AN 031: Using 10/100 Mbps TSE MAC with Ruby SoC

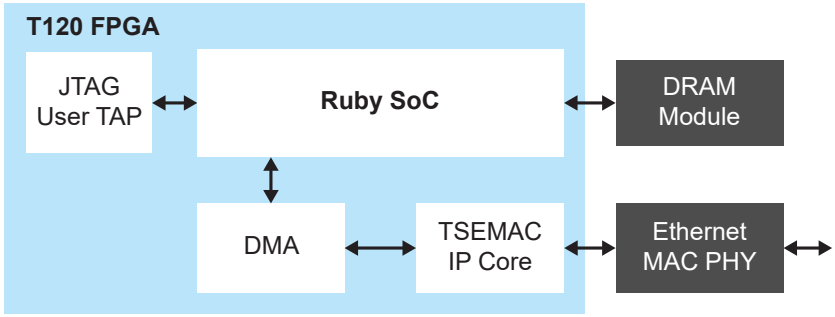
AN031-v1.2
November 2022
www.efinixinc.com

Contents

Introduction

The Ruby SoC with Triple-Speed Ethernet (TSEMAC) example design illustrates how to use a RISC-V processor to control the Triple Speed Ethernet MAC (TSEMAC) IP core and transfer data through DMA (direct memory access). The design supports 10 and 100 Mbps Ethernet speed. The FreeRTOS software framework provides a programming stack to perform various Ethernet functions. The RTL design targets the Trion® T120 BGA324 Development Board. The FreeRTOS software application lets you establish an IPerf3 server to test the overall bandwidth.

Figure 1: Ruby SoC with Triple-Speed Ethernet Example Design Block Diagram



FPGA Support

The example design supports Trion® T120 FPGAs only.

Resource Utilization and Performance

FPGA	Logic Utilization (LUTs)	Memory Blocks	f_{MAX} (MHz)	Language	Efinity Version
T120 F324 I4	19,925	135	TSEMAC: 103.67 SoC: 56.89	Verilog HDL	2020.2

Required Hardware

The example design uses the following hardware from the Trion® T120 BGA324 Development Kit:

- Trion® T120 BGA324 Development Board
- Micro-USB cable
- 12 V power adapter

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

- USB-to-UART module
- 3 jumper wires
- Ethernet cable
- Computer with Ethernet port and Efinity® software installed

Required Software

The example design uses the following software:

- RISC-V SDK for Windows or Ubuntu
- Efinity® software version 2020.1
- FreeRTOS version 10.4.1 (https://github.com/FreeRTOS/FreeRTOS/releases/tag/V10.4.1)[1]
- iPerf3 network speed test tool (https://iperf.fr)
- EchoTool echo client and server (https://github.com/PavelBansky/EchoTool)

Note: Refer to the Ruby RISC-V SoC Hardware and Software User Guide for instructions on installing the RISC-V SDK.

Download and Install the Files

The Ruby SoC with Triple-Speed Ethernet example design includes a hardware project and software example code.

1. Download the example design file, **t120f324-riscv-tsemac-v<version>.zip**, from the Support Center.
2. Unzip the file into a folder at the root level (Windows) or your project directory (Linux).

   Note: For Windows, FreeRTOS has folders and files with long names. Therefore, Efinix recommends that you unzip the files into a folder at the root of the filesystem. Otherwise, you may receive build failures because the filenames are too long for the filesysmte to handle.

3. Copy the FreeRTOS v10.4.1 files into the **embedded_sw/RubyCore/software** directory

The resulting directory structure should be:

- **c:\t120f324-riscv-tsemac**
  - **constraints.sdc**—Software constraints file.
  - **embedded_sw**
    - **RubyCore**
      - **software**
        - **FreeRTOSv10.4.1**
        - **standalone**
  - **ip**—Contains the generated Ruby SoC RTL files.
  - **rubysoc.v_toplevel_system_cpu_logic_cpu_RegFilePlugin_regFile.bin**
  - **rubysoc.v_toplevel_system_ramA_logic_ram_symbol0.bin**
  - **rubysoc.v_toplevel_system_ramA_logic_ram_symbol1.bin**
  - **rubysoc.v_toplevel_system_ramA_logic_ram_symbol2.bin**
  - **rubysoc.v_toplevel_system_ramA_logic_ram_symbol3.bin**
  - **source**—RTL hardware files.
  - **t120f324-riscv-tsemac.bit**—Bitstream file for JTAG programming.
  - **t120f324-riscv-tsemac.hex**—Bitstrean file for SPI or JTAG programming.
  - **t120f324-riscv-tsemac.peri.xml**—Interface Designer file.

