Sapphire RISC-V SoC Hardware and Software User Guide

Contents

Introduction

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

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

ⓘ **Note:** The Sapphire SoC v2.0, released with the Efinity software v2021.2, has significant improvements from previous versions, and you cannot migrate an existing design to it automatically. 易灵思recommends that you use v2.0 or higher for all new designs. You can continue to use previous versions with the Efinity software v2021.1. If you want to migrate an existing design to v2.0, refer to **Migrating to the Sapphire SoC v2.0** on page 52.

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

1. Windows build tools required on Windows platforms only.
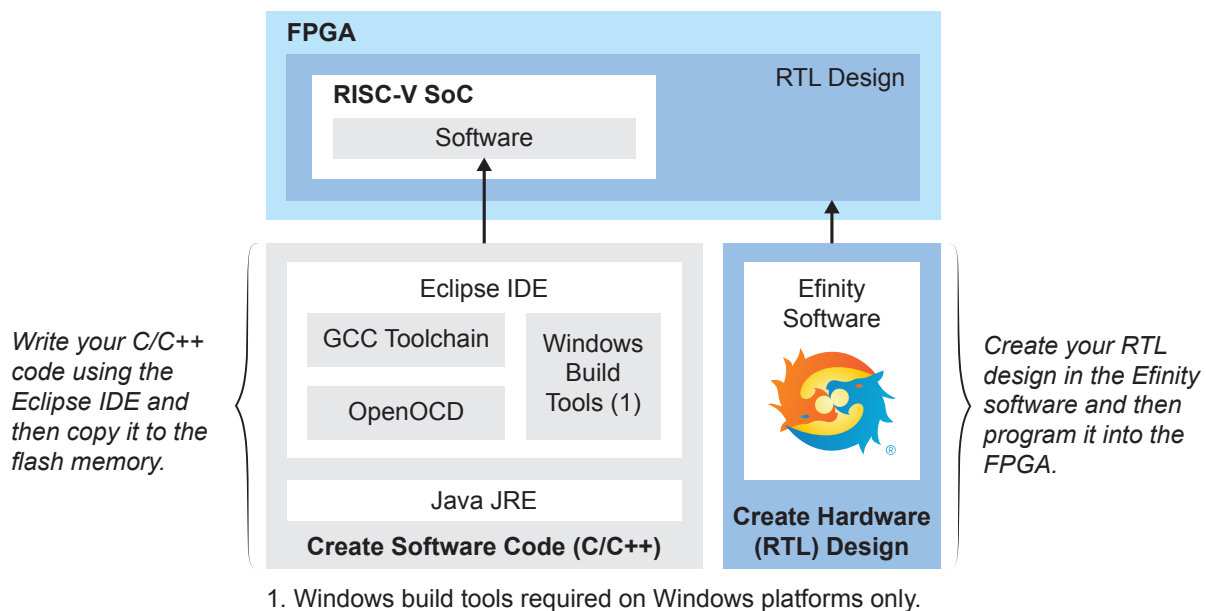
📖 **Learn more:** Refer to the **Sapphire RISC-V SoC Data Sheet** for detailed specifications on the SoC.

VexRiscv RISC-V Core

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

In the Sapphire SoC, the VexRiscv core is user configurable, and can support AXI4 and APB3 bus interfaces and instruction and data caches.

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

Required Software

To write software for the Sapphire SoC, you need the following tools. The SDK is available as a single download in the Support Center for Windows and Ubuntu operating systems.

Efinity® Software

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

Version: 2021.1 or higher

RISC-V SDK

Eclipse MCU—Open-source Java-based development environment that uses plug-ins to extend and customize its functionality. The GNU MCU Eclipse plug-in lets you develop applications for ARM and RISC-V cores.

Version: 2020-09 (4.17.0)

Disk space required: 433 MB (Windows), 433 MB (Linux)

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

Version: 8.3.0-2.3

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

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

Version: 20200421

Disk space required: 9.4 MB (Windows), 7.4 MB (Linux)

GNU MCU Eclipse Windows Build Tool (Windows Only)—This open-source Windows-specific package helps to manage build projects and includes GNU make.

Version: 4.2.1-2-win32-x64

Disk space required: 4.99 MB

Java JRE

Open-source Java 64-bit runtime environment; required for Eclipse.

Version: 8 Update 241

https://www.java.com/en/download/manual.jsp (Java 8 official release)

https://developers.redhat.com/products/openjdk/download (OpenJDK 8 or 11)

http://jdk.java.net/16/ (OpenJDK 16)

---

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

Required Hardware

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

ⓘ **Note:** Some of the software examples provided with the SoC use a UART terminal to display messages. See **Set Up a USB-to-UART Module (Trion)** on page 47 and **Using the On-board UART (钛金系列)** on page 46 for more information.

---

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

Install Software and SoC

Contents:

- **Install the Efinity Software**
- **Install the RISC-V SDK**
- **Install the Java JRE**

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 RISC-V SDK

To install the SDK:

1. Download the file **riscv_sdk_windows-v<version>.zip** or **riscv_sdk_ubuntu-v<version>.zip** from the Support Center.
2. Create a directory for the SDK, such as **c:\riscv-sdk** (Windows) or **home/my_name/riscv-sdk** (Linux).
3. Unzip the file into the directory you created. The complete SDK is distributed as compressed files. You do not need to run an installer.

Windows directory structure:

- **SDK_Windows**
    - **eclipse**—Eclipse application.
    - **GNU MCU Eclipse**—Windows build tools.
    - **openocd**—OpenOCD debugger.
    - **riscv-xpack-toolchain_8.3.0-2.3_windows**—GCC compiler.
    - **run_eclipse.bat**—Batch file that sets variables and launches Eclipse.
    - **setup.bat**—Batch file to set variables for running OpenOCD on the command line to flash the binary.

Ubuntu directory structure:

- **SDK_Ubuntu<version>**
    - **eclipse**—Eclipse application.
    - **openocd**—OpenOCD debugger.
    - **riscv-xpack-toolchain_8.3.0-2.3_linux**—GCC compiler.
    - **run_eclipse.sh**—Shell file that sets variables and launches Eclipse.
    - **setup.sh**—Shell file to set variables for running OpenOCD on the command line to flash the binary.

Install the Java JRE

To install the JRE:

1. Download the 64-bit version of the JRE or JDK for your operating system from
   https://www.java.com/en/download/manual.jsp (Java 8 official release)
   https://developers.redhat.com/products/openjdk/download (OpenJDK 8 or 11)
   http://jdk.java.net/16/ (OpenJDK 16)

2. Follow the installation instructions on the web site to install the JRE.

> ⓘ **Note:** You need a 64-bit version of the Java JRE. If you use a 32-bit version, when you try to launch Eclipse you will get an error that Java quit with exit code 13.

IP Manager

Contents:

- **Customizing the Sapphire SoC**
- **Modify the Bootloader**

The Efinity® IP Manager is an interactive wizard that helps you customize and generate 易灵思® 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 易灵思 development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.

The IP Manager consists of:

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

Generating Sapphire SoC with the IP Manager

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

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

   > (i) **Note:** You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the IP core's user guide or on-line help.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an 易灵思® development board and/or testbench. For SoCs, you can alsooptionally 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.

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

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

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

Generated RTL Files

The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.

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

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

Generated Software Code

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

- **bsp**—Board specific package.
- **config**—Has the Eclipse project settings file and OpenOCD debug configuration settings files for Windows.
- **config_linux**—Has the Eclipse project settings file and OpenOCD debug configuration settings files for Linux.
- **software**—Software examples.
- **tool**—Helper scripts.
- **cpu0.yaml**—CPU file for debugging.

Instantiating the SoC

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

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

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

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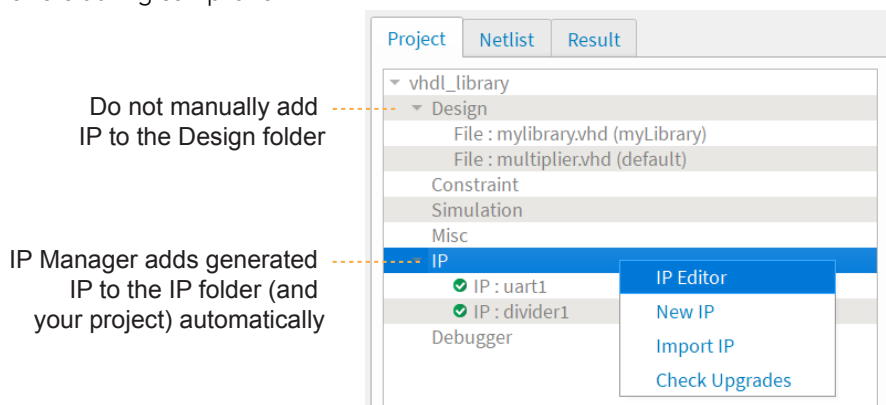
Customizing the Sapphire SoC

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

Table 1: Sapphire SoC Tab Parameters

Parameter	Options	Description
Frequency (MHz)	20 - 400	Enter the frequency in MHz. Default: 100
Peripheral Clock	DISABLE, ENABLE	Choose whether you want to run a dedicated clock for the APB3 slaves (SPI, I2C, GPIO, UART, and user timer) and AXI4 slave.
Peripheral Clock Frequency (MHz)	20 - 200	Enter the peripheral clock frequency in MHz.
Cache	DISABLE, ENABLE	Choose whether you want to include I$ and D$ caches.
Custom Instruction	DISABLE, ENABLE	Choose whether to enable the custom instruction interface.
Linux Memory Management Unit	DISABLE, ENABLE	Choose whether to enable the Linux MMU.
Floating-point Unit	DISABLE, ENABLE	Choose whether to enable the FPU.
Atomic Extension	DISABLE, ENABLE	Choose whether to enable atomic extension instruction support. If you enable the Linux MMU, this option must be enabled and is turned on by default.
Compressed Extension	DISABLE, ENABLE	For cacheless configurations only, choose whether to enable compressed instruction support.

> **Important:** When running the SoC at high frequencies, 易灵思 recommends that you use the TIMING_1 place and route optimization. To set this option:
> 1. Open the Project Editor.
> 2. Click the **Place and Route** tab.
> 3. Double-click the **Value** cell for **--optimization_level**.
> 4. Choose **TIMING_1**.
> 5. Click **OK** and then compile.

Table 2: Sapphire Cache/Memory Tab Parameters

Parameter	Options	Description
Data Cache Way	1, 2, 4, 8	Choose the number of ways for the data cache. Default: 1
Cache Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB	Choose the size of the data cache. Default: 4 KB
Instruction Cache Way	1, 2, 4, 8	Choose the number of ways for the instruction cache. Default: 1
Cache Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB	Choose the size of the instruction cache. Default: 4 KB
External Memory AXI3 Interface	ENABLE, DISABLE	ENABLE: Instantiate the AXI3 interface. DISABLE: Do not use the AXI3 interface.

