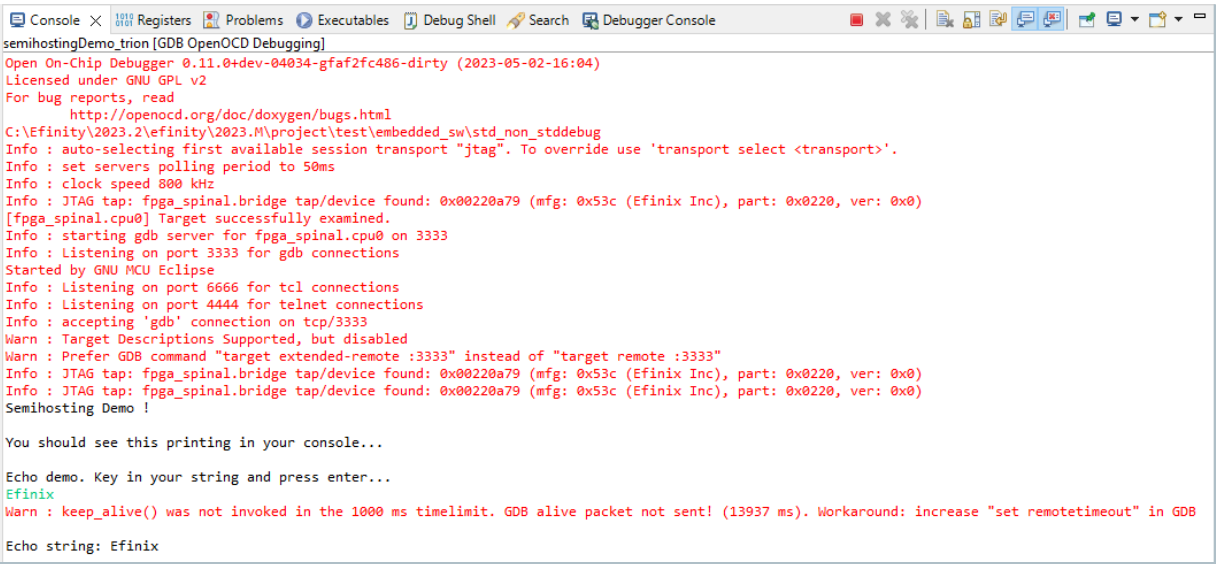
To enable debug messages, set the `DEBUG_PRINTF_EN` directive to 1. This is beneficial for debugging during the development stage.

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 High-Performance 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.



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.

The demo outputs the following messages to a terminal:

```
***Starting SMP Demo***
synced!
processing clock cycles: 4731

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
***Succesfully Ran Demo***
```

temperatureSensorDemo

This example (**temperatureSensorDemo** directory) shows how to communicate with the on-board EMC1413 temperature module on TJ375N529. The demo prints out the temperature of the device every few seconds and alerts user if the temperature exceeds the high-temperature limit. Also, you can enable an extended range of temperature measurements to measure the temperature from -64°C to +191°C. Additonally, you can configure the high or low-temperature limit.

```
Welcome to Temperature Demo for Ti375C529

********************START OF CONFIGURATION***********************
Checking info of the EMC1413 module!
Product ID of temperature sensor:00000021
Status register:00000080
Config register:00000000

Setting up high/low temperature limit!
Set High Temperature limit in internal EMC1413 sensor: 85.0°C
Set High Temperature limit on temperature sensor 1   : 85.0°C
Set High Temperature limit on temperature sensor 2   : 85.0°C
Set Low Temperature limit in internal EMC1413 sensor : 0.0°C
Set Low Temperature limit on temperature sensor 1    : 0.0°C
Set Low Temperature limit on temperature sensor 2    : 0.0°C

Range of the temperature measurement: Default Range (0°C to +127°C)

********************END OF CONFIGURATION**********************

Internal temperature in EMC1413 module   : 38.8750°C
Temperature sensor 1 on Ti375C529 dev kit: 41.5000°C
Temperature sensor 2 on Ti375C529 dev kit: 41.5000°C
```

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.

```
***Starting Uart Echo Demo***
Start typing on terminal to send character...
Echo character: h
Echo character: e
Echo character: l
Echo character: l
Echo character: o
```

UartInterruptDemo

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 an 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:

```
***Starting Uart Interrupt Demo***
Start typing on terminal to trigger uart RX FIFO not empty interrupt ..

Entering uart rx fifo not empty interrupt routine ..
hDone ..

Entering uart rx fifo not empty interrupt routine ..
eDone ..
l
 Entering uart rx fifo not empty interrupt routine ..
lDone ..

Entering uart rx fifo not empty interrupt routine ..
lDone ..
o
 Entering uart rx fifo not empty interrupt routine ..
oDone ..
```

userTimerDemo

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

```
***Starting User Timer Interrupt Demo***
Entering timer 0 interrupt routine ..
Count:1 .. Done
Entering timer 0 interrupt routine ..
Count:2 .. Done
Entering timer 0 interrupt routine ..
Count:3 .. Done
Entering timer 0 interrupt routine ..
Count:4 .. Done
Entering timer 0 interrupt routine ..
Count:5 .. Done
Entering timer 0 interrupt routine ..
Count:6 .. Done
Entering timer 0 interrupt routine ..
Count:7 .. Done
Entering timer 0 interrupt routine ..
Count:8 .. Done
Entering timer 0 interrupt routine ..
Count:9 .. Done
Entering timer 0 interrupt routine ..
Count:10 .. Done
```

Chapter 11

Third-party Debugger

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

The debug scripts are in the **embedded_sw/efx_hard_soc/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

Hardware and Software Migration from Sapphire SoC to Sapphire High-Performance SoC

