



Efinity[®] Transceiver Debugger Tutorial

EFN-XCVR-DEBUG-v1.0
July 2025
www.efinixinc.com



Contents

Introduction.....	3
Transceiver Debugger Overview.....	3
PCIe Read/Write on Linux Tutorial.....	4
Requirements.....	4
Using the PCIe Read/Write Example Design.....	4
Check Status.....	5
Plotting the Eye Diagram.....	5
Performing Internal Register Reads/Writes.....	7
Performing Mass APB Reads/Writes.....	7
Interpreting the Eye Diagram.....	8
BIST Tutorial.....	10
BIST Requirements.....	10
Using the BIST Example Design.....	10
Enable and Run BIST.....	11
Creating Your Own Design.....	14
Working with Multiple APB Bus Masters.....	15
Revision History.....	16

Introduction

The Efinity[®] Transceiver Debugger helps you debug transceiver designs. You can check the PHY and PCIe link status, generate eye diagram plots, and run read/write operations on internal registers. The Transceiver Debugger is available in the Efinity software v2024.2.294.1.19 and higher.

This tutorial describes how to use the Transceiver Debugger with two example designs:

- PCIe read/write on Linux
- BIST

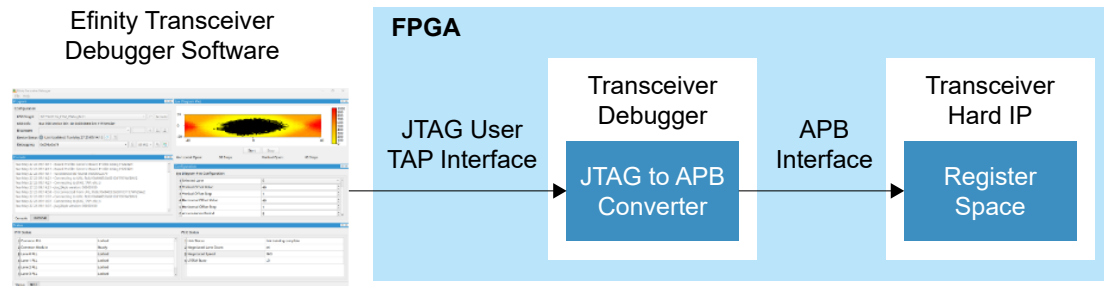
Efnix includes these designs with the Efinity software. To use these designs, you need a Titanium Ti375 N1156 Development Board.

Transceiver Debugger Overview

The Transceiver Debugger uses the JTAG interface to communicate with the register space of the transceiver bank. The tool translates JTAG commands into APB read/write operations to perform various operations in the transceiver bank register space. You can use the Transceiver Debugger to:

- Plot the eye diagram.
- Perform APB read/write operations.
- Check the PCIe link status.
- Configure the transceiver lane in BIST mode.

Figure 1: Efinity Transceiver Debugger Block Diagram



PCIe Read/Write on Linux Tutorial

This design is a modification of the [Ti375 N1156 Development Kit PCIe Reads/Writes on Linux Demonstration Design](#). This tutorial demonstrates the features of the Transceiver Debugger and covers:

- Setting up PCIe link on Titanium Ti375 N1156 Development Board and motherboard.
- Launching and connecting Transceiver Debugger to the board.
- Checking the status of the PCIe link.
- Using terminal read/write to change the link speed.
- Customizing and generating eye plot diagram.
- Using the mass read/write feature to dump the configuration register.

Requirements

To use this example design, you need:

- Test computer running the Linux operating system (the example design was tested on Linux Kernel version 6.8.0.59-generic Ubuntu 22.04.1)
- Computer running Efinity software v2024.2.294.1.19 or higher
- Titanium Ti375 N1156 Development Kit
- USB-C cable

Using the PCIe Read/Write Example Design

This example design is located in the `<Efinity path>/debugger/serdes_debug_tool/examples/Ti375_pcie_devkit_oob` directory. Follow these steps to use the example design:

1. Connect the Titanium Ti375 N1156 Development Board to your computer using a USB cable.
2. Open the Efinity software.
3. Open the Efinity Programmer by choosing **Tools > Open Programmer**.
4. In the Programmer, choose **SPI Active using JTAG Bridge** from the **Programming Mode** drop-down list.
5. Click the **Select Image File** button.
6. Browse to the `<Efinity path>/debugger/serdes_debug_tool/examples/Ti375_pcie_devkit_oob/bitstream/` directory and choose **ti375n1156_oob.hex**.
7. Enter `0x00000000` in the **Starting Flash Address** box.
8. Click the **Start Program** button to download the bitstream file to your board.
9. Set up the hardware as described in "Set Up the Hardware" in the [Titanium Ti375 N1156 Development Kit User Guide](#).
10. Open the Transceiver Debugger (**Tools > Open Transceiver Debugger**).
11. In the Program section, choose **JTAG > USER2** and connect the Transceiver Debugger.