Parameter	Options	Description
External Memory AXI3 Data Width	32, 64, 128, 256	Choose the data width for the AXI3 interface. Default: 128
External Memory AXI3 Address Size	4 MB, 8 MB, 16 MB, 32 MB, 0.5 GB, 1 GB, 1.5 GB, 2 GB, 2.5 GB, 3 GB, 3.5 GB	Choose the address size for the AXI3 interface. Default: 3.5 GB
On-Chip RAM Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 24 KB, 32 KB, 48 KB, 64 KB, 80 KB, 96 KB, 128 KB, 144 KB, 160 KB, 192 KB, 224 KB, 256 KB, 384 KB, 512 KB	Choose the size of the internal BRAM. Default: 4 KB

Table 3: Sapphire Debug Tab Parameters

Parameter	Options	Description
Soft Debug Tap	DISABLE, ENABLE	Choose whether you want to include a soft debug TAP for debugging. DISABLE: Default. The SoC uses the JTAG User TAP interface block to communicate with the OpenOCD debugger. ENABLE: The SoC has a soft JTAG interface to communicate with the OpenOCD debugger. You need to use this setting if you want to use the soft JTAG interface instead of the JTAG User TAP.
Target OpenOCD	Trion T120 BGA324 Development Board Trion T120 BGA576 Development Board Trion T20 BGA256 Development Board Xyloni 钛金系列 Ti60 F225 Development Kit Custom	Chose which board you want to target with OpenOCD. Choose **Custom** to target your own board.
Custom Target OpenOCD	–	Enter the name of your board.
OpenOCD Debug Mode	Turn on by default Turn off by default	Choose whether you want software applications to run in debug mode by default or not. See **Debug with the OpenOCD Debugger** on page 27 for more details.

Table 4: Sapphire UART Tab Parameters

Where *n* is 0, 1, or 2

Parameter	Options	Description
UART *n*	ENABLE, DISABLE	ENABLE: Instantiate the interface. DISABLE: Do not use the interface.
UART *n* Interrupt ID	1 - 25	Choose the interrupt ID for the UART. The IDs default to: UART 0: 1 UART 1: 2 UART 2: 3

Table 5: Sapphire SPI Tab Parameters

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

Parameter	Options	Description
SPI *n*	ENABLE, DISABLE	ENABLE: Instantiate the interface. DISABLE: Do not use the interface.
SPI *n* Interrupt ID	1 - 25	Choose the interrupt ID for the SPI. The IDs default to: SPI 0: 4 SPI 1: 5 SPI 2: 6

Table 6: Sapphire I2C Tab Parameters

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

Parameter	Options	Description
I2C *n*	ENABLE, DISABLE	ENABLE: Instantiate the interface. DISABLE: Do not use the interface.
I2C *n* Interrupt ID	1 - 25	Choose the interrupt ID for the I^2C. The IDs default to: I2C 0: 8 I2C 1: 9 I2C 2: 10

Table 7: Sapphire GPIO Tab Parameters

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

Parameter	Options	Description
GPIO *n*	ENABLE, DISABLE	ENABLE: Instantiate the interface. DISABLE: Do not use the interface.
GPIO *n* Pin Width	1, 2, 4, 8, 16	Choose the number of pins for the GPIO. Default: 8
GPIO *n* Interrupt ID 0	1 - 25	Choose the interrupt ID for the I2C. The IDs default to: GPIO 0: 12 GPIO 1: 14
GPIO *n* Interrupt ID 1	1 - 25	Choose the interrupt ID for the I2C. The IDs default to: GPIO 0: 13 GPIO 1: 15

Table 8: Sapphire APB3 Tab Parameters

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

Parameter	Options	Description
APB Slave Address Size	4 KB - 1 MB	Choose the APB slave size. This setting applies to all APB slaves.
APB3 *n*	ENABLE, DISABLE	ENABLE: Instantiate the interface. DISABLE: Do not use the interface.

Table 9: Sapphire AXI4 Tab Parameters

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

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

Table 10: Sapphire User Interrupt Tab Parameters

Where *n* is 0 or 1.

Parameter	Options	Description
User *n* Interrupt	ENABLE, DISABLE	ENABLE: Instantiate the interface. DISABLE: Do not use the interface.
User *n* Interrupt ID	1 - 25	Choose the interrupt ID. The defaults are: User 0 Interrupt: 16 User 1 Interrupt: 17

Table 11: Sapphire User Timer Base Address Tab Parameters

Where *n* is 0 or 1.

Parameter	Options	Description
User Timer *n*	ENABLE, DISABLE	ENABLE: Instantiate the interface. DISABLE: Do not use the interface.
User Timer *n* Counter Width	12, 16, 32	Choose the user timer bit width. Default: 12
User Timer *n* Prescaler Width	8, 16	Choose the prescaler bit width. Default: 8
User Timer *n* Interrupt ID	1 - 25	Choose the interrupt ID. The defaults are: User Timer 0: 19 User Timer 1: 20 User Timer 2: 21

Table 12: Sapphire Base Address Tab Parameters

Parameter	Options	Description
Address Assignment Method	AUTO, Manual	AUTO: Automatically assign an address to the enabled peripherals. Manual: The user can assign addresses to the enabled peripherals.
External Memory Base Address	–	Displays the base address. You cannot change it.
AXI Slave Base Address	–	Displays the base address when the **Address Assignment Method** is set to AUTO.
Peripheral and IO Base Address	–	When the **Address Assignment Method** is **Manual**, enter the base address value. The wizard automatically rounds the value to 16 MB aligned addresses during IP generation. For example, 0x41234567 is rounded to 0x41000000.
UART*n* Address Offset	–	Displays the base address when the **Address Assignment Method** is set to AUTO.
SPI*n* Address Offset	–	When the **Address Assignment Method** is **Manual**, enter base address value. The wizard automatically rounds the value to 4 KB aligned addresses during IP generation. For example, 0x4123 is rounded to 0x41000.
I2C*n* Address Offset	–	
GPIO*n* Address Offset	–	
User Timer*n* Address Offset	–	
APB3*n* Address Offset	–	Displays the base address when the **Address Assignment Method** is set to AUTO. When the **Address Assignment Method** is **Manual**, enter base address value. The wizard automatically rounds the value to APB sized aligned addresses during IP generation. For example, if the APB size is 64 KB, 0x23456 is rounded to 0x20000.
On-Chip RAM Base Address	–	Displays the base address. You cannot change it.

Modify the Bootloader

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

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

Modify the Bootloader Software with External Memory Enabled

First you need to modify the bootloader code:

1. Open the **bootloaderConfig.h** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/ EfxSapphireSoc/app** directory.
2. Change the `#define USER_SOFTWARE_SIZE` parameter for the new on-chip RAM size and save.
3. In Eclipse, create a new project from the makefile in the **embedded_sw/**<*SoC module*>**/software/standalone/bootloader** directory and compile it.

Modify the Bootloader Software without External Memory Enabled

First, you need to modify the bootloader linker script:

1. Open the **bootloader.ld** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/ EfxSapphireSoc/linker** directory.

2. Replace the MEMORY and PHDRS code with the following code. The <*bootloader_address*> should be 0xF9000000 + (<*memory size*>-1024), where <*memory size*> is your SoC's on-chip RAM size.

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

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

Second you need to modify the bootloader code:

1. Open the **bootloaderConfig** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/ EfxSapphireSoc/app** directory.

2. Change the #define USER_SOFTWARE_SIZE parameter for the new on-chip RAM size and save.

Third, in Eclipse, create a new project from the makefile in the **embedded_sw/**<*SoC module*>**/software/standalone/bootloader** directory and compile it.

Re-Generate the Memory Initialization Files

Next, you need to re-generate the memory initialization files using the **binGen.py** helper script. You find this script in the <*project*>**/embedded_sw/**<*SoC module*>**/tool** directory.

Windows:

Open a command prompt and type these commands:

```
${EFINITY_HOME}/bin/setup.bat
python3 binGen.py -b bootloader.bin -s <RAM size> -f <FPU>
```

Linux:

Open a terminal and type these commands:

```
source
python3 binGen.py -b bootloader.bin -s <RAM size> -f <FPU>
```

where:

- <*RAM size*> is the on-chip RAM size you want to use.
- <*FPU*> indicates whether the floating-point unit is enabled for the SoC. 1: floating-point is enabled, 0: disabled.

This command generates the new memory initialization files. Copy these files into the same directory as your project **.xml** file, replacing the existing files.

Compile your design.

Chapter 3

Program the Board with the Sapphire RTL Design

Contents:

- **About the Example Design**
- **Enable the On-Board 10 MHz Oscillator**
- **Installing USB Drivers**
- **Program the Development Board**

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

About the Example Design

This example targets Trion and 钛金系列 development boards:

- *Trion® T120 BGA324 Development Board*—The RTL design files are in the **T120F324_devkit** directory.
- 钛金系列 *Ti60 F225 Development Board*—The RTL design files are in the **Ti60F225_devkit** directory.

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

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

(i) **Note:** The following description is for the example design using the default settings.

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

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

The Sapphire SoC is configured for:

- 100 MHz frequency
- External memory interface is enabled with a width of 128 and size of 3.5 GB

- Caches are enabled with a size of 4 KB
- 4KB on-chip RAM size
- Soft Debug Tap is disabled
- UART 0 is enabled
- SPI 0 is enabled
- I2C 0 is enabled
- GPIO 0 is enabled
- APB3 0 is enabled
- AXI4 Slave is enabled
- AXI Master 0 is enabled
- User interrupt 0 is enabled

Figure 2: Example Design Block Diagram
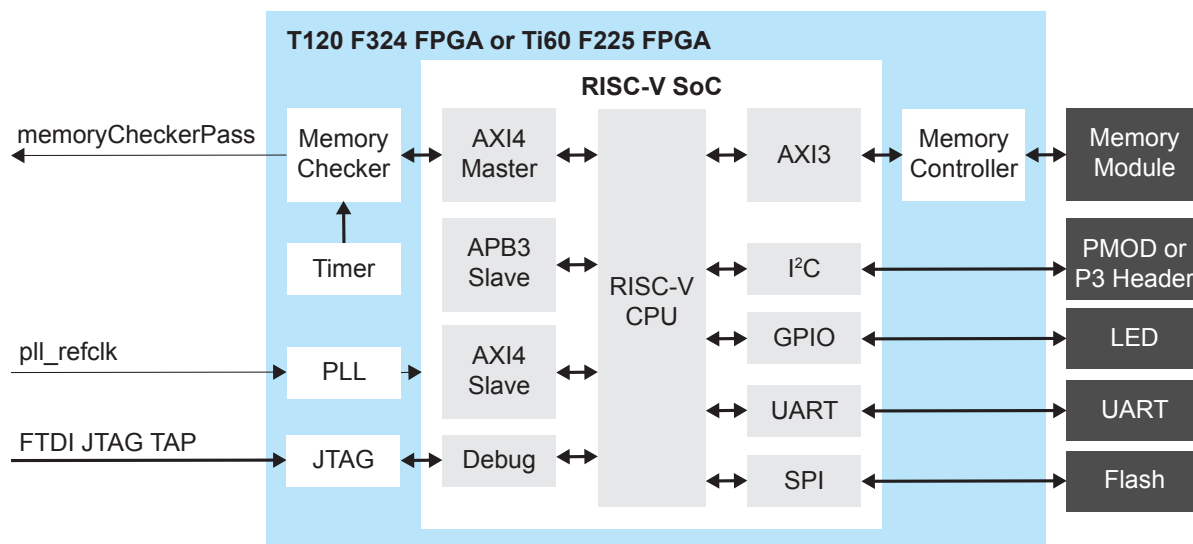
Table 13: Example Design Implementation

FPGA	Logic + Adders	Flipflops	Multipliers or DSP Blocks	Memory Blocks	f_{MAX} (MHz)	Language	Efinity Version
T120 BGA324 C4	8,013	8,506	4	69	105	Verilog HDL	2021.2
Ti60 F225 C4	10,438	9,490	4	81	148	Verilog HDL	2021.2

Enable the On-Board 10 MHz Oscillator

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

Figure 3: Connect Pins 2 and 3 on J10
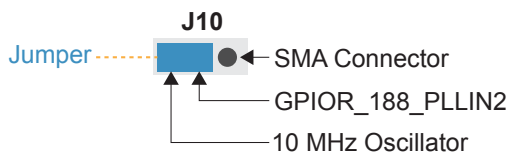
Installing USB Drivers

易灵思 development boards have FTDI chips (FT232H, FT2232H, or FT4232H) to communicate with the USB port and other interfaces such as SPI, JTAG, or UART. These chips have separate channels for each interface. If you install a driver for each interface, each interface appears as a unique FTDI device. If you install a composite driver, all of the separate interfaces appear as a single composite device.