Contents:

Introduction

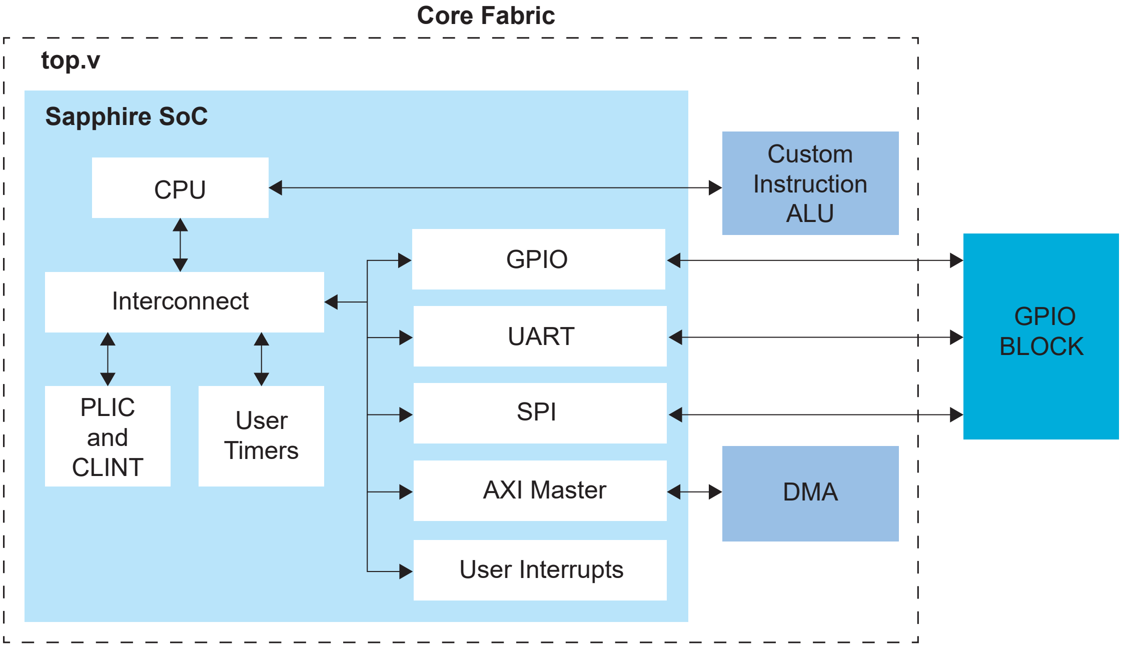
The soft core Sapphire SoC in Elitestek product line is a fully configurable SoC that runs through the FPGA core fabric. There are settings and I/Os you can choose from that allow you to build the SoC that best fits your application. However, if you require firmware that operates at a higher speed, you can select the Sapphire high performance SoC because it has a hardened block embedded in its chip. The quad RISC-V core has a speed of up to 1 GHz. manages and completes complex tasks in a shorter time compared to Sapphire SoC. The architecture is the same for both but the Sapphire high-performance SoC is designed to improve latency and traffic efficiency. Other aspects are very similar to the Sapphire SoC. Thus, the firmware on the Sapphire SoC can run equally well on the Sapphire high-performance SoC. The following section discusses how to port over the design from Sapphire SoC to the Sapphire high-performance SoC.

Hardware

The Sapphire SoC is a design block that has a CPU, peripherals, and I/Os while the Sapphire high performance SoC covers the CPU part and traffic interconnects. It includes the AXI interface ports, custom instruction interface ports, and interrupt ports to core fabric, which allows you to design the peripheral and connects them to the Sapphire high performance SoC.

The following figure illustrates a simplified block diagram of a Sapphire SoC design.

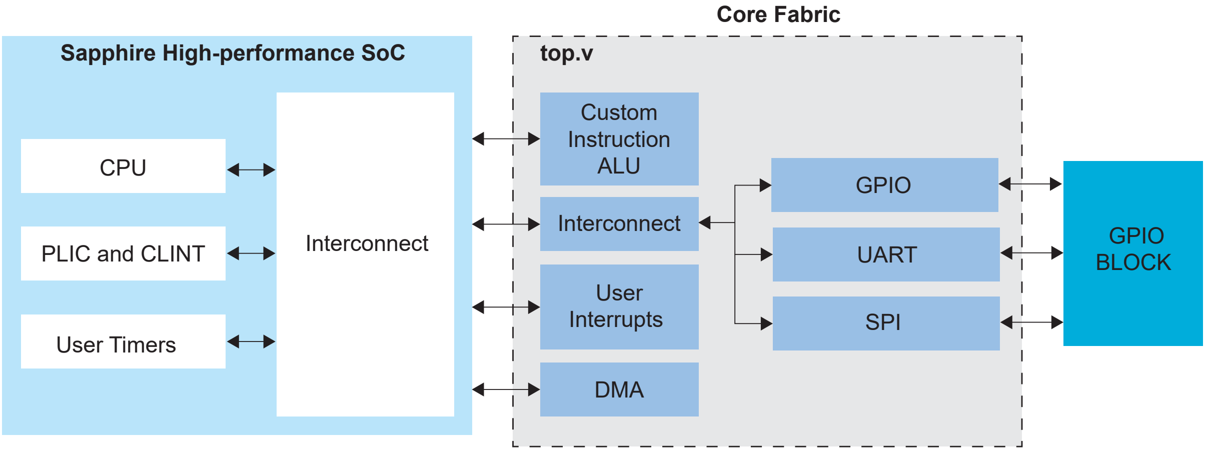
Figure 38: Sapphire SoC Simplified Block Diagram



With reference to the **Figure 38: Sapphire SoC Simplified Block Diagram** on page 82, the CPU and its peripherals are embedded in the core fabric. You are required to instantiate the Sapphire SoC and other logics like DMA and custom logic ALU in the same top file. The input and output pins from the GPIO, UART, and SPI are routed to the GPIO block to communicate to external devices.

