ESISTREAM 62B64B

Get started with VCK190 loopback package

VERSAL AI CORE XCV1902-2MSEVSVA2197

Summary

1 | Hardware setup

2 | From Vivado project creation to link synchronization

3 | Project overview

4 | From Vivado project creation to link synchronization

5 | Acquire RX data (62-bit)
Using Vivado Integrated Logic Analyzer (ILA)

6 | Design example top file testbench simulation

7 | Application Note:
Create an ESIstream Gigabit Transceivers Block Design (.bd) from scratch on Versal.

8 | UART interface protocol

Hardware Setup

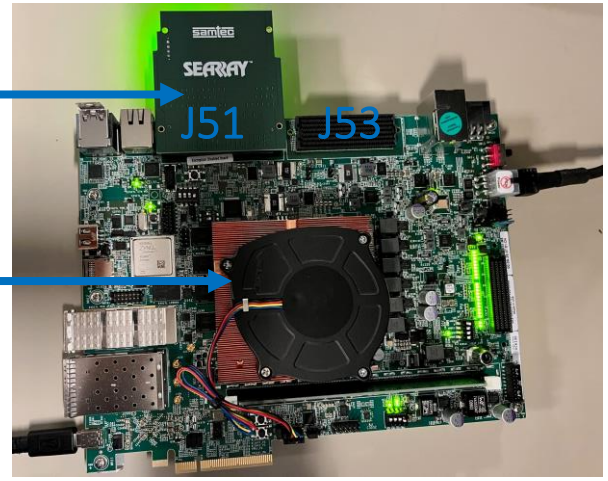
Hardware setup

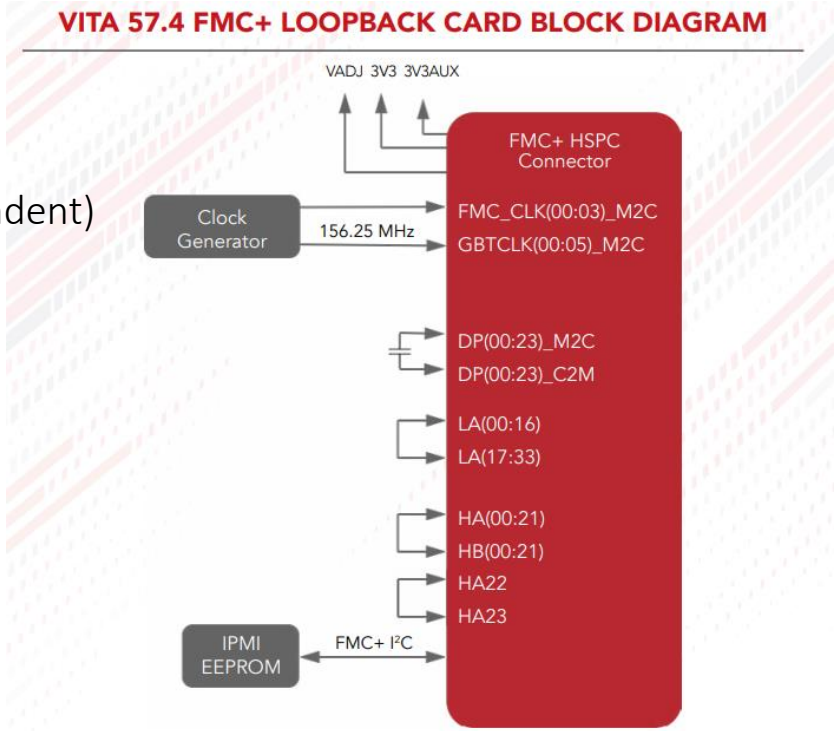
Ordering information

- VCK190 Versal AI Core Evaluation kit product web page.
  - Avnet: EK-VCK190-G by AMD Xilinx Evaluation & Development Kits | Avnet
- SAMTEC VITA 57.4 FMC+ HSPC Loopack Card product web page
  - Avnet REF-197618-01 by Samtec Hardware Development Tools | Avnet
  - Mouser: REF-197618-01 Samtec | Mouser Europe
    - FMC I/O voltage: VADJ=1.2 V, 1.5 V, 1.8 V, 2.5 V or 3.3 V (FPGA carrier card dependent)

Loopback board

VCK190 Evaluation kit

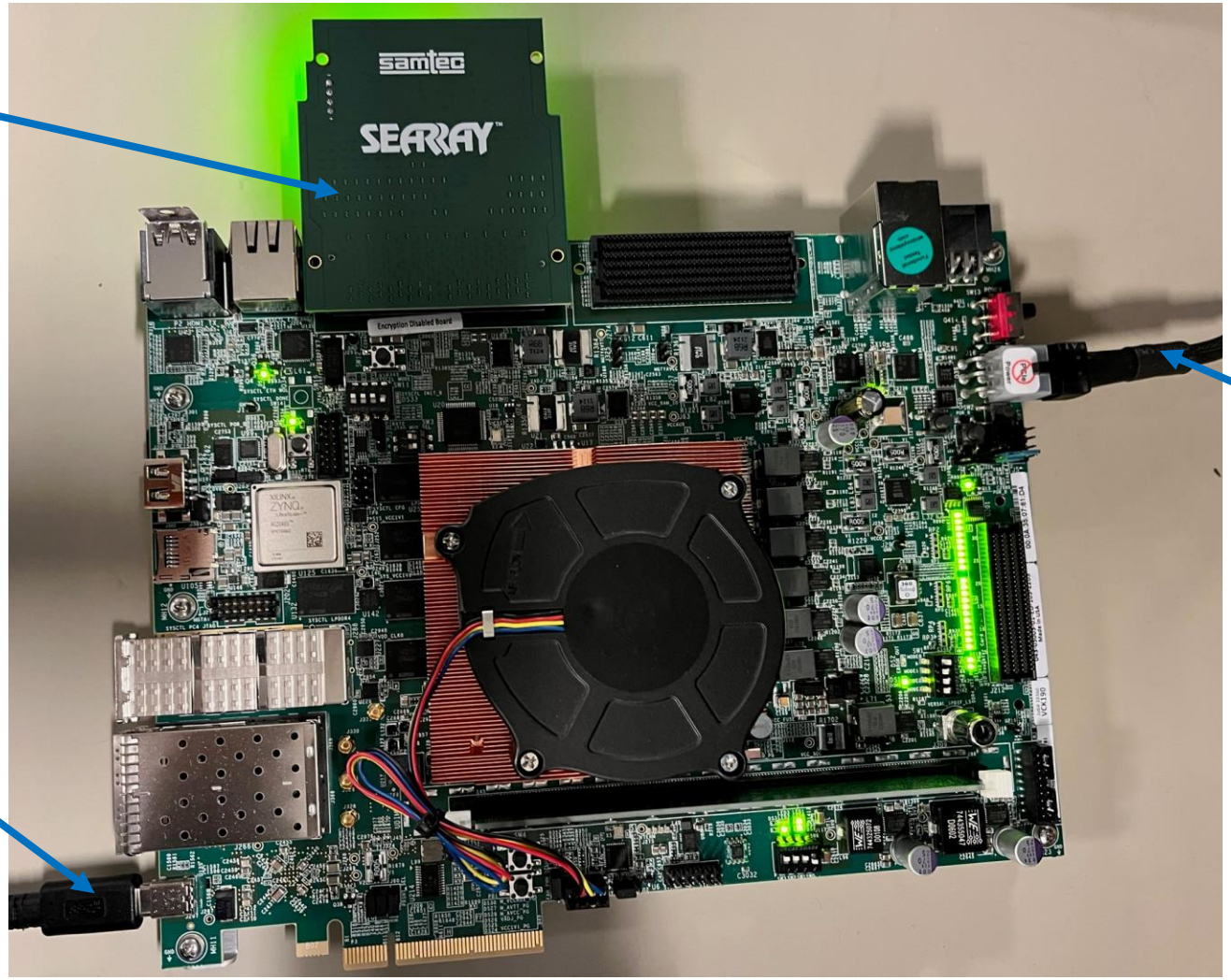




Hardware setup

Connect cables and loopback board on J51

Connect XM107 loopback board

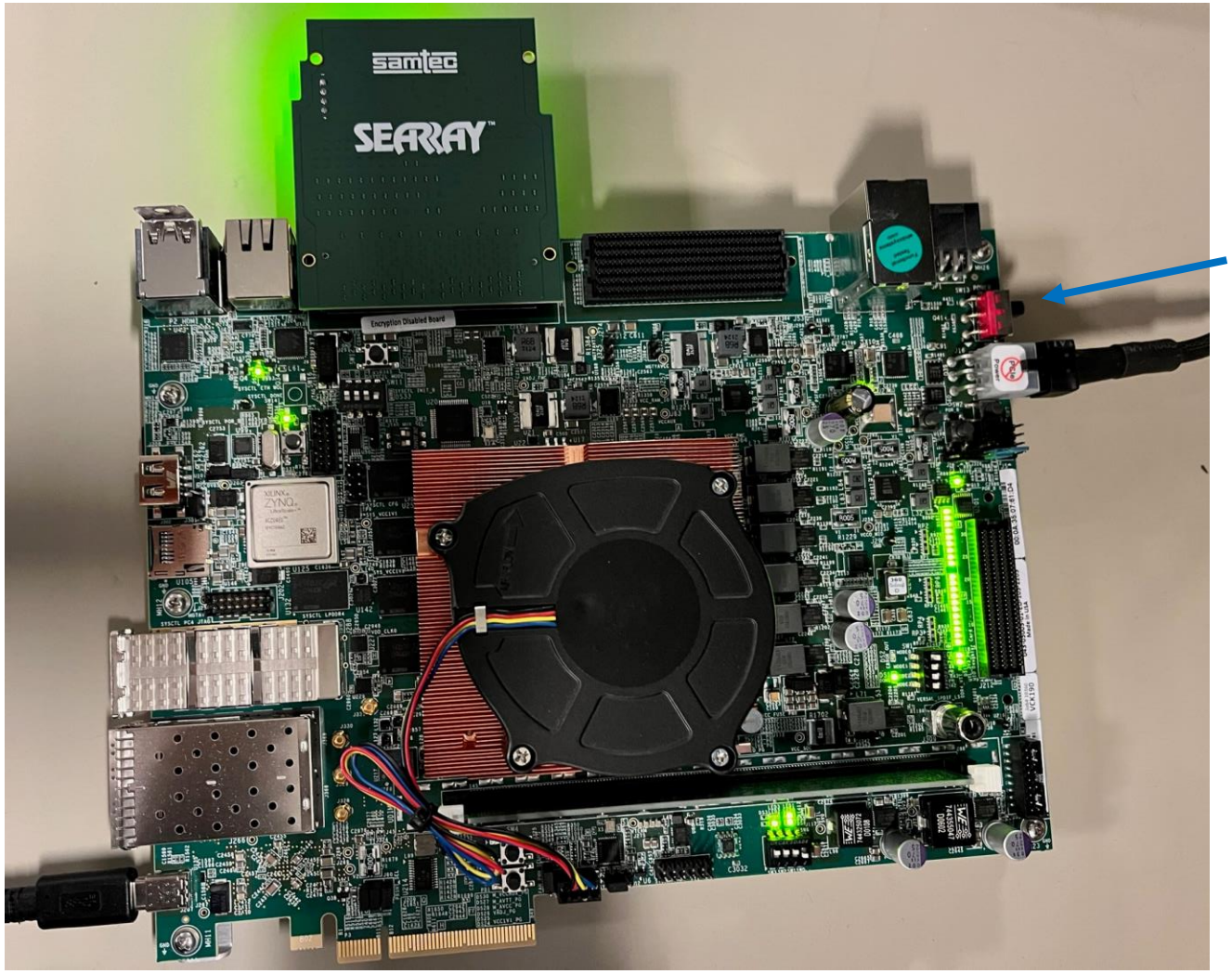


Connect power cable

Connect USB C JTAG + UART cable between PC and VCK190

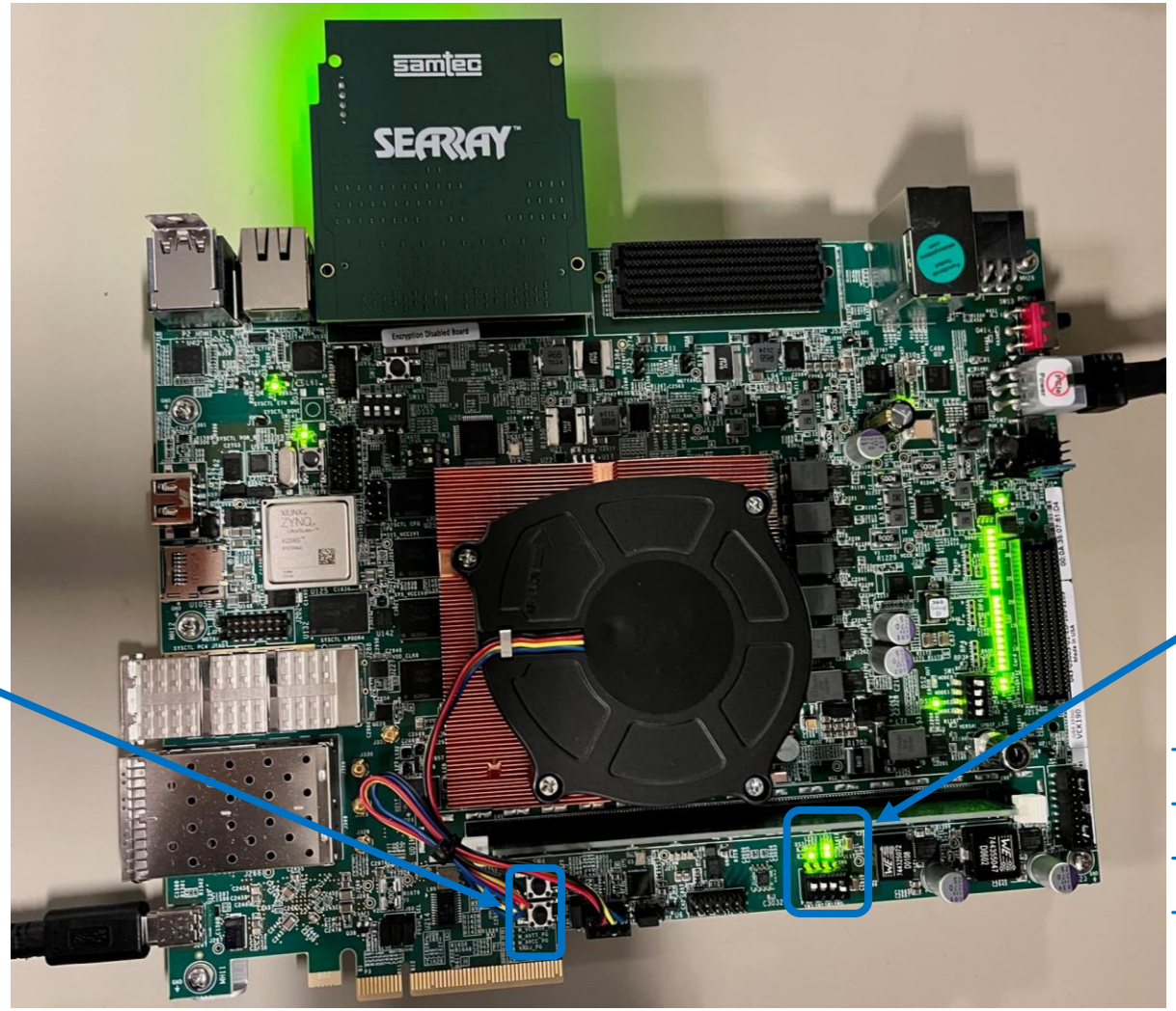
Hardware setup

Power-ON VCK190 Evaluation Kit



User interface

Push-Buttons (x2), Led (x4), Switches (x4)