[1] Get v10.4.1, not the latest release.

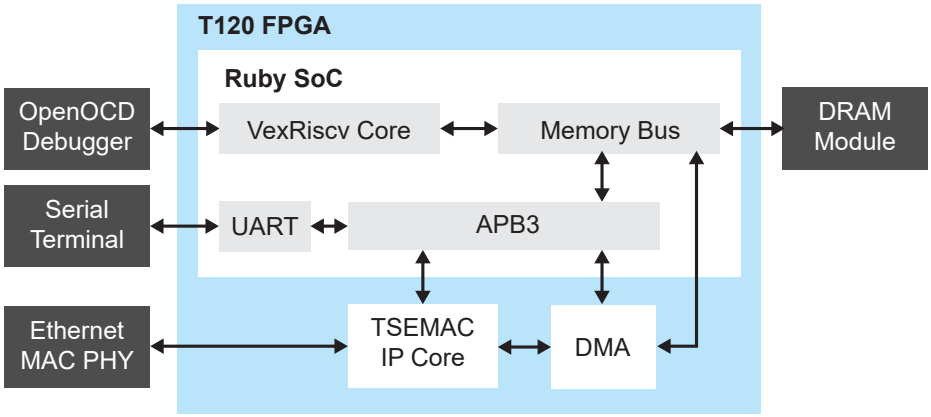
— **t120f324-riscv-tsemac.xml**—Project file.
— **top_rubySoc.v**—Top-level RTL file.

Functional Description

The Ruby SoC with Triple-Speed Ethernet example design combines the Ruby SoC with a DMA block and TSEMAC IP core. The DMA block has a memory-mapped interface that connects to the SoC's memory bus. A streaming interface connects the DMA block to the TSEMAC IP core, which transfers bulk data from/to the TSEMAC without needing to access the RISC-V processor. This arrangement gives the processor more bandwidth to complete other operations. The TSEMAC IP core transmits or receives Ethernet packets to/from the host, and ensures the data transmission over Ethernet meets the media access rules specified in the 802.3-2008 IEEE standard. The RISC-V processor controls the DMA block and TSEMAC IP core through the APB3 bus.