- *Trion® T120 BGA324 Development Board*—Install a composite driver for this board.
- 钛金系列 *Ti60 F225 Development Board*—Install separate drivers for interfaces 0 and 1.

When working with OpenOCD on Windows, you need to install the **libusbK** driver as described in the following sections.

> **(i)** **Note:** If you already installed the **libusb-win32** driver and want to use OpenOCD, uninstall **libusb-win32** and install **libusbK** instead.

Installing Drivers on Windows (钛金系列 Ti60 F225 Development Board)

On Windows, you use software from Zadig to install drivers (**zadig.akeo.ie**). In the Zadig software, the interface names end with *(Interface N)*, where *N* is the channel number.

To install the interface drivers in Windows:

1. Connect the FTDI USB (J12) on the board to your computer with a USB type-C cable.
2. Download the Zadig software from **zadig.akeo.ie**. (You do not need to install it; simply run the downloaded executable.)
3. Run the Zadig software.

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

4. Choose **Options > List All Devices**.
5. Repeat the following steps for each interface. The interface names end with *(Interface N)*, where *N* is the channel number.
   - Select **libusb-win32** or **libusbK** in the **Driver** drop-down list. (Do **not** choose WinUSB.)
   - Click **Replace Driver**.
6. Close the Zadig software.

> **(!)** **Important: Install drivers for interfaces 0 and 1 only.** You do not need to install drivers for interfaces 2 and 3 because when you connect the 钛金系列 Ti60 F225 Development Board to your computer, Windows automatically installs a driver for them.

Installing Drivers on Windows (Trion® T120 BGA324 Development Board)

You use the Zadig software to install the driver, but instead of installing for separate interfaces, you install a composite driver.

1. Connect the board to your computer with the appropriate cable and power it up.
2. Download the Zadig software from **zadig.akeo.ie**. (You do not need to install it; simply run the downloaded executable.)
3. Run the Zadig software.

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

4. Choose **Options > List All Devices**.
5. Turn off **Options > Ignore Hubs or Composite Parents**.
6. Select the Trion® T20 BGA256 Development Kit.
7. Choose **libusbK** in the **Driver** drop-down list.
8. Click **Replace Driver**.
9. Close the Zadig software.

Installing Drivers on Linux (All Kits)

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

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

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

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

Program the Development Board

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

Table 14: Available Example Designs

Board	Location
钛金系列 Ti60 F225 Development Board	Ti60F225_devkit
Trion® T120 BGA324 Development Board	T120F324_devkit

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

Connect the board to your computer using a USB cable.

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

Chapter 4

Simulate

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

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

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

A successful simulation returns the following messages

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

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

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

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

```
#  0 ----------------------------------------
#  0 [EFX_INFO]: Executing custom binary file...
#  0 [EFX_WARN]: Driver sequences are not included in this simulation
#  0 [EFX_WARN]: Monitor sequences are not included in this simulation
#  0 ----------------------------------------
```

You need to develop your own sequence for your application.

Launch Eclipse

Contents:

- **Set Global Environment Variables**

The RISC-V SDK includes the **run_eclipse.bat** file (Windows) or **run_eclipse.sh** file (Linux) that adds executables to your path, sets up envonment variables for the Sapphire BSP, and launches Eclipse. Always use this executable to launch Eclipse; do not launch Eclipse directly.

When you first start working with the Sapphire SoC, you need to configure your Eclipse workspace and environment. Setting up a global development environment for your workspace means you can store all of your Sapphire software code in the same place and you can set global environment variables that apply to all software projects in your workspace. You should use a unique workspace for your Sapphire SoC projects. 易灵思 recommends using the **embedded_sw/<SoC module>** directory as the workspace directory.

ⓘ **Note:** With IP Manager, you can generate multiple SoCs with different options. Using the **embedded_sw/<SoC module>** directory as your workspace means that you can explore more than one SoC by simply switching workspaces.

Follow these steps to launch Eclipse and set up your workspace:

1. Launch Eclipse using the **run_eclipse.bat** file (Windows) or **run_eclipse.sh** file.
2. The launch script prompts you to select your SoC. Type 1 for Sapphire and press enter.
3. If this is the first time you are running Eclipse, create a new workspace that points to the **embedded_sw/<SoC module>** directory. Otherwise, choose **File > Switch Workspace > Other** to choose an existing workspace directory and click **Launch**.

Set Global Environment Variables

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

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

Choose **Window > Preferences** to open the **Preferences** window and perform the following steps.
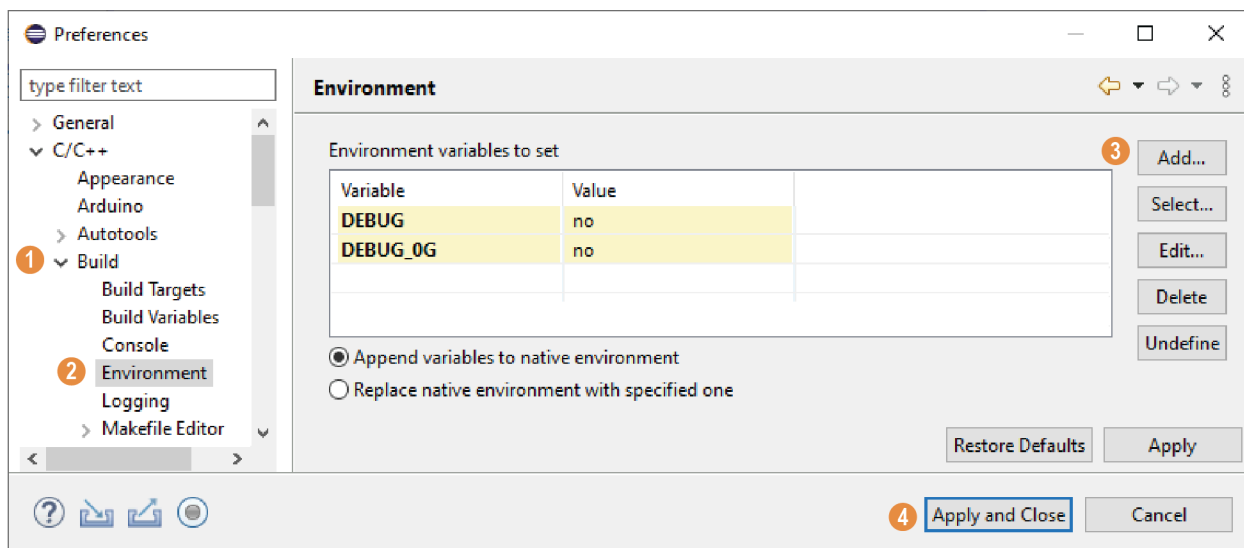
1. In the left navigation menu, expand **C/C++ > Build**.
2. Click **C/C++ > Build > Environment**.
3. Click **Add** and add the following environment variables:

Variable	Value	Description
DEBUG	no	Enables or disables debug mode. no: Debugging is turned off yes: Debugging is enabled
DEBUG_OG	no	Enables or disables optimization during debugging. Use an uppercase letter O not a zero.

4. Click **Apply and Close**.

Chapter 6

Create and Build a Software Project

Contents:

- **Create a New Project**
- **Import Project Settings (Optional)**
- **Enable Debugging**
- **Build**

After you set up your Eclipse workspace, you are ready to create a new project and build it. These instructions walk you through the process using the **axiDemo** example project from the **software** directory.

Create a New Project

In this step you create a new project from the **axiDemo** code example.

1. Launch Eclipse.
2. Select the Sapphire workspace if it is not open by default.
3. Make sure you are in the C/C++ perspective.

Import the **axiDemo** example:

4. Choose **File > New > Makefile Project with Existing Code**.
5. Click **Browse** next to **Existing Code Location**.
6. Browse to the **software/standalone/axiDemo** directory and click **Select Folder**.
7. Select **<none>** in the **Toolchain for Indexer Settings** box.
8. Click **Finish**.

Import Project Settings (Optional)

易灵思 provides a C/C++ project settings file that defines the include paths and symbols for the C code. Importing these settings into your project lets you explore and jump through the code easily.

> ⓘ **Note:** You are not required to import the project settings to build. These settings simply make it easier for you to write and debug code.

To import the settings:

1. Choose **File > Import** to open the **Import** wizard.
2. Expand **C/C++**.
3. Choose **C/C++ > C/C++ Project Settings**.
4. Click **Next**.
5. Click **Browse** next to the **Settings file** box.
6. Browse to one of the following files and click **Open**:

Option	Description
Windows	embedded_sw\<SoC module>\config\project_settings_soc.xml
Linux	embedded_sw/<SoC module>/config_linux/project_settings_soc_linux.xml

7. In the **Select Project** box, select the project name(s) for which you want to import the settings.

8. Click **Finish**.

Eclipse creates a new folder in your project named **Includes**, which contains all of the files the project uses.

After you import the settings, clean your project (**Project > Clean**) and then build (**Project > Build Project**). The build process indexes all of the files so they are linked in your project.

Enable Debugging

If you chose **OpenOCD Debug Mode > Turn On by default** when you configured the SoC, debugging is turned on and you can skip the instructions in this topic.

If you chose **OpenOCD Debug Mode > Turn Off by default** when you configured the SoC, debugging is turned off. Add the environment variables as described in Set Global Environment Variables on page 22 and then change the variables as needed.

*   To run the program for normal operation, keep **DEBUG** set to **no**.
*   To debug with the OpenOCD debugger, set **DEBUG** to **yes**.

In debug mode, the program suspends operation after loading so that you can set breakpoints or perform debug tasks.

To change the debug settings for your project, right-click the project name **axiDemo** in the Project Explorer and choose **Properties** from the pop-up menu.

1. Expand **C/C++ Build**.
2. Click **C/C++ Build > Environment**.
3. Click the **Debug** variable.
4. Click **Edit**.
5. Change the **Value** to yes.
6. Click **OK**.
7. Click **Apply and Close**.

> ⚠ **Important:** When you change the debug value for a project you previously built, you must clean the project (**Project** > **Clean**) before building again. Otherwise, Eclipse gives a message in the Console that there is `Nothing to be done for 'all'`

Build

Choose **Project > Build Project** or click the Build Project toolbar button.

The **makefile** builds the project and generates these files in the **build** directory:
*   **axiDemo.asm**—Assembly language file for the firmware.
*   **axiDemo.bin**—Download this file to the flash device on your board using OpenOCD. When you turn the board on, the SoC loads the application into the RISC-V processor and executes it.

- **axiDemo.elf**—Use this file when debugging with the OpenOCD debugger.
- **axiDemo.hex**—Hex file for the firmware. (Do not use it to program the FPGA.)
- **axiDemo.map**—Contains the SoC address map.

Chapter 7

Debug with the OpenOCD Debugger

Contents:

- **Import the Debug Configuration**
- **Debug**

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

Import the Debug Configuration

To simplify the debugging steps, the Sapphire SoC includes debug configurations that you import. There are several configuration files, depending on which board you use.

Table 15: Debug Configurations

Debug Configuration	Use for
default	Debugging software on Trion® development boards.
default_ti	Debugging software on 钛金系列 development boards.
default_softTap	Debugging software on Trion or 钛金系列 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 49.)