LED3: BE or CB STATUS
LED2: RX LANES READY and VALID STATUS

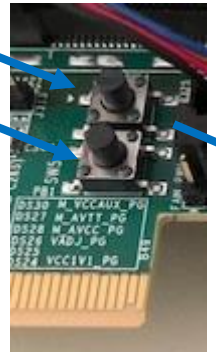
LED1: TX and RX IP READY
LED0: UART READY

‘1’

‘0’

PB0 SYNC

PB1 RST GT



TX DATA CTRL 1 — ‘1’ (ramp enabled)

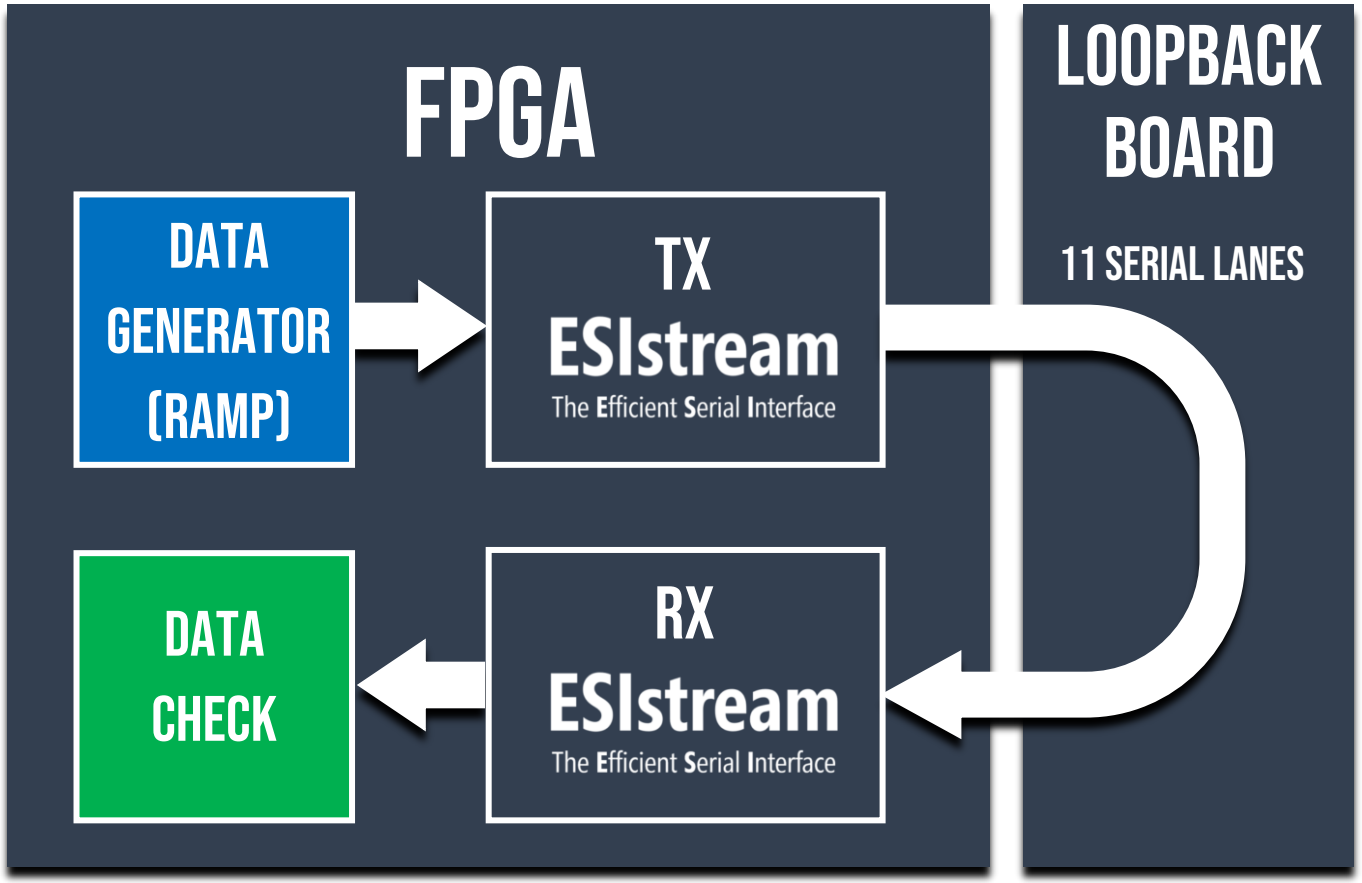
TX CLOCK BIT ENABLE — ‘1’ (enabled)

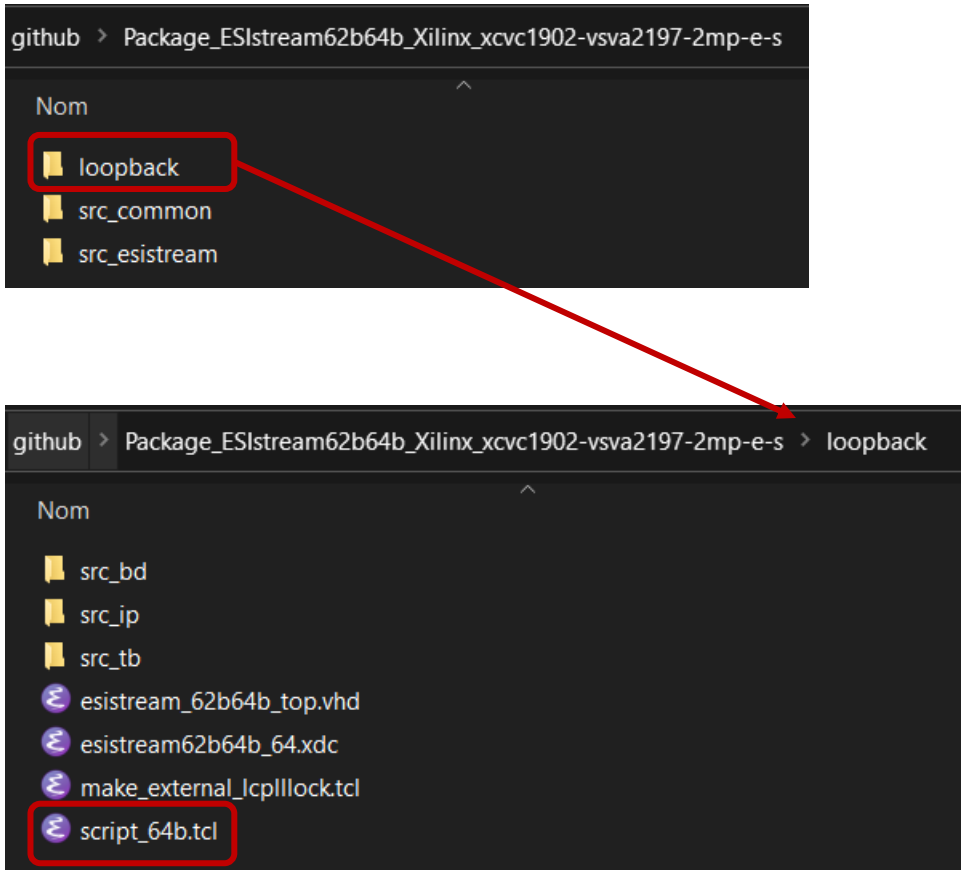
TX and RX PRBS ENABLE — ‘1’ (enabled)

TX DISPARITY BIT ENABLE — ‘1’ (enabled)

Project overview

Project overview

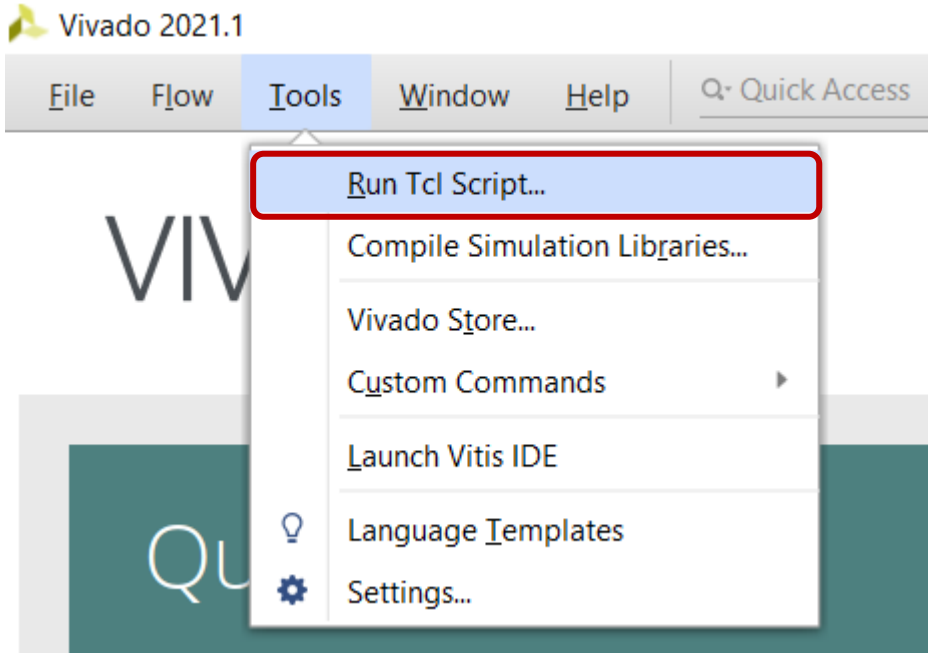




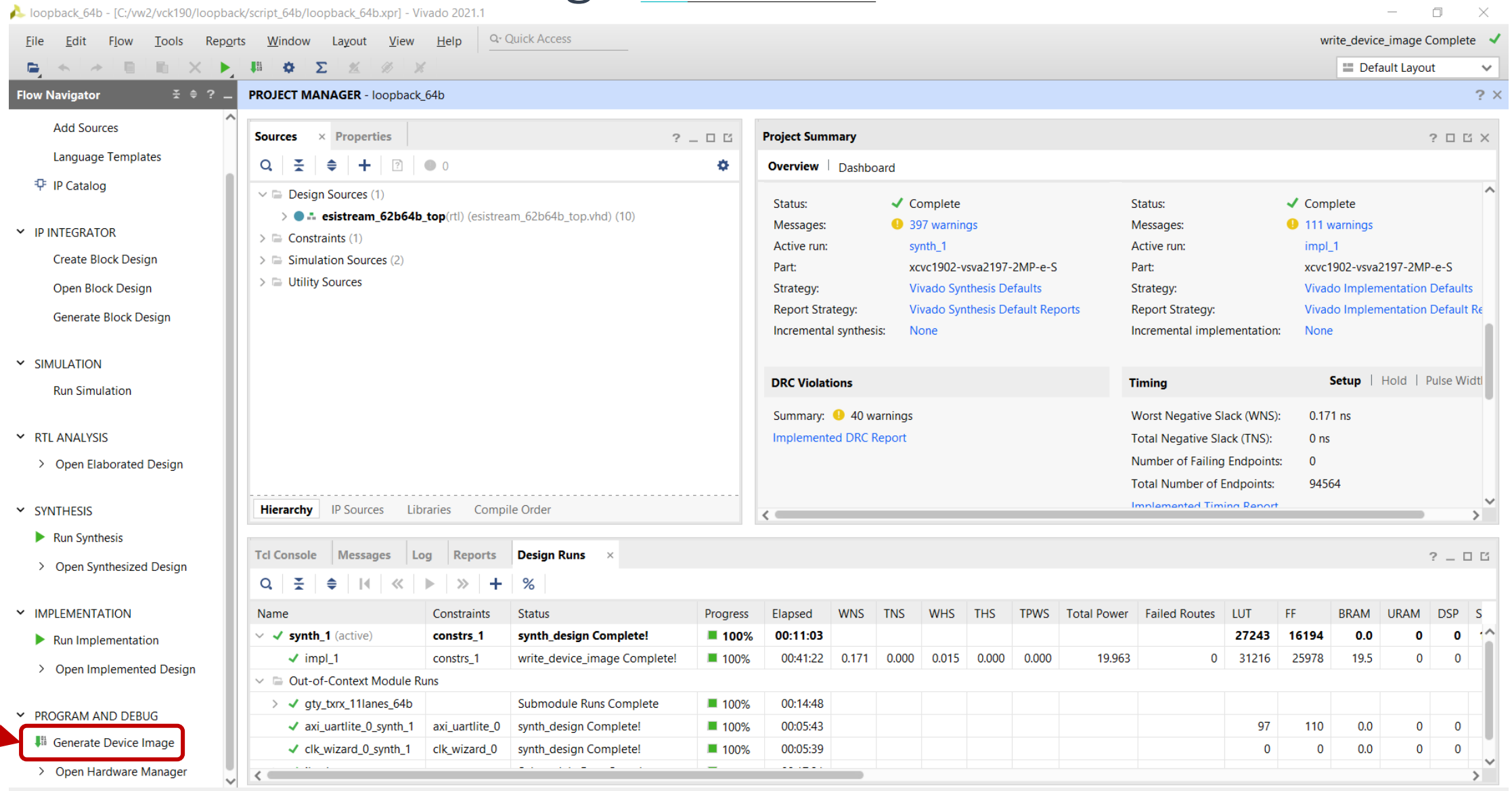
From Vivado project creation to link synchronization

Create Vivado project

- ❑ Open Vivado 2021.1
- ❑ Click on Tools > Run Tcl Script…
- ❑ Launch *script_64b.tcl* available in *loopback/* package directory.
- ❑ Once project creation has completed…
- ❑ Click on *Recent projects* view to open it.