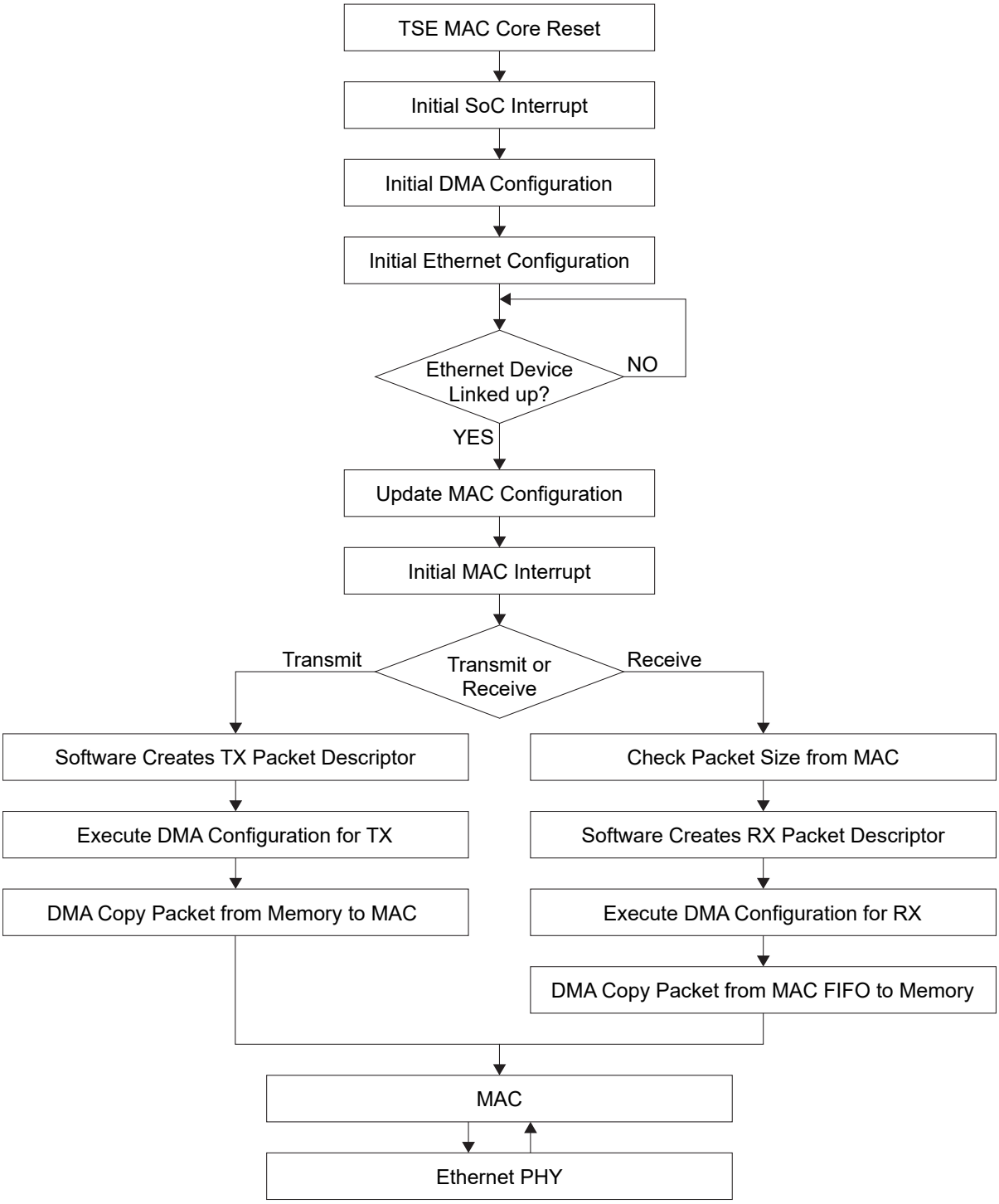
With the UART peripheral, you can use a terminal to display message output. Messages print when the TSEMAC successfully connects to a host or there is packet activity from the host to the FPGA that is requested by a software application, or vice-versa.

Figure 2: Ruby SoC Peripherals



Note: The instructions in this example design assume that you are familiar with the Ruby SoC and the RISC-V SDK.

