



AN 077: USXGMII Example Design Application

AN077-v1.0
April 2026
www.efinixinc.com



Contents

| | |
|--|-----------|
| Introduction..... | 3 |
| Requirements..... | 4 |
| Functional Description..... | 5 |
| USXGMII Sapphire SoC Workflow..... | 6 |
| Timing Constraints..... | 8 |
| Setting Up the Hardware..... | 10 |
| Example Design Files..... | 11 |
| Programming the Board..... | 12 |
| Running the Example Design..... | 13 |
| USXGMII Speed Test Readback..... | 13 |
| Running Normal UDP Packets..... | 15 |
| Setting Up Python on Windows..... | 18 |
| Configuring the Network Device..... | 18 |
| Configuring the ARP Table..... | 19 |
| Configuring the Efinity Debugger..... | 20 |
| Configuring the Python Script..... | 21 |
| Running the eth.py Script..... | 22 |
| Jumbo Packet Test..... | 25 |
| Troubleshooting..... | 27 |
| Revision History..... | 28 |

Introduction

This example design demonstrates a simple Ethernet operation using the USXGMII Multirate Ethernet core in conjunction with the Ethernet 10G MAC core. The design showcases data transmission and reception through a straightforward loopback mechanism, providing a reference for system integration and validation.

The USXGMII Multirate Ethernet core supports IEEE Std 802.3 Clause 37 Auto-Negotiation (AN-37), enabling operation across multiple data rates, which include 10G, 5G, 2.5G, 1G, and 100M. This capability allows dynamic selection of link speeds across the supported data rates.



Learn more: This application note is for advanced users. If you are a beginner, refer to the following documents for better understanding on the software and hardware applications:

- [Efinity Software User Guide](#)
- [Get Started with the Titanium Ti375 N1156 Development Kit](#)
- [Ethernet 10G MAC Core User Guide](#)
- [USXGMII Multirate Ethernet Core User Guide](#)
- [Efinity Programmer User Guide](#)
- [Titanium Ethernet 10GBase-KR User Guide](#)

Requirements

Use the Titanium Ti375 N1156 Development Kit which includes the following:

- USB Type-C to Type-A cable
- 12 V, 6.25 A universal power adapter with 5.5 mm DC power converter
- Cooling fan
- Thermal pad
- Jumpers
- 4 standoffs

Additionally, you need the following hardware for this demonstration:

- Windows- or Linux-based computer with a 10 Gigabit Ethernet SFP+ port.



Note: Efinix had tested the example design on an Intel NIC X520-10G-xS-X8 with SFP+ port, and Wavelink 5 Gbps PCIe Network Card WL-NWP003 with an RJ-45 connector.

- Ethernet modules:
 - Two SFP+ modules supporting USXGMII (required for board-to-board or board-to-computer communication). Efinix uses the following SFP+ modules for hardware testing:
 - One FS SFP-10GM-T-30 Generic compatible 10M/100M/1G/2.5G/5G/10Gbps NBASE-T SFP+ copper 30m RJ-45 transceiver module.
 - One F-tone FTCS-10G-T-USXGMII 100M/1G/2.5G/5G/10Gb Copper-T RJ-45 30M transceiver module.
- Ethernet cables:
 - Fiber optic cable (required if the SFP+ modules are optical modules).
 - CAT6 or superior copper cable (required if the Ethernet modules support RJ-45 connector).

Software

This example design runs on:

- Efinity software v2025.2.288.4.15 or later.
- Efinity® RISC-V Embedded Software IDE v2025.2.0.4

Additional software:

- A Wireshark software to monitor the Ethernet packet traffic.
- This example design includes Eth Example Python script. Users must install Python on the target machine to run the Eth Example script. The Eth Example script has been tested with Python version v3.14.0.

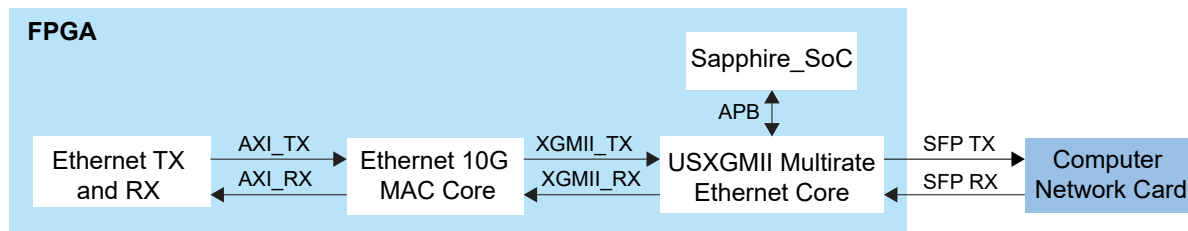
Functional Description

The example design implements USXGMII in quad 1 lane 2 using the USXGMII Multirate Ethernet core and the Ethernet 10G MAC core.

For this example design, the USXGMII Multirate Ethernet core is responsible for handling the data rate change. The USXGMII Multirate Ethernet core is a soft IP that implements 5 data rates of 10G, 5G, 2.5G, 1G, and 100M.

The Ethernet 10G MAC core is a configurable core that supports Ethernet speeds of up to 10 Gbps in full-duplex mode. It complies with the IEEE Std. 802.3-2008 specification and operates at 156.25 MHz. The Ethernet 10G MAC core used in the example design includes hardware support for MAC address filtering, enabling the system to selectively accept or discard incoming Ethernet frames based on their destination MAC address. This feature helps reduce unnecessary processing by ensuring that only relevant traffic is forwarded to higher layers of the system.

Figure 1: Design Data Flow



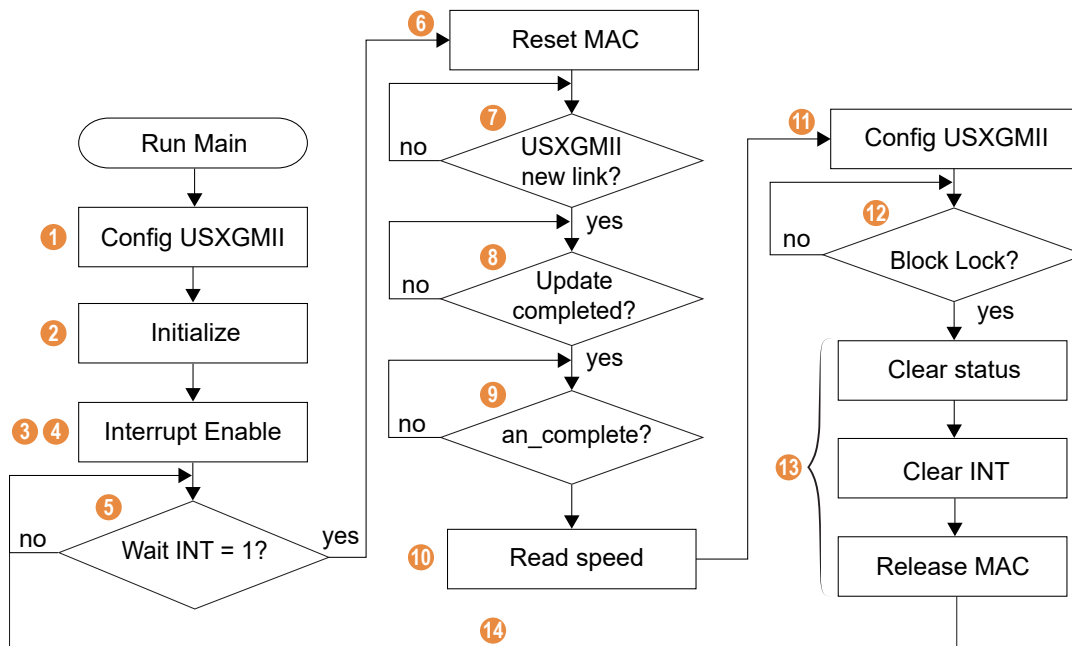
USXGMII Sapphire SoC Workflow

The following workflow shows the operation of the Ethernet 10G MAC core and USXGMII Multirate Ethernet core, configured using a Sapphire SoC. The Sapphire SoC communicates with the USXGMII Multirate Ethernet core through the APB interface. The Sapphire SoC performs the initialization and configuration to enable an interrupt signal that is connected to the IRQ transceiver. When the transceiver IRQ is triggered, the Sapphire SoC resets the Ethernet 10G MAC core and updates it with a new USXGMII data rate.