Check Status

To check the PHY status:

1. Click the **Status** tab.

Figure 2: Status Tab

PHY Status		PCIE Status	
1	Common PLL	Locked	1 Link Status
2	Common Module	Ready	link training ...
3	Lane 0 PLL	Locked	2 Negotiated Lane ...
4	Lane 1 PLL	Locked	x4
5	Lane 2 PLL	Locked	3 Negotiated Speed
6	Lane 3 PLL	Locked	16G
			4 LTSSM State
			L0

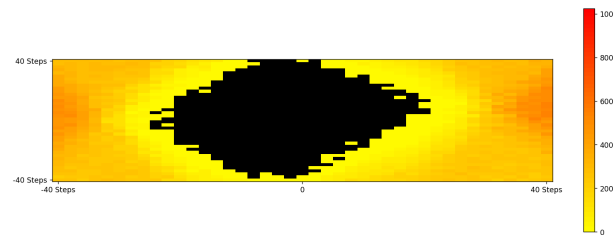
The PHY status shows that the Common Module is ready and all PLLs are locked.

The PCIE status shows that the PCIe Gen4 x4 link is established and the LTSSM state is L0.

Plotting the Eye Diagram

In the **Eye Diagram Plot** section, click **Start** to begin plotting the eye diagram with the default settings. The following figure shows the plot with the default settings.

Figure 3: Eye Diagram Plot with Horizontal Offset Value = 40 and Vertical Offset Value = 40



To adjust the plot, use the settings under **Eye Diagram Plot Configuration** to specify the horizontal offset and vertical offset, and the number of steps. The following figures show examples.

Figure 4: Eye Diagram Plot with Horizontal Offset Value = 63 and Vertical Offset Value = 127

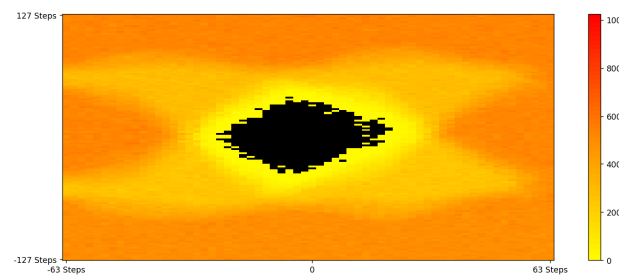


Figure 5: Eye Diagram plot with Horizontal and Vertical Offset Steps = 8

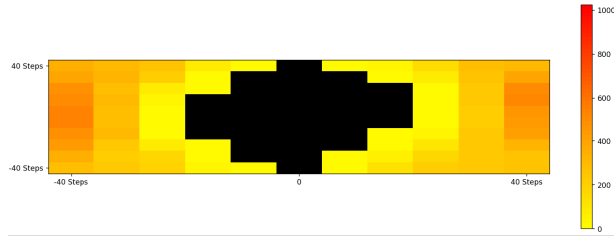
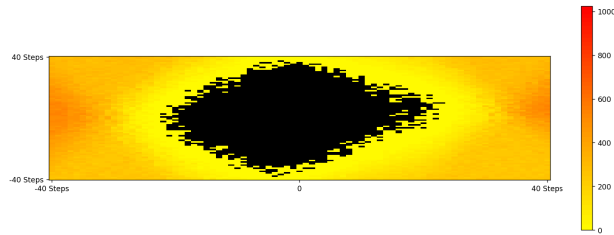


Figure 6: Eye Diagram Plot with Horizontal and Vertical Offset Steps = 1



Configure the accumulation period to adjust the sample count for each position. Efinix recommends using a higher value for more accurate measurement.