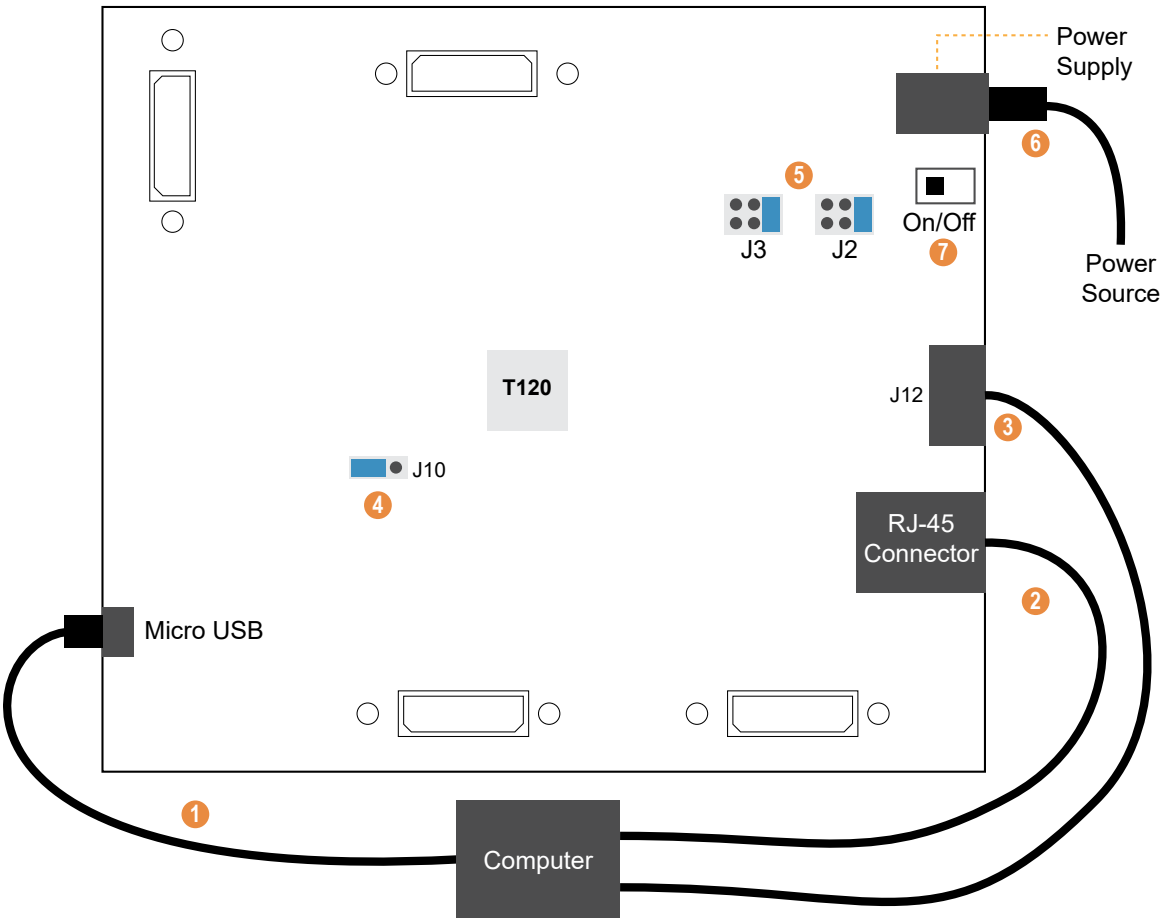
Figure 3: TSEMAC Example Design Flowchart



Set Up the Hardware

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

Figure 4: Hardware Setup



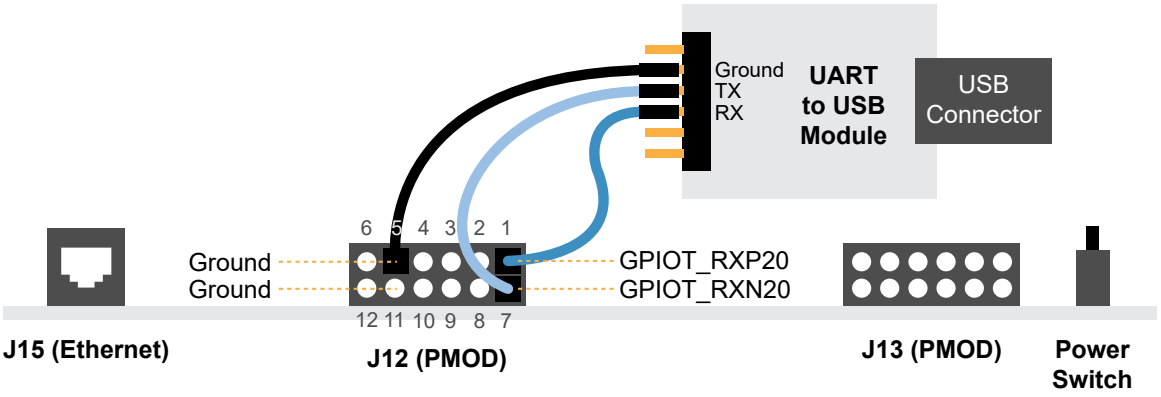
Important: Make sure that the Trion® T120 BGA324 Development Board is turned off before connecting any cables.