Learn more: For more details on the Sapphire SoC configuration workflow, refer to the Sapphire SoC Configuration Guideline in [Sapphire RISC-V SoC Hardware and Software User Guide](#).

Figure 2: USXGMII Soft SoC Workflow



The following details the workflow of the USXGMII soft SoC:

1. Configure the USXGMII to enable the TX and RX by writing 1'b0 to `rx_sync_reset` and writing 1'b1 to `tx_datapath_en` and `signal_ok`. These bits are available in `control_register[2:0]`.
2. Clear `status_register` status by writing 1'b1 into `ctc_o_u_flow`, `hi_bit_error`, `tx_fault`, and `rx_fault`.
3. Configure the Sapphire SoC INT interrupt.
4. Enable USXGMII interrupt by writing 1'b1 to `usxgmii_new_link_info_en` and `usxgmii_link_sts_upd_en`. These bits are available in `interrupt_enable_register[25:24]`.
5. Wait for the INT interrupt to happen.
6. When an interrupt happens, reset the MAC.
7. Periodically poll for `usxgmii_new_link_info` until the returned value is 1'b1 to indicate the USXGMII new link. This bit is available in the `interrupt_status_register[25]`.
8. Then, periodically poll for the `usxgmii_link_sts_up` until the returned value is 1'b1 to indicate that the link status update has been completed. This bit is available in `interrupt_status_register[24]`.

9. Then, periodically poll for the `an_complete` until the returned value is `1'b1` to indicate that the USXGMII auto-negotiation has been completed. This bit is available in `status_register[1]`.
10. After a new link is established, the current speed is read.
11. Next, configure the `usx_speed` with the new USXGMII speed. These bits are available in `control_register[16:14]`.
12. Poll the `block_lock` until the returned value is `1'b1` to indicate that the USXGMII has achieved block synchronization. This bit is available in `status_register[0]`.
13. Finally, clear the `status_register` status, clear the INT interrupt, and release the MAC.
14. Return to step 5. Wait for the INT interrupt to happen.



Learn more: For details on the registers used, refer to the Register Map chapter in the [Titanium Ethernet 10GBase-KR User Guide](#).

Timing Constraints

The USXGMII Multirate Ethernet core requires timing constraints for its clock division module, TX FIFO, and RX FIFO.

Clock Constraints

The following constraints `create_clock` and `create_generated_clock` are for `L<x>_PCS_CLK` and `L<x>_MAC_CLK`, where `x` is the lane number 0, 1, 2, and 3. In the following example, `Q1_L2_PCS_CLK` refers to `L<x>_PCS_CLK`, and `Q1_L2_mac_clk_div2` and `Q1_L2_mac_clk_div4` refers to `L<x>_MAC_CLK`.

Figure 3: Example Constraint `create_clock` and `create_generated_clock`

```
create_clock -period 6.4 -name Q1_L2_PCS_CLK [get_ports {Q1_L2_PCS_CLK}]

create_generated_clock [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk_y.inst_12/
inst_clkdiv/mac_clk_div2~FF[Q] \
-source [get_ports Q1_L2_PCS_CLK] \
-divide by 2 \
-name Q1_L2_mac_clk_div2

create_generated_clock [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk_y.inst_12/
inst_clkdiv/mac_clk_div4~FF[Q] \
-source [get_ports Q1_L2_PCS_CLK] \
-divide by 4 \
-name Q1_L2_mac_clk_div4
```

Refer to the Clock Sources section in the [USXGMII Multirate Ethernet Core User Guide](#) for more details.

Set False Path Constraints

You set the `set_false_path` constraints for the USXGMII Multirate Ethernet core FIFO. Use one constraint for each lane. The path constraints for the `genblk<y>.inst_1<x>` where `x` and `y` values should be based on the following table.

Table 1: `set_false_path` `x` and `y` Values

| Lane | x | y |
|------|---|---|
| 0 | 0 | 4 |
| 1 | 1 | 3 |
| 2 | 2 | 2 |
| 3 | 3 | 1 |

Refer to the table of SDC instance names Timing Constraints: SDC of the [USXGMII Multirate Ethernet Core User Guide](#) for more details.

Figure 4: Example of set_false_path for Lane 2

```

set_false_path \
  -from [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/cfg_div2_apb~FF|CLK]
  \
  -to [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_cfg_div2/
      async_reg*~FF|D]

set_false_path \
  -from [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/cfg_div4_apb~FF|
CLK] \
  -to [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_cfg_div4/
      async_reg*~FF|D]

set_false_path \
  -from [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/s_cfg_div2_diff/
toggle_src~FF|CLK] \
  -to [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/s_cfg_div2_diff/
genblk1.efx_asyncreg_toggle_src/async_reg*~FF|D]

set_false_path \
  -from [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/s_cfg_div4_diff/
toggle_src~FF|CLK] \
  -to [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/s_cfg_div4_diff/
genblk1.efx_asyncreg_toggle_src/async_reg*~FF|D]

set_false_path \
  -from [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_tx_rpl/
inst_fifo_tx_rpl/xefx_fifo_ctl/async_clk.waddr_cntr_gry_r*~FF|CLK] \
  -to [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_tx_rpl/
inst_fifo_tx_rpl/xefx_fifo_ctl/async_clk.wr2rd_addr_sync/async_reg*~FF|D]

set_false_path \
  -from [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_tx_rpl/
inst_fifo_tx_rpl/xefx_fifo_ctl/async_clk.raddr_cntr_gry_r*~FF|CLK] \
  -to [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_tx_rpl/
inst_fifo_tx_rpl/xefx_fifo_ctl/async_clk.xrd2wr_addr_sync/async_reg*~FF|D]

set_false_path \
  -from [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_rx_rpl/
inst_fifo_rx_rpl/xefx_fifo_ctl/async_clk.waddr_cntr_gry_r*~FF|CLK] \
  -to [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_rx_rpl/
inst_fifo_rx_rpl/xefx_fifo_ctl/async_clk.wr2rd_addr_sync/async_reg*~FF|D]

set_false_path \
  -from [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_rx_rpl/
inst_fifo_rx_rpl/xefx_fifo_ctl/async_clk.raddr_cntr_gry_r*~FF|CLK] \
  -to [get_pins inst_Q1_usxgmii_an/u_efx_usxgmii_an_37/genblk2.inst_l2/inst_rx_rpl/
inst_fifo_rx_rpl/xefx_fifo_ctl/async_clk.xrd2wr_addr_sync/async_reg*~FF|D]

```

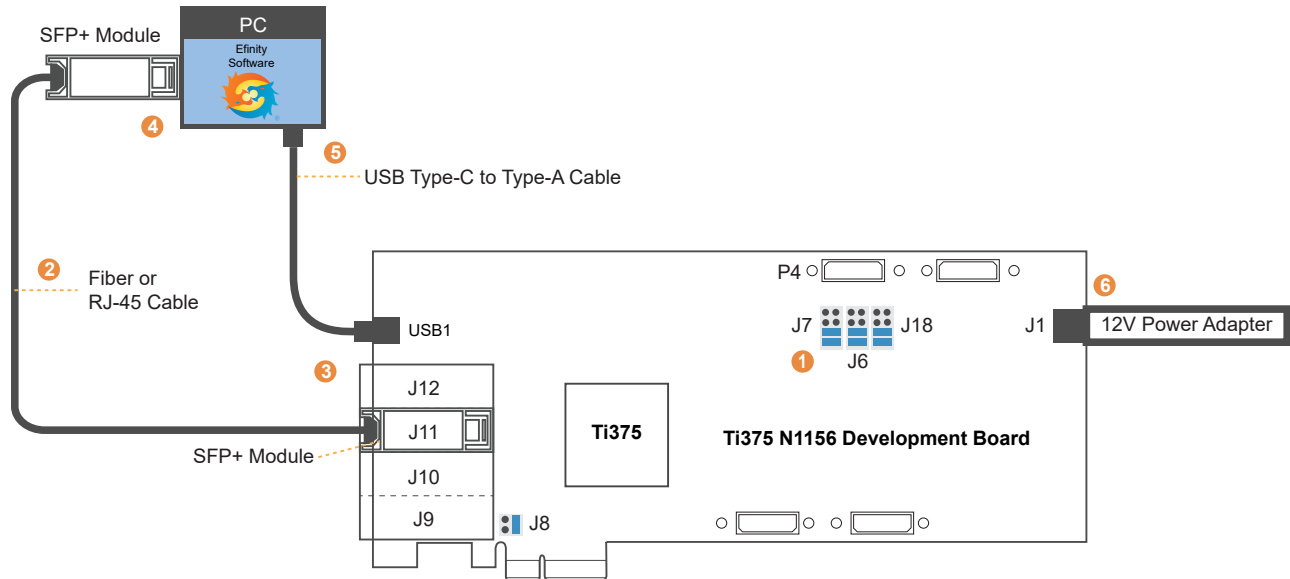
Setting Up the Hardware

The section guides you in setting up the Titanium Ti375 N1156 Development Board to a computer via SFP+ module, and setting the jumpers on the board.