Figure 7: Eye Diagram Plot with Accumulation Period = 3

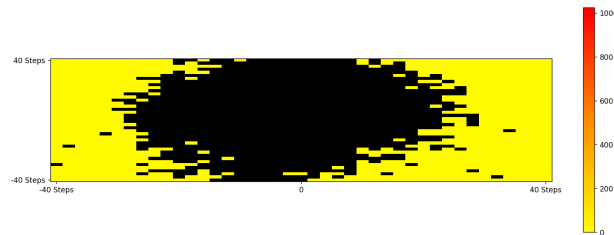
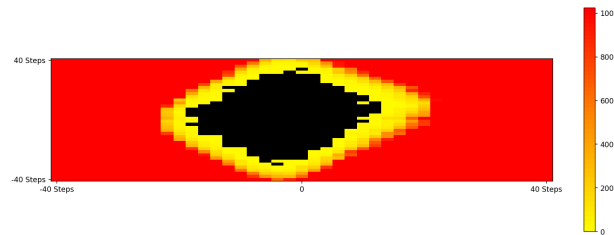


Figure 8: Eye Diagram Plot with Accumulation Period = 15



Performing Internal Register Reads/Writes

This topic explains how to read/write to the registers on the transceiver IP. The following example shows how to use this feature to initiate PCIe link retraining to change the speed.

1. Click the **Terminal** tab.
2. Turn on **Single Command**.
3. In the **Command** box, enter `read:0x100050` and click **Send**. The Transceiver Debugger reads the register.



Note: As described in the [Titanium PCIe Controller Registers User Guide](#), the Linkwidth Control Register @0x50 allows the endpoint device to request link information to retrain for a specific link width or link speed.

Refer to "Configuring Registers with the APB Interface" in the [Titanium PCIe Controller User Guide](#) for information on configuring the upper bits of the local management bus.

4. In the **Command** box, enter `Write:0x100050,0x800000F` and click **Send**. The Transceiver Debugger writes to the register to change the PCIe link speed to Gen1.



Note: The register write operation does not support masked writes. You need to identify and preserve any unchanged values when configuring the write data to prevent unwanted changes.

5. Check the PCIe status and verify the speed change operation.

Performing Mass APB Reads/Writes

Follow these steps to use the mass APB read/write feature to dump the PCIe Capability Register for Physical Function 0 into the example design.

1. Create a text file (**.txt**) in your preferred directory.
2. Enter the following commands in the text file. These commands read all of the PCIe capability register addresses.

```
read:0x000000
read:0x000004
read:0x000008
read:0x00000C
read:0x000010
read:0x000014
read:0x000018
read:0x00001C
read:0x000020
read:0x000024
read:0x000028
read:0x00002C
read:0x000030
read:0x000034
read:0x000038
read:0x00003C
read:0x000040
read:0x000044
read:0x000048
read:0x00004C
read:0x000050
read:0x000054
read:0x000058
read:0x00005C
read:0x000060
read:0x000064
read:0x000068
read:0x00006C
read:0x000070
read:0x000074
read:0x000078
```

```
read:0x00007C
read:0x000080
```

3. Click the **Terminal** tab in the Transceiver Debugger.
4. Turn on the **Mass Read/Write** option.
5. Click the Open Text File button next to **Input File**.
6. Browse to the text file you just created, select it, and click **Open**.
7. If you want to save the output to a file, turn on **Output File** and specify or choose the output file name. Turn off the **Output File** option if you do not want to save output to a text file.
8. Click **Start** to begin the mass APB read/write operations.

Interpreting the Eye Diagram

The Transceiver Debugger generates the eye diagram by accumulating errors at every position within the defined range. Each colored pixel in the eye diagram represents the error counts. Using the color map on the right side of the **Eye Diagram Plot** section, you can determine the error count on a scale from 1 to 1,024 based on the color of each pixel. A black pixel indicates no errors at that position.

Hovering your cursor over any pixel shows the error count in the top right corner of the eye diagram.

Figure 9: Error Count for Eye Diagram



You can use this formula to evaluate the bit error rate (BER) for any individual pixel:

$$BER = \frac{Error}{Samplesize}$$

Calculate the sample size using this formula:

$$Samplesize = 2^{accumperiod+1} \times SamplesperCycle$$

Where:

- accum_period = Accumulation period specified in the Eye Diagram Plot configuration settings.
- SamplesperCycle = 4 for 32-bit SerDes width interfaces, or 2 for any other width.

Use the following equation to calculate the BER at the specified position in **Figure 3: Eye Diagram Plot with Horizontal Offset Value = 40 and Vertical Offset Value = 40** on page 5:



Note: The PCIe Gen4 protocol uses a 32-bit transceiver width.