To import a debug configuration and use it to launch a debug session:

1. Launch Eclipse by running the **run_eclipse.bat** file (Windows) or **run_eclipse.sh** (Linux).
2. Select a workspace (if you have not set one as a default).
3. Open the **axiDemo** project or select it under **C/C++ Projects**.
4. Right-click the **axiDemo** project name and choose **Import**.
5. In the Import dialog box, choose **Run/Debug > Launch Configurations**.
6. Click **Next**. The Import Launch Configurations dialog box opens.
7. Browse to the following directory and click **OK**:

Option	Description
Windows	embedded_sw\\<SoC module>\config
Linux	embedded_sw/<SoC module>/config_linux

8. Check the box next to **config** (Windows) or **config_linux** (Linux).
9. Click **Finish**.
10. Right-click the **axiDemo** project name and choose **Debug As > Debug Configurations**.
11. Enter `axiDemo` in the **Project** box.
12. Enter `build\axiDemo.elf` in the **C/C++ Application** box.

13. *Windows only:* you need to change the path to the **cpu0.yaml** file:

    a. Click the **Debugger** tab.

    b. In the **Config options** box, change `${workspace_loc}` to the full path to the **<SoC module>** directory.

> ⓘ **Note:** For the **cpu0.yaml** path, make sure to use **** as the directory separator because the first slash escapes the second one. For example, use:
>
> **c:\\Efinity\\2021.2\\project\\<project name>\\embedded_sw\\<SoC module> \\cpu0.yaml**

14. Click **Debug**.

> ⓘ **Note:** If Eclipse prompts you to switch to the Debug Perspective, click **Switch**.

Debug

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

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

The axiDemo example blinks the LEDs and prints messages on a UART terminal. Refer to **Using a UART Module** on page 46 for steps on setting it up.

Figure 4: Perform Debugging
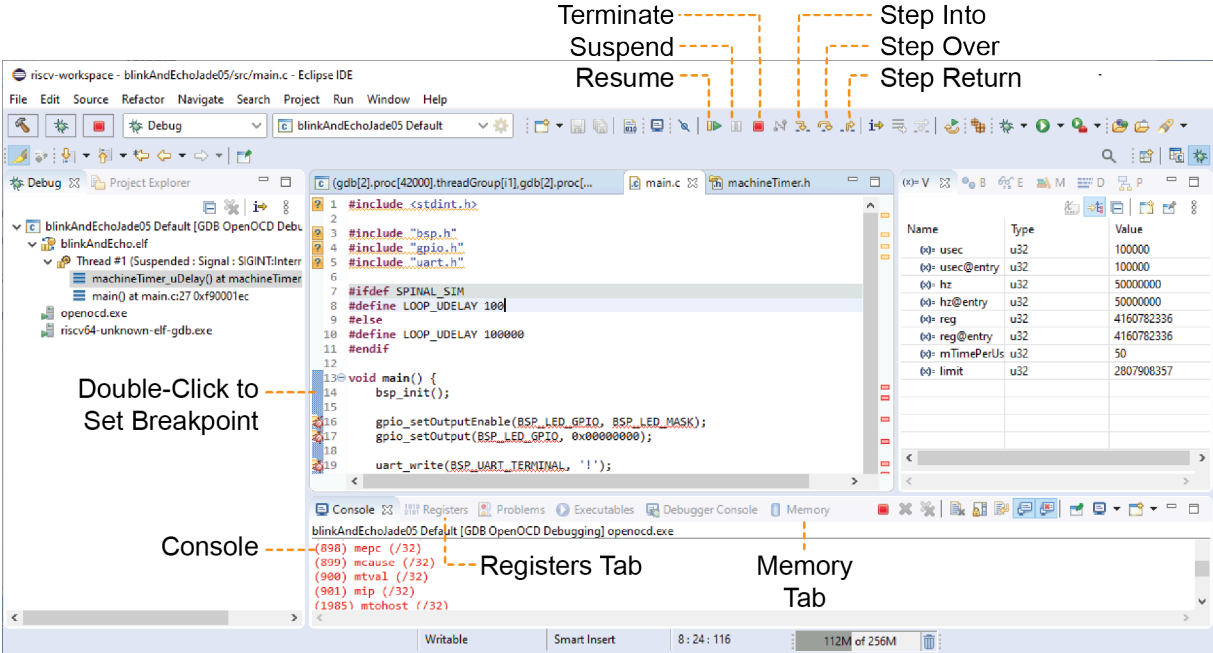
> **Learn more:** For more information on debugging with Eclipse, refer to **Running and debugging projects** in the Eclipse documentation.

Chapter 8

Create Your Own RTL Design

Contents:

- **Target another FPGA**
- **Target another 易灵思 Board**
- **Target Your Own Board**
- **Create a Custom AXI4 Slave Peripheral**
- **Create a Custom APB3 Peripheral**
- **Use another DDR DRAM Module (Trion Only)**
- **Use the I2C Interface for DDR Calibration**
- **Remove Unused Peripherals from the RTL Design**

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

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

Target another FPGA

To change the design to target a different FPGA:

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

Target another 易灵思 Board

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

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

Table 16: Provided FTDI Configuration Files

File	Use for
ftdi.cfg	Trion development board
ftdi_ti.cfg	钛金系列 Ti60 F225 Development Board

If you do not want to re-generate the SoC, you can also change the target 易灵思 board manually by editing the **.cfg** file. However, if you want to target your own board, refer to **Target Your Own Board** on page 31 because the 易灵思 drivers specifically target the FTDI chips used on 易灵思 boards and your board will probably not have that chip. To target a different 易灵思 development board manually, follow these steps with the development board attached to the computer:

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

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

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

Target Your Own Board

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

Using the FTDI C232HM-DDHSL-0 JTAG cable

The Sapphire SoC also includes a configuration file for the FTDI C232HM-DDHSL-0 JTAG cable (**c232hm_ddhsl_0.cfg**), which bridges between your computer's USB connector and the JTAG signals on the FPGA. If you use this cable to connect your board to your computer, you can simply use this configuration file instead of **ftdi.cfg** or **ftdi_ti.cfg**

Note: Refer to **Connect the FTDI Cable** on page 50 for instructions on using the cable.

Using another JTAG Cable or Module

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

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

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

You can also write your own configuration file if desired.

Follow these instructions when debugging with your own board:

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

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

Using an FTDI Chip on your Board

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

Create a Custom AXI4 Slave Peripheral

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

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

Create a Custom APB3 Peripheral

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

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

Use another DDR DRAM Module (Trion Only)

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

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

Use the I^{2}C Interface for DDR Calibration

You can use the I^{2}C interface to calibrate and reset the DDR DRAM interface on the Trion® T120 BGA324 Development Board or Trion® T120 BGA576 Development Board. If you want to use calibration:

1. In the Efinity Interface Designer, select the DDR block and turn on **Enable Control** in the Block Editor's **Control** tab. Save.

2. In your RTL design, connect the I^{2}C interface to the DDR block's I^{2}C interface. See the following example code:

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

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

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

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

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

Remove Unused Peripherals from the RTL Design

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

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

To disable I^{2}C 0, remove the signal name in () as shown below:

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

Create Your Own Software

Contents:

- **Deploying an Application Binary**
- **About the Board Specific Package**
- **Address Map**
- **Example Software**

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

Note: The Sapphire SoC does not currently support floating point calculations, such as sine and cosine.

Deploying an Application Binary

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

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

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

Boot from a Flash Device

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

To boot from a SPI flash device:

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

   Note: It takes ~300 ms to clone a 124 KByte user binary (this is the default size).

3. The Sapphire SoC jumps to logical address 0x0000_1000 to execute the user binary.

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 Eclipse and set up the debug environment for your project.
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 27 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 file, select it, and click **Open**.
7. Click the Add Image button a second time.
8. Browse to the RISC-V application binary file, select it, and click **Open**.
9. Specify the **Flash Address** as follows:

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

Figure 5: Combining a Bitstream and RISC-V Application Binary
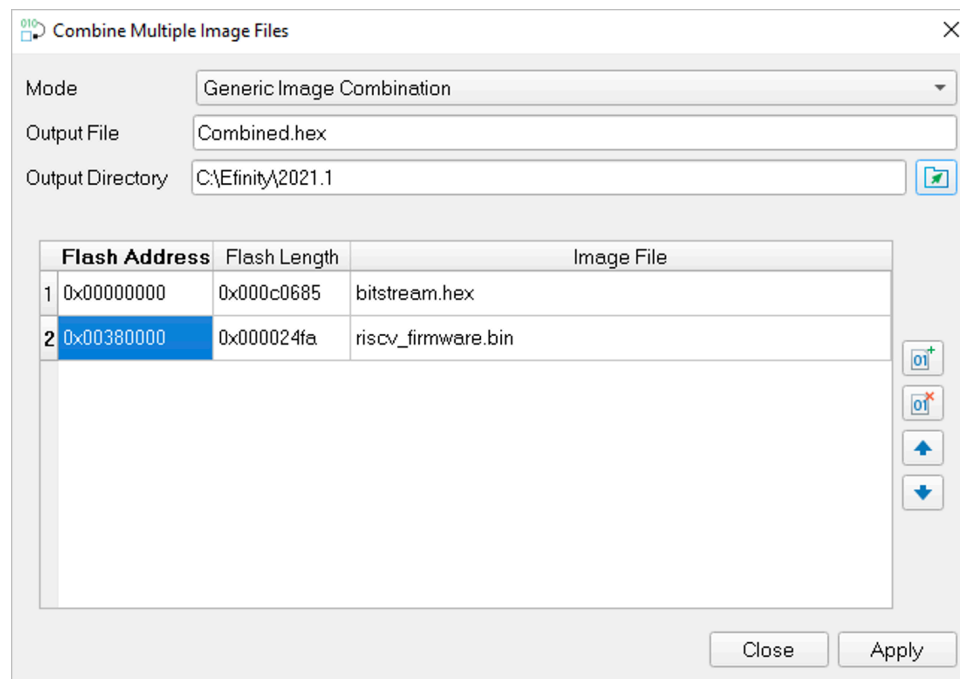
10. Click **Apply**. The software creates the combined **.hex** file in the specified **Output Directory** (the default is the project **outflow** directory).
11. Program the flash with the **.hex** file using **Programming Mode > SPI Active**.
12. Reset the FPGA or power cycle the board.

Copy a User Binary to the Flash Device (2 Terminals)

To boot from a flash device, you need to copy the binary to the device. These instructions describe how to use two command prompts or terminals to flash the user binary file.

> **Note:** If you want to store the binary in the same flash device that holds the FPGA bitstream, refer to **Copy a User Binary to Flash (Efinity Programmer)** on page 35 instead.

You use two command prompts or terminals:
- The first terminal opens an OpenOCD connection to the SoC.
- The second connects to the first terminal to write to the flash.

> **Important:** If you are using the OpenOCD debugger in Eclipse, terminate any debug processes before attempting to flash the memory.

Set Up Terminal 1

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

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

```
    -f bsp\efinix\EfxSapphireSoc\openocd\flash.cfg
```

Linux:

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

The OpenOCD server connects and begins listening on port 4444.

Set Up Terminal 2

1. Open a second command prompt or shell.
2. Enable telnet if it is not turned on. **Turn on telnet (Windows)**
3. Open a telnet local host on port 4444 with the command `telnet localhost 4444`.
4. In the OpenOCD shell or command prompt, use the following command to flash the user binary file:

```
flash write_image erase unlock <path>/<filename>.bin 0x380000
```

Where *<path>* is the full, absolute path to the **.bin** file.

