Opal (Xyloni) RISC-V SoC Hardware and Software User Guide

UG-RISCV-OPALX-v1.2
November 2021
www.elitestek.com

Copyright © 2021. All rights reserved. 易灵思, the 易灵思 logo, the 钛金系列 logo, Quantum, Trion, and Efinity are trademarks of 易灵思®. All other trademarks and service marks are the property of their respective owners. All specifications subject to change withoutnotice.

Contents

Introduction ..... **iv**
VexRiscv RISC-V Core iv
Required Software v
Required Hardware v

Chapter 1: Install Software and SoC **6**
Install the Efinity® Software 6
Install the SoC Files 6
Install the RISC-V SDK 7
Install the Java JRE 7
Installing USB Drivers 7

Chapter 2: About the RTL Example Design **9**

Chapter 3: Launch Eclipse **10**
Set Global Environment Variables 10

Chapter 4: Create and Build a Software Project **12**
Create a New Project 12
Import Project Settings (Optional) 13
Enable Debugging 14
Build 15

Chapter 5: Debug with the OpenOCD Debugger **16**
Import the Debug Configuration 16
Debug 17
Enable Telnet on Windows 18
Open a Terminal 19

Chapter 6: Create Your Own RTL Design **20**
Create a Custom APB3 Peripheral 20
Remove Unused Peripherals from the RTL Design 20

Chapter 7: Create Your Own Software **21**
Deploying an Application Binary 21
Boot from a Flash Device 21
Boot from the OpenOCD Debugger 21
Copy a User Binary to the Flash Device 22
About the Board Specific Package 23
Address Map 24
Example Software 25
blinkAndEcho Example 26
EfxApb3Example 26
i2cDemo Example 26
readFlash Example 26
spiDemo Example 27
timerAndGpioInterruptDemo Example 27
userInterruptDemo Example 27
writeFlash Example 28
Xyloni_SelfTest Example 28

Chapter 8: Troubleshooting **29**
Error 0x80010135: Path too long (Windows) 29
OpenOCD Error: timed out while waiting for target halted 29

OpenOCD error code (-1073741515)..... 30
OpenOCD Error: no device found..... 30
OpenOCD Error: failed to reset FTDI device: LIBUSB_ERROR_IO..... 30
OpenOCD Error: target 'fpga_spinal.cpu0' init failed..... 31
Eclipse Fails to Launch with Exit Code 13..... 31
Undefined Reference to 'cosf'..... 31

Chapter 9: API Reference..... 32
Control and Status Registers..... 32
GPIO API Calls..... 33
I^2C API Calls..... 37
I/O API Calls..... 42
Machine Timer API Calls..... 44
PLIC API Calls..... 44
SPI API Calls..... 45
SPI Flash Memory API Calls..... 47
UART API Calls..... 49
Handling Interrupts..... 51
Revision History..... 54

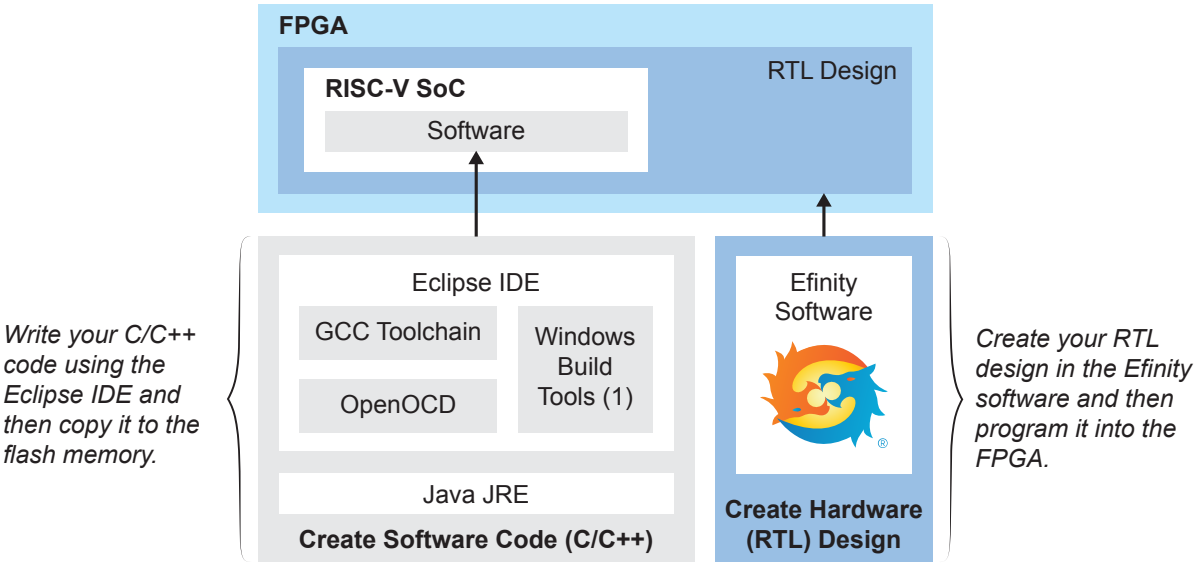
Introduction

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

- Build RTL designs using the Opal 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: This user guide describes how to use the Opal SoC that is provided with the Efinity® v2020.2 or higher. Previous versions of the SoC were available as downloads in the Support Center. Although the functionality of the SoC is essentially the same, the IP Manager allows you to set parameters to customize the Opal SoC, and the resulting directory structure is different.

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



1. Windows build tools required on Windows platforms only.

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

VexRiscv RISC-V Core

The Opal 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 and C extensions, has five pipeline stages (fetch, decode, execute, memory, and writeback), and a configurable feature set.