The equivalent design of **Figure 38: Sapphire SoC Simplified Block Diagram** on page 82 in the Sapphire high-performance SoC should look like the following figure.

Figure 39: Sapphire High-Performance Simplified Block Diagram



In Sapphire high-performance SoC, you should focus on connecting your logic to the interface pins provided by the SoC. The connection between the Sapphire SoC and the peripheral should be detached. You can put them as the top-level pins to be used later to connect to the Sapphire high-performance SoC interfaces pins. To ease the integration process, Elitestek recommends you the following steps.

Go to **IP Manager** and select **Sapphire High-Performance SoC** under the **Processor and Peripherals Tab**.

1. On the **HRB** or **Hardened RISC-V Block** page, select your desired interface.
2. On the **SLB-I** or **SLB-II** page, select your desired peripheral to instantiate for your design.
3. On the **PLL Configuration** page, enter your clock frequency to run the CPU and interfaces.
4. On the **LPDDR4 Configuration** page, enter the basic configuration to enable the LPDDR4/4x controller.
5. On the **Embedded Software** page, enter the debug type to use in the linker script information.
6. Click **Generate** once you are done with the configuration.

The IP Manager helps to create soft IPs like SPI, UART, and GPIO, and attach to an interconnect. The interconnect is connected to the AXI4 slave interface of Sapphire high-performance SoC. Additionally, it enables the hardened peripherial, e.g., PLL, LPDDR4, JTAG user tap, GPIO block, and hardened SoC block according to your selection in the IP Manager. Furthermore, the IP Manager connects the top-level pins to the hardened SoC block, thus eliminates the need to insert them manually.

Once the generation is completed, you can connect your logic like DMA or custom instruction ALU onto the top file. The IP Manager generates the example top file for your reference, so you do not need to code everything from scratch. The example top file is available at location **ip/EfxSapphireHpSoc_slb/EfxSapphireHpSoc_wrapper.v**.

You can compile the project once you have finished adding your logic to the top file.

Software

The software, which normally refers to the application and driver source code, is compatible with the Sapphire SoC and the Sapphire high-performance SoC. However, you should know the discrepancies between the file structure and content.

1. Obsolete and exclude legacy print functions.

   In Sapphire SoC, there are some legacy codes inherited from the very first version of SoC like Jade, Ruby, and Opal. These codes are excluded to keep cleaner and manageable code structures that deliver to customers. You should update your legacy UART print function to unified `bsp_printf` function. These functions include:

bsp_printHex
bsp_printHex_lower
bsp_printHexDigit
bsp_printHexByte
bsp_printReg
bsp_print

2. Obsolete old definition

   The old definition from legacy SoC like `BSP_MACHINE_TIMER`, `BSP_MACHINE_TIMER_HZ`, `machineTimer_setmp`, `machineTimer_getTime`, `machineTimer_uDelay`, `bsp_putString`, `configMTIME_BASE_ADDRESS`, `configMTIMECMP_BASE_ADDRESS`, `configCPU_CLOCK_HZ`, `BSP_LED_GPIO`, `BSP_LED_MASK` have been removed.
3. **soc.h**, **freertosHalConfig.h**, **print.h**, **print_full.h**, and **semihosting.h** moved from **bsp/efinix/EfxSapphireSoc/app** to **bsp/efinix/EfxSapphireSoc/include** folder.
4. Every hardware definition for demo **bsp/efinix/EfxSapphireSoc/app** has been removed and replaced with **userDef.h** in the demo folder. The **bsp/efinix/EfxSapphireSoc/app** folder redefined to put the middleware like *FATfs* and *LWIP*.
5. Demo folders are restructured and rearranged according to functionality. More demos are added to the folder compared to Sapphire SoC.
6. The size region of the linker script that targets external memory is defaulted to 324KB.
7. The JTAG clock frequency for debugging is defaulted to 6MHz.
8. SoC on-chip RAM is not loaded with SPI flash bootloader by default, you need to compile the bootloader and configure the IP Manager with bootloader hex/bin you compiled.

Chapter 13

Watchdog Timer

Contents:

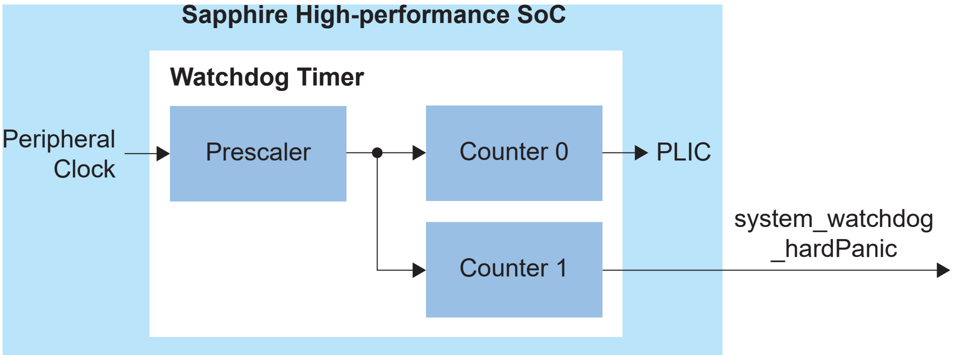
Introduction

The watchdog timer is a safety feature used to monitor a system's proper functioning. Its main purpose is to automatically recover or reset the system in case of software malfunctions or unexpected behavior. This helps to prevent the system from getting hung or entering an unsafe state. The watchdog timer continuously counts towards a preset limit. The software should periodically reset the watchdog timer before reaching the preset limit. If the software fails to reset the watchdog timer because of unexpected issues, the watchdog timer triggers interrupt and a panic signal when it reaches the preset limit.