> **(i)** **Note:** For Windows, use \\ as the directory separators.

Close Terminals

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

> **(!)** **Important:** OpenOCD cannot be running in Eclipse when you are using it in a terminal. If you try to run both at the same time, the application will crash or hang. Always close the terminals when you are done flashing the binary.

Reset the FPGA

Press the reset button (SW2) on the development board.

About the Board Specific Package

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

Table 17: BSP Files

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

Address Map

ℹ️ **Note:** Because the address range might be updated, 易灵思 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 **soc.h**.

Table 18: Default Address Map, Interrupt ID, and Cached Channels

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

Device	Parameter	Size	Interrupt ID	Region
Off-chip memory	SYSTEM_DDR_BMB	4 MB to 3.5 GB	–	Cache
GPIO 0	SYSTEM_GPIO_0_IO_CTRL	4K	[0]: 12 [1]: 13	I/O
GPIO 1	SYSTEM_GPIO_1_IO_CTRL	4K	[0]: 14 [1]: 15	I/O
I^{2}C 0	SYSTEM_I2C_0_IO_CTRL	4K	8	I/O
I^{2}C 1	SYSTEM_I2C_1_IO_CTRL	4K	9	I/O
I^{2}C 2	SYSTEM_I2C_2_IO_CTRL	4K	10	I/O
Core timer	SYSTEM_CLINT_CTRL	4K	–	I/O
PLIC	SYSTEM_PLIC_CTRL	4K	–	I/O
SPI master 0	SYSTEM_SPI_0_IO_CTRL	4K	4	I/O
SPI master 1	SYSTEM_SPI_1_IO_CTRL	4K	5	I/O
SPI master 2	SYSTEM_SPI_2_IO_CTRL	4K	6	I/O
UART 0	SYSTEM_UART_0_IO_CTRL	4K	1	I/O
UART 1	SYSTEM_UART_1_IO_CTRL	4K	2	I/O
UART 2	SYSTEM_UART_2_IO_CTRL	4K	3	I/O
User timer 0	SYSTEM_USER_TIMER_0_CTRL	4K	19	I/O
User timer 1	SYSTEM_USER_TIMER_1_CTRL	4K	20	I/O
User timer 2	SYSTEM_USER_TIMER_2_CTRL	4K	21	I/O
User peripheral 0	IO_APB_SLAVE_0_CTRL	4K to 1 MB	–	I/O
User peripheral 1	IO_APB_SLAVE_1_CTRL	4K to 1 MB	–	I/O
User peripheral 2	IO_APB_SLAVE_2_CTRL	4K to 1 MB	–	I/O
User peripheral 3	IO_APB_SLAVE_3_CTRL	4K to 1 MB	–	I/O
User peripheral 4	IO_APB_SLAVE_4_CTRL	4K to 1 MB	–	I/O
On-chip BRAM	SYSTEM_RAM_A_BMB	4 - 512 KB	–	Cache
AXI user slave	SYSTEM_AXI_A_BMB	1 KB to 256 MB	–	I/O
External interrupt	–	–	[0]: 16 [1]: 17	I/O

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

Example Software

To help you get started writing software for the Sapphire, 易灵思 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 an Eclipse terminal and connect to the UART module.**

Table 19: Example Software Code

Directory	Description
apb3Demo	This example shows how to implement an ABP3 slave.
Axi4Demo	This example illustrates how to implement a user AXI4 slave.
bootloader	This software is the bootloader for the system.
common	Provides linking for the makefiles.
compatibilityDemo	This example shows how to migrate a software application from the Sapphire SoC v1.1 to v2.0 or higher.
coreTimerInterruptDemo	This example shows how to use the clint timer with interrupt.
customInstructionDemo	This example illustrates how to implement a custom instruction.
dhrystone	This example is a synthetic computing benchmark program.
driver	This directory contains the system drivers for the peripherals (I^2C, UART, SPI, etc.). Refer to **API Reference** on page 59 for details.
fpuDemo	This example shows how to use the floating-point unit.
gpioDemo	This example shows how to control the GPIO and its interrupt.
i2cDemo	This example shows how to connect to an MCP4725 digital-to-analog converter (DAC) using an I^2C peripheral.
i2cSlaveDemo	This example illustrates how an I^2C slave communicates with the master.
memTest	This code performs a memory address and data test.
spiReadFlash	This example shows how to read from a SPI flash device.
spiWriteFlash	This example shows how to write to a SPI flash device.
spiDemo	This code reads the device ID and JEDEC ID of a SPI flash device and echoes the characters on a UART.
uartEchoDemo	This example shows how to use the UART.
uartInterruptDemo	This exmple shows how to use a UART interrupt.
userInterruptDemo	This example demonstrates user interrupts with UART messages.
userTimerDemo	This example shows how to use the user timer with interrupt.

Axi4Demo Design

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

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

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

apb3Demo

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

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

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

compatibilityDemo

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

This demo runs a machine timer with interrupt, which was available in v1.1. See **Migrating to the Sapphire SoC v2.0** on page 52 for additional migration details.

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

coreTimerInterruptDemo

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

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

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

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

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

Passed!
```

dhrystone Example

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

The following code shows example results:

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

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

fpuDemo

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

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

```
fpu math demo !
rv32i (base-extension) is capable to perform floating-point calculation but
 rv32i requires
```

```
more processing time and instruction to calculate the result enable fpu with
 d-extension
will sharply improve processing time and decrease app binary size

sine processing clock cycles:879

cosine processing clock cycles:864

tangent processing clock cycles:1148

square root processing clock cycles:2171

division processing clock cycles:377


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

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

gpioDemo

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

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

i2cDemo Example

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

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

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

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

i2cSlaveDemo Design

This example illustrates how an I^{2}C slave communicates with the master. It uses a 16-bit address and 16-bit data register for read and write. The slave is ready to access the master after the `Init Done` message displays on the UART.

memTest Example

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

```
Memory test
Passed
```

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

spiReadFlashDemo Example

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

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

spiWriteFlashDemo Example

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

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

spiDemo Example

The SPI example (**spiDemo** directory) provides example code for reading the device ID and JEDEC ID of the SPI flash device on the development board.
*   The default base address map of the SPI flash master is 0xF801_4000.
*   The default SCK frequency is half of the SoC system clock frequency.
*   The default base address of the UART is 0xF801_0000 with a default baud rate of 115200.

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

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

uartEchoDemo

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

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

UartInterruptDemo Example

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

The application displays messages on a UART terminal:

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

userInterruptDemo Example

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

The application displays the messages on a UART terminal:

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

userTimerDemo

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

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

FreeRTOS Examples

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

Download the FreeRTOS

The freeRTOS examples require you to download FreeRTOS.

1. Download the FreeRTOS zip file from **https://www.freertos.org**.
2. Unzip the files into the **embedded_sw/<SoC module>/software** directory.

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

freertosDemo

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

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

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

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

freertosDemo2

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

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

freertosUartInterruptDemo Example

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

The application displays messages on a UART terminal:

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

Using a UART Module

Contents:

- **Using the On-board UART (**钛金系列**)**
- **Set Up a USB-to-UART Module (Trion)**
- **Open a Terminal**
- **Enable Telnet on Windows**

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

Using the On-board UART (钛金系列)

The 钛金系列 Ti60 F225 Development Board has a USB-to-UART converter connected to the Ti60's GPIOL_01 and GPIOL_02 pins. To use the UART, simply connect a USB cable to the FTDI USB connector on the 钛金系列 Ti60 F225 Development Board and to your computer.

> ⓘ **Note:** The board has an FTDI chip to bridge communication from the USB connector. FTDI interface 2 communicates with the on-board UART. You do not need to install a driver for this interface because when you connect the 钛金系列 Ti60 F225 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 4 devices listed as USB Serial Port (COMn) where n is the assigned port number. Note the COM number for the third device; that is the UART.

Finding the COM Port (Linux)

In a terminal, type the command:

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

The terminal displays a list of attached devices.

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

The UART is `/dev/ttyUSB2`.

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

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

Figure 6: Connect the UART Module to PMOD Connector J12
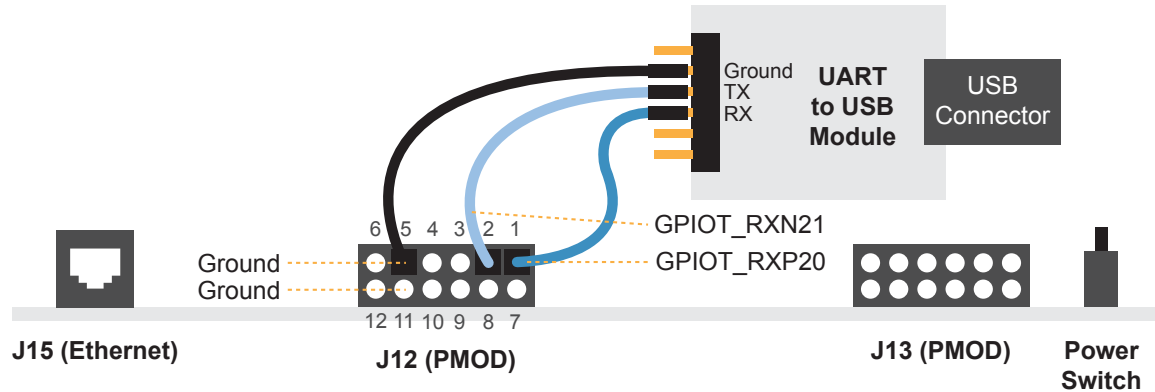
1. Connect the UART module to the PMOD port J12
   - *RX*—GPIOT_RXP20, which is pin 1 on PMOD J12
   - *TX*—GPIOT_RXN21, which is pin 2 on PMOD J12
   - *Ground*—Use ground pin 5 or 11 on PMOD J12.
2. Plug the UART module into a USB port on your computer. The driver should install automatically if needed.

Finding the COM Port (Windows)

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

Finding the COM Port (Linux)

In a terminal, type the command:

```
dmesg | grep ttyUSB
```

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

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

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

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

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

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

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

Open a Terminal

You can use any terminal program, such as Putty, termite, or the built-in Eclipse terminal, to connect to the UART. These instructions explain how to use the Eclipse terminal; the others are similar.

1. In Eclipse, choose **Window > Show View > Terminal**. The Terminal tab opens.
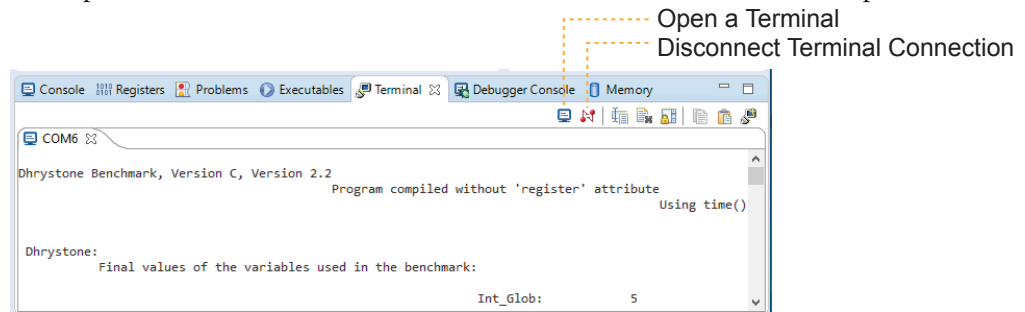
2. Click the Open a Terminal button.
3. In the **Launch Terminal** dialog box, enter these settings:

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 11

Using a Soft JTAG Core for Example Designs

Contents:

- **Connect the FTDI Cable**

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

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

Note: 易灵思 recommends you use the C232HM-DDHSL-0 FTDI chip cable rather than a JTAG mini-module because the software generated by the IP Manager includes the debug configuration file for the cable.