In the Opal SoC, the cacheless VexRiscv core supports an APB3 bus interface and can run at speeds up to 0.98 DMIPS/MHz.

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

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

Required Software

To write software for the Opal 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: 2020.2 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: 8.3.0-1.1

Disk space required: 1.78 GB (Windows), 1.73 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: 2.12-20190422-1053

Disk space required: 3.8 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)

Required Hardware

- Xyloni Development Board
- Micro-USB cable
- Computer or laptop

Chapter 1

Install Software and SoC

Contents:

- Install the Efinity Software
- Install the SoC Files
- Install the RISC-V SDK
- Install the Java JRE
- Installing USB Drivers

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 SoC Files

To install the Opal SoC:

1. Create a directory called **riscv** at the root level of your file system.
2. Download the files from Github in the **xyloni/design/soc_Opal_t8** directory and save them into the **riscv** directory.

The files are organized in this directory structure:

- **soc_Opal_t8**
  - **soc_Opal_hw_t8**—Hardware files.
    - **source**—RTL source code for the T8 BGA81 FPGA.
    - **Xyloni_kit**—Example Efinity® project targeting the Xyloni Development Board.
  - **soc_Opal_sw_t8**—Software files.
    - **bsp**—Board specific package.
    - **software**—Software examples.
    - **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.
    - **cpu0.yaml**—CPU file for debugging.

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_8.3.0-1.1_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_8.3.0-1.1_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.

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.

If you have not already done so, install separate interface drivers for the Xyloni Development Board. When working with OpenOCD, you need to install the **libusbK** driver as described in the following section.

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

Separate Interfaces

You install drivers for separate interfaces when you want to use each interface independently.

Important: For some 易灵思 development boards, Windows automatically installs drivers for some interfaces when you connect the board to your computer. You do not need to install another driver for these interfaces. Refer to the user guide for your development board for specific driver installation requirements.

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. 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.

Installing Drivers on Linux

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

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

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

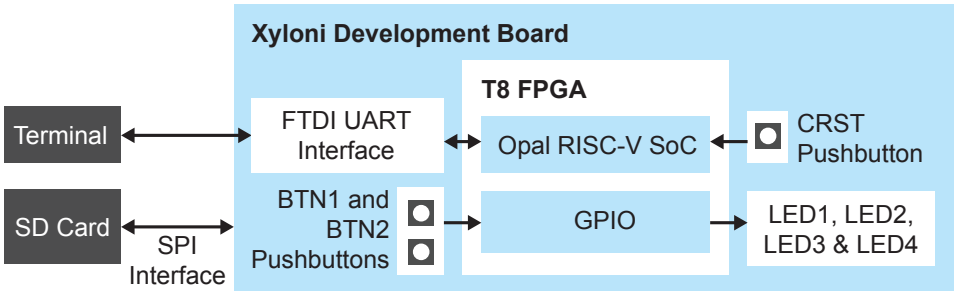
Note: If your board was connected to your computer before you executed these commands, you need to disconnect and re-connect it.

Chapter 2

About the RTL Example Design

易灵思® preloads the Xyloni Development Board with an example design and firmware. The design includes two functions, the invert LED operation and the read SD card information operation. The read SD card information operation requires a terminal program in a computer and an SD card inserted into the SD card slot to display the SD card information.

Figure 2: Xyloni Development Board Example Design Block Diagram



Learn more: Refer to Xyloni_SelfTest Example on page 28 for information on running the design.

In addition to the default firmware, you can run other example software as described in Example Software on page 25. You can use the pre-loaded example design to run this other example software without making any RTL changes.

Chapter 3

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 Opal BSP, and launches Eclipse. Always use this executable to launch Eclipse; do not launch Eclipse directly.

When you first start working with the Opal 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 Opal 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 Opal SoC projects. 易灵思 recommends using the **soc_Opal_t8** directory as the workspace directory.

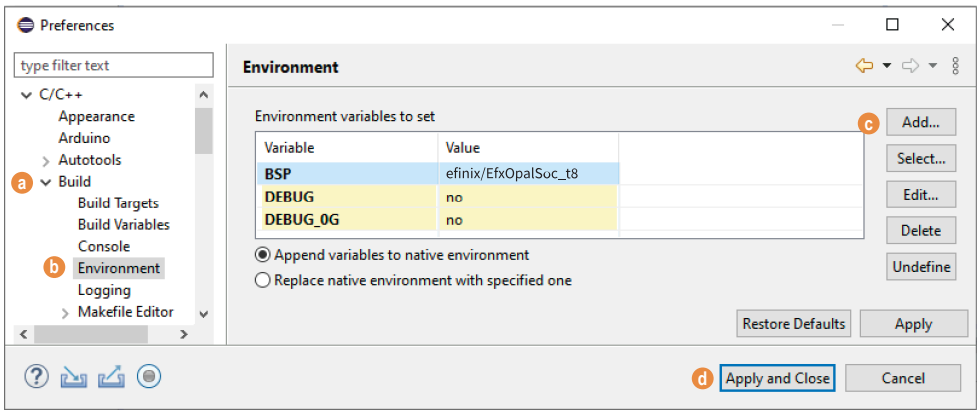
Follow these steps to launch Eclipse and set up your workspace:

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 4 for Opal_T8 and press enter.
3. If this is the first time you are running Eclipse, create a new workspace that points to the **soc_Opal_sw_t8** directory. Otherwise, choose **File > Switch Workspace > Other** to choose an existing workspace directory and click **Launch**.