Functional Description

The watchdog timer in Sapphire SoC has a prescaler and two counters, offers a 2-stage interrupt/panic.

Figure 40: Sapphire High-Performance SoC Watchdog Timer Clock Diagram



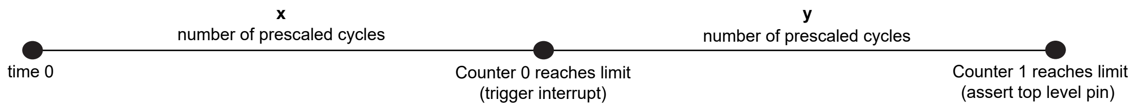
The watchdog timer has two counters, counter 0 and counter 1. Both counters run simultaneously and each counter has its own limit. When the software resets the watchdog timer, both counters reset too. If the watchdog timer does not reset,

- When counter 0 has reached its limit:
  1. The watchdog timer sends an interrupt to the PLIC, which is triggered as an external interrupt in the software.
  2. During the interrupt routine, you can try to recover the software or prepare for a proper shutdown or reset.
- When counter 1 has reached its limit:
  1. The watchdog timer asserts the top level pin, `system_watchdog_hardPanic`.
  2. You can use the signal from this pin to implement their reset or recovery logic for the system.

Setting Limits for Both Counters

Use Case: Counter 1 is designed to reach its limit later than counter 0, so that it gives ample time for recovery or preparation for shutdown in the interrupt routine.

Figure 41: Setting Limits for Counter 0 and Counter 1



In this case,

Limit of counter 0 = x

Limit of counter 1 = x + y

Chapter 14

Using a UART Module

Contents:

A number of the software examples display messages on a UART terminal. You can simply connect a USB cable to the board and to your computer.

Using the On-board UART

The TJ-Series Ti375 C529 Development Board features a USB-to-UART converter connected to the device's GPIOR_145 and GPIOR_165 pins. To use the UART, simply connect a USB cable between the FTDI USB connector on the targeted development board and your computer.

Note: The board has an FTDI chip to bridge communication from the USB connector. FTDI interface 2 on the communicates with the on-board UART. You do not need to install a driver for this interface because when you connect the TJ-Series Ti375 C529 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.

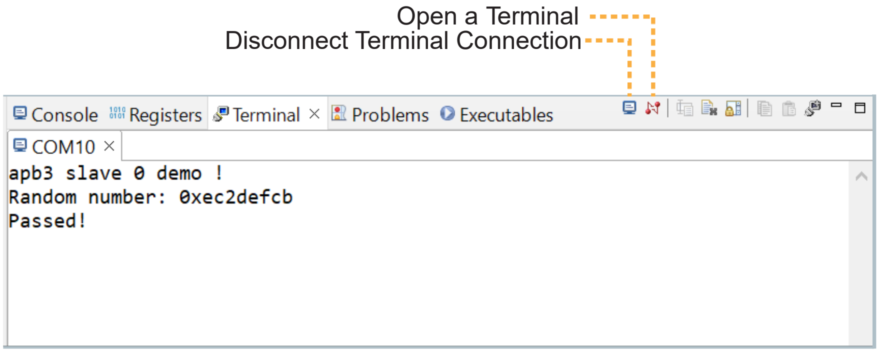
```
/dev/ttyUSB0 /dev/ttyUSB1 /dev/ttyUSB2 /dev/ttyUSB3
```

The UART is `/dev/ttyUSB2`.

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.



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 15

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, Elitestek 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 20: 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 21: 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 22: 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 23: 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 24: 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 High-Performance SoC. Semihosting acts as a bridge between your host machine and the Sapphire High-Performance 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.

Elitestek 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.

Elitestek 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 25: Preprocessor Directives

Switch	Description	Default
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.	Enabled
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 16

Using a Soft JTAG Core for Example Designs

Contents:

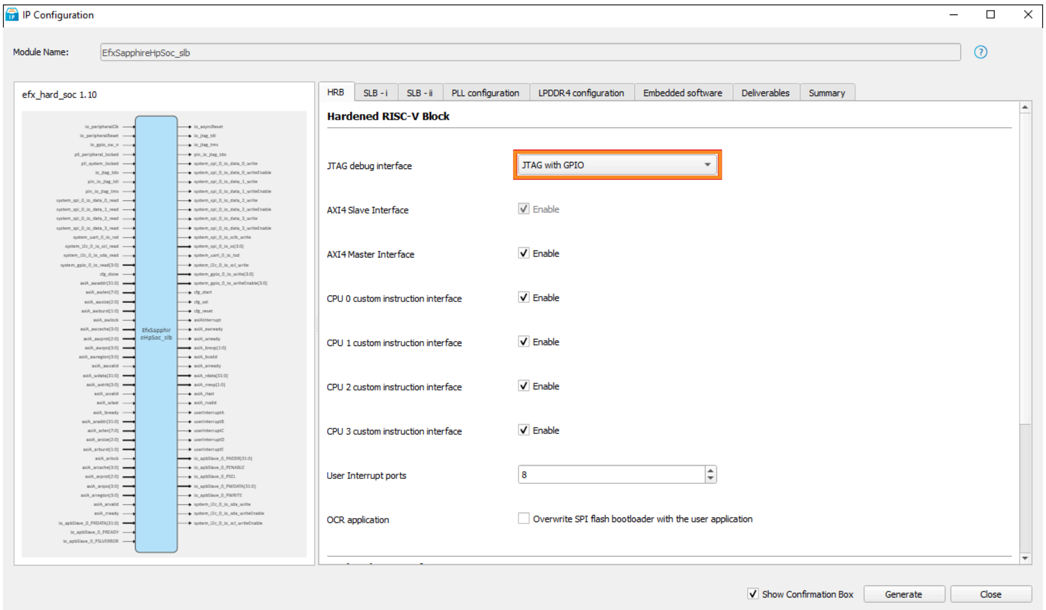
The Efinity® Debugger uses the hard JTAG TAP interface. Out of the box, the TJ375 N529 example design also uses the hard JTAG TAP interface for OpenOCD. If you try to use the same USB connection to the development board for both applications at the same time, there will be conflict. To solve this problem, you use a soft JTAG block to handle the OpenOCD JTAG communication with the Sapphire High-Performance SoC. The TJ-Series TJ375N529 Development Board allocates channel 0 on the on-board FTDI as the soft JTAG connection to Sapphire High-Performance SoC. It is not required to connect any additional FTDI cable to use soft JTAG communication.

