Sapphire RV64 SoC User Guide

UG-SAPPHIRE-RV64-v1.0
May 2026
www.efinixinc.com

Copyright © 2026. All rights reserved. Efinix, the Efinix logo, Efinity, Quantum, Titanium, Topaz, and Trion are trademarks of Efinix, Inc. All other trademarks and service marks are the property of their respective owners. All specifications subject to change without notice.

Contents

Introduction.....iii
Efinity® RISC-V Embedded Software IDE.....v
Required Software.....vi
Required Hardware.....vi

Chapter 1: Install Software and SoC.....7
Install the Efinity Software.....7
Install the Efinity RISC-V Embedded Software IDE.....8

Chapter 2: IP Manager.....9
Customizing the Sapphire RV64 SoC.....12
SoC Configuration Guideline.....17
Modify the Bootloader.....18
Updating Bootloader with Efinity BRAM Initial Content Updater.....21

Chapter 3: Program the Board with the Sapphire RV64 RTL Design.....24
About the Example Design.....25
Installing USB Drivers.....27
Program the Development Board.....29

Chapter 4: Simulate.....30

Chapter 5: Boot Sequence.....31
Boot Sequence: Case A.....32
Boot Sequence: Case B.....33
Boot Sequence: Case C.....34
Booting Multiple Cores.....34

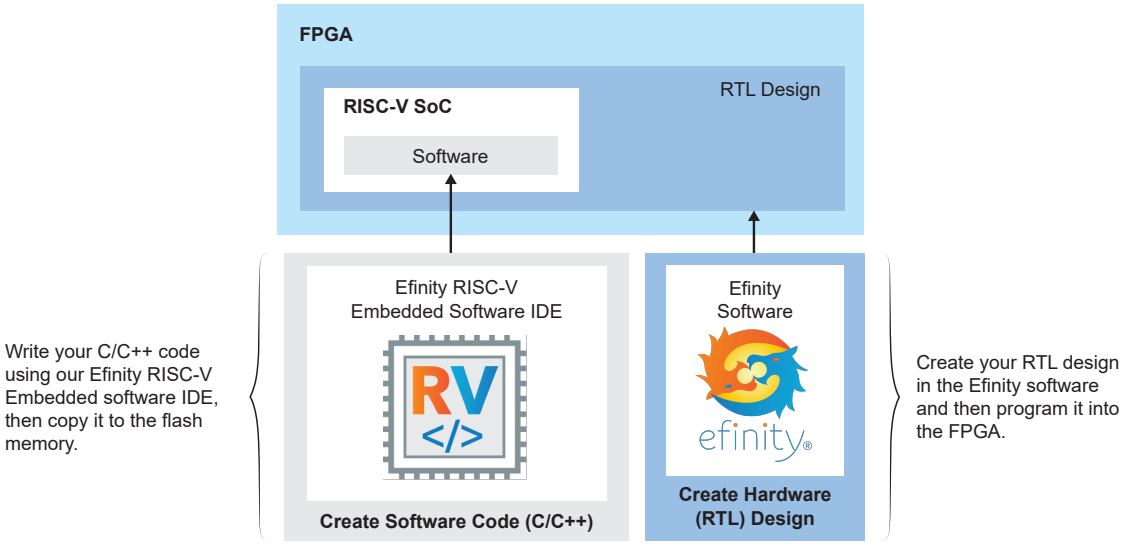
Chapter 6: Create Your Own RTL Design.....36
Target another FPGA.....37
Target another Efinix Board.....38
Target Your Own Board.....39
Create a Custom AXI4 Master Peripheral.....41
Create a Custom APB3 Peripheral.....41
Use another DDR DRAM Module (Trion Only).....41
Use the I^2C Interface for DDR Calibration (Trion only).....42
Remove Unused Peripherals from the RTL Design.....42

Chapter 7: Create Your Own Software.....43
Deploying an Application Binary.....44
Boot from a Flash Device.....44
Boot from the OpenOCD Debugger.....44
Copy a User Binary to Flash (Efinity Programmer).....45
About the Board Specific Package.....46
List of BSP Files.....47
Revision History.....48

Introduction

Efinix provides a configurable, cacheable soft Sapphire RV64 SoC, that optionally includes a DDR DRAM controller interface. The Sapphire RV64 SoC supports a variety of peripherals. You can choose which peripherals you want by configuring the SoC in the IP Manager. This core is similar to the open-source VexiiRiscv and has been optimized for Titanium and Trion FPGAs.

Figure 1: Designing Hardware and Software for the Sapphire RV64 SoC



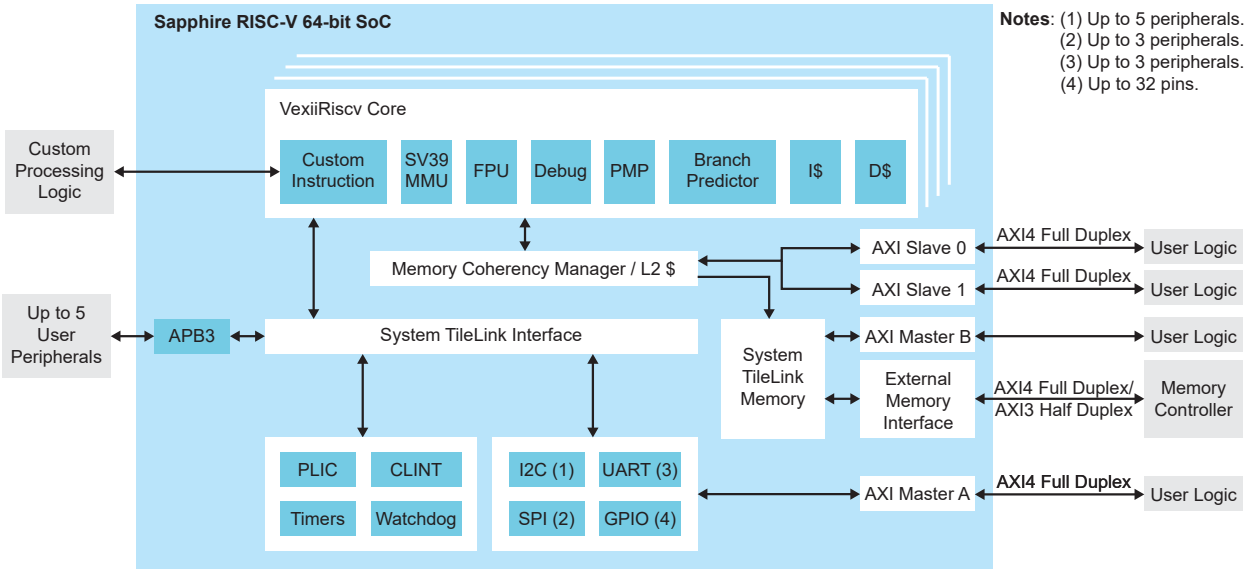
The Sapphire RV64 SoC incorporates 1 to 4 64-bit RISC-V processors that have an instruction cache and data cache with up to 8-ways, each way is equipped with 4 KB, 64 KB to 512 KB 8-ways L2 cache, 4 - 512 KB of on-chip RAM, and a variety of peripherals (including 1 - 5 APB3 slave peripherals and 2 AXI slaves and 2 AXI masters). You can configure the operating frequency from 20 to 400 MHz (the design's f_{MAX} limits the actual performance).

Other optional CPU micro-architectures, such as floating-point unit (FPU), a dynamic branch predictor, software and hardware prefetchers, custom instruction interfaces, physical memory protection, and an SV39 memory management unit (MMU), can be configured with the IP manager.

The SoC includes a CLINT timer, a platform local interrupt controller, a watchdog timer, 1 - 5 I^2C peripherals, 1 - 3 UARTs, 1 - 3 user timers, 1 - 8 user interrupts, and 1 - 3 SPI masters. The default configuration supports a half-duplex and a full-duplex AXI bus with up to 512 bits to communicate with the Efinix LPDDR4x controller, DDR3 controller, or HyperRAM controller.

- DDR controller—This core uses the Trion or Titanium FPGAs hard DDR DRAM interface to reset an external DRAM module (resets and reinitializes the Trion or Titanium FPGA's DDR interface, which includes the DDR module(s)).
- HyperRAM controller—This core controls HyperRAM memory modules. You can customize the SoC using the IP Manager in the Efinity® software.

Figure 2: Sapphire RV64 SoC Block Diagram



Learn more:

- Refer to the **Sapphire RV64 SoC Data Sheet** for detailed specifications on the SoC.
- Refer to the **Efinity RISC-V Embedded Software IDE User Guide** for information on the IDE software.
- For information on the API and other detailed information regarding the SoC, refer to the on-line documentation at **RISC-V API and Examples**.