Set Global Environment Variables

You need to set two environment variables for OpenOCD. 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.

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 4

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 **EfxApb3Example** example project from the **software** directory.

Create a New Project

In this step you create a new project from the **EfxApb3Example** code example.

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

Import the **EfxApb3Example** example:

4. Choose **File > New > Makefile Project with Existing Code**.
5. Click **Browse** next to **Existing Code Location**.
6. Browse to the **software/standalone/EfxApb3Example** 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.

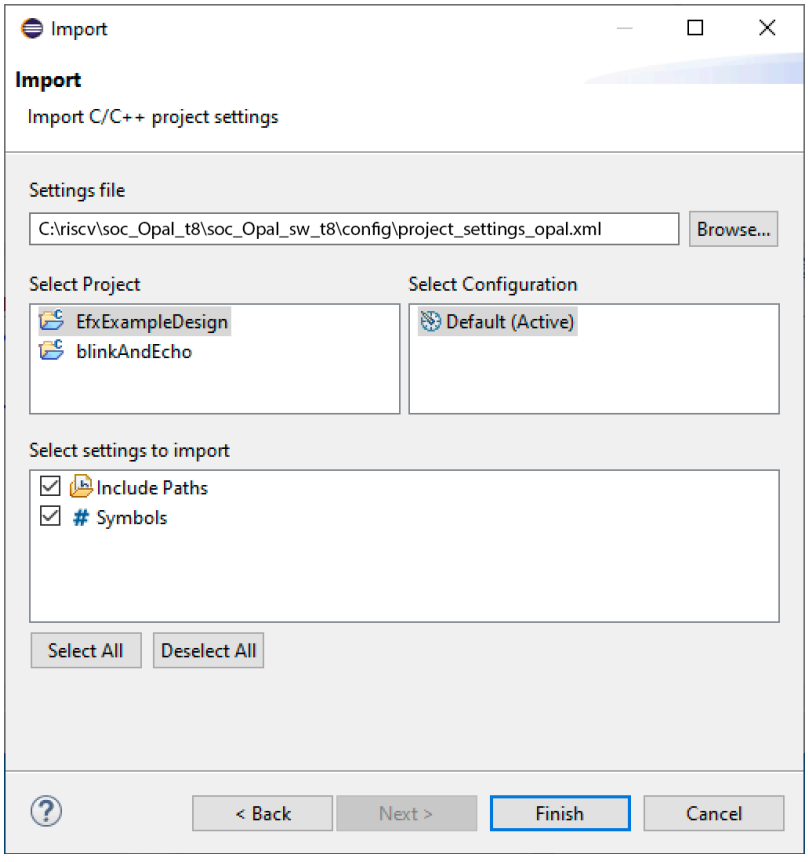
Note: You are not required to import the project settings to build. These settings simply make it easier for you to write and debug code.

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	**soc_Opal_t8\soc_Opal_sw_t8\config\project_settings_opal.xml**
Linux	**soc_Opal_t8/soc_Opal_sw_t8/config_linux/ project_settings_opal_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.

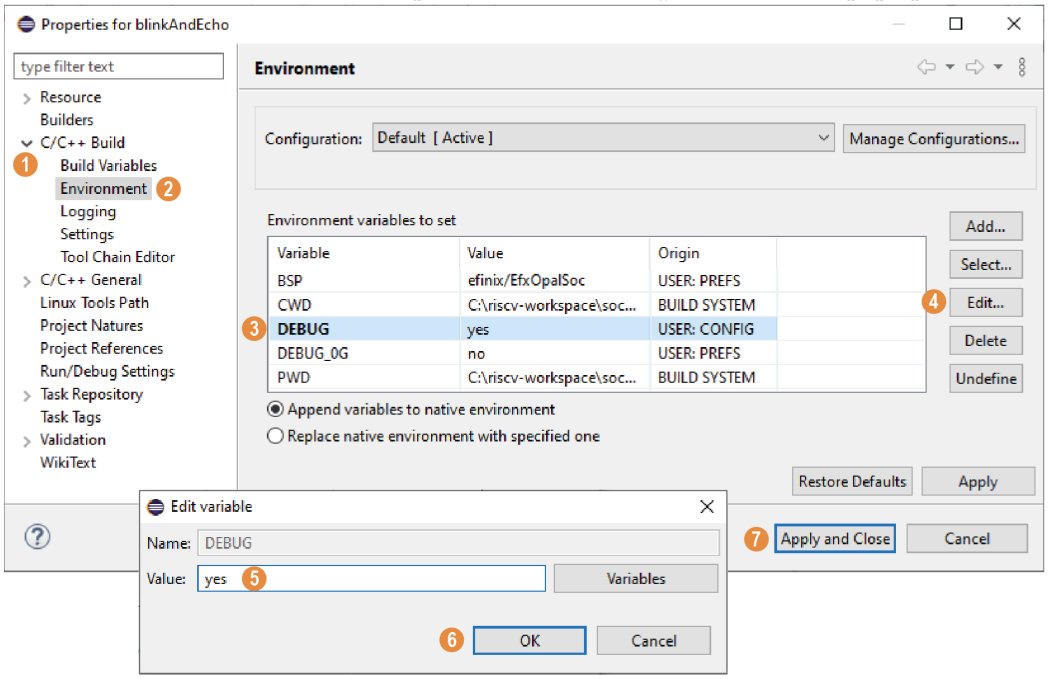
Enable Debugging

When you set up your workspace, you defined an environment variable for debugging with a default value of **no**.