Connect the FTDI Cable

When you turn on the Enable Soft JTAG TAP option in the IP Configuration wizard, the example design assigns the JTAG pins to resources in the interface design. Use the following figures to connect the table to the JTAG pins.

Note: If you have not already done so, install the driver for the FTDI cable as described in **Installing USB Drivers** on page 19.

Connecting to the 钛金系列 Ti60 F225 Development Board

Figure 7: Connecting the C232HM-DDHSL-0 Cable
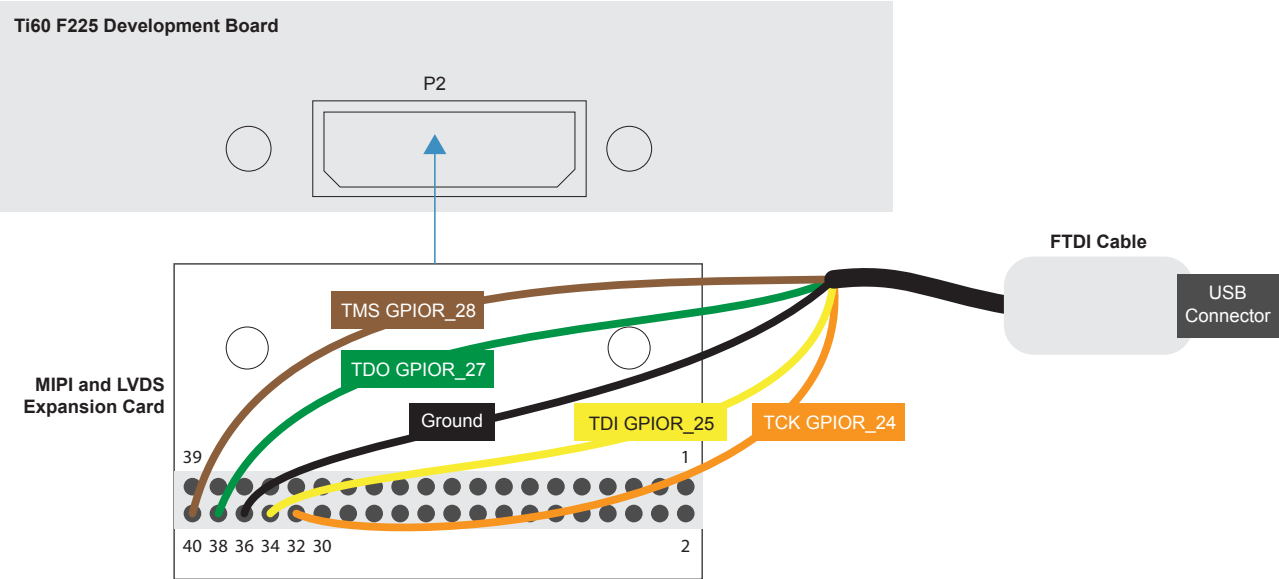
Note: If you have not already done so, install the driver for the FTDI cable as described in **Installing USB Drivers** on page 19.

For the Trion® T120 BGA324 Development Board, connect these pins:

Table 20: FDTI to PMOD Connections

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

Connecting to the Trion® T120 BGA324 Development Board

Figure 8: Connecting the C232HM-DDHSL-0 Cable
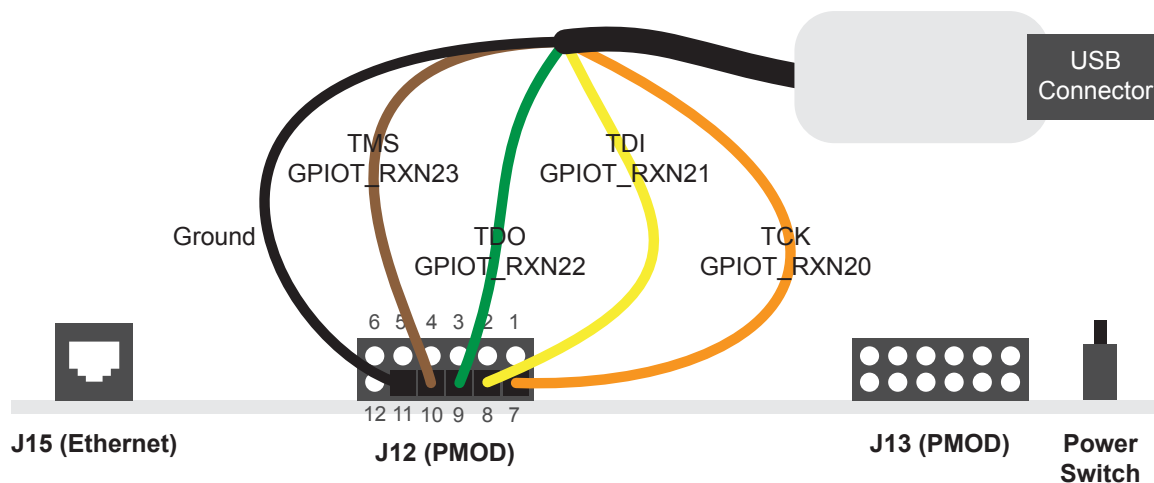
> **Note:** If you have not already done so, install the driver for the FTDI cable as described in **Installing USB Drivers** on page 19.

For the Trion® T120 BGA324 Development Board, connect these pins:

Table 21: FDTI to PMOD Connections

Port	Resource	PMOD (J12) Pin
TCK	GPIOT_RXN20	7
TDI	GPIOT_RXN21	8
TDO	GPIOT_RXN22	9
TMS	GPIOT_RXN23	10
GND	–	5 or 1

Debugging in Eclipse

1. Open your Eclipse project.
2. Run or debug the software with the OpenOCD debugger using the **default_softTap** launch configuration.
3. Refer to **Debug with the OpenOCD Debugger** on page 27 for complete instructions.
4. Open the Debugger to perform hardware debugging.

Chapter 12

Migrating to the Sapphire SoC v2.0

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

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

IP Configuration Wizard

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

Table 22: IP Configuration Settings

Tab	Option	Setting	Notes
SOC	Peripheral Clock	DISABLE	In v1.*x*, the APB3 peripherals are driven by the system clock. In v2.0, set this option to DISABLE.
	Custom Instruction	DISABLE	Not supported in v1.*x*
	Linux Memory Management Unit	DISABLE	
	Floating-point Unit	DISABLE	
	Atomic extension	DISABLE	
Cache/ Memory	Data Cache Way	1	In v1.0, the SoC has a fixed I$ and D$ cache way (1 way) and size (4 KB).
	Data Cache Size	4 KB (v1.0) 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, or 32 KB (v1.1)	In v1.1, the wizard supports 1 ways and 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, or 32 KB
	Instruction Cache Way	1	
	Instruction Cache Size	4 KB (v1.0) 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, or 32 KB (v1.1)	
	External Memory AXI3 Interface	DISABLE (v1.0) ENABLE or DISABLE (v1.1)	In v1.*x*, an external memory interface is not supported with a cacheless CPU
	On-Chip RAM Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, or 512 KB	The v1.*x* SoC supports fewer sizes for On-Chip RAM. Choose one of these options in v2.0 for compatibility.

Tab	Option	Setting	Notes
Debug	Target OpenOCD	See v2.0 options	This option is not supported in v1.0. This option is the same in v1.1 and v2.0.
	Custom Target OpenOCD	See v2.0 options	This option is not supported in v1.0. This option is the same in v1.1 and v2.0.
	OpenOCD Debug Mode	Any	This option is not supported in v1.x. However, you can choose either option because it sets Eclipse environment variables and does not affect the SoC.

Debug Configuration

The **default_softTap** debug configuration file is updated in v2.0. Therefore, you cannot use the **default_softTap** generated with v1.1 with v2.0. If you are using v2.0, you need to remove the old **default_softTap** debug configuration from your Eclipse project and import the v2.0 one. See **Import the Debug Configuration** on page 27 for instructions.

Application Software

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

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

Note: compatibilityDemo on page 40 provides an example of how to use **compability.h** and **bsp.h**.

Chapter 13

Troubleshooting

Contents:

- Error 0x80010135: Path too long (Windows)
- OpenOCD Error: timed out while waiting for target halted
- Memory Test
- OpenOCD error code (-1073741515)
- OpenOCD Error: no device found
- OpenOCD Error: failed to reset FTDI device: LIBUSB_ERROR_IO
- OpenOCD Error: target 'fpga_spinal.cpu0' init failed
- Eclipse Fails to Launch with Exit Code 13
- Efinity Debugger Crashes when using OpenOCD
- Undefined Reference to 'cosf'

Error 0x80010135: Path too long (Windows)

When you unzip the SDK on Windows, you may get the error message:

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

Error 0x80010135: Path too long
```

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

OpenOCD Error: timed out while waiting for target halted

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

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

Memory Test

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

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

(!) Important: If you are using the OpenOCD debugger in Eclipse, terminate any debug processes before performing this test.

Set Up Terminal 1

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

   Windows:

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

   Linux:

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

   The OpenOCD server connects and begins listening on port 4444.

Set Up Terminal 2

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

Close Terminals

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

> ⚠ **Important:** OpenOCD cannot be running in Eclipse when you are using it in a terminal. If you try to run both at the same time, the application will crash or hang. Always close the terminals when you are done flashing the binary.

Reset the FPGA

Press the reset button (SW2) on the development board.

OpenOCD error code (-1073741515)

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

OpenOCD Error: no device found

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

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

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

1. Open the Efinity Programmer.
2. Click the **Refresh USB Targets** button to display the board name in the **USB Target** drop-down list.
3. Make note of the board name.
4. In a text editor, open the **ftdi.cfg** (Trion) or **ftdi_ti.cfg** (钛金系列) file in the **/bsp/efinix/EFXSapphireSoC/openocd** directory.

5. Change the `ftdi_device_desc` setting to match your board name. For example, use this code to change the name from Trion T20 Development Board to Trion T20 Developer Board:

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

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

OpenOCD Error: failed to reset FTDI device: LIBUSB_ERROR_IO

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

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

OpenOCD Error: target 'fpga_spinal.cpu0' init failed

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

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

This error occurs because the path to the **cpu0.yaml** file is incorrect, specifically the slashes for the directory separators. You should use:
- a single forward slash (/)
- 2 backslashes (\\)

For example, either of the following are good:

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

Eclipse Fails to Launch with Exit Code 13

The Eclipse software requires a 64-bit version of the Java JRE. If you use a 32-bit version, when you try to launch Eclipse you will get an error that Java quit with exit code 13.

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

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 49 for details.

Undefined Reference to 'cosf'

You may receive an error similar to this when using calculating square root, sine, or cosine with floating-point numbers in your application. The Sapphire SoC does not currently support floating point.

Chapter 14

API Reference

Contents:

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

Control and Status Registers

csr_clear()

Usage	`csr_clear(csr, val)`
Include	**driver/riscv.h**
Description	Clear a CSR.

csr_read()

Usage	`csr_read(csr)`
Include	**driver/riscv.h**
Description	Read from a CSR.
Example	`csrr (t0, mepc) // Write mepc in regfile[t0]`

csr_read_clear()

Usage	`csr_read_clear(csr, val)`
Include	**driver/riscv.h**
Description	CSR read and clear bit.

csr_read_set()

Usage	`csr_read_set(csr, val)`
Include	**driver/riscv.h**
Description	CSR read and set bit.

csr_set()

Usage	`csr_set(csr, val)`
Include	**driver/riscv.h**
Description	CSR set bit.

csr_swap()

Usage	`csr_write(csr, val)`
Include	**driver/riscv.h**
Description	Swaps values in the CSR.

csr_write()

Usage	`csr_write(csr, val)`
Include	**driver/riscv.h**
Description	Write to a CSR.
Example	`csrw (mepc, t0); // Write regfile[t0] in mepc`

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 struct of I^2C setting value
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'`

gpio_getFilteringStatus()

Usage	gpio_getFilteringStatus(reg)
Parameters	[IN] reg struct of I^{2}C setting value
Include	**driver/i2c.h**
Description	Read the 32-bit I^{2}C register filter hit 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
``` |

gpio_getInput()