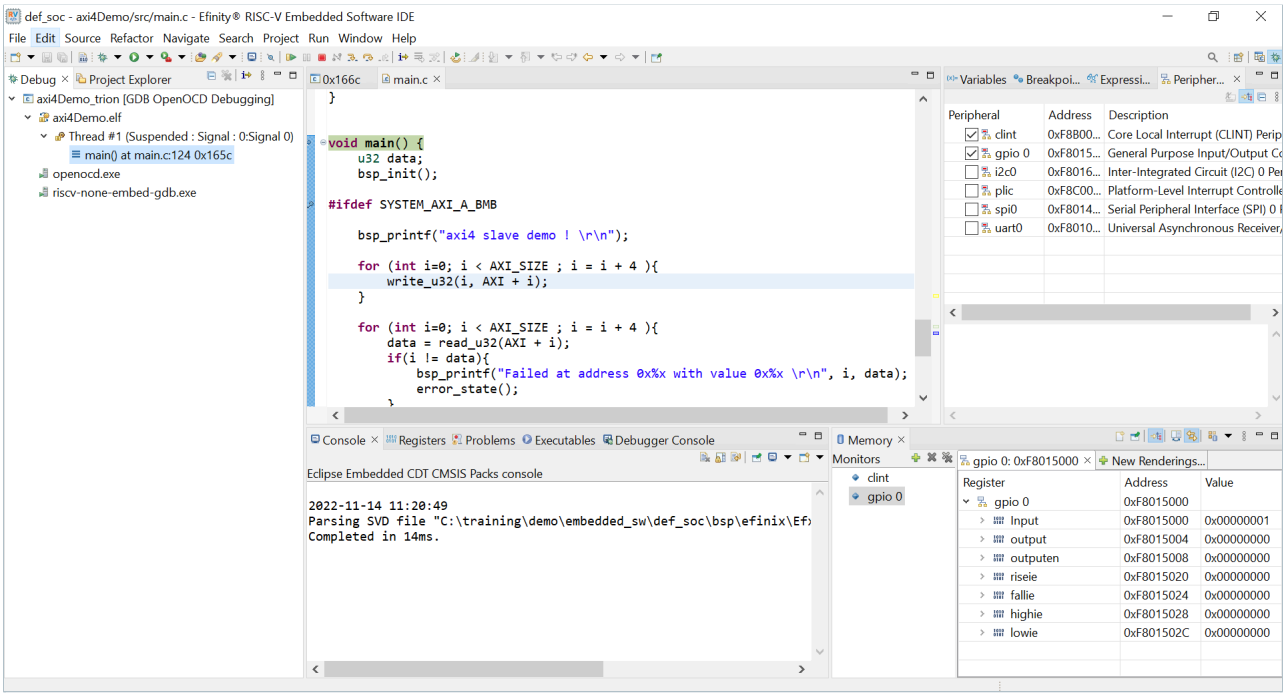
Efinity® RISC-V Embedded Software IDE

The Efinity® RISC-V Embedded Software IDE is an Eclipse-based Integrated Development Environment (IDE) powered by Ashling's ***RiscFree***™ IDE for Sapphire RV32 and RV64 SoC. It provides a complete and seamless environment for RISC-V C and C++ software development.

Features include:

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

Figure 3: Efinity RISC-V Embedded Software IDE



Required Software

To write software for the Sapphire RV64 SoC, you need the following tools. The Efinity RISC-V Embedded Software IDE installer for Windows and Linux operating systems are available in the **Efinity page of the Support Center**.

Efinity® Software

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

Efinity RISC-V Embedded Software IDE

The Efinity RISC-V Embedded Software IDE is an Eclipse-based Integrated Development Environment (IDE) powered by Ashling's ***RiscFree***™ IDE for Sapphire RV64 SoC and provides a complete provides a complete, seamless environment for RISC-V C and C++ software development. The RISC-V IDE includes the following packages:

Disk space required: 4 GB (Windows and Linux)

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

Version: 13.4.0

Disk space required: 1.43 GB (Windows), 1.40 GB (Linux)

OpenOCD Debugger—The open-source Open On-Chip Debugger (OpenOCD) software includes configuration files for many debug adapters, chips, and boards. Many versions of OpenOCD are available. The Efinix RISC-V flow requires a custom version of OpenOCD that includes the VexRiscv 32-bit and VexiiRiscv 64-bit RISC-V processor.

Version: 0.11.0 (2026-03-24)

Disk space required: 4.23 MB (Windows), 3.7 MB (Linux)

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

Required Hardware

- Titanium Ti375 C529 Development Board
- 12 V power cable
- USB-C cable
- Computer or laptop

Chapter 1

Install Software and SoC

Contents:

- Install the Efinity Software
- Install the Efinity RISC-V Embedded Software IDE

Install the Efinity Software

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

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

Install the Efinity RISC-V Embedded Software IDE

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

To install the Efinity RISC-V Embedded Software IDE:

Windows

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

Linux

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

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

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

Note:

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

Chapter 2

IP Manager

Contents:

- Customizing the Sapphire RV64 SoC
- SoC Configuration Guideline
- Modify the Bootloader

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

The IP Manager consists of:

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

Generating Sapphire RV64 SoC with the IP Manager

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

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

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

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

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

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

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

Generated RTL Files

The IP Manager generates these files and directories:

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

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

Generated Software Code

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

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

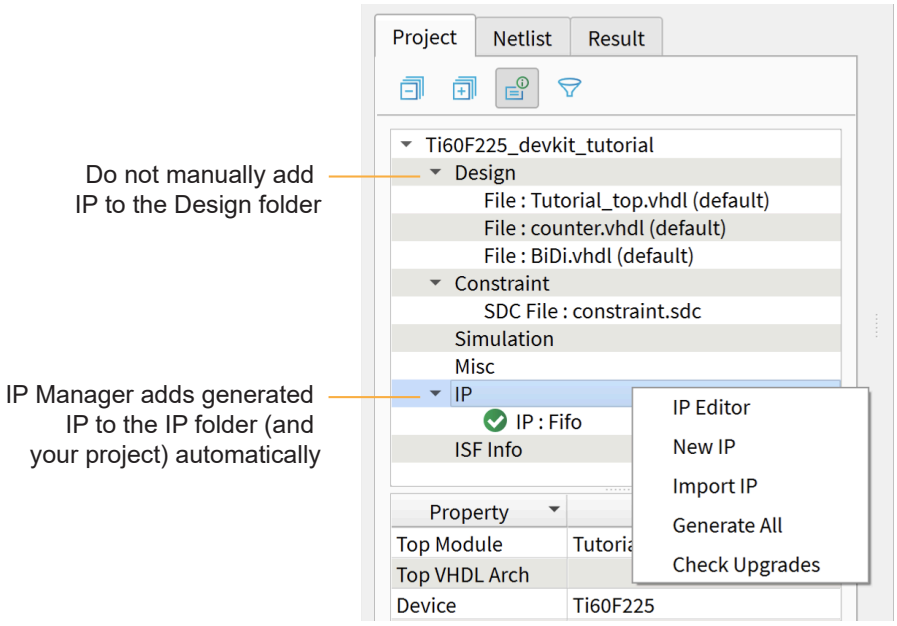
Instantiating the SoC

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

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

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

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



Customizing the Sapphire RV64 SoC

You customize the Sapphire RV64 SoC using the IP Configuration wizard. The parameters are arranged on tabs so you can click through them more easily.

Table 1: Sapphire RV64 SoC CPU and ISA Tab Parameters

Parameter	Option	Description
Number of cores	1 - 4	Enter the number of CPU cores. Default: 1
Atomic Extension	On, off	Choose whether to enable atomic extension instruction support.
Compressed Extension	On, off	Choose whether to enable compressed instruction support.
Floating-point Extension	F-Extension only, F and D-Extension	Choose whether to enable single or double precision for the FPU.
Bit Manipulation Extension	On, off	Choose whether to enable bit manipulation instruction support.
Cache Block Management Extension	On, off	Choose whether to enable cache management instruction support

Table 2: Sapphire RV64 SoC CPU Microarchitecture Tab Parameters

Parameter	Option	Description
Branch Prediction	On, off	Choose whether to enable branch predictor with BTB, RAS and Gshare.
Hardware Prefetcher	On, off	Choose whether to enable hardware prefetcher with RPT algorithm.
Software Prefetcher	On, off	Choose whether to enable software prefetcher.
Supervisor Mode	On, off	Choose whether to enable supervisor mode with SV39 MMU.
Physical Memory Protection	On, off	Choose whether to enable physical memory protection.
Custom Instruction	On, off	Choose whether to enable custom instruction

Table 3: Sapphire RV64 SoC Memory Tab Parameters

Parameter	Option	Description
L1 Data and Instruction Cache way	1, 2, 4 or 8	Choose the L1 data and instruction cache way number (4KB per way).
L2 Cache	On, off	Choose whether to enable L2 cache.
L2 Cache Size	64KB, 128KB, 256KB and 512KB	Choose the L2 cache size.
On-Chip RAM Size	4 KB, 8 KB, 16 KB, 24 KB, 32 KB, 48 KB, 64 KB, 80 KB, 96 KB, 128 KB, 144 KB, 160 KB, 192 KB, 224 KB, 256 KB, 384 KB, 512 KB	Choose the size of the internal BRAM. Default: 4 KB
External Memory Interface	On, off	Instantiate the external memory interface. Default: On Off: The external memory interface is not used.