Enabling Soft JTAG in Static Example Design

To enable soft JTAG for TJ375 N529 Sapphire High-Performance SoC, follow these steps.

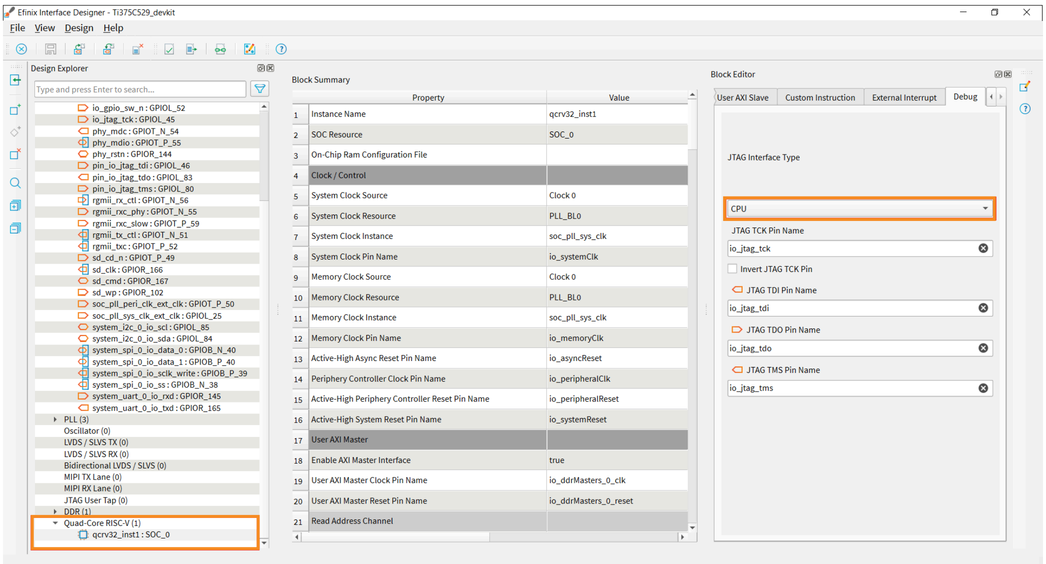
1. Open the Sapphire High-Performance SoC IP Configuration on your current design with Sapphire High-Performance SoC IP. In the **HRB** tab, select **JTAG with GPIO** for JTAG debug interface selection before regenerating the IP. The GPIO JTAG pins (`io_jtag_*`) are included in the GPIO blocks.

Figure 42: Selecting JTAG with GPIO in the HRB Tab



2. Open Efinity Interface Designer. Click on **Quad-Core RISC-V (1)** block in the **Design Explorer**. In the **Block Editor**, click on **Debug** tab. Ensure that the **JTAG Interface Type** for the Quad-Core RISC-V block is configured to **CPU** option. The CPU indicates that JTAG is using the TAP controller from the CPU while selecting the **FPGA** option indicates that JTAG is using the TAP controller from the FPGA device.

Figure 43: Setting JTAG Interface Type for Quad-Core RISC-V Block



3. In the top module (i.e., **top_soc.v**), remove and replace the following I/Os:

Table 26: Remove and Replace I/O

I/O Type	I/O Name
Remove	
Input	ut_jtagCtrl_tdi, ut_jtagCtrl_enable, ut_jtagCtrl_capture, ut_jtagCtrl_shift,ut_jtagCtrl_update, ut_jtagCtrl_reset, jtagCtrl_tdo.
Output	ut_jtagCtrl_tdo, jtagCtrl_tdi, jtagCtrl_enable, jtagCtrl_capture, jtagCtrl_shift, jtagCtrl_update, jtagCtrl_reset.
Replace	
Input	io_jtag_tdo, pin_io_jtag_tdi, pin_io_jtag_tms.
Output	io_jtag_tdi, io_jtag_tms, pin_io_jtag_tdo.

4. In the top module (i.e., **top_soc.v**), remove and replace the following I/Os in the `EfxSapphireHpSoc_slb` module instantiation:

Remove I/O
.jtagCtrl_tdi (jtagCtrl_tdi), .jtagCtrl_tdo (jtagCtrl_tdo), .jtagCtrl_enable (jtagCtrl_enable), .jtagCtrl_capture (jtagCtrl_capture), .jtagCtrl_shift (jtagCtrl_shift), .jtagCtrl_update (jtagCtrl_update), .jtagCtrl_reset (jtagCtrl_reset), .ut_jtagCtrl_tdi (ut_jtagCtrl_tdi), .ut_jtagCtrl_tdo (ut_jtagCtrl_tdo), .ut_jtagCtrl_enable (ut_jtagCtrl_enable), .ut_jtagCtrl_capture (ut_jtagCtrl_capture), .ut_jtagCtrl_shift (ut_jtagCtrl_shift), .ut_jtagCtrl_update (ut_jtagCtrl_update), .ut_jtagCtrl_reset (ut_jtagCtrl_reset).