Important: Turn off the power supply before connecting the connectors on your computer and the Titanium Ti375 N1156 Development Board.

Figure 5: Visual Overview of Hardware Setup



Follow these steps to setup your hardware for the demonstration design:

1. Ensure that the jumpers are set as shown in the following table:

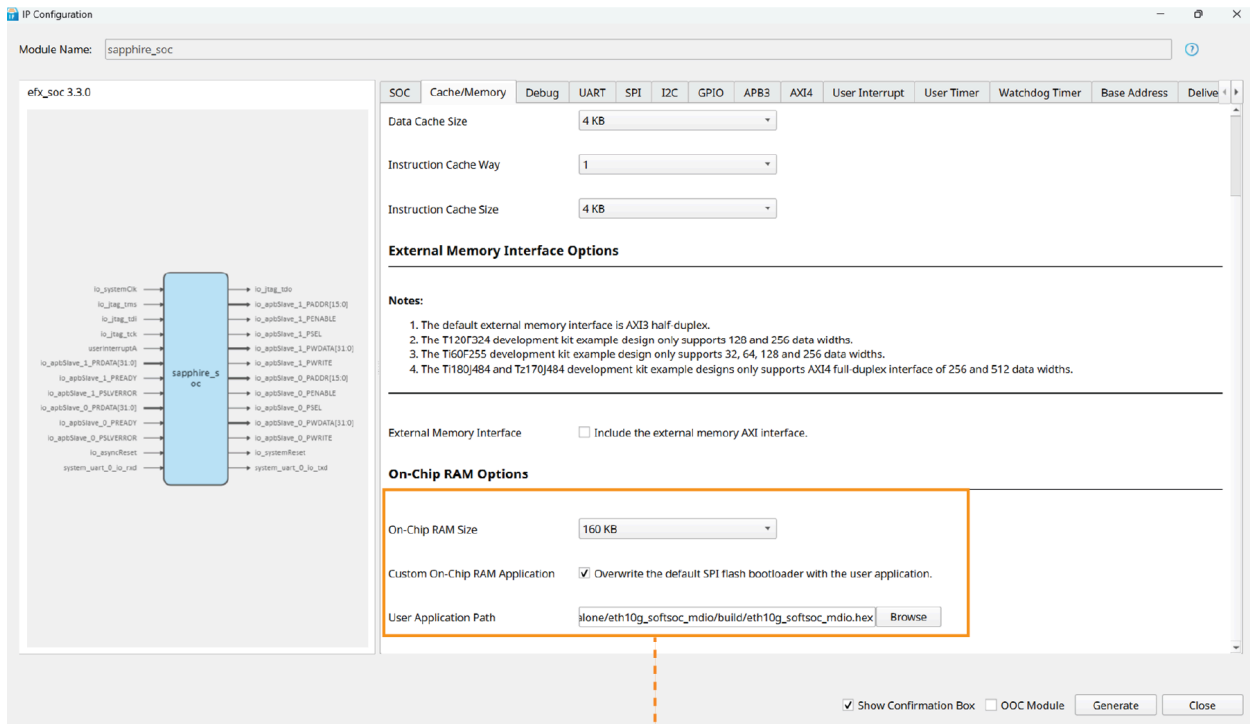
| Header | Pins to Short |
|--------|---------------|
| J6 | 5-6 7-8 |
| J7 | 5-6 7-8 |
| J18 | 5-6 7-8 |

2. Check your network interface card for the type of connector to use. Use a fiber, an RJ-45 cable, or a direct access copper (DAC) to connect the SFP+ modules on the computer to the Titanium Ti375 N1156 Development Board.
3. Connect the SFP+ module (F-tone FTCS-10G-T-USXGMII) to the SFP+ port J11 on the Titanium Ti375 N1156 Development Board.
4. Then, connect the other end to the network interface card, with SFP+ (FS SFP-10GM-T-30) when connected to Intel NIC X520-10G-XS or connect the RJ-45 connector to the Wavelink 5 Gbps PCIe Network Card WL-NWP003.
5. Connect the USB header on the Titanium Ti375 N1156 Development Board to a USB port on your computer.
6. Ensure the 12V power adapter is turned off and connect to J1 port on the Titanium Ti375 N1156 Development Board. Turn on the 12V power adapter.

Example Design Files

This example design does not include the Sapphire SoC application binary. The Sapphire SoC has been loaded using the **Custom On-Chip RAM Application**. The **Custom On-Chip RAM Application** is enabled in the Sapphire SoC to overwrite the default SPI Flash bootloader with the user application. You do not need to load the application separately.

Figure 6: Custom On-Chip RAM Application



The Custom On-Chip RAM Application to overwrite the default SPI flash bootloader with the image file stated in the User Application Path.

Table 2: Design Example Files and Directories

| Folder | Description |
|--|---|
| rtl | Folder containing common RTL files for the project. |
| embedded_sw\sapphire_soc\software \standalone\eth10g_softsoc_mdio | Folder containing soft_SoC project files. |
| ip | Folder containing IP. |
| bitstream | Folder containing pre-compiled bitstream. |

Programming the Board

Before programming the board, download the example design files from the [Efinix Example Design](#) website.

Follow these steps to program the Titanium Ti375 N1156 Development Board with the bitstream.

1. Open the Efinity software.
2. Browse to the file path `<path>/usxgmii eth10g_softsoc_mdio.xml` to open the project file.
3. Open the Efinity Programmer.
4. Choose **Programming Mode > SPI Active using JTAG Bridge**.
5. For the **Bitstream File**, browse to the **/bitstream** folder and select **eth10g_softsoc_mdio.hex**.
6. Click **Start Program** to begin programming the Titanium Ti375 N1156 Development Board with the selected bitstream file.
7. Press the **SW2 CRESET_N** button on the Titanium Ti375 N1156 Development Board to reset the FPGA.

Running the Example Design

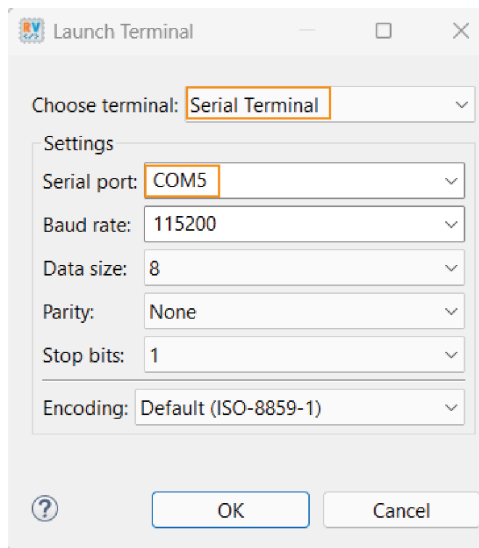
This section describes how to run various hardware tests. You should already be familiar with Efinity RISC-V Embedded Software IDE, Efinity Programmer, and Debugger. For more information on the Efinity Programmer, refer to the [Efinity Software User Guide](#).

USXGMII Speed Test Readback

The example design prints the connection status onto the serial terminal. You can use any terminal programs, such as Putty, Termit, or the built-in terminal in the Efinity RISC-V Embedded Software IDE, to connect to the UART. The following instructions explain how to use the built-in terminal. These instructions are similar for the other terminal programs.

1. Launch the Efinity RISC-V Embedded Software IDE. Choose **Window > Show View > Terminal**.
2. Click **Open a Terminal** button.
3. Once the Launch Terminal window is launched, set the following:

Figure 7: Launch the Terminal



- In **Choose Terminal**, choose the **Serial Terminal** option.
 - In **Serial port**, choose **COM n** , where n is the port number for your UART module.
 - Other settings can remain as default.
4. Click **OK** and the terminal opens up a connection to the UART.