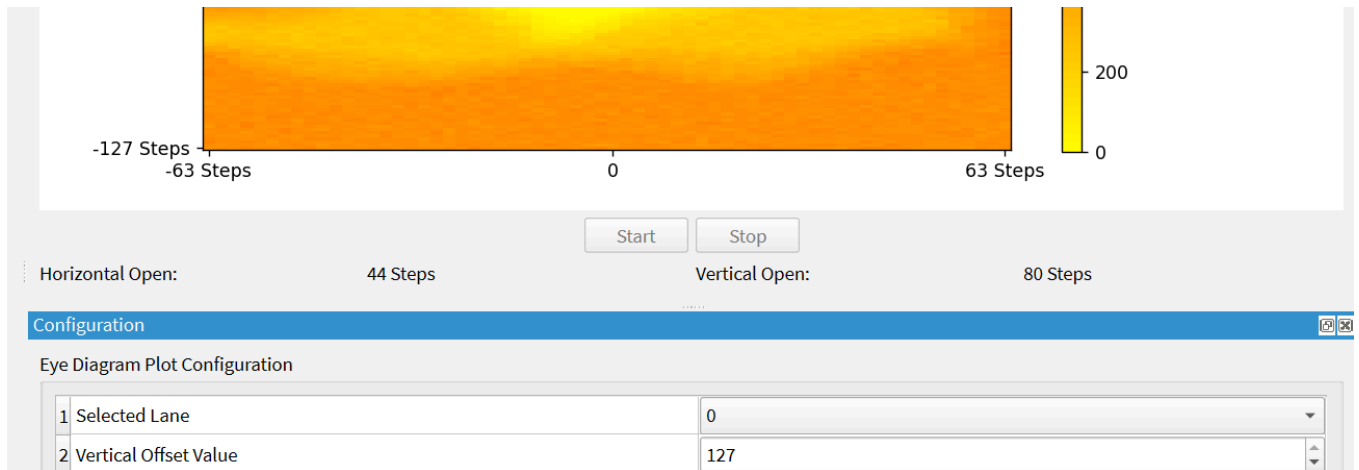
$$BER = \frac{1}{2^{7+1} \times 4} \quad BER = 9.77 \times 10^{-4}$$

Based on the generated eye diagram, the Transceiver Debugger provides a measure of the horizontal opening and vertical opening of the eye. This measurement accounts for the region with 0 sampled errors in the eye diagram.



Note: The Transceiver Debugger only provides measurements if the eye diagram is bounded within the plot canvas.

Figure 10: Eye Opening Measurement of Eye Diagram Plot



Each horizontal step is $1/64$ unit intervals (UI). To convert the previous measurements to UI:

$$\text{Horizontalopen} = \frac{44}{64}$$

$$\text{Horizontalopen} = 0.6875\text{UI}$$

Each vertical step is nominally 3 mV. To convert the previous measurements to mV:

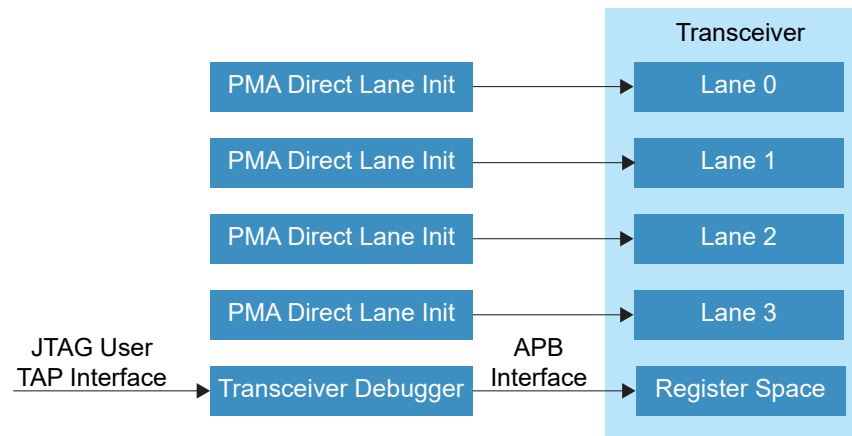
$$\text{Verticalopen} = 80 \times 3$$

$$\text{Verticalopen} = 240\text{mV}$$

BIST Tutorial

This design has four PMA Direct lanes using transceiver quad 1. The PMA lanes follow the power-up sequence describe in "Power-Up Sequence" in the [Titanium PMA Direct User Guide](#). After the device is in A0 power state, you can configure the lane to enter the Built-in Self-Test (BIST) mode.