- 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 **EfxApb3Example** 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:

- **EfxApb3Example.asm**—Assembly language file for the firmware.
- **EfxApb3Example.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.
- **EfxApb3Example.elf**—Use this file when debugging with the OpenOCD debugger.
- **EfxApb3Example.hex**—Hex file for the firmware. (Do not use it to program the FPGA.)
- **EfxApb3Example.map**—Contains the SoC address map.

Chapter 5

Debug with the OpenOCD Debugger

Contents:

- Import the Debug Configuration
- Debug
- Enable Telnet on Windows
- Open a Terminal

With the Xyloni Development Board programmed and the software built, you are ready to configure the OpenOCD debugger and perform debugging. These instructions use the **EfxApb3Example** example to explain the steps required.

Import the Debug Configuration

To simplify the debugging steps, the Opal SoC includes a debug configuration that you import.

Note: If you have already imported the launch configuration described in Import the Run Configuration, you only need to perform steps 7 and 12 to debug.

1. Right-click the **EfxApb3Example** project name and choose **Import**.
2. In the Import dialog box, choose **Run/Debug > Launch Configurations**.
3. Click **Next**. The Import Launch Configurations dialog box opens.
4. Browse to the following directory and click **OK**:

Option	Description
Windows	soc_Opal_t8\embedded_sw\soc_Opal_sw_t8\config
Linux	soc_Opal_t8/embedded_sw/soc_Opal_sw_t8/config_linux

5. Check the box next to **config** (Windows) or **config_linux** (Linux).
6. Click **Finish**.
7. Right-click the **EfxApb3Example** project name and choose **Debug As > Debug Configurations**.
8. Choose **GDB OpenOCD Debugging > default** (Trion FPGAs) or **default_ti** (钛金系列 FPGAs).
9. Enter `EfxApb3Example` in the **Project** box.
10. Enter `build\EfxApb3Example.elf` in the **C/C++ Application** box.
11. *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_Opal_sw_t8** 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:\\riscv\\soc_Opal_t8\\soc_Opal_sw_t8\\cpu0.yaml

12. Click **Debug**.

Note: When you click **Debug**, the debugger sends a soft reset to the SoC, and then writes the user binary file to logical address 0x0000_1000, which is the starting address of the external memory. The Ruby Vision SoC then jumps to logical address 0x0000_1000 to execute the user binary.

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

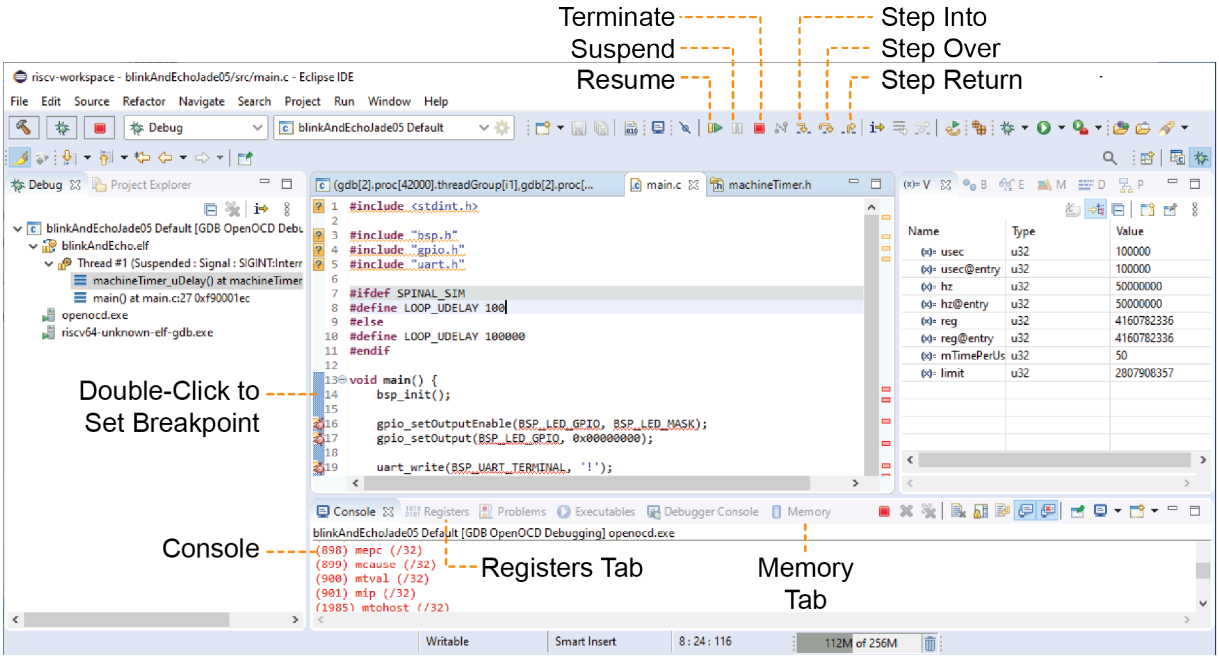
Debug

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 EfxApb3Example example blinks the LEDs and prints messages on a UART terminal. Refer to **Enable Telnet on Windows** on page 18 and **Open a Terminal** on page 19 for more instructions on setting up a terminal.