The computer receives data from the Titanium Ti375 N1156 Development Board and prints out the status on the terminal. You can change the link speed on the computer side and watch the auto-negotiation restarting and adjusting to the new speed.

Figure 8: Computer Terminal Status

```

Problems Tasks Console Properties Terminal x
COMS x
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 1
Info : usxgmii speed is 100M
Info : usxgmii register 0xC00200 is 0x80301003
Info : Block lock is 1
Info : usxgmii pcs_status is 0x38000003
Info : usxgmii pcs_status is 0x00000003
Info : usxgmii intr clear 0xC00260 is 0x00000000
Info : usxgmii cfg done!
INT : Normal Work,Waiting Interrupt.
*****
INT : Interrupt Trigger,Enter the Interrupt handing.
Info : USXGMII new link information seen.
Info : usxgmii_link_sts_upd is 0,0x02000000
Info : USXGMII link status update complete.
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 0
Info : an_complete status is 1
Info : usxgmii speed is 10G
Info : usxgmii register 0xC00200 is 0x80311003
Info : Block lock is 1
Info : usxgmii pcs_status is 0x38000003
Info : usxgmii pcs_status is 0x00000003
Info : usxgmii intr clear 0xC00260 is 0x00000000
Info : usxgmii cfg done!
INT : Normal Work,Waiting Interrupt.
*****

```

Running Normal UDP Packets

The example design is intended to transmit UDP packets and supports multiple operating modes. The transmit packet can be customized in the `vio0` tab. You can configure the UDP packet test that supports incremental pattern generation, random packet generation, and loopback test using the Python script. Refer to [Setting Up Python on Windows](#) on page 18 for more details. Use the `Q1_L2_pat_type` to configure the different tests:

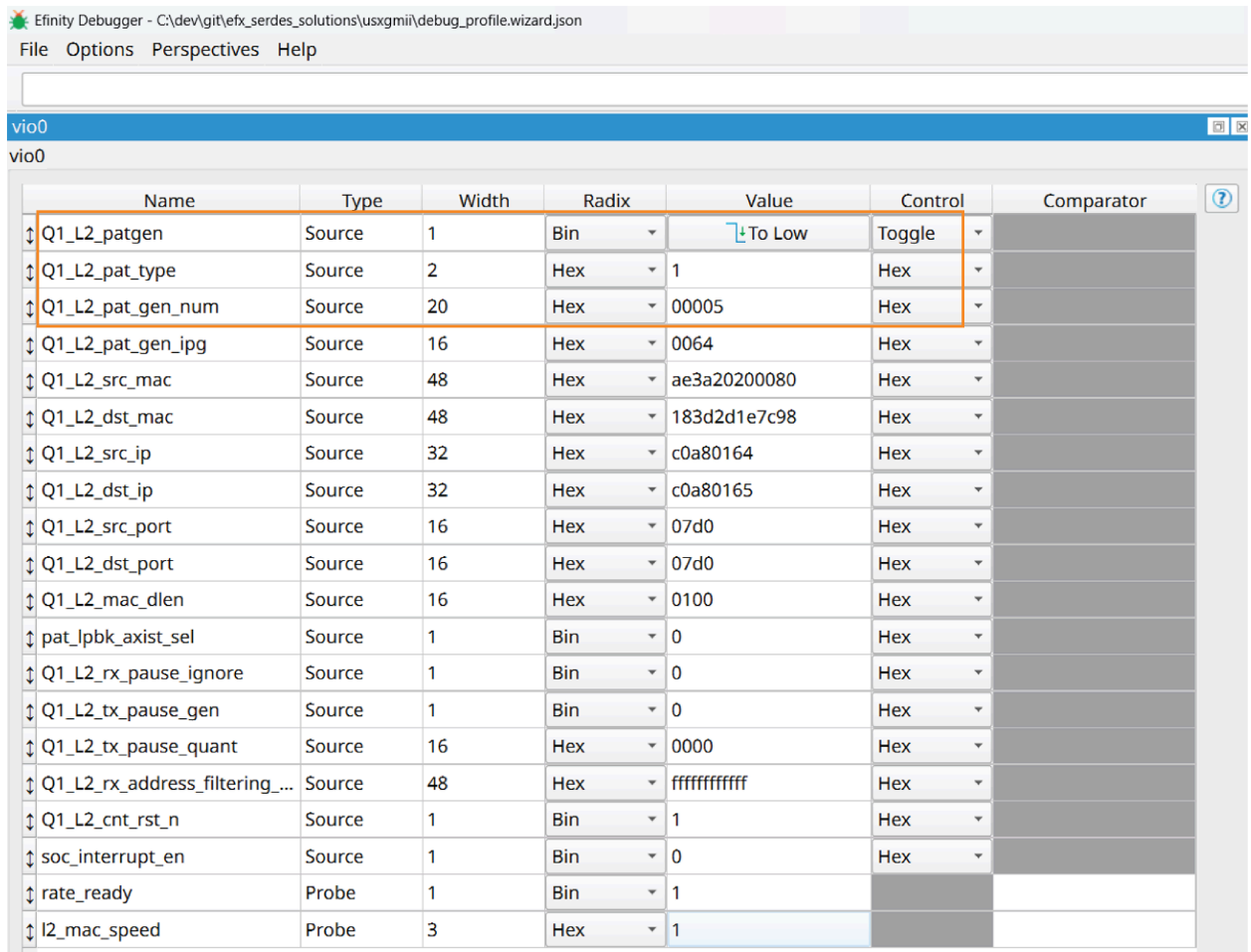
- `Q1_L2_pat_type = 0`, for incremental pattern.
- `Q1_L2_pat_type = 1`, for random pattern.
- `Q1_L2_pat_type = 2`, for loopback test (used with Python test in the following section).

To send the UDP packets, follow these steps:

1. Launch the Wireshark application.
2. Open the Efinity Debugger.
3. Click **Connect Debugger** button in the Efinity Debugger to link the Debugger to the Titanium Ti375 N1156 Development Board.
4. In the `vio0` tab, ensure that the `Q1_L2_pat_type` is set to `0` for incremental pattern.
5. Set the `Q1_L2_patgen` to **Toggle**. When `Q1_L2_patgen` is asserted high, packet transmission is initiated.

The number of packets transmitted is determined by **Q1_L2_pat_gen_num**.
 If **Q1_L2_pat_gen_num** is set to 0, packets are transmitted continuously until **Q1_L2_patgen** is deasserted.