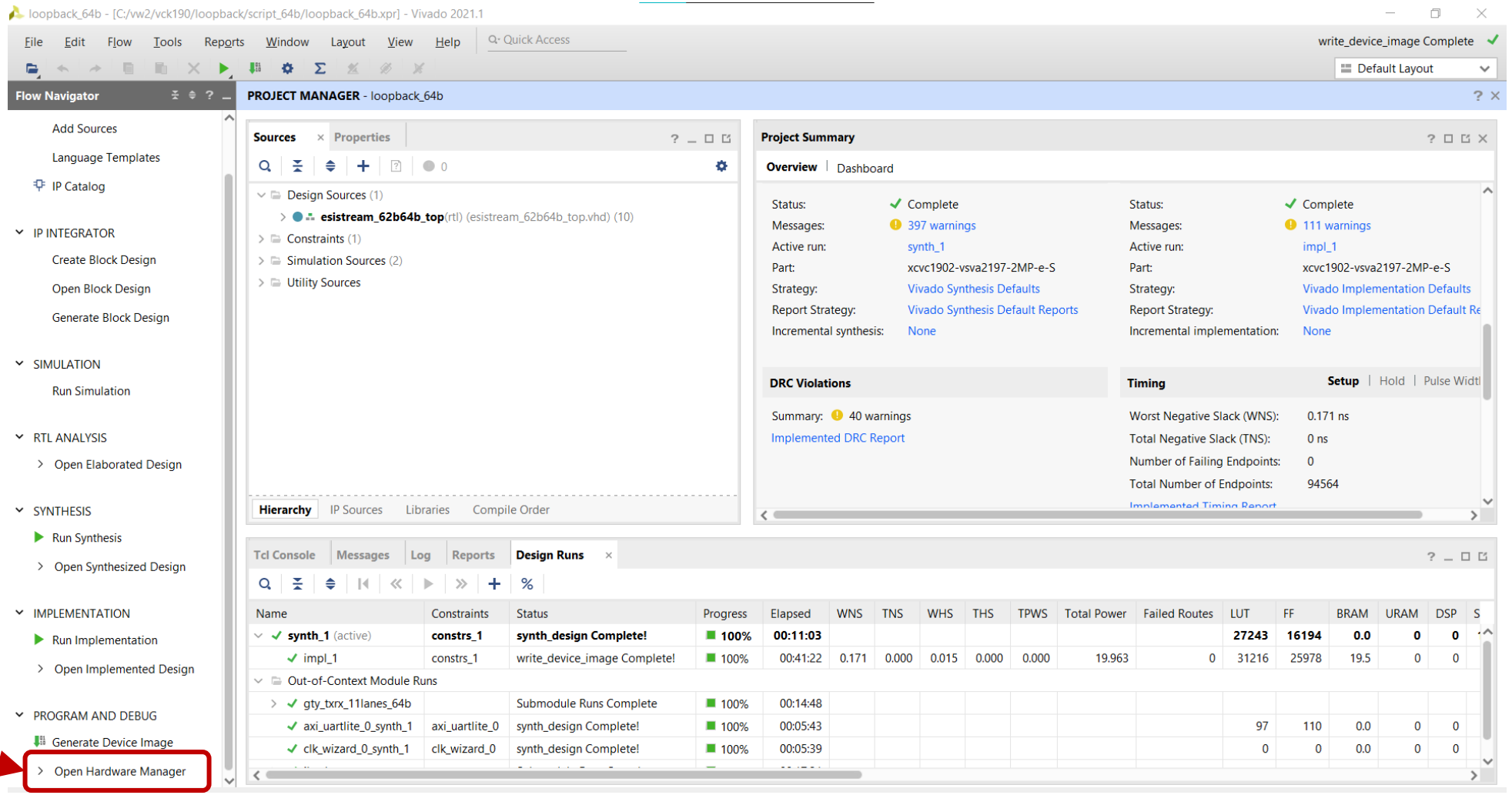


Generate Device image



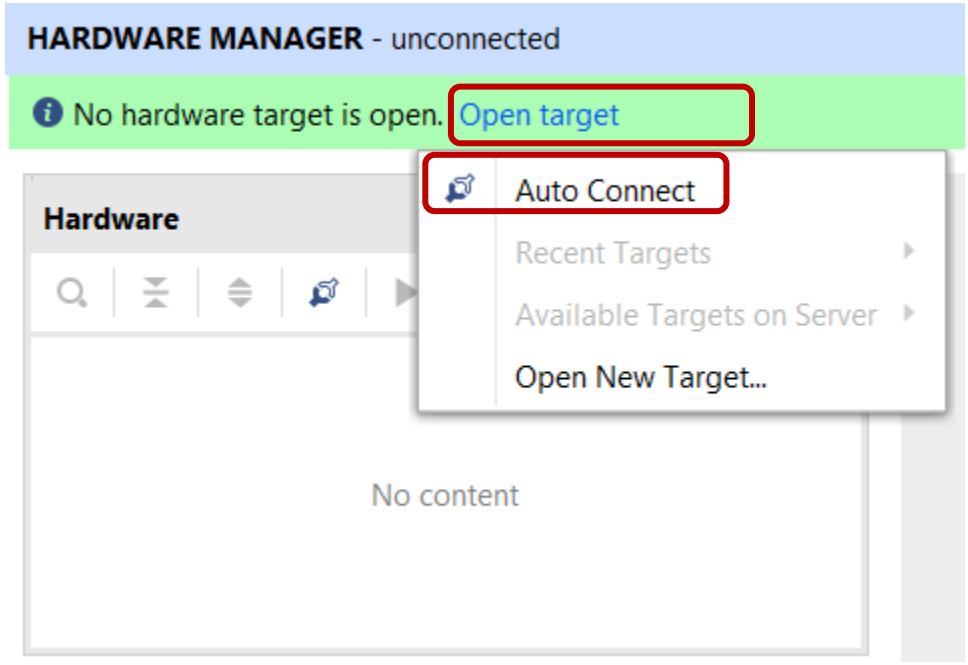
Load Device image

Open hardware Manager



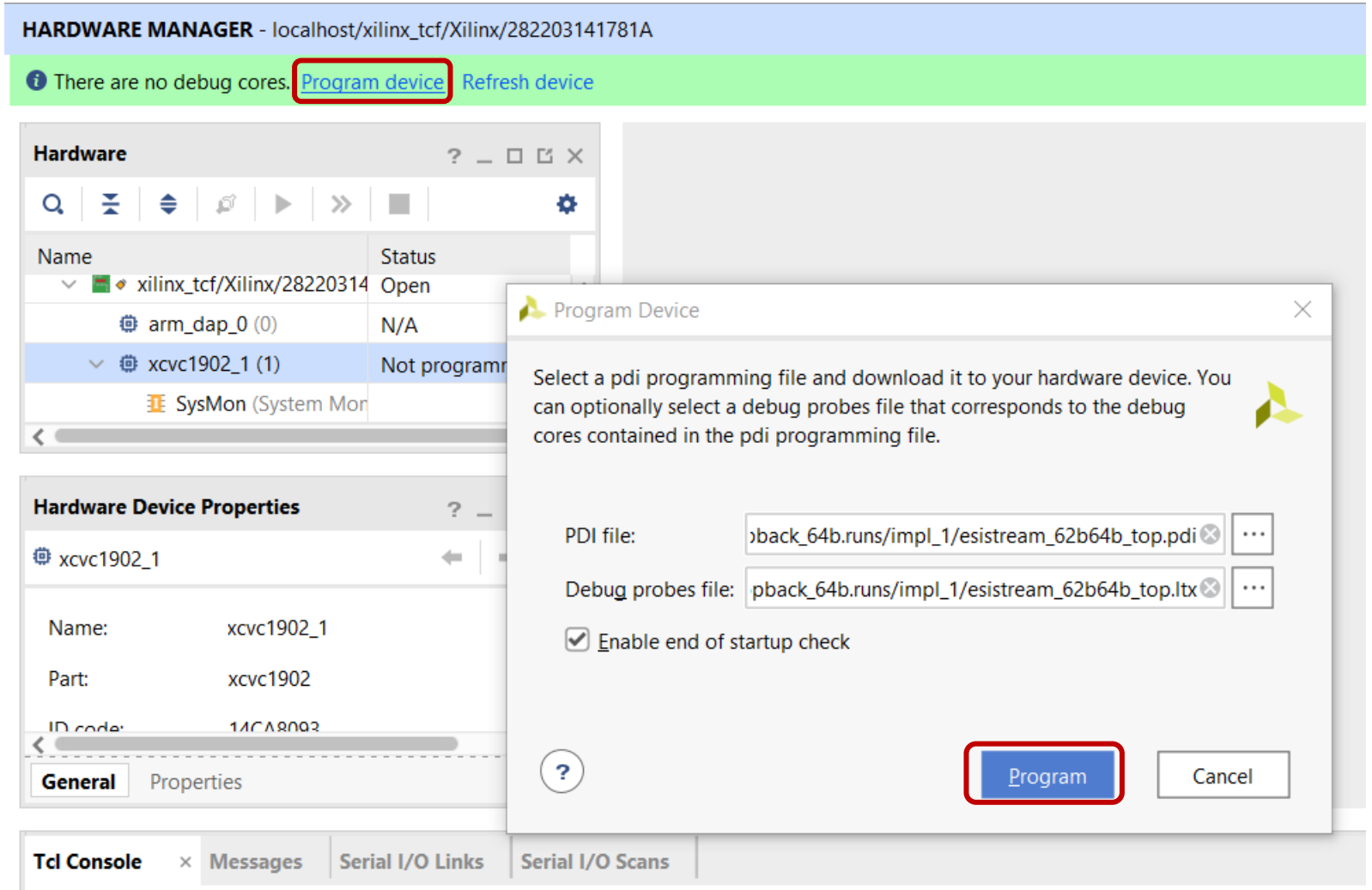
Load Device image

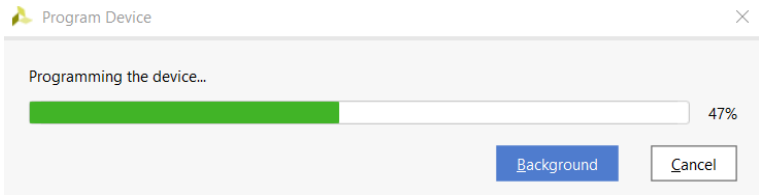
Click on Open target and Auto Connect



Load Device image

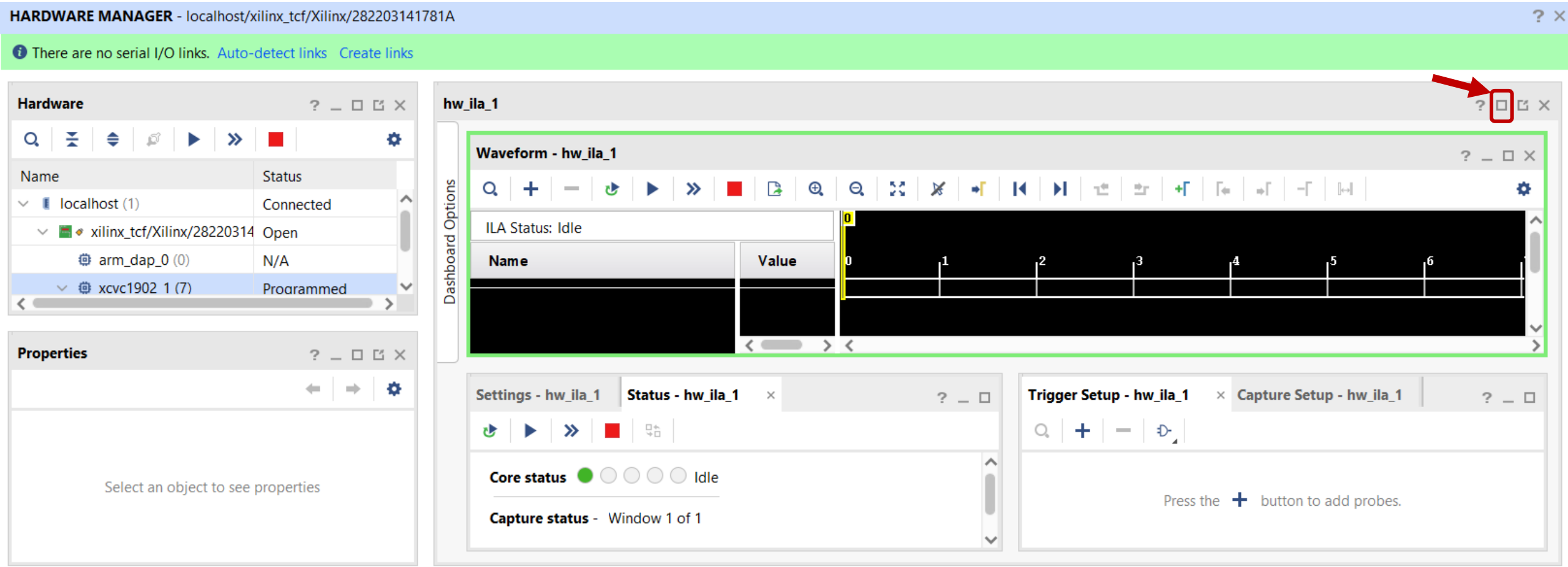
Click on Program device and Program





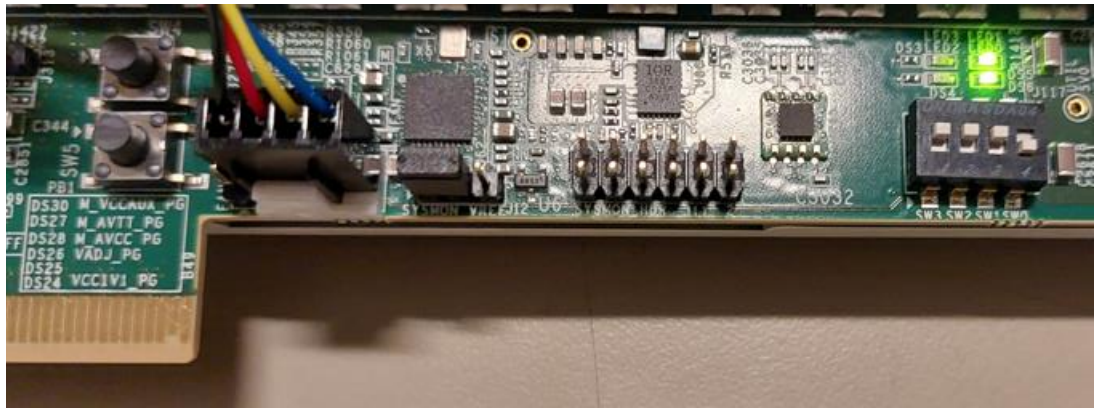
Start link synchronization

Maximize ILA window

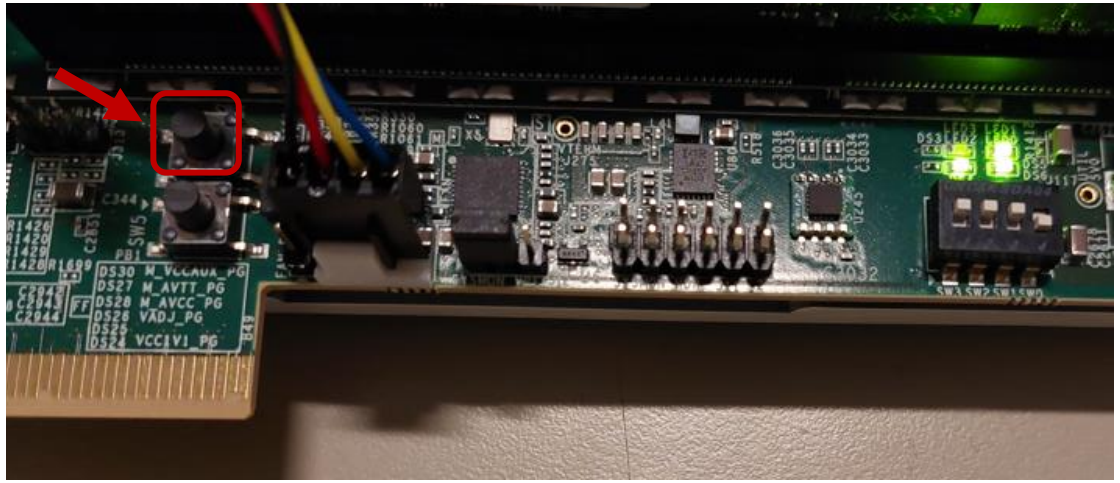


Start link synchronization

- Check UART is ready and ESIstream TX and RX transceivers are ready: LED0 ON and LED 1 ON



- Push SYNC button (PB0)
  - The loopback communication between ESIstream TX and RX through the loopback board has started.
- Check RX LANES are ready, LED2 ON.
- Check there is no communication error, LED3 OFF.
  - If one bit error is detected on one HSSL then LED3 is turned ON.