Figure 3: Perform Debugging



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

Enable Telnet on Windows

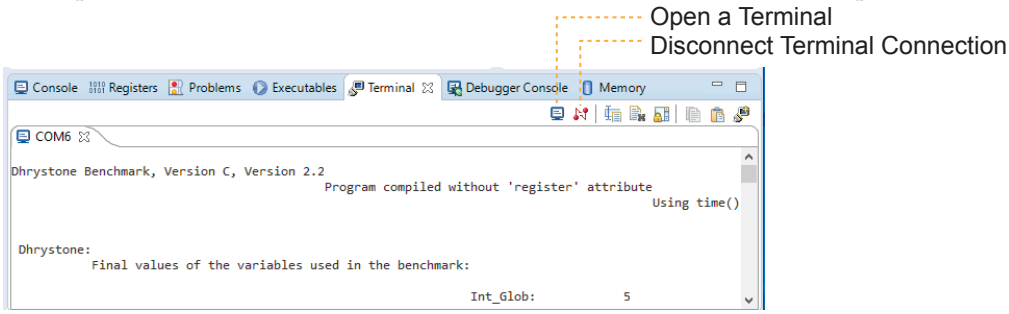
Windows does not have telnet turned on by default. Follow these instructions to enable it:

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**.

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.

Chapter 6

Create Your Own RTL Design

Contents:

- Create a Custom APB3 Peripheral
- Remove Unused Peripherals from the RTL Design

After you have explored the Opal 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.

Create a Custom APB3 Peripheral

This simple APB3 peripheral example shows how to implement an APB3 slave wrapper.

- Refer to **apb3_slave.v** in the **Xyloni_kit** directory for the RTL design.
- Refer to **main.c** in the **soc_Opal_t8/soc_Opal_sw_t8/software/standalone/EfxApb3Example/src** directory for the C code.

Remove Unused Peripherals from the RTL Design

The Opal 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 OpalSoc OpalSoc_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^2C 0, remove the signal name in () as shown below:

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

Chapter 7

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 Opal SoC, you can develop your own software applications.

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

Deploying an Application Binary

During normal operation, the user binary application file (**.bin**) is stored in a SPI flash device. When the FPGA powers up, the Opal SoC copies your binary application file from the SPI flash device to the on-chip memory, and then begins execution.

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

Note: The settings in the linker prevent user access to the on-chip RAM 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 Opal SoC copies your binary application file from a SPI flash device to the on-chip memory, and then begins execution. The SPI flash binary address starts at 0x00E_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 4 KByte user binary file from the flash device at physical address 0x00E_0000 to the on-chip memory.

   Note: It takes ~10 ms to clone a 4 KByte user binary (this is the default size).
3. The Opal SoC executes 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. When you click **Debug**, the debugger sends a soft reset to the SoC, and then writes the user binary file to logical address 0xF900_0000, which is the starting address of the on-chip memory.
4. The Opal SoC jumps to logical address 0xF900_0000 to execute the user binary.
5. The user binary is suspended on boot up. Click the Resume button to start the program.

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

Copy a User Binary to the Flash Device

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

- 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 (Trion):

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

Windows (钛金系列):

```
openocd.exe -f bsp\efinix\EfxOpalSoc\openocd\ftdi_ti.cfg
   -c "set CPU0_YAML cpu0.yaml"
   -f bsp\efinix\EfxOpalSoc\openocd\flash_ti.cfg
```

Linux (Trion):

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

Linux (钛金系列):

```
openocd -f bsp/efinix/EfxOpalSoc/openocd/ftdi_ti.cfg
   -c "set CPU0_YAML cpu0.yaml"
   -f bsp/efinix/EfxOpalSoc/openocd/flash_ti.cfg
```

The OpenOCD server connects and begins listening on port 4444.

Set Up Terminal 2

1. Open a second command prompt or shell.
2. Enable telnet if it is not turned on. **Turn on telnet (Windows)**
3. Open a telnet local host on port 4444 with the command `telnet localhost 4444`.

4. In the OpenOCD shell or command prompt, use the following command to flash the user binary file:

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

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

Note: For Windows, use \\ as the directory separators.

Close Terminals

When you finish:

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

Important: OpenOCD cannot be running in 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 (CRST) on the development board.

About the Board Specific Package

The board specific package (BSP) defines the address map and aligns with the Opal SoC hardware address map. The BSP files are located in the **bsp/efinix/EfxOpalSoC_t8** subdirectory.

Table 1: 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

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 2: Default Address Map, Interrupt ID, and Cached Channels

Device	Parameter	Size	Interrupt ID	Region
GPIO	SYSTEM_GPIO_0_IO_APB	4K	[0]: 12 [1]: 13	I/O
I^2C	SYSTEM_I2C_0_IO_APB	4K	8	I/O
Machine timer	SYSTEM_MACHINE_TIMER_APB	4K	31	I/O
PLIC	SYSTEM_PLIC_APB	4K	–	I/O
SPI master 0	SYSTEM_SPI_0_IO_APB	4K	4	I/O
SPI master 1[(2)]	SYSTEM_SPI_1_IO_APB	4K	5	I/O
UART	SYSTEM_UART_0_IO_APB	4K	1	I/O
User peripheral	IO_APB_SLAVE_0_APB	64K	–	I/O
On-chip BRAM	SYSTEM_RAM_A_BMB	4 KB	–	Cache
External interrupt	–	–	25	I/O

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

[(2)] The open-source Opal SoC available on Github has 2 SPI masters. Other variations only have 1.

Example Software

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

Note: Many of these examples display messages on a UART. The Xyloni Development Board has an on-board UART module, and you can connect to it with a terminal program.

Learn how to open an Eclipse terminal and connect to the UART.