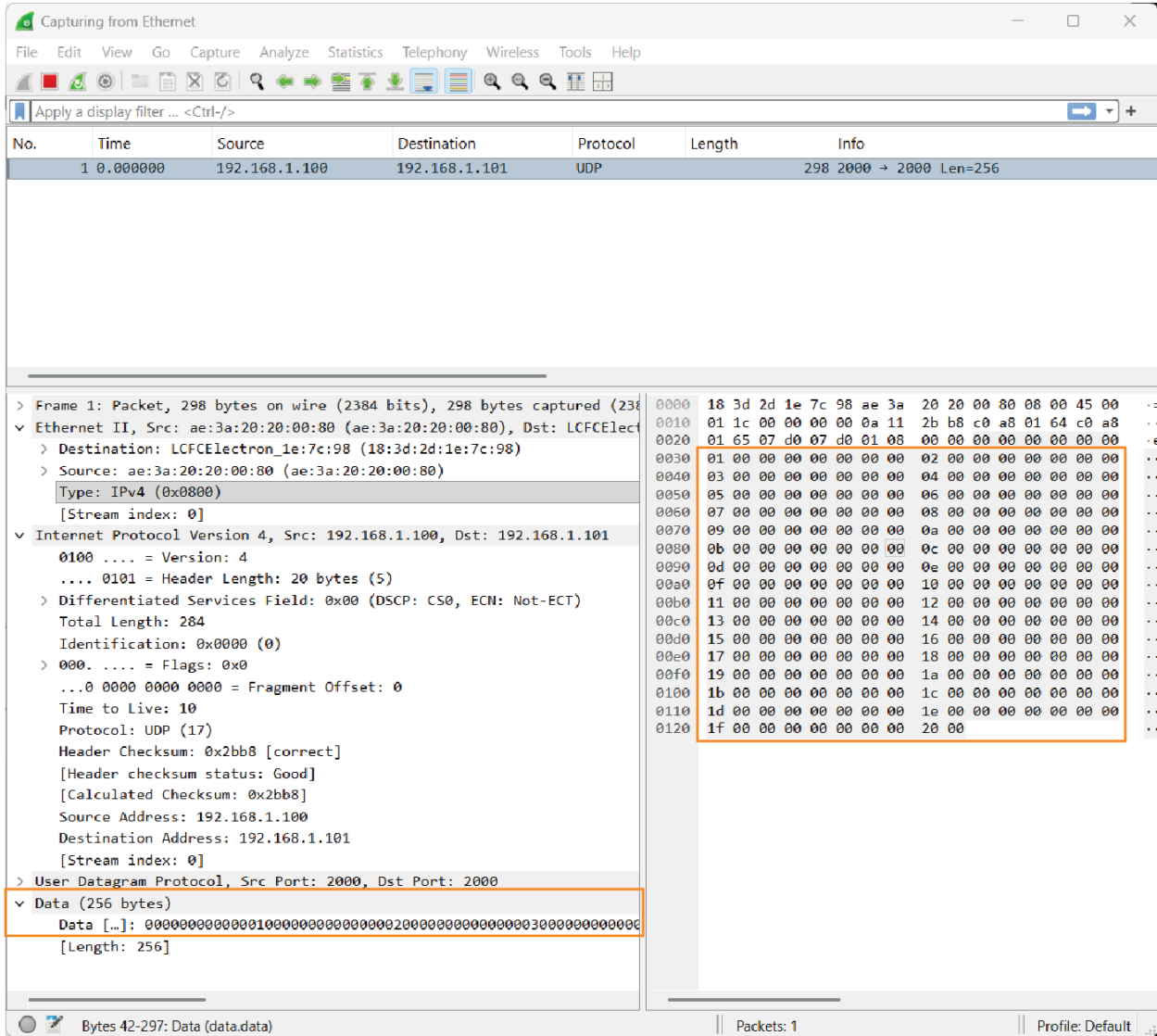
Figure 9: Running the UDP Packets



Launch Wireshark to view the packets received from the UDP pattern generator, where a fixed payload is sent. Here, you can see the data increments. Additionally, you can change the packet size by modifying the **Q1_L2_mac_dlen** value.

The output is shown in **Figure 10: Viewing of Data Packets Received in Wireshark** on page 17.

Figure 10: Viewing of Data Packets Received in Wireshark



Setting Up Python on Windows

This example design comes with a simple Python read and write script that interfaces with the FPGA. Before executing the Python script, you must establish communication between the FPGA and the host PC.

Configuring the Network Device

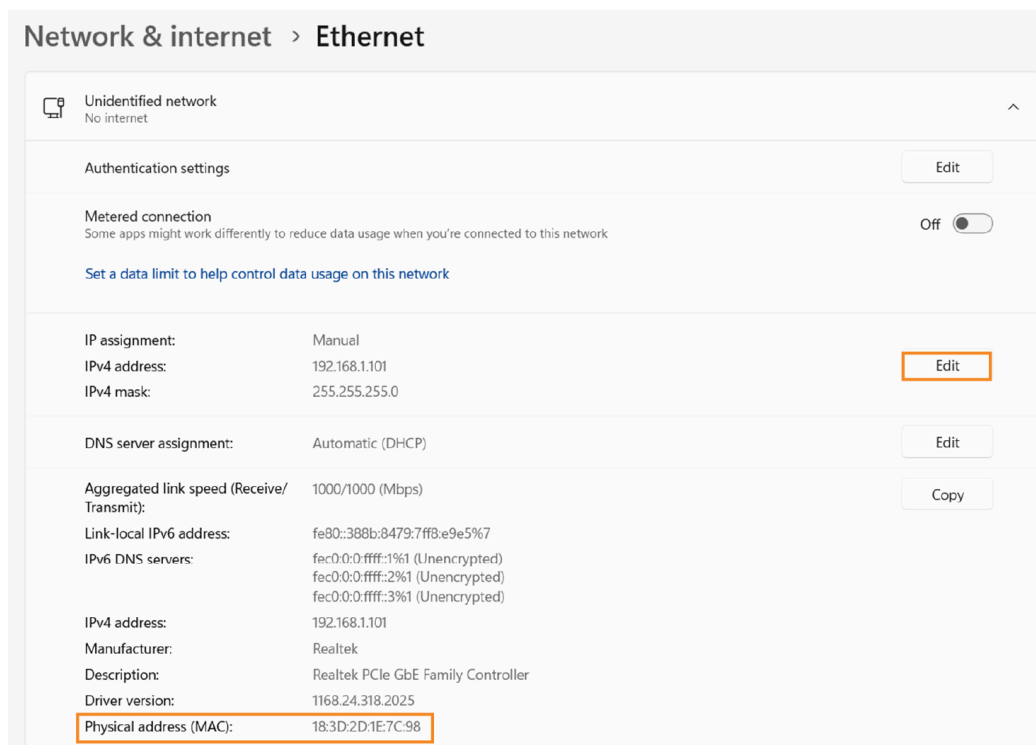
The MAC physical address on your PC is used as the Ethernet 10G MAC core address filtering. The MAC address is unique to the network card used. The Windows MAC address can be obtained by following these steps:

1. Click the **Start** button on your PC's desktop.
2. Type **Settings** in the Search box.
3. Select **Network & Internet**.



Note: The Physical address (MAC) of your PC is set as the destination MAC in the Efinity Debugger. See [Configuring the Efinity Debugger](#) on page 20.

Figure 11: Obtaining the MAC Address



4. Click the **Edit** button beside the IPv4 address to input the IP address that you plan to use. In this example design, the IP address used is **192.168.1.101**.



Note: The IP address used for the FPGA must match the IP address used in the Python script.

Figure 12: Setting the Address in IPv4

Configuring the ARP Table

An ARP (Address Resolution Protocol) table is essential to map logical IP addresses to physical MAC addresses, enabling devices to communicate on a local area network (LAN). It acts as a cache, reducing network congestion by eliminating the need to broadcast an ARP request for every packet transmission. You must set up the ARP table to allow the installed network device in the PC to communicate with the FPGA. This is because the FPGA does not support ARP responses. You need to add a permanent ARP entry manually to your operating system's ARP cache. This method prevents the operating system from issuing an ARP request when sending a packet to an IP address.

1. Open a Command Prompt in Administrator mode.
2. Type `netsh interface ipv4 add neighbors "Ethernet" "192.168.1.100" "ae-3a-20-20-00-80" store=persistent` in the Command Prompt to map the IP address and the MAC address.
3. Ensure the ARP table is set correctly by typing `arp -a` in the Command Prompt.

Figure 13: Generating the ARP Table

```

Administrator: Command Prompt
C:\Windows\System32>netsh interface ipv4 add neighbors "Ethernet" "192.168.1.100" "ae-3a-20-20-00-80" store=persistent

C:\Windows\System32>arp -a

Interface: 192.168.1.101 --- 0x7
Internet Address      Physical Address      Type
192.168.1.100         ae-3a-20-20-00-80    static
224.0.0.13            01-00-5e-00-00-0d    static
224.0.0.22            01-00-5e-00-00-16    static
224.0.0.251          01-00-5e-00-00-fb    static
  
```

Configuring the Efinity Debugger

To configure the Efinity Debugger, follow these steps:

1. The MAC and IP address must match the values set in the Efinity Debugger. In the Efinity Debugger, you must ensure that the MAC addresses, IP addresses, and ports are configured correctly. The settings used in this example design are as follows:
 - Fpga IP—192.168.1.100 (hexadecimal value of C0A80164)
 - Host IP—192.168.1.101 (hexadecimal value of C0A80165)
 - Source Port—2000
 - Destination Port—2000
2. Ensure that the `Q1_L2_pat_type` is set to 2 for loopback testing.
3. You may configure the number of packets sent to the host. You may either send a fixed number of packets by configuring `Q1_L2_pat_gen_num`, or send continuously by setting `Q1_L2_pat_gen_num` to 0.