Figure 11: Block Diagram of BIST Example Design



This tutorial demonstrates the features of the Transceiver Debugger and covers:

- Setting up the example design on the Titanium Ti375 N1156 Development Board.
- Configuring and using BIST on PMA Direct lanes.

BIST Requirements

For this BIST tutorial you need:

- Computer running Efinity® software v2025.1.110 or higher
- Titanium Ti375 N1156 Development Kit
- USB-C cable
- (Optional) SFP + loopback module

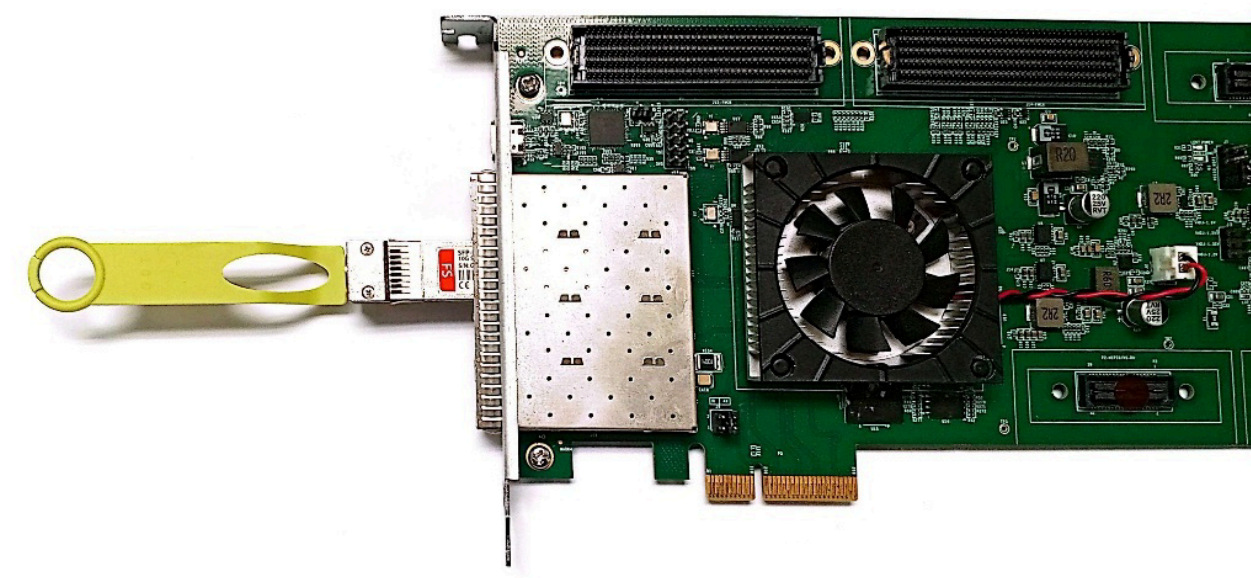
Using the BIST Example Design

The example design files are located in the `<Efinity path>/debugger/serdes_debug_tool/examples/ti375n1156_bist_example/` directory. Follow these steps to set up the example design:

1. Connect the Titanium Ti375 N1156 Development Board to your computer using a USB cable.
2. Open the Efinity software.
3. Open the Efinity Programmer by choosing **Tools > Open Programmer**.
4. In the Programmer, choose **SPI Active using JTAG Bridge** from the **Programming Mode** drop-down list.
5. Click the **Select Image File** button.
6. Browse to the `<Efinity path>/debugger/serdes_debug_tool/examples/ti375n1156_bist_example/bitstream/` directory and choose `ti375n1156_devkit_PMA Direct_idle.hex`.

7. Enter 0x00000000 in the **Starting Flash Address** box.
8. Click the **Start Program** button to download the bitstream file to your board.
9. Press SW2 on the Titanium Ti375 N1156 Development Board to reset the FPGA.
10. (Optional) Insert the SFP+ Loopback Module into SFP+ Connector 2 (J12), as shown in **Figure 12: SFP+ Loopback Module Plugged into SFP+ Connector 2 (J12)** on page 11.
11. Open the Transceiver Debugger by choosing **Tools > Open Transceiver Debugger**.
12. Select **JTAG > USER2** and connect the Transceiver Debugger.

