易灵思

Edge Vision SoC User Guide

UG-EVSOC-v5.0
February 2023
www.elitestek.com

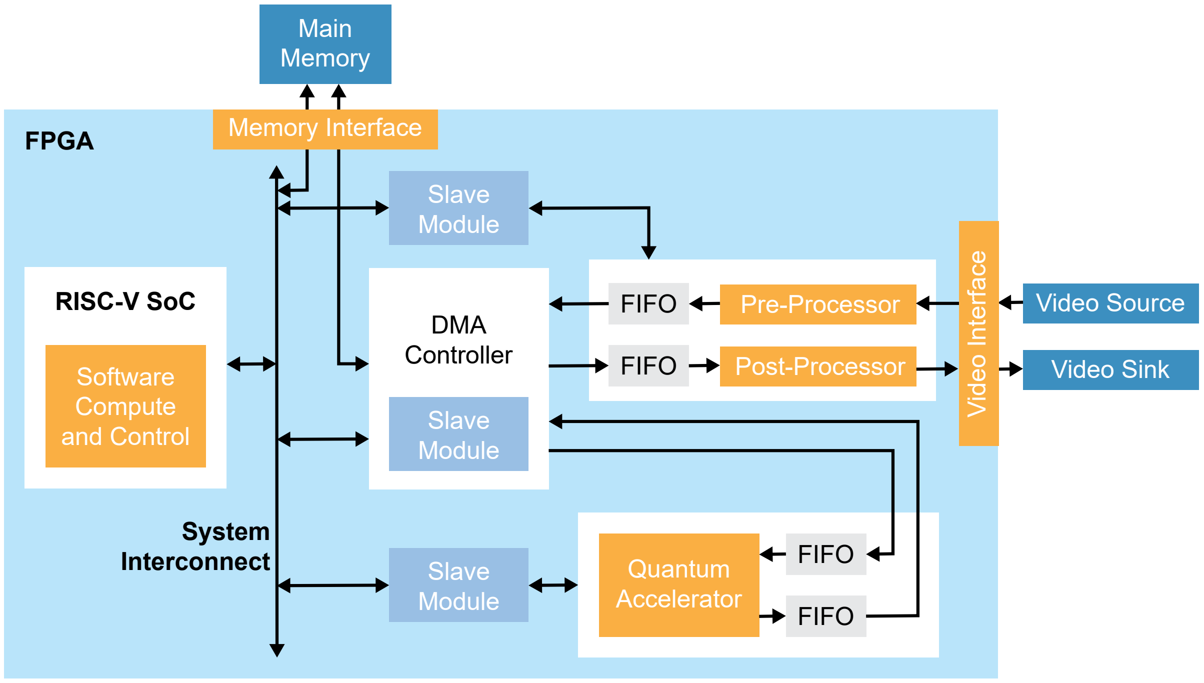
Contents

Introduction

Artificial intelligence (AI) and vision at the edge is an emerging domain that spans a wide range of applications such as automotive, industrial vision, retail, and robotics. The 易灵思® Edge Vision SoC framework, which uses a soft RISC-V SoC, facilitates faster time-to-market for edge vision applications. The Edge Vision SoC framework supports crucial domain-specific embedded software functions, hardware accelerator modules, memory and I/O related interfaces, peripherals, and controllers.

The Edge Vision SoC framework uses Quantum accelerators to facilitate hardware/software partitioning and achieve the desired performance. Within this framework, 易灵思 provides example designs for specific functions, such as video processing, AI object detection, machine learning, multi-camera fusion, etc.

Figure 1: Edge Vision SoC Framework



This user guide describes how to use example designs built with the Edge Vision SoC framework. These designs illustrate use cases for the framework, specifically, hardware/software co-design for video processing. Additionally, the designs show how you can control the FPGA hardware using software, that is, you can enable different hardware acceleration functions by changing firmware in the RISC-V processor.

These designs present these concepts in the context of video filtering functions; however, you can use the same designs with your own hardware accelerator block instead of the provided filtering functions. The designs help you explore accelerating computationally intensive functions in hardware and using software to control that acceleration.

Chapter 1

Quick Start

Contents:

Installation

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.

Important: To compile the projects described in this user guide, you must have the Efinity® software version 2022.2.322.1.8 or higher.

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

Install the Edge Vision SoC

To install the edge vision SoC files:

1. Create a directory called **riscv** at the root level of your file system.
2. Unzip the files into the **riscv** directory.

The files are organized in this directory structure:

Folder Name	Description	Board
T120F324_640_480	Image signal processing example design with 640x480 resolution project files	Trion T120 BGA324 Development Board
T120F324_1280_720	Image signal processing example design with 1280x720 resolution project files	Trion T120 BGA324 Development Board
T120F324_1280_720_dualCam	Dual-camera example design with 1280x720 resolution project files	Trion T120 BGA324 Development Board
T120F576_640_480	Image signal processing example design with 640x480 resolution project files	Trion T120 BGA576 Development Board
T120F576_1280_720	Image signal processing example design with 1280x720 resolution project files	Trion T120 BGA576 Development Board
T120F576_1280_720_dualCam	Dual-camera example design with 1280x720 resolution project files	Trion T120 BGA576 Development Board
Ti60F225_dsi	Image signal processing example design Mini-DSI panel example design project files	钛金系列 Ti60 F225 Development Board
Ti180M484_hdmi	Image signal processing example design with 1920x1080, 1280x720, and 640x480 resolution project files.	钛金系列 Ti180 M484 Development Board

Each project folder includes the following subfolders:
- **embedded_sw**—sw files
- **sim**—Testbench files (not available in dual-camera examples)
- **source**—RTL source files
- **ip**—IP core project files

Install the Efinity RISC-V Embedded Software IDE

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

To install the RISC-V Embedded Software IDE:

Windows

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

Linux

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

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

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

Note: You need to purchase a development kit to obtain a software license to download the Efinity software and Efinity RISC-V Embedded Software IDE.

Important: To compile the projects described in this user guide, you must have the Efinity® RISC-V Embedded Software IDE version 2022.2.3.

Set Up the Hardware

Required Hardware

Trion® T120 BGA576 Development Board and Trion® T120 BGA324 Development Board

The example designs use the following hardware from the Trion® T120 BGA324 Development Kit or Trion® T120 BGA576 Development Kit:

- Trion® T120 BGA576 Development Board or Trion® T120 BGA324 Development Board
- Raspberry Pi Camera Connector Daughter Card
- MIPI and LVDS Expansion Daughter Card
- One Raspberry Pi v2 camera module
- One 15-pin flat cable
- Micro-USB cable
- 12 V power adapter

You also need the following hardware, which you provide yourself:

- USB-to-UART module
- 3 jumper wires
- Jumpers
- 1080p monitor with HDMI connector
- HDMI cable
- Computer with Efinity® software installed

For the Dual-Camera example design, you also need a second Raspberry Pi v2 camera module, Raspberry Pi Camera Connector Daughter Card, and 15-pin flat cable.

钛金系列 Ti60 F225 Development Board

The example designs use the following hardware from the 钛金系列 Ti60 F225 Development Kit:

- 钛金系列 Ti60 F225 Development Board
- 1 Mini-DSI Panel Connector Daughter Card
- 1 Dual MIPI to DSI Converter Daughter Card
- 1 Dual Raspberry Pi Camera Connector Daughter Card
- 1 Raspberry Pi v2 camera module
- 1 Mini-DSI panel
- 1 15-pin flat cable
- 1 30-pin flat cable
- 1 USB type-C cable
- Universal AC to DC power adapter
- Jumpers

You also need the following hardware, which you provide yourself:

- Computer with Efinity® software installed

钛金系列 Ti180 M484 Development Board

The example designs use the following hardware from the 钛金系列 Ti180 M484 Development Kit:

- 钛金系列 Ti180 M484 Development Board
- 1 Raspberry Pi v2 Camera Module
- 1 Dual Raspberry Pi Camera Connector Daughter Card
- 1 HDMI Connector Daughter Card
- 2 22-pin flat cable
- 1 15-pin flat cable
- 1 USB type-C cable
- FMC-to-QSE Adapter card
- Universal AC to DC power adapter

You also need the following hardware, which you provide yourself:

- Computer with Efinity® software installed

Set Up the Development Board

The following figures show the hardware setup steps. If you have not already done so, attach standoffs to the board.

Important: Make sure that the development board is turned off before connecting any cards or cables.

Trion Development Boards

Figure 2: Hardware Setup Trion® T120 BGA324 Development Board

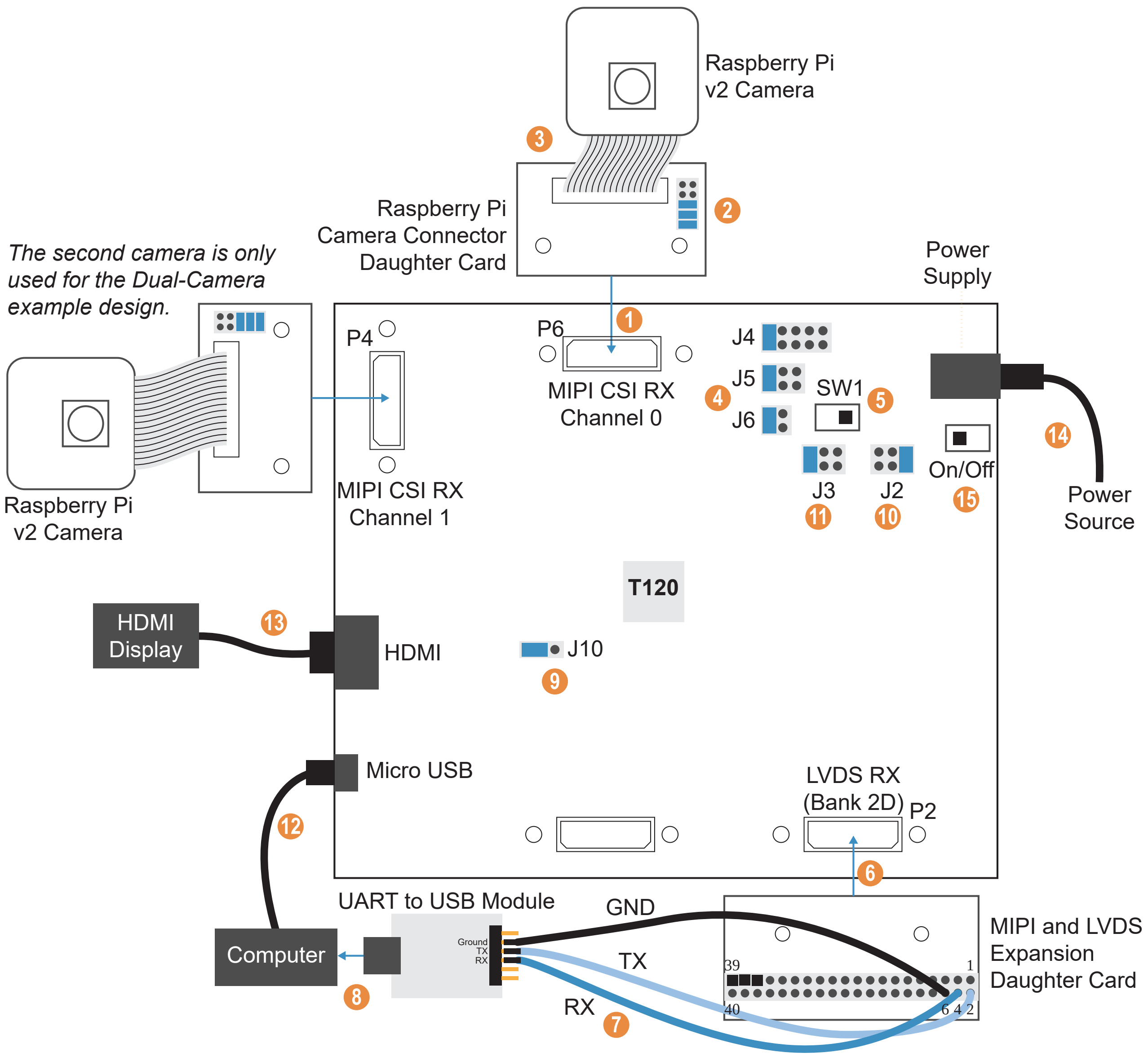
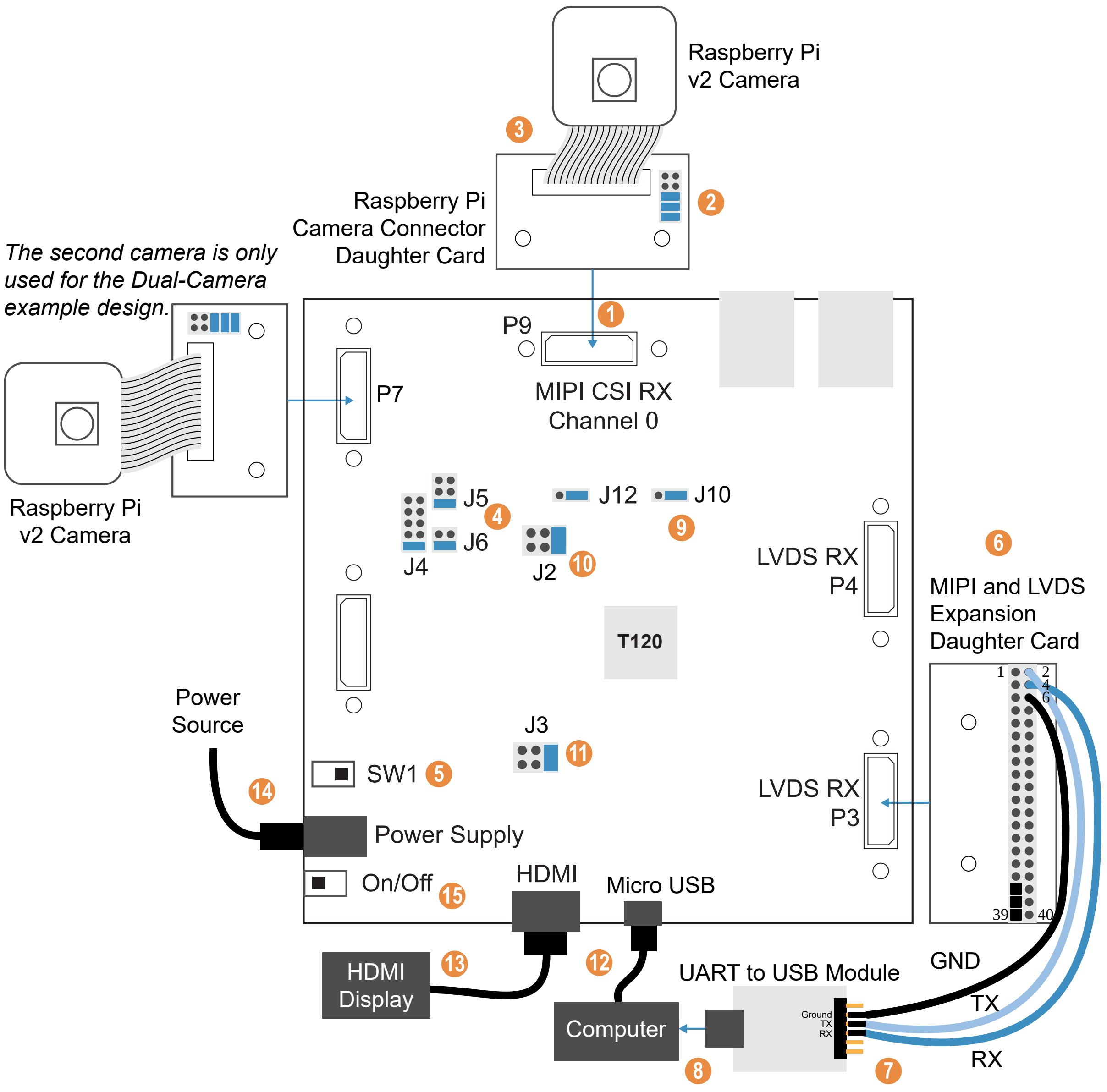


Figure 3: Hardware Setup Trion® T120 BGA576 Development Board



Set Up the Camera

1. Connect the Raspberry Pi Camera Connector Daughter Card to the board at P6.
2. On the daughter card, connect the following pins with jumpers: 1 - 2, 3 - 4, and 5 - 6.
3. Connect the Raspberry Pi v2 camera module to the daughter card using the 15-pin flat cable.
4. Connect jumpers to set the power sequence:
   - *VSUP1 (J4)*—Connect pins 9 - 10 (default)
   - *VSUP2 (J5)*—Connect pins 5 - 6 (default)
   - *VSUP3 (J6)*—Connect pins 3 - 4 (default)
5. Slide SW1 to position 3, which enables the power up sequence circuit for the MIPI CSI-2 cameras.

Note: If you are using the Dual-Camera example design, set up the second camera by following steps 1 - 3 and connecting the daughter card to:
- P4 for Trion® T120 BGA324 Development Board
- P7 for Trion® T120 BGA576 Development Board

Set Up the UART Module

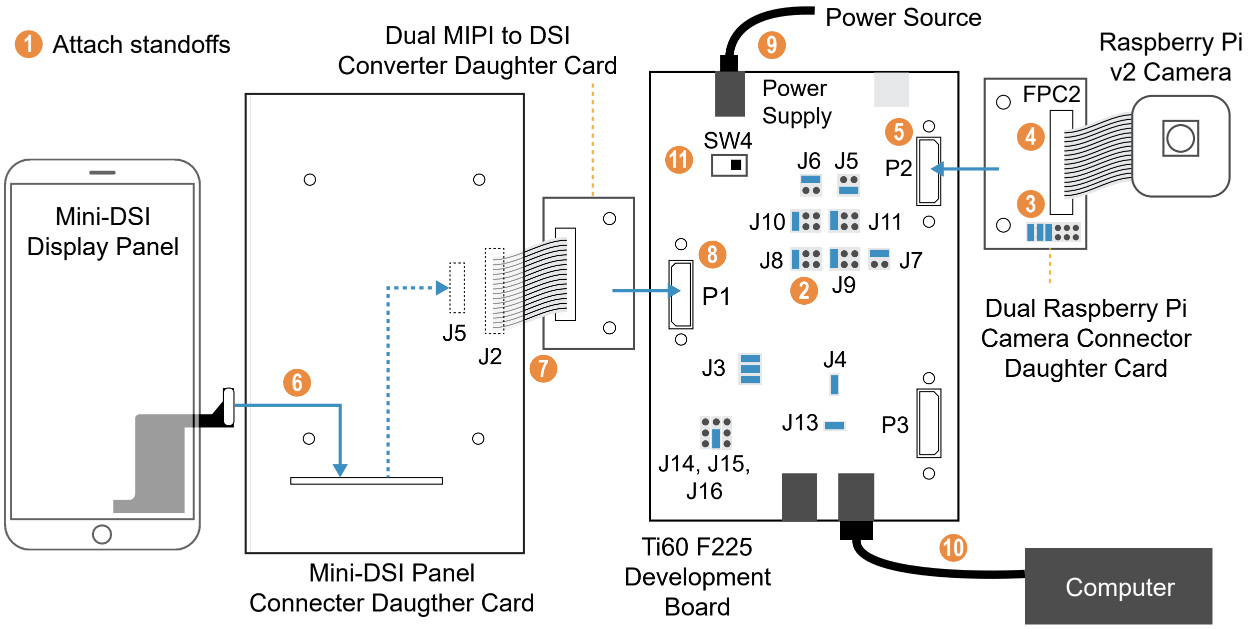
6. Connect the MIPI and LVDS Expansion Daughter Card to the board at:
   - P2 for Trion® T120 BGA324 Development Board
   - P3 for Trion® T120 BGA576 Development Board
7. Connect jumper wires to the daughter card and UART module:
   - RX to pin 4
   - TX to pin 2
   - GND to pin 6
8. Connect the UART module to your computer.