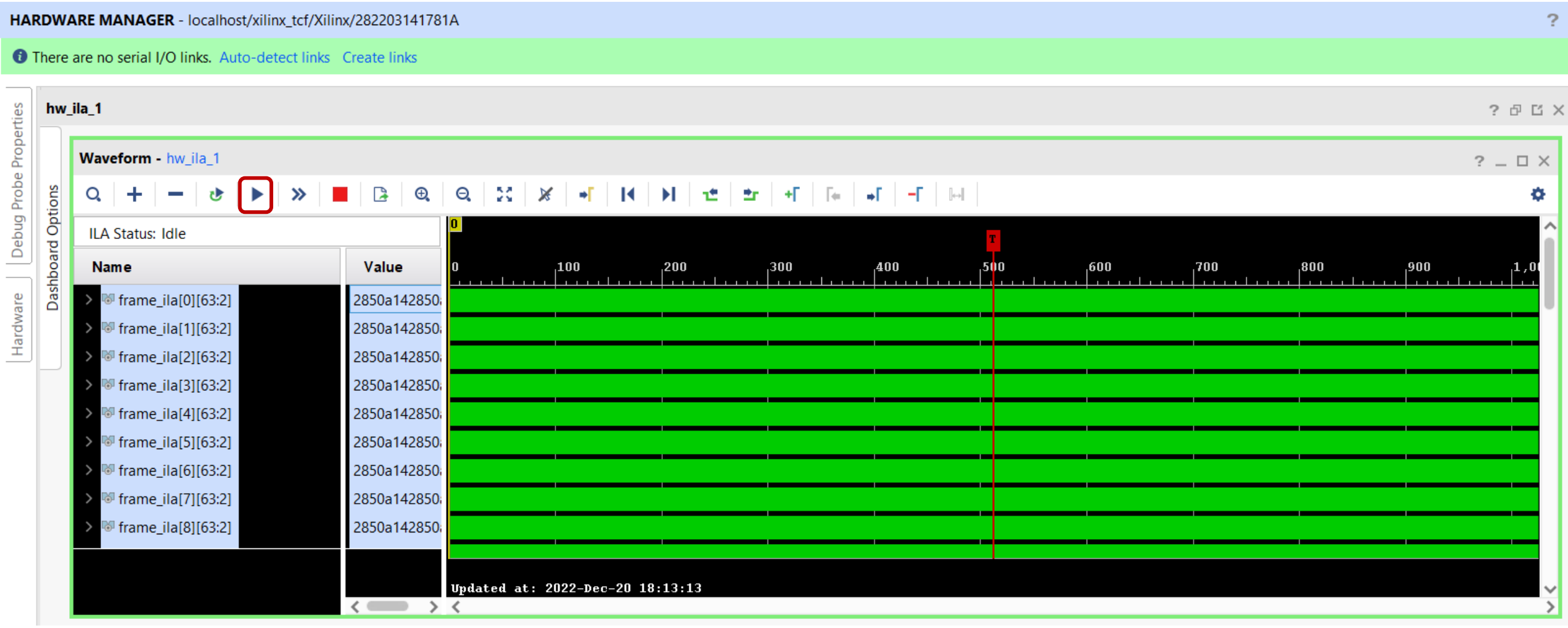


Acquire RX data (62-bit)

Using Vivado Integrated Logic Analyzer (ILA)

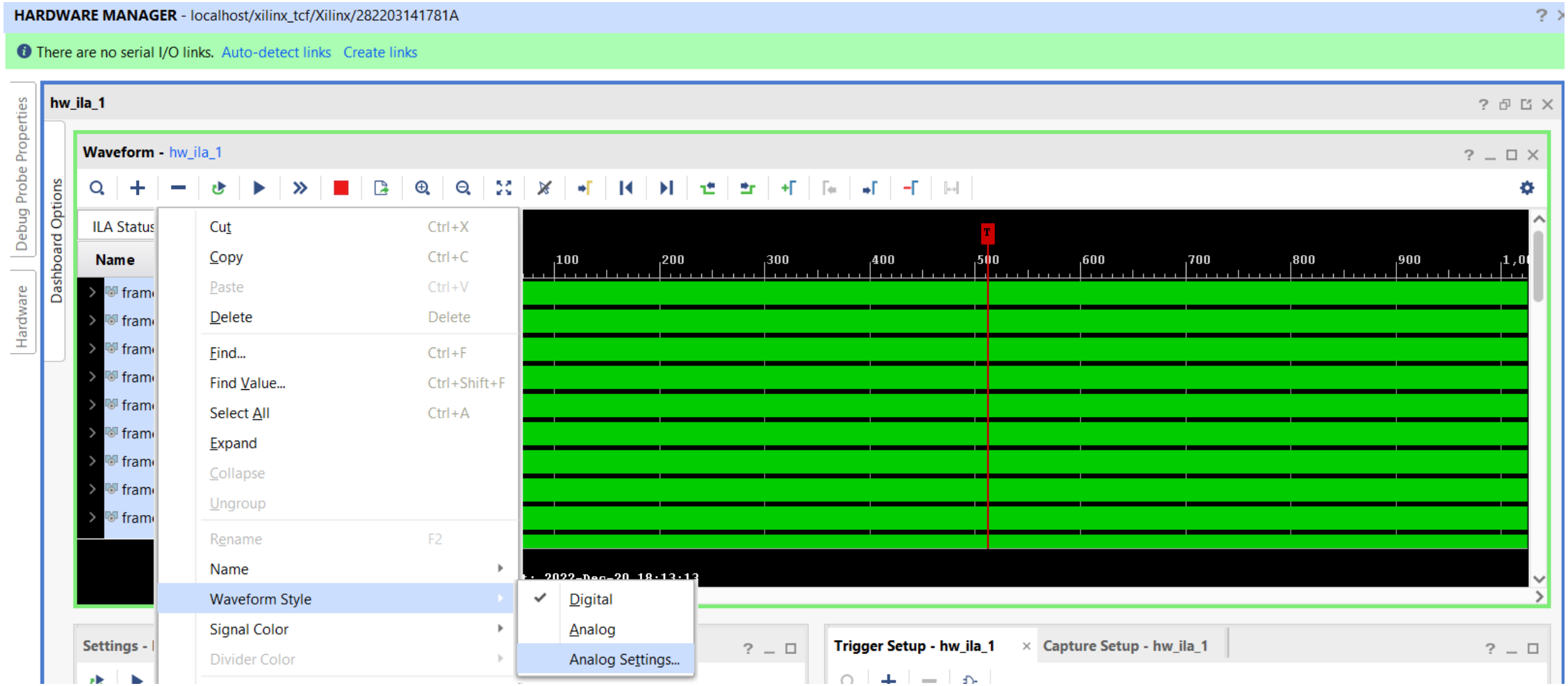
Acquire data in Vivado using JTAG and ILA.

In hardware manager, click on play button.



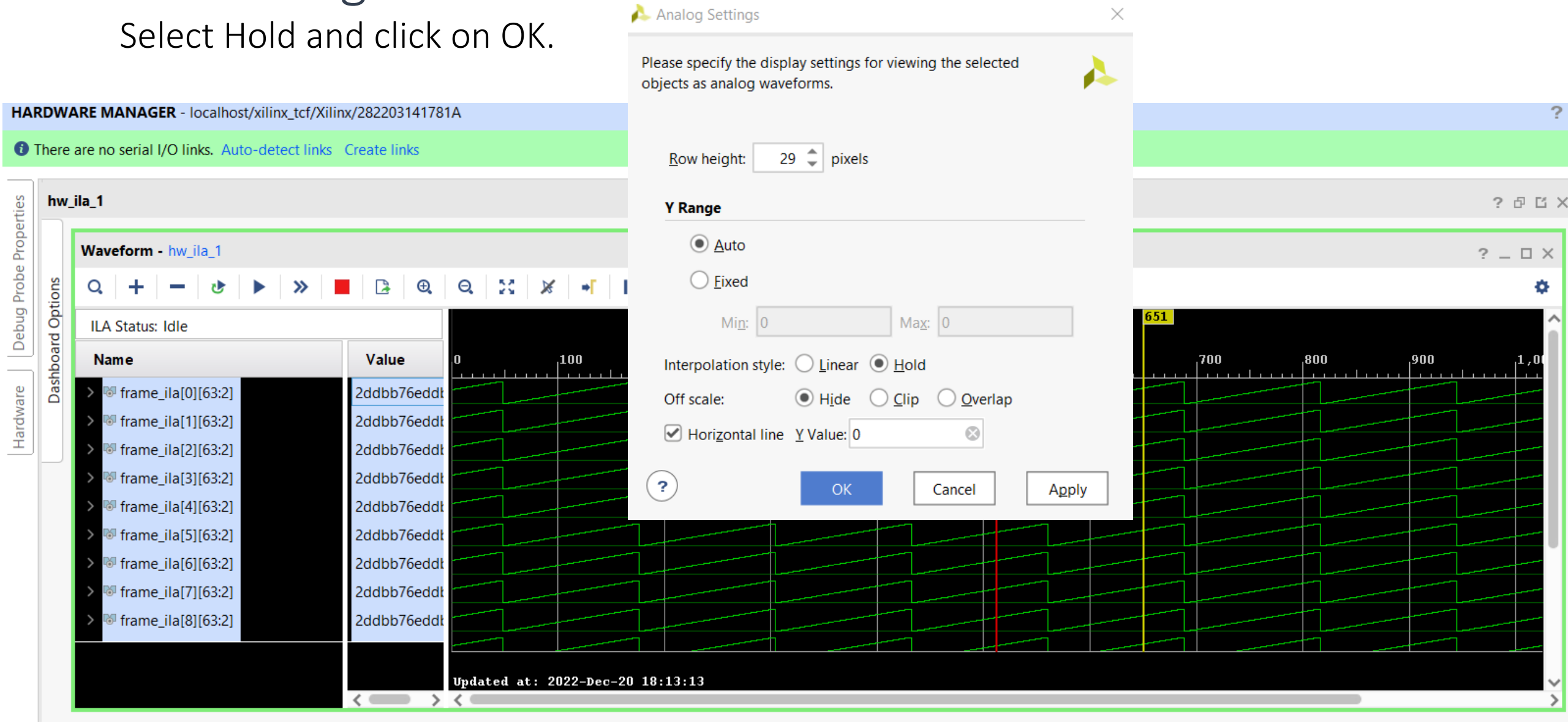
ILA Analog view

Select all lanes + right-click, then click on Waveform > Analog Settings…



ILA Analog view

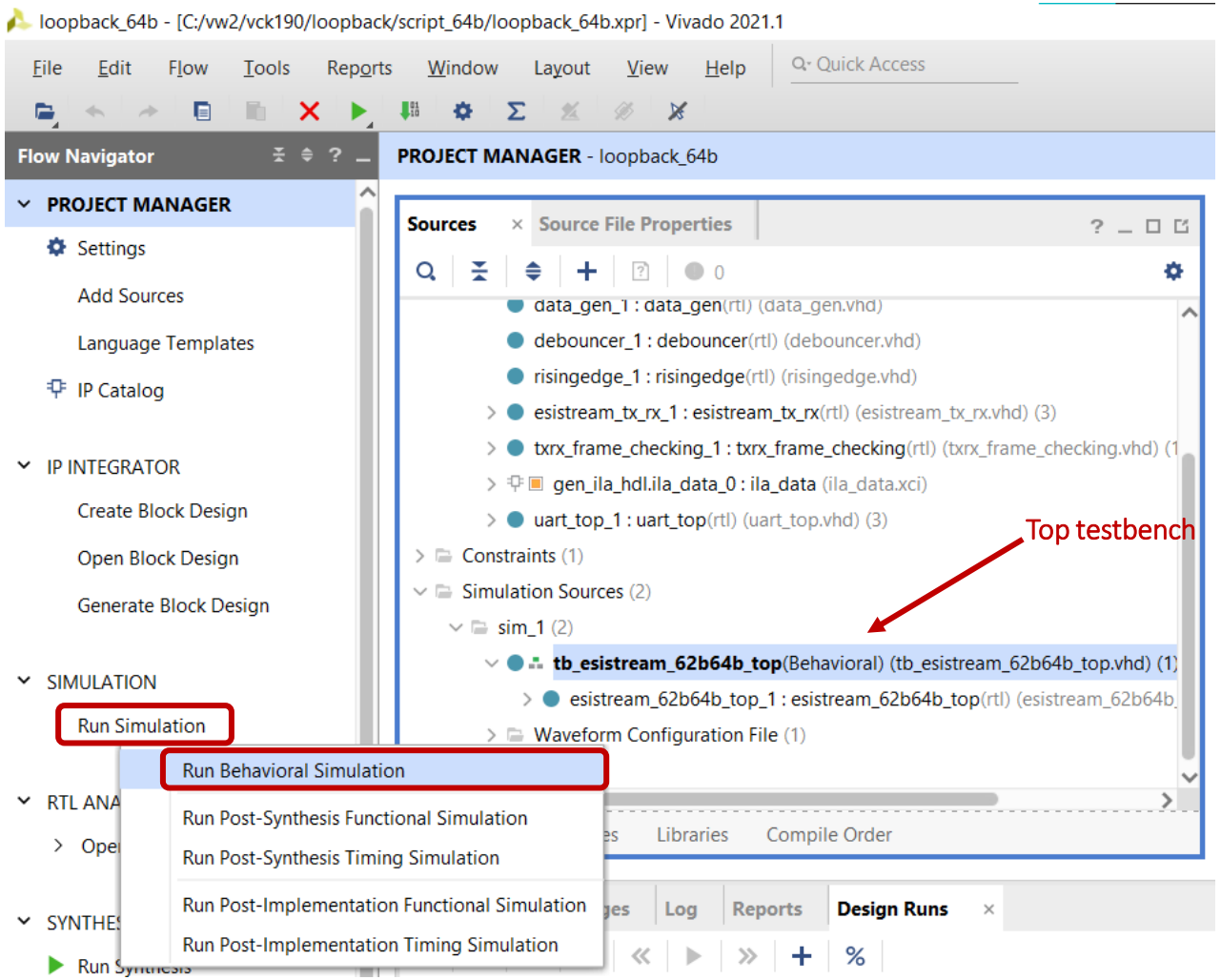
Select Hold and click on OK.