Replace I/O
.io_jtag_tdi (io_jtag_tdi), .io_jtag_tdo (io_jtag_tdo), .io_jtag_tms (io_jtag_tms), .pin_io_jtag_tdi (pin_io_jtag_tdi), .pin_io_jtag_tdo (pin_io_jtag_tdo), .pin_io_jtag_tms (pin_io_jtag_tms).

5. Compile the design. The Sapphire High-Performance SoC can now be debugged through soft JTAG port when launched with the `*_softTap.launch` (single core) or `*_softTap_mc.launch` (multi core) in the Efinity RISC-V Embedded Software IDE.

Note: On the TJ-Series TJ375 N529 Development Board, you must ensure that the J22 and PJ17 pin headers are not shunted. Shunting these pin headers can cause communication issues with the JTAG signals.

Chapter 17

Troubleshooting

Contents:

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 High-Performance SoC design.
- You may not have the correct PLL settings to work with the Sapphire High-Performance 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.
- Make sure J22 and PJ17 pin headers are not shunted when using soft JTAG.

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** on page 95 for details.

Chapter 18

API Reference

Contents:

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

Control and Status Registers

Note: Refer to Sapphire High-Performance 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^2C
Include	**driver/i2c.h**
Description	Read the 32-bit I^2C 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^2C
Include	**driver/i2c.h**
Description	Read the 32-bit I^2C 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^2C
Returns	[OUT] 32-bit I^2C register interrupt flag
Include	**driver/i2c.h**
Description	Read the 32-bit I^2C 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^2C
Returns	[OUT] 32-bit I^2C register master status
Include	**driver/i2c.h**
Description	Read the 32-bit I^2C 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.

I²C 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 I²C
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 I²C 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 I²C
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^2C
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^2C 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^2C
Include	**driver/i2c.h**
Returns	[OUT] 32-bit current master status: [0] I^2C 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^2C 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^2C
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^2C 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 High-Performance 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 I²C [IN] `value` I²C 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 I²C 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 I²C [IN] `filterID` filter configuration ID number [IN] `config` struct of I²C 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 I²C
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 I²C
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 I²C busy status.

i2c_masterStatus()

Usage	`int i2c_masterStatus(u32 reg)`
Parameters	[IN] `reg` base address of specific I^2C
Include	**driver/i2c.h**
Returns	[OUT] 32-bit current master status: [0] I^2C 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^2C status.

i2c_masterDrop()

Usage	`void i2c_masterDrop(u32 reg)`
Parameters	[IN] `reg` base address of specific I^2C
Include	**driver/i2c.h**
Description	Change the I^2C 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^2C
Include	**driver/i2c.h**
Description	Assert start condition.

i2c_masterRestart()

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

i2c_masterStartBlocking()

Usage	`void i2c_masterStartBlocking(u32 reg)`
Parameters	[IN] `reg` base address of specific I^2C
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^2C
Include	**driver/i2c.h**
Description	Restart the I^2C 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^2C
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^2C
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^2C
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^2C
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^2C [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^2C
Include	**driver/i2c.h**
Description	Transmit acknowledge.

i2c_txAckBlocking()

Usage	`void i2c_txAckBlocking(u32 reg)`
Parameters	[IN] `reg` base address of specific I^2C
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^2C
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^2C [IN] `byte` 8 bits data to send out
Include	**driver/i2c.h**
Description	Transfers one byte to the I^2C slave.

i2c_txByteRepeat()

Usage	`void i2c_txByteRepeat(u32 reg, u8 byte)`
Parameters	[IN] `reg` base address of specific I^2C [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^2C
Include	**driver/i2c.h**
Description	Transfers a NACK.

i2c_txNackRepeat()

Usage	`void i2c_txNackRepeat(u32 reg)`
Parameters	[IN] `reg` base address of specific I^2C
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^2C
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^2C
Returns	[OUT] 1 bit acknowledge
Include	**driver/i2c.h**
Description	Receive an acknowledge from the I^2C slave.

i2c_rxData()

Usage	`u32 i2c_rxData(u32 reg)`
Parameters	[IN] `reg` base address of specific I^2C
Returns	[OUT] 1 byte data from I^2C slave
Include	**driver/i2c.h**
Description	Receive one byte data from I^2C slave.

i2c_rxNack()

Usage	`int i2c_rxNack(u32 reg)`
Parameters	[IN] `reg` base address of specific I^2C
Returns	[OUT] 1 bit no acknowledge. Return 1 if NACK is received.
Include	**driver/i2c.h**
Description	Receive no acknowledge from the I^2C 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^2C [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^2C [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^2C [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^2C [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 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_f2m_dual()

Usage	`void spiFlash_f2m_dual(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 address to read the data [IN] `memoryAddress` RAM address to write the data [IN] `size` size of data to copy
Description	Read data from `flashAddress` and copy to `memoryAddress` of specific size with chip select with dual data lines - half duplex.

spiFlash_f2m_dual_withGpioCs()

Usage	`void spiFlash_f2m_dual(u32 spi, u32 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 address to read the data [IN] `memoryAddress` RAM address to write the data [IN] `size` size of data to copy
Description	Read data from `flashAddress` and copy to `memoryAddress` of specific size with GPIO chip select with dual data lines - half duplex.

spiFlash_f2m_quad()