Connect Cables and Jumpers

9. On header J10, connect pins 2 - 3 with a jumper (default) to use the on-board oscillator.
10. On J2, connect pins 1 - 2 (default).
11. On J3, connect pins 5 - 6.
12. Connect a USB cable to the board and to your computer.
13. Connect an HDMI cable to the board and to your HDMI display.
14. Connect the 12 V power supply to the board connector and to a power source.
15. Turn on the board using the power switch.

钛金系列 Development Boards

Figure 4: Hardware Setup 钛金系列 Ti60 F225 Development Board



1. Attach standoffs to the board if you have not already done so.
2. On the 钛金系列 Ti60 F225 Development Board, connect the following jump

Header	Net Name	State	Short Pins
J3	VCC	0.95V	1 and 2
J3	VCC	0.95V	3 and 4
J3	VCC	0.95V	5 and 6
J4	VCCAUX	1.8V	1 and 2
J5	VCCIO33_TR	3.3V	1 and 2
J6	VCCIO33_TL	1.8V	3 and 4
J7	VCCIO33_BR	1.8V	3 and 4
J8	VCCIO3A	1.2V	5 and 6
J9	VCCIO3B	1.2V	5 and 6
J10	VCCIO4A	1.2V	5 and 6
J11	VCCIO4B	1.2V	5 and 6
J13	SPI_ENA	Short	1 and 2
J14	FX3_PMODE0	Float	Unconnected
J15	FX3_PMODE1	Pull-up	1 and 2
J16	FX3_PMODE2	Float	Unconnected

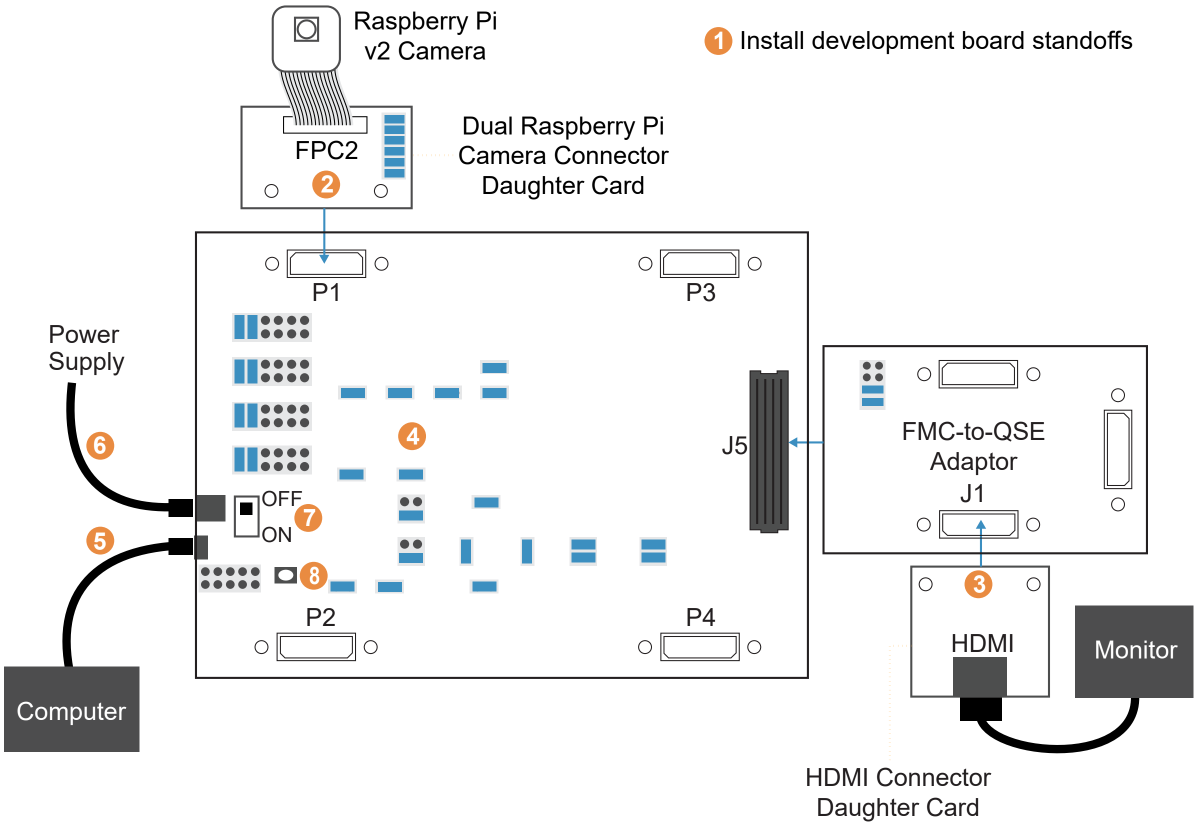
3. On the Raspberry Pi Camera Connector Daughter Card, connect the following pins with jumpers: 1 - 2, 3 - 4, and 5 - 6.
4. Connect the Raspberry Pi v2 camera module to the FPC2 connector of the Raspberry Pi Camera Connector Daughter Card using the 15-pin flat cable.
5. Connect the Raspberry Pi Camera Connector Daughter Card to the P2 header of the 钛金系列 Ti60 F225 Development Board

6. Connect the Mini-DSI Display panel to header J5 of the Mini-DSI Panel Connector Daughter Card using the attached 48-pin flat cable.

   Note: Insert the Mini-DSI Display cable through the small opening to connect to header J5. The header is at the bottom of the board.

7. Connect connector J2 of Mini-DSI Panel Connector Daughter Card to connector J2 of Dual MIPI to DSI Converter Daughter Card using a flat cable.
8. Connect the Dual MIPI to DSI Converter Daughter Card to the P1 header of the 钛金系列 Ti60 F225 Development Board.
9. Ensure the board power switch is turned off, then connect the 12 V power cable to the board connector and to a power source.
10. Connect the USB header J12 to USB port of your computer.
11. Turn on the board's power switch.

Figure 5: Hardware Setup 钛金系列 Ti180 M484 Development Board


Raspberry Pi v2 Camera
1 Install development board standoffs
FPC2
2
Dual Raspberry Pi Camera Connector Daughter Card
P1
P3
Power Supply
6
4
J5
FMC-to-QSE Adaptor
J1
OFF
ON
7
5
8
3
P2
P4
HDMI
Monitor
Computer
HDMI Connector Daughter Card

Important: Always switch off the power supply and board's power switch before attaching or detaching cables and daughter cards.

1. Install standoffs on the 钛金系列 Ti180 M484 Development Board as described in Installing Standoffs.
2. Attach the Dual Raspberry Pi Camera Connector Daughter Card to the P1 connector of the 钛金系列 Ti180 M484 Development Board.
3. Attach the HDMI Connector Daughter Card to the J5 connector of the 钛金系列 Ti180 M484 Development Board.
4. Ensure that all boards have the following jumper settings:

Board	Header	Pins to Connect
钛金系列 Ti180 M484 Development Board	J9	N.C.
	J10, J11, J12, J13, PT1, and PT17	1 - 2 and 3 - 4
	PT2, PT3, PT4, PT5, PT6, PT7, PT8, PT9, PT10, PT11, PT12, PT13, PT14, PT15, and PT16	1 - 2
FMC-to-QSE Adapter Card	J5	5 -6 and 7 - 8
Dual Raspberry Pi Camera Connector Daughter Card	J1	1 - 2, 3 - 4, 5 - 6, 7 - 8, 9 - 10, and 11 - 12

5. Connect the USB header to a USB port of your computer.
6. Ensure the power supply and board power switch (PSW1) are turned off, then connect the 12 V power cable to the board connector and a power source.
7. Turn on the power supply and the board's power switch (PSW1).
8. After turning on the board, press pushbutton SW1 (`CRESET_N`).

The board LEDs light up to indicate the following board status.

Table 1: Board LED Outputs

LED	Description
LED11, LED12, and LED13 turned on	Power good
LED1 turned on	FPGA configuration done
LED7 turned on	DDR configuration done
LED2 turned on	DDR memory test done

Note: If the monitor displays an abnormal video stream, press SW1 to reset the example design.

You can use the lens adjustment tool included in the kit to set the Raspberry Pi v2 camera module focus when the video is streaming.

Installing USB Drivers

To program Trion® or 钛金系列 FPGAs using the Efinity® software and programming cables, you need to install 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. Refer to the 易灵思 development kit user guide for details on installing drivers for the development board.

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

Installing Drivers on Windows

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

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

Board	Interface to Install Driver
Trion® T120 BGA324 Development Board Trion® T120 BGA576 Development Board	Install drivers for all interfaces (0 and 1).
钛金系列 Ti60 F225 Development Board	Install drivers for interfaces 0 and 1 only. Windows automatically installs a driver for interfaces 2 and 3 when you connect the board to your computer.
钛金系列 Ti180 M484 Development Board	Install driver for interface 1 only.

To install the driver:

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

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

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

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

Installing Drivers on Linux (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

To use the example designs for the Edge Vision SoC framework, you must program the RTL design into the board.

Note: Efinity® projects are located in the **efinity_project** directory.

1. There are two ways of obtaining the Edge Vision SoC framework bitstream:
   - Open the project (**.xml**) for the design you want to use, review it, and compile it in the Efinity® software.
   - Use the provided pre-compiled bitstream file, **edge_vision_soc.hex** in the **efinity_project** directory.

   Note: The provided bitstream file is compiled for SPI active configuration mode. Refer to the Efinity Software User Guide for more information about generating bitstreams for other configuration modes.

2. Use the Efinity® Programmer and SPI active mode to download the bitstream file to your board.

   Note: For SPI Flash, Ti180 M484 only supports the download of the bitstream file to the board through **SPI Active using JTAG Bridge** programming mode.

   Additional Information

   To program the FPGA bitstream into the SPI flash through **SPI Active using JTAG Bridge**:

   a. In the Efinity Programmer, select the **SPI Active using JTAG Bridge** programming mode.
   b. In Image, choose **Select Image File > evsoc_ispExample.hex**.
   c. In Auto configure JTAG Bridge Image, choose **Select Image File > jtag_spi_flash_loader_dual.bit** (available in efinity_project directory) file.
   d. Ensure that the **Starting Flash Address** is set to `0x000000`.
   e. Click **Start Program**.

   Note: You use SPI active mode because you need to reset the FPGA. 易灵思 also includes the **edge_vision_soc.bit** file to be used with JTAG mode. Using JTAG mode requires you to reprogram the bitstream into the board when you reset the board.

3. If you are using SPI active mode, 易灵思 recommends that you press the CRESET pushbutton on the development board to reset the FPGA. This reset ensures that the external memory initialization happens before the user application runs.

Learn more: Instructions on how to use the Efinity® software is available in the Support Center.

Using Efinity RISC-V Embedded Software IDE

You use Eclipse to manage your software projects. The following sections explain how to set up your Efinity RISC-V Embedded Software IDE environment, create a project and build it. Additionally, this section walks you through how to lauch and set up the debugger for use with the Sapphire SoC.

Launch Efinity RISC-V Embedded Software IDE

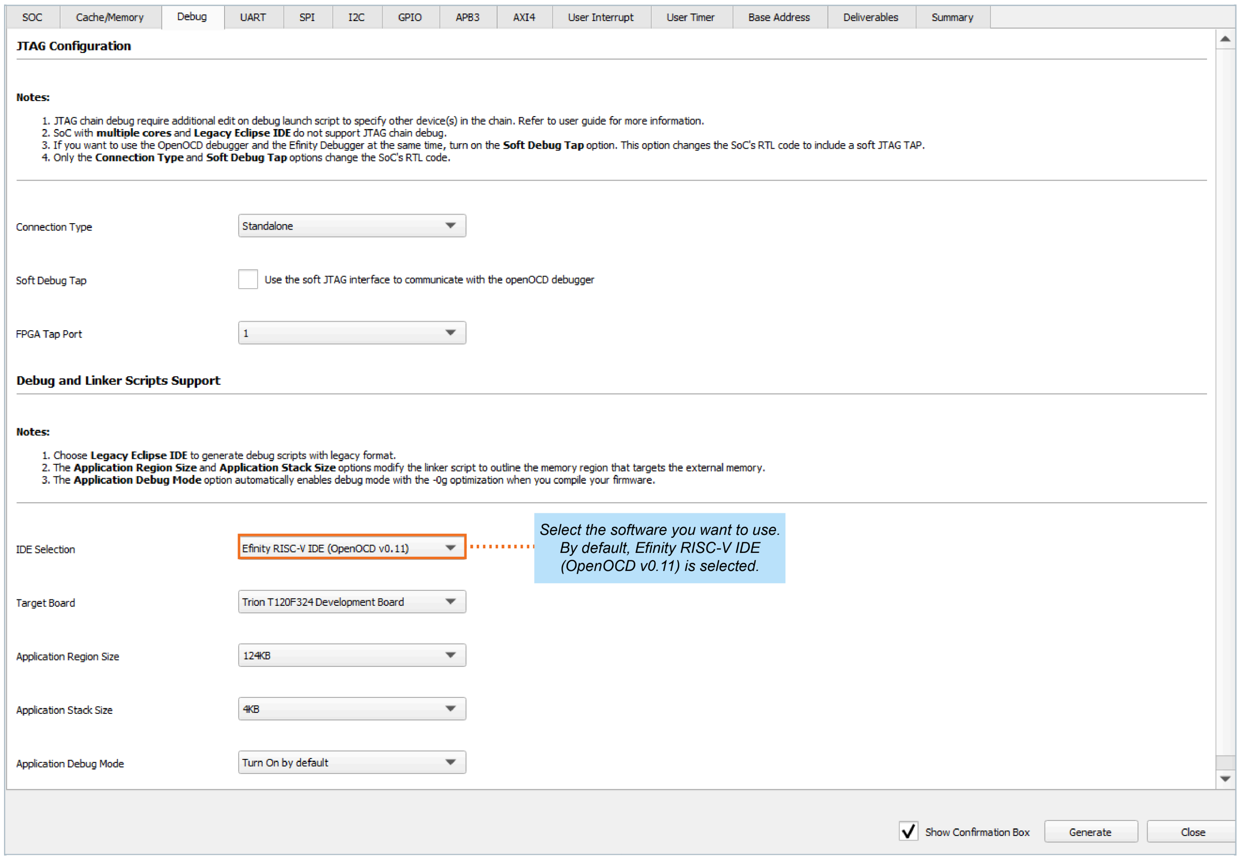
Sapphire SoC IDE Backward Compatibility

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

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

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

Figure 6: IDE Selection Parameter in the Debug tab



JTAG Configuration

Notes:

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

Connection Type	Standalone
Soft Debug Tap	☐ Use the soft JTAG interface to communicate with the openOCD debugger
FPGA Tap Port	1

Debug and Linker Scripts Support

Notes:

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

IDE Selection	Efinity RISC-V IDE (OpenOCD v0.11)
Target Board	Trion T120F324 Development Board
Application Region Size	124KB
Application Stack Size	4KB
Application Debug Mode	Turn On by default

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

Launching the Efinity RISC-V Embedded Software IDE

Windows

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

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

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

Linux

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

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

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

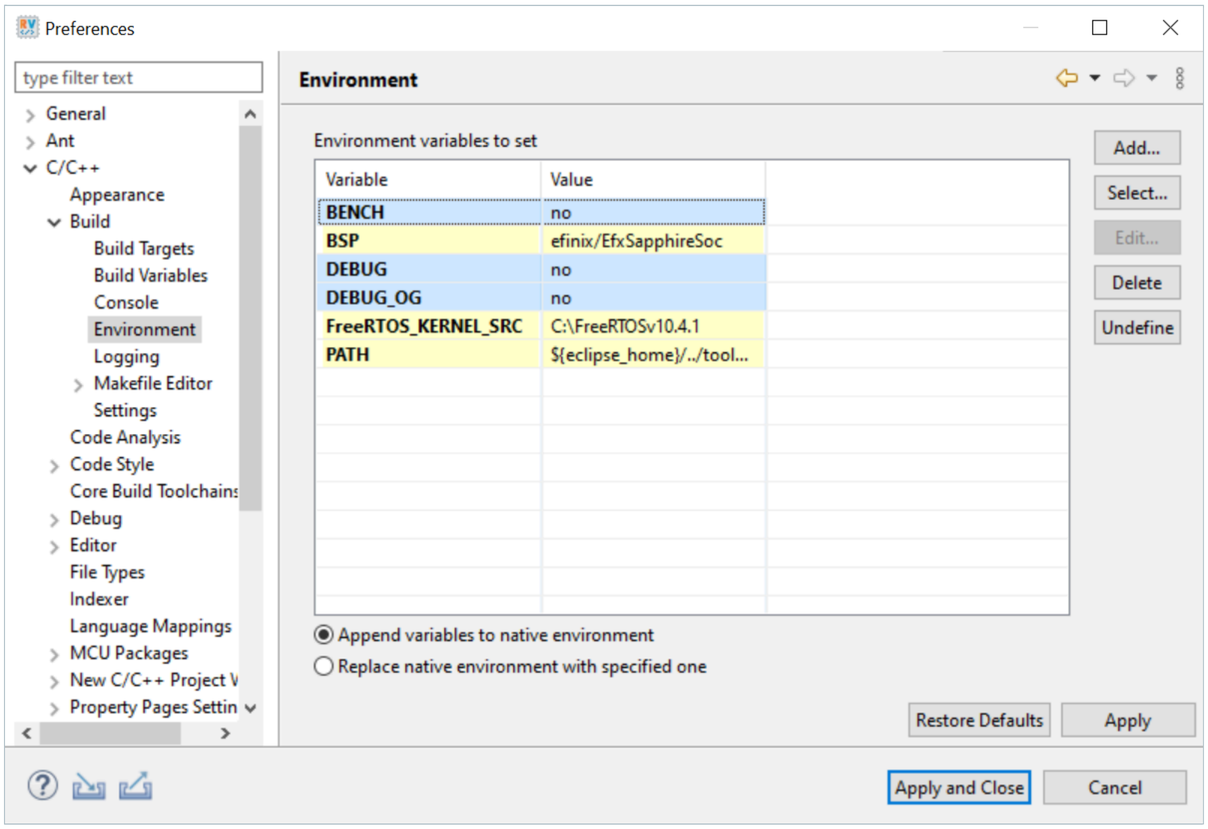
Note: You can choose any location for your workspace. If you have selected a folder that does not exist, the IDE automatically creates a folder for you.

