Xilinx Answer 68134
UltraScale and UltraScale+ FPGA Gen3 Integrated Block for PCI Express - Integrated Debugging Features and Usage Guide

Important Note: This downloadable PDF of an Answer Record is provided to enhance its usability and readability. It is important to note that Answer Records are Web-based content that are frequently updated as new information becomes available. You are reminded to visit the Xilinx Technical Support Website and review (Xilinx Answer 68134) for the latest version of this Answer.

Introduction

Prior to Vivado 2016.3 release, a manual insertion of ILA core was required to probe signals and find out the LTSSM transitions during the link training process. To do an eye scan of a PCI Express link, users had to opt for a manual approach such as the reference design provided in XAPP1198. Another major issue in debugging PCI express issues in UltraScale devices was interpreting the scrambled data on a PIPE interface. All of these difficulties have been addressed in the Vivado 2016.3 release of UltraScale and UltraScale+ PCI Express cores. The core configuration now comes with the following three integrated debug options.

- Enable JTAG Debugger
- Enable In system IBERT
- Enable Descrambler of Gen3 Mode

This document describes all of these debug features in detail with screenshots to make it easier for users to understand its implementation and usage.

JTAG Debugger

The JTAG Debugger provides users with a visual representation of the ltssm state transitions during the link training, PHY reset FSM transitions and the receiver detect status on each lane of a PCI Express link. PHY reset FSM is an internal state machine that is used by the PCIe core.

Figure 1 shows the architecture of the JTAG debugger implemented in the PCIe core when this option is enabled in the core configuration GUI. The ltssm transitions, receiver detect status and PHY reset FSM status are stored in the block rams. The stored data is read through AXI JTAG Debugger via Tcl interface. A tcl script, *test_rd.tcl*, is provided with the core generation which is executed in the Vivado Tcl console. The script reads the data stored in the memory and outputs the following set of files:

- *pcie_debug_info_trc.dat*
- *pcie_debug_ltssm_trc.dat*
- *pcie_debug_rst_trc.dat*
- *pcie_debug_static_info.dat*
- *rxdet.dat*

The following Tcl scripts are generated along with the generation of the PCI Express core.

- *draw_ltssm.tcl*
- *draw_reset.tcl*

- *draw_rxdet.tcl*
- *test_rd.tcl*

These scripts take in the *.dat files and plot graphs for LTSSM transitions, PHY reset FSM transitions and receiver detect status.

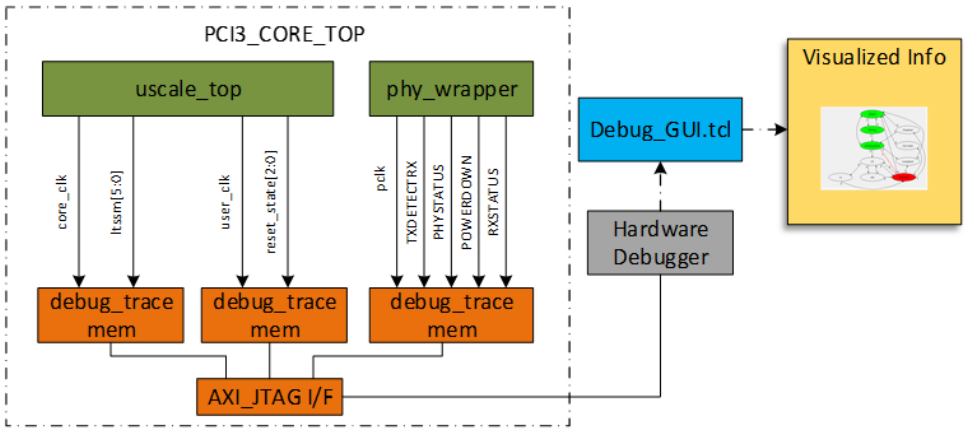

Figure 1 - JTAG Debugger Architecture

The screenshots below show the step-by-step instruction for using JTAG Debugger in the PCI Express example design on a KCU105 development board.

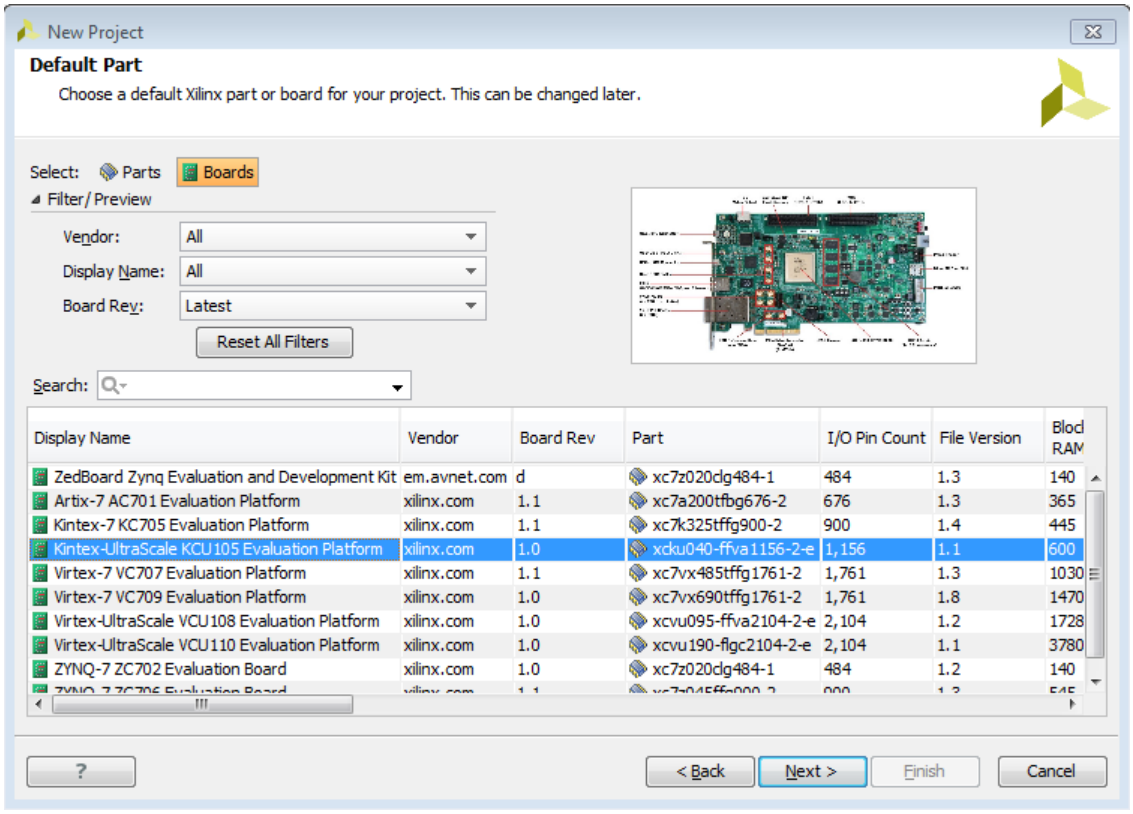

Figure 2 – Create a Vivado project targeting KCU105 Development Board

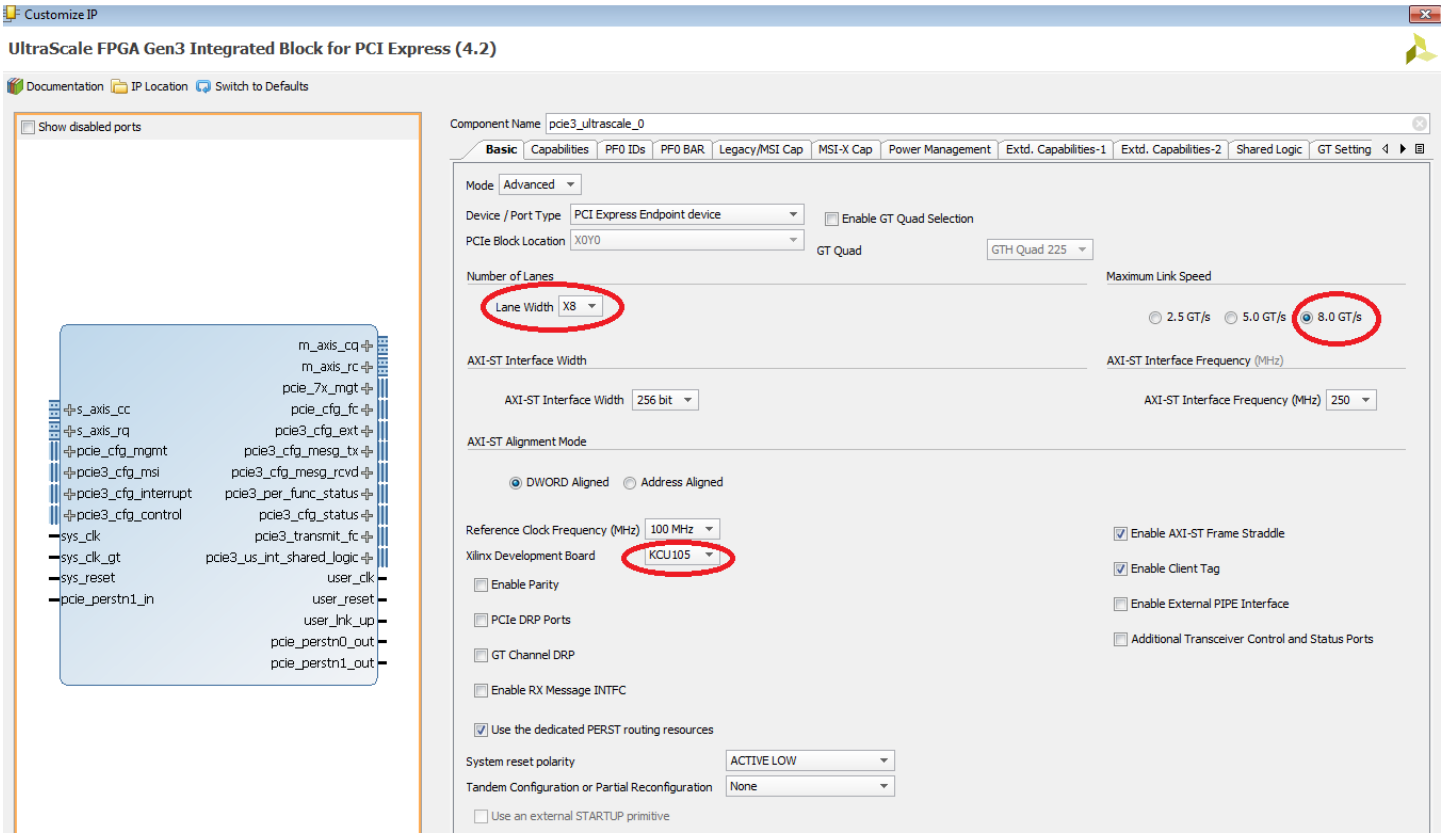

Figure 3 – Select Gen3 link speed, x8 lane width and KCU105 board. Only Gen3 is currently supported; there is no restriction in lane width

Note: *Gen1 and Gen2 link speed support will be enabled in a future release.*

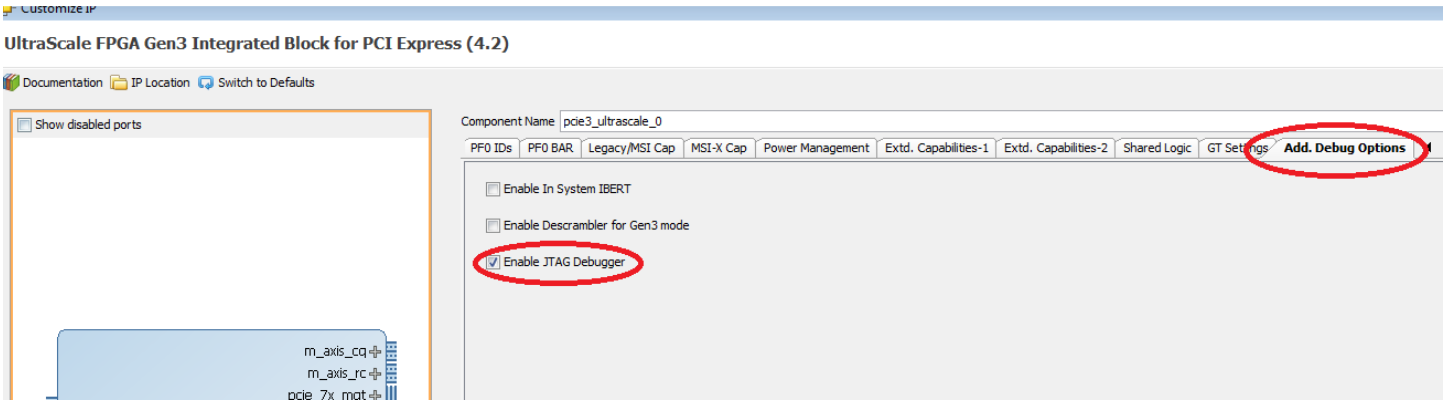

Figure 4 – Enable JTAG Debugger

Note: *All three options can be selected in the same design. The options are selected one at a time in this document for illustration purpose only.*

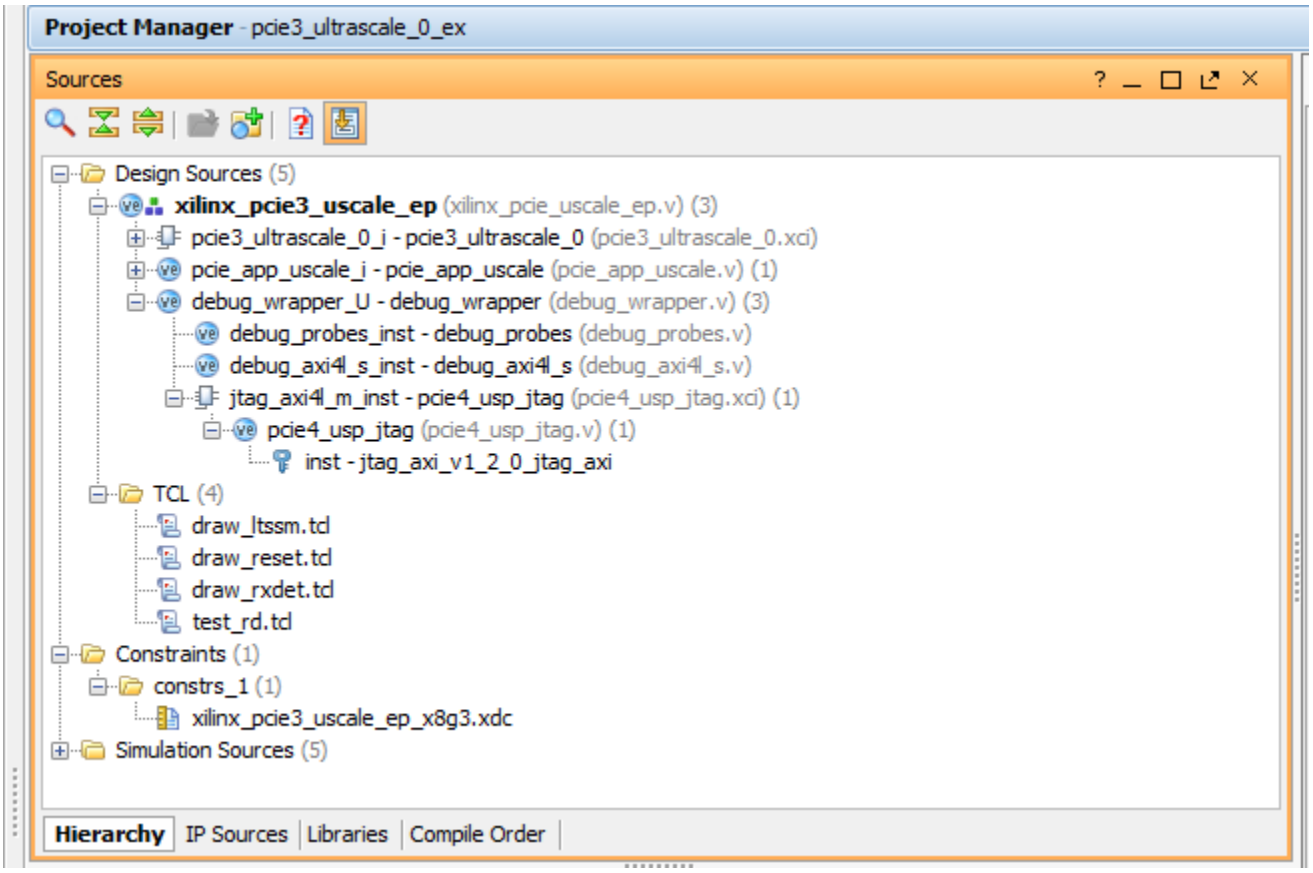

Figure 5 – Generate the core and open the example design. The source hierarchy should show debug_wrapper and axi_jtag instantiation and list of all generated Tcl files.