Table 3: Example Software Code

Directory	Description
blinkAndEcho	This example blinks an LED and prints a string on the UART terminal.
bootloader	This software is the bootloader for the system.
common	Provides linking for the makefiles.
driver	This directory contains the system drivers for the peripherals (I^2C, UART, SPI, etc.). Refer to API Reference on page 32 for details.
EfxApb3Example	This example shows how to implement an ABP3 slave.
i2cDemo	This example shows how to connect to an MCP4725 digital-to-analog converter (DAC) using an I^2C peripheral.
readFlash	This example shows how to read from 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.
timerAndGpioInterruptDemo	This example shows how to use use interrupts with a timer and GPIO.
userInterruptDemo	This example demonstrates user interrupts with UART messages.
writeFlash	This example shows how to write to a SPI flash device.
Xyloni_SelfTest	This software is pre-loaded onto the Xyloni development board.

blinkAndEcho Example

The blink and echo example (**blinkAndEcho** directory) is a simple example that shows how to use a register pointer to output data for the GPIO and UART. The application blinks LEDs on the Xyloni Development Board. When you type a character, it echoes it on a UART terminal.

EfxApb3Example

This simple software design illustrates how to use an APB3 slave peripheral. The RISC-V processor controls the first 2 bits of address 0xf8800000; these 2 bits are connected to the Xyloni Development Board's LEDs. When the processor writes a 1, all of the LEDs turn on; for a 0, all of them turn off. The application also displays the message `Opal Soc T8: Example Design` on a UART terminal.

i2cDemo Example

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

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

- *SCL*—GPIOR_35, which is pin 17 on header J1
- *SDA*—GPIOR_36, which is pin 18 on header J1

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

In this example:

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

readFlash Example

The read flash example (**readFlash** directory) shows how to read data from the SPI flash device on the development board. The software reads 124K of data starting at address 0xe0000, 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
```

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.

```
Hello world
Device ID : 17
CMD 0x9F : EF4018
CMD 0x9F : EF4018
...
```

timerAndGpioInterruptDemo Example

The GPIO interrupt example (**timerAndGpioInterruptDemo** directory) provides example code for implementing a rising edge interrupt trigger with a GPIO pin. When an interrupt occurs, a UART terminal displays Hello world and then the timer interval. It continues to print the timer interval until you suspend or stop the application.

```
Hello world
BSP_MACHINE_TIMER 0
BSP_MACHINE_TIMER 1
...
```

In this example:

- `void trap()` traps entries on exceptions and interrupt events
- `void externalInterrupt()` triggers a GPIO interrupt event

userInterruptDemo Example

The user interrupt example (**userInterruptDemo** directory) uses one bit from an APB3 slave peripheral as an interrupt signal to RISC-V processor. The main routine sets up an interrupt routine, then triggers an interrupt signal to the user interrupt port by programming bit 2 on the ABP3 slave to high.

When the RISC-V processor receives the interrupt signal, program execution jumps from the main routine to the interrupt (or priority) routine. The interrupt routine sets bit 2 low so the processor can leave the interrupt routine.

The application displays the messages on a UART terminal:

```
User Interrupt Demo, waiting for user interrupt...
Entered User Interrupt Routine
Turn off Interrupt Signal
Leaving User Interrupt Routine
```

writeFlash Example

The read flash example (**writeFlash** directory) shows how to write data to the SPI flash device on the development board. The software writes data starting at address 0xe0000, 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
```

Xyloni_SelfTest Example

This example is pre-loaded into the Xyloni Development Board. The design has two operations:

- *Invert LED*—Reverses the direction of the LED blinking pattern when you press BTN2.
- *Read SD card*—Press BTN1 to start the test. Then press Enter and the software reads information about the inserted SD card and displays it on a UART terminal.

Refer to the **Xyloni Development Kit User Guide** for detailed instructions on running the example.

Important: You cannot use Debug mode in OpenOCD because there is insufficient on-chip user memory for this design. Instead, run the program for normal operation, and set the **DEBUG** variable to **no**. See **Enable Debugging** on page 14.

Chapter 8

Troubleshooting

Contents:

- Error 0x80010135: Path too long (Windows)
- OpenOCD Error: timed out while waiting for target halted
- 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
- 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 SoC design.
- You may not have the correct PLL settings to work with the SoC.
- Your computer does not have enough memory to run the program.

To solve this problem:

- Make sure that all of the cables are securly connected to the board and your computer.
- Check the JTAG connection.
- If you programmed the Xyloni Development Board with another design, you need to restore it with the example design.