Set Up the Efinity RISC-V Embedded Software IDE

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

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

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



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

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

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

Table 2: Environment Settings for Preferences Window

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

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

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

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

include
include
include
```

Note: Refer to this link: https://gcc.gnu.org/onlinedocs/gcc-8.3.0/gcc/Optimize-Options.html for more information regarding the options that control the optimization settings.

Create, Import, and Build a Software Project

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

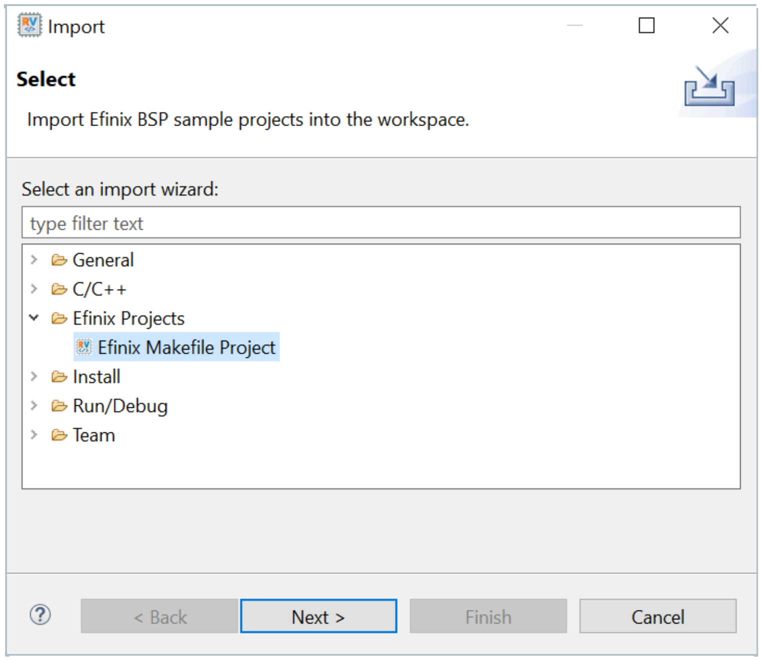
Note: For more details on how to create a new project, refer to Sapphire RISC-V SoC Hardware and Software User Guide.

Import Sample Projects

易灵思 provides sample projects to help you get started with EVSoC frameworks. The followings steps use EVSoC examples to explain the steps to import sample project into the IDE:

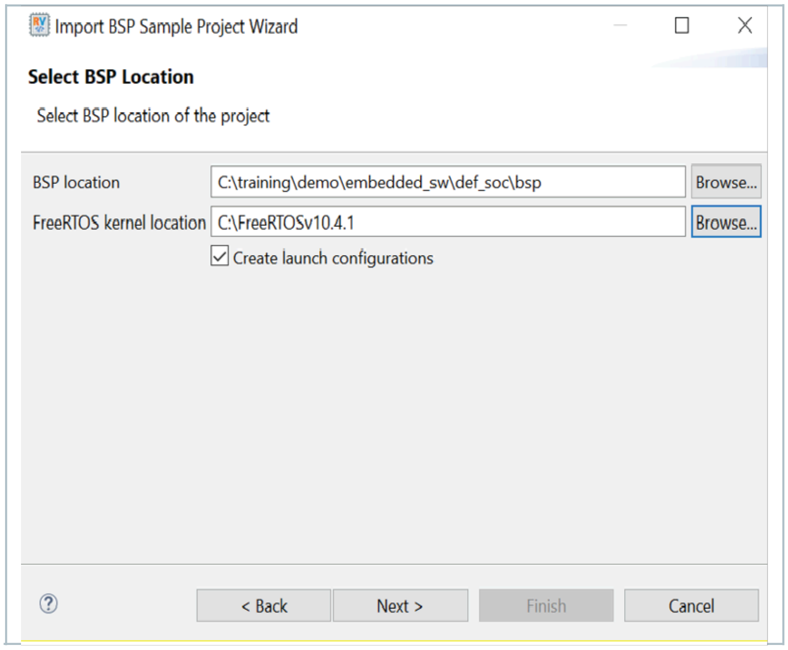
1. In the **Project Explorer**, click on **Import Projects...** to open the **Import** wizard.
2. In the **Import** wizard, select 易灵思 **Makefile Project** in 易灵思 **Projects** and click **Next**.

Figure 7: Import Wizard



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

Figure 8: Import BSP Sample Project Wizard



Select BSP Location

Select BSP location of the project

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

FreeRTOS kernel location: C:\FreeRTOSv10.4.1

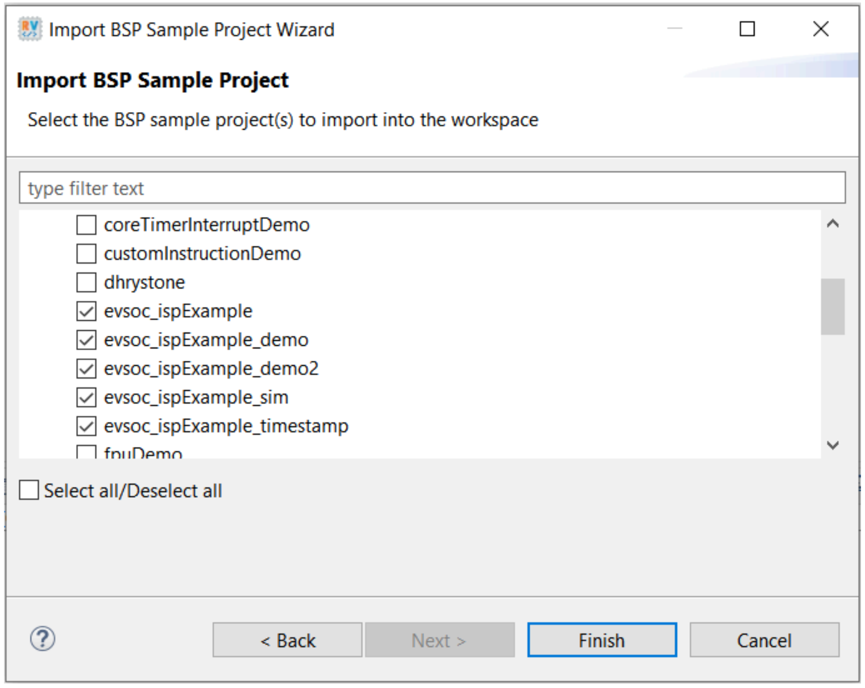
☑ Create launch configurations

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

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

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

Figure 9: Import BSP Sample Project Wizard – List of Projects


Import BSP Sample Project Wizard

Import BSP Sample Project

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

type filter text

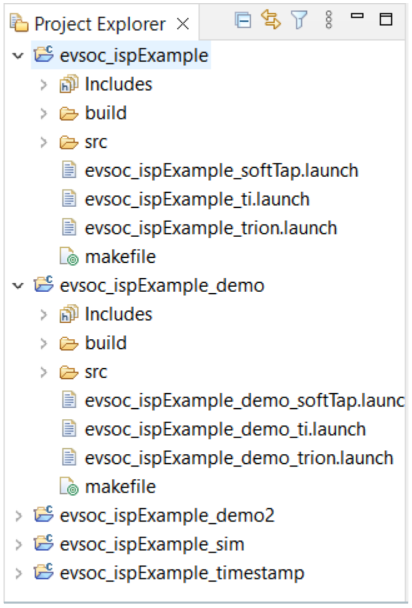
- [ ] coreTimerInterruptDemo
- [ ] customInstructionDemo
- [ ] dhrystone
- [x] evsoc_ispExample
- [x] evsoc_ispExample_demo
- [x] evsoc_ispExample_demo2
- [x] evsoc_ispExample_sim
- [x] evsoc_ispExample_timestamp
- [ ] fpuDemo

- [ ] Select all/Deselect all

Note: If you have custom programs that need to be exported to the IDE, you may either copy the programs into existing folders (FreeRTOS or Standalone) or you can create a folder at the same level as FreeRTOS and Standalone folders. Automatically, the IDE identifies the folder as a sub-category.

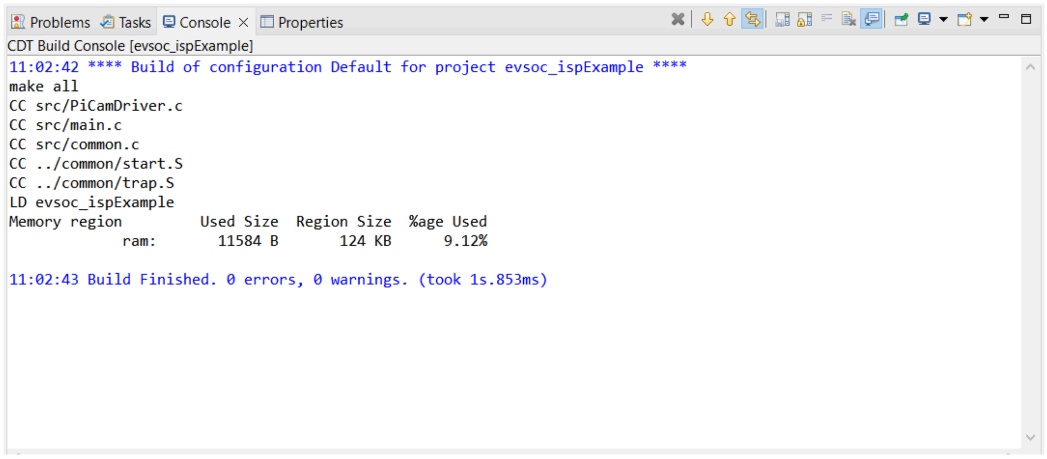
The selected sample projects are imported into the listed workspace in the **Project Explorer** pane.

Figure 10: Project Explorer Pane showing all the Imported Projects


```
Project Explorer
v evsoc_ispExample
    > Includes
    > build
    > src
      evsoc_ispExample_softTap.launch
      evsoc_ispExample_ti.launch
      evsoc_ispExample_trion.launch
      makefile
v evsoc_ispExample_demo
    > Includes
    > build
    > src
      evsoc_ispExample_demo_softTap.launc
      evsoc_ispExample_demo_ti.launch
      evsoc_ispExample_demo_trion.launch
      makefile
> evsoc_ispExample_demo2
> evsoc_ispExample_sim
> evsoc_ispExample_timestamp
```

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

Figure 11: Output Console showing the evsoc_ispExample Standalone Project Built Successfully



```
11:02:42 **** Build of configuration Default for project evsoc_ispExample ****
make all
CC src/PiCamDriver.c
CC src/main.c
CC src/common.c
CC ../common/start.S
CC ../common/trap.S
LD evsoc_ispExample
Memory region         Used Size  Region Size  %age Used
             ram:       11584 B       124 KB      9.12%

11:02:43 Build Finished. 0 errors, 0 warnings. (took 1s.853ms)
```

Build

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

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

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

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

Using a UART Terminal

When working with the Image Signal Processing design in Eclipse, it is helpful to have a Telnet terminal open to send commands and view messages via the UART interface. The following sections explain how to enable Telnet in Windows and how to open a UART terminal.

Enable Telnet on Windows

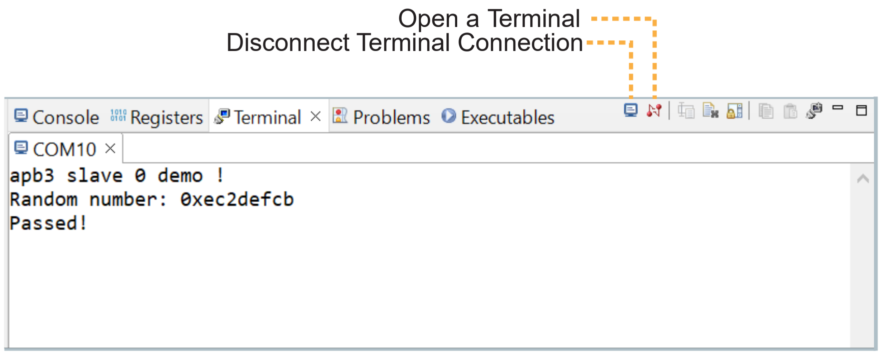
Windows does not have telnet turned on by default. Follow these instructions to enable it:

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 terminal in the Efinity RISC-V Embedded Software IDE, to connect to the UART. These instructions explain how to use the built-in terminal while the others are similar.

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



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

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

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

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

Run the Application with Efinity RISC-V Embedded Software IDE

With the development board programmed and the software built, you are ready to configure the Efinity RISC-V Embedded Software IDE and run the application binary. These instructions use the **evsoc_ispExample** example to explain the steps required.

Launch the Run Script

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

To launch the evsoc_ispExample project:

1. Right-click on **evsoc_ispExample** > **evsoc_ispExample<family>.launch**.
2. Select **Run As** > **evsoc_ispExample_<family>**. Efinity RISC-V Embedded Software IDE launches the OpenOCD Debugger for the project.

View the Results

With the board programmed and the firmware downloaded to the RISC-V processor, you should see RGB video displayed on the display.

Launch the Debug Script

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

Table 3: Debug Configurations

Launch Script	Description
evsoc_ispExample_trion.launch	Debugging software on Trion® development boards.
evsoc_ispExample_ti.launch	Debugging software on 钛金系列 development boards
evsoc_ispExample_softTap.launch	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.)

Note: You may receive TAP ID warnings in the Eclipse console when trying to debug a device with **softTap** enabled that is not the Trion T120F324 device. These warnings do not affect the debugging process. To remove these warnings, refer to the troubleshooting guide, Warning when Debug with softTap on page 71.

To debug the evsoc_ispExample project:

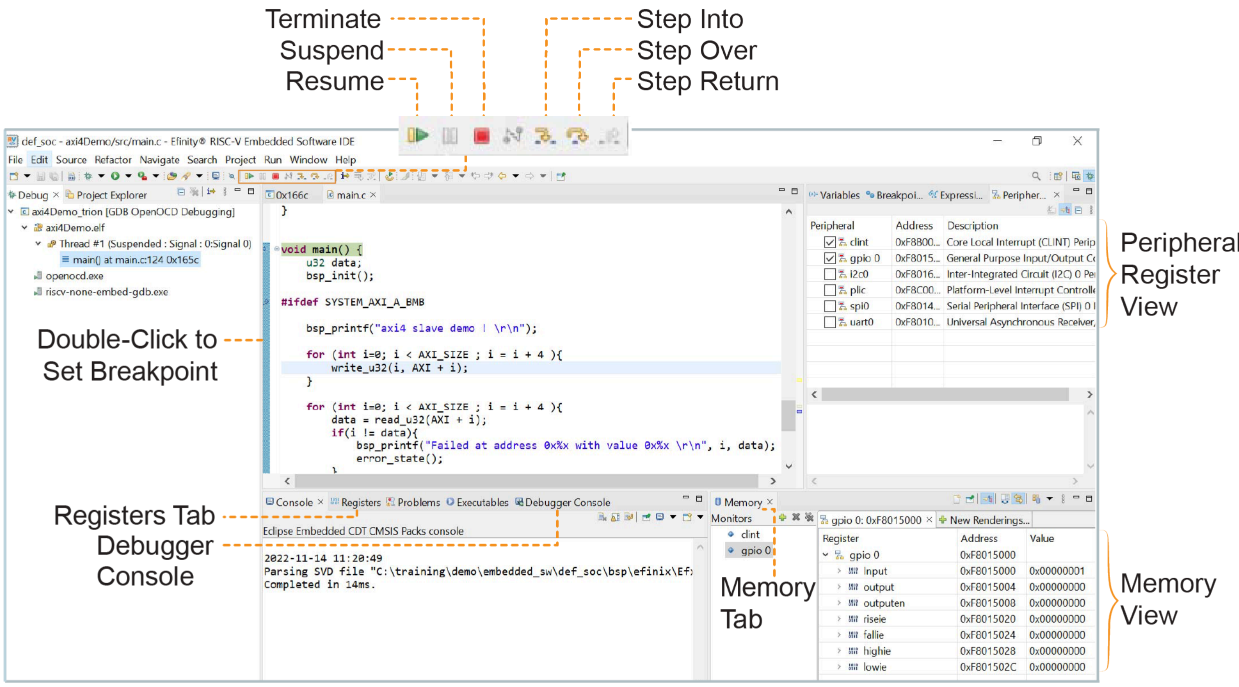
1. Right-click **evsoc_ispExample** > **evsoc_ispExample_<family>.launch**.
2. Choose **Debug As** > **evsoc_ispExample** > **evsoc_ispExample_<family>**. Efinity RISC-V Embedded Software IDE launches the OpenOCD debugger for the project.
3. Click **Debug**.
4. **Confirm Perspective Switch** window would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

Debug

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

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

Figure 12: Perform Debugging



```
}

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

#ifdef SYSTEM_AXI_A_BMB

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

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

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

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

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

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

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

Using the Efinity® Debugger and OpenOCD Simultaneously

The Efinity® Debugger uses the hard JTAG TAP interface. Out of the box, the Edge Vision SoC Framework 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 need to modify the RTL design to 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).

Note: 易灵思 recommends you use the C232HM-DDHSL-0 FTDI chip cable rather than a JTAG mini-module because the Edge Vision SoC includes the debug configuration file for the cable.

To use the two debuggers simultaneously:

1. Open the Efinity project you want to use.
2. Open the **edge_vision_soc.v** file.
3. Add the following line before the `edge_vision_soc` module declaration:

```
`define SOFT_TAP
```

4. Save and close the **edge_vision_soc.v** file.
5. Regenerate the Sapphire IP with **Soft Debug Tap** parameter enabled.
6. Open the Interface Designer and make these changes:

   a. Remove the JTAG User Tap block.

   b. Create these GPIO blocks for the soft JTAG pins:
      - `io_jtag_tck`—Configure as input; enable Schmitt trigger.
      - `io_jtag_tdi`—Configure as input.
      - `io_jtag_tdo`—Configure as output.
      - `io_jtag_tms`—Configure as input.

      Note: Make sure that the instance names and pin names match the soft JTAG I/O ports in the `edge_vision_soc` module in the **edge_vision_soc.v** file (top-level RTL).

   c. In the Resource Assigner, assign the soft JTAG ports to the I/O pins you want to use. For example, for the Trion® T120 BGA324 Development Board and 钛金系列 Ti180 M484 Development Board, you can use these resource assignments, which are connected to header J11:

Figure 13: Connecting the FTDI Module or C232HM-DDHSL-0 Cable (T120 BGA324)

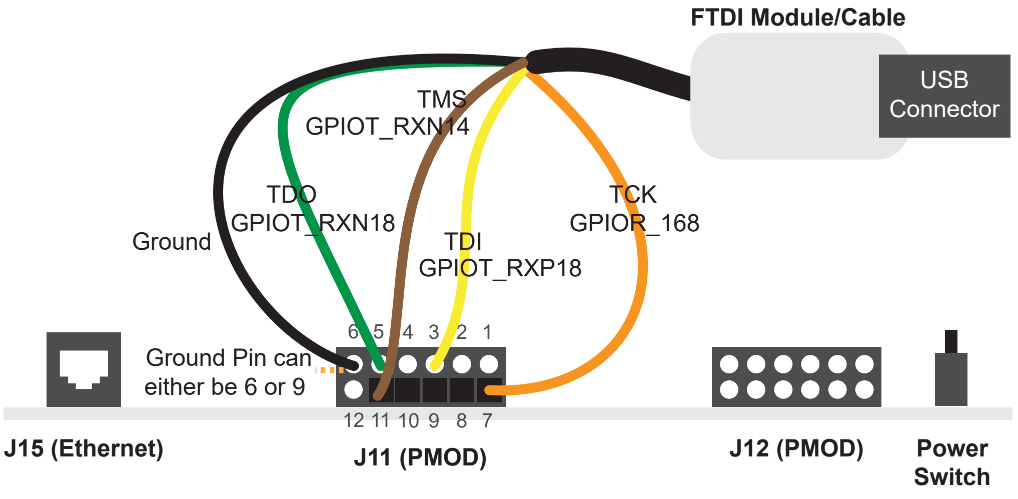


Table 4: FTDI to PMOD Connections (Trion T120 BGA324)

Signal	Resource	J11 Pin
`io_jtag_tck`	GPIOR_168	7
`io_jtag_tdi`	GPIOT_RXP18	3
`io_jtag_tdo`	GPIOT_RXN18	5
`io_jtag_tms`	GPIOT_RXN14	11
`io_jtag_gnd`	–	6/9

Figure 14: Connecting the FTDI Module or C232HM-DDHSL-0 Cable (Ti180 M484)

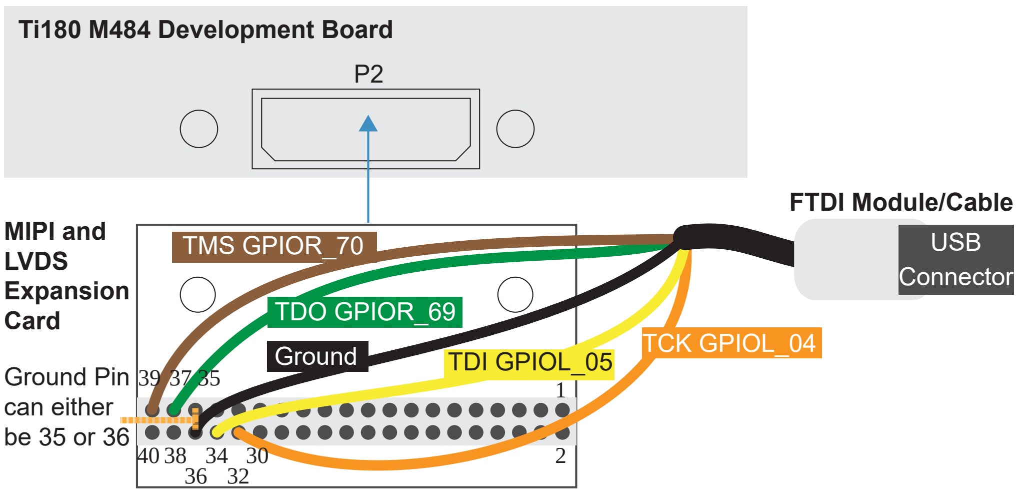


Table 5: FTDI to Daughter Card Connections (钛金系列 Ti180 M484)

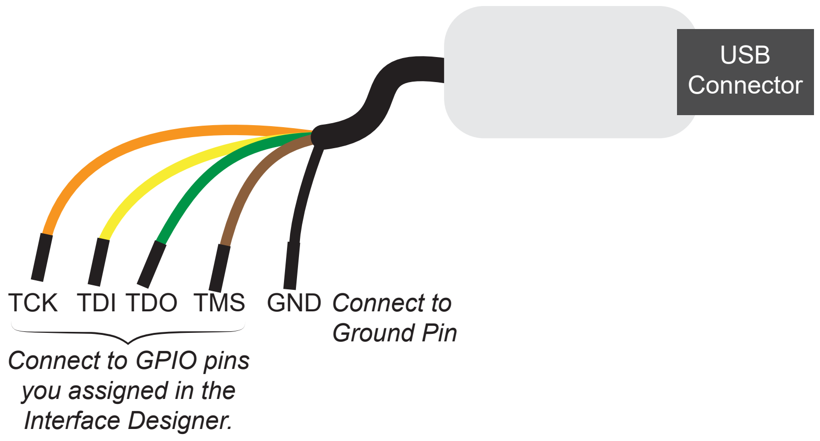
Signal	Resource	MIPI and LVDS Expansion Daughter Card (P2) Pin
`io_jtag_tck`	GPIOL_04	32
`io_jtag_tdi`	GPIOL_05	34
`io_jtag_tdo`	GPIOR_69	37
`io_jtag_tms`	GPIOR_70	39
`io_jtag_gnd`	-	35/36

Note: When using the C232HM-DDHSL-0 FTDI chip cable, make sure that the assigned I/O pins use the 3.3V I/O standard and the I/O pin you assign for `io_jtag_tck` supports Schmitt Trigger.

d. Check the design to ensure that you have connected everything correctly. The Interface Designer issues warnings or error messages if it finds design issues.
e. Save and exit the Interface Designer.