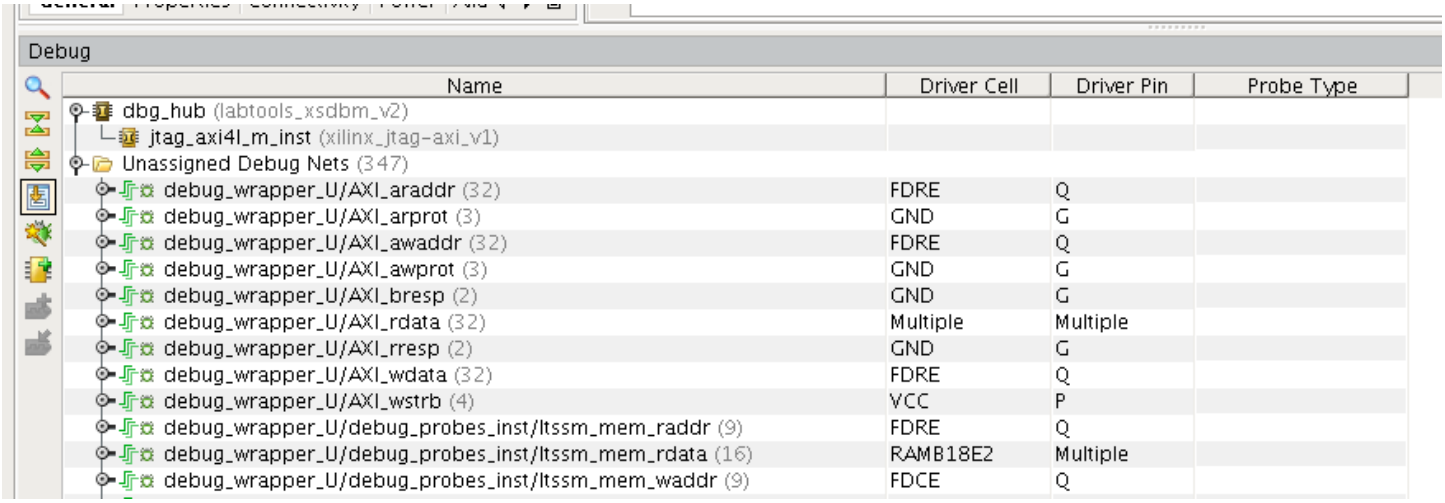

Figure 6 - The debug window view after synthesis

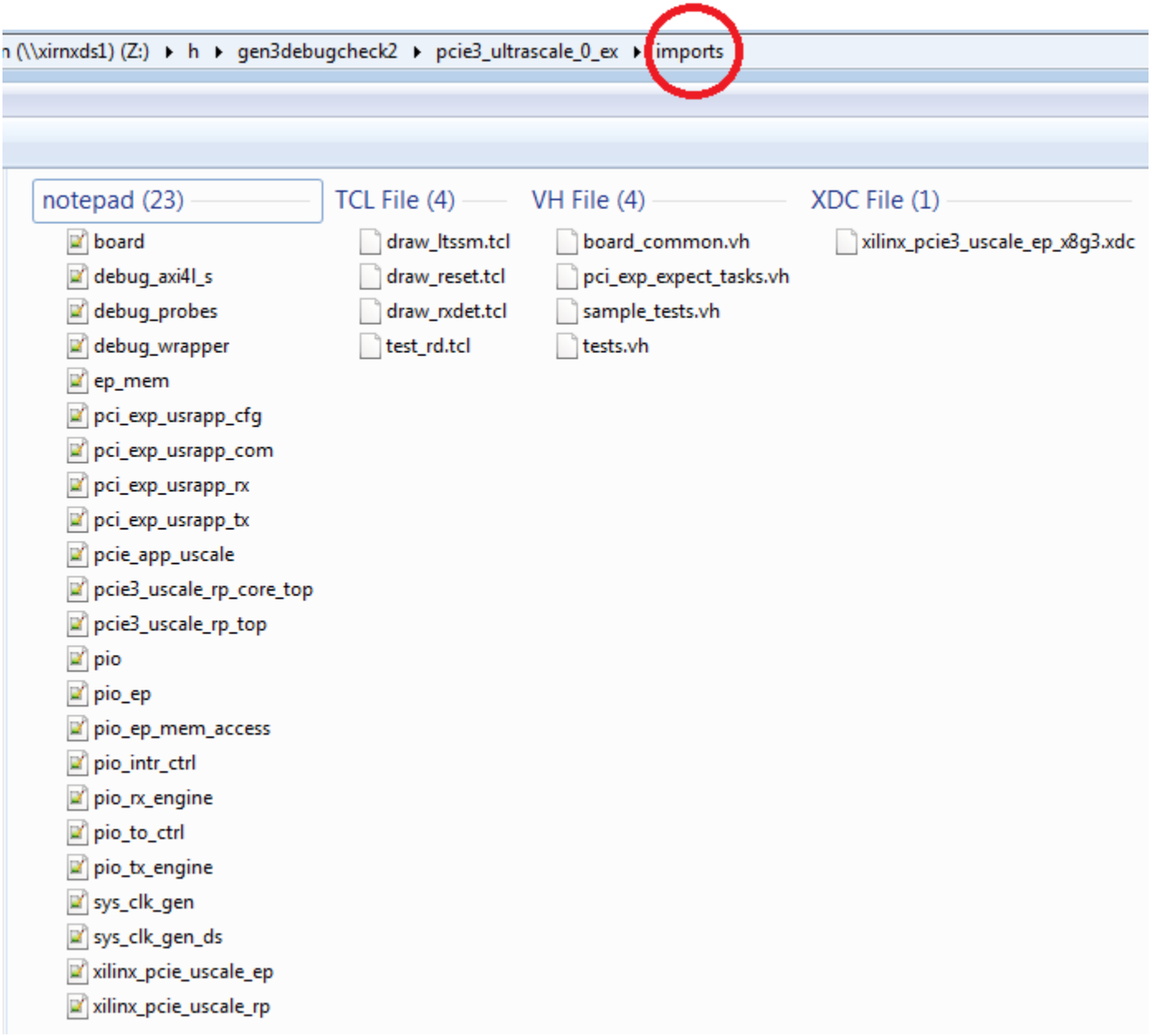

Figure 7 – The Tcl files are located in the 'imports' folder inside the example design project directory

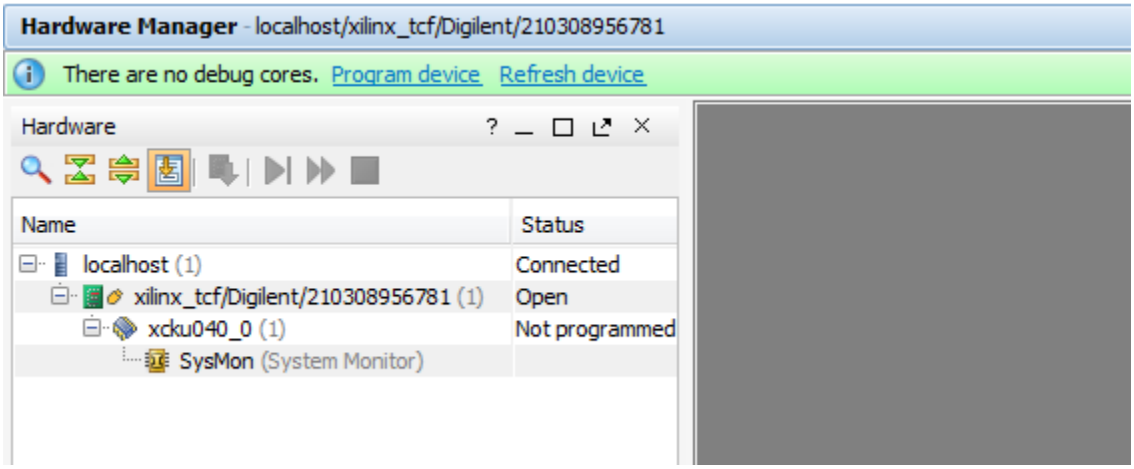

Figure 8 - Open Hardware Manager and connect to KCU105 board

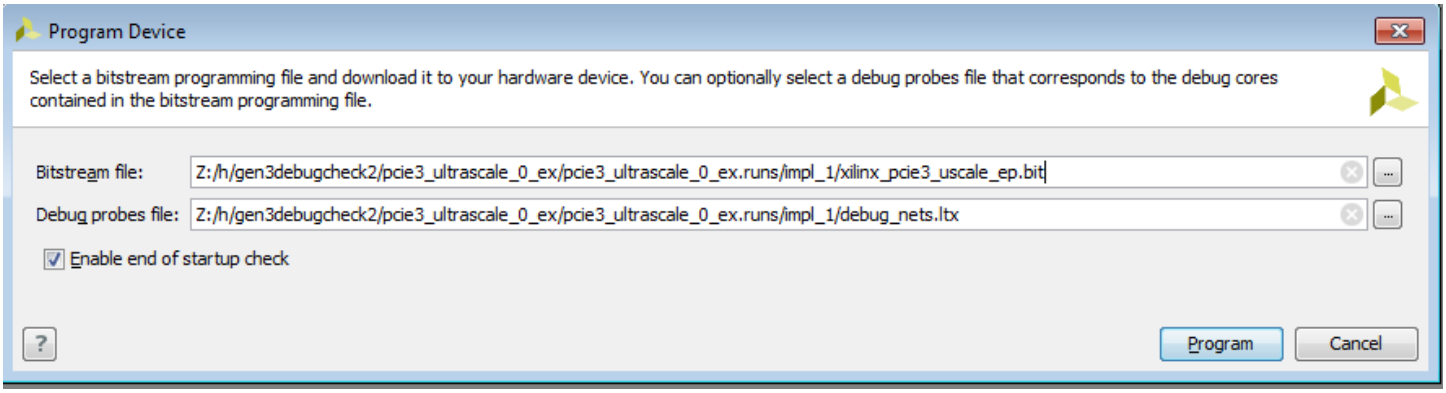

Figure 9 - Program the device with the generated bit and ltx files

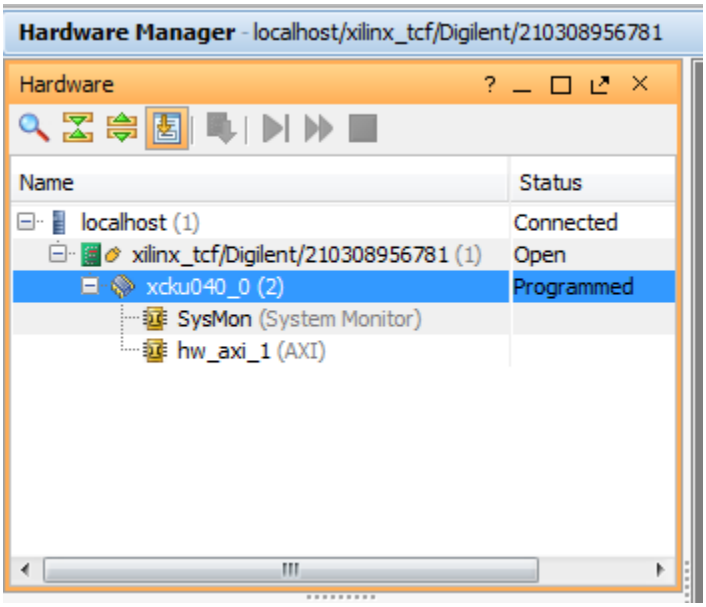

Figure 10 – After the bit file has been programmed, the Hardware window should list hw_axi_1

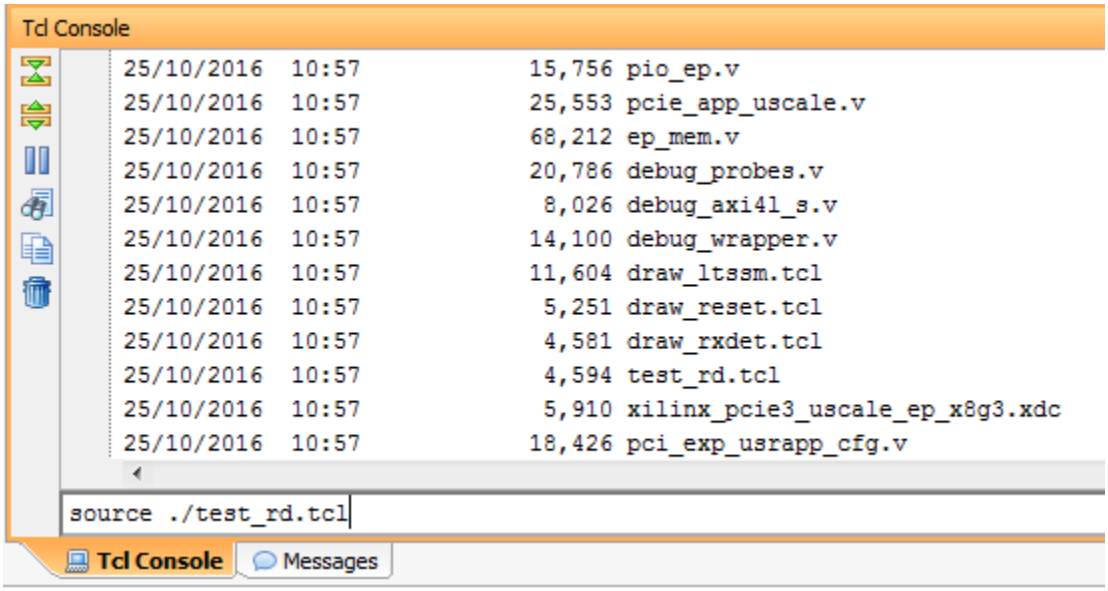

Figure 11 - Source test_rd.tcl in Vivado Tcl Console, located in 'imports' folder