1. Connect a USB cable to the board and to your computer.
2. Connect an Ethernet cable to the RJ-45 jack on the board and to your computer.
3. Connect a USB-to-UART module to the J12 header. See Set Up a USB-to-UART Module (Trion) on page 8 for more details.
4. On header J10, connect pins 2 - 3 with a jumper (default) to use the on-board oscillator.
5. Leave the jumper on J2 at the defaults (connecting pins 1 - 2). On J3, connect pins 1 - 2.
6. Connect the 12 V power supply to the board connector and to a power source.
7. Turn on the board using the power switch.

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

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

Figure 5: Connect the UART Module to PMOD Connector J12



1. Connect the UART module to the PMOD port J12
   - *RX*—GPIOT_RXP20, which is pin 1 on PMOD J12
   - *TX*—GPIOT_RXN20, which is pin 7 on PMOD J12
   - *Ground*—Use ground pin 5 or 11 on PMOD J12.
2. Plug the UART module into a USB port on your computer. The driver should install automatically if needed.

Finding the COM Port (Windows)

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

Finding the COM Port (Linux)

In a terminal, type the command:

```
dmesg | grep ttyUSB
```

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

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

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

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

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

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

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

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**.

Set Your Computer's IP Address

Make these settings to your computer's Ethernet network adapter:

1. Set the IP address for your computer to 192.168.0.222.
2. Set the default gateway to 192.168.0.1.
3. Set the Subnet mask to 255.255.255.0.
4. Set the DNS server to 8.8.8.8.

By default, the example application software has an FPGA IP address of 192.168.0.55, a gateway of 192.168.0.1, and expects your computer address to be 192.168.0.222. You can change these defaults (as well as the default MAC addresss) in the **FreeRTOSIPConfig.h** file in the **src** directory. If you change these defaults, you need to re-build the project in Eclipse and download the new **.elf** file to the board.

Program the Trion® T120 BGA324 Development Board

The Trion® T120 BGA324 Development Board ships pre-loaded with an example design that sends color bars to an HDMI monitor. To use the Ruby SoC with Frame Buffer example design, you must program the design into the board.

1. Open the project (**t120f324-riscv-tsemac.xml**) in the Efinity® software and review it.
2. Use the Efinity® Programmer and SPI active mode to download the bitstream file to your board. The example includes a bitstream file, **t120f324-riscv-tsemac.hex**. You use SPI active mode because you need to reset the FPGA.
3. Press SW2 (CRESET) on the Trion® T120 BGA324 Development Board to reset the FPGA. This reset ensures that the DDR 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 the Software Examples

The example comes with all of the standard Ruby software as well as two FreeRTOS examples in the **freertosDemoIPerf** and **freertosDemoTSE** directories. You can build the software yourself or you can use the pre-compiled **.elf** files.

Using this Example with the RISC-V SDK

Before working with the software included with this example design, you should already be familiar with using the Ruby SoC and RISC-V SDK. Specifically, you should know how to:

- Launch Eclipse using the **run_eclipse.bat** file (Windows) or **run_eclipse.sh** file (Linux) scripts.
- Set up global environment variables.
- Create and build a software project.
- Debug using the OpenOCD debugger.
- Open a UART terminal.

Learn more: Refer to the Ruby RISC-V SoC Hardware and Software User Guide for detailed instructions on how to perform these tasks.

Create a New Project

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

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

Import the **freertosDemoIPerf** example:

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

The **freertosDemoIPerf** project folder displays in the Project Explorer. The folder has the makefile and **main.c** source code as well as launch scripts for the OpenOCD Debugger.

Import Project Settings (Optional)