Figure 12: SFP+ Loopback Module Plugged into SFP+ Connector 2 (J12)

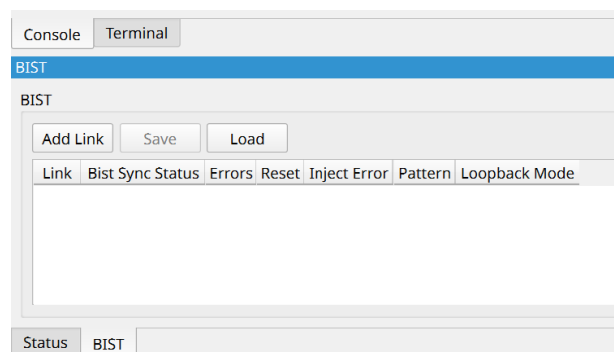


Enable and Run BIST

In the Transceiver Debugger, perform these steps to enable and run BIST:

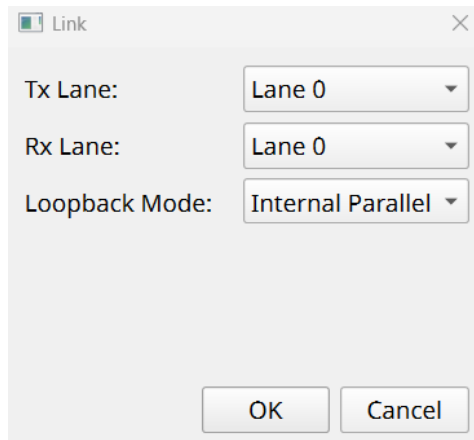
1. Click the **BIST** tab.

Figure 13: BIST Tab



- Click **Add Link** to add a lane and configure it in BIST mode.

Figure 14: BIST Configuration Window



- Create three links with the following configuration:

TX Lane	RX Lane	Loopback Mode
Lane 0	Lane 0	Internal Parallel
Lane 1	Lane 1	Internal Serial
Lane 2	Lane 2	External Serial

- The Transceiver Debugger shows the BIST sync status for the configured lanes as shown in the following table:

BIST Status	Description
A020	BIST inactive
A022	BIST running
A02A, A02E	Bit error detected



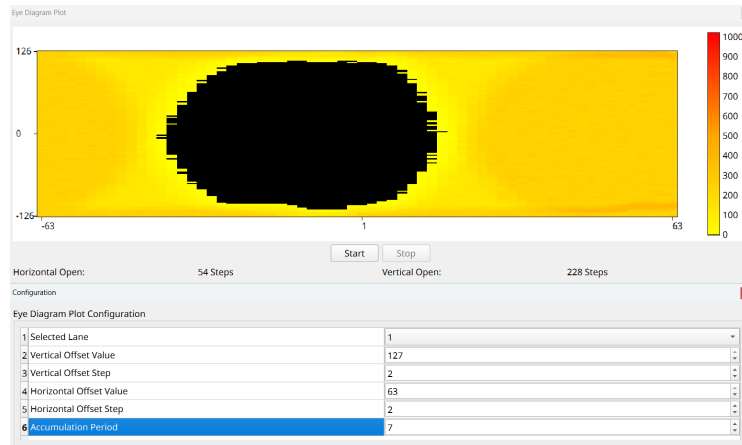
Note: If you do not use the optional SFP+ module for Lane 2, the **Bist Sync Status** column shows **A020**, indicating that BIST is inactive because the loopback path is unavailable.

- Click **Inject Error** to increment the error count.
- Click **Reset** to reset the error count.
- Plot the eye diagram for Lane 1.



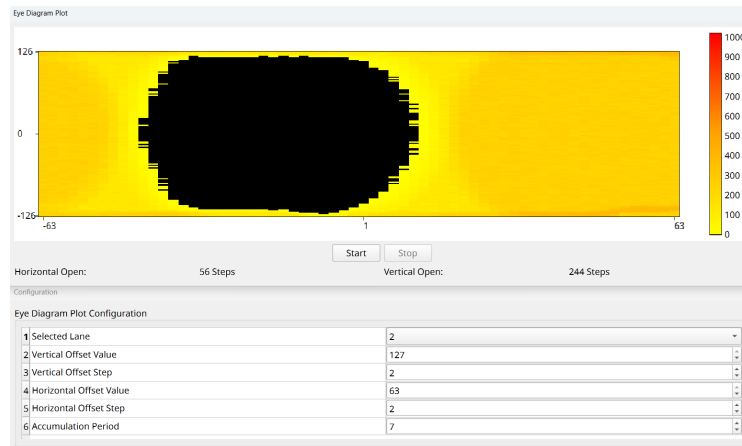
Note: The Transceiver Debugger can only plot the eye diagram while data is actively being transferred in the serial analog RX data path.