Usage	gpio_getInput(GPIO_Reg, value)
Parameters	[IN] GPIO_Reg struct of GPIO register [IN] value GPIO pin bitwise
Include	**driver/gpio.h**
Description	Get input from a GPIO.

gpio_getInterruptFlag()

Usage	gpio_getInterruptFlag(reg)
Parameters	[IN] reg struct of I^{2}C setting value
Include	**driver/i2c.h**
Description	Read the 32-bit I^{2}C register interrupt flag with a call back function.
| Example | ```
Int flag = gpio_getInterruptFlag(I2C_CTRL) & I2C_INTERRUPT_DROP;
// Get Drop interrupt flag from Interrupt register
//[2] I2C_INTERRUPT_TX_DATA
//[3] I2C_INTERRUPT_TX_ACK
//[7] I2C_INTERRUPT_DROP
//[16] I2C_INTERRUPT_CLOCK_GEN_BUSY
//[17] I2C_INTERRUPT_FILTER
``` |

gpio_getMasterStatus()

Usage	gpio_getMasterStatus(reg)
Parameters	[IN] reg struct of I^{2}C setting value
Include	**driver/i2c.h**
Description	Read the 32-bit I^{2}C register master status with a call back function.
| Example | ```
int status = gpio_getMasterStatus(I2C_CTRL) & I2C_MASTER_BUSY;
// Get master busy status from status register
[0]I2C_MASTER_BUSY
[4]I2C_MASTER_START
[5]I2C_MASTER_STOP
[6]I2C_MASTER_DROP
``` |

gpio_getOutput()

Usage	`gpio_getOutput(GPIO_Reg, value)`
Parameters	[IN] GPIO_Reg struct of GPIO register
	[IN] value GPIO pin bitwise
Include	**driver/gpio.h**
Description	Read the output pin.

gpio_getOutputEnable()

Usage	`gpio_getOutputEnable(GPIO_Reg, value)`
Parameters	[IN] GPIO_Reg struct of GPIO register
	[IN] value GPIO pin bitwise
Include	**driver/gpio.h**
Description	Read GPIO output enable.

gpio_setOutput()

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

gpio_setOutputEnable()

Usage	`gpio_setOutputEnable(GPIO_Reg, value)`
Parameters	[IN] GPIO_Reg struct of GPIO register
	[IN] value GPIO pin bitwise
Include	**driver/gpio.h**
Description	Set GPIO as an output enable.

gpio_setInterruptRiseEnable()

Usage	`gpio_etInterruptRiseEnable(GPIO_Reg, value)`
Parameters	[IN] GPIO_Reg struct of GPIO register
	[IN] value GPIO pin bitwise
Include	**driver/gpio.h**
Description	Set an interrupt on the rising edge of the GPIO.

gpio_setInterruptFallEnable()

Usage	`gpio_setInterruptFallEnable(GPIO_Reg, value)`
Parameters	[IN] GPIO_Reg struct of GPIO register
	[IN] value GPIO pin bitwise
Include	**driver/gpio.h**
Description	Set an interrupt on the falling edge of the GPIO.

gpio_setInterruptHighEnable()

Usage	`gpio_setInterruptHighEnable(GPIO_Reg, value)`
Parameters	[IN] GPIO_Reg struct of GPIO register
	[IN] value GPIO pin bitwise
Include	**driver/gpio.h**
Description	Set an interrupt when the GPIO is high.

gpio_setInterruptLowEnable()

Usage	`gpio_setInterruptLowEnable(GPIO_Reg, value)`
Parameters	[IN] GPIO_Reg struct of GPIO register
	[IN] value GPIO pin bitwise
Include	**driver/gpio.h**
Description	Set an interrupt when the GPIO is low.

I^2C API Calls

i2c_applyConfig()

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

i2c_clearInterruptFlag()

Usage	`void i2c_clearInterruptFlag(u32 reg, u32 value)`
Parameters	[IN] `reg` struct of I^2C setting value
	[IN] `value` I^2C interrupt register
Include	**driver/i2c.h**
Description	Clear the I^2C interrupt flag.

i2c_disableInterrupt()

Usage	`void i2c_disableInterrupt(u32 reg, u32 value)`
Parameters	[IN] `reg` struct of I^{2}C setting value [IN] `value` I^{2}C 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
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` struct of I^{2}C setting value [IN] `value` I^{2}C 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
Include	**driver/i2c.h**
Description	Enable I^{2}C interrupt.

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

i2c_filterEnable()

Usage	`void i2c_filterEnable(u32 reg, u32 filterId, u32 config)`
Parameters	[IN] `reg` struct of I^{2}C setting value [IN] `filterID` filter configuration ID number [IN] `config` struct of I^{2}C configuration
Include	**driver/i2c.h**
Description	Enable the filter configuration.

i2c_listenAck()

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

i2c_masterBusy()

Usage	`void i2c_masterBusy(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Get the I^{2}C busy status.

i2c_masterStatus()

Usage	`int i2c_masterStatus(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Get the I^{2}C status.

i2c_masterDrop()

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

i2c_masterStart()

Usage	`void i2c_masterStart(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Change the I^{2}C master to the start status.

i2c_masterStartBlocking()

Usage	`void i2c_masterStartBlocking(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Asserts a start condition.

i2c_masterStop()

Usage	`void i2c_masterStop(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Change the I^{2}C master to the stop status.

i2c_masterStopBlocking()

Usage	`void i2c_masterStartBlocking(u32 reg)`
Parameters	[IN] `reg` struct of I²C register
Include	**driver/i2c.h**
Description	Asserts a stop condition.

i2c_masterStopWait()

Usage	`void i2c_masterStopWait(u32 reg)`
Parameters	[IN] `reg` struct of I²C register
Include	**driver/i2c.h**
Description	The stop condition is wait busy..

i2c_setFilterConfig()

Usage	`void i2c_setFilterConfig(u32 reg, u32 filterId, u32 config)`
Parameters	[IN] `reg` struct of I²C setting value [IN] `filterID` filter configuration ID number [IN] `config` struct of I²C configuration: • [9:0] I2C slave address • [14] I2C_FILTER_10BITS • [15] I2C_FILTER_ENABLE
Include	**driver/i2c.h**
Description	Set the filter configuration.
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` struct of I²C register
Include	**driver/i2c.h**
Description	Transmit acknowledge.

i2c_txAckBlocking()

Usage	`void i2c_txAckBlocking(u32 reg)`
Parameters	[IN] `reg` struct of I²C register
Include	**driver/i2c.h**
Description	Assert an ACK on the SDA pin.

i2c_txAckWait()

Usage	`void i2c_txAckWait(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Wait for an acknowledge to transmit.

i2c_txByte()

Usage	`void i2c_txByte(u32 reg, u8 byte)`
Parameters	[IN] `reg` struct of I^{2}C register [IN] `byte` 8 bits data to send out
Include	**driver/i2c.h**
Description	Transfers one byte to the I^{2}C slave.

i2c_txByteRepeat()

Usage	`void i2c_txByteRepeat(u32 reg, u8 byte)`
Parameters	[IN] `reg` struct of I^{2}C register [IN] `byte` 8 bits data to send out
Include	**driver/i2c.h**
Description	Send a byte and then wait until it is fully transmited on the I^{2}C bus.

i2c_txNack()

Usage	`void i2c_txNack(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Transfers a NACK.

i2c_txNackRepeat()

Usage	`void i2c_txNackRepeat(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Send a NACK and then wait until it is fully transmited on the I^{2}C bus.

i2c_txNackBlocking()

Usage	`void i2c_ txNackBlocking(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Assert a NACK on the SDA pin.

i2c_rxAck()

Usage	`int i2c_rxAck(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Returns	[OUT] 1 bit acknowledge
Include	**driver/i2c.h**
Description	Receive an acknowledge from the I^{2}C slave.

i2c_rxData()

Usage	`unit32_t i2c_rxData(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Returns	[OUT] 1 byte data from I^{2}C slave
Include	**driver/i2c.h**
Description	Receive one byte data from I^{2}C slave.

i2c_rxNack()

Usage	`int i2c_rxNack(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Returns	[OUT] 1 bit no acknowledge
Include	**driver/i2c.h**
Description	Receive no acknowledge from the I^{2}C slave.

I/O API Calls

read_u8()

Usage	`u8 read_u8(u32 address)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address
Description	Read address with unsigned 8 bits.

read_u16()

Usage	`u16 read_u16(u32 address)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address
Description	Read address with unsigned 16 bits.

read_u32()

Usage	`u32 read_u32(u32 address)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address
Description	Read address with unsigned 32 bits.

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` timer interrupt
	[IN] `cmp` timer compare register
	[IN] `hart_id` hart ID
Description	Set a timer value to trigger an interrupt.

clint_getTime()

Usage	`u64 clint_getTime(u32 p)`
Include	**driver/clint.h**
Parameters	[IN] `p` timer interrupt
Returns	[OUT] 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` timer interrupt
Description	Use the timer to make a delay.

User Timer API Calls

prescaler_setValue()

Usage	`void prescaler_setValue(u32 reg, u32 value)`
Include	**driver/prescaler.h**
Parameters	[IN] `reg` struct of user timer register [IN] `prescaler` 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` struct of user timer register [IN] `value` user timer configuration value
Description	Set the user timer configuration.

timer_setLimit()

Usage	`void timer_setLimit(u32 reg, u32 value)`
Include	**driver/timer.h**
Parameters	[IN] `reg` struct of user timer register [IN] `value` user timer configuration value
Description	Set the limit value for the timer to generate an interrupt.

timer_getValue()

Usage	`void timer_getValue(u32 reg)`
Include	**driver/timer.h**
Parameters	[IN] `reg` struct of user timer register
Returns	Timer value
Description	Get the timer value.

timer_clearValue()

Usage	`void timer_clearValue(u32 reg)`
Include	**driver/timer.h**
Parameters	[IN] `reg` struct of user timer register
Description	Clear the timer value and set 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 register structure [IN] `gateway` interrupt type [IN] `priority` interrupt priority
Description	Set 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 register structure [IN] `target` HART number [IN] `gateway` interrupt type [IN] `enable`
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 register structure [IN] `target` HART number [IN] `threshold` enable = 1
Description	Masks individual interrupt sources for the HART.

plic_claim()

Usage	`u32 plic_claim(u32 plic, u32 target)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC register structure [IN] `target` HART number
Description	Claim the PLIC interrupt

plic_release()

Usage	`void plic_release(u32 plic, u32 target, u32 gateway)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC register structure [IN] `target` HART number [IN] `gateway` interrupt type
Description	Release the PLIC interrupt.