Design example top file testbench simulation

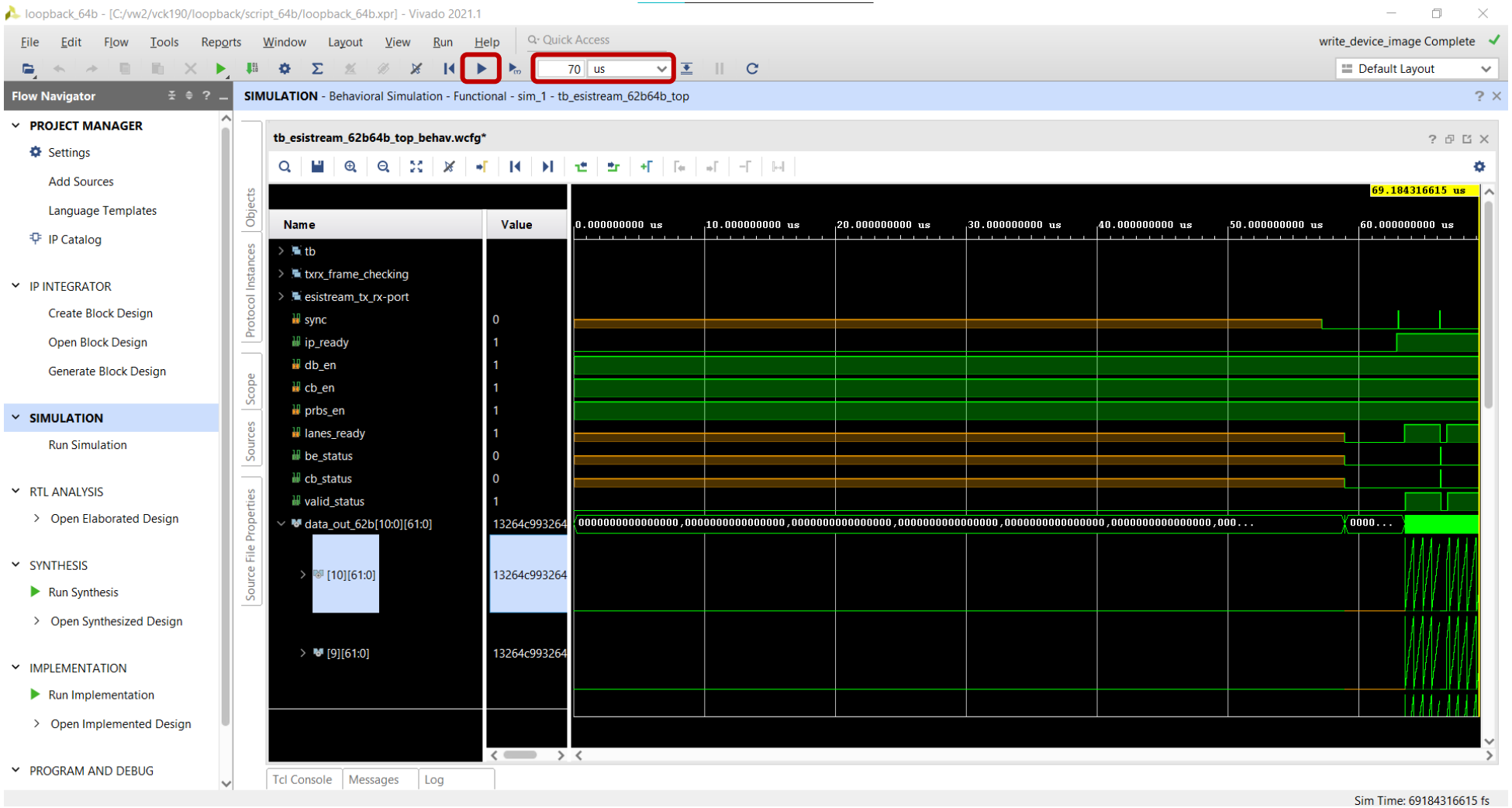
Launch Vivado Simulator

Click on *Run simulation* and *Run Behavioral Simulation* to simulate the whole design.

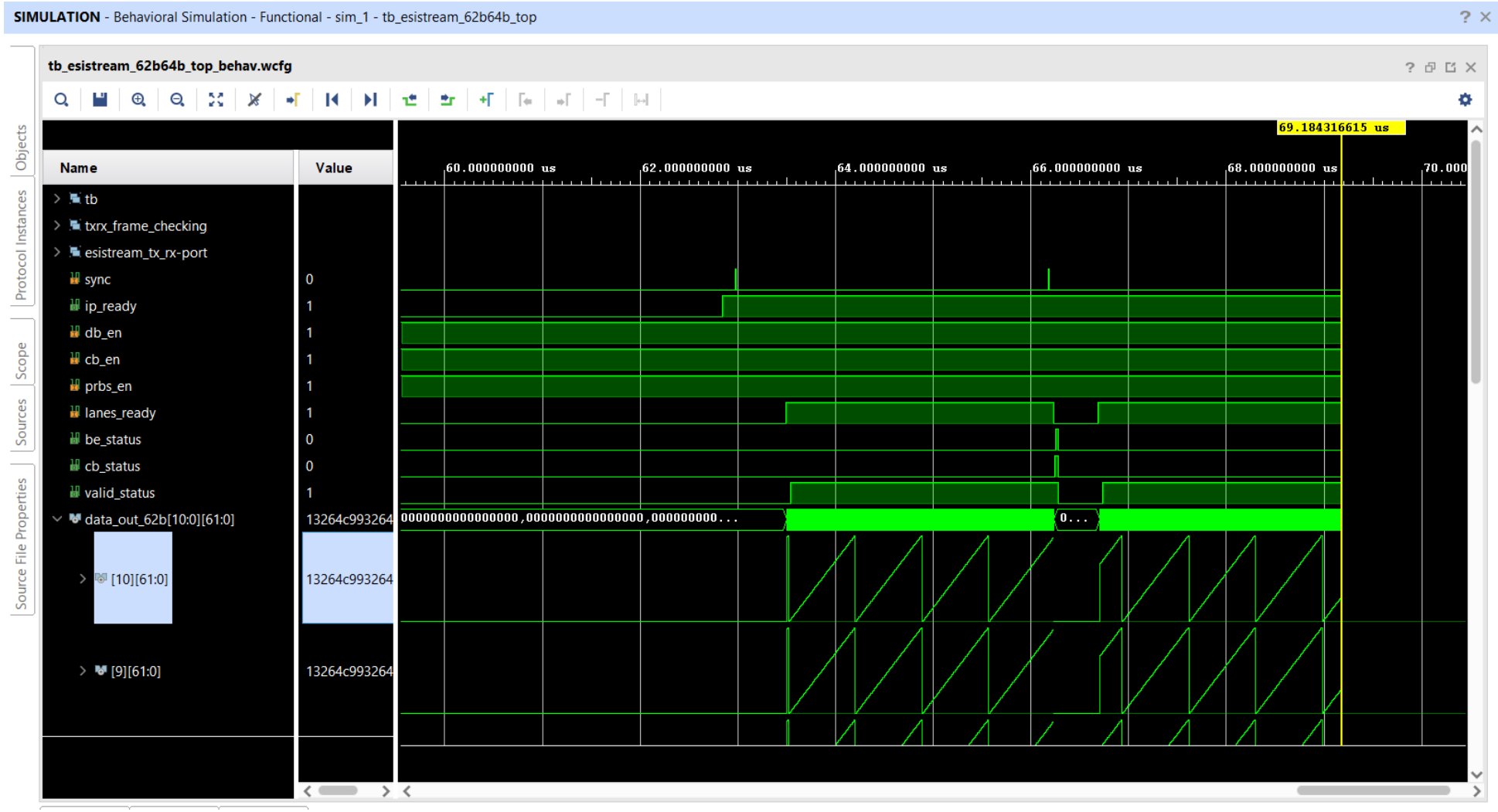


Run the simulation for 70 µs

Click on Play button.



Run the simulation for 70 µs (zoom)



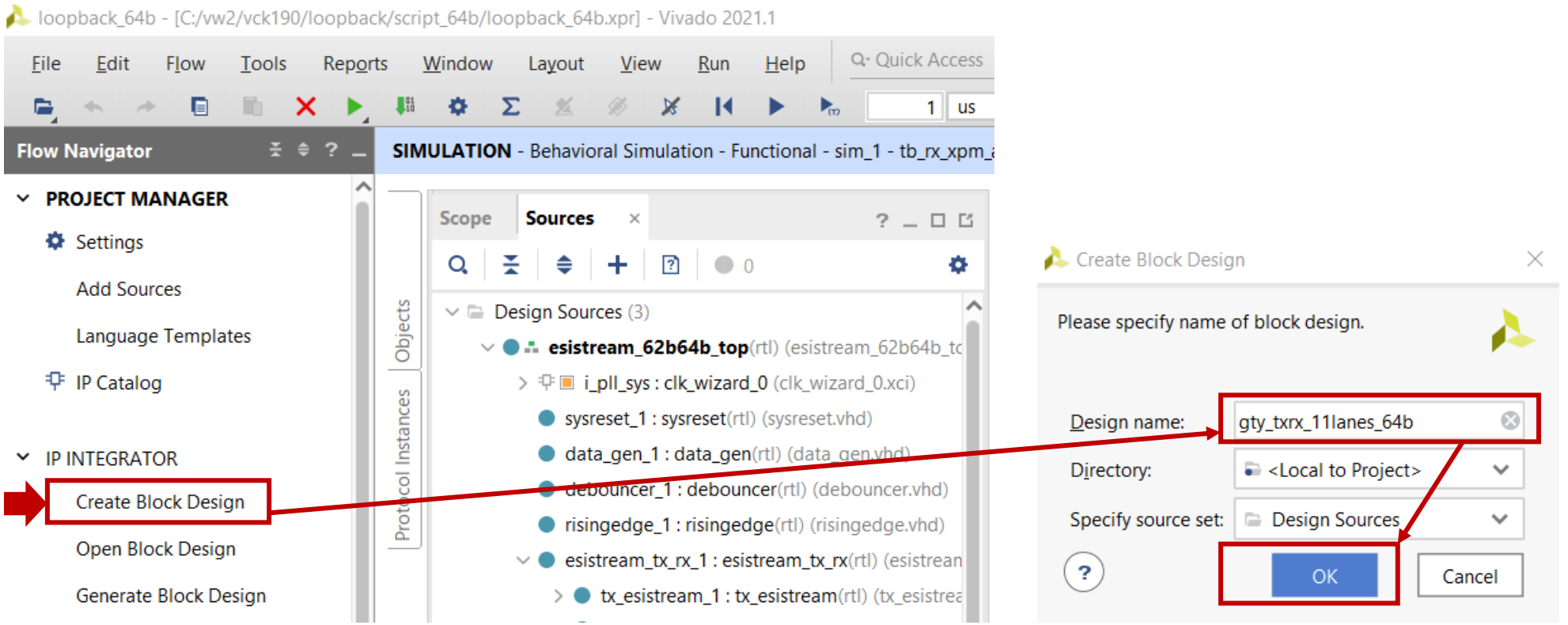
Application Note:

Create an ESIstream Gigabit Transceivers Block Design (.bd) from scratch on Versal.

ESIstream
The Efficient Serial Interface

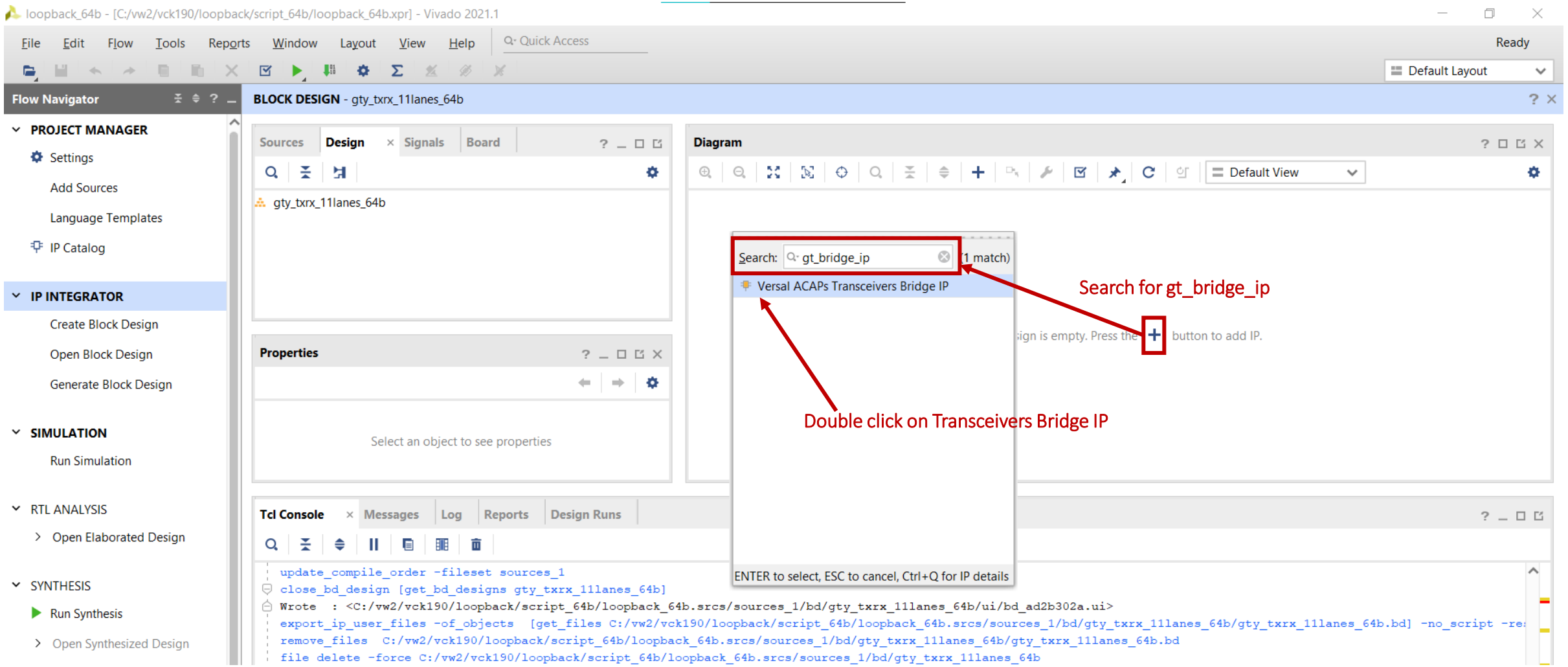
Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.