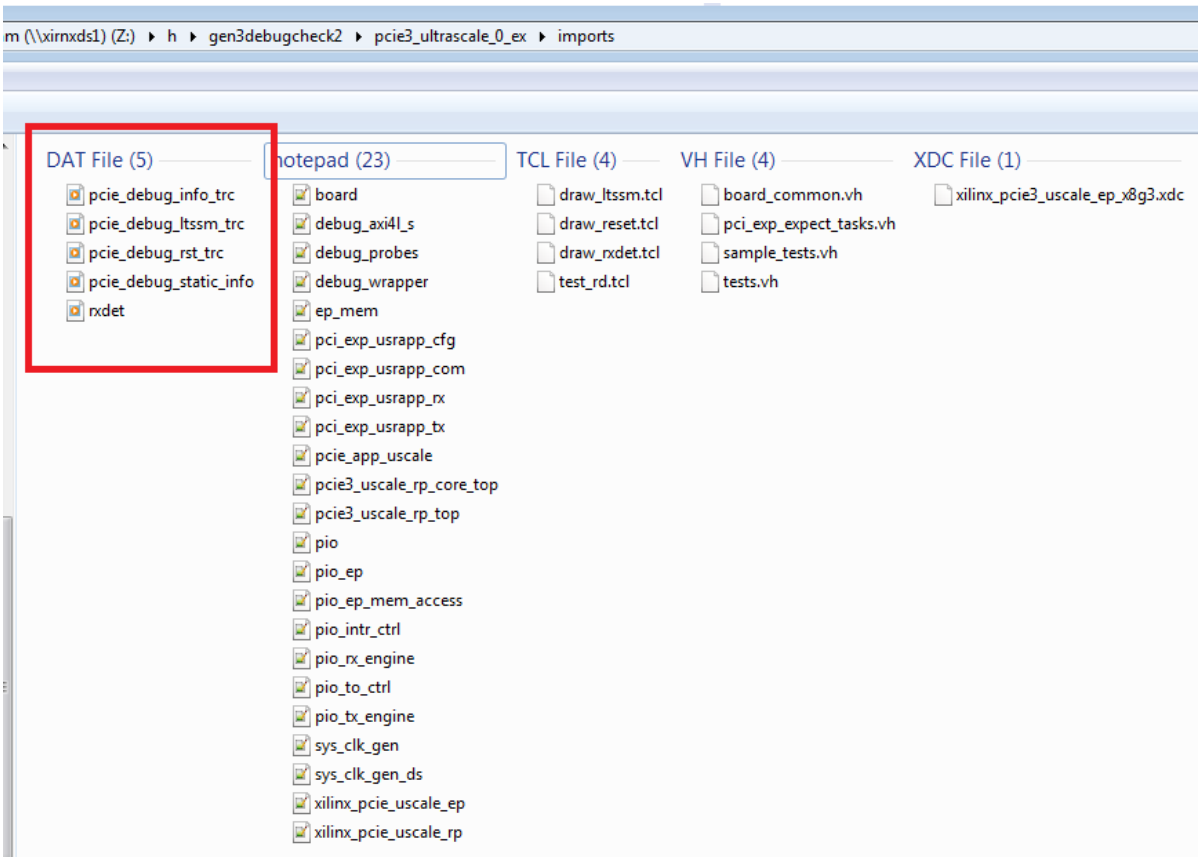

Figure 12 - *dat files are generated after running test_rd.tcl

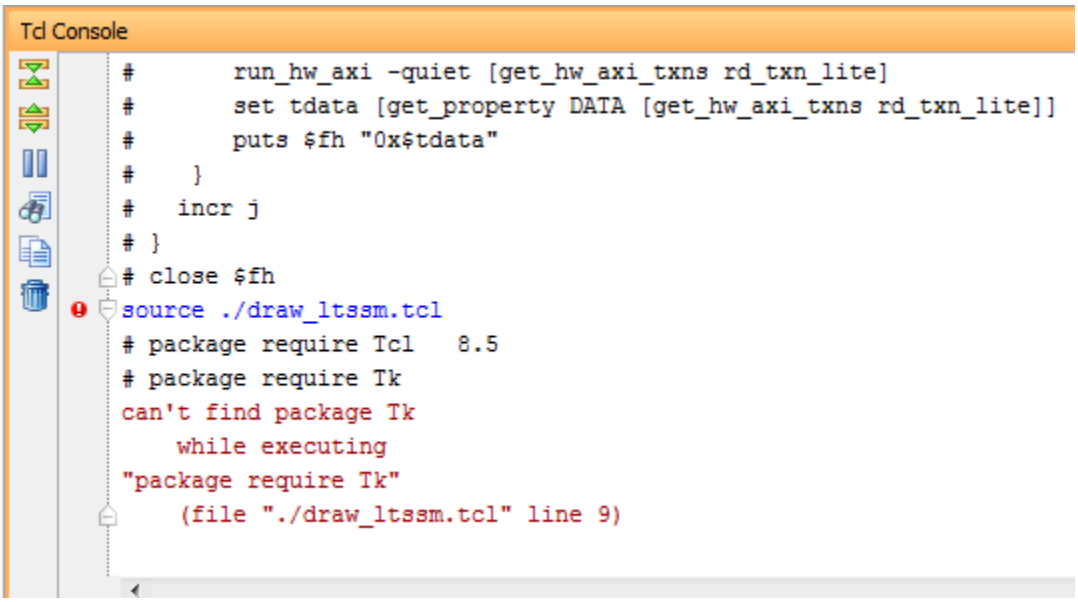

Figure 13 – Error if draw_ltssm.tcl, draw_reset.tcl and draw_rxdet.tcl are sourced in the Vivado Tcl Console

Note: *draw_ltssm.tcl, draw_reset.tcl and draw_rxdet.tcl are separate from Vivado.*

After the *data files have been generated, double click on the Tcl files in the 'imports' directory to generate the respective graphs. To generate the graph, users should make sure Tcl/Tk 8.5 package have been installed. Go to the links below to download the Tcl/Tk packages for the platform that is being used.

- http://www.activestate.com/activetcl/downloads
- https://www.tcl.tk/software/tcltk/download.html

DOWNLOAD TCL: OTHER PLATFORMS AND VERSIONS

Version	Windows (x86)	Windows (64-bit, x64)	Mac OS X (10.5+, x86_64/x86)	Linux (x86)	Linux (x86_64)
8.6.4.1	Windows Installer (EXE)	Windows Installer (EXE)	Mac Disk Image (DMG)	AS Package	AS Package
8.5.18.0	Windows Installer (EXE)	Windows Installer (EXE)	Mac Disk Image (DMG)	AS Package	AS Package

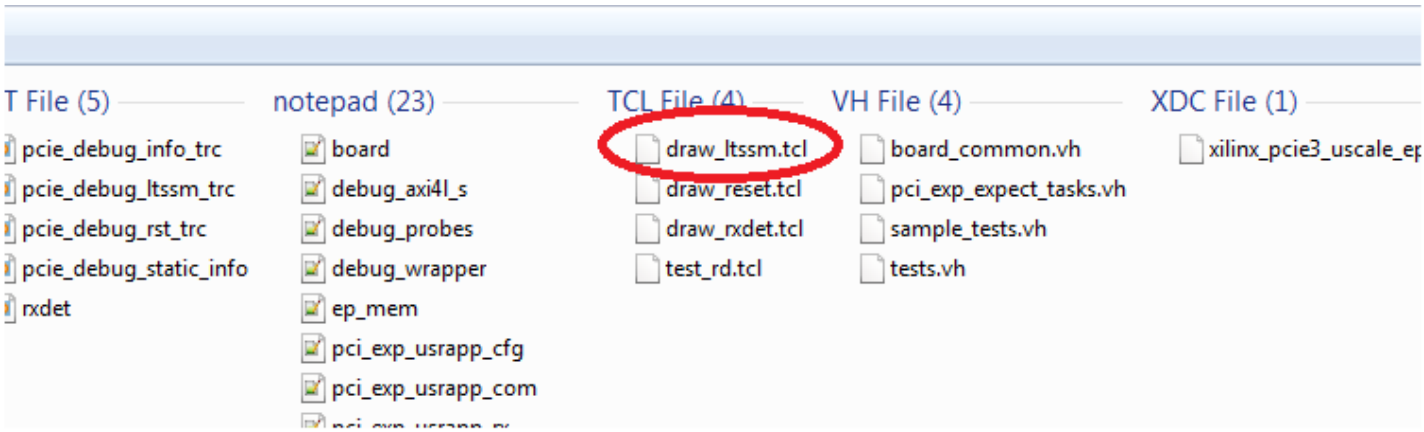

Figure 14 –Double click on draw_ltssm.tcl to generate LTSSM graph

Note: Use the following commands in Linux

- wish draw_ltssm.tcl &
- wish draw_reset.tcl &
- wish draw_rxdet.tcl &

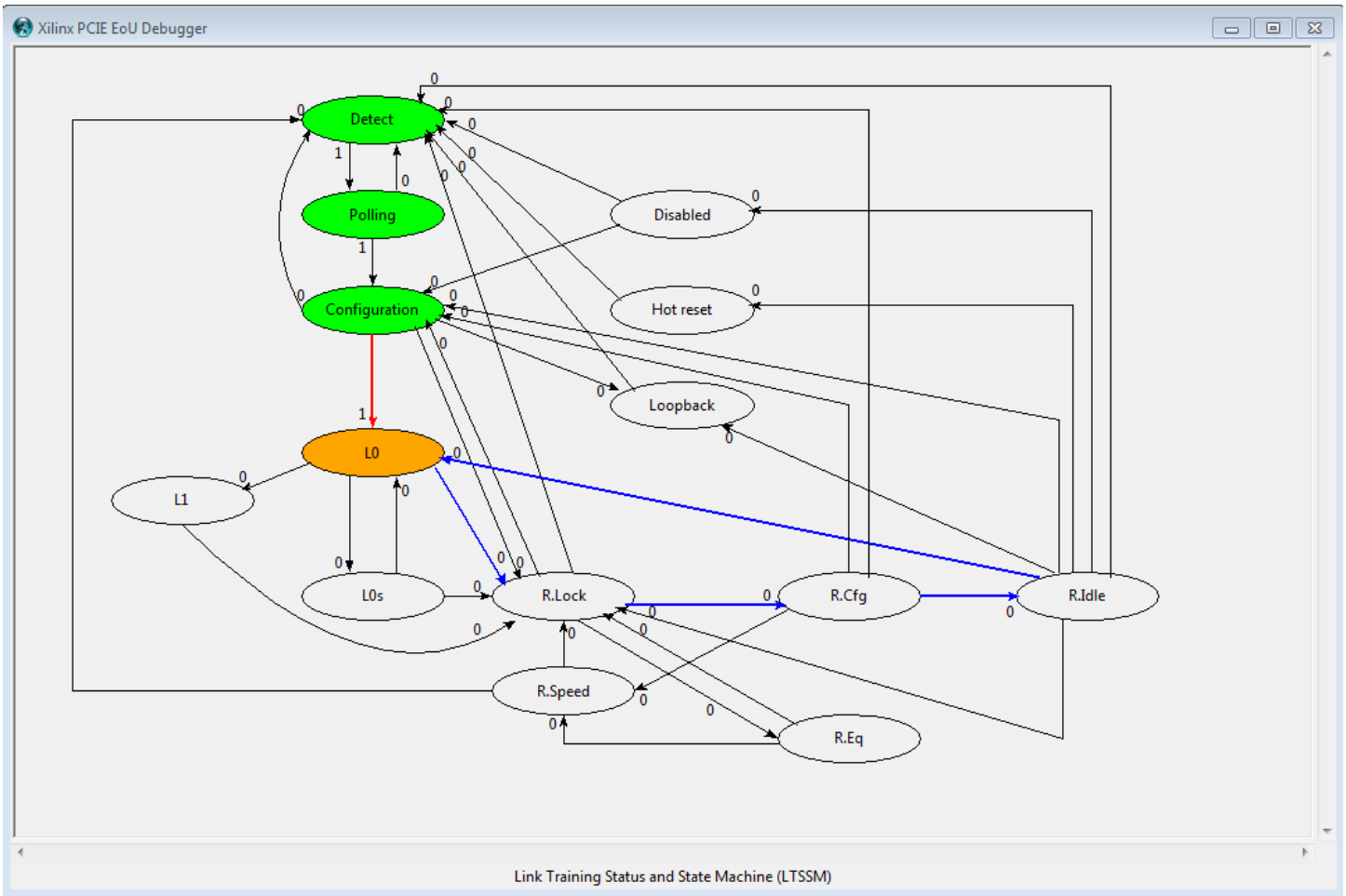

Figure 15 – Generated LTSSM graph

Note on generated LTSSM graph:

- *The orange color indicates the last ltssm state of the capture window of the ltssm signal.*
- *The red arrow indicates the last transition of ltssm state in the capture window.*
- *The green color indicates the states that ltssm transitioned to during the capture window.*
- *The number on the arrow between the ltssm states shows the number of times the transition took place between the two ltssm states in the direction pointed to by the arrow.*
- *The blue color in the arrows in Figure 15 can be ignored. It should be black; it will be corrected in a future release of the IP.*