Usage	`void spiFlash_f2m_quad(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 address to read the data [IN] `memoryAddress` RAM address to write the data [IN] `size` size of data to copy
Description	Read data from `flashAddress` and copy to `memoryAddress` of specific size with chip select with quad data lines - half duplex. Please define `DEFAULT_ADDRESS_BYTE` or `MX25_FLASH` to support the quad data lanes.

spiFlash_f2m_quad_withGpioCs()

Usage	`void spiFlash_f2m_withGpioCs(u32 spi, u32 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 address to read the data [IN] `memoryAddress` RAM address to write the data [IN] `size` size of data to copy
Description	Read data from `flashAddress` and copy to `memoryAddress` of specific size with GPIO chip select with quad data lines - half duplex

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, 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 = 5;
	spiCfg.ssHold = 2;
	spiCfg.ssDisable = 7;

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 = 5;
	spiCfg.ssHold = 2;
	spiCfg.ssDisable = 7;

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)`

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.

spiFlash_manufacturer_id()

Usage	`void spiFlash_manufacturer_id_(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID
Description	Get SPI flash manufacturer ID.

spiFlash_exit4ByteAddr()

Usage	`void spiFlash_exit4ByteAddr(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` SPI base address [IN] `cs` chip select/slaveID
Description	Exit 4-byte addressing. Ensure the addressing mode is set to 3-byte before accessing the SPI Flash. Calling `spiFlash_manufacturer_id_ function` before exiting 4-byte addressing.

spiFlash_exit4ByteAddr_withGpioCs()

Usage	`void spiFlash_exit4ByteAddr_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	Exit 4-byte addressing with GPIO chip select. Ensure the addressing mode is set to 3-byte before accessing the SPI Flash. Calling `spiFlash_manufacturer_id_ function` before exiting 4-byte addressing.

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()

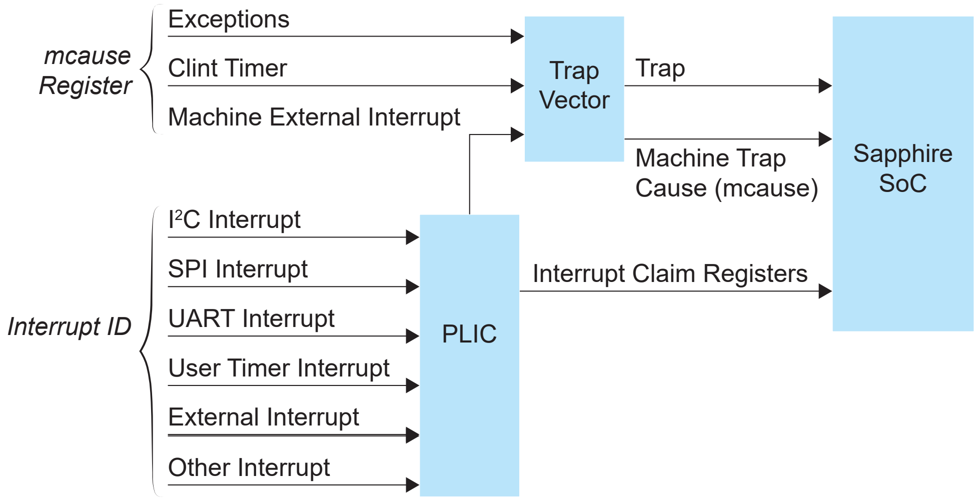
Usage	`void instruction_cache_invalidate(void)`
Include	**driver/vexriscv.h**
Description	Invalidate the whole instruction cache. Critical to ensure the instruction coherency between the cache and the main memory.