Figure 14: Settings in Efinity Debugger

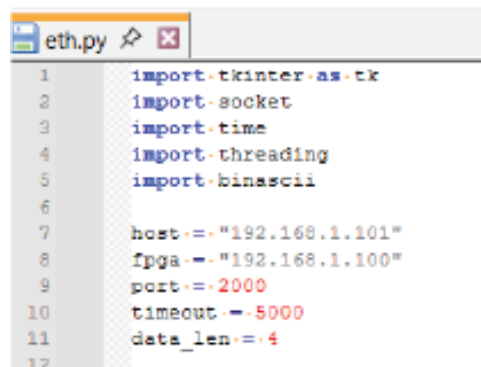
| Name | Type | Width | Radix | Value | Control | Comparator |
|--------------------------------|--------|-------|-------|--------------|---------|------------|
| Q1_L2_patgen | Source | 1 | Bin | To High | Toggle | |
| Q1_L2_pat_type | Source | 2 | Hex | 2 | Hex | |
| Q1_L2_pat_gen_num | Source | 20 | Hex | 00005 | Hex | |
| Q1_L2_pat_gen_ipg | Source | 16 | Hex | 0064 | Hex | |
| Q1_L2_src_mac | Source | 48 | Hex | ae3a20200080 | Hex | |
| Q1_L2_dst_mac | Source | 48 | Hex | 183d2d1e7c98 | Hex | |
| Q1_L2_src_ip | Source | 32 | Hex | c0a80164 | Hex | |
| Q1_L2_dst_ip | Source | 32 | Hex | c0a80165 | Hex | |
| Q1_L2_src_port | Source | 16 | Dec | 02000 | Hex | |
| Q1_L2_dst_port | Source | 16 | Dec | 02000 | Hex | |
| Q1_L2_mac_dlen | Source | 16 | Hex | 0100 | Hex | |
| pat_lpbk_axist_sel | Source | 1 | Bin | 0 | Hex | |
| Q1_L2_rx_pause_ignore | Source | 1 | Bin | 0 | Hex | |
| Q1_L2_tx_pause_gen | Source | 1 | Bin | 0 | Hex | |
| Q1_L2_tx_pause_quant | Source | 16 | Hex | 0000 | Hex | |
| Q1_L2_rx_address_filtering_... | Source | 48 | Hex | fffffffffff | Hex | |
| Q1_L2_cnt_rst_n | Source | 1 | Bin | 1 | Hex | |
| soc_interrupt_en | Source | 1 | Bin | 0 | Hex | |
| rate_ready | Probe | 1 | Bin | 1 | | |
| l2_mac_speed | Probe | 3 | Hex | 1 | | |

Configuring the Python Script

To use a different IP address and port, you need to configure the Python script:

1. Open the **eth.py** script in any editor of your preference. The Python script is in the /**scripts/** folder.
2. (Optional) Update the **host**, **fpga**, and **port** value if you want to use a value other than the default.

Figure 15: Configuring the Host, FPGA, and Port in Python Script



```
eth.py
1  import tkinter as tk
2  import socket
3  import time
4  import threading
5  import binascii
6
7  host = "192.168.1.101"
8  fpga = "192.168.1.100"
9  port = 2000
10 timeout = 5000
11 data_len = 4
12
```

3. Save and close the file.

Running the eth.py Script

The Eth Example Python script handles the UDP packet transmission and reception. You can send and receive 32 bits of hexadecimal data (additional data is truncated). The Eth Example Python script continuously reads and displays the incoming packet.

You need to configure the Efinity Debugger `Q1_L2_pat_type = 2` for loopback testing and set the `Q1_L2_pat_gen_num = 0` for continuous mode. Then, toggle the `Q1_L2_patgen` to high when it is ready to start sending packets from the FPGA.

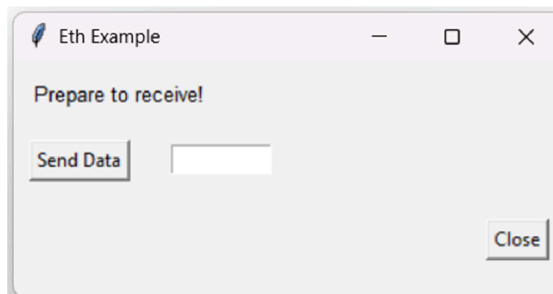
Remember: Ensure that the FPGA is configured correctly. Refer to [Configuring the Efinity Debugger](#) on page 20.

To run the `eth.py` script, follow these steps:

1. Open a command prompt or terminal and change to the directory filepath: `<project folder>\usxgmii\scripts\`.
2. Type `python eth.py` and press **Enter** to launch the Eth Example Python script. An Eth Example window shows up.

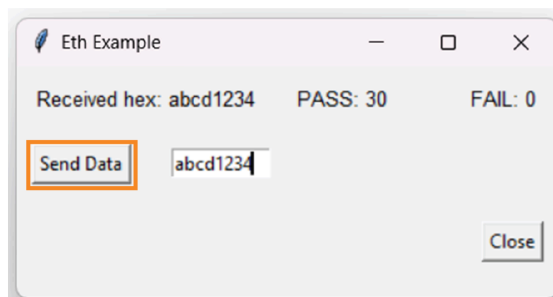
Remember: Refer to [Configuring the Network Device](#) on page 18 to set the MAC and IP address.

Figure 16: Running the eth.py Script



3. Click **Send Data** after entering your desired value.

Figure 17: Sending Data from Eth Example



4. Click the **Close** button in the Eth Example to close the socket connection.

The example design includes a debug core. You can use the Efinity Debugger to view the transceiver status and the Ethernet 10G MAC core statistic reports. Follow these steps:

1. In Efinity Debugger, click **Connect Debugger** to link the Debugger to the Titanium Ti375 N1156 Development Board.
2. In the **la0** tab, click **Add Trigger Condition** to add a trigger point. Choose the targeted trigger condition from the list of available signals, e.g., **Q1_L2_rx_axis_mac_tvalid**.
3. Choose **R(0-to-1 transition)** to set the triggering condition value.
4. (Optional) If you need to add more trigger conditions, repeat steps 3 and 4.
5. Choose **Trigger Condition** to set the triggering conditions for multiple signals.
6. Click **Run** to start the Logic Analyzer.



Note: Refer to Using the Debug Perspective in [Efinity Software User Guide](#) for detailed steps to view the waveform.

Results:

Upon completion, the Efinity Debugger produces a waveform when the triggering conditions are met. The `tvalid` is asserted whenever the Python script sends a valid packet to the FPGA. See [Figure 18: Debug Perspective GUI - Logic Analyzer](#) on page 23.

Figure 18: Debug Perspective GUI - Logic Analyzer

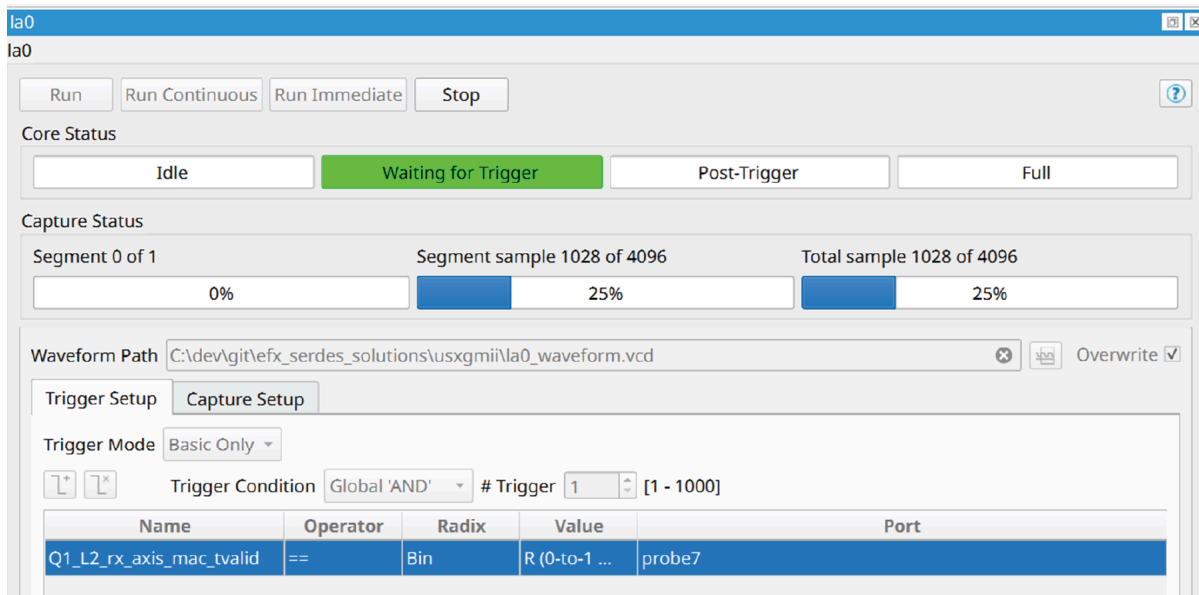
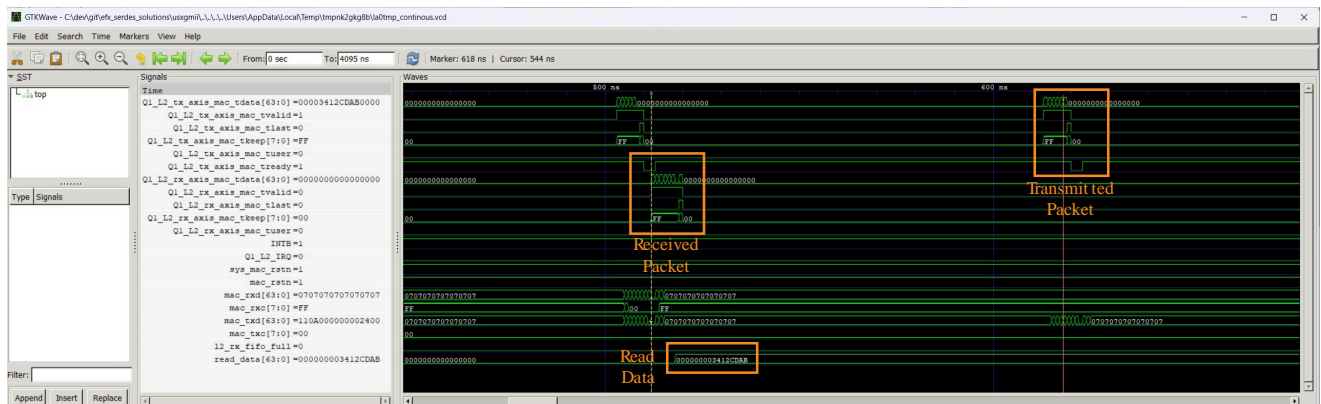