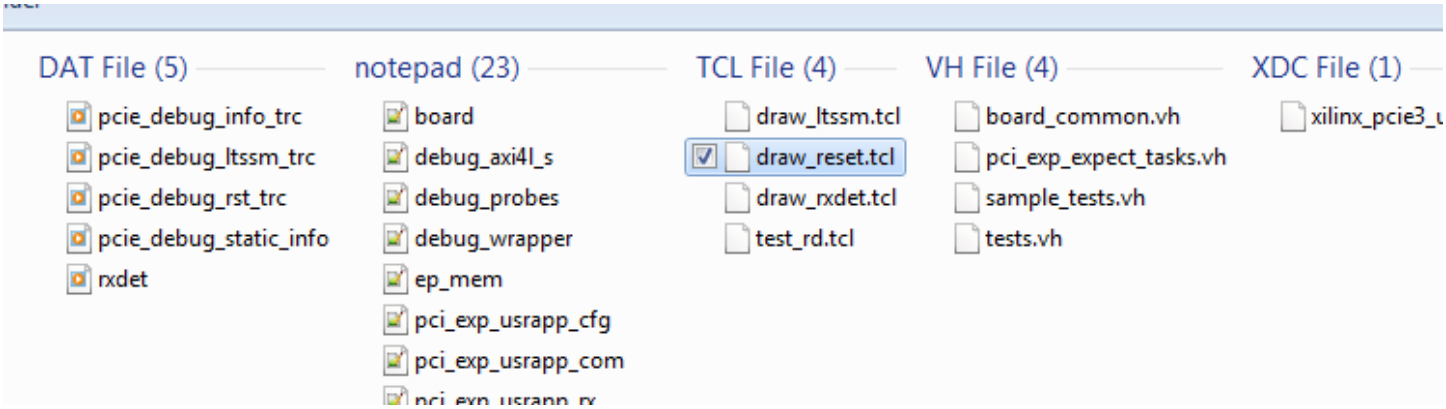

Figure 16 – Double click on draw_reset.tcl to generate PHY reset FSM graph

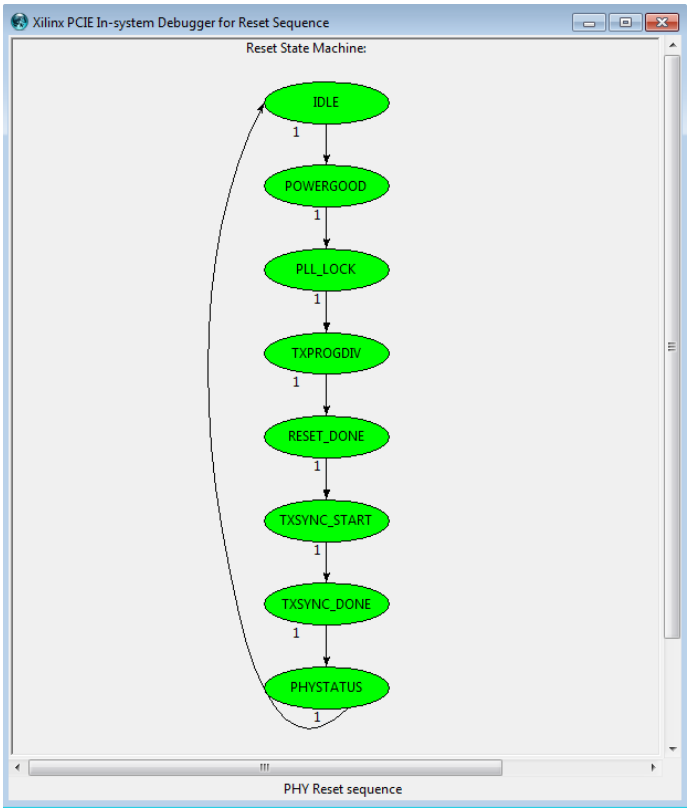

Figure 17 - PHY Reset FSM Graph

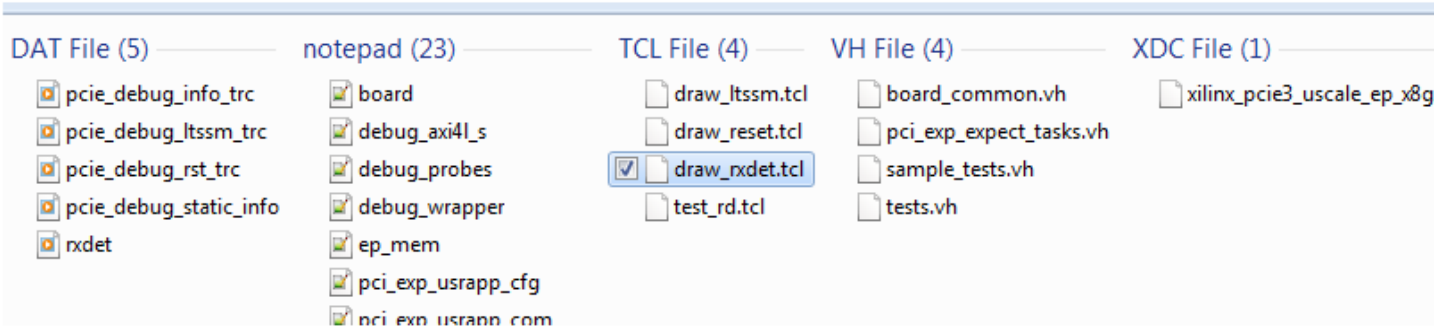

Figure 18 - Double Click on draw_rxdet.tcl to check the status of receiver detect on each lane

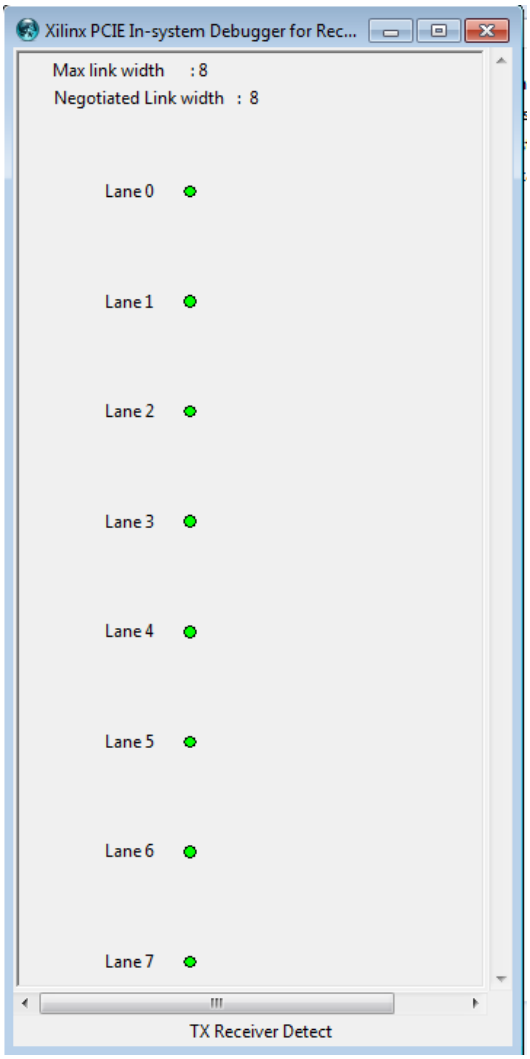

Figure 19 – Receiver successfully detected on all 8 lanes

In-System IBERT

In-System IBERT is a powerful feature, integrated into the Vivado 2016.3 core. This allows users to capture an eye diagram in real-time without any additional effort. An in-system eye scan is valuable in PCI express applications because placing a PCI Express link in loopback is not practical and generally not possible.

The screenshots below show the step-by-step instruction for using In-System IBERT in PCI Express Example design on a KCU105 development board.

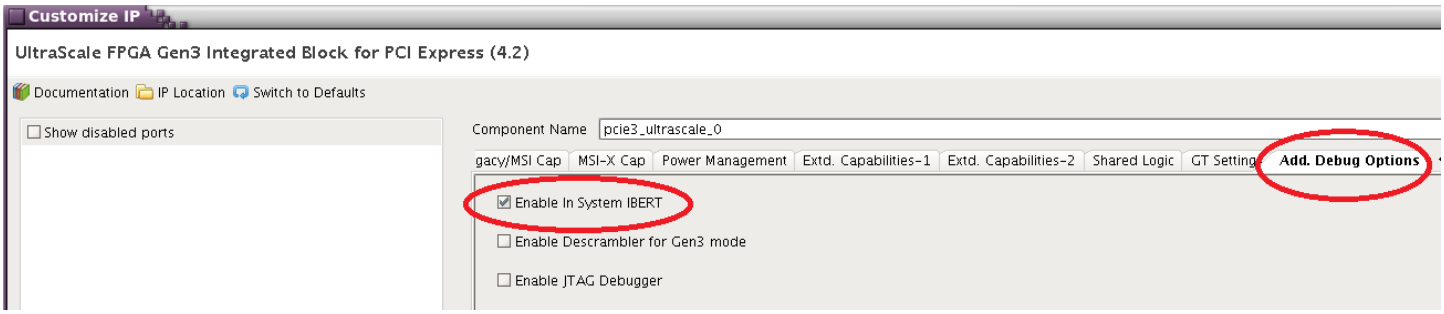

Figure 20 - Enable In-System IBERT

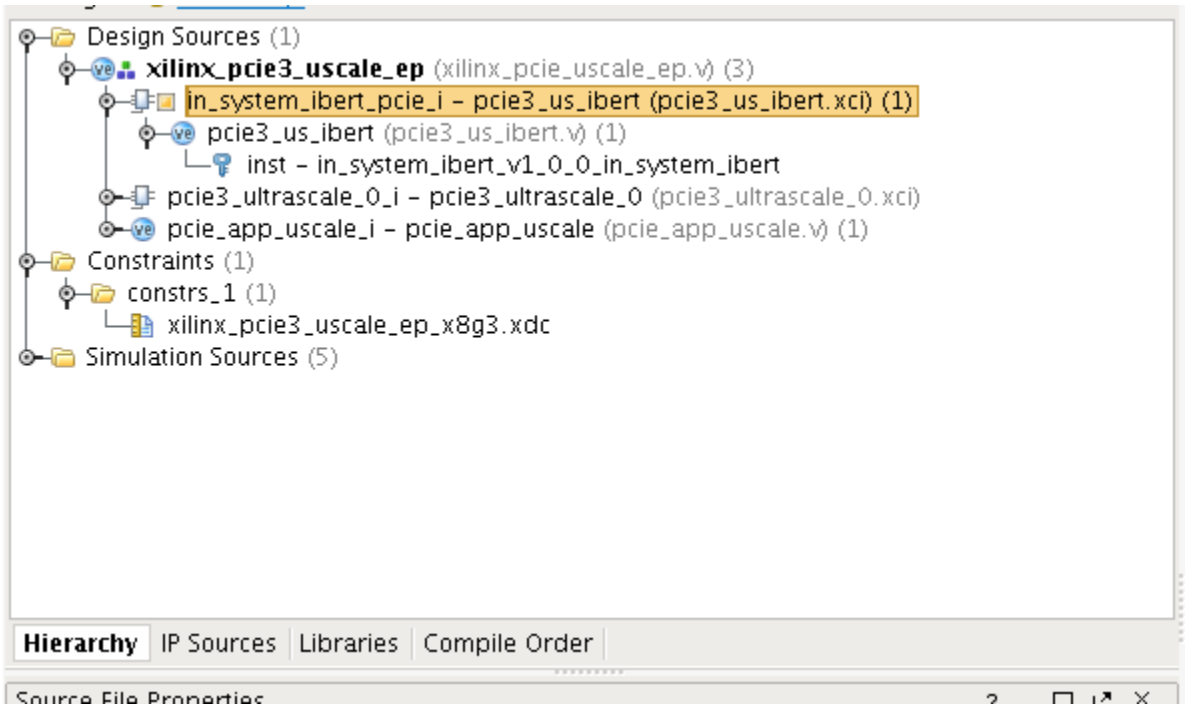

Figure 21 – IBERT instantiation in the generated example design

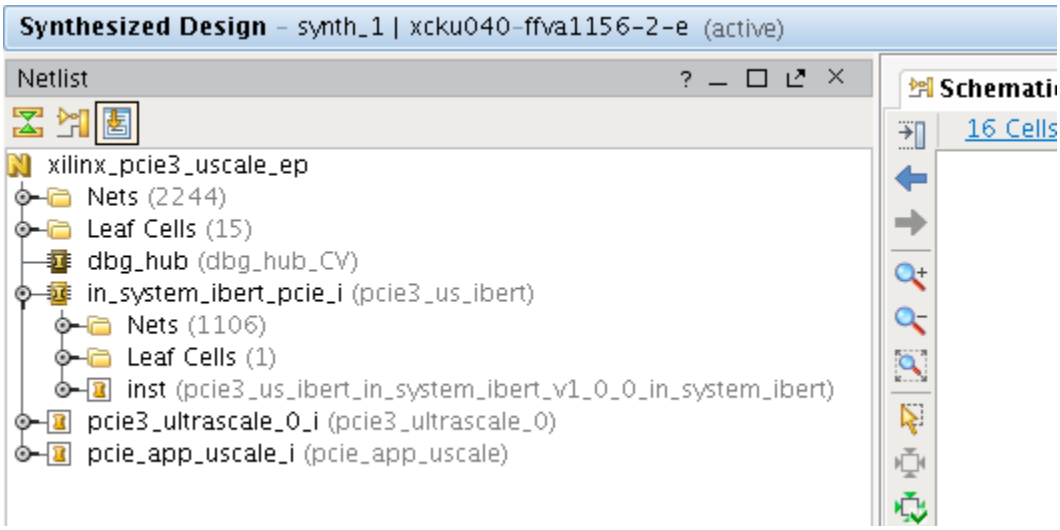

Figure 22 - IBERT instantiation in the synthesized design

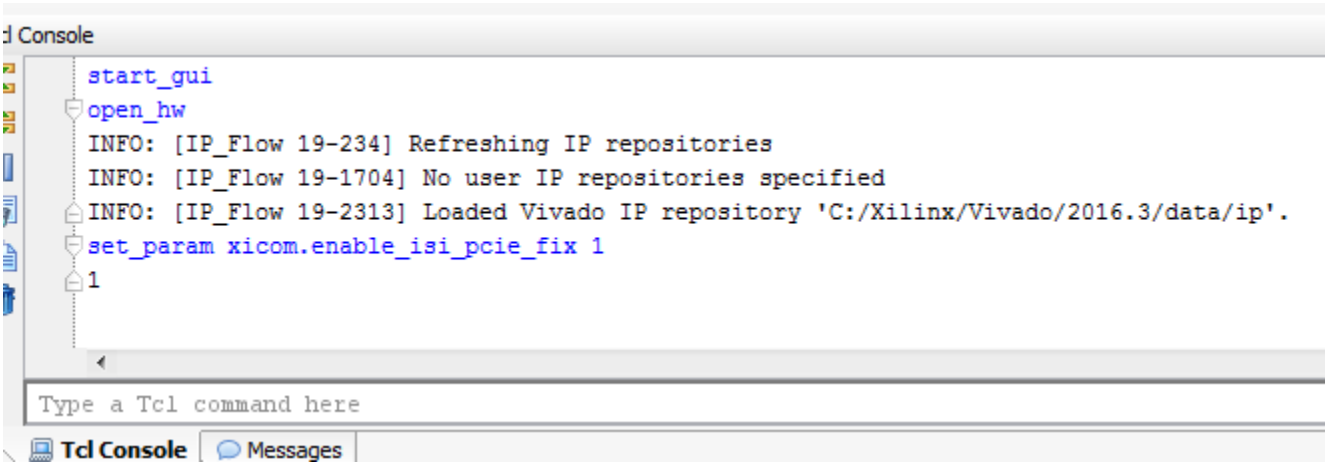

Figure 23 - Before programming the device execute set_param xicom.enable_isi_pcie fix 1