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 Opal 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/EFXOpalSoC/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 hav e 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 '*'
```

Important: 易灵思 recommends using the **libusbK** driver, which you install using the Zadig software. See Installing USB Drivers for information.

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.

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 Opal SoC does not currently support floating point.

Chapter 9

API Reference

Contents:

- Control and Status Registers
- GPIO API Calls
- I2C API Calls
- I/O API Calls
- Machine Timer API Calls
- PLIC API Calls
- SPI API Calls
- SPI Flash Memory API Calls
- UART API Calls
- Handling Interrupts

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`

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^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_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^2C setting value
Include	**driver/i2c.h**
Description	Read the 32-bit I^2C register interrupt flag with a call back function.

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

gpio_getMasterStatus()

Usage	`gpio_getMasterStatus(reg)`
Parameters	[IN] reg struct of I^2C setting value
Include	**driver/i2c.h**
Description	Read the 32-bit I^2C register master status with a call back function.

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

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.

I²C API Calls

i2c_applyConfig()

Usage	`void i2c_applyConfig(u32 reg, I2c_Config *config)`
Parameters	[IN] `reg` struct of I²C setting value [IN] `config` struct of I²C configuration
Include	**driver/i2c.h**
Description	Apply I²C configuration to register or for initial configuration.

i2c_clearInterruptFlag()

Usage	`void i2c_clearInterruptFlag(u32 reg, u32 value)`
Parameters	[IN] `reg` struct of I²C setting value [IN] `value` I²C interrupt register
Include	**driver/i2c.h**
Description	Clear the I²C interrupt flag.

i2c_disableInterrupt()

Usage	`void i2c_disableInterrupt(u32 reg, u32 value)`
Parameters	[IN] `reg` struct of I²C setting value [IN] `value` I²C 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 I²C 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_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^2C register
Include	**driver/i2c.h**
Description	Change the I^2C master to the start status.

i2c_masterStartBlocking()

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

i2c_masterStop()

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

i2c_masterStopBlocking()

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

i2c_masterStopWait()

Usage	`void i2c_masterStopWait(u32 reg)`
Parameters	[IN] `reg` struct of I^2C 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 I^{2}C setting value [IN] `filterID` filter configuration ID number [IN] `config` struct of I^{2}C 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 I^{2}C register
Include	**driver/i2c.h**
Description	Transmit acknowledge.

i2c_txAckBlocking()

Usage	`void i2c_txAckBlocking(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C 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^2C 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^2C bus.