Figure 19: Data Packet Waveform



You can observe that the number of transmitted and received frames through the **Q1_L2_cnt_tx_frame_transmitted_good** and **Q1_L2_cnt_rx_frame_received_good** is incremented. Additionally, the number of dropped error RX frames through address filtering can be observed in **Q1_L2_cnt_rx_frame_filtered_by_address**.



Learn more: For more information, refer to the Statistic Reporting section in the [Ethernet 10G MAC Core User Guide](#).

Figure 20: Transceiver Status and Ethernet 10G MAC Core Statistic Reporting

Efinity Debugger - C:\dev\git\efx_serdes_solutions\usxgmii\debug_profile.wizard.json
 File Options Perspectives Help

vio1

| Name | Type | Width | Radix | Value | Control | Comparator |
|--|-------|-------|-------|-----------|---------|------------|
| Q1_L2_phy_init_done | Probe | 1 | Bin | 1 | | |
| Q1_L2_cnt_tx_frame_transmitted_go... | Probe | 32 | Dec | 000000010 | | |
| Q1_L2_cnt_tx_frame_pause_mac_ctrl | Probe | 32 | Dec | 000000000 | | |
| Q1_L2_cnt_tx_frame_error_txfifo_ove... | Probe | 32 | Dec | 000000000 | | |
| Q1_L2_cnt_tx_frame_is_fe | Probe | 32 | Hex | 00000000 | | |
| Q1_L2_cnt_rx_frame_received_good | Probe | 32 | Dec | 000000017 | | |
| Q1_L2_cnt_rx_frame_error_fcs | Probe | 32 | Dec | 000000000 | | |
| Q1_L2_cnt_rx_frame_pause_mac_ctrl | Probe | 32 | Dec | 000000000 | | |
| Q1_L2_cnt_rx_frame_errors | Probe | 32 | Dec | 000000178 | | |
| Q1_L2_cnt_rx_frame_received_total | Probe | 32 | Dec | 000000195 | | |
| Q1_L2_cnt_rx_frame_undersized | Probe | 32 | Dec | 000000000 | | |
| Q1_L2_cnt_rx_frame_oversized | Probe | 27 | Dec | 000000000 | | |
| Q1_L2_cnt_rx_frame_mismatched_le... | Probe | 32 | Dec | 000000000 | | |
| Q1_L2_cnt_rx_frame_filtered_by_add... | Probe | 32 | Dec | 000000178 | | |
| Q1_L2_rpt_rx_frame_length | Probe | 14 | Hex | 0072 | | |
| Q1_L2_BLOCK_LOCK | Probe | 1 | Bin | 1 | | |
| Q1_L2_HI_BER | Probe | 1 | Bin | 0 | | |
| Q1_L2_IRQ | Probe | 1 | Bin | 0 | | |
| Q1_L2_PCS_STATUS | Probe | 1 | Bin | 1 | | |
| Q1_L2_PHY_INTERRUPT | Probe | 1 | Bin | 0 | | |
| Q1_L2_PMA_XCVR_POWER_STATE_ACK | Probe | 4 | Hex | 1 | | |
| Q1_L2_PMA_XCVR_PLLCLK_EN_ACK | Probe | 1 | Bin | 1 | | |
| Q1_L2_PMA_RX_SIGNAL_DETECT | Probe | 1 | Bin | 1 | | |
| Q1_L2_PMA_XCVR_POWER_STATE_REQ | Probe | 4 | Hex | 1 | | |
| Q1_L2_PMA_XCVR_PLLCLK_EN | Probe | 1 | Bin | 1 | | |
| Q1_PMA_CMN_READY | Probe | 1 | Bin | 1 | | |
| clk50m_rstn | Probe | 1 | Bin | 1 | | |
| Q1_L2_pcs_rst_n | Probe | 1 | Bin | 1 | | |

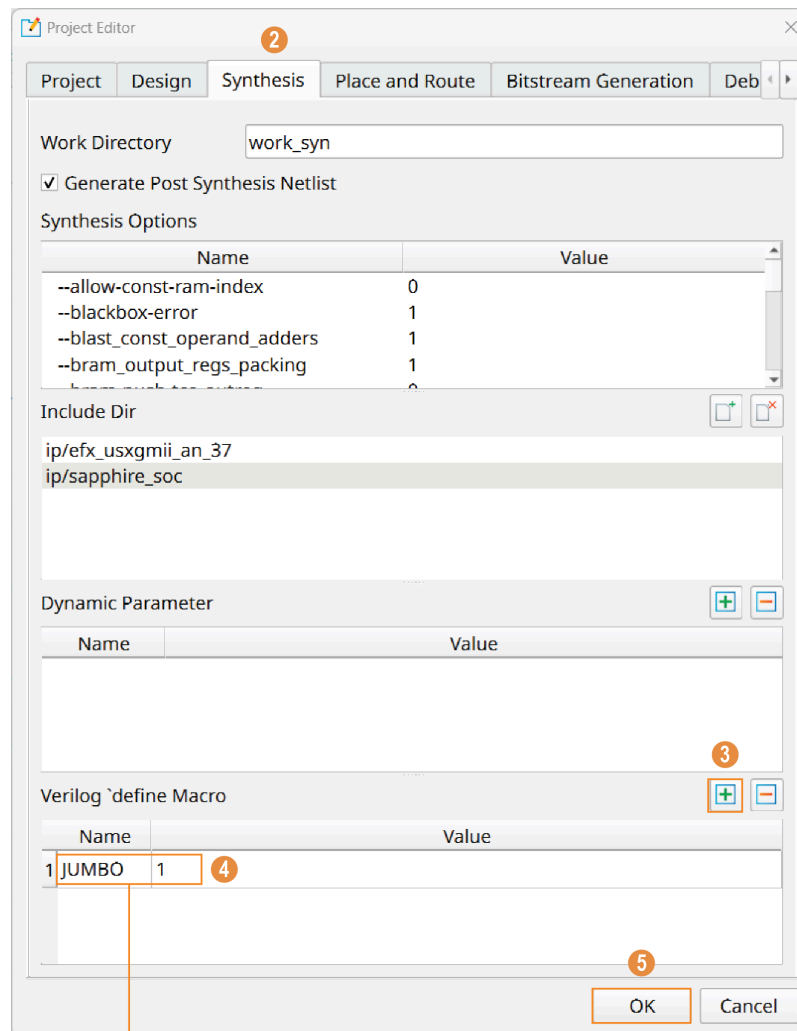
vio0 vio1 la0

Jumbo Packet Test

To run the jumbo packet test, you must edit the example design to include the `JUMBO` macro and compile to generate a new bitstream. This step disconnects the existing packet generator and connects it to the 10G packet generator that supports jumbo packets. Follow these steps to add the `JUMBO` macro:

1. In the Efinity software, choose **File > Edit Projects**.
2. In Project Editor, click **Synthesis**,
3. In Verilog 'define Macro, click (+) to add the `JUMBO` macro.
4. Then, click on the empty box beside the `JUMBO` macro to edit the value. This can be any decimal value (0 - 9) of your choice.
5. Click **OK**.