AXI Interface Type	AXI3, AXI4	Use an AXI4 full duplex interface. Default: AXI4 AXI3: Use an AXI3 half duplex interface.
External Memory Data Width	32, 64, 128, 256, 512	Choose the data width for the AXI interface. Default: 256
External Memory AXI ID Width	6, 8	Choose the width of AXI ID.
External Memory Address Size	4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, 0.5 GB, 1 GB, 1.5 GB, 2 GB, 2.5 GB, 3 GB, 3.5 GB,4.0 GB, 4.5 GB, 5.0 GB, 5.5 GB, 6.0 GB, 6.5 GB, 7.0 GB, 7.5 GB, 8.0 GB	Choose the address size for the AXI interface. Default: 1 GB

Table 4: Sapphire RV64 SoC Clock Tab Parameters

Parameter	Option	Description
Frequency (MHz)	20 - 400	Enter the frequency in MHz. Default: 200
Peripheral Clock	On, off	Choose whether you want to run a dedicated clock for the APB3 slaves (SPI, I2C, GPIO, UART, and user timer) and AXI4 master A.
Peripheral Clock Frequency (MHz)	20 - 200	Enter the peripheral clock frequency in MHz.

Table 5: Sapphire RV64 Debug Tab Parameters

Parameter	Option	Description
FPGA Tap Port	1, 2, 3, 4	Choose which Tap port you want to target with the OpenOCD Debugger. This option is only applicable when you are using the JTAG User Tap interface block to communicate with the OpenOCD Debugger.
Hardware Breakpoint	Disabled, 1 - 4	Number of hardware breakpoints. This hardware breakpoint is a program type breakpoint.
Performance Counter	Disabled, 1 - 9	Number of performance counter.
Co-Debug	On, off	Choose whether co-debug mode to share the tap with FPGA Debugger.

Parameter	Option	Description
Target Board/ Cable/Module	Trion T120 BGA324 Development Board Trion T120 BGA576 Development Board Trion T20 BGA256 Development Board Xyloni Titanium Ti60 F225 Development Board Titanium Ti180 J484 Development Board Titanium Ti375 N1156 Titanium Ti375 C529 Topaz Tz170 J484 C232HM-DDHSL-0 (Soft debug) FTDI Module FT2232H (Soft debug) FTDI Module FT4232 (Soft debug) ISX-DLC_EF001 Programming Cable Custom	Choose which board you want to target with OpenOCD. Choose **Custom** to target your own board.
Application Region Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB	Modify the linker script to outline the region for the user application. This option is only applicable for SoCs with external memory. For SoCs with internal memory, the region size is determined by the on-chip RAM size.
Application Heap Size	256 B, 512 B, 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB	Modify the linker script to specify the application heap size. The size is limited to a maximum of 60% of region size.
Application Stack Size	256 B, 512 B, 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB	Modify the linker script to specify the application stack size. The size is limited to a maximum of 60% of region size.
FreeRTOS Region Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB	Modify the linker script to outline the region for the FreeRTOS. This option is only applicable for SoCs with external memory. For SoCs with internal memory, the region size is determined by the on-chip RAM size.
FreeRTOS Heap Size	256 B, 512 B, 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB	Modify the linker script to specify the FreeRTOS heap size. The size is limited to a maximum of 60% of region size.
FreeRTOS Stack Size	256 B, 512 B, 1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB	Modify the linker script to specify the FreeRTOS stack size. The size is limited to a maximum of 60% of region size.

Table 6: Sapphire RV64 User Interface Tab Parameters

Where *n* is 0 or 1.

Parameter	Option	Description
Number of APB3	Disable, 1-5	Choose number of instantiations of APB3 slaves.
APB3 Slave Address Size	4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1MB	Choose the APB slave size. This setting applies to all APB slaves. Default: 64 KB
AXI Master A	On, Off	On: Instantiate the interface. Off: Do not use the interface.
AXI Master A Address Size	1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB	Choose the size of the AXI Master A.
AXI Master B	On, Off	On: Instantiate the interface. Off: Do not use the interface.
AXI Master B Address Size	1 MB, 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB, 512 MB, 1 GB, 2 GB, 4 GB, 8 GB, 16 GB, 32 GB, 64 GB	Choose the size of the AXI Master B.
AXI Master B Data Width	32, 64, 128, 256, 512	Choose the data width of the AXI Master B.
AXI Slave *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
AXI Slave *n* Data Width	32, 64, 128, 256, 512	Choose the width of the AXI Slave. The AXI Slave width must not be larger than the external memory data width.

Table 7: Sapphire RV64 GPIO Tab Parameters

Where *n* is 0 or 1.

Parameter	Options	Description
GPIO *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
GPIO *n* Bit Width	1, 2, 4, 8, 16, 32	Choose the number of pins for the GPIO. Default: 4 (GPIO 0), 8 (GPIO 1)

Table 8: Sapphire RV64 UART Tab Parameters

Where *n* is 0 or 1

Parameter	Options	Description
UART *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
UART *n* FIFO Depth	64, 128, 256	Specify the depth of UART's FIFO.

Table 9: Sapphire RV64 I2C Tab Parameters

Parameter	Options	Description
I2C	Disabled, 1 - 5	Choose the number for the I^2C.

Table 10: Sapphire RV64 SPI Tab Parameters

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

Parameter	Options	Description
SPI *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
SPI *n* Data Width	8 - 16	Configure the data width for the SPI interface. **Note:** Only applicable for SPI 1 and SPI 2.
SPI *n* Chip Select Width	1 - 8	Choose the number of Chip select required for the SPI interface.
SPI *n* FIFO Depth	64, 128, 256, 512, 1024, 2048	Specify the depth of SPI's FIFO.

Table 11: Sapphire RV64 User Interrupt Tab Parameters

Parameter	Options	Description
User Interrupt	Disabled, 1 - 8	Choose the number of the user interrupt.

Table 12: Sapphire RV64 User Timer Tab Parameters

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

Parameter	Options	Description
User Timer *n*	On, off	On: Instantiate the interface. Off: Do not use the interface.
User Timer *n* Counter Width	12, 16, 32	Choose the counter bit width. Default: 12
User Timer *n* Prescaler Width	8, 16	Choose the prescaler bit width. Default: 8

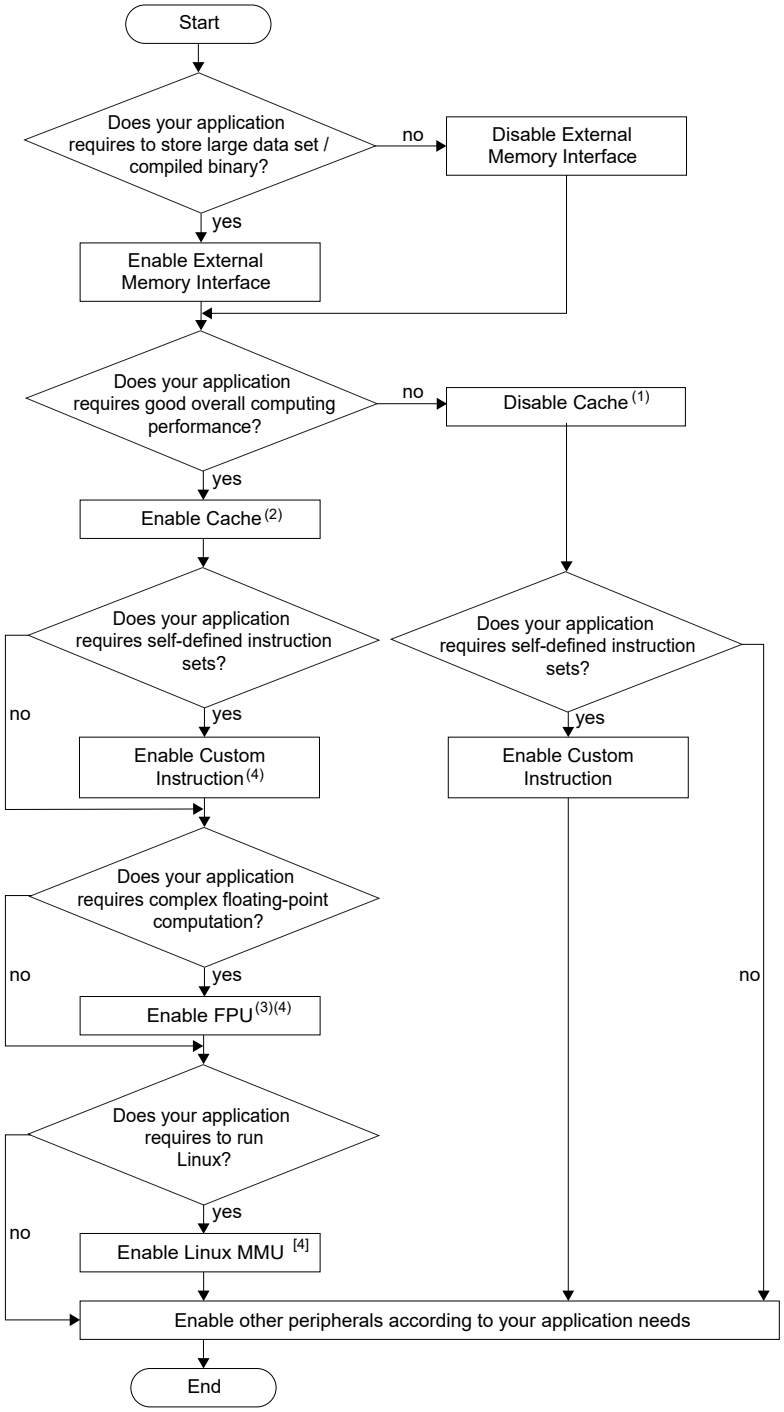
Table 13: Sapphire RV64 Watchdog Timer Parameters