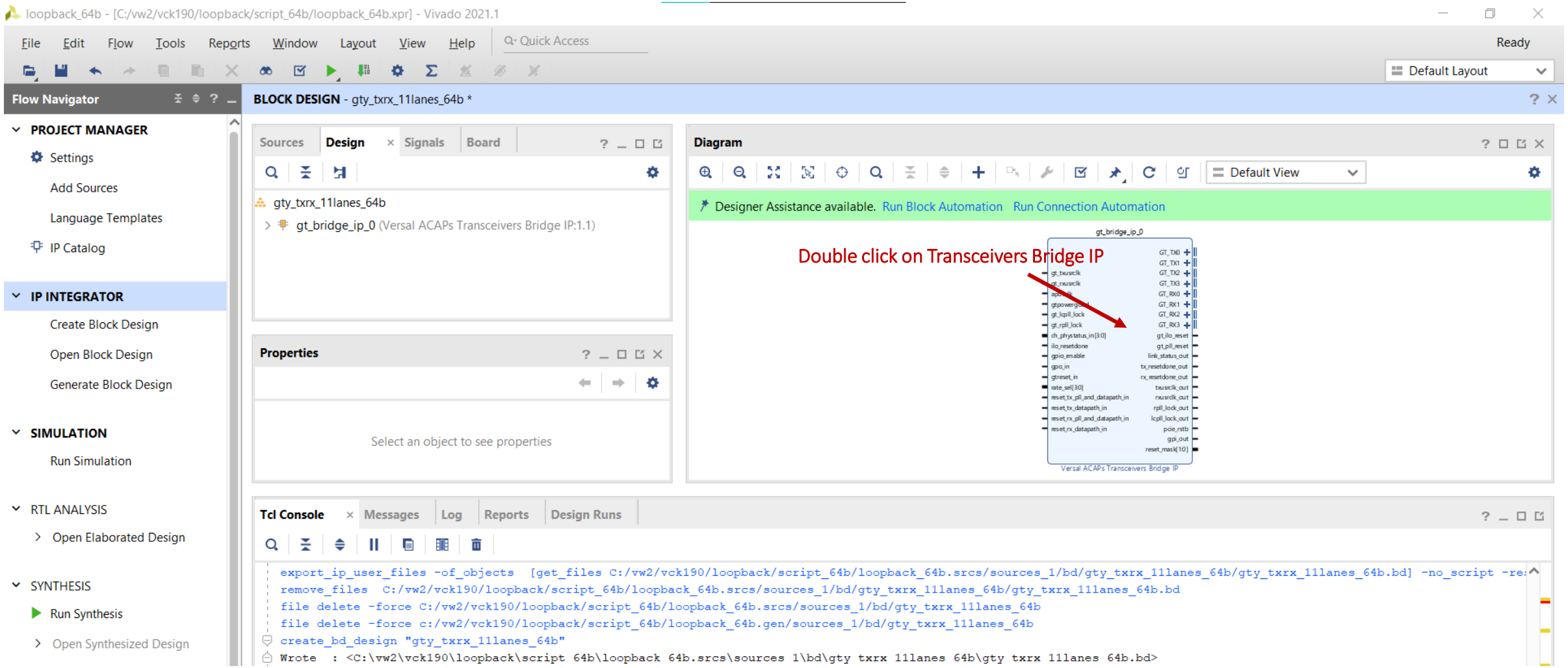
Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.



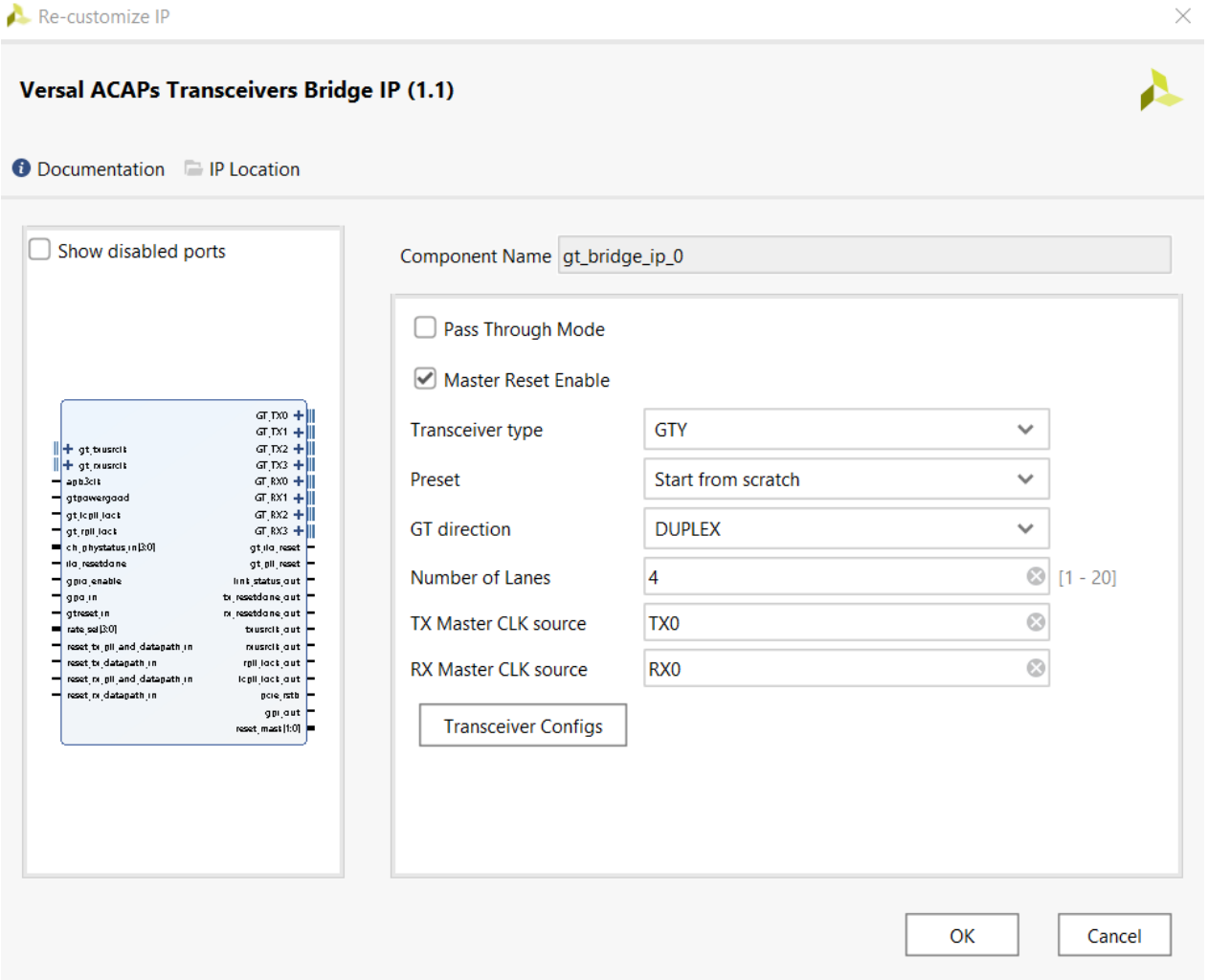
Application Note

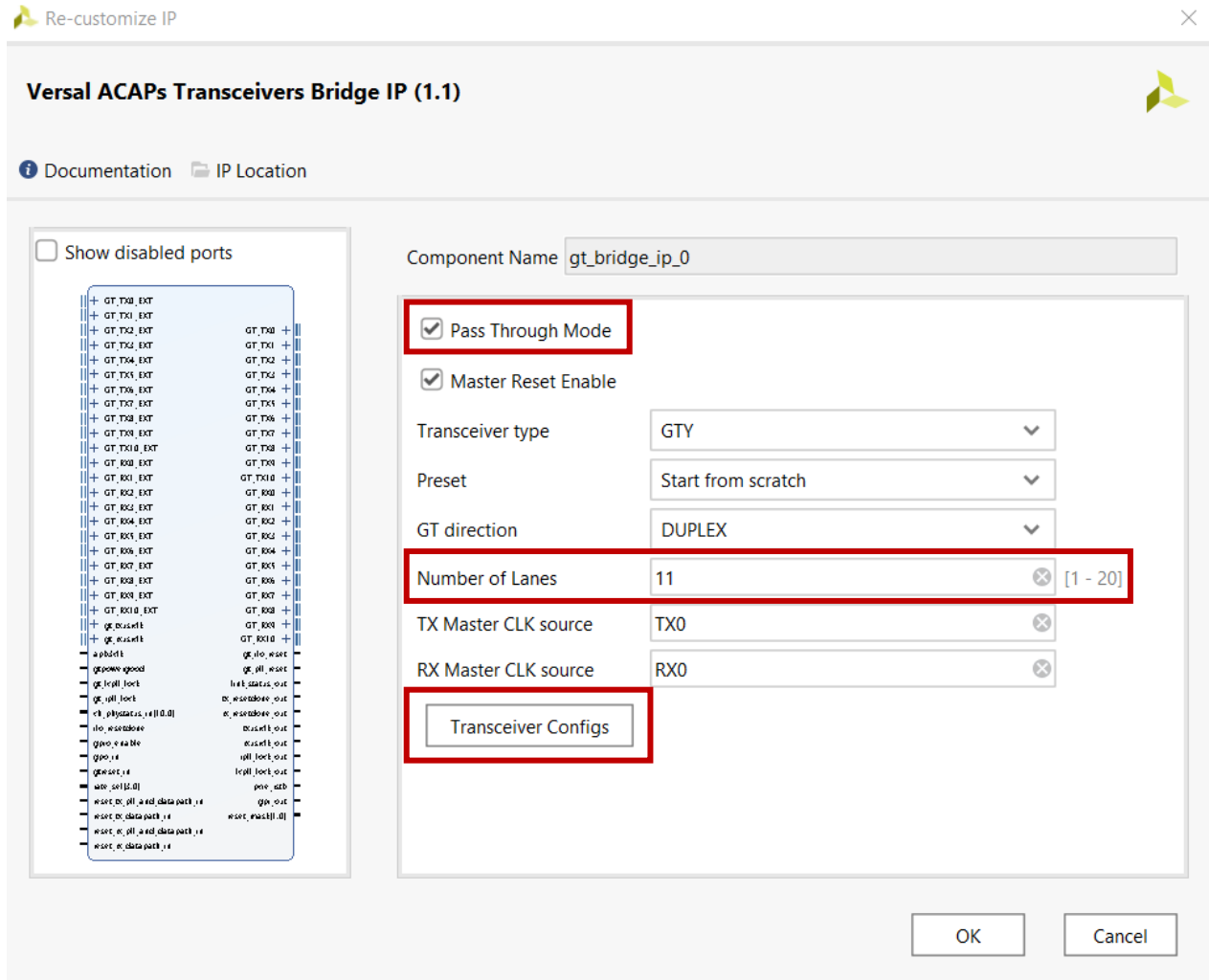
Create ESIstream Gigabit transceivers block design (.bd) from scratch.



Application Note

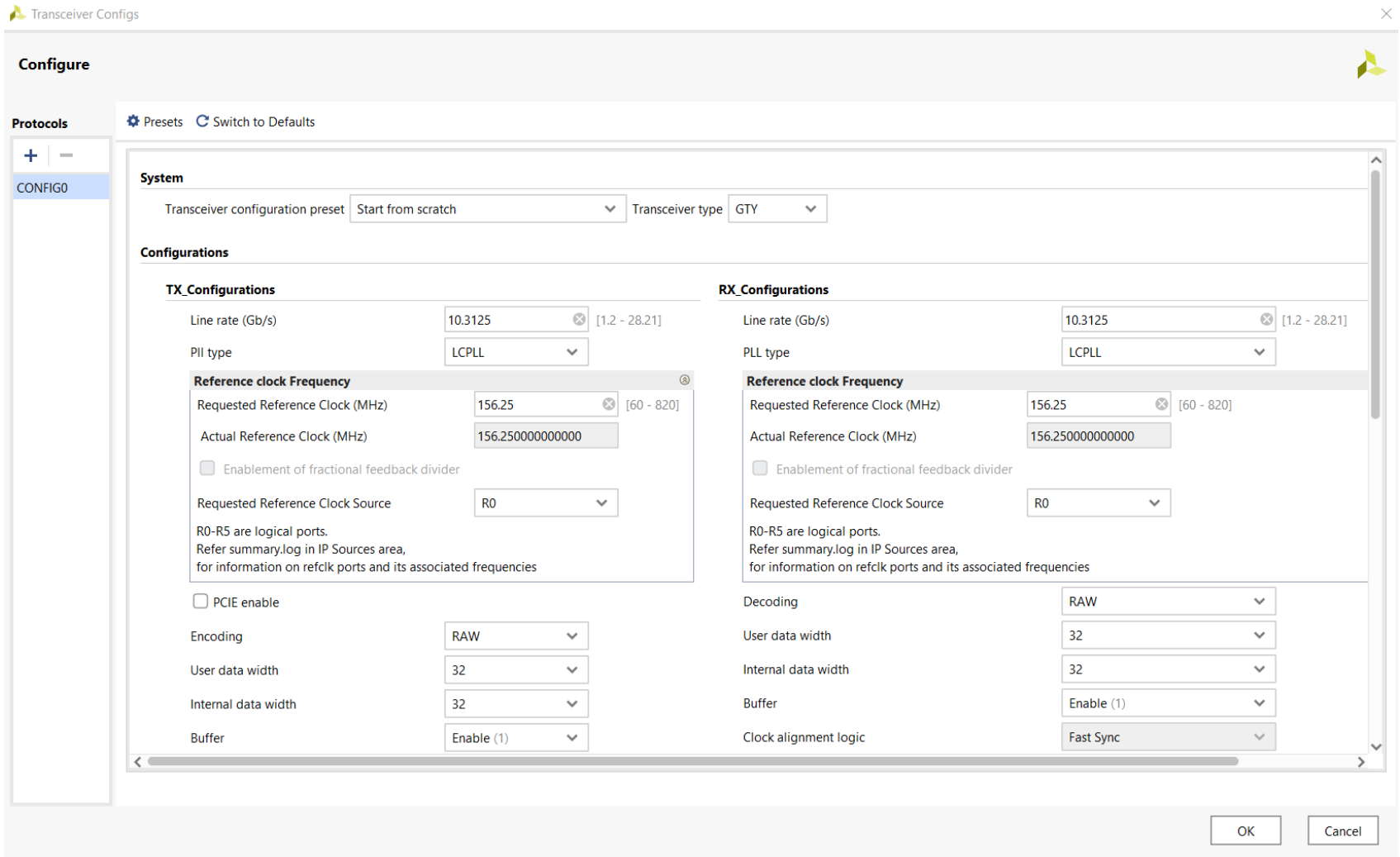
Create ESIstream Gigabit transceivers block design (.bd) from scratch.