i2c_txNack()

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

i2c_txNackRepeat()

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

i2c_txNackBlocking()

Usage	`void i2c_ txNackBlocking(u32 reg)`
Parameters	[IN] `reg` struct of I^2C 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^2C register
Returns	[OUT] 1 bit acknowledge
Include	**driver/i2c.h**
Description	Receive an acknowledge from the I^2C slave.

i2c_rxData()

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

i2c_rxNack()

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

Machine Timer API Calls

machineTimer_setCmp()

Usage	`void machineTimer_setCmp(u32 p, u64 cmp)`
Include	**driver/machineTimer.h**
Parameters	[IN] `p` machine timer interrupt [IN] `cmp` machine timer compare register
Description	Set a timer value to trigger an interrupt.

machineTimer_getTime()

Usage	`u64 machineTimer_getTime(u32 p)`
Include	**driver/io.h**
Parameters	[IN] `p` machine timer interrupt
Returns	[OUT] timer value
Description	Gets the timer value.

machineTimer_uDelay()

Usage	`u64 machineTimer_uDelay(u32 usec, u32 hz, u32 reg)`
Include	**driver/io.h**
Parameters	[IN] `usec` microseconds [IN] `hz` core frequency [IN] `reg` machine timer interrupt
Description	Use the machine timer to make a delay.

PLIC API Calls

plic_set_priority()

Usage	`void plic_set_priority(u32 plic, u32 gateway, u32 priority)`
Include	**driver/io.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/io.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/io.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/io.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/io.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 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_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_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_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.

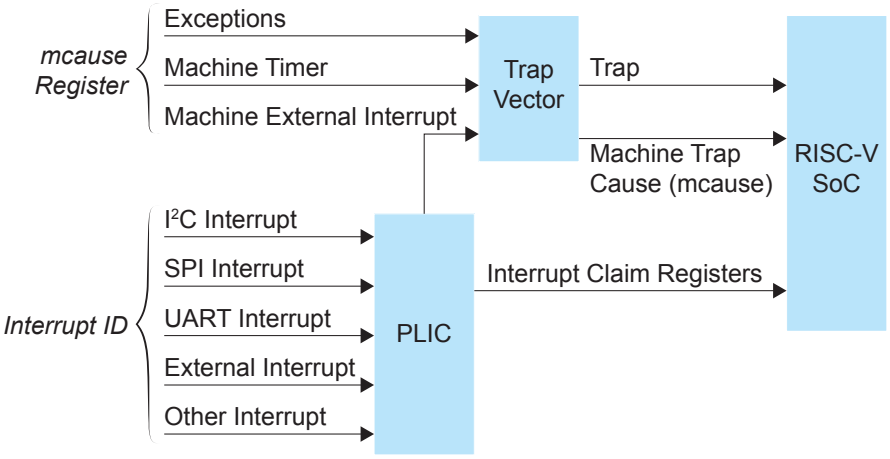
uart_writeAvailability()

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 4: Types of Interrupts

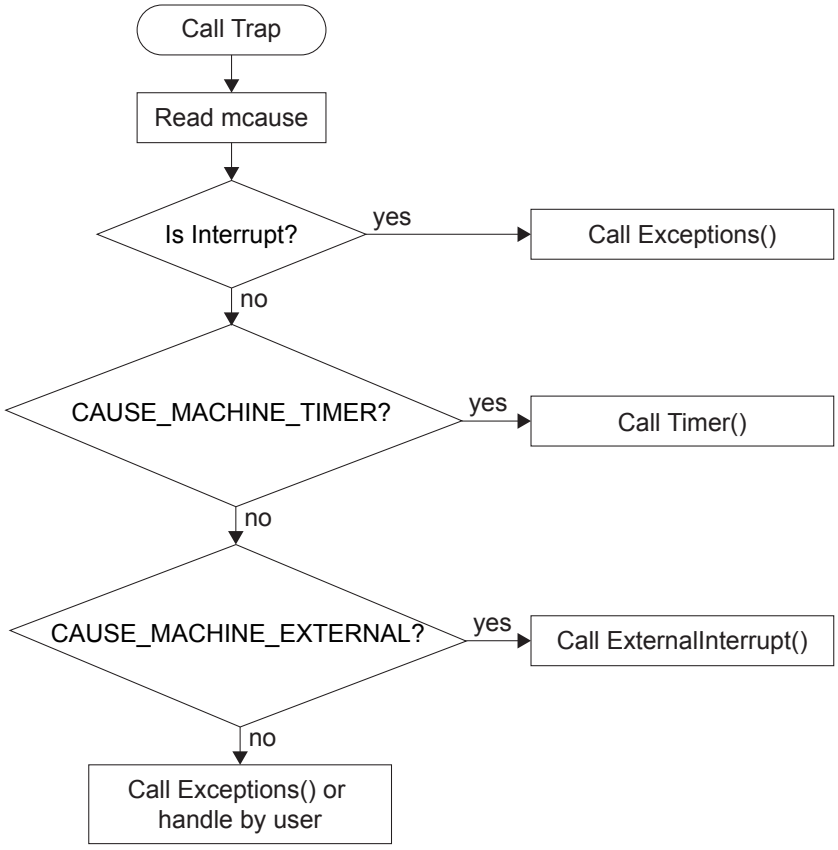


Trap Vectors

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 5: Handling Trap Vectors

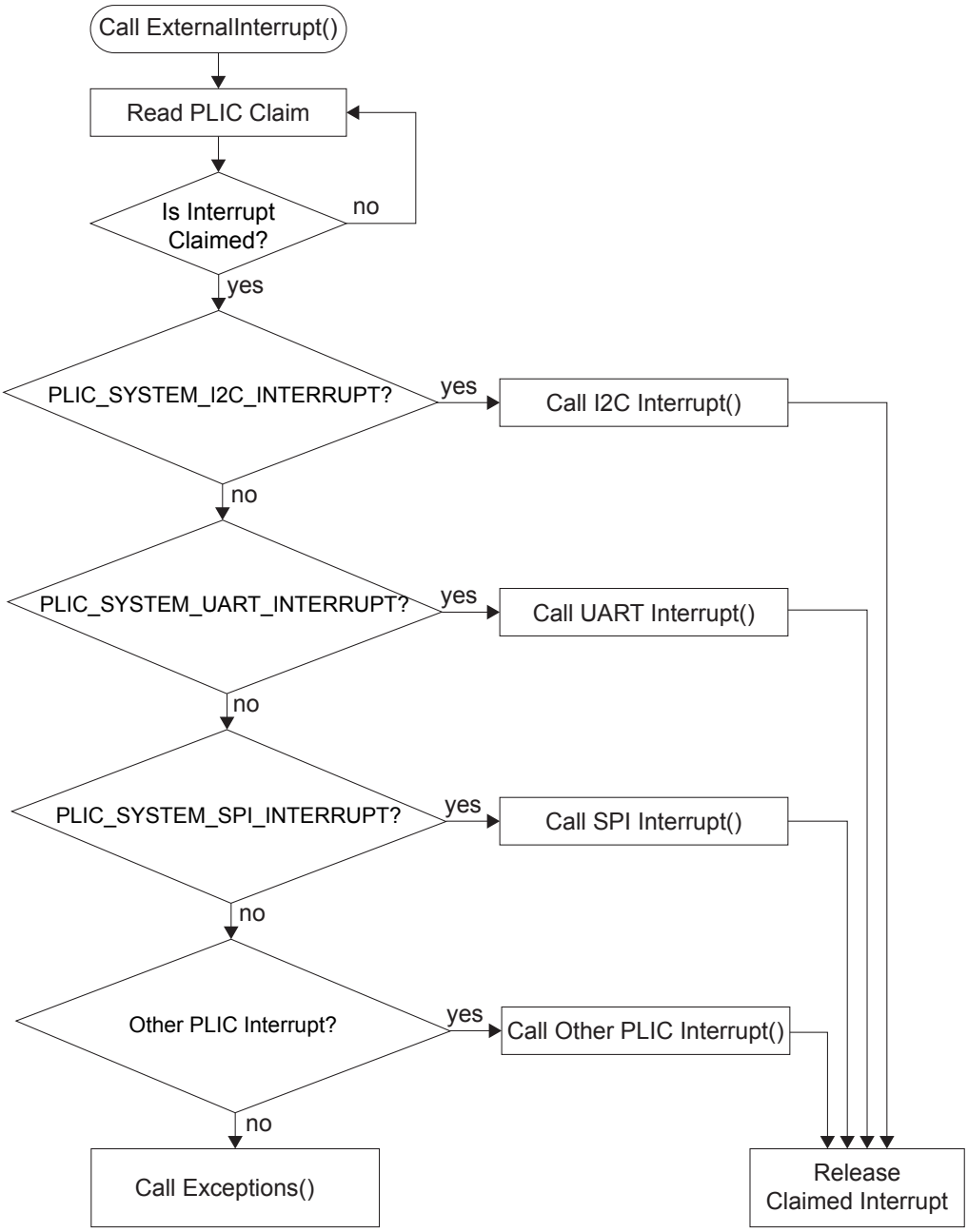


PLIC Interrupts

The PLIC collects external interrupts and is also used for CAUSE_MACHINE_EXTERNAL cases. Read the interrupt claim registers (PLIC claim) to identify the source of the external interrupt. Refer to Address Map on page 24for a list of the interrupt IDs.

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

Figure 6: Handling PLIC Interrupts



Revision History

Table 4: Revision History

Date	Version	Description
November 2021	1.2	Added information about the flow for handling interrupts in the API Reference chapter. (DOC-398) Updated UART and GPIO API calls. Added link to OpenJDK (DOC-457) Eclipse BSP environment variable no longer needed.
February 2021	1.1	Corrected instructions when launching Eclipse: choose option 4 Opal_T8. (DOC-384) Corrected description for APB3 RTL example. (DOC-384) Added a note that the SoC does not support floating-point numbers.
November 2020	1.0	Initial release.