Parameter	Option	Description
Watchdog Timer	On, Off	On: Instantiate the interface. Off: Do not use the interface.
Watchdog Timer Prescaler Width	8, 16, 24, 32	Choose the prescaler bit width. Default: 24
Watchdog Timer Counter Width	16, 24, 32	Choose the counter bit width. Default: 16

SoC Configuration Guideline

Sapphire RV64 SoC is highly adaptive to different use cases. It is configurable to get the best balance between performance and resources. The following flow chart is a simple guideline to help you choose the configuration that suits your needs.

Figure 4: Sapphire RV64 Soc Configuration Guideline



Notes :

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

Modify the Bootloader

When you generate the Sapphire RV64 SoC, the IP Manager creates a pre-built bootloader **.bin** to target the on-chip RAM size you selected. If you assigned the peripheral addresses manually, you need to create a custom bootloader according to the following instructions.

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

Modify the Bootloader Software with External Memory Enabled

First, you need to modify the bootloader code:

1. Open the **bootloaderConfig.h** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/EfxSapphireSocRV64/software/standalone/bootloader** directory.
2. Change the `#define USER_SOFTWARE_SIZE` parameter for the new on-chip RAM size and save.

In Efinity RISC-V Embedded Software IDE, import the bootloader project and build the project to generate a new **bootloader.hex** file.

Modify the Bootloader Software without External Memory Enabled

First, you need to modify the bootloader linker script:

1. Open the **bootloader.ld** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/EfxSapphire RV64Soc/linker** directory.
2. Replace the `MEMORY` section with the following code. The <*bootloader_address*> should be 0x0001000000 + (<*memory size*>-3072), where <*memory size*> is your SoC's on-chip RAM size. This will reserve 3,072 bytes for bootloader application.

```
MEMORY
{
    start (wxai!r) : ORIGIN = 0x0001000000, LENGTH = 512
    ram   (wxai!r) : ORIGIN = <bootloader_address>, LENGTH = 3,072
}
```

Second you need to modify the bootloader code:

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

In Efinity RISC-V Embedded Software IDE, import the bootloader project and build the project to generate a new **bootloader.hex** file.

Regenerate the Memory Initialization Files

Next, you need to regenerate the memory initialization files using the **binGen.py** helper script.

1. You can find this script in the *<project>*/**embedded_sw/***<SoC module>*/**tool directory**. Use the following command to find the script:

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

   where: *<RAM size>* is the on-chip RAM size you want to use.
2. This command generates the new memory initialization files. Replace the existing files by copying these files into the *<SoC module>* IP directory.
3. Compile your design.

Modify the Bootloader Software to Enable Multi-Data Lines

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

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

Table 14: Multi-Data Lines Interface with FPGAs and Flash Devices

Development Kit	Flash Device	Number of Data Lines Connected
T8BGA81	W25Q80DLSNIG	2
T20BGA256	W25Q32JVSSIQ	2
T120BGA324	W25Q128JVSIQ	4
T120BGA576	W25Q128JVSIQ	4
Xyloni	W25Q128JVSIM	2
Ti60F225	W25Q64JWSSIQ	2
Ti180J484/ Ti180M484/ Tz170J484	MX25U25645GZ4I00	4