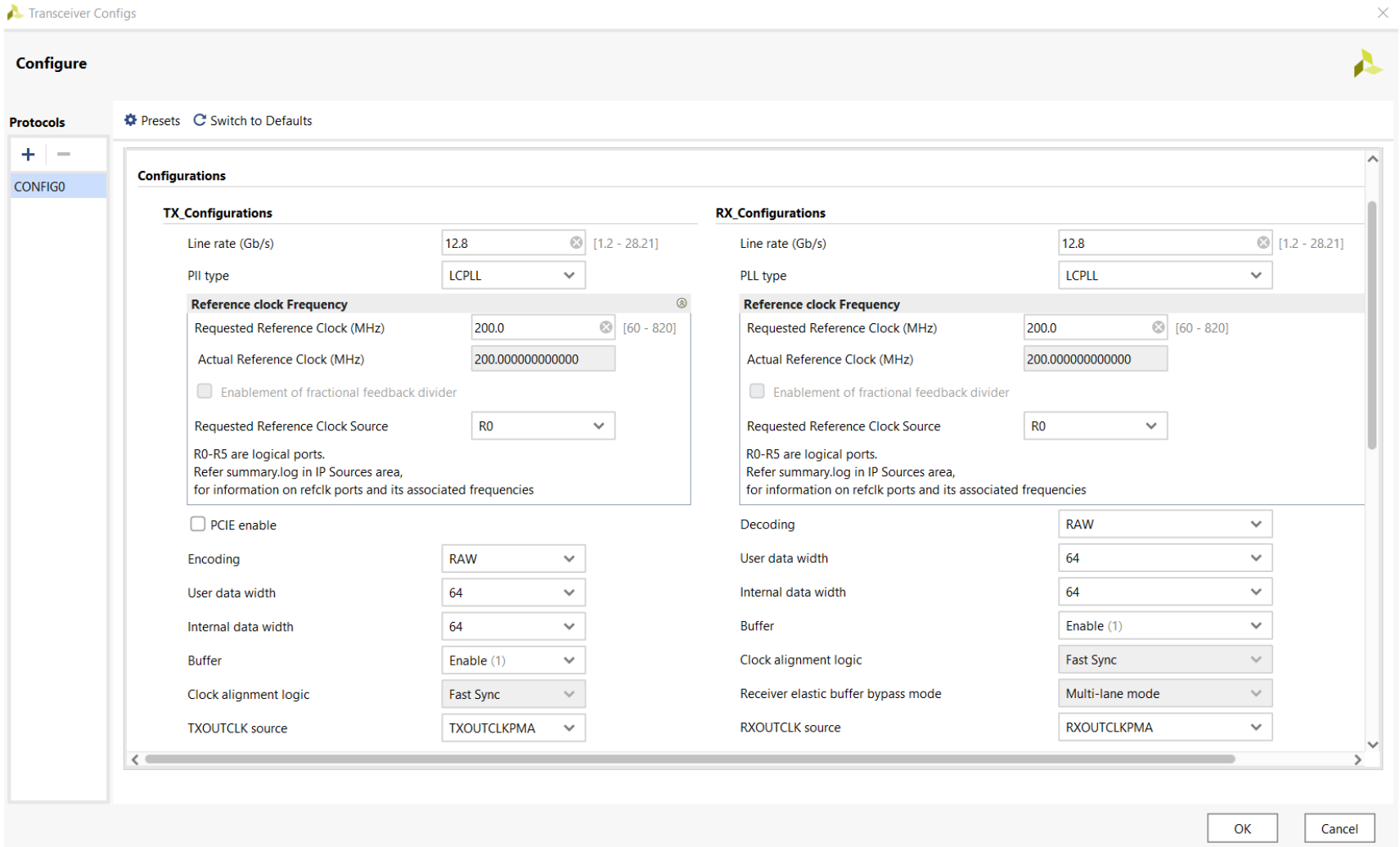
Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.



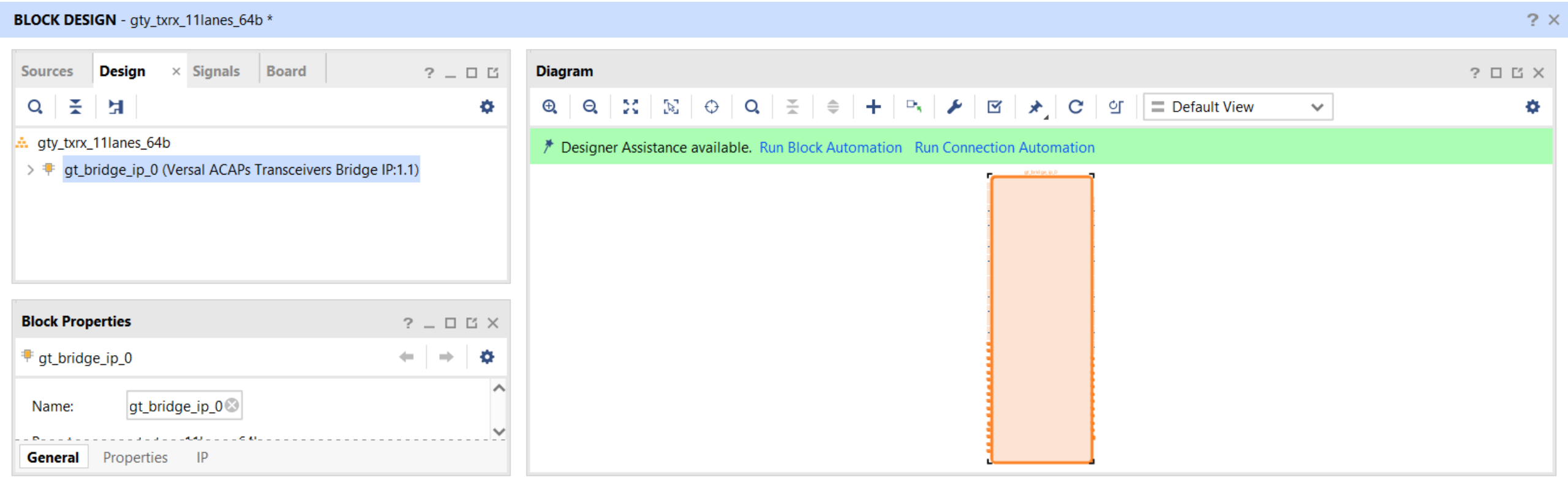
Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.



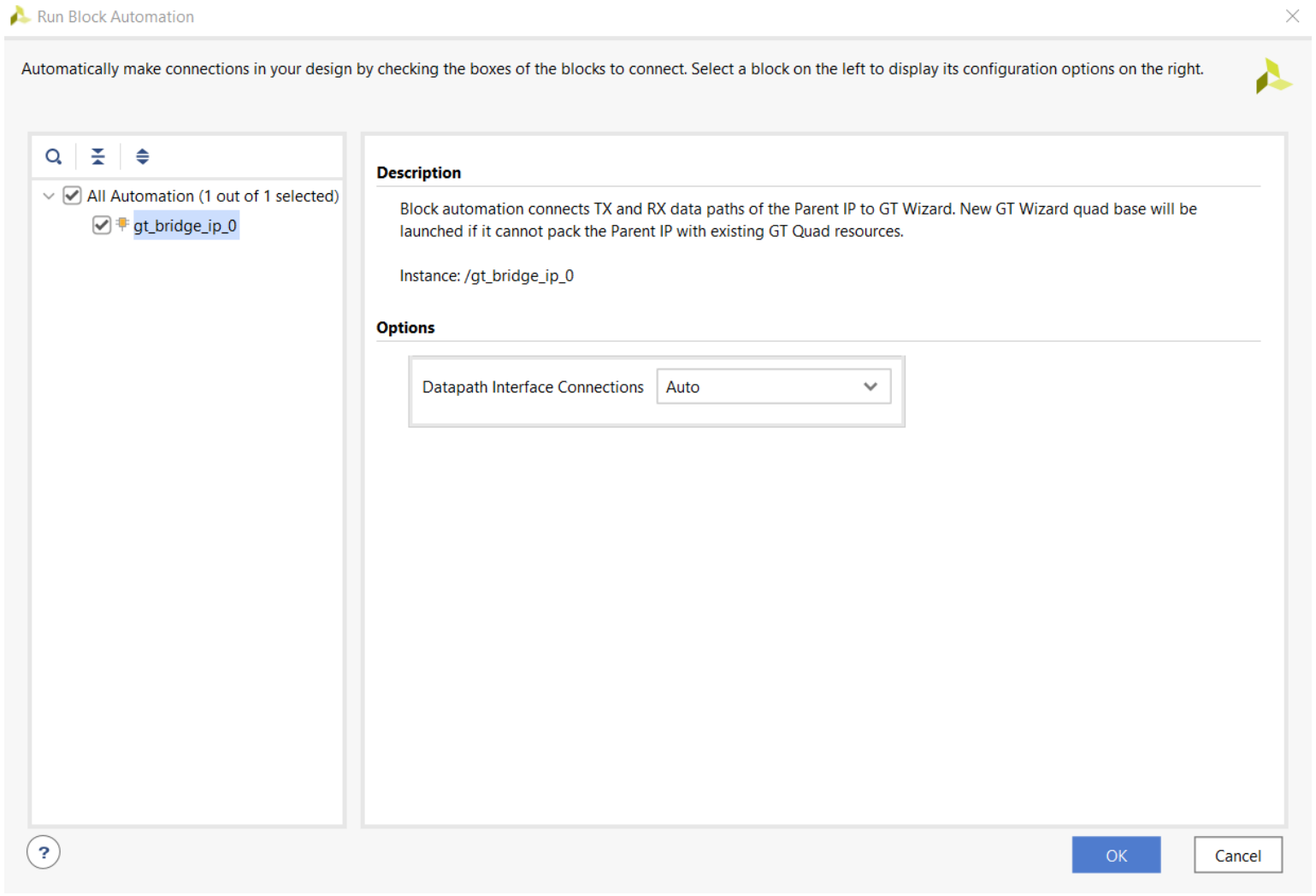
Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.



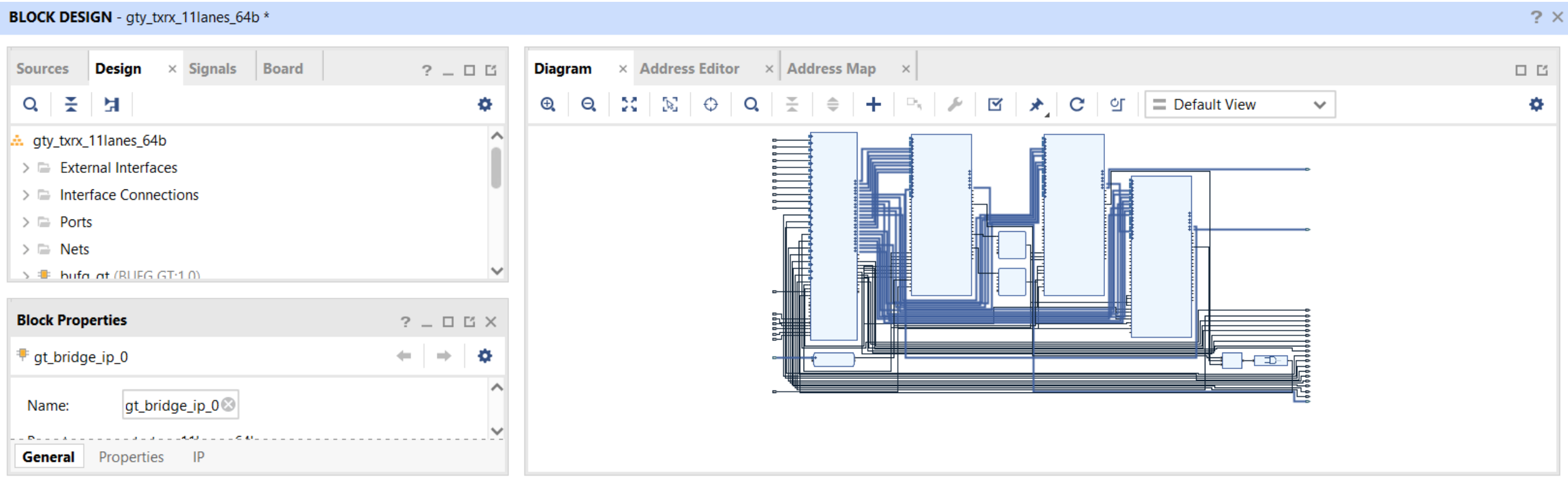
Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.



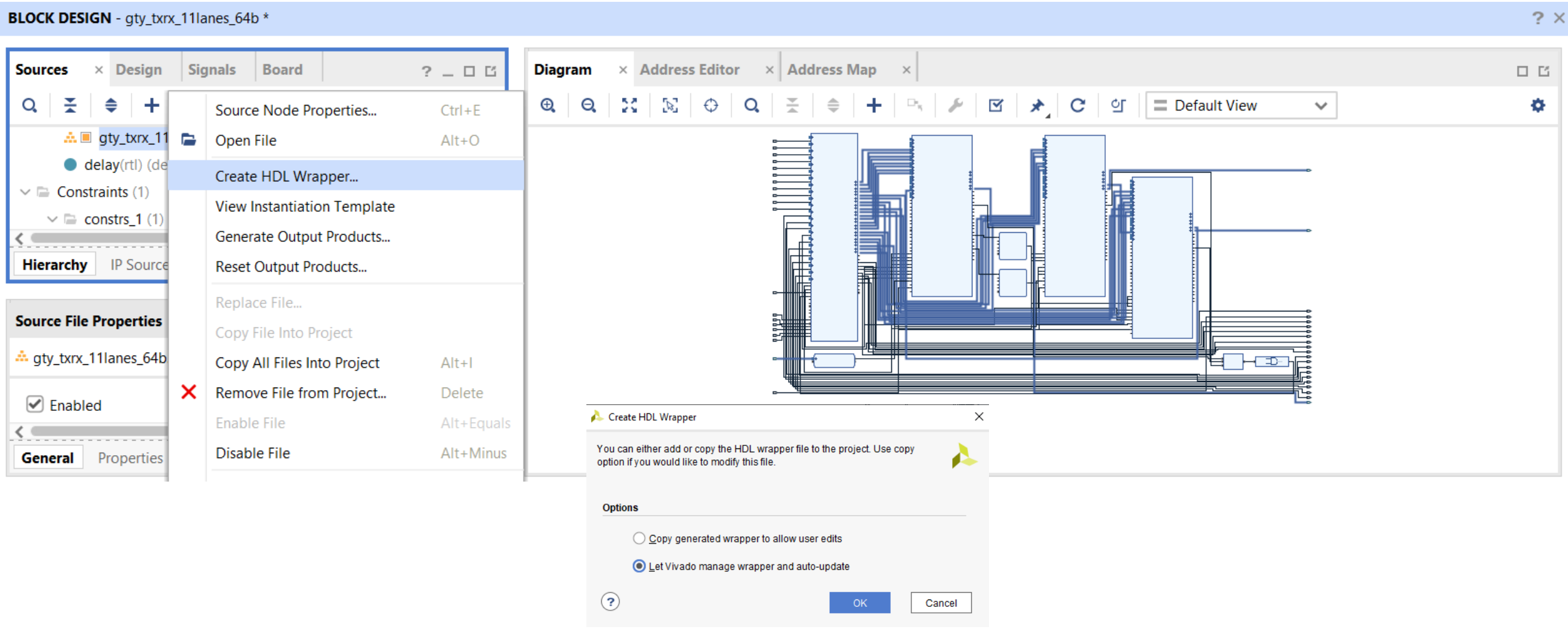
Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.