Figure 15: Eye Diagram Plot of Internal Serial Loopback BIST



- (Optional) If you are using the SFP+ loopback module, plot the eye diagram for Lane 2.

Figure 16: Eye Diagram Plot of External Serial Loopback BIST with SFP+ Loopback Module



Creating Your Own Design

The following steps outline the process for creating your own design.

1. Include the Transceiver Debugger design files into your project. The files are in the `<Efinity path>/debugger/serdes_debug_tool/rtl/` directory.



Note: The Efinity software only supports Transceiver Debugger cross-version compatibility when the Efinity software version checker recognizes and accepts the Transceiver Debugger version ID specified in the `version.vh` file.

2. Open the Interface Designer.
3. Create a new JTAG User Tap instance for the Transceiver Debugger. Choose a **JTAG Resource** (JTAG_USER1 - 4) that is not used for any other purpose; the Transceiver Debugger cannot share the JTAG resource with other applications.
4. Complete your design, compile, and program the Titanium Ti375 N1156 Development Board with your design's bitstream.
5. Open the Transceiver Debugger. In the **Program** section of the Transceiver Debugger, choose the JTAG User Tap resource (USER1 - 4) that matches what you chose in the Interface Designer. The Transceiver Debugger is the APB master for the transceiver.
6. Connect the APB port to the APB port of the targeted transceiver bank.

The following code shows an example Transceiver Debugger instantiation:

```
//Odd parity generation for APB bus
genvar i;
generate
    for (i=0; i<4; i=i+1)
    begin
        assign q2_USER_APB_PWDATA_PAR[i] = ~(^ q2_USER_APB_PWDATA[i*8+:8]);
    end
endgenerate
assign q2_USER_APB_PSTRB = 4'hf;
assign q2_USER_APB_PSTRB_PAR = ~(^q2_USER_APB_PSTRB);

transceiverDebugger uTransceiverDebugger(
    .CAPTURE (jtag_inst2_CAPTURE),
    .DRCK (jtag_inst2_DRCK),
    .RESET (jtag_inst2_RESET),
    .RUNTEST (jtag_inst2_RUNTEST),
    .SEL (jtag_inst2_SEL),
    .SHIFT (jtag_inst2_SHIFT),
    .UPDATE (jtag_inst2_UPDATE),
    .TCK (jtag_inst2_TCK),
    .TDI (jtag_inst2_TDI),
    .TMS (jtag_inst2_TMS),
    .TDO (jtag_inst2_TDO),

    .clk (apbclk),
    .rst (user_rst),

    .cdb_penable (q2_USER_APB_PENABLE),
    .cdb_pwrite (q2_USER_APB_PWRITE),
    .cdb_paddr (q2_USER_APB_PADDR),
    .cdb_pwdata (q2_USER_APB_PWDATA),
    .cdb_prdata (q2_USER_APB_PRDATA),
    .cdb_pready (q2_USER_APB_PREADY),
    .cdb_psel (q2_USER_APB_PSEL)
```



Note: Efinix recommends that you include additional logic for odd parity generation to access the PCIe Controller Register Space.