7. Add a Virtual I/O or Logic Analyzer core to your design. For instructions, refer to the Debugging chapter in the Efinity Software User Guide.
8. Compile the design.
9. Download the resulting bitstream file to your board using the Efinity® Programmer and a USB cable connected to the board.
10. Connect the cable to your board using the following figure as a guide. For example, for the Trion® T120 BGA324 Development Board, you can connect ground to pin 9 on header J11.

Figure 15: Connecting the C232HM-DDHSL-0 Cable


USB Connector
TCK
TDI
TDO
TMS
GND
Connect to Ground Pin
Connect to GPIO pins you assigned in the Interface Designer.

Note: If you have not already done so, install the driver for the FTDI cable as described in Installing USB Drivers on page 16.

11. Open your Eclipse project.
12. Run or debug the software with the OpenOCD debugger using the **default_softTap** launch configuration. Refer to Launch the Debug Script on page 29 for complete instructions.
13. Open the Efinity® Debugger to perform hardware debugging.

Deploying the Application Binary

You can deploy the application binary in two ways:

- Boot using the OpenOCD debugger
- Boot from a flash device

This section explains the steps to boot from flash device. The steps to boot using OpenOCD debugger are explained in Launch the Run Script on page 29.

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

Boot from a Flash Device

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

To boot from a SPI flash device:

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

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

Note: Refer to Boot Sequence of Sapphire RISC-V SoC Hardware and Software User Guide for other possible boot scenarios.

Boot from the OpenOCD Debugger

To boot from the OpenOCD debugger:

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

Note: Refer to Launch the Debug Script on page 29 for complete instructions.

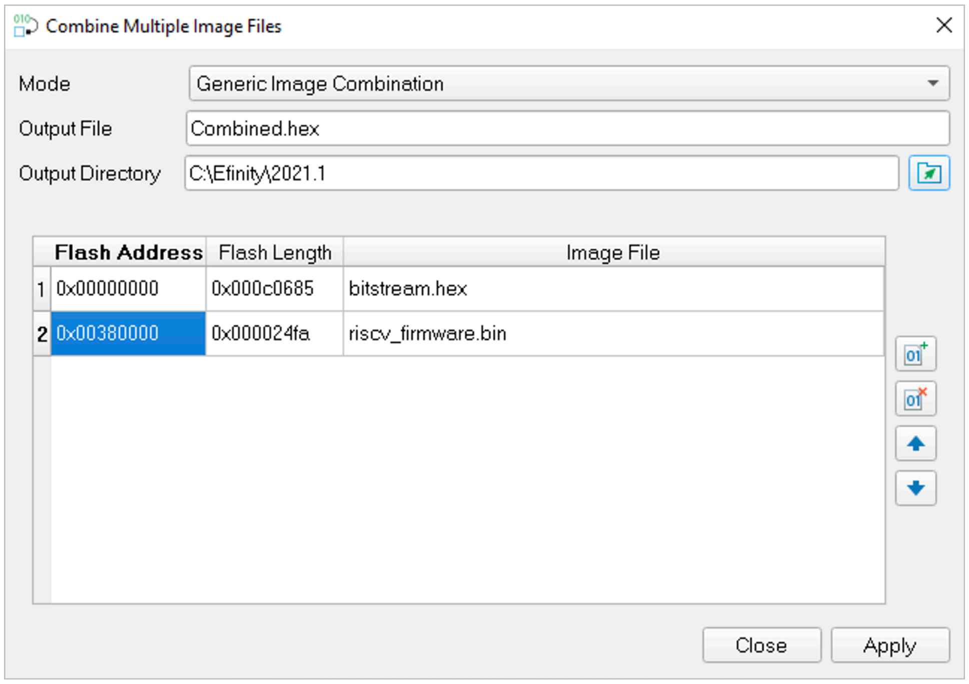
Copy a User Binary to Flash (Efinity Programmer)

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

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

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

Figure 16: Combining a Bitstream and RISC-V Application Binary



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

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

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

Note: You can also use two terminals to copy the application binary to flash. Refer to Appendix: Copy a User Binary to the Flash Device (2 Terminals) of Sapphire RISC-V SoC Hardware and Software User Guide.

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) instead.

You use two command prompts or terminals:

- The first terminal opens an OpenOCD connection to the SoC.
- The second connects to the first terminal to write to the flash.

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

Set Up Terminal 1

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

Efinity RISC-V Embedded Software IDE Selected

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

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

   Windows:

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

   Linux:

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

   The OpenOCD server connects and begins listening on port 4444.

Legacy Eclipse IDE Selected

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

   Windows:

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

   Linux:

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

   The OpenOCD server connects and begins listening on port 4444.

Set Up Terminal 2

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

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

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

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

Close Terminals

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

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

Reset the FPGA

Press the reset button on the development board:
- Trion development boards—SW2
- 钛金系列 development boards—SW3

Chapter 2

Image Signal Processing Example Design

Contents:

So you can get started easily with the Image Signal Processing example design, 易灵思 provides RTL designs targeting the following development boards:

- Trion® T120 BGA324 Development Board
- Trion® T120 BGA576 Development Board
- 钛金系列 Ti60 F225 Development Board
- 钛金系列 Ti180 M484 Development Board

In this design, the Hardware Accelerator demonstrates how you can use software (RISC-V firmware) to control the FPGA hardware (acceleration functions).

Figure 17: Image Signal Processing Example Block Diagram Trion Development Boards

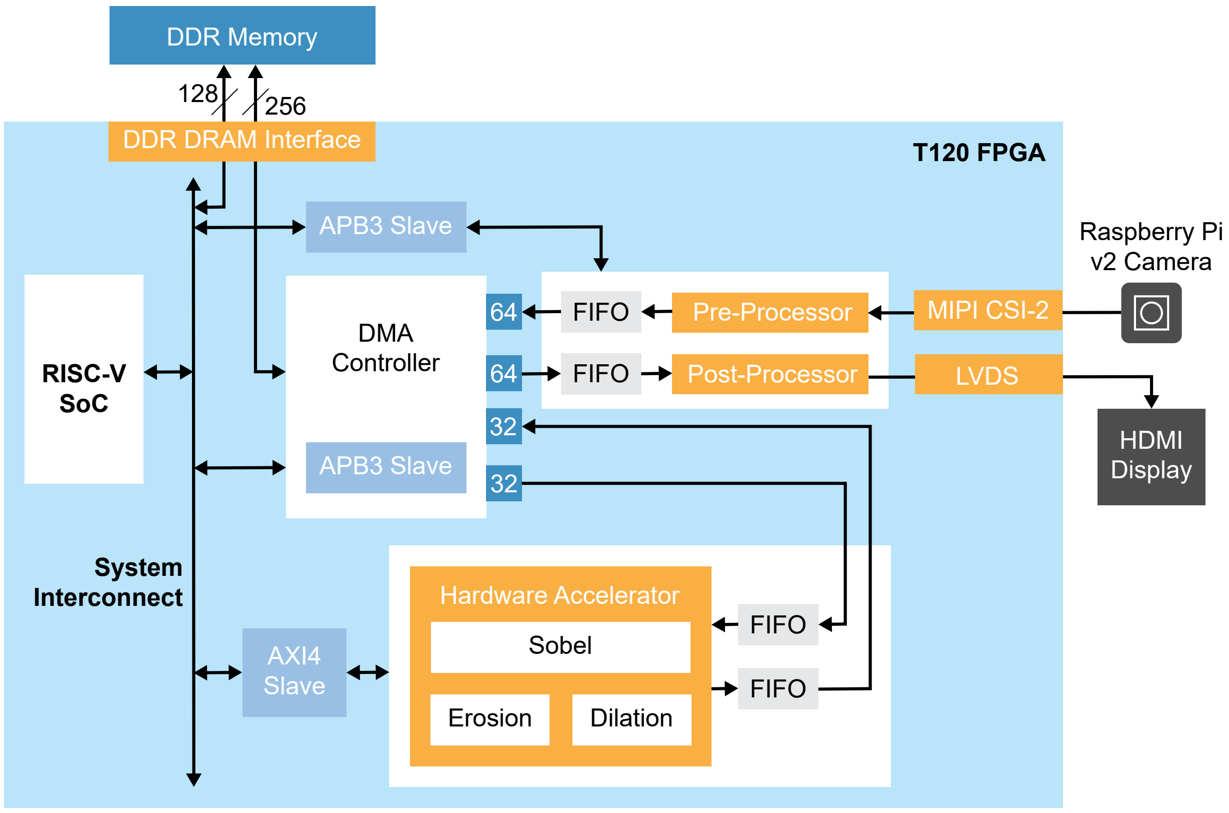


Figure 18: Image Signal Processing Example Block Diagram 钛金系列 Ti60 F225 Development Board

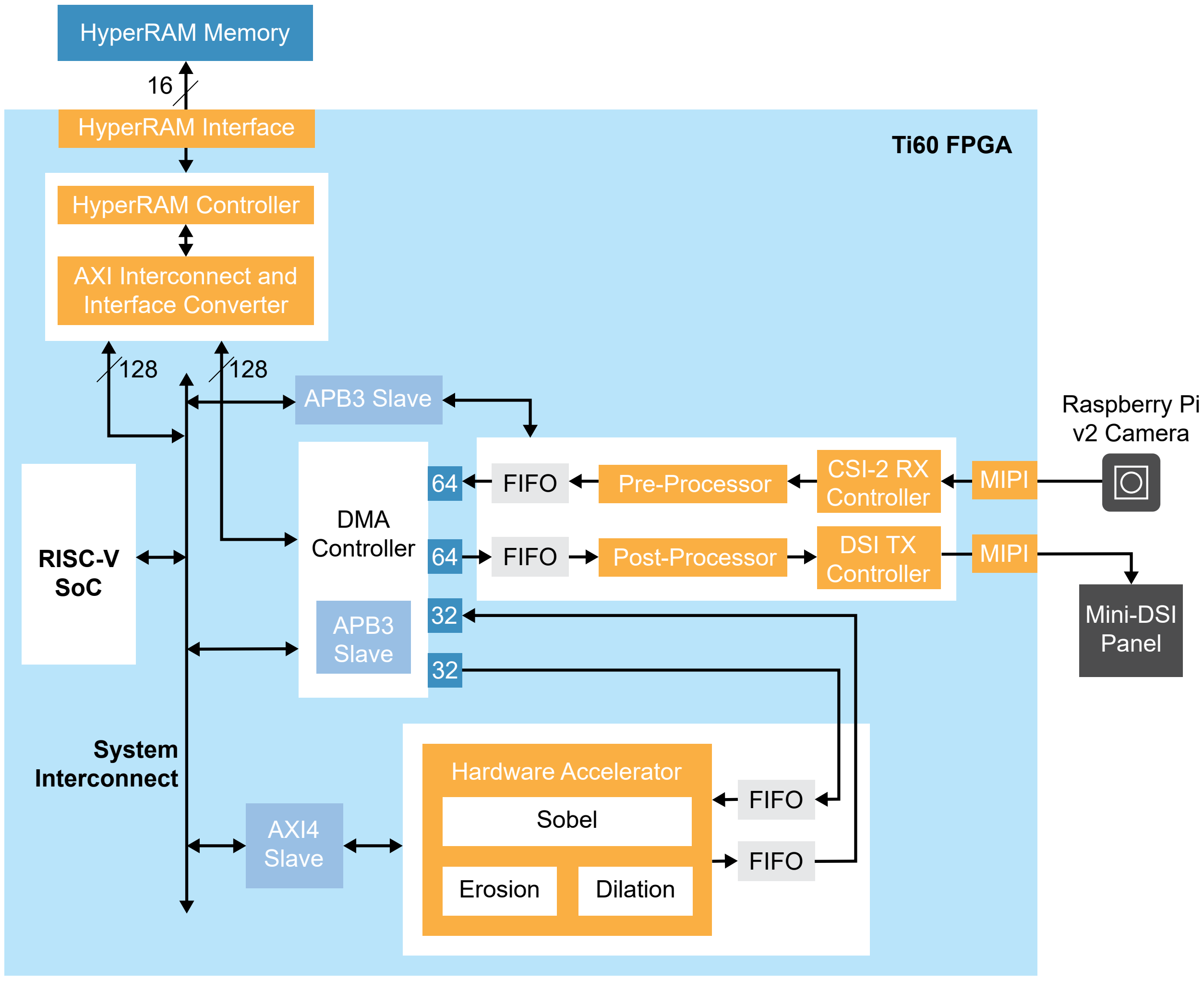
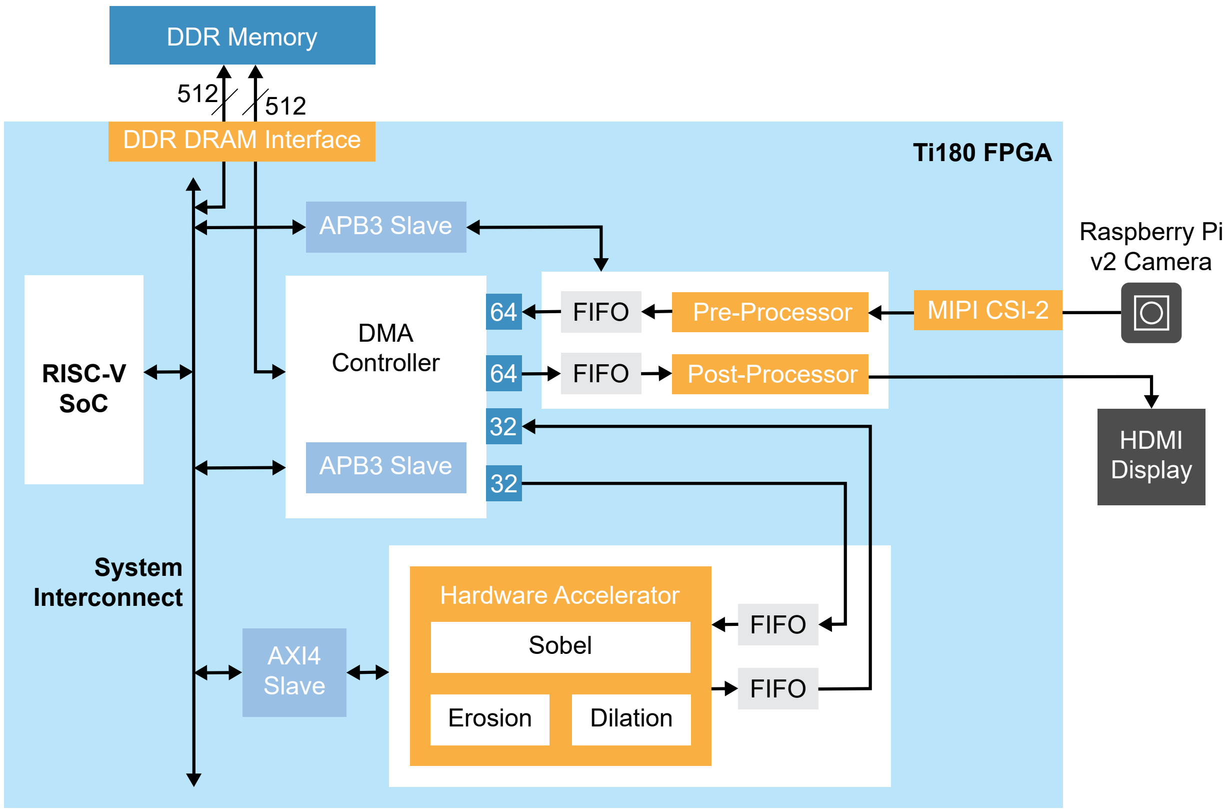


Figure 19: Image Signal Processing Example Block Diagram 易灵思 Ti180 M484 Development Board



Trion® T120 BGA324 Development Kit Design Resource Utilization

The following table shows the resources used for each block as compiled in the Efinity® software v2022.2. Overall, the designs use:

- *640 x 480 project*—34,519 logic elements (LEs)
- *1280 x 720 project*—33,686 LEs

Table 6: Example Design Implementation (640 x 480)

Block	Flipflops	Adders	LUTs	Memory Blocks	Multipliers
Complete Design	16,411	3,562	23,498	192	4
RISC-V SoC	7,168	796	6,076	54	4
DMA Controller	7,557	1,247	16,092	92	0
Camera controls	766	967	645	22	0
Display controls	172	112	112	10	0
Hardware Accelerator	635	424	407	14	0

Table 7: Example Design Implementation (1280 x 720)

Block	Flipflops	Adders	LUTs	Memory Blocks	Multipliers
Complete Design	16,418	3,577	22,653	204	4
RISC-V SoC	7,168	796	6,090	54	4
DMA Controller	7,549	1,247	15,193	92	0
Camera controls	767	968	679	22	0
Display controls	173	113	85	10	0
Hardware Accelerator	648	437	426	26	0

钛金系列 Ti60 F225 Development Kit Design Resource Utilization

The following table shows the resources used for each block as compiled in the Efinity® software v2022.2. Overall, the designs uses 33,862 eXchangeable Logic and Routing (XLR) Cells.