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

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

```
#define SPI    (spi_hwreg_t* )SYSTEM_SPI_0_IO_CTRL
#define USER_SOFTWARE_MEMORY    0x0800000000
#define USER_SOFTWARE_FLASH      0x00600000
#define USER_SOFTWARE_SIZE 0x01F000 //124KB

#define SINGLE_SPI 1 //define DUAL_SPI for dual data SPI or QUAD_SPI for quad data SPI
spiFlash_instance_t Flash = {
  .cs = 0,
  .inst = &(spi_instance_t){
    .hwreg      = SPI,
    .cpol       = LOW,
    .cpha       = DATA_SAMPLED_RISE_EDGE,
    .mode       = FULL_DUPLEX_SINGLE_LINE
},
};

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

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

Updating Bootloader with Efinity BRAM Initial Content Updater

The Efinity BRAM Initial Content Updater provides a convenient way to modify the default firmware (either the bootloader or other applications) within the Sapphire RV64 SoC on-chip RAM. This process enables you to update the on-chip RAM initial content without recompiling the entire project.

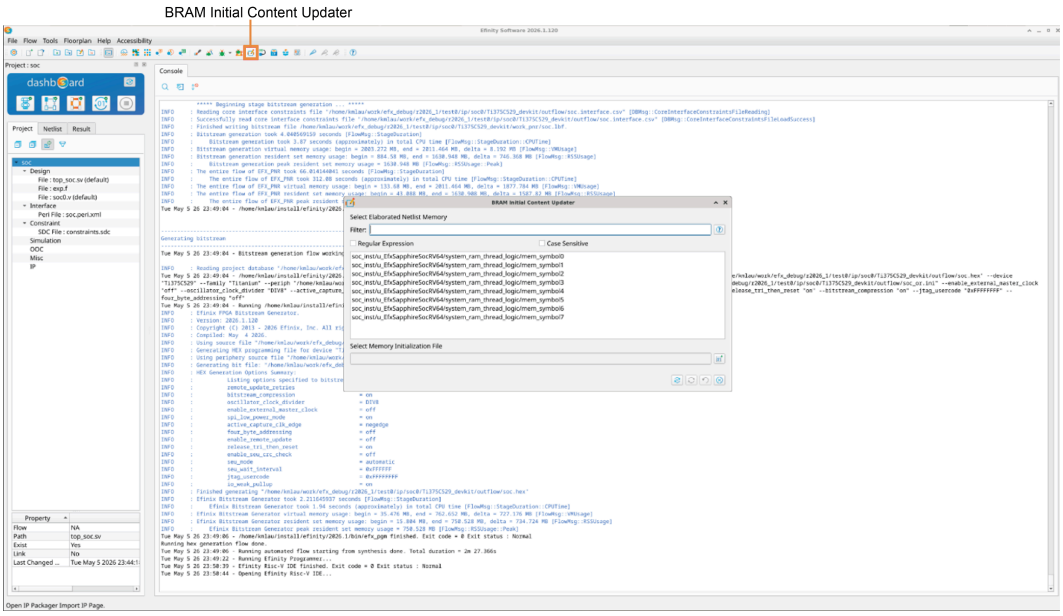
To update the on-chip RAM initial content, follow these steps:

1. *Compile and locate the* ***.hex*** *file:* Compile your new application in Efinity RISC-V Embedded Software IDE and locate the corresponding **.hex** file that contains the compiled code.
2. *Generate the Sapphire RV64 SoC with the new application:* By using the Sapphire RV64 SoC IP Configurator, update the default on-chip RAM application with your new application compiled in the previous step. You may refer to the **Modify the Bootloader Software with External Memory Enabled** on page 18 on how to use the **Custom On-Chip RAM Application** in the Sapphire RV64 SoC IP Configurator. and you are now ready for the updating process.

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

3. *Locate the new binaries for your application:* After the Sapphire RV64 SoC is generated with your application, locate the new binaries, **EfxSapphireSocRV64.v_toplevel_system_ram_thread_logic_mem_symbol<n>.bin** where <n> is the range from 0 to 3 (up to 7 if FPU extension is enabled).
4. *Open the BRAM Initial Content Updater:* Click on the **BRAM Initial Content Updater** tab to launch the **BRAM Initial Content Updater**.

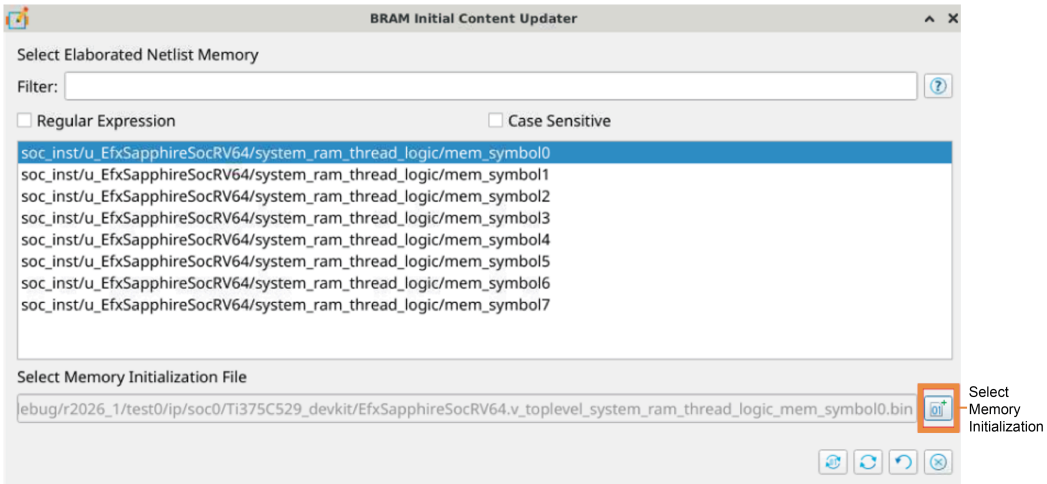
Figure 5: Open the BRAM Initial Content Updater



5. *Select Memory Initialization File:* In the **BRAM Initial Content Updater** window, locate the Sapphire RV64 SoC BRAM that you would like to update and click on the ***_symbol0**. In the **Select Memory Initialization File** section, click the **Select Memory Initialization** tab. Browse to the updated

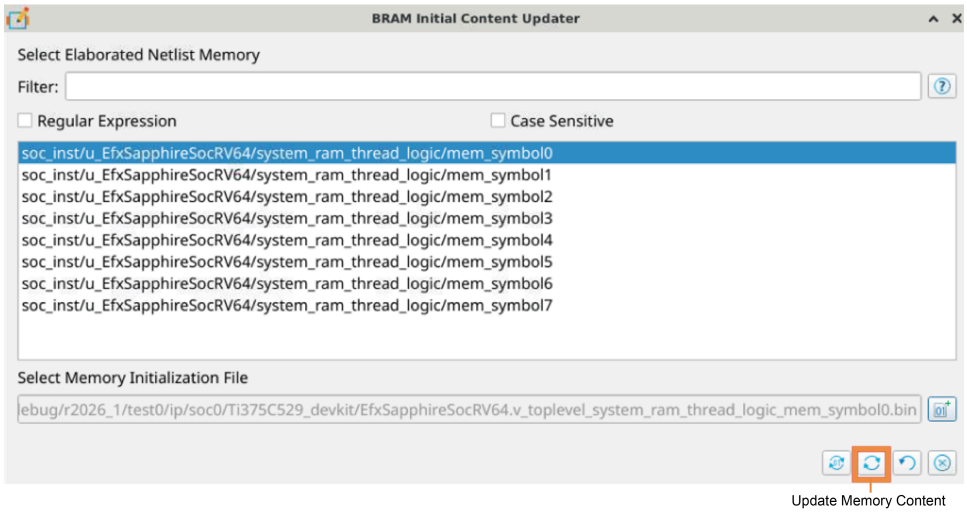
EfxSapphireSoc.v_toplevel_system_ramA_logic_ram_symbol<n>.bin and click **Open**.

Figure 6: Select Memory Initialization File



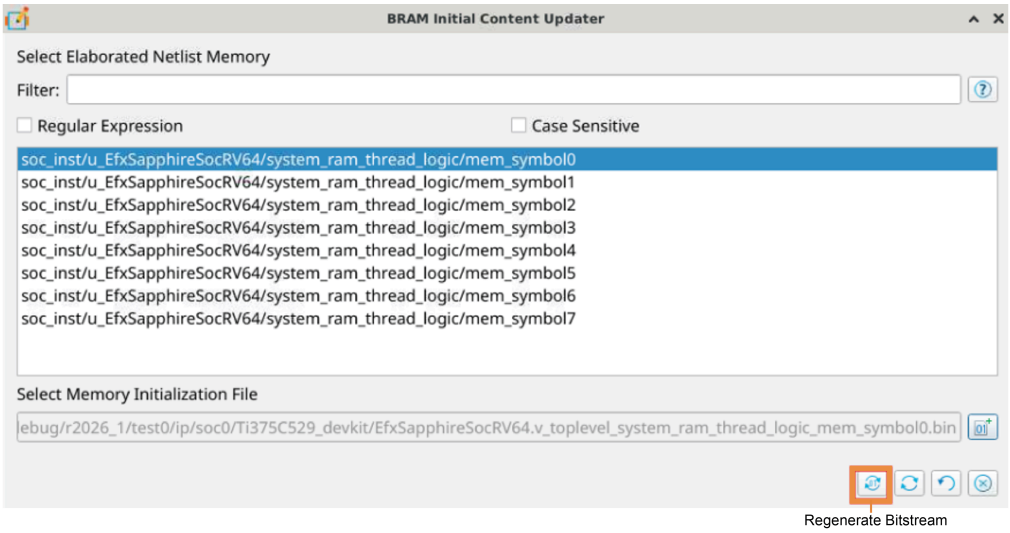
6. *Update the BRAM:* Click on the **Update Memory Content** tab to update the **symbol0** BRAM with the new application **symbol0** binary.

Figure 7: Update the BRAM with New Application



7. *Update the remaining BRAM:* Repeat steps 5 and 6 for all the available symbol files. Update the BRAM with the corresponding binary. For example, update ***_ram_symbol2** BRAM with **EfxSapphireSoc.v_toplevel_system_ramA_logic_ram_symbol2.bin** binary file.
8. *Generate the new bitstream:* Click on the **Regenerate Bitstream** icon to regenerate the bitstream. The newly generated bitstreams are located in the outflow folder.

Figure 8: Generate the New Bitstream



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

Chapter 3

Program the Board with the Sapphire RV64 RTL Design

Contents:

- About the Example Design
- Installing USB Drivers
- Program the Development Board

Before working with software code, Efinix recommends that you program your board with an RTL design that instantiates the Sapphire RV64 SoC. When you generate the Sapphire RV64 SoC with the IP Manager, you can optionally generate an example Efinity® project and bitstream file to get you started quickly.

About the Example Design

This example targets Trion and Titanium development boards:

- *Titanium Ti375 C529 Development Board*—The RTL design files are in the **Ti375C529_devkit** directory.

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

The example design automatically connects UART0, SPI0, I2C0, GPIO0, the JTAG pins, and the PLL source clock pins to top-level ports, and assigns I/O pins to these peripherals once enabled. If you add more of these peripherals, you need to manually connect and create I/O assignments.

The example design uses PLL settings to look for the best effort multiplier and divider values.

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

This example design uses a simple dual-port RAM module to write to and read from the development board's memory module using the AXI interface. For the Titanium Ti375 C529 Development Board, the design uses the board's LPDDR4/LPDDR4x DRAM module.

The example design software blinks an LED and displays messages on a UART terminal.

The Sapphire RV64 SoC is configured for:

- 200 MHz system frequency, 50 MHz peripheral frequency
- All ISAs are enabled except Zicbom
- All microarchitecture features are enabled
- External memory interface is enabled with a width of 256 and a size of 1.0 GB
- L1 data cache and instruction cache set to one way with a cache size of 4 KB
- L2 cache set to 64 KB
- 4 KB on-chip RAM size
- UART 0 is enabled
- SPI 0 is enabled
- GPIO 0 is enabled
- APB3 0 is enabled
- AXI4 Slave 0 and 1 are enabled
- AXI Master A and B are enabled

Figure 9: Example Design Block Diagram

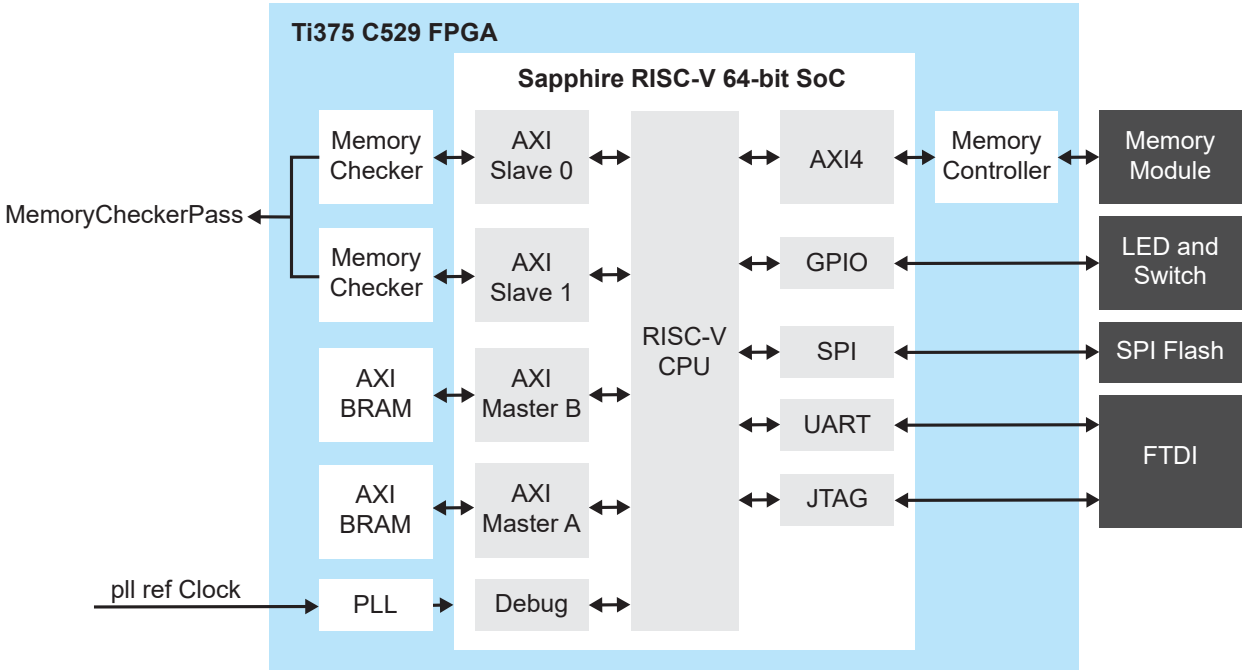


Table 15: Example Design Implementation

FPGA	Logic + Adders	Flipflops	Multipliers or DSP Blocks	Memory Blocks	f_{MAX} (MHz)	Language	Efinity Version
Ti375C529	42,457	35,061	17	296	204	Verilog HDL	2026.1

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

Installing USB Drivers

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

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

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

Installing Drivers on Windows

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

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

Board	Interface to Install Driver
Trion® T120 BGA324 Development Board	Install drivers for all interfaces (0 and 1).
Titanium Ti60 F225 Development Board	Install drivers for interfaces 0 and 1 only. Windows automatically installs a driver for interfaces 2 and 3 when you connect the board to your computer.
Titanium Ti180 J484 Development Board, Topaz Tz170 J484 Development Board	Install driver for interface 1 only.

To install the driver:

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

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

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

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

Installing Drivers on Linux

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

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

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

Note: If your board was connected to your computer before you executed these commands, you need to disconnect it, then reconnect it.

Program the Development Board

When you generate the Sapphire RV64 SoC in the IP Manager, you can optionally generate an example design targeting an Efinix development board. Example designs include a bitstream file, **soc.hex**, so you can get started quickly without having to compile the design.

Table 16: Available Example Designs

Board	Location
Titanium Ti60 F225 Development Board	Ti60F225_devkit
Titanium Ti180 J484 Development Board	Ti180J484_devkit
Trion® T120 BGA324 Development Board	T120F324_devkit
Topaz Tz170 J484 Development Board	Tz170J484_devkit

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

Connect the board to your computer using a USB cable.

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

Chapter 4

Simulate

1. Open a terminal (Linux) or Command Prompt (Windows).
2. Change to your SoC module's **Testbench** directory.
3. Set up the Efinity environment:
   - Linux: `source /<path to Efinity>/bin/setup.sh`
   - Windows: `source c:\<path to Efinity>\bin\setup.bat`
4. To simulate with the default software application, use the command `python3 run.py`. A successful simulation returns the following messages:

```
#  [EFX_INFO]: EfxSapphireSocRV64 is out of reset state                #
 [EFX_INFO]: Test Pass
```

5. (Optional) If you want to simulate with your own software application (you need to develop your own sequence for your application):
   - Copy your application binary into the **Testbench** directory.
   - Simulate with the command
     `python3 run.py -b <path to app><application>.bin`

Argument	Description
-h	Help menu.
-b	Convert the user application into memory initialization files for external memory.
-l	Convert the user application into memory initialization files for the SoC on-chip RAM.

Note: By default, the memory initialization files contain a bootloader application that fetches the data size corresponding to the on-chip RAM size you selected in the IP Manager. If you would like to build a custom bootloader, you can refer to **Modify the Bootloader** on page 18.

Chapter 5

Boot Sequence

Contents:

- Boot Sequence: Case A
- Boot Sequence: Case B
- Boot Sequence: Case C
- Booting Multiple Cores

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

Figure 10: Boot Sequence Flow Chart

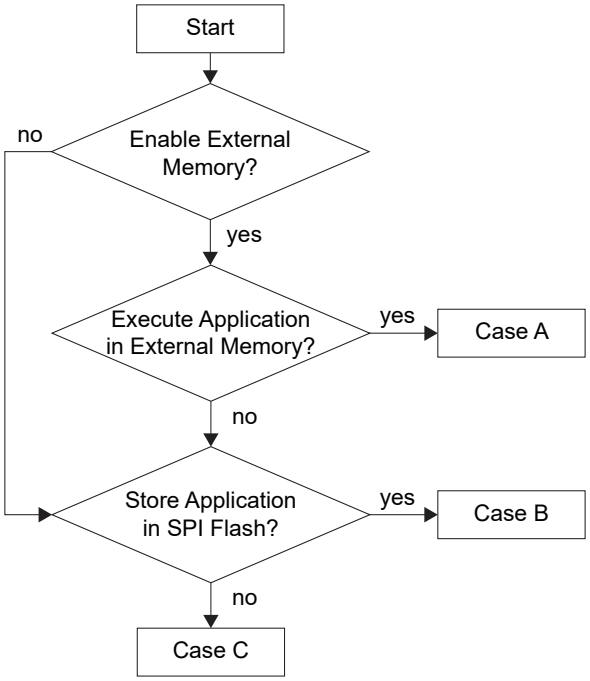


Table 17: User Application

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

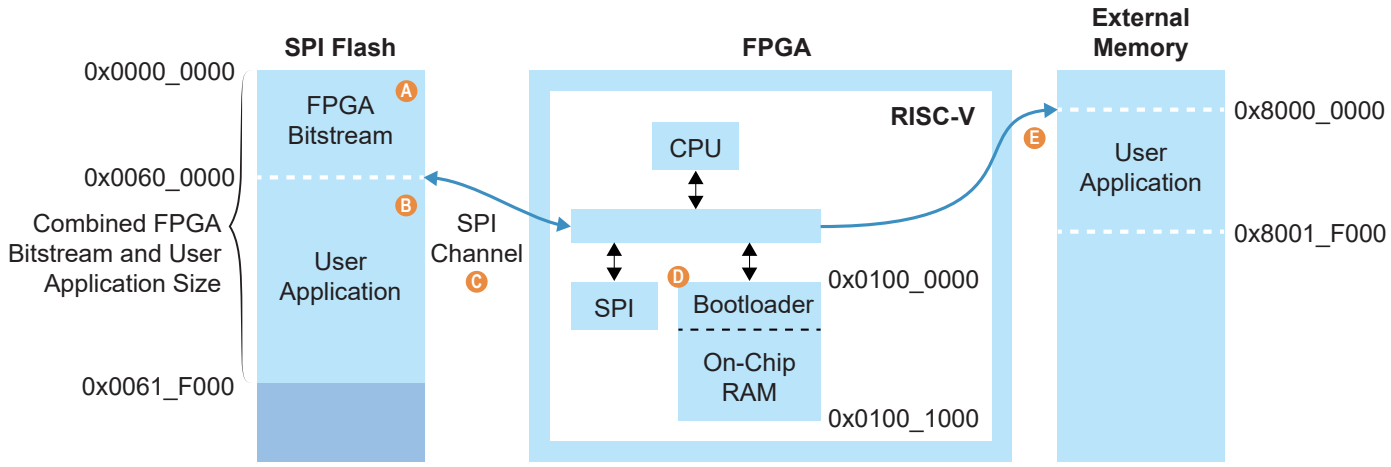
The following sections describe these cases in more detail.

The Sapphire RV64 SoC supports multiple cores; Booting Multiple Cores on page 34 describes the programming sequence.

Boot Sequence: Case A

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

Figure 11: Boot Sequence Diagram



Notes:

A. The bitstream has a default start address of 0x0000_0000 in the Efinity Programmer.
B. The application has a start address of 0x0060_0000.
C. The bootloader reads the SPI flash data from 0x0060_0000.
D. The CPU starts at 0x0100_0000. The default On-Chip RAM size is 4 KB.
E. The bootloader copies the SPI flash data to external memory address 0x8000_0000 and redirects the address to 0x8000_0000 for execution.

The system starts from the PC's 0x0100_0000, which is the starting address of the on-chip RAM. The bootloader, which reads a larger user application from the SPI flash, is embedded by default.

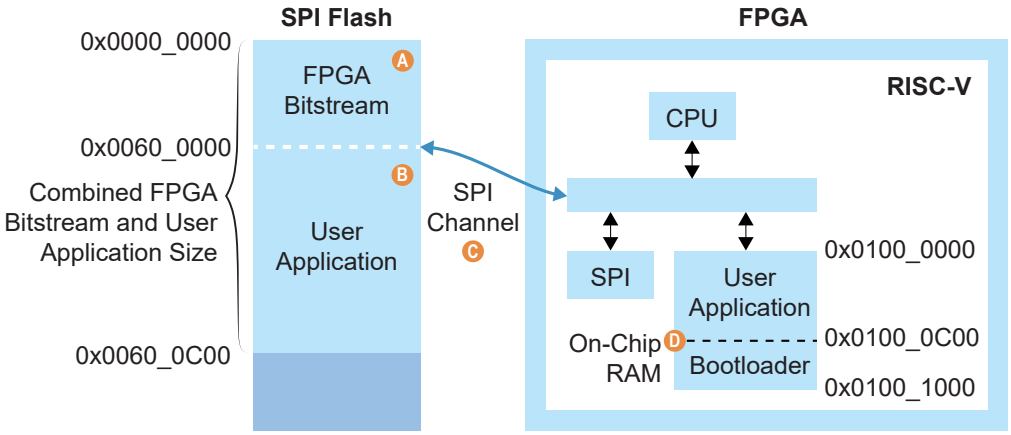
1. The PC starts at the system address 0x0100_0000 of the on-chip RAM.
2. The bootloader starts reading the SPI Flash address 0x60_0000 for the user application.
3. The bootloader writes the user application to external memory starting from system address 0x8000_0000.
4. The bootloader finishes reading the user application from the SPI flash.
5. The PC jumps to system address 0x8000_0000 and starts to execute the user application.
6. All accesses remain in the external memory space, which is malloc() by default (unless you specify the on-chip RAM space in the software code)

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

Boot Sequence: Case B

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

Figure 12: Boot Sequence Diagram



Notes:

A. The bitstream has a default start address of 0x0000_0000 in the Efinity Programmer.
B. The application has a start address of 0x0060_0000.
C. The bootloader reads the SPI flash data from 0x0060_0000.
D. The bootloader copies the SPI flash data to the On-Chip RAM 0x0100_0000 and redirects the address to 0x0100_0C00 for execution.

- The last 1 KB of On-Chip RAM is reserved for the bootloader.
- The user application should not exceed the size 0xC00 which breaks the bootloader that is stored at 0x0100_0C00.

The boot sequence is:

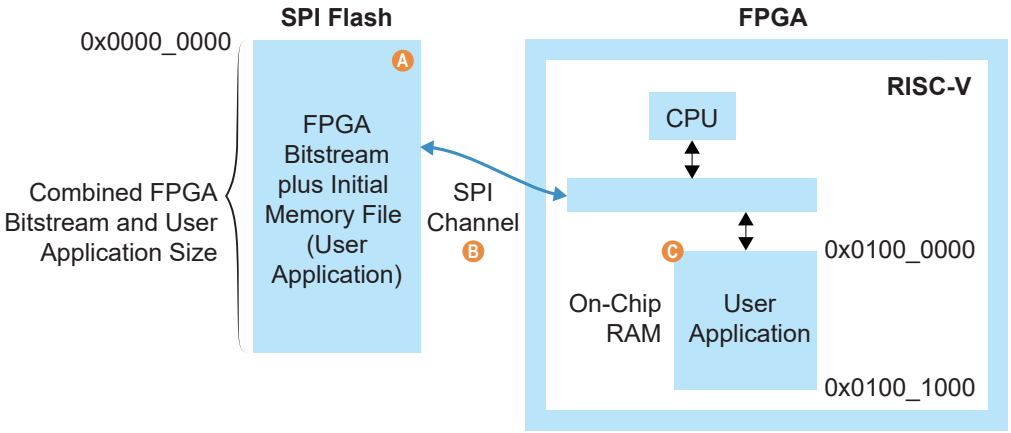
1. The PC starts at the system address 0x0100_0000 of the on-chip RAM and the PC jumps to 0x0100_0C00 for bootloader execution.
2. The bootloader starts reading the SPI Flash address 0x0060_0000.
3. The bootloader writes the user application to On-Chip RAM starting from system address 0x0100_0000.
4. The bootloader finishes reading the user application from the SPI flash.
5. The PC jumps to system address 0x0100_0000 and starts to execute the user application.

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

Boot Sequence: Case C

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

Figure 13: Boot Sequence Diagram



Notes:

A. The bitstream has a default start address of 0x0000_0000 in the Efinity Programmer.
B. The application initial memory file is synthesized with the FPGA bitstream with address 0x0100_0000 for the RISC-V application.
C. The CPU starts at 0x0100_0000.

The boot sequence is:

1. The system starts from the PC's 0x0100_0000, which is the starting address of the On-Chip RAM.
2. The user application is already compiled with the bitstream. It starts executing automatically from the FPGA's BRAM.

Booting Multiple Cores

If you configure multiple cores, the Sapphire RV64 SoC has two or more identical processors that share a common main memory and the same set of hardware I/Os. The processors can execute programs simultaneously; one processor can access the processed data or result from other processors because they are connected in a shared backplane.

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

Figure 14: Boot Sequence for Multiple Cores

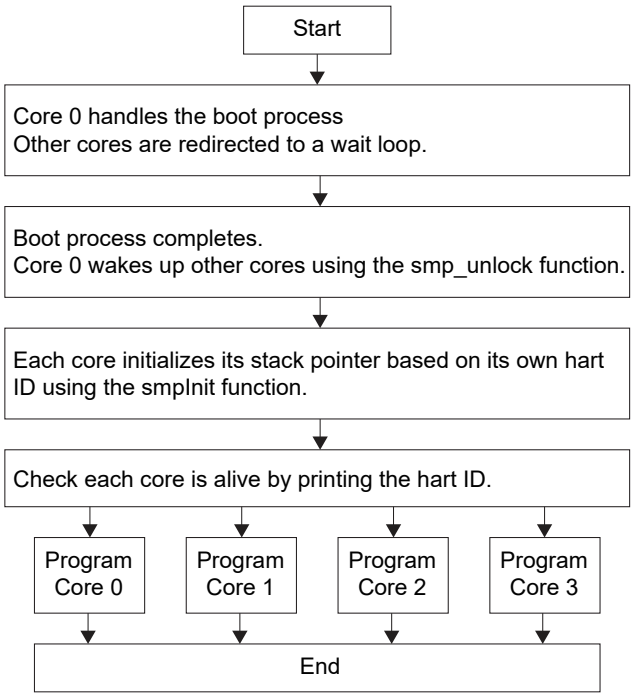


Table 18: SMP Helper Functions

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

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

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

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

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

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

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

Chapter 6

Create Your Own RTL Design

Contents:

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

After you have explored the Sapphire RV64 SoC using the included example Efinity® project, you can use these tips to modify the design for your own use.

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

Target another FPGA

To change the design to target a different FPGA:

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

Target another Efinix Board

The Sapphire RV64 SoC BSP includes FTDI configuration files that specify the FTDI device VID and PID and board description for Efinix development boards.

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

Table 19: Provided FTDI Configuration Files

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

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

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

1. Open the Efinity® Programmer.
2. Click the **Refresh USB Targets** button to display the board name in the **USB Target** drop-down list.
3. Make note of the board name.
4. In a text editor, open the **ftdi.cfg** or **ftdi_ti.cfg** file in the **embedded_sw/**<*SoC module*>**/bsp/efinix/EfxSapphireSocRV64/openocd** directory.
5. Change the `ftdi_device_desc` setting to match the board name. For example, use this code to change the name from Trion T120F576 Development Board to Trion T120F324 Development Board:

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

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

Target Your Own Board

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

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

The Sapphire RV64 SoC does not support GPIO based debug module, also known as soft tap in RV32. If you prefer to do debugging simultaneously with the FPGA debugger, you should enable **Co-debug** mode in the **JTAG Configuration** section under the **Debug** tab. If you use an external JTAG cable to connect your board to your computer, you can use these configuration files; **ftdi.cfg**, **ftdi_ti.cfg**, or **ftdi_tz.cfg**.

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

Updating OpenOCD Configuration for External FTDI Cable

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

Table 20: OpenOCD Confuguration File Setting for External FTDI Cable

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

Launching OpenOCD for Your Own Board

The standard launch scripts only support the following:

- ***_ti**: Standard Titanium FTDI + SoC hard JTAG Port
- ***_tz**: Standard Topaz FTDI + SoC hard JTAG Port
- ***_trion**: Standard Trion FTDI + SoC hard JTAG Port

To use an external FTDI Cable (i.e., C232HM-DDHSL-0 Programming Cable) with the SoC hard JTAG Port (using device TAP Controller), you are required to modify the debug configuration to correctly target the FPGA device.

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

1. Select the preferred external JTAG Cable in the IP Manager when configuring the Sapphire RV64 SoC.
2. Import your desired project (i.e., gpioDemo) in the Efinity RISC-V Embedded Software IDE.
3. Modify the debug configuration file as mentioned in **Updating OpenOCD Configuration for External FTDI Cable** on page 39.
4. Click **Apply** and **Debug** to launch your application.

Using another JTAG Cable or Module

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

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

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

You can also write your own configuration file if desired.

Follow these instructions when debugging with your own board:

1. Connect your JTAG cable to the board and to your computer.
2. Copy the OpenOCD configuration file for your cable to the **bsp/efinix/EfxSapphireSocRV64/openocd** directory.
3. Follow the instructions for debugging, except target your configuration file instead of the **ftdi.cfg** (Trion), **ftdi_ti.cfg** (Titanium), or **ftdi_tz.cfg** (Topaz) file.