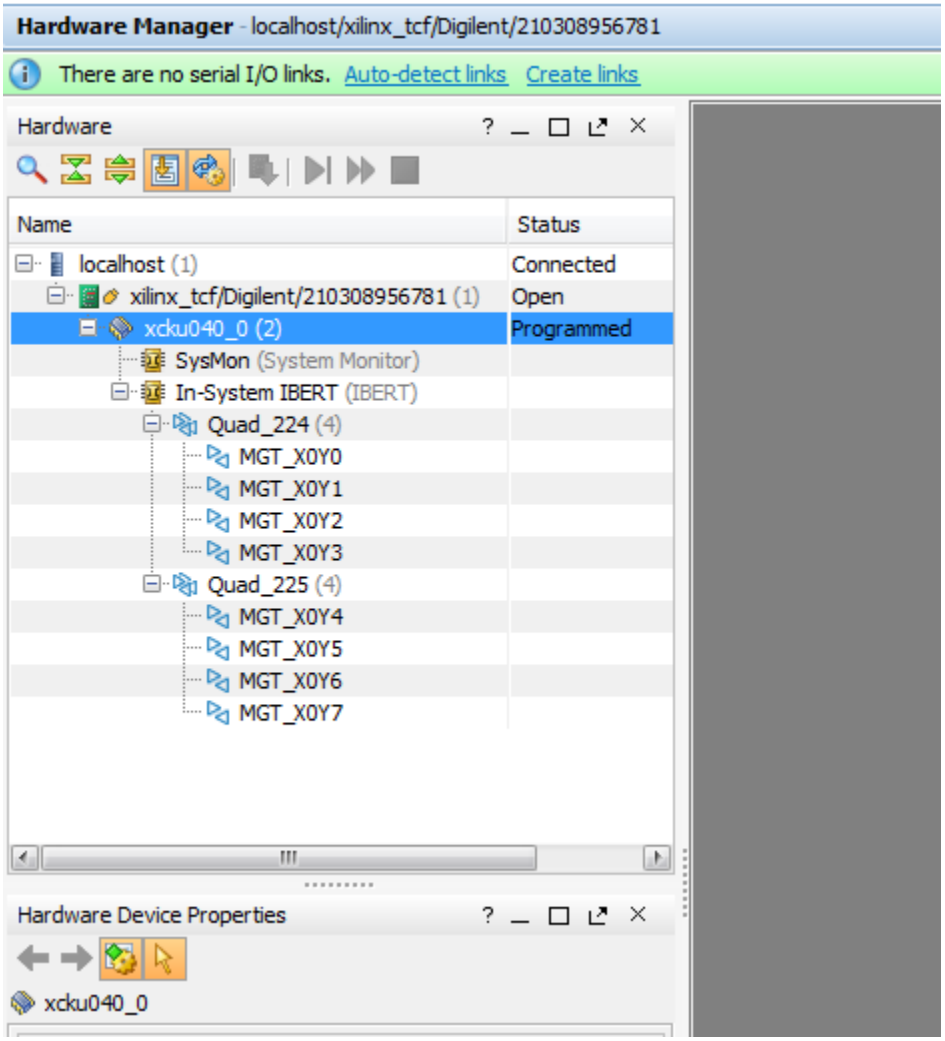

Figure 24 – In-System IBERT instantiation in Hardware Manager after programming the device

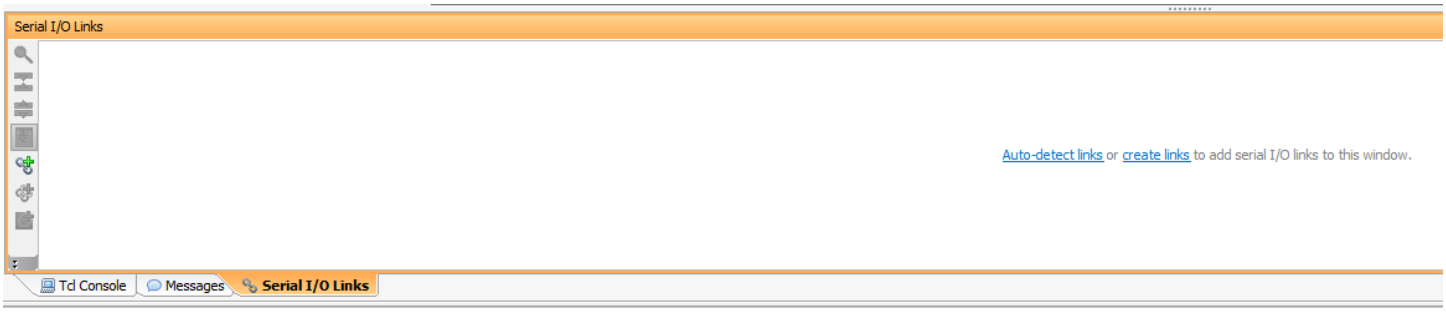

Figure 25 – 'Auto-detect links' and 'Create links' in 'Serial I/O Links' tab of the Hardware Manager

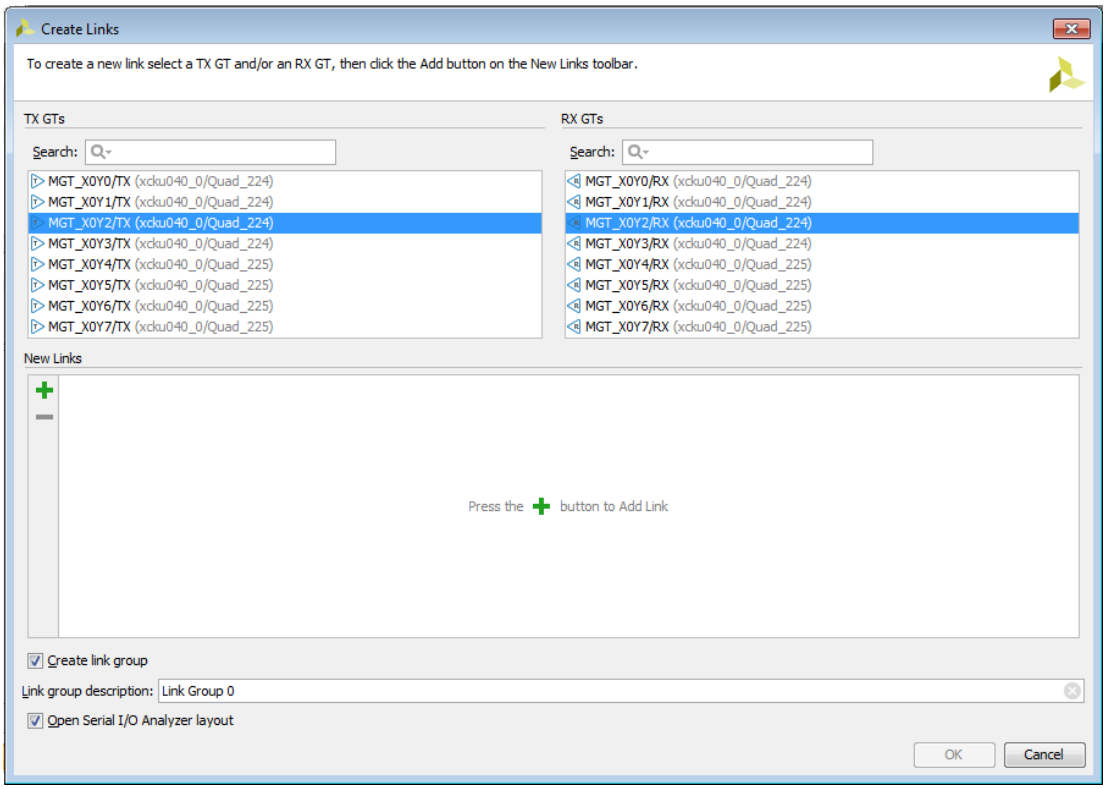

Figure 26 - Create link by clicking on 'Create links' in 'Serial I/O Links'

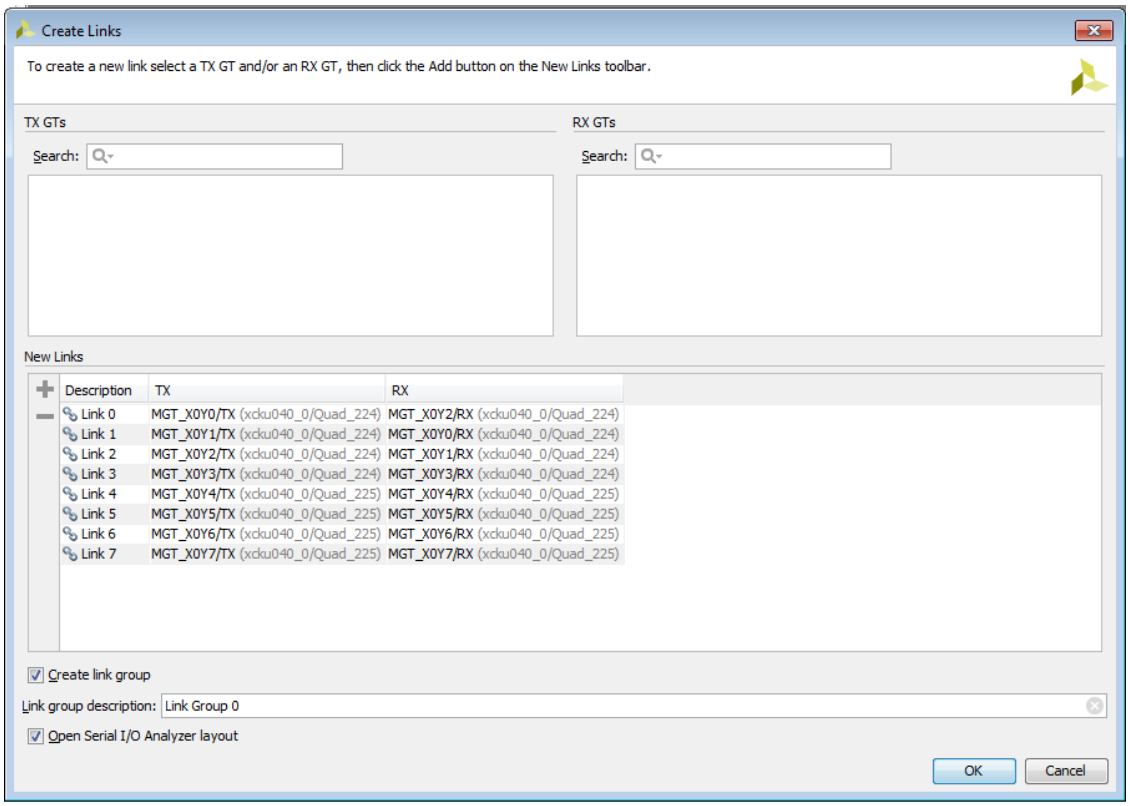

Figure 27 – In System IBERT x8 link

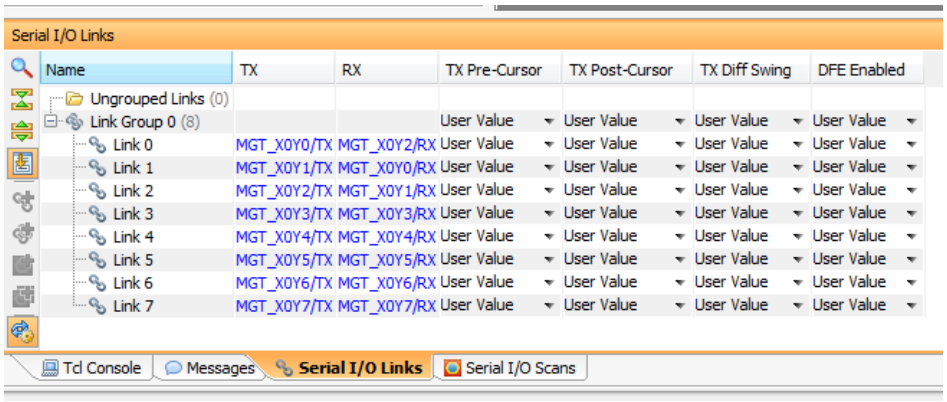

Figure 28 – Serial I/O Links

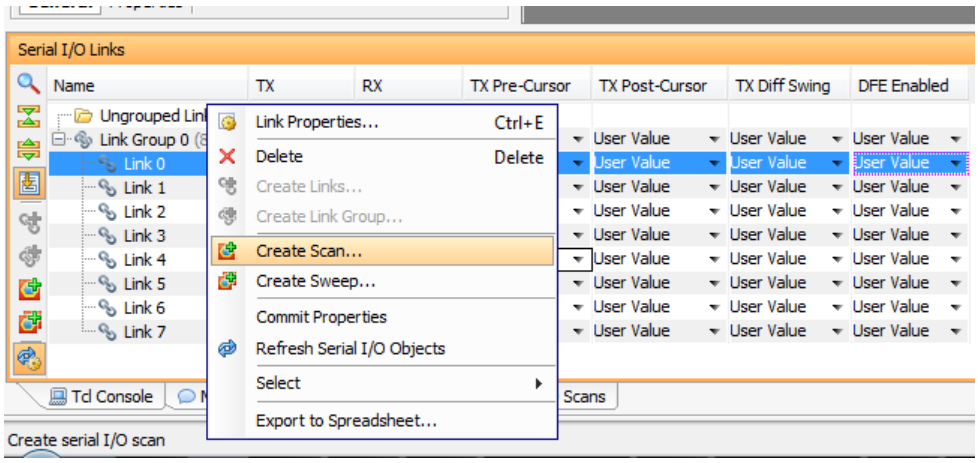

Figure 29 – Scan link

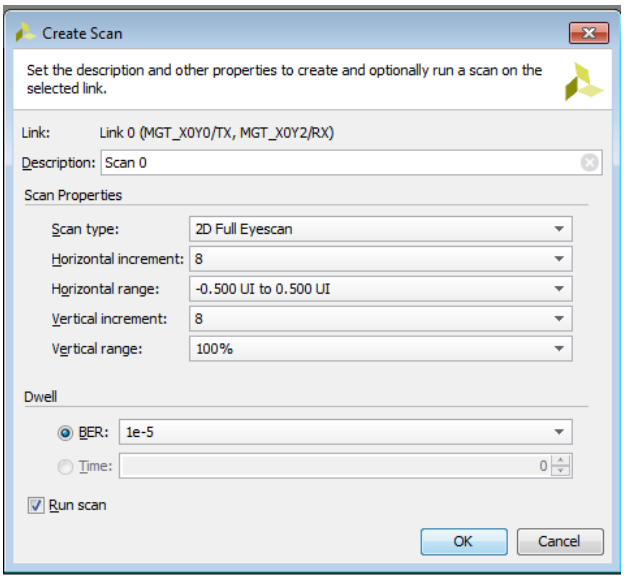

Figure 30 – Default scan options

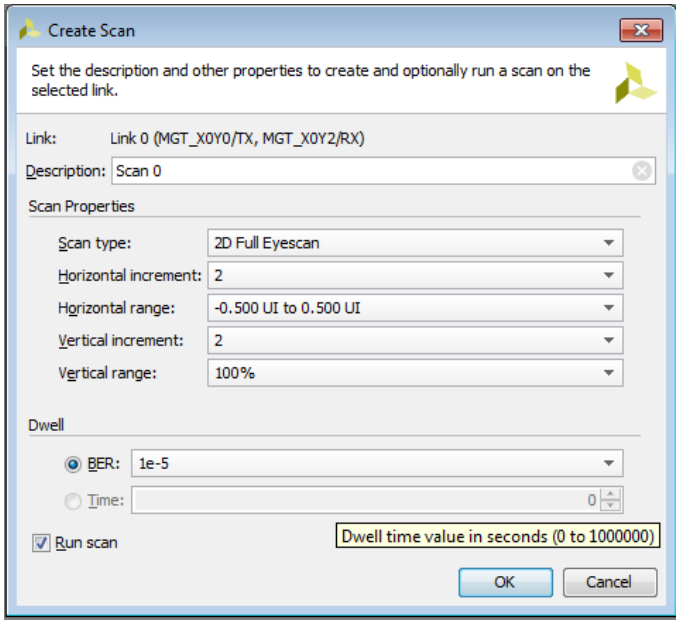

Figure 31 - For better results, try Horizontal and Vertical increment by 2 instead of the default value