Block	Flipflops	Adders	LUTs	Memory Blocks (M10K)	DSP
Complete Design	17,902	3,980	21,625	189	4
RISC-V SoC	6,724	697	5,482	49	4
DMA Controller	4,762	824	6,072	53	0
CSI-2 RX Controller Core	844	194	2,036	15	0
DSI TX Controller Core	1,737	409	3,482	19	0
HyperRAM Controller Core	1,153	305	2,096	22	0
Camera	809	919	646	11	0
Display	338	174	374	8	0
Hardware Accelerator	646	427	405	8	0

钛金系列 Ti180 M484 Development Kit Design Resource Utilization

The following table shows the resources used for each block as compiled in the Efinity® software v2022.2. Overall, the designs uses 46,150 eXchangeable Logic and Routing (XLR) Cells.

Block	Flipflops	Adders	LUTs	Memory Blocks (M10K)	DSP
Complete Design	26,697	3660	29,088	257	4
RISC-V SoC	15,434	815	10,503	105	4
DMA Controller	8,362	973	15,031	64	0
CSI-2 RX Controller Core	618	204	1,595	17	0
Camera	740	946	671	11	0
Display	707	223	560	46	0
Hardware Accelerator	555	410	317	14	0

Sapphire RISC-V SoC

The image signal processing example design uses a Sapphire RISC-V SoC generated using the Efinity IP Manager.

The Sapphire SoC supports a highly flexible hardware/software co-design methodology, so you can choose whether to perform compute in the RISC-V processor or in hardware. The RISC-V SoC can be used to perform overall system control and execute algorithms that are inherently sequential or require flexibility.

Learn more: For more information, refer to the **Sapphire RISC-V SoC Hardware and Software User Guide**.

DMA Controller

The image signal processing example design uses a DMA Controller core generated from the Efinity IP Manager. The DMA controller facilitates communication between the external memory and other building blocks in the design. It stores frames of data into the external memory, sends and receives data to/from the hardware acceleration block, and sends data to the post-processing engine.

Learn more: For more information, refer to the **DMA Controller Core User Guide**.

Video Capture and Pre-Processing

The Image Signal Processing design uses a Raspberry Pi v2 camera module as the input sensor to the system. The video comes from the camera to the MIPI CSI-2 RX interface hardblock (Trion) or MIPI CSI-2 RX Controller core (钛金系列) at 1920 x 1080 resolution, RAW10 format (10 bits per raw pixel), 4 pixels per clock. The frame rate varies depending on the camera driver setting.

The Pre-Preprocessor receives the video from the camera and performs these functions:

- Adjusts the RGB gain according to the settings you choose in the RISC-V firmware.
- Changes the number of pixels per clock from 4 to 2.
- Converts RAW to RGB.
- Crops or scales the video for the subsequent processing.
- Optionally converts the RGB pixel data to grayscale pixel data.

The Trion development board example designs include an option for you to either crop or scale (bilinear method) the input resolution to the output resolution. To set using crop or scale, set the `CROP_SCALE` parameter in the **edge_vision_soc.v** file with the following values:

- *0*—Crop to output resolution (default)
- *1*—Scale to output resolution

For 钛金系列 Ti60 F225 Development Board design, the input video resolution is cropped to 1080 x 1080 and then scaled-down to 540 x 540 using the nearest-neighbor method.

When it has finished these functions, the Pre-Preprocessor stores the data into the external memory via the DMA Controller.

Firmware running on the RISC-V processor specifies whether the RGB or grayscale data is stored in the external memory. So you can choose to implement the conversion to grayscale in hardware, or to perform that function in the RISC-V processor instead.

For the Ti180 design, the default is the processing and display of the 1920x1080 resolution. The design can be modified for 1280x720 or 640x480 resolution by uncomment the define setup for specific resolution in the top level edge_vision_soc.v file location: **..../Ti180M484_hdmi/edge_vision_soc.v**. Example:

- To select resolution: uncomment only one at a time to select resolution of pre-processing and display.
  - Define Full_HD_1920x1080_60Hz 1 → for 1920x1080 resolution
  - Define HD Ready_1280x720_60Hz 1 → for 1280x720 resolution
  - Define SD_640x480_60Hz 1 → for 640x480 resolution
- User needs to update in **main.c** file. for riscv applications to match with FPGA hardware settings.
- File location: **..../evsoc_ispExample/src/main.c**. Example:
  - #define FRAME_WIDTH 1920
  - #define FRAME_HEIGHT 1080

  - //#define FRAME WIDTH 1280
  - //#define FRAME WIDTH 1280

  - //#define FRAME WIDTH 640
  - //#define FRAME WIDTH 480

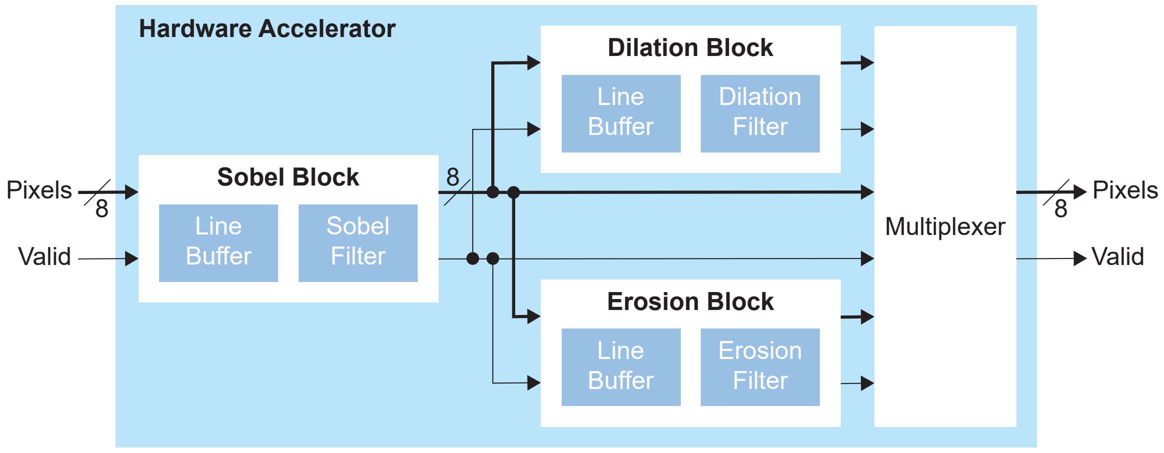
Hardware Accelerator

The design provides several standard functions in the Hardware Accelerator. (These functions are also available as software code.) By default, the design illustrates hardware/software co-design: the RISC-V processor executes the RGB to grayscale conversion as embedded software while the Hardware Accelerator performs the Sobel edge detection, binary erosion, and binary dilation.

- *Sobel filter*—Performs edge detection.
- *Binary erosion*—Removes line detail by ANDing all windowed pixels.
- *Binary dilation*—Strengthens line detail by ORing all windowed pixels.

The Hardware Accelerator modules are implemented in a pipelined, streaming architecture, and operates in three modes: Sobel, Sobel with dilation, or Sobel with erosion. You specify the mode in the firmware running on the RISC-V processor.

Figure 20: Hardware Acceration Block Diagram



In the RTL design, a standard wrapper encompasses the acceleration functions, input and output FIFOs, as well as debug and control registers that interface with an AXI4 slave module that connects to the RISC-V processor. The FIFOs and control logic fetch input data from the external memory and store output data to external memory through the DMA Controller. This wrapper is designed to provide a standard interface so you can drop in your own acceleration functions easily.

Post-Processing and Display

The Post-Processor generates video signals (HSYNC, VSYNC, and VALID) based on the target display resolution, and retrieves pixel data (red, blue, and green) from the display DMA FIFO. For Trion development boards designs, video signals and pixel data are packed and output through LVDS interface. Where as for 钛金系列 Ti60 F225 Development Board designs, video signals and pixel data are passed to the DSI TX Controller core, which facilitates the display on Mini-DSI panel.

Working with the Example

Working with the Image Signal Processing example design involves these steps:

1. Set up the hardware and download a bitstream file to the development board. (see **Program the Development Board** on page 18)
2. Set up an Eclipse project and configure the OpenOCD debugger. (see **Launch Efinity RISC-V Embedded Software IDE** on page 19)
3. Download the firmware binary to the RISC-V processor running in the Trion T120 FPGA. (see **Run the Application with Efinity RISC-V Embedded Software IDE** on page 29 or **Deploying an Application Binary**)

Out of the box, the design captures video from the Raspberry Pi camera and sends it to the display. However, you can modify the software to enable the filtering functions as described in **Customizing the Firmware** on page 50.

Additionally, the RTL design also supports firmware that read and writes image data to flash memory on the development board. Refer to **Flash Read and Write Software** on page 55 for details.

Generating IP

The Efinity® project(s) provided with this example design incorporate IP cores. Although the IP exists in the project, the zip file does not include all generate files. before you compile the project, you need to generate the IP.

1. Open the project in the Efinity® software. The Project pane under the Dashboard should show all the files in the design as well as the IP cores.
2. Right-click on the name of an IP core to open a context-sensitive menu.
3. Choose Generate. The IP Manager generates the deliverables for the IP core.
4. Repeat step 3 for all IP cores in the project.

As you are working with the example design, you may want to experiment with different settings for the IP core. In that case, right-click the IP core and choose Configure. The IP Configuration wizard opens. You can adjust the settings and re-generate the modified IP core.

Simulate

The Image Signal Processing example includes simulation files for use with ModelSim simulators. The testbench includes system-level SoC simulation but does not cover the video I/O interfaces, for example MIPI RX, LVDS, and DSI TX interfaces. The input to the simulation testbench is based on off-line RGB frame data. The **file image_64_48.vh** is 64 x 48 resolution, and `image_640_480.vh` is 640 x 480 resolution. The smaller resolution file is useful for speeding up the overall simulation time. The pixel data format is 32 bits per pixel {8'd0, 8-bit BLUE, 8-bit GREEN, 8-bit RED}. You can select the image data header file in `tb_soc.v`. You can also update the corresponding `FRAME_WIDTH` and `FRAME_HEIGHT` parameters.

The Trion development board design simulation is based on a simple DDR DRAM model; the 256- and 128-bit ports connect to two independent simple DDR models. Therefore, data written to one port is not synchronized with the other. Hardware/software codesign simulation is only feasible when the DMA Controller shares the same 128-bit DDR port as the RISC-V processor (connected to the DDR master port of the RISC-V system interconnect). The 钛金系列 development board design simulation is based on a HyperRAM model.

In the firmware created for simulation purposes (**evsoc_ispExample_sim**), the RGB to grayscale conversion is performed in the camera Pre-Processor block by default; RISC-V procecssor does not perform and the algorithmic computations.

Output checking is done on incoming data fetched by the DMA Controller into the Post-Processor; simulation does not check the display format. The intermediate grayscale and display output frame data is dumped to text files for verification.

To simulate:

1. Open a Command Prompt (Windows) or terminal (Linux).
2. Change to the directory to **../<project>/sim**.
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`.

By default, the **run.py** scripts simulate the **evsoc_ispExample_sim** application. Use the following command to simulate with a different application instead of the default:

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

For vision-based systems, you can examine the output of individual processing blocks through visualization. With the provided simulation testbench, intermediate pixel data is dumped to text files, and you can convert the pixel data into a standard image format to verify the result using an image viewer. Alternatively, you can perform a pixel-by-pixel comparison if you have an equivalent software implementation of all of the algorithmic blocks. You can do the comparison with your preferred method, such as MATLAB, Python, C or C++, which serves as the golden reference model.

About the Software

The Image Signal Processing design has software code in the **embedded_sw/SapphireSoc/ software/standalone** directory.

Before you use this example, review the instructions on using Using Efinity RISC-V Embedded Software IDE on page 19 so you are familiar with how to work with Eclipse projects.

Table 8: Software Directory Structure

Directory	Description
standalone/common	Provides linking for the makefiles.
standalone/driver	This directory contains the system drivers for the RISC-V peripherals (I^2C, UART, SPI, etc.). Refer to API Reference on page 72 for details.
standalone/evsoc_ispExample	Contains the software code for normal operation.
standalone/ evsoc_ispExample_demo	This design is similar to evsoc_ispExample except you can control the hardware acceleration function using UART commands in a terminal. With this method, you can view the different acceleration options without recompiling the software code.
standalone/ evsoc_ispExample_demo2(1)	This design is similar to evsoc_ispExample except you can control the hardware acceleration function using switches on the development board.
standalone/evsoc_ispExample_sim	Contains the software code for simulation.
standalone/ evsoc_ispExample_timestamp	This example is similar to evsoc_ispExample except it adds a frame counter and a timestamp to count the processing frame rates of the different scenarios.

Note: Software app folders are moved into standalone directory for compatibility with Efinity RISC-V Embedded Software IDE.

The **main.c** file in the `src` subdirectory contains several functional blocks. You can change the software operation by commenting and uncommenting certain sections (described later in this document). The camera capture, RISC-V processing, and Hardware Accelerator sections are within a while loop, which executes iteratively to process the video stream.

(1) Not available for 钛金系列 Ti60 F225 Development Board design.

Table 9: main.c Functional Blocks

Code Section	Function
DMA-Related Functions	Provides interrupt functions for the DMA Controller. This section only applies to the evsoc_ispExample_demo and evsoc_ispExample_demo2 examples.
Demo-Related Functions	Includes code to run the example via the UART or switches on the board. This section only applies to the evsoc_ispExample_demo and evsoc_ispExample_demo2 examples.
Setup PICAM & HDMI Display	Initializes the Raspberry Pi camera. Initializes the HDMI display. Sets the RGB gain values for the camera pre-processing block.
Setup DMA	Initializes the DMA controller. Sets the DMA priority for available the DMA channels (0 has highest priority).
Trigger Display	Initializes the test image display content (vertical colour bars a red dot in each corner) for display verification purposes. Indicates the memory source address for display data retrieval. Triggers the display MM2S DMA in self-restart mode (only once). Delays for 5 seconds. You will see a test image on the display for 5 seconds before entering video streaming mode.
Camera Capture	Selects the RGB or grayscale pixel output from the camera block. Indicates the memory destination address to store the camera data. Triggers the camera S2MM DMA and indicates when S2MM DMA initialization completes. Triggers capture/storage of one frame in the camera block. Triggers continuous mode to facilitate processing without frame-drop if scaler is enabled in camera building block. This code is commented out by default. When enabling this mode, comment-out the select RGB or grayscale pixel output and capture/storage of one frame in the camera block within the while loop.
RISC-V Processing	This code is commented out by default. Executes the filtering software functions. The code references the library **isp.h**, which contains the rgb2grayscale, sobel_edge_detection, binary_erosion, and binary dilation functions.
HW Accelerator	This code is commented out by default. Sets the threshold value for Sobel edge detection. Selects the hardware accelerator mode (Sobel only, Sobel plus dilation, or Sobel plus erosion). Indicates the memory source address for input data retrieval (MM2S). Selects the DMA transfer length based on the selected hardware accelerator mode. Triggers the hardware accelerator MM2S DMA. Indicates the memory destination address for storing the output data (S2MM). Triggers the hardware accelerator S2MM DMA and indicates when S2MM DMA initialization completes.
Check AXI4 Slave Status (HW Accelerator)	This code is commented out by default. Use these code snippets in other code segments to print the intermediate status of debug registers, for example, within the hardware accelerator section.
Check APB3 Slave Status (Camera & Display)	This code is commented out by default. Use these code snippets in other code segments to print the intermediate status of debug registers, for example, within the Trigger Display or Camera capture section.

Customizing the Firmware

Once you have the basic out-of-box streaming video working, you can begin to customize software. The **main.c** file includes commented code that you can enable to perform various functions, some in hardware and some in software. After you change the **main.c** file, you must recompile the software and download the new firmware file (**.elf**) to the board. The following sections describe how to perform the customizations.

Grayscale and Sobel

This customization turns on the grayscale and Sobel filters. The RISC-V processor performs the grayscale conversion while the Hardware Accelerator performs the Sobel filtering. Make these changes in the **main.c** file using the Eclipse project you have already set up.

In the Trigger Display section, comment out the code for `cam_array` and uncomment the code for `sobel_array`:

```
/*
  //Initialize test image in cam_array
   for (int y=0; y<IMG_HEIGHT; y++) {
      for (int x=0; x<IMG_WIDTH; x++) {
         if ((x<3 && y<3) || (x>=IMG_WIDTH-3 && y<3) || (x<3 && y>=IMG_HEIGHT-3) ||
 (x>=IMG_WIDTH-3 && y>=IMG_HEIGHT-3)) {
            cam_array [y*IMG_WIDTH + x] = 0x000000FF; //RED
         } else if (x<(IMG_WIDTH/4)) {
            cam_array [y*IMG_WIDTH + x] = 0x0000FF00; //GREEN
         } else if (x<(IMG_WIDTH/4 *2)) {
            cam_array [y*IMG_WIDTH + x] = 0x00FF0000; //BLUE
         } else if (x<(IMG_WIDTH/4 *3)) {
            cam_array [y*IMG_WIDTH + x] = 0x000000FF; //RED
         } else {
            cam_array [y*IMG_WIDTH + x] = 0x00FF0000; //BLUE
         }
      }
   }
*/
   //Initialize test image in sobel_array
   for (int y=0; y<IMG_HEIGHT; y++) {
      for (int x=0; x<IMG_WIDTH; x++) {
         if ((x<3 && y<3) || (x>=IMG_WIDTH-3 && y<3) || (x<3 && y>=IMG_HEIGHT-3) ||
 (x>=IMG_WIDTH-3 && y>=IMG_HEIGHT-3)) {
            sobel_array [y*IMG_WIDTH + x] = 0x000000FF; //RED
         } else if (x<(IMG_WIDTH/4)) {
            sobel_array [y*IMG_WIDTH + x] = 0x0000FF00; //GREEN
         } else if (x<(IMG_WIDTH/4 *2)) {
            sobel_array [y*IMG_WIDTH + x] = 0x00FF0000; //BLUE
         } else if (x<(IMG_WIDTH/4 *3)) {
            sobel_array [y*IMG_WIDTH + x] = 0x000000FF; //RED
         } else {
            sobel_array [y*IMG_WIDTH + x] = 0x00FF0000; //BLUE
         }
      }
   }
```

Also in the Trigger Display section, comment the DMA source address for the camera and uncomment the one for the Sobel filter:

```
//SELECT start address of to be displayed data accordingly
//dmasg_input_memory(DMASG_BASE, DMASG_DISPLAY_MM2S_CHANNEL, CAM_START_ADDR, 16);
//dmasg_input_memory(DMASG_BASE, DMASG_DISPLAY_MM2S_CHANNEL, GRAYSCALE_START_ADDR, 16);
dmasg_input_memory(DMASG_BASE, DMASG_DISPLAY_MM2S_CHANNEL, SOBEL_START_ADDR, 16);
```

Perform these additional code changes:

1. Uncomment the whole RISC-V Processing section, which activates the `rgb2grayscale` function.
2. Uncomment the whole HW Accelerator section, which activates the Hardware Accelerator for Sobel filtering.
3. Clean and then build the project.
4. Press the CRESET pushbutton on the development board.

   Note: You MUST reset the board before downloading the new **.elf** to the board. Otherwise the video will not display correctly.

5. Download the resulting **.elf** file to the board.

Place an object with obvious sharp color edges, for example a calendar, in front of the Raspberry Pi camera. The display shows the detected edges in a binary frame (black and white).

Grayscale, Sobel, and Dilation

This customization builds on the previous example. In addition to the grayscale and Sobel filering, this example adds the dilation function. The RISC-V processor performs the grayscale conversion while the Hardware Accelerator performs the Sobel filtering and dilation. Make these changes in the **main.c** file using the Eclipse project you have already set up.

Note: Make the code changes described in the Grayscale and Sobel on page 50 example if you have not already done so.