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

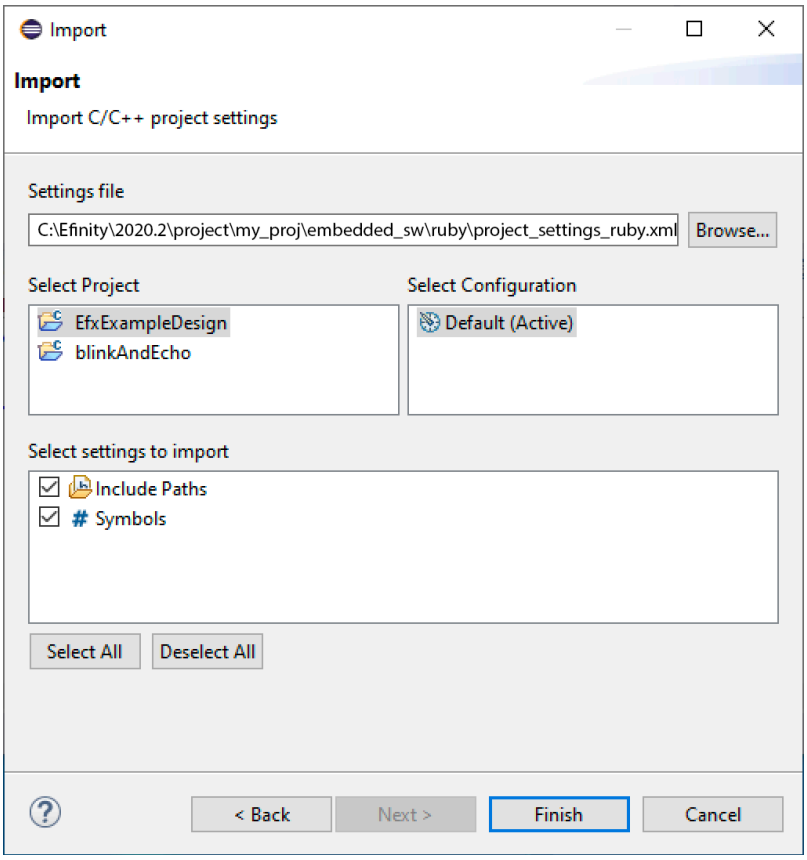
Note: You are not required to import the project settings to build. These settings simply make it easier for you to write and debug code.

To import the settings:

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

Option	Description
Windows	embedded_sw\RubyCore\config\project_settings_ruby.xml
Linux	embedded_sw/RubyCore/config_linux/project_settings_ruby.xml

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



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

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

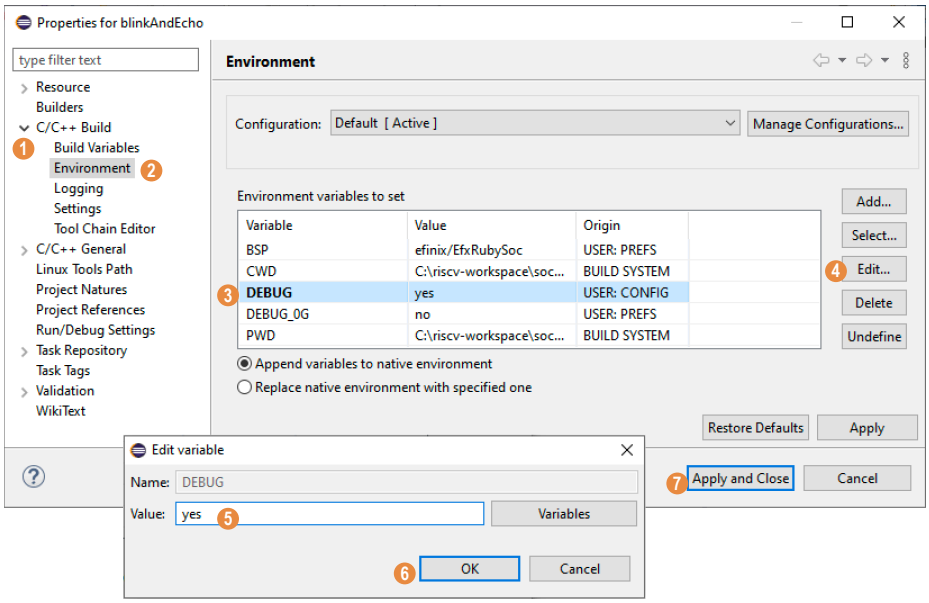
Enable Debugging

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

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

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

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



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

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

Build

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

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

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

Note: If your project will not clean or build, check that the path to FreeRTOS in the makefile matches it's location in your filesystem.

Windows: If the project cleans but the build fails, make sure that you saved the files at the root of the filesystem, e.g., **c:\t120f324-riscv-tsemac**. FreeRTOS uses long folder and filenames.