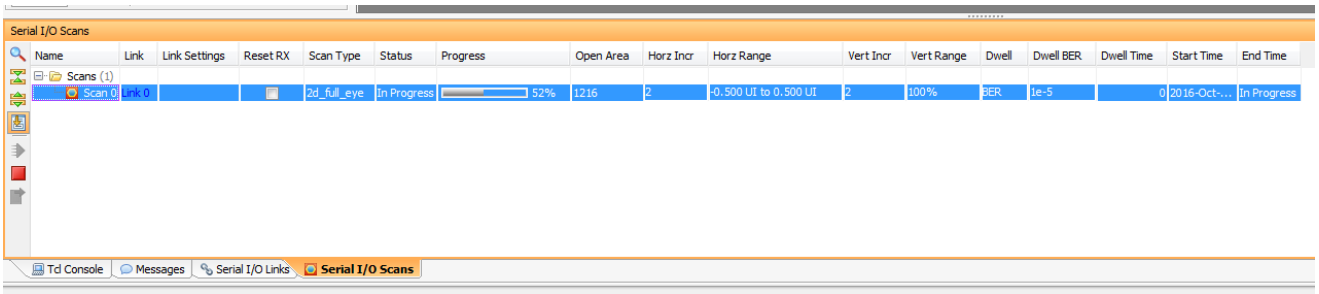

Figure 32 - Scan in progress

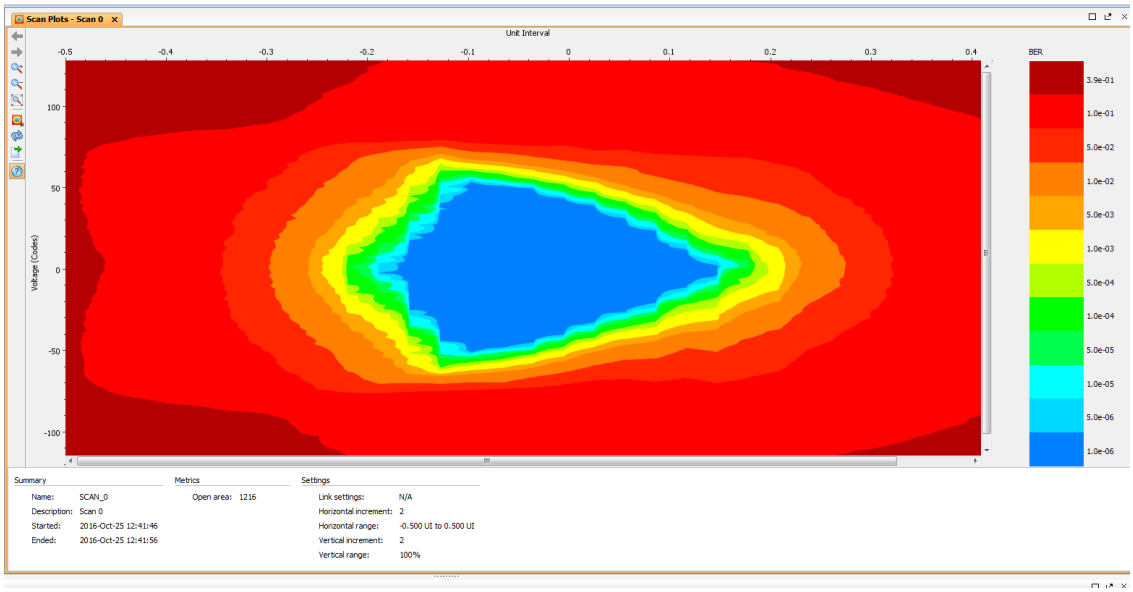

Figure 33 - Generated Eye Diagram

Note: *'Enable In-System IBERT' should not be used with the 'Falling Edge Receiver Detect' option in the GT Settings tab.*

Descrambler Module

The data on the PIPE interface is scrambled and hence the incoming and outgoing data cannot be read on this interface. The ability to interpret the data on this interface is especially useful where packets presented by the endpoint user application do not show up at the host or vice versa. Being able to identify the corresponding packets on the PIPE interface confirms whether the ingress packets definitely made it into the FPGA and whether the egress packets were definitely presented to the transceiver (PHY) by the PCIe MAC Hard IP.

In Vivado 2016.3, a new feature has been added where the user has an option to enable descrambler module to descramble the PIPE data. Figure 34 shows where the descrambler module sits. The descrambled data is read through an ILA. The instantiated descrambler module is encrypted and only provides a way for hardware-only support to debug Gen3 designs on the board; simulation with the descrambler module is not supported.

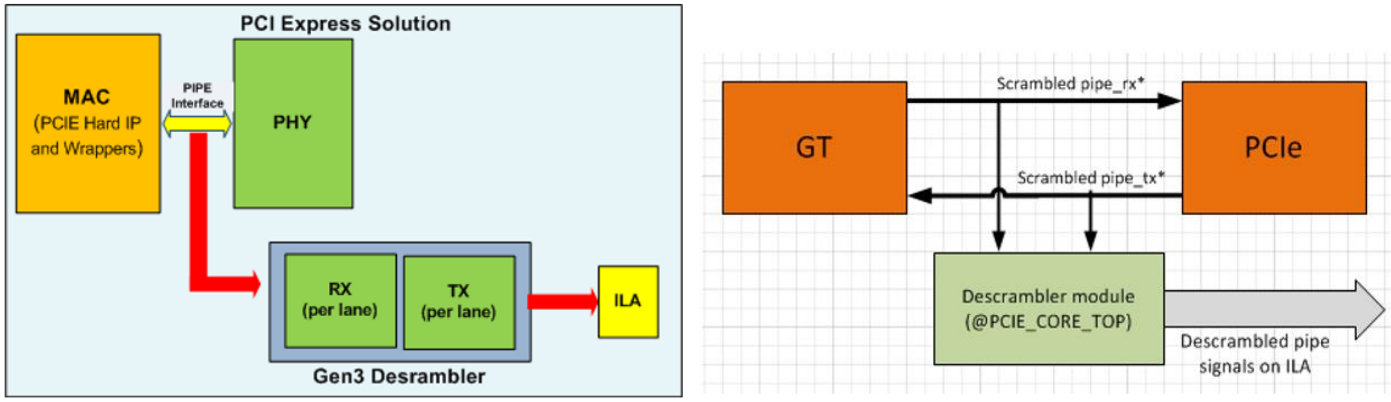

Figure 34 – Descrambler Module

The screenshots below show the step-by-step instruction for enabling the Descrambler Module and viewing the descrambled data on the ILA waveform in the PCI Express Example design on a KCU105 development board.

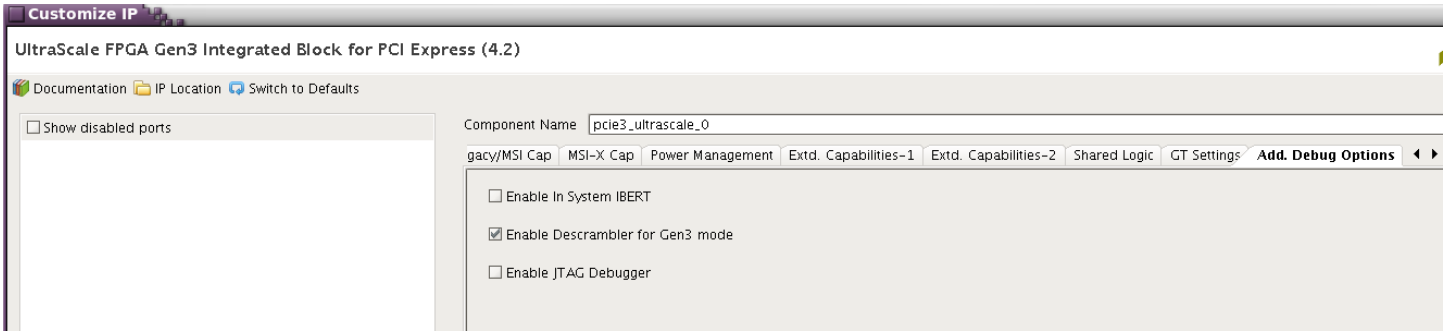

Figure 35 - Enable Descrambler for Gen3 Mode

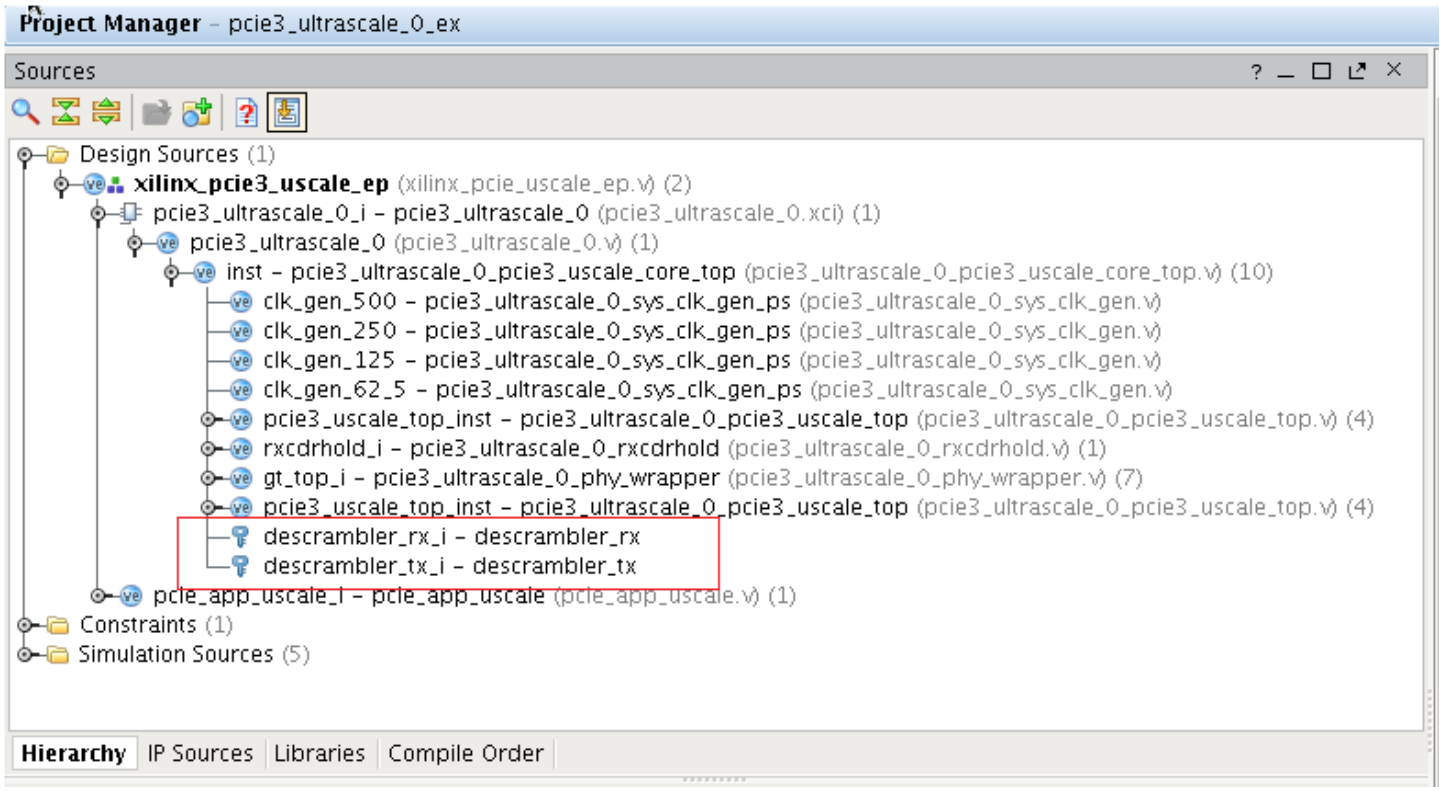

Figure 36 - TX Descrambler and RX Descrambler instantiation

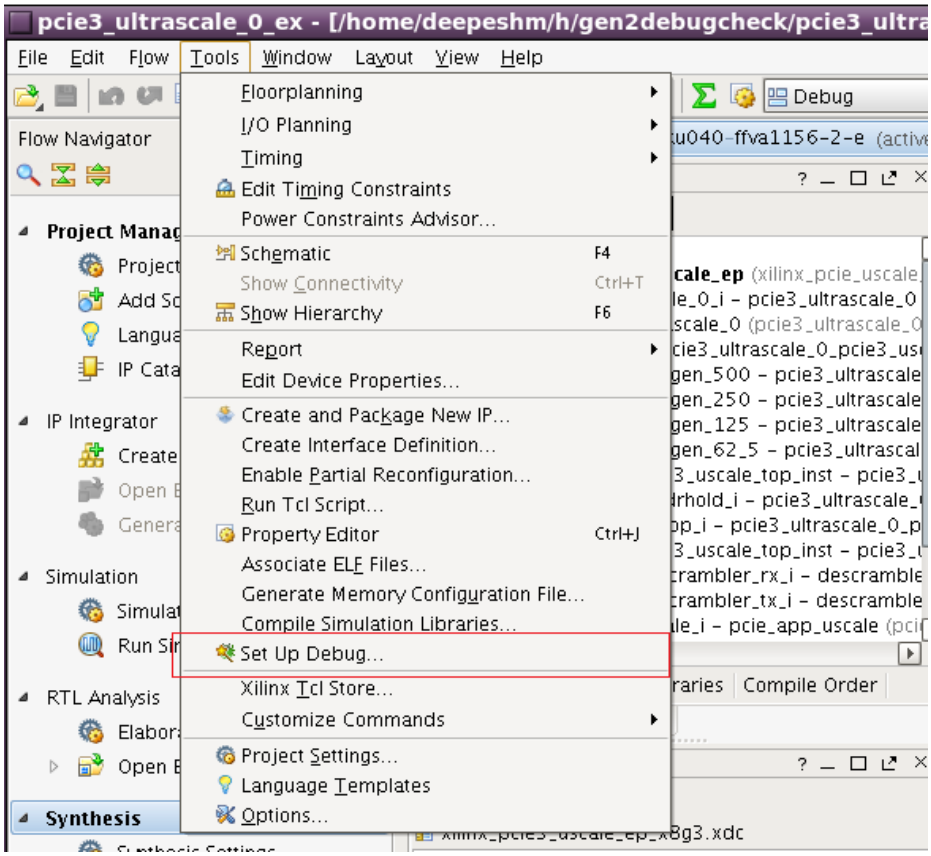

Figure 37 – After synthesizing the design, run 'Set Up Debug'