In the Hardware Accelerator section, comment out the `Sobel only - Default` line and uncomment the `Sobel+Dilation` line.

```
//SELECT HW accelerator mode - Make sure match with DMA transfer length setting
//write_u32(0x00000000, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET);   //2'd0: Sobel only -
 Default
write_u32(0x00000001, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET); //2'd1: Sobel+Dilation
//write_u32(0x00000002, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET); //2'd2: Sobel+Erosion
```

Also in the Hardware Accelerator section, comment out the `Sobel only` line and uncomment the `Sobel + Dilation/Erosion` line.

```
//SELECT dma transfer length - Make sure match with HW accelerator mode selection
//Additonal data is required to be fed for line buffer(s) data flushing
//dmasg_direct_start(DMASG_BASE, DMASG_HW_ACCEL_MM2S_CHANNEL,
 ((IMG_WIDTH*IMG_HEIGHT)+(IMG_WIDTH+1))*4, 0);      //Sobel only
dmasg_direct_start(DMASG_BASE, DMASG_HW_ACCEL_MM2S_CHANNEL,
 ((IMG_WIDTH*IMG_HEIGHT)+(2*IMG_WIDTH+2))*4, 0); //Sobel + Dilation/Erosion
```

Press the CRESET pushbutton on the development board.

Clean and rebuild the project, then download the resulting **.elf** file to the board.

Place the same object you used in the Grayscale and Sobel on page 50 example in front of the Raspberry Pi camera to see the difference. The display shows the detected edges with bolder, thicker lines.

Grayscale, Sobel, and Erosion

This customization builds on the previous example. In addition to the grayscale and Sobel filering, this example adds the erosion function. The RISC-V processor performs the grayscale conversion while the Hardware Accelerator performs the Sobel filtering and erosion. Make these changes in the **main.c** file using the Eclipse project you have already set up.

Note: Make the code changes described in the Grayscale and Sobel on page 50 example if you have not already done so.

In the Hardware Accelerator section, comment out the `Sobel only - Default` line and uncomment the `Sobel+Erosion` line.

```
//SELECT HW accelerator mode - Make sure match with DMA transfer length setting
//write_u32(0x00000000, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET);   //2'd0: Sobel only -
 Default
//write_u32(0x00000001, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET); //2'd1: Sobel+Dilation
write_u32(0x00000002, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET); //2'd2: Sobel+Erosion
```

If you have not already done so, in the Hardware Accelerator section, comment out the `Sobel only` line and uncomment the `Sobel + Dilation/Erosion` line.

```
//SELECT dma transfer length - Make sure match with HW accelerator mode selection
//Additonal data is required to be fed for line buffer(s) data flushing
//dmasg_direct_start(DMASG_BASE, DMASG_HW_ACCEL_MM2S_CHANNEL,
 ((IMG_WIDTH*IMG_HEIGHT)+(IMG_WIDTH+1))*4, 0);     //Sobel only
dmasg_direct_start(DMASG_BASE, DMASG_HW_ACCEL_MM2S_CHANNEL,
 ((IMG_WIDTH*IMG_HEIGHT)+(2*IMG_WIDTH+2))*4, 0); //Sobel + Dilation/Erosion
```

Press the CRESET pushbutton on the development board.

Clean and rebuild the project, then download the resulting **.elf** file to the board.

Place the same object you used in the Grayscale and Sobel on page 50 example in front of the Raspberry Pi camera to see the difference. The display shows the detected edges with fainter, thinner lines.

Using Your Own Hardware Accelerator

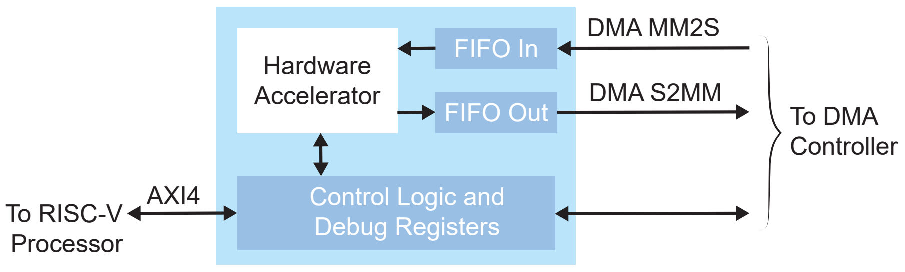
The Image Signal Processing example has a Hardware Accelerator that performs filtering functions. The top-level wrapper (**hw_accel_wrapper.v**) comprises the Hardware Accelerator, FIFO buffers, and control logic and debug registers. The wrapper has ports that connect to the DMA Controller and an AXI4 slave module.

The slave interface is used for communication between the RISC-V processor and Hardware Accelerator, mainly for debug and control purposes. Although this design uses an AXI4 slave interface, you can change the wrapper to use other slave interfaces such as APB3 or AXI4-lite.

- For an example APB3 interface, refer to the **apb3_cam.v** file. This file implements an APB3 slave module that is shared by the camera and display blocks for communication with the RISC-V processor.
- For an example AXI4-lite interface, refer to the APB3 to AXI4-Lite Core in the Support Center.

The DMA Controller facilitates the input/output data stream between the Hardware Accelerator and main memory. FIFOs handle any differences between the input/output data rate and Hardware Accelerator data processing rate. Control logic in the wrapper file generates the FIFOs and DMA read/write related signals, according to the behaviour of the Hardware Accelerator (for example, the required input/output data patterns or flow, firmware setup for DMA transfers, etc).

Figure 21: Hardware Accelerator Connections



If you want to drop in your own custom accelerator, you may need to make adjustments to the wrapper and to the firmware. The following sections describes areas you should consider.

Slave Interface Changes

First, decide which type of slave interface you want to use, AXI4, APB3, or AXI4-lite. Next, determine what communication you need between the RISC-V processor and Hardware Accelerator for for setup control and debug registers. For example, you can use the RISC-V processor to trigger the start of computations and handshaking, set up different operating modes, or probe critical signals for debugging purposes. Finally, update the control logic and debug registers in the Hardware Accelerator wrapper, according to your design.

DMA Controller Changes

The first step is to determine the input(s) and output(s) data requirements. The DMA Controller uses AXI-ST interfaces to the Hardware Accelerator. By default, the design has 32-bit input and output streams to retrieve data and store it from/to the main memory.

The DMA Controller supports asynchronous mode for individual channels. If you need the input/output stream to run at a different clock rate, update the clock signal connection to the respective DMA Controller channel(s).

Learn more: The Image Signal Processing example design includes a DMA Controller core generated from the Efinity IP Manager. For more information, refer to the **DMA Controller Core User Guide**.

FIFO Changes

Determine the FIFO requirements (data width, depth, mode, etc.) and the read/write behaviours based on the needs of your custom hardware accelerator, and modify the FIFOs accordingly. Next, examine potential FIFO underflows and overflows and adjust the FIFO control signals or DMA transfer settings accordingly.

Learn more: The Image Signal Processing example design includes a FIFO core generated from the Efinity IP Manager. For more information, refer to the **FIFO Core User Guide**.

RISC-V Firmware Changes

After modifying RTL side of the design, you must update the RISC-V firmware to match your changes. For example, modify **axi4_hw_accel.h** if you add more addressing offset.

If you use a different slave interface, create a corresponding header for ease of access in **main.c**. Refer to the Camera Capture, HW Accelerator, Check AXI4 Slave Status (HW Accelerator), and Check APB3 Slave Status (Camera & Display) sections in **main.c** for example APB3 and AXI4 slave interface usage. The following code snippet shows some example read/write usage:

```
//Write to AXI4 slave - SET Sobel edge detection threshold via AXI4 slave
write_u32(100, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG0_OFFSET); //Default value 100; Range 0 to
 255
//Read from AXI4 slave - RETRIEVE HW accelerator DMA FIFOs status
rdata = axi_slave_read32(EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG4_OFFSET);

//Write to APB3 slave - SELECT RGB or grayscale output from camera pre-processing block.
EXAMPLE_APB3_REGW(EXAMPLE_APB3_SLV, EXAMPLE_APB3_SLV_REG3_OFFSET, 0x00000000); //RGB
//Read from APB3 slave - RETRIEVE camera & display DMA FIFOs status
rdata = example_register_read(EXAMPLE_APB3_SLV_REG6_OFFSET);
```

When you use a custom hardware accelerator, you should update the DMA transfer setting in the firmware. For example, update the DMA transfer length per captured frame based on the accelerator's behaviour. (The third argument of `dmasg_direct_start()` is the transfer length in number of bytes.) By default, the example design uses direct mode DMA transfer.

```
//For HW accel MM2S (fetch data from main memory to HW accel building block)
dmasg_direct_start(DMASG_BASE, DMASG_HW_ACCEL_MM2S_CHANNEL, ((IMG_WIDTH*IMG_HEIGHT)+(IMG_WIDTH
+1))*4, 0);

//For HW accel S2MM (store data from HW accel building block to main memory)
dmasg_direct_start(DMASG_BASE, DMASG_HW_ACCEL_S2MM_CHANNEL, (IMG_WIDTH*IMG_HEIGHT)*4, 0);
```

Flash Read and Write Software

The flash read and write firmware can read from or write to the Winbond SPI NOR flash memory on the development board. This software code is located in the **<project>/embended_sw/SapphireSoc/software/standalone** directory.

Note: The flash read and write examples target the T120 640x480 resolution design projects. By referring to the provided examples, you can modify the design for other development boards or use cases.

Table 10: Software Directory Structure

Directory	Description
standalone/common	Provides linking for the makefiles.
standalone/driver	This directory contains the system drivers for the RISC-V peripherals (I^2C, UART, SPI, etc.). Refer to API Reference on page 72 for details.
standalone/evsoc_readFlash	This example shows you how to read an image from flash memory on board and display the retrieved image on an HDMI monitor.
standalone/evsoc_writeFlash	This example shows how to write a 640 x 480 RGB color image to the flash memory on the board.

Note: Software app folders are moved into standalone directory for compatibility with Efinity RISC-V Embedded Software IDE.

The **main.c** file in the `src` subdirectory contains several functional blocks.

Table 11: evsoc_readFlash main.c Functional Blocks

Code Section	Function
Setup HDMI Display	Initializes the HDMI display.
Setup DMA	Initializes the DMA controller. Sets the DMA priority for available the DMA channels (0 has highest priority).
Read Flash	Initializes the display content from an off-line image. Reads the memory as a continuous stream of data.
Trigger Display	Displays the image.

Table 12: evsoc_writeFlash main.c Functional Blocks

Code Section	Function
Flash Erase	This example performs a 64 KB block erase 15 times to cover all of the memory sectors that the software will write to.
Write to Flash	Writes the image data to the flash memory. The example uses RGB888 as the image format.

Writing to the Flash

This firmware writes a 640 x 480 RGB888 image to the flash memory. Before you use this example, review the instructions on using Using Efinity RISC-V Embedded Software IDE on page 19 so you are familiar with Eclipse projects.

Important: The default BSP for the Sapphire SoC has a 124K memory length. That size is too small for the image, so you need to change the memory length to 1600K in the **default.ld** file in the **embedded_sw/SapphireSoc/bsp/efinix/EfxSapphireSoC/linker/** directory.

```
MEMORY
{
  /* ram  (wxai!r) : ORIGIN = 0x00001000, LENGTH = 124K */
  ram  (wxai!r) : ORIGIN = 0x00001000, LENGTH = 1600K
}
```

To use this example:

1. Create a new Eclipse project with the makefile in **evsoc_writeFlash**.
2. Build the project.
3. Download the resulting **.elf** to the development board.
4. When you run the software you receive these messages:

```
Hello 易灵思 Edge Vision SoC!!
Initialization..
Erase Flash Start..
Erase Flash End..
Write Flash Start..
Write Flash End..
```

Note: The 640 x 480 image data is compiled into the firmware binary, which leads to a larger binary file than usual and a longer write time. Therefore, it may take 1 - 2 minutes to write the binary to the flash memory.

Reading from the Flash

To use this example:

1. Create a new Eclipse project with the makefile in **evsoc_readFlash**.
2. Build the project.
3. Download the resulting **.elf** to the development board.
4. When you run the software, you receive the following messages, and the retrieved color image displays on the HDMI monitor.

```
Hello 易灵思 Edge Vision SoC!!

Init HDMI I2C.....Done !!
Init DMA.....Done !!

Init SPI.....Done !!
Reading offline image from flash..
Done !!

Trigger display DMA..
Done !!
```

Customizing the Firmware

The example illustrates these basic concepts:

- Initializing your user data in a header.
- Including user data in firmware.
- Writing the user data to flash memory.
- Retrieving the stored user data from flash memory.

Because the flash memory is non-volatile, it retains the user data even after power cycling. You can use the example as a starting point for writing and/or reading any user data you need for your application, for example, neural network coefficients.

Note: If you are planning to write and then read data and want to change the starting flash memory address, make sure that you use the same value for the `StartAddress` (write) and `FLASH_START_ADDR` (read) parameters.

Write Customizations

In **main.c** you can change the starting address for the flash memory:

```
#define StartAddress 0x3B0000 //User data location
```

You can also change the page length and number of pages. In the example, the `page_len` value is for the Winbond flash memory on the development board. If you are using a different flash device, adjust this parameter accordingly.

The `num_page` value depends on how many bytes of user data will be written to the flash (`num_page` = total number of bytes /`page_len`). If the total number bytes cannot be divided evenly, the last page write is a partial page.

```
page_len = 256;   //256 bytes per page program for deployed Winbond flash
num_page = 3600;  //For 640x480 image, (640x480x3bytes) / 256bytes = 3600 pages
```

To change the image that is written to the flash memory, replace the image content in the **img_data.h** file with your own image data. Additionally, instead of **img_data.h**, you can refer to your own data file with your own content.

Read Customizations

In **main.c** you can change the starting address for the flash memory:

```
//User data location in flash (non-volatile)
#define FLASH_START_ADDR      0x003B0000
```

If you are not storing an image, simply remove the frame height and width parameters.

Chapter 3

Dual-Camera Example Design

Contents:

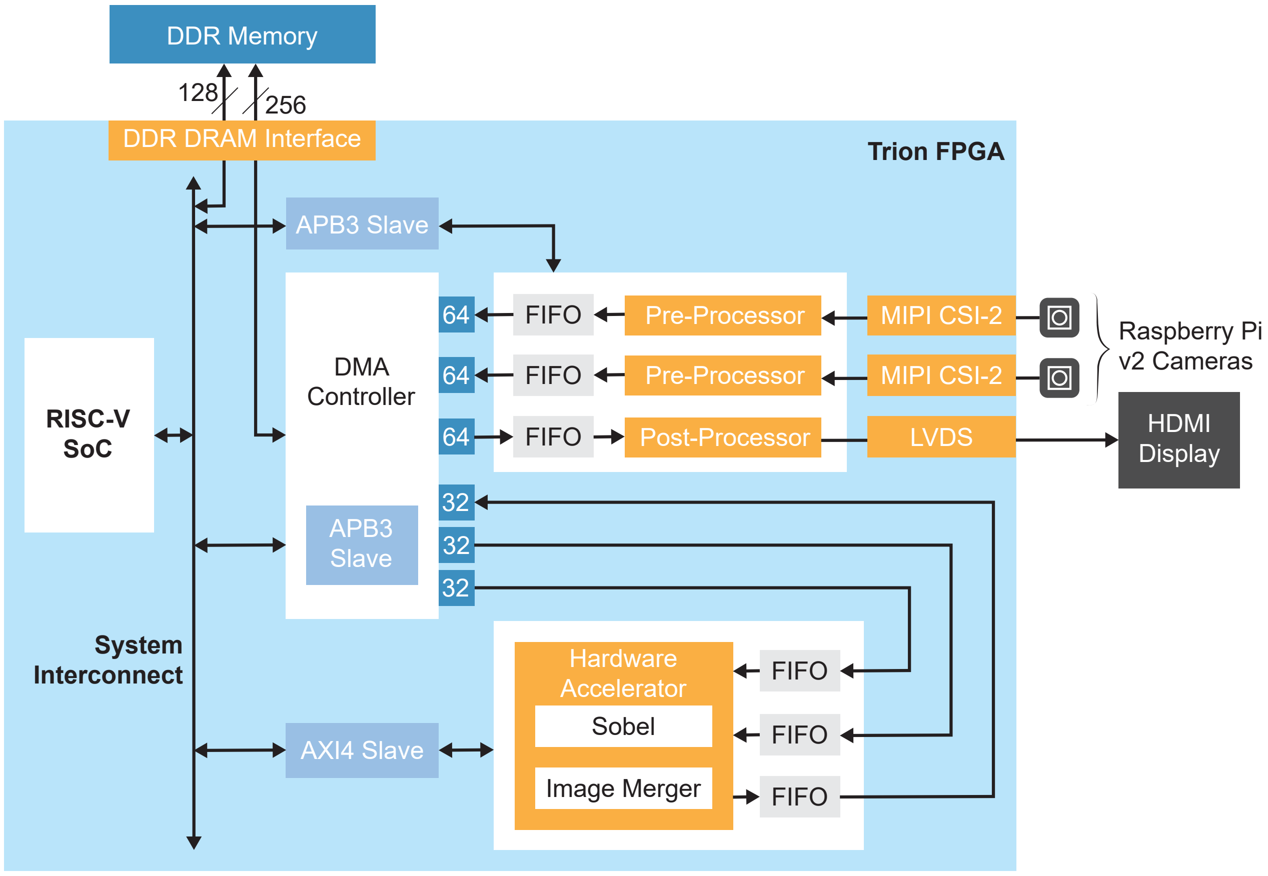
Multi-camera vision systems are used in a variety of applications such as video surveillance, security systems, robotics, automotive, and drones. Using multiple cameras can resolve occlusion issues, provide coverage for a wider area, and produce more accurate geometries.

The Dual-Camera example design targets the following development boards:

- Trion® T120 BGA324 Development Board
- Trion® T120 BGA576 Development Board

This design is similar to the Image Signal Processing example design, except it uses two Raspberry Pi v2 cameras and includes an image merger block to combine the video streams. The provided code includes an Efinity® project for video output at 1280 x 720,.

Figure 22: Dual-Camera Example Block Diagram



Trion® T120 BGA324 Development Kit Design Resource Utilization

The following table shows the resources used for each block as compiled in the Efinity® software v2022.2 for dualcam. Overall, the design uses 38,789 logic elements (LEs).

Table 13: Example Design Implementation (Dualcam)

Block	Flipflops	Adders	LUTs	Memory Blocks	Multipliers
Complete Design	19,092	4,909	25,534	264	4
RISC-V SoC	7,168	796	6,132	54	4
DMA Controller	9,219	1,453	17,192	112	0
Camera controls	1,534	1,936	1,335	44	0
Display controls	173	113	107	10	0
Hardware Accelerator	871	595	557	44	0

Sapphire RISC-V SoC

The dual-camera example design uses a Sapphire RISC-V SoC generated using the Efinity IP Manager.

The Sapphire SoC supports a highly flexible hardware/software co-design methodology, so you can choose whether to perform compute in the RISC-V processor or in hardware. The RISC-V SoC can be used to perform overall system control and execute algorithms that are inherently sequential or require flexibility.

Learn more: For more information, refer to the **Sapphire RISC-V SoC Hardware and Software User Guide**.

DMA Controller

The dual-camera example design uses a DMA Controller core generated from the Efinity IP Manager. The DMA controller for dual-camera example has 6 channels to accommodate the additional outbound (from the camera) and inbound (to the Hardware Accelerator) channels required for the second camera.

Learn more: For more information, refer to the **DMA Controller Core User Guide**.

Video Capture and Pre-Processing

The video capture and pre-processing blocks are the same as used in the Image Signal Processing example design, except the Dual-Camera design processes two camera streams instead of one. For more details, refer to Video Capture and Pre-Processing on page 44.