```
-f <path>/bsp/efinix/EfxSapphireSocRV64/openocd/<my cable>.cfg
```

Using an FTDI Chip on your Board

When you configure the Sapphire RV64 SoC in the IP Configuration wizard, choose **Target OpenOCD > Custom**. Then, specify your board name. When you generate the SoC, the **ftdi.cfg**, **ftdi_ti.cfg**, or **ftdi_tz.cfg** file is populated with your board name. Edit the file for your board's VID and PID.

Create a Custom AXI4 Master Peripheral

When you generate an example design for the Sapphire RV64 SoC, the IP Manager creates an example AXI4 peripheral and software code that you can use as a template to create your own peripheral. This example uses a simple dual-port RAM design to write to and read from the CPU through the AXI4 interface.

- Refer to the `axi4_slave` module in **design_modules.v** in the **Ti375C529_devkit**, **Ti60F225_devkit**, or **Ti180J484_devkit** directory for the RTL design.
- Refer to **main.c** in the **embedded_sw/<SoC module>/software/standalone/axiBDemo/src** directory for the C code.

Create a Custom APB3 Peripheral

When you generate an example design for the Sapphire RV64 SoC, the IP Manager creates an APB3 peripheral and software code that you can use as a template to create your own peripheral. This simple example shows how to implement an APB3 slave wrapper.

- Refer to the `apb3_slave` module in **design_modules.v** in the **Ti375C529_devkit**, **Ti60F225_devkit**, or **Ti180J484_devkit** directory for the RTL design.
- Refer to **main.c** in the **embedded_sw/<SoC module>/software/standalone/apb3Demo/src** directory for the C code.

Use another DDR DRAM Module (Trion Only)

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

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

Use the I²C Interface for DDR Calibration (Trion only)

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

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

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

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

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

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

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

Remove Unused Peripherals from the RTL Design

The Sapphire RV64 SoC includes a variety of peripherals. If you do not want to use a peripheral, simply remove the signal name from within the parentheses () in the SapphireSoc_inst definition in the top-level Verilog HDL file. For example, the SoC instantiation has these signals:

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

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

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

Chapter 7

Create Your Own Software

Contents:

- Deploying an Application Binary
- About the Board Specific Package
- List of BSP Files

Now that you have explored the methodology for designing with the Sapphire RV64 SoC, you can develop your own software applications.

Deploying an Application Binary

During normal operation, your user binary application file (**.bin**) is stored in a SPI flash device. When the FPGA powers up, the Sapphire RV64 SoC copies your binary file from the SPI flash device to the DDR DRAM module, and then begins execution.

For debugging, you can load the user binary (**.elf**) directly into the Sapphire RV64 SoC using the OpenOCD Debugger. After loading, the binary executes immediately.

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

Boot from a Flash Device

When the FPGA boots up, the Sapphire RV64 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 0x0060_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 0x0060_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 RV64 SoC jumps to logical address 0x0000_1000 to execute the user binary.

Note: Refer to Boot Sequence on page 31 for other possible boot scenarios.

Boot from the OpenOCD Debugger

To boot from the OpenOCD debugger:

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

Note: Refer to Debug with the OpenOCD Debugger in Efinity RISC-V Embedded Software IDE User Guide for complete instructions.

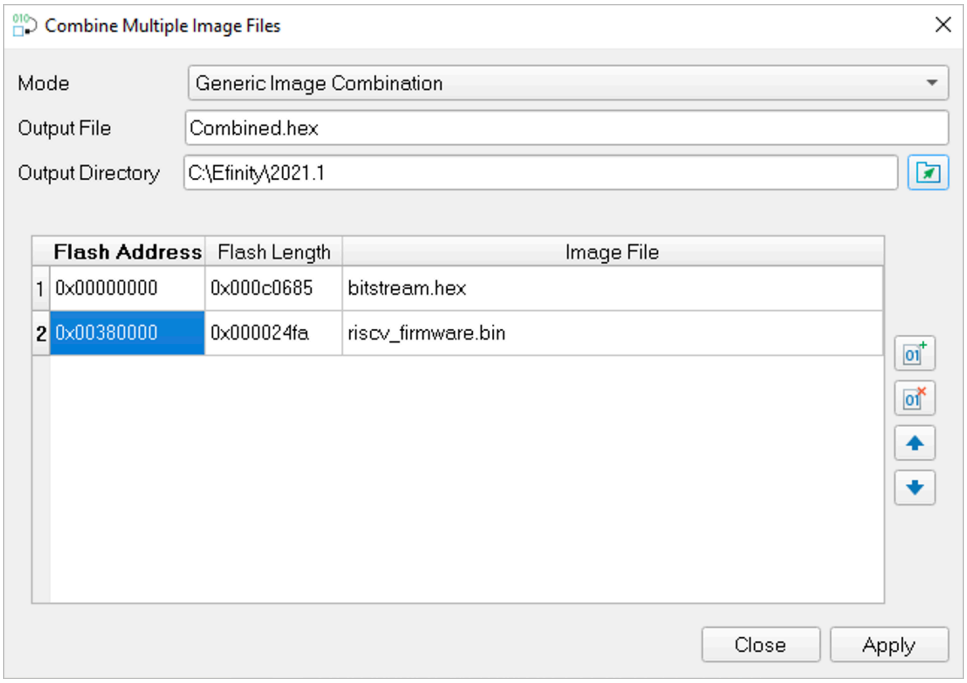
Copy a User Binary to Flash (Efinity Programmer)

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 15: Combining a Bitstream and RISC-V Application Binary



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

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).

About the Board Specific Package

The board specific package (BSP) defines the address map and aligns with the Sapphire RV64 SoC hardware address map. The BSP files are located in the **bsp/efinix/EfxSapphireSocRV64** subdirectory.

Table 21: BSP Files

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

List of BSP Files

Table 22: List of Files

Directory	Files/Folder	Description
tool	**binGen.py**	A python script to convert RISC-V firmware binary into FPGA memory initialization files.
software	**freeRTOS**	FreeRTOS example software folder.
–	**freeRTOS/common/**	FreeRTOS configuration files and HAL definition.
	freeRTOS/freertosDemo/	FreeRTOS example software.
	standalone	Baremetal example software and driver folder.
	standalone/ • bootloader • coremark • customInstruction • dhrystone • fpu • gpio • i2c • i2ctrl • smp • spiFlash • timer • uart • vexriscv	Baremetal example software.

Revision History

Table 23: Revision History

Date	Version	Description
May 2026	1.0	Initial release.