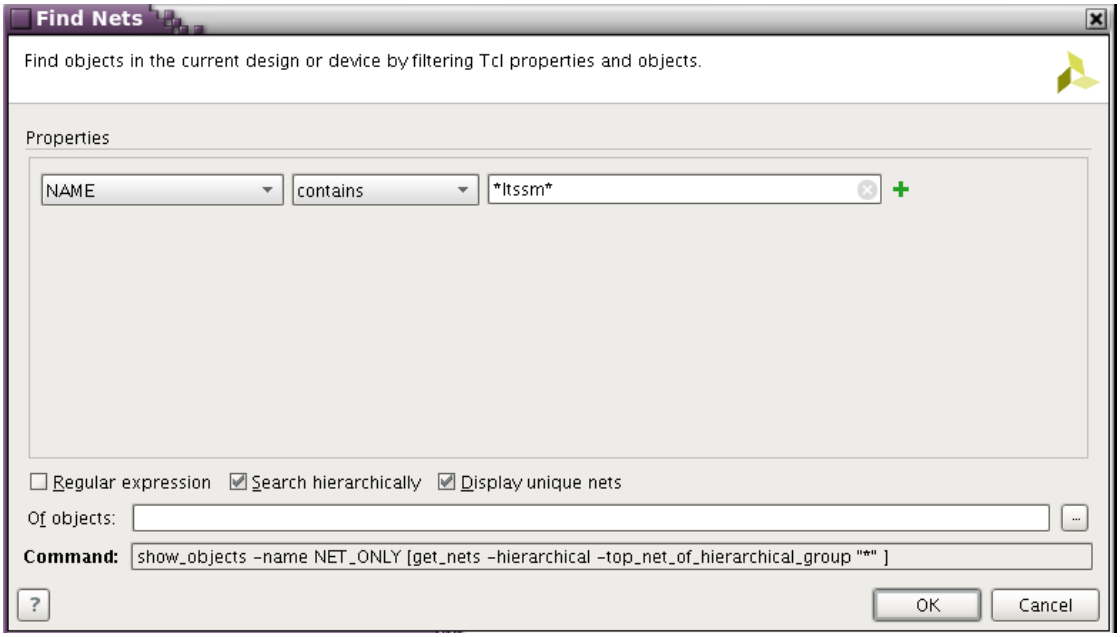

Figure 38 – Search cfg_ltssm_state signal

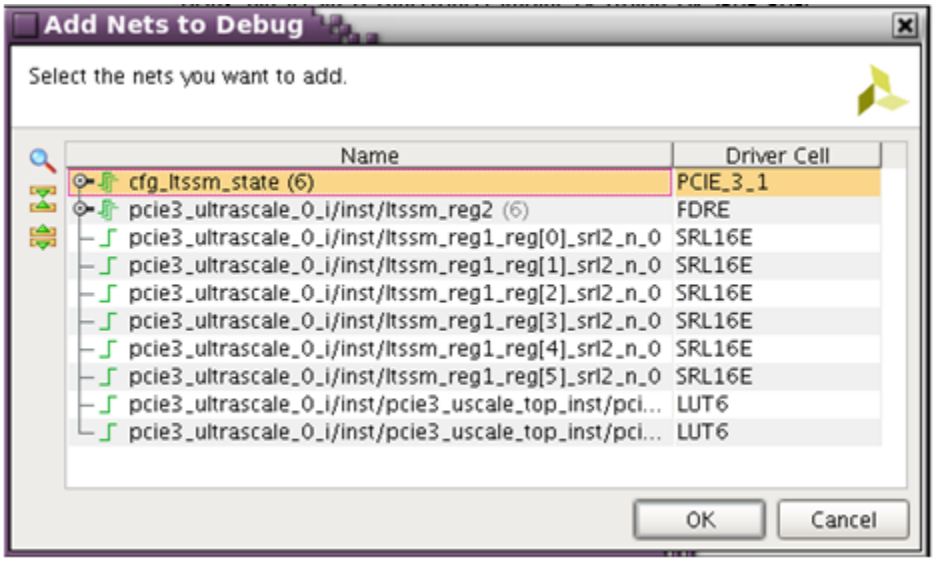

Figure 39 - Add cfg_ltssm_state to Debug

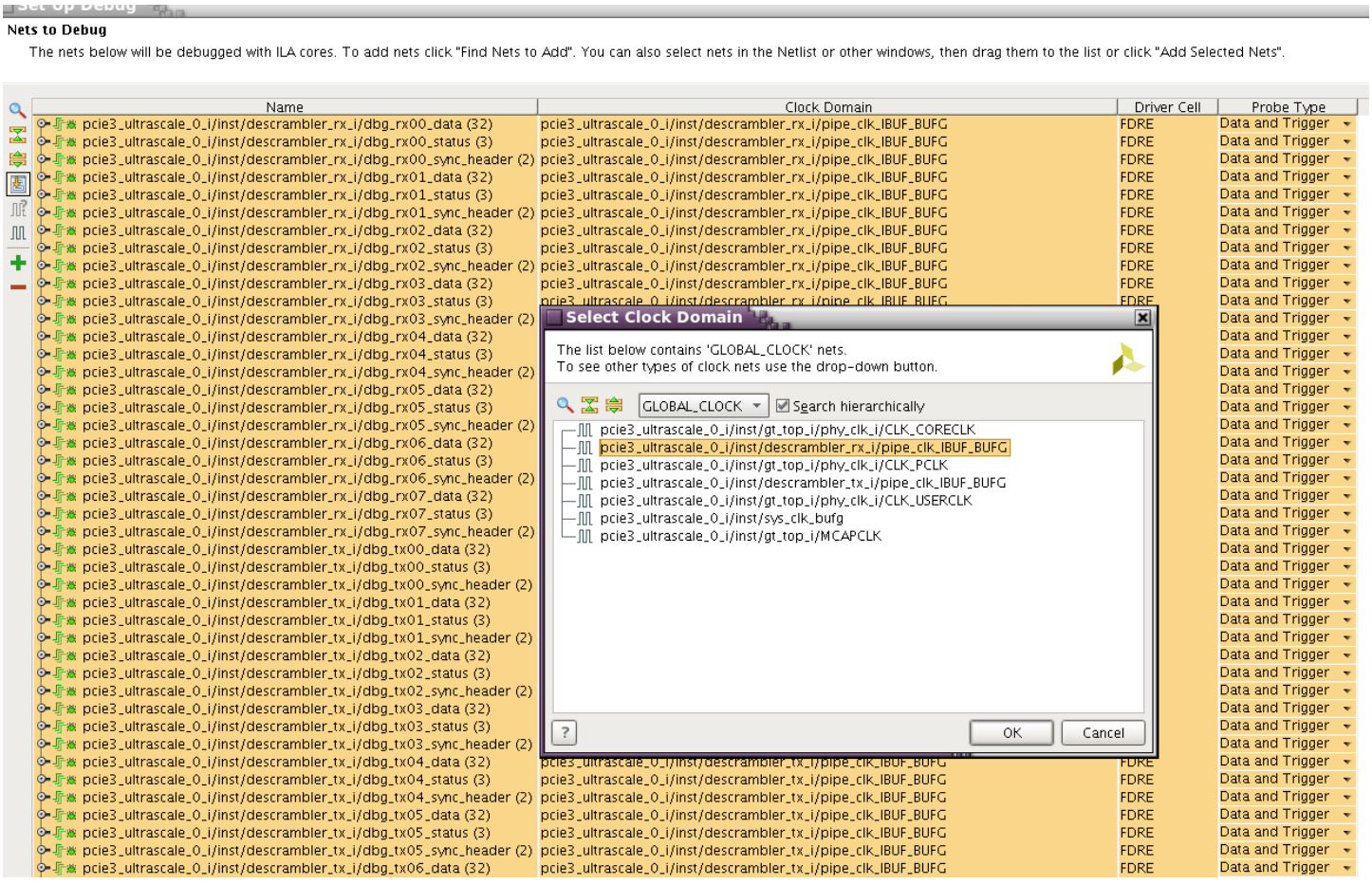

Figure 40 - Select the same clock domain for all signals

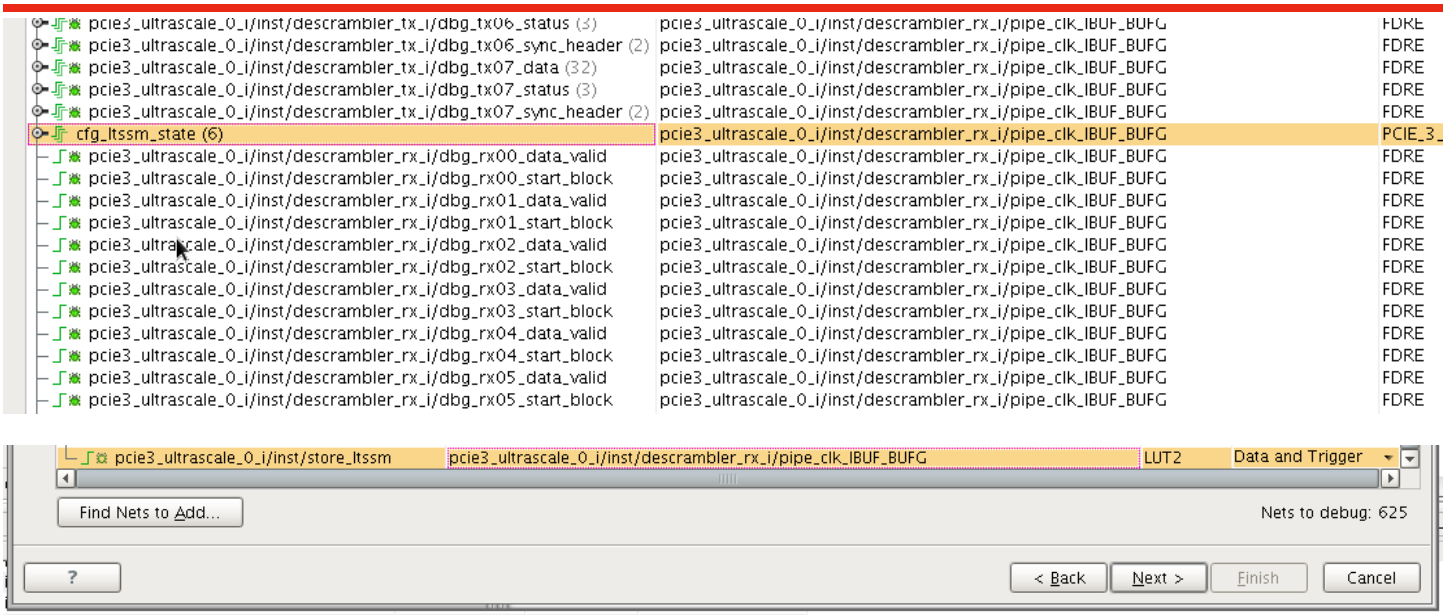

Figure 41 – In addition to signals related to descrambler, there will be two additional signals: store_ltssm and cfg_ltssm_state in the debug nets

Note: store_ltssm is used to capture data on every transition of the ltssm states. To do that, enable 'Capture Control' in 'Set up Debug' as shown in Figure 42.

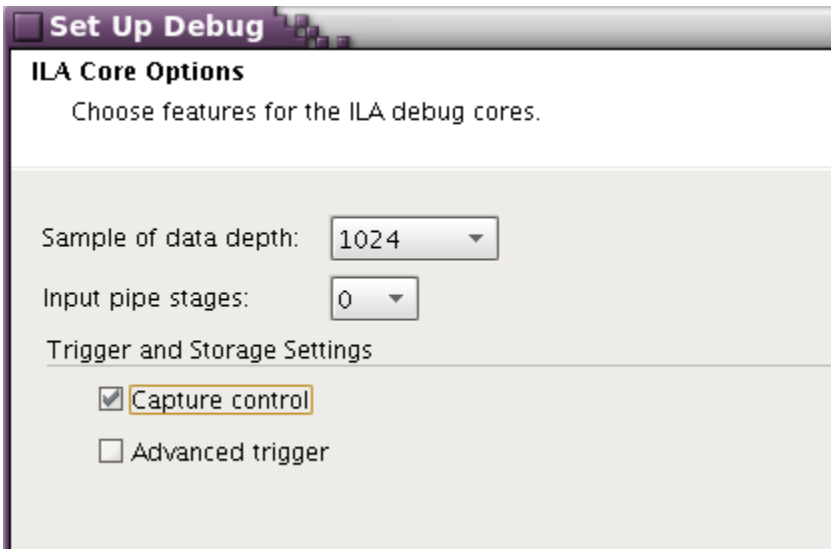

Figure 42 - Enable 'Capture Control'

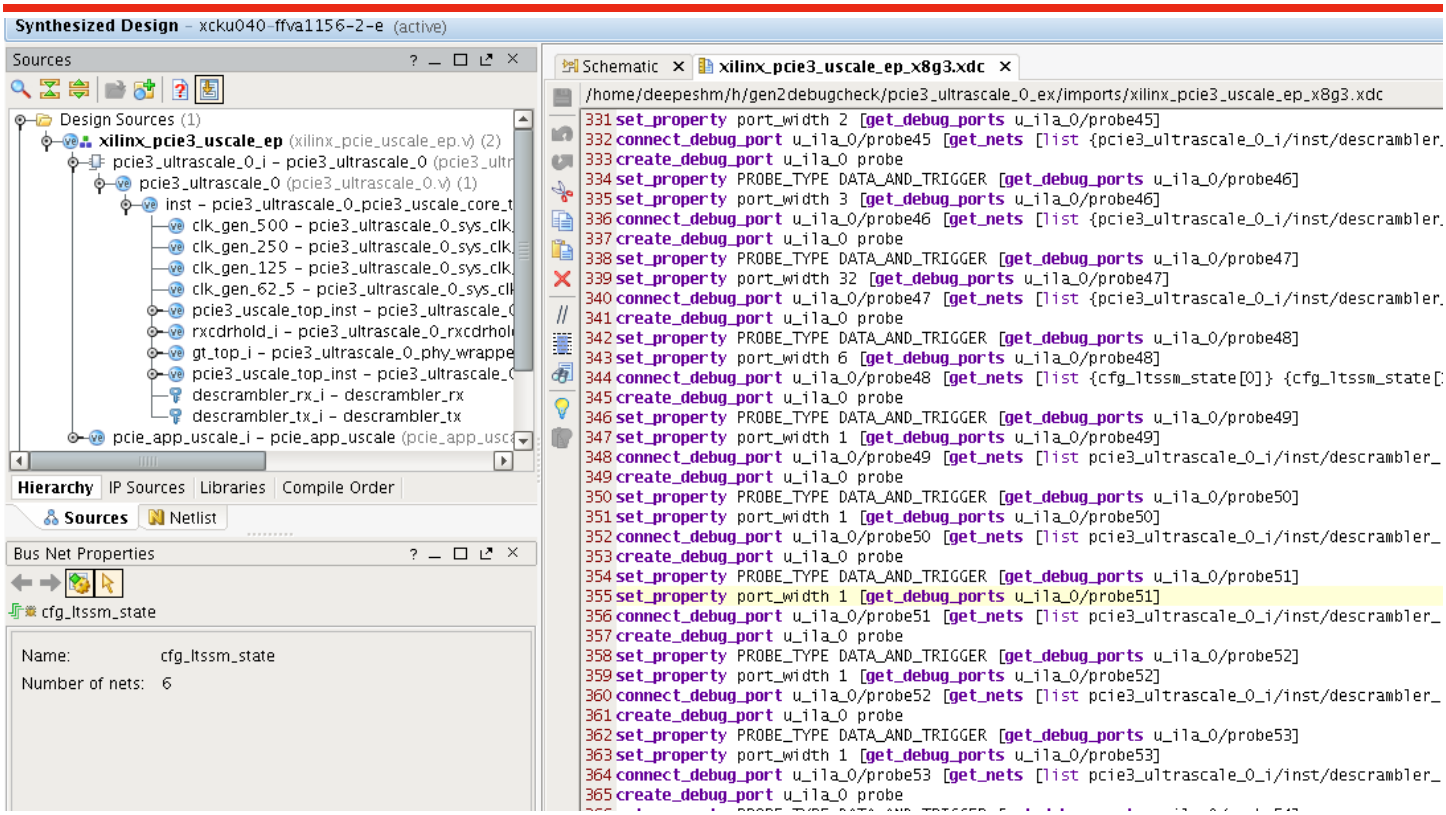

Figure 43 – After the 'Set up Debug' is complete, save the project. The XDC file should be updated with the ILA constraints

Select 'store_ltssm' in the Trigger Setup.