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 44: Types of Interrupts

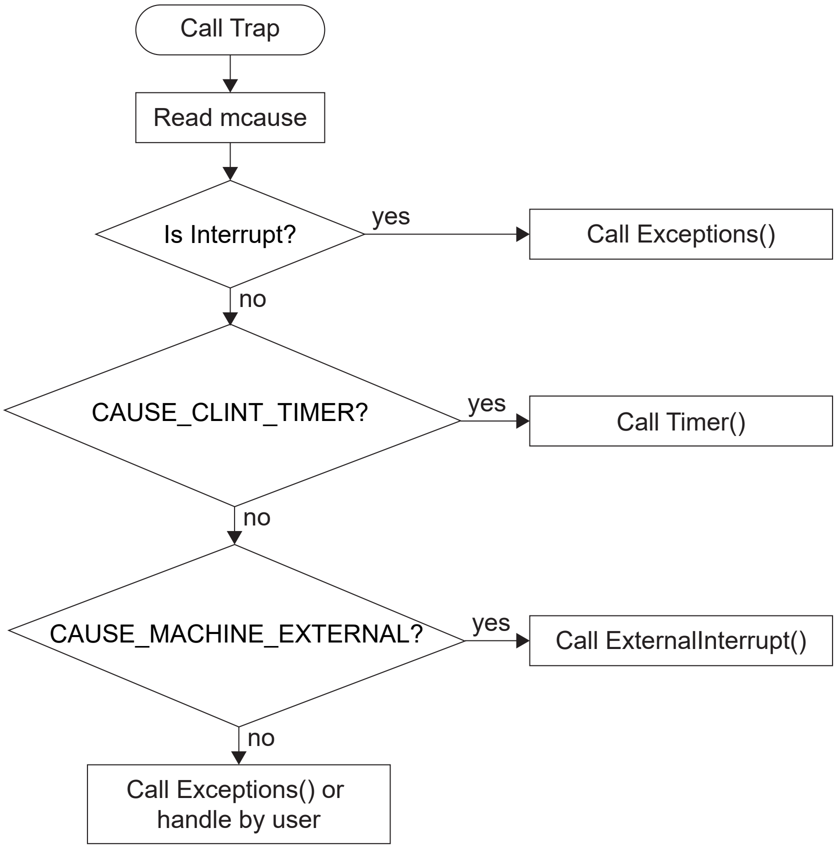


Trap Vectors

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 45: Handling Trap Vectors



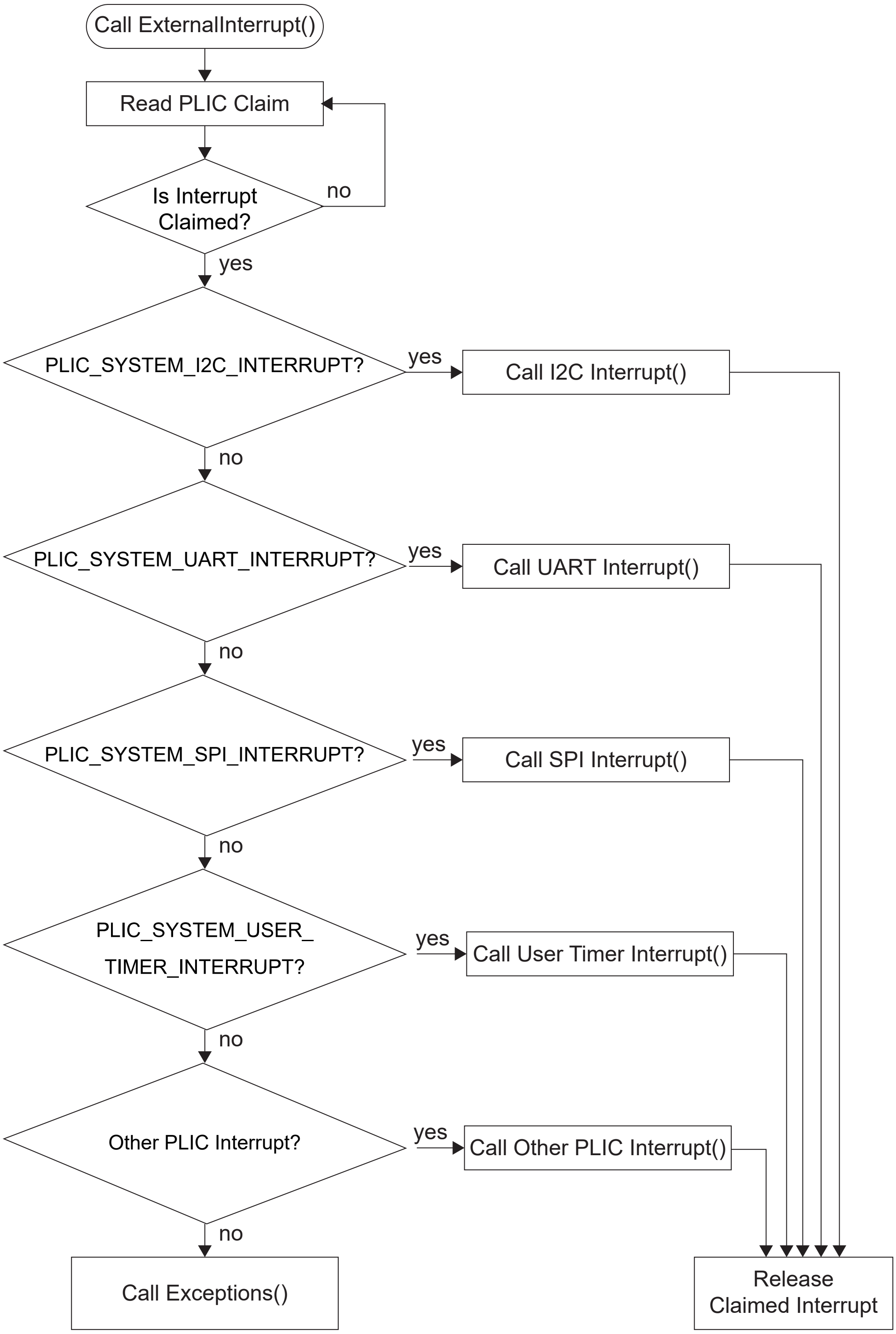
PLIC Interrupts

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 57 for a list of the interrupt IDs.

Note: For the Sapphire High-Performance 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 46: Handling PLIC Interrupts



Chapter 19

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 27: Inline Assembly Syntax

<table>
<tr><th>Syntax</th><th>Description</th></tr>
<tr><td>_asm_</td><td>Indicates the start of the inline assembly block.</td></tr>
<tr><td>asm-qualifiers</td><td>Optional qualifiers that you can use to specify various attributes of the inline assembly, such as constraints, options, or flags, e.g., _volatile_</td></tr>
<tr><td>"assembly_instructions_string"</td><td>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:
<pre>_asm_ _volatile_
(
"label:"
"nop/n"
"j label"
);</pre>
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.
</td></tr>
<tr><td>:"output_operand_list"</td><td>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.</td></tr>
<tr><td>:"input_operand_list"</td><td>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.</td></tr>
<tr><td>:"clobbered_resource_list"</td><td>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.</td></tr>
</table>

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 28: 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 29: 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 30: 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 31: 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[2] CPU.

Figure 47: RISC-V Base Unprivileged Integer Register State

XLEN-1 … 0
x0 / zero
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x16
x17
x18
x19
x20
x21
x22
x23
x24
x25
x26
x27
x28
x29
x30
x31
XLEN

XLEN-1 … 0
pc
XLEN

(2) 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 32: 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.

Revision History

Table 33: Revision History

Date	Version	Description
December 2024	2.0	Added topic Hardware and Software Migration from Sapphire SoC to Sapphire High-Performance SoC. (DOC-1893) Added Watchdog Timer chapter. (DOC-2098)
June 2024	1.0	Intitial release. (DOC-1893)