SPI API Calls

spi_applyConfig()

Usage	`void spi_applyConfig(Spi_Reg *reg, Spi_Config *config)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `config` struct of the SPI configuration
Description	Applies the SPI configuration to to a register for initial configuration.

spi_cmdAvailability()

Usage	`spi_cmdAvailability(Spi_Reg *reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register
Description	Read the SPI command buffer.

spi_diselect()

Usage	`void spi_select(Spi_Reg *reg, uint32_t slaveId)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `slaveId` ID for the slave
Description	De-asserts the SPI select (SS) pin.

spi_read()

Usage	`uint8_t spi_write(Spi_Reg *reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register
Returns	[OUT] `reg` One byte of data
Description	Receives one byte from the SPI slave.

spi_rspOccupancy()

Usage	`spi_rspOccupancy(Spi_Reg *reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register
Description	Read the occupancy buffer.

spi_select()

Usage	`void spi_select(Spi_Reg *reg, uint32_t slaveId)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `slaveId` ID for the slave
Description	Asserts the SPI select (SS) pin.

spi_write()

Usage	`void spi_write(Spi_Reg *reg, uint8_t data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `data` 8 bits of data to send out
Description	Transfers one byte to the SPI slave.

spi_writeRead()

Usage	`uint8_t spi_writeRead(Spi_Reg *reg, uint8_t data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `data` 8 bits of data to send out
Returns	[OUT] `reg` one byte of data
Description	Transfers one byte to the SPI slave and receives one byte from the SPI slave.

SPI Flash Memory API Calls

spiFlash_f2m_()

Usage	`void spiFlash_f2m_(Spi_Reg * spi, u32 flashAddress, u32 memoryAddress, u32 size)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `flashAddress` flash device address [IN] `memoryAddress` memory address [IN] `size` programming address size
Description	Copy data from the flash device to memory.

spiFlash_f2m()

Usage	`void spiFlash_f2m(Spi_Reg * spi, u32 cs, u32 flashAddress, u32 memoryAddress, u32 size)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `cs` chip select [IN] `flashAddress` flash device address [IN] `memoryAddress` memory address
Description	Copy data from the flash device to memory with chip select control.

spiFlash_f2m_withGpioCs()

Usage	`void spiFlash_f2m_withGpioCs(Spi_Reg * spi, Gpio_Reg *gpio, u32 cs, u32 flashAddress, u32 memoryAddress, u32 size)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `gpio` reg struct of the GPIO register [IN] `cs` chip select [IN] `flashAddress` flash device address [IN] `memoryAddress` memory address [IN] `size` programming address size
Description	Flash device from the SPI master with GPIO chip select.

spiFlash_diselect()

Usage	`void spiFlash_diselect(Spi_Reg *spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `cs` chip select
Description	De-asserts the SPI flash device from the master chip select.

spiFlash_diselect_withGpioCs()

Usage	`void spiFlash_diselect_withGpioCs(Gpio_Reg *gpio, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `gpio` reg struct of the GPIO register [IN] `cs` chip select
Description	De-asserts the SPI flash device from the master with the GPIO chip select.

spiFlash_init_()

Usage	`void spiFlash_init_(Spi_Reg * spi)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register
Description	Initialize the SPI reg struct.

spiFlash_init()

Usage	`void spiFlash_init(Spi_Reg * spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `cs` chip select
Description	Initialize the SPI reg struct with chip select de-asserted.

spiFlash_init_withGpioCs()

Usage	`void spiFlash_init_withGpioCs(Spi_Reg * spi, Gpio_Reg *gpio, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `gpio` reg struct of the GPIO register [IN] `cs` chip select
Description	Initialize the SPI reg struct with GPIO chip select de-asserted.

spiFlash_read_id_()

Usage	`u8 spiFlash_read_id_(u32 spi)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register
Returns	8-bit SPI flash ID
Description	Read the ID from the flash.

spiFlash_read_id()

Usage	`u8 spiFlash_read_id(u32 spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `cs` chip select
Returns	8-bit SPI flash ID
Description	Read the ID from the flash with chip select.

spiFlash_select()

Usage	`void spiFlash_select(Spi_Reg *spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `cs` chip select
Description	Select the SPI flash device.

spiFlash_select_withGpioCs()

Usage	`spiFlash_select_withGpioCs(Gpio_Reg *gpio, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `gpio` reg struct of the GPIO register [IN] `cs` chip select
Description	Select the SPI flash device with the GPIO chip select.

spiFlash_software_reset()

Usage	`void spiFlash_software_reset(Spi_Reg * spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `cs` chip select
Description	Reset the SPI flash.

spiFlash_wake_()

Usage	`void spiFlash_wake_(Spi_Reg * spi)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register
Description	Release power down from the SPI master.

spiFlash_wake()

Usage	`void spiFlash_wake(Spi_Reg * spi, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `cs` chip select
Description	Release power down from the SPI master with chip select.

spiFlash_wake_withGpioCs()

Usage	`void spiFlash_wake_withGpioCs(Spi_Reg * spi, Gpio_Reg *gpio, u32 cs)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` reg struct of the SPI register [IN] `gpio` reg struct of the GPIO register [IN] `cs` chip select
Description	Release power down from the SPI master with the GPIO chip select.

UART API Calls

uart_applyConfig()

Usage	`char uart_applyConfig(Uart_Reg *reg, Uart_Config *config)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register [IN] `config` struct of the UART configuration
Description	Applies the UART configuration to to a register for initial configuration.

uart_emptyInterruptEna()

Usage	`uart_emptyInterruptEna(u32 reg char ena)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register [IN] `ena` Enable interrupt
Description	Enable the TX FIFO empty interrupt.

uart_NotemptyInterruptEna()

Usage	`uart_NotemptyInterruptEna(u32 reg char ena)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register [IN] `ena` Enable interrupt
Description	Enable the RX FIFO not empty interrupt.

uart_read()

Usage	`char uart_read(Uart_Reg *reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register
Returns	[OUT] `reg` character that is read
Description	Reads a character from the UART slave.

uart_readOccupancy()

Usage	`uint32_t uart_readOccupancy(Uart_Reg *reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register
Description	Read the number of bytes in the RX FIFO.

uart_status_read()

Usage	`uart_status_read(u32 reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register
Returns	[OUT] 32-bit status register from the UART
Description	Read the UART status.

uart_status_write()

Usage	`uart_status_write(u32 reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register [IN] `data` input data for the UART status.
Returns	[OUT] 32-bit status register from the UART
Description	Write the UART status.

uart_write()

Usage	`void uart_write(Uart_Reg *reg, char data)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register [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` struct of the UART register [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(Uart_Reg *reg, char* str)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register [IN] `str` string to write
Description	Write a string to the UART TX.

uart_writeAvailability()

Usage	`uart_writeAvailability(Uart_Reg *reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register
Description	UART read/write FIFO.

Handling Interrupts

There are two kinds of interrupts, trap vectors and PLIC interrupts, and you handle them using different methods.

Figure 9: 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 10: Handling Trap Vectors

```
                    ╭─────────────╮
                    │  Call Trap  │
                    ╰──────┬──────╯
                           │
                    ┌──────┴──────┐
                    │ Read mcause │
                    └──────┬──────┘
                           │
                           ▼
                    ◇ Is Interrupt? ◇ ──yes──▶  ┌─────────────────┐
                           │                     │ Call Exceptions()│
                           no                    └─────────────────┘
                           │
                           ▼
              ◇ CAUSE_CLINT_TIMER? ◇ ──yes──▶  ┌──────────────┐
                           │                    │ Call Timer()  │
                           no                   └──────────────┘
                           │
                           ▼
        ◇ CAUSE_MACHINE_EXTERNAL? ◇ ──yes──▶  ┌──────────────────────┐
                           │                    │ Call ExternalInterrupt()│
                           no                   └──────────────────────┘
                           │
                           ▼
              ┌────────────────────┐
              │ Call Exceptions() or│
              │   handle by user    │
              └────────────────────┘
```

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 38 for a list of
the interrupt IDs.

ⓘ **Note:** For the Sapphire SoC, the interrupt IDs are user configurable. Refer to the interrupt IDs that you set
in the IP Manager for each peripheral. The Address Map shows the default values.

The following flow chart shows how the PLIC handles interrupts.The PLIC identifies the
interrupt ID and processes the corresponding interrupts.

Figure 11: Handling PLIC Interrupts

```
                        ╭──────────────────────╮
                        │ Call ExternalInterrupt() │
                        ╰──────────────────────╯
                                   │
                                   ▼
                        ┌──────────────────┐
                  ┌────▶│  Read PLIC Claim │◀────┐
                  │     └──────────────────┘     │
                  │              │               │
                  │              ▼               │
                  │        ◇ Is Interrupt ◇  no  │
                  │        ◇   Claimed?   ◇──────┘
                  │              │
                  │             yes
                  │              │
                  │              ▼
                  │   ◇ PLIC_SYSTEM_I2C_INTERRUPT? ◇  yes   ┌──────────────────┐
                  │   ◇                            ◇──────▶│ Call I2C Interrupt() │───┐
                  │              │                          └──────────────────┘   │
                  │              no                                                │
                  │              ▼                                                 │
                  │   ◇ PLIC_SYSTEM_UART_INTERRUPT? ◇  yes  ┌──────────────────┐   │
                  │   ◇                             ◇─────▶│ Call UART Interrupt()│──┤
                  │              │                          └──────────────────┘   │
                  │              no                                                │
                  │              ▼                                                 │
                  │   ◇ PLIC_SYSTEM_SPI_INTERRUPT? ◇  yes   ┌──────────────────┐   │
                  │   ◇                            ◇──────▶│ Call SPI Interrupt() │──┤
                  │              │                          └──────────────────┘   │
                  │              no                                                │
                  │              ▼                                                 │
                  │   ◇ PLIC_SYSTEM_USER_        ◇  yes     ┌────────────────────────┐│
                  │   ◇ TIMER_INTERRUPT?         ◇────────▶│ Call User Timer Interrupt()│┤
                  │              │                          └────────────────────────┘│
                  │              no                                                  │
                  │              ▼                                                   │
                  │   ◇ Other PLIC Interrupt?  ◇  yes       ┌──────────────────────┐ │
                  │   ◇                        ◇──────────▶│ Call Other PLIC Interrupt()│┤
                  │              │                          └──────────────────────┘ │
                  │              no                                                  ▼
                  │              ▼                                          ┌──────────────────┐
                  │     ┌──────────────────┐                               │     Release      │
                  │     │ Call Exceptions() │                               │ Claimed Interrupt│
                  │     └──────────────────┘                               └──────────────────┘
```

Revision History

Table 23: Revision History

Date	Version	Description
December 2021	3.0	Updated the SDK version numbers. Updated the IP Manager Configuration Wizard description for new configuration options. Added instructions for using the Ti60 F225 Development Board and example design. Updated instructions for Eclipse global environment variables. Explained new Efinity Programmer feature for programming a flash device with a combined user bitstream and application binary. Updated register map. Updated the API Reference for new driver support.
October 2021	2.1	Corrected incomplete instructions for copying a user binary to flash. (DOC-576)
October 2021	2.0	IP Manager options changed for the updated Sapphire wizard. (DOC-544) Updated the address map. (DOC-544) Updated the example design description for the new features in the design. (DOC-544) New simulation instructions. (DOC-544) New instructions for changing the bootloader RAM size. (DOC-544) Changed the EfxApb3Example, EfxAxi4Example, and userInterruptDemo example descriptions. (DOC-544) Changed the TX pin number for the instructions on setting up a USB-to-UART module. (DOC-544) When using the Soft Debug Tap option, the IP Manager connects the pins for you. (DOC-544) Described the pins needed to connect an FTDI cable to the Trion® T120 BGA324 Development Board when using the Soft Debug Tap option. (DOC-544)
August 2021	1.1	Corrected typo in example design name in topics describing Eclipse and OpenOCD (EfxAxi4Example instead of EfxAxiExample). (DOC-517)
July 2021	1.0	Initial release.