Application Note

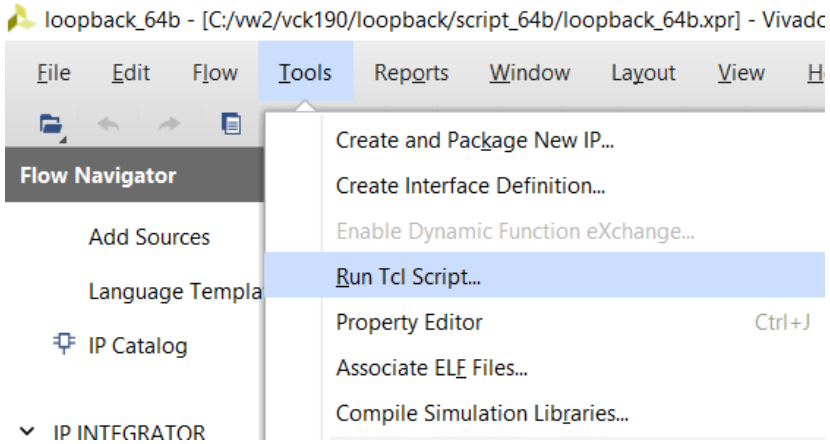
Create ESIstream Gigabit transceivers block design (.bd) from scratch.

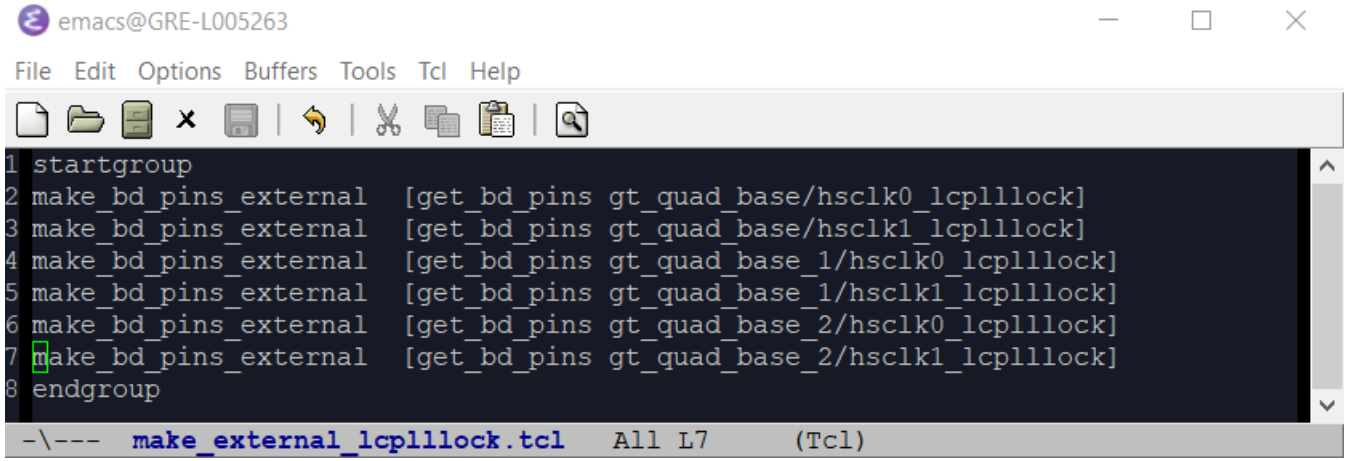


Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.

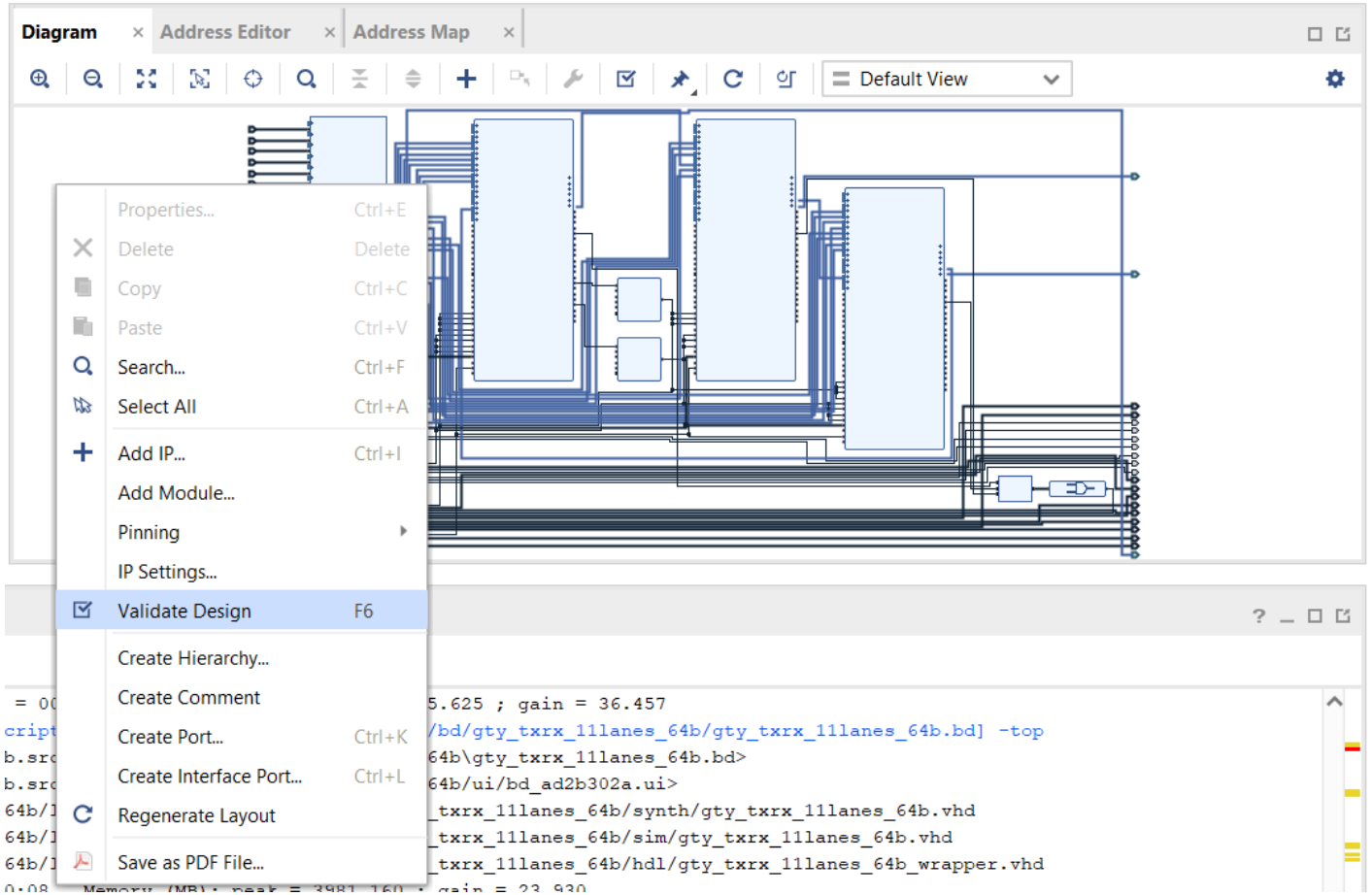
- ❑ Make external hsclk0_lcplllock, hsclk1_lcplllock signals from gt_quad_base, gt_quad_base_1 and gt_quad_base_2
- ❑ Tools > Run Tcl Script…
- ❑ Launch make_external_lcplllock.tcl
- ❑ The Validate design and Create HDL wrapper…

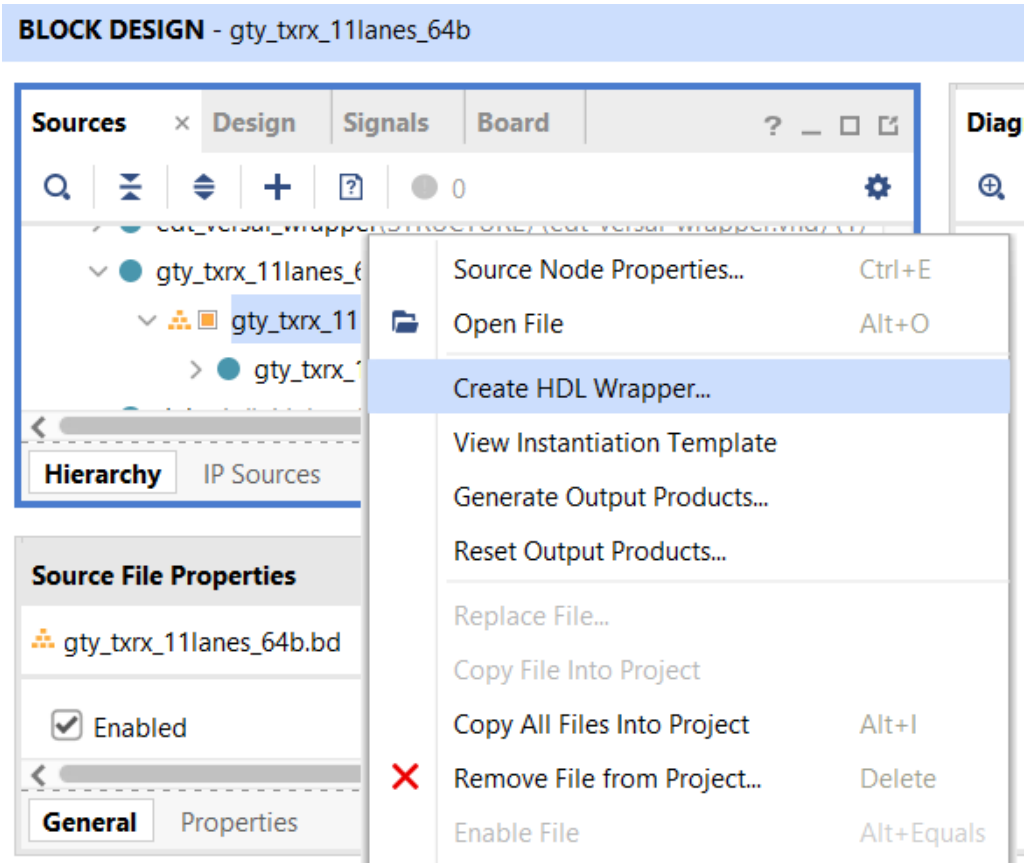




Application Note

Create ESIstream Gigabit transceivers block design (.bd) from scratch.





UART

UART - FPGA interface protocol

Write command

- The design embeds a UART slave which uses the following configuration:
  - Baud rate: 115200
  - Data Bits: 8
  - No parity

The UART frames layer protocol defined here allows to perform read and write operations on the registers listed in the register map.

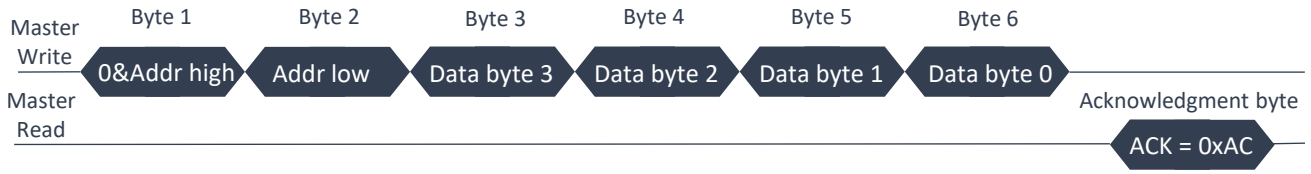
- **Register Write operation:**

The UART master must send the data in the order described on the figure below to be able to write a FPGA register.

Firstly, the master send the 15-bit register address and then the 32-bit data word.

- The **most significant bit of the first transmitted byte (bit 7) must be set to 0 for write operation.**
- The bits 6 down to 0 of the first transmitted byte contain the bit 14 down to 8 of the register address.
- The second byte contains the bit 7 down to 0 of the register address.
- The third byte contains the bit 31 down to 24 of the register data.
- The fourth byte contains the bit 23 down to 16 of the register data.
- The fifth byte contains the bit 15 down to 8 of the register data.
- The sixth byte contains the bit 7 down to 0 of the register data.

Finally, the master read the acknowledgment word to check that the communication has been done correctly. The acknowledgment word is a single byte of value 0xAC (172 is the decimal value).



UART - FPGA interface protocol

Read command

- The design embeds a UART slave which uses the following configuration:
  - Baud rate: 115200
  - Data Bits: 8
  - No parity

The UART frames layer protocol defined here allows to perform read and write operations on the registers listed in the register map.

- **Register Read Operation**

The UART master must send the data in the order described on the figure below to be able to read a FPGA register value. Firstly, the master send the 15-bit register address.

- The **most significant bit of the first transmitted byte (bit 7) must be set to 1 for read operation.**
- The bits 6 down to 0 of the first transmitted byte contain the bit 14 down to 8 of the register address.
- The second byte contains the bit 7 down to 0 of the register address.

Then, the master read the data and the acknowledgment word to check that the communication has been done correctly. The acknowledgment word is a single byte of value 0xAC (172 is the decimal value).

- The third byte contains the bit 31 down to 24 of the register data.
- The fourth byte contains the bit 23 down to 16 of the register data.
- The fifth byte contains the bit 15 down to 8 of the register data.
- The sixth byte contains the bit 7 down to 0 of the register data.