Working with Multiple APB Bus Masters

If you are working with multiple APB master interfaces, Efinix recommends using the APB Interconnect IP core, which is available in the IP Manager. Refer to the [APB Interconnect Core User Guide](#) for more information on the IP core. The following code shows an example for multiple APB masters.

```

genvar i;
generate
  for (i=0; i<4; i=i+1)
    begin
      assign q0_USER_APB_PWDATA_PAR[i] = ~(^ q0_USER_APB_PWDATA[i*8+8]);
    end
endgenerate
assign q0_USER_APB_PSTRB = 4'hf;
assign q0_USER_APB_PSTRB_PAR = ~(^q0_USER_APB_PSTRB);

localparam APB_MASTER_CNT = 2;
localparam APB_DATA_WIDTH = 32;
localparam APB_ADDR_WIDTH = 24;

wire [APB_MASTER_CNT-1:0]          s_apb_psel_i;
wire [APB_MASTER_CNT-1:0]          s_apb_penable_i;
wire [APB_MASTER_CNT-1:0]          s_apb_pwrite_i;
wire [APB_MASTER_CNT*APB_ADDR_WIDTH-1:0] s_apb_paddr_i;
wire [APB_MASTER_CNT*APB_DATA_WIDTH-1:0] s_apb_pwdata_i;
wire [APB_MASTER_CNT-1:0]          s_apb_pready_o;
wire [APB_MASTER_CNT-1:0]          s_apb_pslverr_o;
wire [APB_MASTER_CNT*APB_DATA_WIDTH-1:0] s_apb_prdata_o;

assign s_apb_psel_i = {q0_USER_XCVRDBG_APB_PSEL ,q0_USER_CFG_APB_PSEL };
assign s_apb_penable_i = {q0_USER_XCVRDBG_APB_PENABLE,q0_USER_CFG_APB_PENABLE};
assign s_apb_pwrite_i = {q0_USER_XCVRDBG_APB_PWRITE ,q0_USER_CFG_APB_PWRITE };
assign s_apb_paddr_i = {q0_USER_XCVRDBG_APB_PADDR ,q0_USER_CFG_APB_PADDR };
assign s_apb_pwdata_i = {q0_USER_XCVRDBG_APB_PWDATA ,q0_USER_CFG_APB_PWDATA };
assign {q0_USER_XCVRDBG_APB_PREADY ,q0_USER_CFG_APB_PREADY } = s_apb_pready_o;
assign {q0_USER_XCVRDBG_APB_PSLVERR,q0_USER_CFG_APB_PSLVERR} = s_apb_pslverr_o;
assign {q0_USER_XCVRDBG_APB_PRDATA ,q0_USER_CFG_APB_PRDATA } = s_apb_prdata_o;

u0_apb_interconnect u_u0_apb_interconnect
(
  .clk ( q0_AXI_CLK ),
  .rst_n ( user_rst ),
  .apb_psel_o ( ^q0_USER_APB_PSEL ),
  .apb_penable_o ( ^q0_USER_APB_PENABLE ),
  .apb_pwrite_o ( q0_USER_APB_PWRITE ),
  .apb_paddr_o ( q0_USER_APB_PADDR ),
  .apb_pwdata_o ( q0_USER_APB_PWDATA ),
  .apb_pready_i ( q0_USER_APB_PREADY ),
  .apb_pslverr_i ( q0_USER_APB_PSLVERR ),
  .apb_prdata_i ( q0_USER_APB_PRDATA ),

  .s_apb_psel_i ( s_apb_psel_i ),
  .s_apb_penable_i ( s_apb_penable_i ),
  .s_apb_pwrite_i ( s_apb_pwrite_i ),
  .s_apb_paddr_i ( s_apb_paddr_i ),
  .s_apb_pwdata_i ( s_apb_pwdata_i ),
  .s_apb_pready_o ( s_apb_pready_o ),
  .s_apb_pslverr_o ( s_apb_pslverr_o ),
  .s_apb_prdata_o ( s_apb_prdata_o )
);

transceiverDebugger uTransceiverDebugger(
  .CAPTURE (jtag_inst2_CAPTURE),
  .DRCK (jtag_inst2_DRCK),
  .RESET (jtag_inst2_RESET),
  .RUNTEST (jtag_inst2_RUNTEST),
  .SEL (jtag_inst2_SEL),
  .SHIFT (jtag_inst2_SHIFT),
  .UPDATE (jtag_inst2_UPDATE),
  .TCK (jtag_inst2_TCK),
  .TDI (jtag_inst2_TDI),
  .TMS (jtag_inst2_TMS),
  .TDO (jtag_inst2_TDO),

  .clk (apbclk),
  .rst (user_rst),

```

```

.cdb_penable (q0_USER_XCVRDBG_APB_PENABLE),
.cdb_pwrite (q0_USER_XCVRDBG_APB_PWRITE),
.cdb_paddr (q0_USER_XCVRDBG_APB_PADDR),
.cdb_pwdata (q0_USER_XCVRDBG_APB_PWDATA),
.cdb_prdata (q0_USER_XCVRDBG_APB_PRDATA),
.cdb_pready (q0_USER_XCVRDBG_APB_PREADY),
.cdb_psel (q0_USER_XCVRDBG_APB_PSEL)
);

```

Revision History

Table 1: Revision History

Date	Version	Description
July 2025	1.0	Initial Release.