Import the Debug Configuration

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

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

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

Option	Description
Windows	**../<project>\embedded_sw\SapphireSoc\config**
Linux	**..\<project>\embedded_sw/SapphireSoc/config_linux**

5. Check the box next to **config** (Windows) or **config_linux** (Linux).
6. Click **Finish**.
7. Right-click the **freertosDemoIPerf** project name and choose **Debug As > Debug Configurations**.
8. Choose **GDB OpenOCD Debugging > default** (Trion FPGAs) or **default_ti** (Titanium FPGAs).
9. Enter `freertosDemoIPerf` in the **Project** box.
10. Enter `build\freertosDemoIPerf.elf` in the **C/C++ Application** box.
11. *Windows only:* you need to change the path to the **cpu0.yaml** file:
    a. Click the **Debugger** tab.
    b. In the **Config options** box, change `${workspace_loc}` to the full path to the **RubyCore** directory.

    Note: For the **cpu0.yaml** path, make sure to use **** as the directory separator because the first slash escapes the second one. For example, use:
    ...\\<project>\\embedded_sw\\SapphireSoc

12. Click **Debug**.

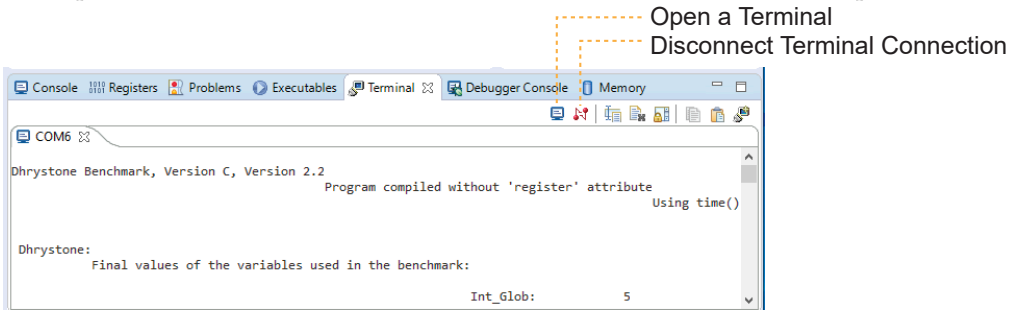
    Note: When you click **Debug**, the debugger sends a soft reset to the SoC, and then writes the user binary file to logical address 0x0000_1000, which is the starting address of the external memory. The Ruby SoC then jumps to logical address 0x0000_1000 to execute the user binary.

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

Open a Terminal

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

1. In Eclipse, choose **Window > Show View > Terminal**. The Terminal tab opens.



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

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

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

Iperf3 Server with FreeRTOS and TCP Stack

This application shows how FreeRTOS and a TCP stack TCP function with the TSEMAC IP core. The iPerf3 tool lets you actively measure the maximum achievable bandwidth on IP networks. You can tune parameters such as timing, buffers, and protocols. With this application you use the iPerf3 client to test the speed with the FPGA's Ipref3 server.

1. In Eclipse, run the application.
2. Click the **Resume** button or press F8 to resume code operation.
3. Go to your serial terminal. You should see the following messages display:

```
Hello world, this is FreeRTOS
Wait Ethernet Link up.....................OK
Info : Phy Linked up on 100Mbps.
Info : Set Mac Speed.
vIPerfTask: created TCP server socket 311024 bind port 5001: 0 listen 0
Use for example:
iperf3 -c C0A81F37 ip --port 5001 --bytes 10M [ -R ]
```

Note: If the Ethernet does not link up, check that you have set up the IP address and other network settings for the correct network adapter.