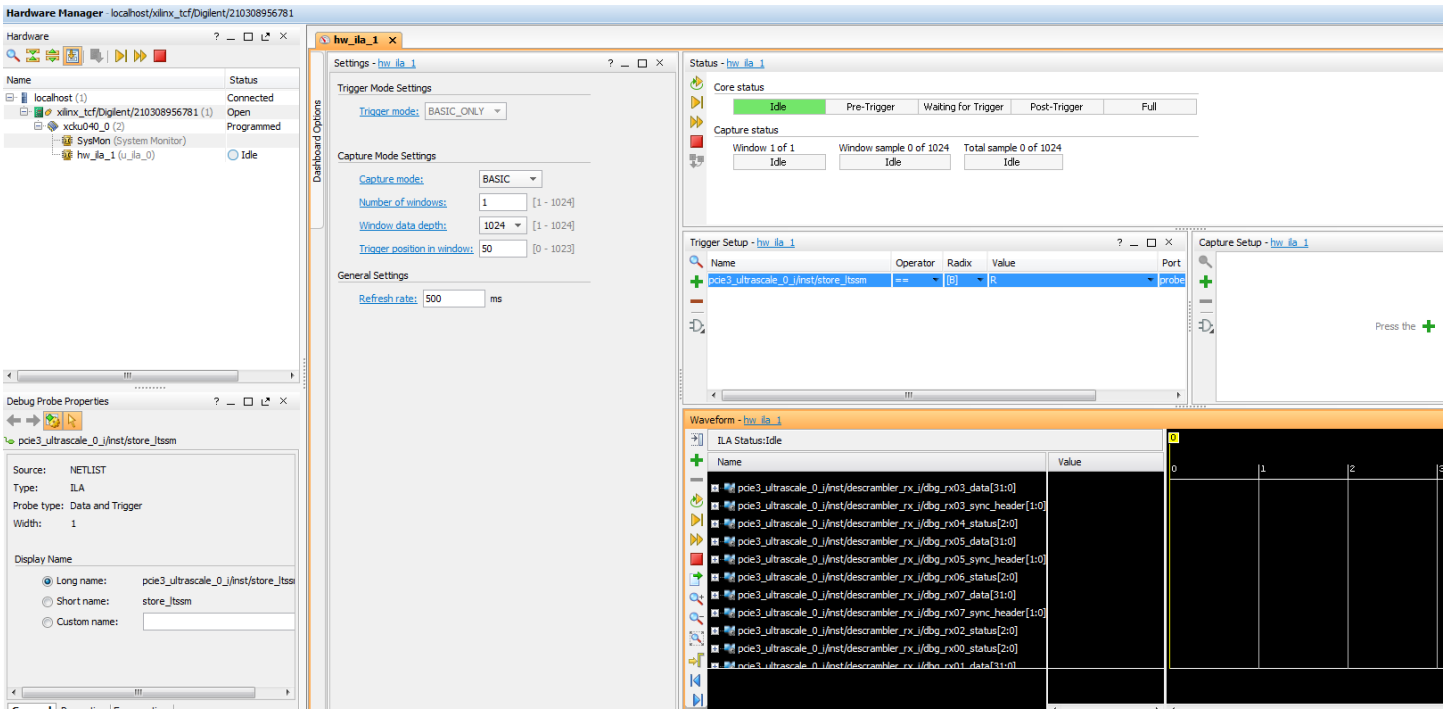

Figure 44 - Hardware Manager after programming the device

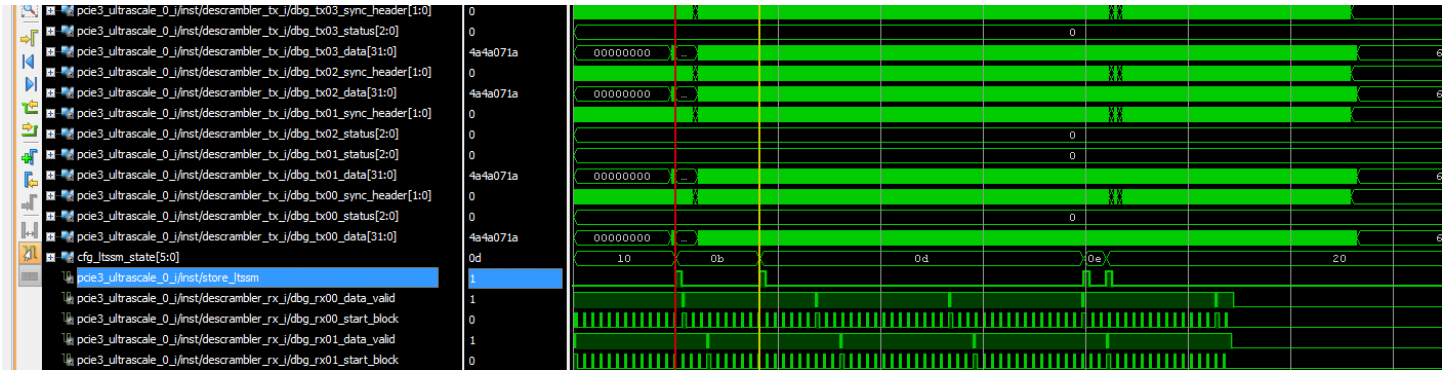

Figure 45 - ILA Capture

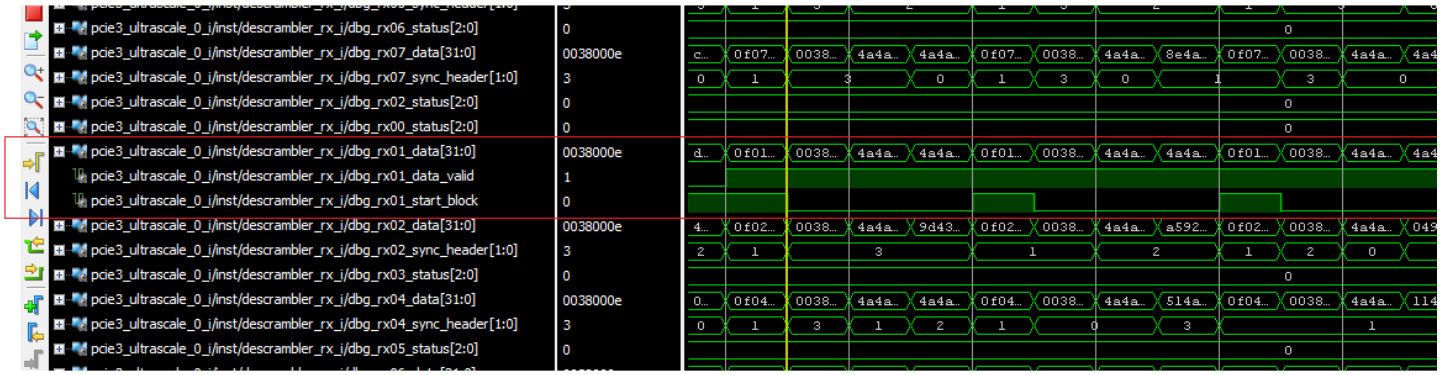

Figure 46 - Zoomed in View showing descrambled data on RX PIPE interface

cfg_ltssm_state	Output	6	LTSSM State. Shows the current LTSSM state: 00: Detect.Quiet 01: Detect.Active 02: Polling.Active 03: Polling.Compliance 04: Polling.Configuration 05: Configuration.Linkwidth.Start 06: Configuration.Linkwidth.Accept 07: Configuration.Lanenum.Accept 08: Configuration.Lanenum.Wait 09: Configuration.Complete 0A: Configuration.Idle 0B: Recovery.RcvrLock 0C: Recovery.Speed 0D: Recovery.RcvrCfg 0E: Recovery.Idle 10: L0 11-16: Reserved 17: L1.Entry 18: L1.Idle

Figure 47 – LTSSM States from PG213

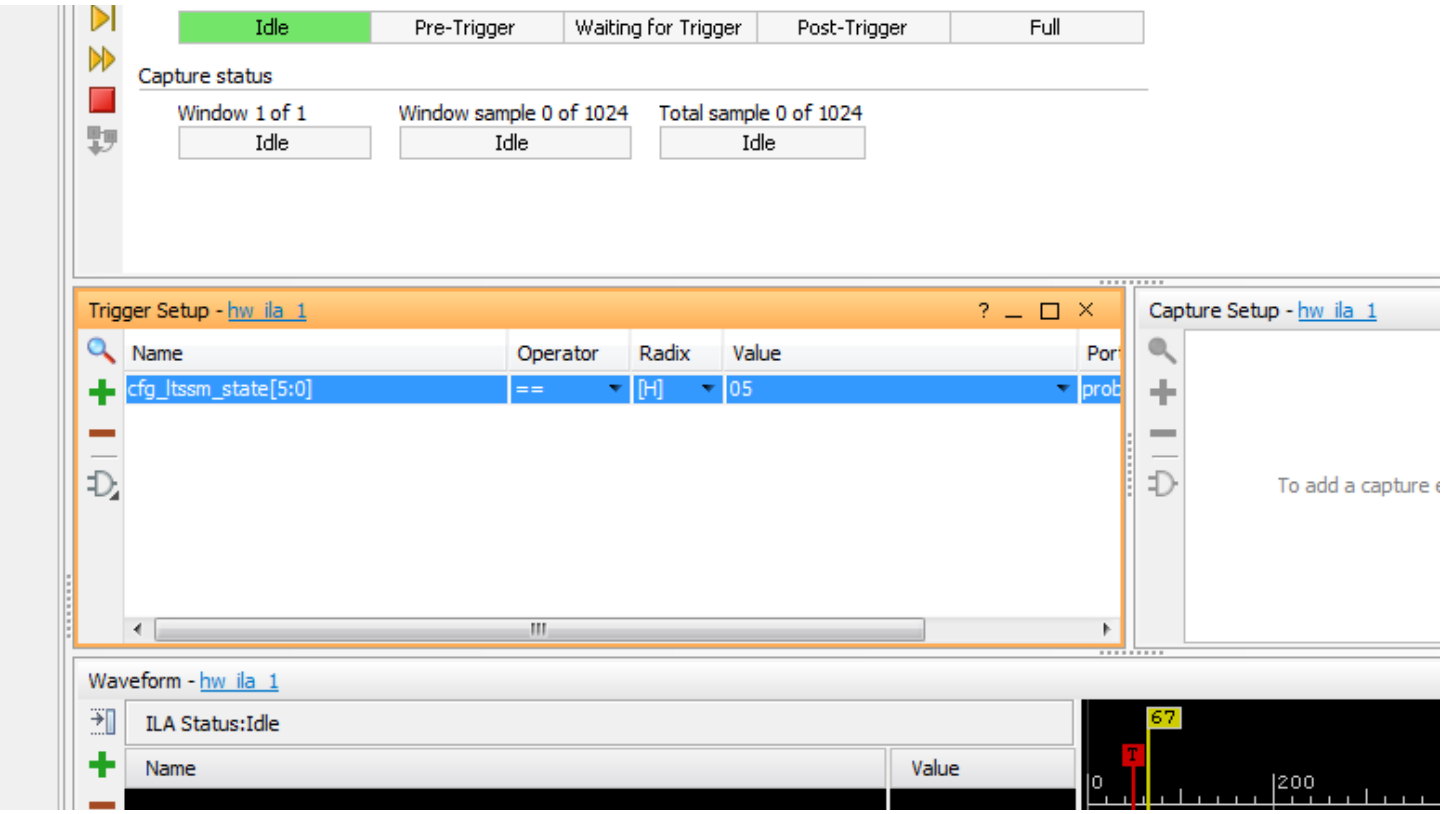

Figure 48 - Trigger Setup to trigger when cfg_ltssm_state is '05' (Configuration.Linkwidth.Start)

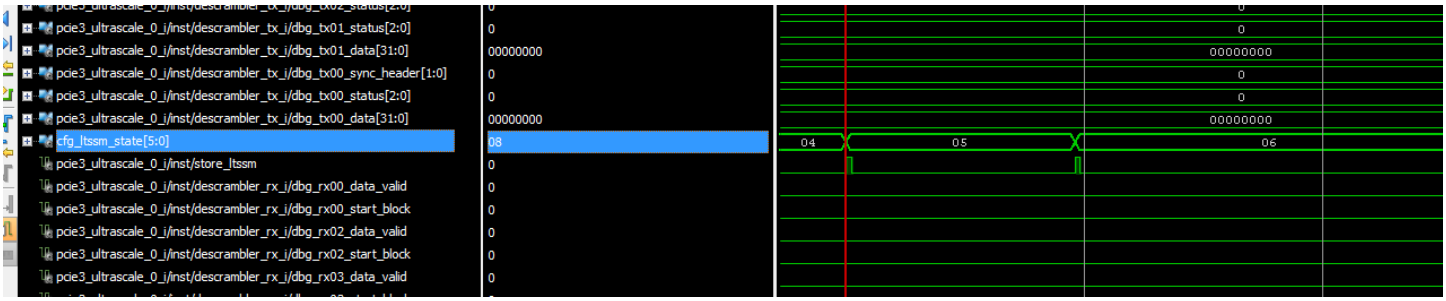

Figure 49 – Waveform Capture with trigger on cfg_ltssm_state = ‘05’

Revision History

11/20/2016 - Initial release