Figure 21: Adding the `JUMBO` Macro



After adding the `JUMBO` macro, you can edit the value.

Troubleshooting

If the Python script does not receive an Ethernet packet from the FPGA, but is captured by Wireshark, you need to use Wireshark to check for traffic of packets transferred from the FPGA to the host PC. The packets are likely filtered or discarded by the operating system's network stack before reaching the application software. Wireshark operates at a lower level, and it can capture all traffic that goes in and out of the network interface, regardless of destination. The operating system only allows packets with valid addresses to be passed to the respective applications. The following are the solutions:

- If there is a firewall or antivirus block, you need to temporarily disable the network firewall to prevent Windows from filtering or discarding packets before reaching the application.
 1. Type **Control Panel** in the Windows Search. Choose **Control Panel > Windows Defender Firewall > Turn Windows Defender on or off**. See **Figure 23: Opening Windows Defender Firewall** on page 27.
 2. In the Private network settings, select **Turn off Windows Defender Firewall (not recommended)**.
- For incorrect IP address and port binding, you must ensure that the IP address and Port settings are set correctly. The settings in the Debugger and the Python script must match. Refer to **Configuring the Efinity Debugger** on page 20 and **Configuring the Python Script** on page 21.

Figure 23: Opening Windows Defender Firewall

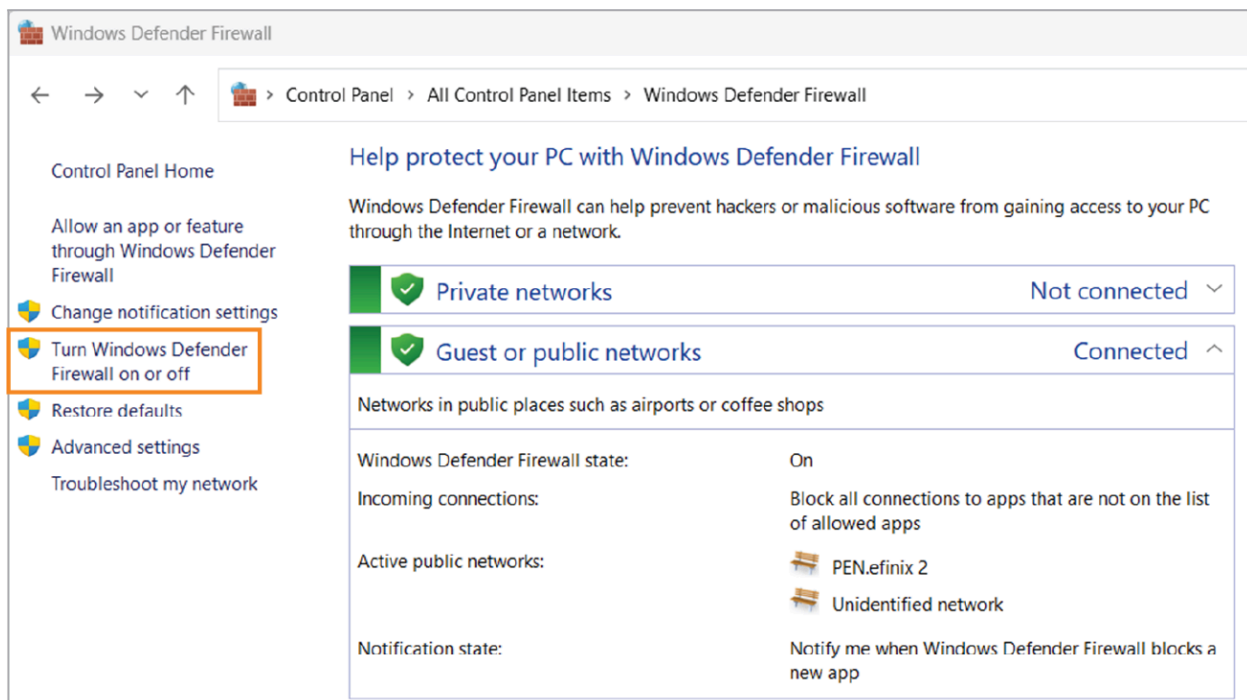
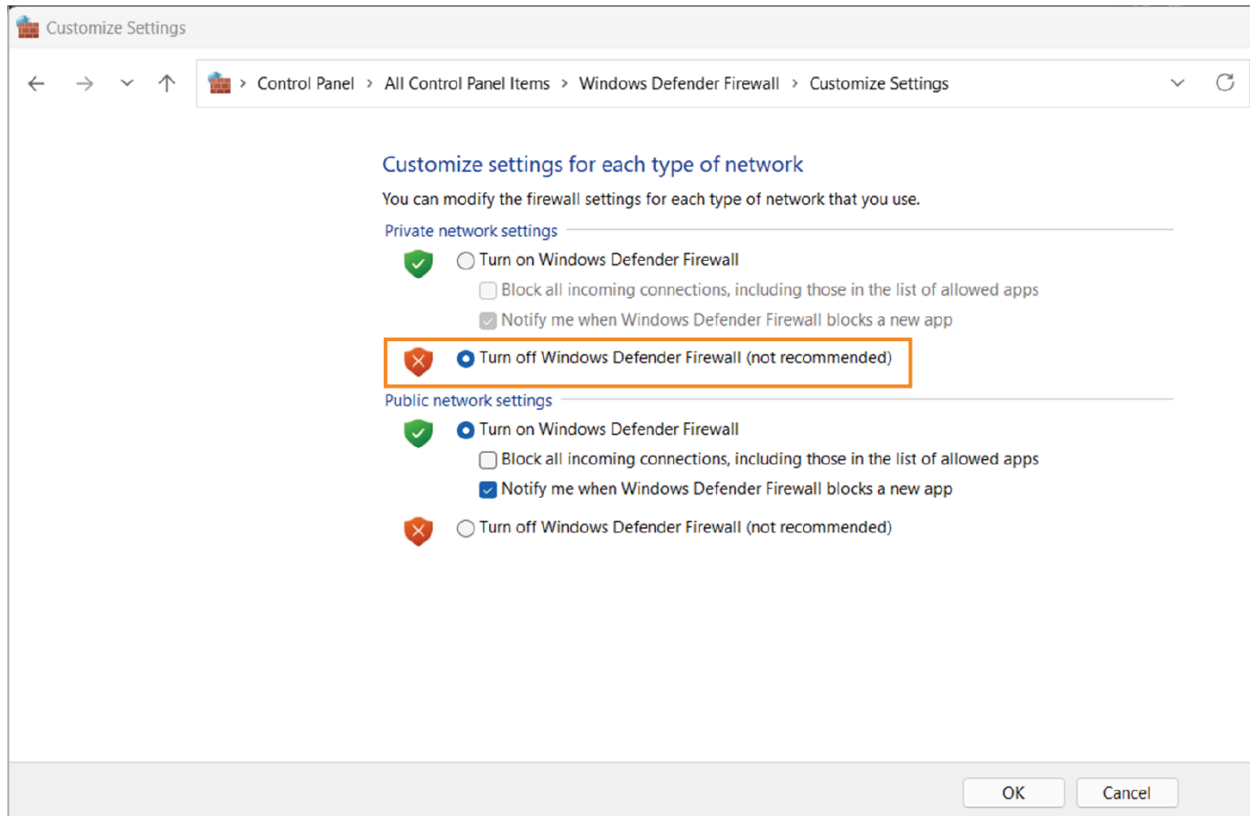


Figure 24: Turn Off the Windows Defender Firewall



Revision History

Table 3: Document Revision History

| Date | Version | Description |
|------------|---------|------------------|
| April 2026 | 1.0 | Initial release. |