4. Open a second terminal or command prompt.
5. Change to the directory has the iPerf3 executable.
6. Type the command:

```
.\iperf3.exe -c 192.168.31.55 -p 5001 -n 10M -4
```

You should see output similar to the following in the terminal:

```
Connecting to host 192.168.31.55, port 5001
[  4] local 192.168.31.222 port 60953 connected to 192.168.31.55 port 5001
[ ID] Interval           Transfer     Bandwidth
[  4]   0.00-1.00  sec    896 KBytes    7.33 Mbits/sec
[  4]   1.00-2.00  sec    640 KBytes    5.25 Mbits/sec
[  4]   2.00-3.00  sec    640 KBytes    5.24 Mbits/sec
...
```

Echo ServerClient with FreeRTOS and TCP/UDP Stack

This application shows how to use FreeRTOS to communicate using Ethernet through a TCP and UDP stack. The FPGA functions as a client or server. TCP and UDP client/server capability are enabled by default. To change this setting, modify the following statements in **main.c**:

```
#define mainCREATE_SIMPLE_UDP_CLIENT_SERVER_TASKS     1
#define mainCREATE_TCP_ECHO_TASKS_SINGLE              1
#define mainCREATE_TCP_ECHO_SERVER_TASK               1
```

1. Click the **Resume** button or press F8 to resume code operation.
2. Go to your serial terminal. You shoud the following messages display:

```
Hello world, this is FreeRTOS
vTaskStartScheduler
Wait Ethernet Link up.....................OK
Info : Phy linked up on 100Mbps.
Info : Set Mac Speed.

IP Address: 192.168.0.55
Subnet Mask: 255.255.255.0
Gateway Address: 192.168.0.1
DNA Server Address: 8.8.8.8
```

Note: If the Ethernet does not link up, check that you have set up the IP address and other network settings for the correct network adapter.

3. Open a second terminal or command prompt.
4. Change to the directory has the **echotool.exe** executable.

Note: The **echotool.exe** application requires .NET 3.5 Windows support. If it is not installed, Windows prompts you that **An app on your PC needs the following Windows feature: .NET Framework 3.5 (includes .NET 2.0 and 3.0)** Click the link to download and install it.

FPGA as TCP Client, Computer as TCP Server

Type the command:

```
echotool.exe /p tcp /s 7
```

You should see output similar to the following in the terminal:

```
Client 192.168.0.55:19759 accepted at 4:03:00 PM
4:03:00 PM received [Hello Freertos+TCP Demo on Trion !! 0]
4:03:00 PM received [Hello Freertos+TCP Demo on Trion !! 1]
4:03:00 PM received [Hello Freertos+TCP Demo on Trion !! 2]
...
```

FPGA as TCP Server, Computer as TCP Client

Type the command:

```
echotool.exe 192.168.0.55 /p tcp /r 7
```

You should see output similar to the following in the terminal:

```
Hostname 192.168.0.55 resolved as 192.168.0.55

Reply from 192.168.0.55:7, time 2 ms OK
Reply from 192.168.0.55:7, time 1 ms OK
...
```

In the serial terminal, you should see messages similar to:

```
Received = TCP echo from DESKTOP-65SCMT3
Received = TCP echo from DESKTOP-65SCMT3
...
```

FPGA as UDP Client, Computer as UDP Server

Type the command:

```
echotool.exe /p udp /s 7
```

You should see output similar to the following in the terminal:

```
Waiting for UDP connection on port 7. Press any key to exit.
4:04:27 PM from 192.168.0.55:23784 received [Hello Freertos+UDP Demo on
 Tron !! 0]
4:04:27 PM from 192.168.0.55:23784 received [Hello Freertos+UDP Demo on
 Tron !! 8]
...
```

FPGA as UDP Server, Computer as UDP Client

Type the command:

```
echotool.exe 192.168.0.55 /p udp /r 7
```

You should see output similar to the following in the terminal:

```
Reply from 192.168.0.55:7, time 3 ms OK
Reply from 192.168.0.55:7, time 3 ms OK
...
```

In the serial terminal, you should see messages similar to:

```
Received = UDP echo from DESKTOP-65SCMT3
Received = UDP echo from DESKTOP-65SCMT3
...
```

Revision History

Table 1: Revision History

Date	Version	Description
November 2022	1.2	Added example flowchart. (DOC-980)
March 2021	1.1	Corrected the pin numbers for PMOD Connector J12 in Trion® T120 BGA324 Development Board. Added description stating that this design supports up to 100 Mbps Ethernet speed. Updated document title.
January 2021	1.0	Initial release.