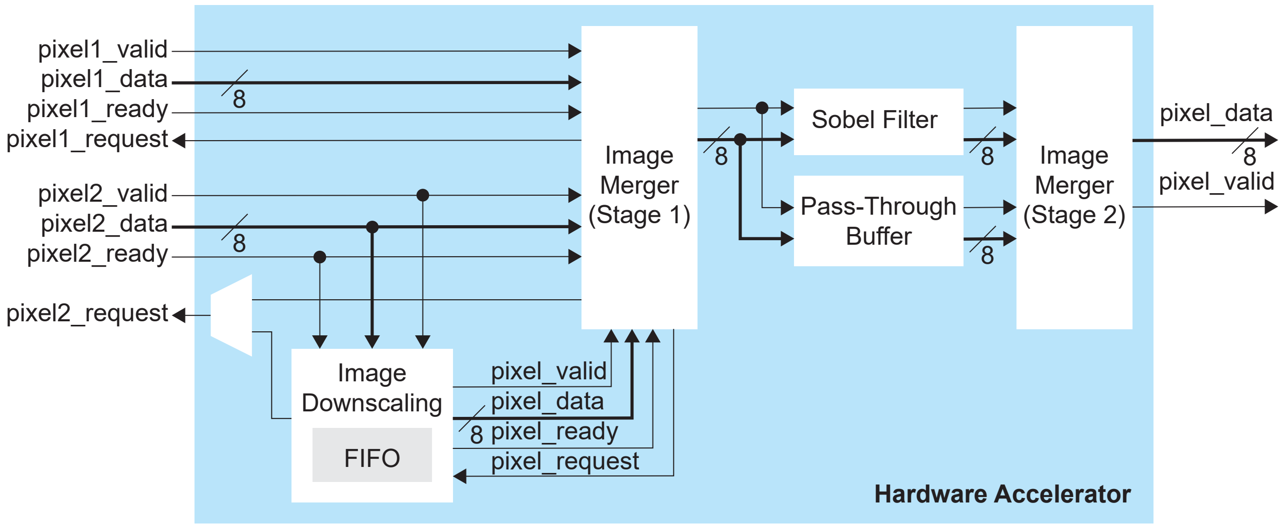
Hardware Accelerator

The design provides several standard functions in the Hardware Accelerator.

- *Image merger*—Combines the video streams from the two cameras.
- *Sobel filter*—Performs edge detection.
- *Grayscale filter*—Converts the RGB video to grayscale.

The Pass-Through Buffer can transfer RGB (default) or grayscale video. The Sobel filter only operates on grayscale video.

Figure 23: Hardware Acceration Block Diagram



The Hardware Accelerator modules are implemented in a pipelined, streaming architecture, and operate in several modes:

Table 14: Filtering Modes

Mode	Camera 1	Camera 2
1	Pass through	Pass through
2	Pass through	Sobel filter
3	Sobel filter	Pass through
4	Sobel filter	Sobel filter

Table 15: Image Merging Modes

Mode	Description
1	Merge the frames from two cameras; each camera's frame is cropped in half.
2	Create a downscaled overlay from camera 2 and merge it on top of the frame data from camera 1.

In the RISC-V firmware, in the camera capture section, you can select the RGB or grayscale video output from the camera building block.

Like the Image Signal Processing design, a standard wrapper encompasses the acceleration functions, input and output FIFOs, as well as debug and control registers as a standard interface so you can drop in your own acceleration functions easily.

Post-Processing and Display

The Post-Processor generates video signals (HSYNC, VSYNC, and DE), and pixel data for the LVDS interface. It creates a TX video signal that conforms to the VESA specifications (operating frequency, blanking, etc.,) for a 1280 x 720, 60 Hz HDMI flat panel display. It operates using a `tx_slowclk` of 37.12 MHz. The Post-Processor fetches the frame data for the HDMI display from the external memory.

Working with the Example

Working with the Dual-Camera example design involves these steps:

1. Set up the hardware and follow the instructions for adding a second camera. (see Set Up the Development Board on page 10)
2. Download a bitstream file to the development board. (see Program the Development Board on page 18)
3. Set up an Eclipse project and configure the OpenOCD debugger. (see Using Efinity RISC-V Embedded Software IDE on page 19)
4. Download the firmware binary to the RISC-V processor running in the Trion T120 FPGA. (see Run the Application with Efinity RISC-V Embedded Software IDE on page 29 or Deploying an Application Binary)

Out of the box, the design captures video from both Raspberry Pi cameras, crops the frames, and sends the combined RGB stream to the HDMI display in pass-through mode. However, you can modify the software to output grayscale instead of RGB video, enable the Sobel filter function on either or both video streams, and to show the images overlaid instead of cropped. Refer to Customizing the Firmware on page 64 for instructions on how to make these changes.

Note: Before using the RTL design, follow the instructions in Generating IP on page 46.

About the Software

The Dual-Camera design has software code in the **<project>/embended_sw/SapphireSoc/software/standalone** directory. Before you use this software, review the instructions on using Using Efinity RISC-V Embedded Software IDE on page 19 so you are familiar with Eclipse projects.

Table 16: Software Directory Structure

Directory	Description
standalone/common	Provides linking for the makefiles.
standalone/driver	This directory contains the system drivers for the RISC-V peripherals (I^2C, UART, SPI, etc.). Refer to API Reference on page 72 for details.
standalone/evsoc_dualCam	Contains the software code for normal operation.
standalone/evsoc_dualCam_demo	This design is similar to evsoc_dualCam except you can control the hardware acceleration function using UART commands in a terminal. With this method, you can view the different acceleration options without recompiling the software code.

Note: Software app folders are moved into standalone directory for compatibility with Efinity RISC-V Embedded Software IDE.

The **main.c** file in the `src` subdirectory contains several functional blocks. You can change the software operation by commenting and uncommenting certain sections (described later in this document). The camera capture, RISC-V processing, and Hardware Accelerator sections are within a while loop, which executes iteratively to process the video stream.

Table 17: main.c Functional Blocks

Code Section	Function
UART-Related Functions Demo-Related Functions	Provides functions to allow you to interact with the demo using a UART. This section only applies to the evsoc_dualCam_demo example.
Setup PICAM & HDMI Display	Initializes the Raspberry Pi cameras. Initializes the HDMI display. Sets the RGB gain values for the camera pre-processing block.
Setup DMA & UART	Initializes the DMA controller and UART (evsoc_dualCam_demo example only). Sets the DMA priority for available the DMA channels (0 has highest priority).
Trigger Display	Initializes the test image display content (vertical colour bars a red dot in each corner) for display verification purposes. Indicates the memory source address for display data retrieval. Triggers the display MM2S DMA in self-restart mode (only once). Delays for 5 seconds. You will see a test image on the display for 5 seconds before entering video streaming mode.
Camera Capture	Selects the RGB or grayscale pixel output from the camera blocks. Indicates the memory destination address to store the camera data. Triggers the cameras' S2MM DMA and indicates when S2MM DMA initialization completes. Triggers capture/storage of one frame in the camera blocks.
RISC-V Processing	This section is a placeholder you can use for your own functions.
HW Accelerator	Sets the threshold value for Sobel edge detection. Selects the filtering mode (pass through or Sobel filter). Selects the image merging mode (cropped or overlaid). Indicates the memory source address for input data retrieval (MM2S). Triggers the hardware accelerator MM2S DMA. Indicates the memory destination address for storing the output data (S2MM). Triggers the hardware accelerator S2MM DMA and indicates when S2MM DMA initialization completes.
Check AXI4 Slave Status (HW Accelerator)	This code is commented out by default. Use these code snippets in other code segments to print the intermediate status of debug registers, for example, within the hardware accelerator section.
Check APB3 Slave Status (Camera & Display)	This code is commented out by default. Use these code snippets in other code segments to print the intermediate status of debug registers, for example, within the Trigger Display or Camera capture section.

Customizing the Firmware

Once you have the basic out-of-box streaming video working, you can begin to customize software. The **main.c** file includes commented code that you can enable to perform various functions. After you change the **main.c** file:

1. Clean and then build the project.
2. Press CRESET pushbutton on the development board.

Note: You MUST reset the board before downloading the new **.elf** to the board. Otherwise the video will not display correctly.

3. Download the resulting **.elf** file to the board.

Apply Grayscale Filter

The default firmware passes through an RGB stream. To change it to grayscale, comment out the RGB code and uncomment the grayscale code for one or both cameras.

For example, to enable the grayscale filter on camera 2:

```
//SELECT RGB or grayscale output from camera pre-processing block (cam1)
EXAMPLE_APB3_REGW(EXAMPLE_APB3_SLV, EXAMPLE_APB3_SLV_REG3_OFFSET, 0x00000000);   //RGB
//EXAMPLE_APB3_REGW(EXAMPLE_APB3_SLV, EXAMPLE_APB3_SLV_REG3_OFFSET, 0x00000001); //grayscale

//SELECT RGB or grayscale output from camera pre-processing block (cam2)
//EXAMPLE_APB3_REGW(EXAMPLE_APB3_SLV, EXAMPLE_APB3_SLV_REG7_OFFSET, 0x00000000); //RGB
EXAMPLE_APB3_REGW(EXAMPLE_APB3_SLV, EXAMPLE_APB3_SLV_REG7_OFFSET, 0x00000001);   //grayscale
```

Apply Sobel Filter

You can adjust which video stream has the Sobel filter applied by commenting out the camera 1/camera 2 passthrough and uncommenting your filtering choice under `//SELECT HW accelerator mode - Processing`. By default, camera 1 and camera 2 are both pass through.

For example, to enable the Sobel filter on camera 2:

```
//write_u32(0x00000000, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET);
//2'd0: Cam source 1 - Passthrough;       Cam source 2 - Passthrough

write_u32(0x00000001, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET);
//2'd1: Cam source 1 - Passthrough;       Cam source 2 - Processed (Sobel)

//write_u32(0x00000002, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET);
//2'd2: Cam source 1 - Processed (Sobel); Cam source 2 - Passthrough

//write_u32(0x00000003, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG1_OFFSET);
//2'd3: Cam source 1 - Processed (Sobel); Cam source 2 - Processed (Sobel)
```

Note: If you want to use the Sobel filter for a camera stream, you also need to enable the grayscale filter for it. You cannot use the Sobel filter on an RGB stream.

Merge with Downscale Overlay

By default, the design merges cropped frame data from the two cameras. Camera one is cropped on the left side and camera two is cropped on the right side.

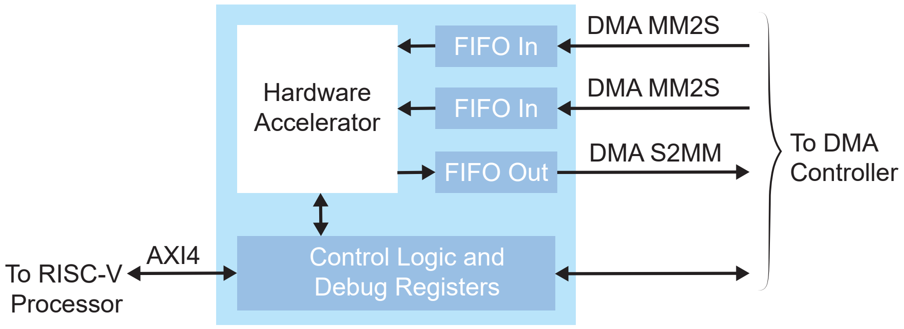
To show the video with camera two video downscaled and overlaid on the camera one stream, comment out the cropping comamnd and uncomment the line under `//SELECT HW accelerator mode - Merging`.

```
//1'b0: Merge frame data from cam source 1 (cropped by half, left side) and cam source 2
 (cropped by half, right side)
//write_u32(0x00000000, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG2_OFFSET);
//1'b1: Merge frame data from cam source 1 (main) and cam source 2 (downscaled overlay)
write_u32(0x00000001, EXAMPLE_AXI4_SLV+EXAMPLE_AXI4_SLV_REG2_OFFSET);
```

Using Your Own Hardware Accelerator

The wrapper for the Dual-Camera Hardware Accelerator is similar to the Image Signal Processing example, except for the number of inbound channels. The Dual-Camera example has two inbound DMA channels that read frame data in parallel from the two camera sources (stored in external memory) and send it to the Hardware Accelerator. If you want to replace the image merger with your own block, your system should be able to handle two input data streams.

Figure 24: Hardware Accelerator Connections



Refer to the following topics for other adjustments you may need to make to the wrapper and to the firmware:

- Slave Interface Changes on page 53
- DMA Controller Changes on page 54
- FIFO Changes on page 54
- RISC-V Firmware Changes on page 54

Chapter 4

Troubleshooting

Contents:

Error 0x80010135: Path too long (Windows)

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

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

Error 0x80010135: Path too long
```

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

OpenOCD Error: timed out while waiting for target halted

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

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

Memory Test

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

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

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

Set Up Terminal 1

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

Efinity RISC-V Embedded Software IDE Selected

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

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

   Windows:

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

   Linux:

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

   The OpenOCD server connects and begins listening on port 4444.

Legacy Eclipse IDE Selected

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

   Windows:

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

   Linux:

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

   The OpenOCD server connects and begins listening on port 4444.

Set Up Terminal 2

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

Close Terminals

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

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

Reset the FPGA

Press the reset button on your development board:
- *Trion® T120 BGA324 Development Board*—SW2
- 钛金系列 *Ti60 F225 Development Board*—SW3
- 钛金系列 *Ti180 M484 Development*
Board—SW1

OpenOCD error code (-1073741515)

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

OpenOCD Error: failed to reset FTDI device: LIBUSB_ERROR_IO

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

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

OpenOCD Error: 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 fails with an error similar to the following:

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

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

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

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.

OpenOCD Error: target 'fpga_spinal.cpu0' init failed

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

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

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

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

For example, either of the following are good:

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

Eclipse Fails to Launch with Exit Code 13

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

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

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

Efinity Debugger Crashes when using OpenOCD

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

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

Exception in thread "main"

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

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

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

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

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

   to

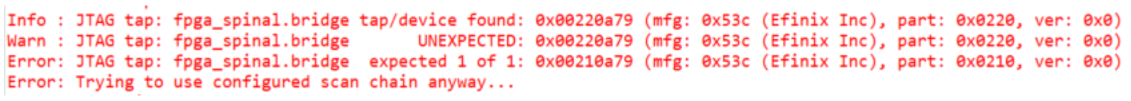
   ```
   LDSCRIPT ?= ${BSP_PATH}/linker/default_i.ld
   ```

3. Recompile the application.

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

Warning when Debug with softTap

You may receive the following warnings in the Efinity RISC-V Embedded Software IDE console when trying to debug a device other than T120F324 with soft debug tap interface. These warnings do not cause any issues to launch the debugging process.



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

To resolve the unwanted warnings, you may add the following command:

Debug Configuration > **Debugger** > **OpenOCD Setup** > **Config options**.

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

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

Eclipse Fails to Set UART in Terminal

You may receive the following error message in the UART terminal when trying to use UART interface:

```
Failed to open port COMn with selected setting.
The error from the serial driver:
Error setting DCB (31)
```

This error is typically caused because you did not install the USB-to-UART module driver.

Chapter 5

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)// Read regfile[t0] in mepc`

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_swap(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^{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_getFilteringHit(I2C_CTRL) == 1)`
	`// Check filter hit value, bit [7] from slave address,`
	`// read ='1' write ='0'`

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

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 status with a call back function.

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

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

gpio_getInput()

Usage	`gpio_getInput(GPIO_Reg)`
Parameters	[IN] GPIO_Reg struct of GPIO register
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`

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

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`

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

gpio_getOutput()

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

gpio_getOutputEnable()

Usage	`gpio_getOutputEnable(GPIO_Reg)`
Parameters	[IN] GPIO_Reg struct of GPIO register
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_setInterruptRiseEnable(GPIO_Reg, value)`
Parameters	[IN] GPIO_Reg struct of GPIO register [IN] value GPIO pin bitwise [IN] value GPIO Rise Interrupt Enable 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 [IN] value GPIO Fall Interrupt Enable 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 [IN] value GPIO High Interrupt Enable 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 [IN] value GPIO Low Interrupt Enable bitwise
Include	**driver/gpio.h**
Description	Set an interrupt when the GPIO is low.

I^2C API Calls

i2c_getFilteringHit()

Usage	`I2c_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 to register filter hit with a call back function.

Example	`if(i2c_getFilteringHit(I2C_CTRL) == 1)`
	`// Check filter hit value, bit [7] from slave address,`
	`// read ='1' write ='0'`

i2c_getFilteringStatus()

Usage	`I2c_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(i2c_getFilteringStatus(I2C_CTRL) == 1)`
	`// Check filter hit value, bit [7] from slave address,`
	`// read ='1' write ='0'`

i2c_getInterruptFlag()

Usage	`I2c_getInterruptFlag(reg)`
Parameters	[IN] `reg` 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 = i2c_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`

i2c_getMasterStatus()

Usage	`I2c_getMasterStatus(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	`int status = i2c_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`

i2c_applyConfig()

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

i2c_clearInterruptFlag()

Usage	`void i2c_clearInterruptFlag(u32 reg, u32 value)`
Parameters	[IN] `reg` struct of I^{2}C setting value [IN] `value` I^{2}C interrupt register
Include	**driver/i2c.h**
Description	Clear the I^{2}C 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: [0] I2C_INTERRUPT_RX_DATA [1] I2C_INTERRUPT_RX_ACK [2] I2C_INTERRUPT_TX_DATA [3] I2C_INTERRUPT_TX_ACK [4] I2C_INTERRUPT_START [5] I2C_INTERRUPT_RESTART [6] I2C_INTERRUPT_END [7] I2C_INTERRUPT_DROP [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: [0] I2C_INTERRUPT_RX_DATA [1] I2C_INTERRUPT_RX_ACK [2] I2C_INTERRUPT_TX_DATA [3] I2C_INTERRUPT_TX_ACK [4] I2C_INTERRUPT_START [5] I2C_INTERRUPT_RESTART [6] I2C_INTERRUPT_END [7] I2C_INTERRUPT_DROP [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 state.

i2c_masterRestart()

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

i2c_masterStartBlocking()

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

i2c_masterRestartBlocking()

Usage	`void i2c_masterRestartBlocking(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Restart the I^{2}C master by sending a start condition. Wait for the master to start the process.

i2c_masterStop()

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

i2c_masterStopBlocking()

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

i2c_masterStopWait()

Usage	`void i2c_masterStopWait(u32 reg)`
Parameters	[IN] `reg` struct of I^{2}C register
Include	**driver/i2c.h**
Description	Asserts a wait condition and waits for the master to be available.

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	Transmit knowledge and wait for it to complete.

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	Transfer a NACK and wait for the completion.

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
Returns	[OUT] 8-bit data
Description	Read 8-bit data from the specified address.

read_u16()

Usage	`u16 read_u16(u32 address)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address
Returns	[OUT] 16-bit data
Description	Read 16-bit data from the specified address.

read_u32()

Usage	`u32 read_u32(u32 address)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address
Returns	[OUT] 32-bit data
Description	Read 32-bit data from the specified address.

write_u8()

Usage	`void write_u8(u8 data, u32 address)`
Include	**driver/io.h**
Parameters	[IN] `data` SoC address data [IN] `address` SoC address
Description	Write 8 bits unsigned data to the specified address.

write_u16()

Usage	`void write_u16(u16 data, u32 address)`
Include	**driver/io.h**
Parameters	[IN] `data` SoC address data [IN] `address` SoC address
Description	Write 16 bits unsigned data to the specified address.

write_u32()

Usage	`void write_u32(u32 data, u32 address)`
Include	**driver/io.h**
Parameters	[IN] `data` SoC address data [IN] `address` SoC address
Description	Write 32 bits unsigned data to the specified address.

write_u32_ad()

Usage	`void write_u32_ad(u32 address, u32 data)`
Include	**driver/io.h**
Parameters	[IN] `address` SoC address [IN] `data` SoC address data
Description	Write 32 bits unsigned data to the specified address.

Core Local Interrupt Timer API Calls

clint_setCmp()

Usage	`void clint_setCmp(u32 p, u64 cmp, u32 hart_id)`
Include	**driver/clint.h**
Parameters	[IN] `p` timer interrupt [IN] `cmp` timer compare register [IN] `hart_id` hart ID
Description	Set a timer value to trigger an interrupt when the value is reached.

clint_getTime()

Usage	`u64 clint_getTime(u32 p)`
Include	**driver/clint.h**
Parameters	[IN] `p` 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] value `prescaler` value
Description	Set the user timer prescaler value.

timer_setConfig()

Usage	`void timer_setConfig(u32 reg, u32 value)`
Include	**driver/timer.h**
Parameters	[IN] `reg` 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 by setting it to 0.

PLIC API Calls

plic_set_priority()

Usage	`void plic_set_priority(u32 plic, u32 gateway, u32 priority)`
Include	**driver/plic.h**
Parameters	[IN] `plic` PLIC 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 for specific HART.

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 for specific HART.

SPI API Calls

spi_applyConfig()

Usage	`void spi_applyConfig(u32reg, 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	`u32 spi_cmdAvailability(u32reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register
Returns	[OUT] SPI TX FIFO availability (16 bits)
Description	Reads the number of bytes for TX FIFO availability.

spi_diselect()

Usage	`void spi_diselect(u32reg, u32 slaveId)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `slaveId` ID for the slave
Description	De-asserts the selected SPI (SS) pin based on the slaveId.

spi_read()

Usage	`u8 spi_read(u32 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_read32()

Usage	`u32 spi_read32(u32 reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register
Returns	[OUT] `reg` Data (up to 16 bits)
Description	Receives up to 16 bits of data from the SPI slave.

spi_rspOccupancy()

Usage	`u32 spi_rspOccupancy(u32 reg)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register
Returns	[OUT] SPI RX FIFO occupancy (16 bits)
Description	Read the number of bytes for RX FIFO occupancy .

spi_select()

Usage	`void spi_select(u32 reg, u32slaveId)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `slaveId` ID for the slave
Description	Asserts the SPI select (SS) pin on the selected slave.

spi_write()

Usage	`void spi_write(u32reg, u8 data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `data` 8 bits of data to send out
Description	Transfers one byte to the SPI slave.

spi_write32()

Usage	`void spi_write32(u32 reg, u32 data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `data` up to 16 bits of data to send out
Description	Transfers up to 16 bits to the SPI slave.

spi_writeRead()

Usage	`u8 spi_writeRead(u32 reg, u8 data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` 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_writeRead32()

Usage	`u32 spi_writeRead32(u32 reg, u32 data)`
Include	**driver/spi.h**
Parameters	[IN] `reg` struct of the SPI register [IN] `data` up to 16 bits of data to send out
Returns	[OUT] `reg` Up to 16 bits of data
Description	Transfers up to 16 bits of data to the SPI slave and receives up to 16 bits of data from the SPI slave.

SPI Flash Memory API Calls

spiFlash_f2m_()

Usage	`void spiFlash_f2m_(u32 spi, u32 flashAddress, u32 memoryAddress, u32 size)`
Include	**driver/spiFlash.h**
Parameters	[IN] `spi` 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(u32 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(u32 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(u32 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(u32 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_(u32 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(u32 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(u32 spi, u32 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	[OUT] 8-bit SPI flash ID
Description	Read the ID from the flash.

spiFlash_read_id()

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

spiFlash_select()

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

spiFlash_select_withGpioCs()

Usage	`spiFlash_select_withGpioCs(u32 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(u32 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_(u32 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(u32 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(u32 spi, u32 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	`void uart_applyConfig(u32 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_TX_emptyInterruptEna()

Usage	`void uart_TX_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_RX_NotemptyInterruptEna()

Usage	`void uart_RX_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(u32reg)`
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	`u32 uart_readOccupancy(u32reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register
Returns	[OUT] `reg` FIFO occupancy
Description	Read the number of bytes in the RX FIFO.

uart_status_read()

Usage	`u32 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	Refers to UART Status Register: 0x0000_0004 in the Sapphire Datasheet.

uart_status_write()

Usage	`uart_status_write(u32 reg, char data)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register [IN] `data` input data for the UART status.
Description	Write the UART status.

uart_write()

Usage	`void uart_write(u32 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(u32 reg, const 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 .

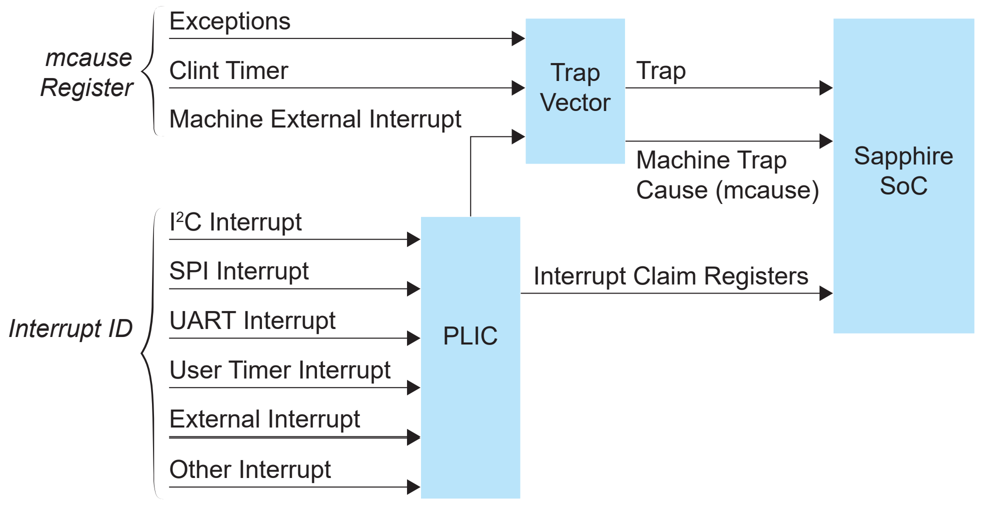
uart_writeAvailability()

Usage	`u32 uart_writeAvailability(u32 reg)`
Include	**driver/uart.h**
Parameters	[IN] `reg` struct of the UART register
Returns	[OUT] `reg` FIFO availability
Description	Read the number of bytes in the TX FIFO.

Handling Interrupts

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

Figure 25: Types of Interrupts

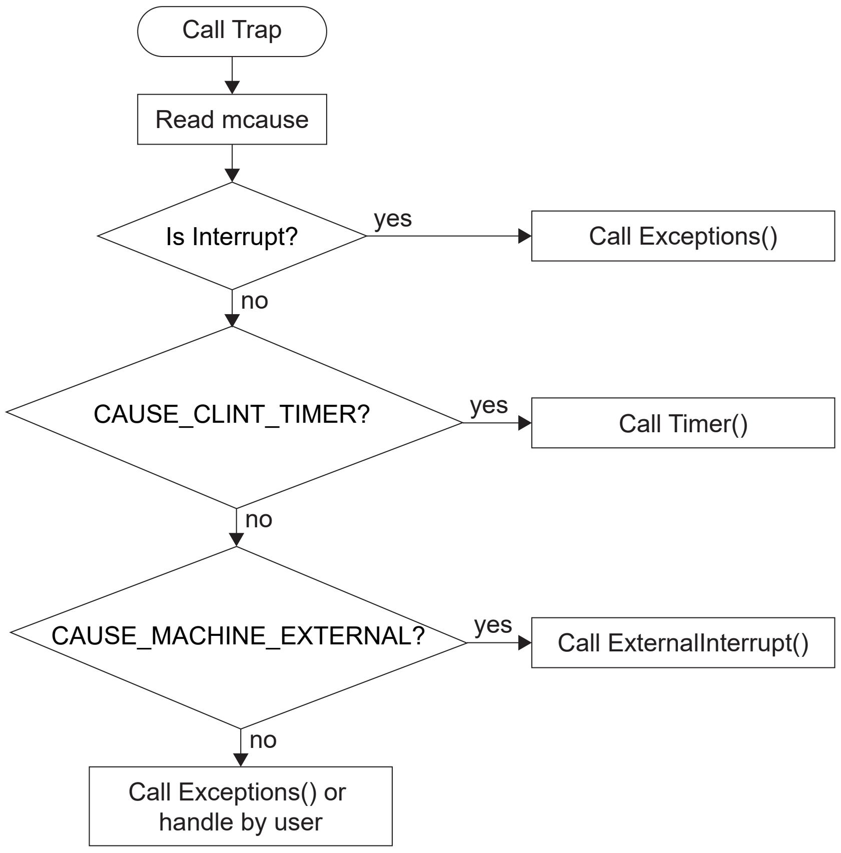


Trap Vectors

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 26: Handling Trap Vectors



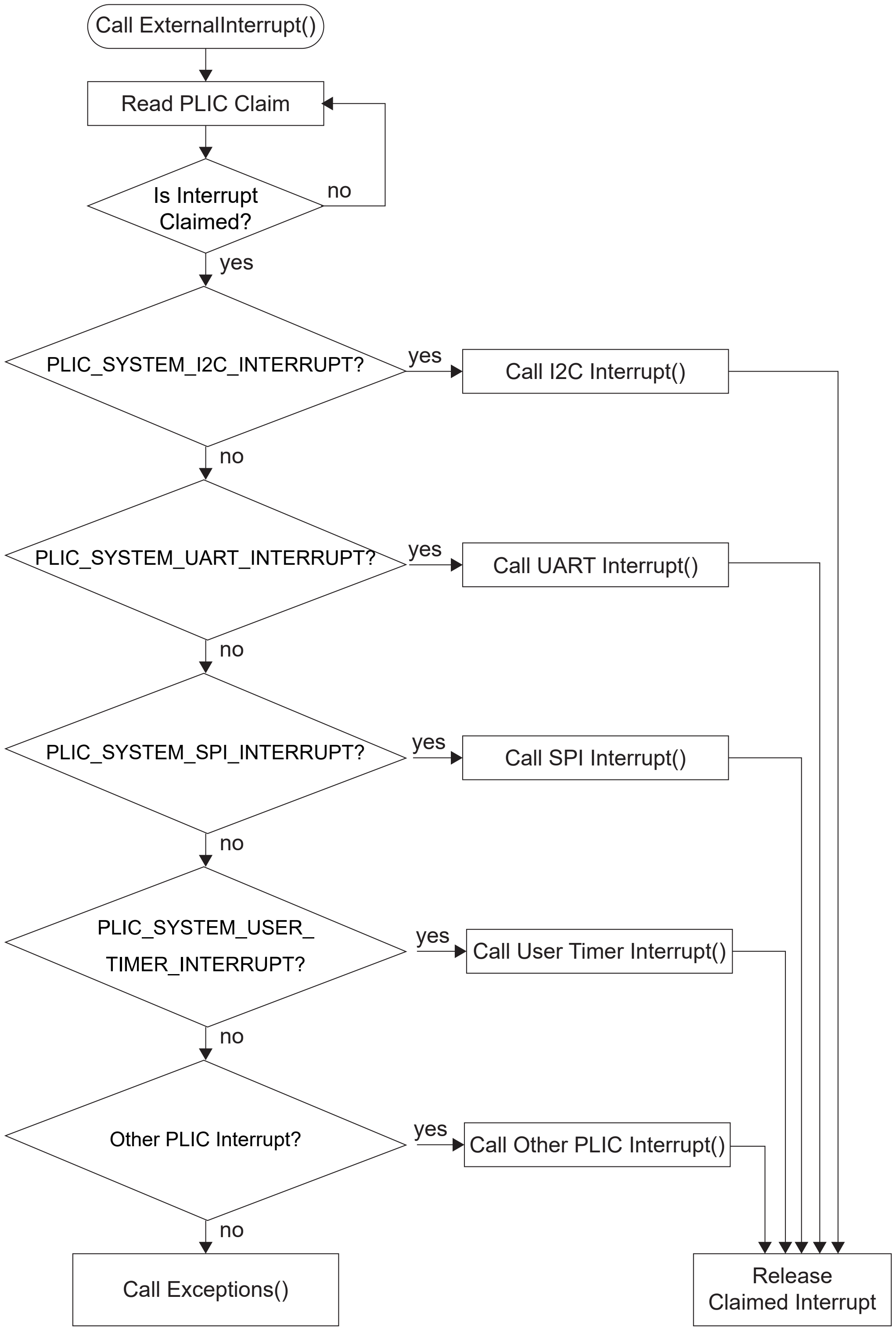
PLIC Interrupts

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 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 27: Handling PLIC Interrupts


Call ExternalInterrupt()
Read PLIC Claim
Is Interrupt Claimed?
no
yes
PLIC_SYSTEM_I2C_INTERRUPT?
yes
Call I2C Interrupt()
no
PLIC_SYSTEM_UART_INTERRUPT?
yes
Call UART Interrupt()
no
PLIC_SYSTEM_SPI_INTERRUPT?
yes
Call SPI Interrupt()
no
PLIC_SYSTEM_USER_ TIMER_INTERRUPT?
yes
Call User Timer Interrupt()
no
Other PLIC Interrupt?
yes
Call Other PLIC Interrupt()
no
Call Exceptions()
Release Claimed Interrupt

Appendix: Software Requirement

Install the RISC-V SDK

To install the SDK:

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

Windows directory structure:

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

Ubuntu directory structure:

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

Install the Java JRE

To install the JRE:

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

Note: You need a 64-bit version of the Java JRE. When you launch Eclipse using a 32-bit version, you get an error that Java quits with exit code `13`.

Appendix: Using Eclipse and OpenOCD (RISC-V SDK)

You use Eclipse to manage your software projects and OpenOCD for debugging. The following sections explain how to set up your Eclipse environment, create a project and build it. Additionally, this section walks you through how to set up the OpenOCD debugger for use with the Sapphire SoC.

Note: The instructions in this user guide assume you are using the RISC-V SDK v1.4 or higher.

Launch Eclipse

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 environment variables for the Sapphire BSP, and launches Eclipse. Always use this executable to launch Eclipse; do not launch Eclipse directly.

Follow these steps to launch Eclipse:

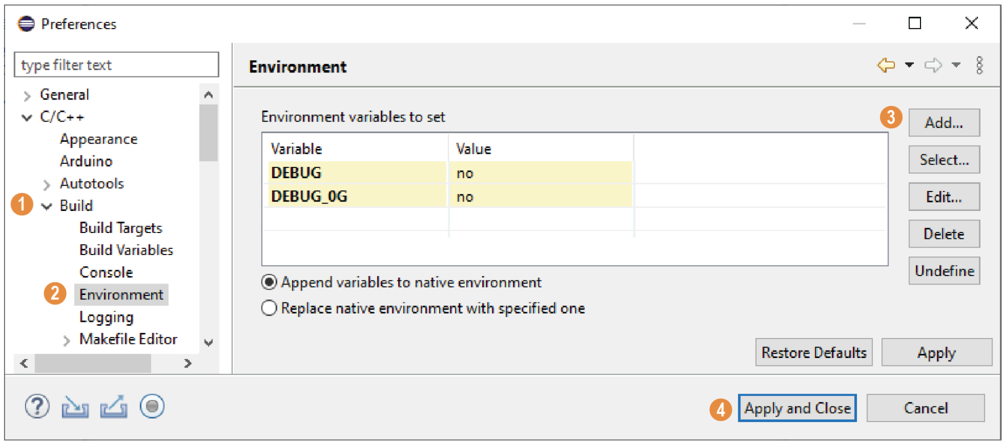
1. Launch Eclipse using the **run_eclipse.bat** file (Windows) or **run_eclipse.sh** file.
2. The launch script prompts you to select your SoC. Type 1 for Sapphire and press enter.
3. If this is the first time you are running Eclipse, create a new workspace that points to **.../<project>/embedded_sw/SapphireSoc**.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

Note: 易灵思 provides several RISC-V SoCs. Using the top-level SoC directory as your workspace means that you can explore more than one SoC by simply switching workspaces.

Set Up the Eclipse Workspace

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 RISC-V software code in the same place and you can set global environment variables that apply to all software projects in your workspace.

1. Create your workspace environment variables; these variables apply to all projects in your workspace. Choose **Window > Preferences** to open the **Preferences** window.



Variable	Value
DEBUG	no
DEBUG_OG	no

(● Append variables to native environment) ○ Replace native environment with specified one

2. In the left navigation menu, 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**.

Create a New Project

Note: The following steps use the evsoc_ispExample as an example to explain the steps to create a new project. You can refer to these steps to import and build other EVSoC code examples provided.

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

Note: You can skip steps 1 to 3 if you have already launch Eclipse and set up the Eclipse workspace.

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 **evsoc_ispExample** example:

4. Choose **File** > **New** > **Makefile Project with Existing Code**.
5. Click **Browse** next to **Existing Code Location**.
6. Browse to the **software/standalone/evsoc_ispExample** 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	..\<project>\embedded_sw\SapphireSoc\config\project_settings_soc.xml
Linux	../<project>/embedded_sw/ SapphireSoc/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.

Revision History

Table 18: Revision History

Date	Version	Description
February 2023	5.0	Updated Installing Windows USB Driver. Updated table in topic Install the Edge Vision SoC. Replaced topic Install the RISC-V SDK → Install the Efinity RISC-V Embedded Software IDE. Moved Install the RISC-V SDK to appendix. Added 钛金系列 Ti180 M484 Development Board in sub-topic Required Hardware. Added in hardware setup for 钛金系列 Ti180 M484 Development Board in sub-topic 钛金系列 Development Boards. Updated topic Installing USB Drivers. Added note and steps to program the FPGA bitstream into SPI flash in topic Program the Development Board. Replaced topics in Using Eclipse and OpenOCD with topics from Efinity RISC-V Embedded Software IDE: • Launch Efinity RISC-V Embedded Software IDE • Create, Import and Build a Software Project • Launch the Debug Script Moved topics in Using Eclipse and OpenOCD to appendix. Added sub-topic Launch the Run Script under the topic Run the Application and topic Debug with Efinity RISC-V Embedded Software IDE. Added in table and images for 钛金系列 Ti180 M484 and image for Trion T120 BGA324 in topic Using the Efinity Debugger and OpenOCD Simultaneously. Updated title in Figure: Image Signal Processing Example Block Diagram 钛金系列 Ti60 F225 Development Board. Added in figure for 钛金系列 Ti180 M484 Development Board in the topic Image Signal Processing Example Design. Updated tables in Trion T120 BGA324 Development Kit Resource Utilization, TitaniumTi60 F225 Development Kit Resource Utilization and 钛金系列 Ti180 M484 Development Kit Resource Utilization. Added in Ti80 design description in topic Video Capture and Pre-Processing. Updated sub-topic About the Software in both main topic of Image Signal Processing Example Design and Dual Camera example Design. Added in sub-topics Exception in thread "main" and Warning when Debug with softTap in topic Troubleshooting.
June 2022	4.2	Corrected Hardware Setup Trion® T120 BGA576 Development Board figure. (DOC-832)
March 2022	4.1	Added note about example design bit files Updated Dual-camera design resource utilization table.
March 2022	4.0	Updated for RISC-V update from Ruby to Sapphire. Updated design resource utilization. Updated example design directories.

Date	Version	Description
November 2021	3.0	Added link to OpenJDK (DOC-457) Added information about the flow for handling interrupts in the API Reference chapter. (DOC-398) Updated the GPIO, I2C, and UART API calls. (DOC-454) Rearranged chapters and topics. Added Trion® T120 BGA576 Development Board and 钛金系列 Ti60 F225 Development Board support.
April 2021	2.0	Added description of flash write and read software and dual-camera example design. (DOC-395) Added instructions on debugging with the Efinity® Debugger and OpenOCD simultaneously.
February 2021	1.4	Updated Edge Vision SOC framework block diagram.
January 2021	1.3	Updated image signal processing block diagram.
January 2021	1.2	Updated the jumper settings. Added note that the SoC does not support floating-point calculations.
December 2020	1.1	Updated to support 1280 x 720 display resolution. Added description for evsoc_ispExample_demo, evsoc_ispExample_demo2, and evsoc_ispExample_timestamp examples. Updated the steps for copying a user binary to flash.
November 2020	1.0	Initial release.