ESIstream

The Efficient Serial Interface

ESIstream 14B16B

MICROCHIP FPGA MPF300T-1FCG115E2

USER GUIDE

ESIstream is a license-free high efficiency serial interface protocol based on 14b/16b encoding.

Its main benefits are low overhead and ease of hardware implementation.

INTRODUCTION

The package ESIstream allows to generate VHDL design examples to get started with ESIstream 14B16B High-speed serial interface.

This package provides VHDL sources, constraint files, TCL scripts for project generation, VHDL testbench files for simulation and python scripts.

For technical support, please get the team involved and contact us using ESIstream contact web page or at GRE-HOTLINE-BDC@Teledyne.com

TERMINOLOGY

ADC	Analog to Digital Converter
ASIC	Application-Specific Integrated Circuit
BE	Bit Error
CB	Clock Bit
CDR	Clock and Data Recovery
DAC	Digital to Analog Converter
ESIstream	the Efficient Serial Interface
ESS	ESIstream Synchronization Sequence
FAS	Frame Alignment Sequence
FPGA	Field Programmable Gate Array
GT	Gigabit Transceiver
HSSL	High Speed Serial Lane
ILA	Integrated Logic Analyzer (a Vivado feature)
LFSR	Linear Feedback Shift Register
PAS	PRBS Alignment sequence
PL	Programmable Logic
PLL	Phase Locked Loop
PRBS	Pseudo-Random Binary Sequence
RX	Receiver
SSO	Slow Synchronization Output. GT reference clock.
TX	Transmitter
UART	Universal Asynchronous Receiver Transmitter
UI	Unit Interval: time to send a bit through the serial interface.
Xcvr	Transceiver

DISCLAIMER

This is free and unencumbered software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or distribute this software, either in source code form or as a compiled bitstream, for any purpose, commercial or non-commercial, and by any means.

In jurisdictions that recognize copyright laws, the author or authors of this software dedicate any and all copyright interest in the software to the public domain. We make this dedication for the benefit of the public at large and to the detriment of our heirs and successors. We intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this software under copyright law.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

THIS DISCLAIMER MUST BE RETAINED AS PART OF THIS FILE AT ALL TIMES.

TABLE OF CONTENTS

FIGURES

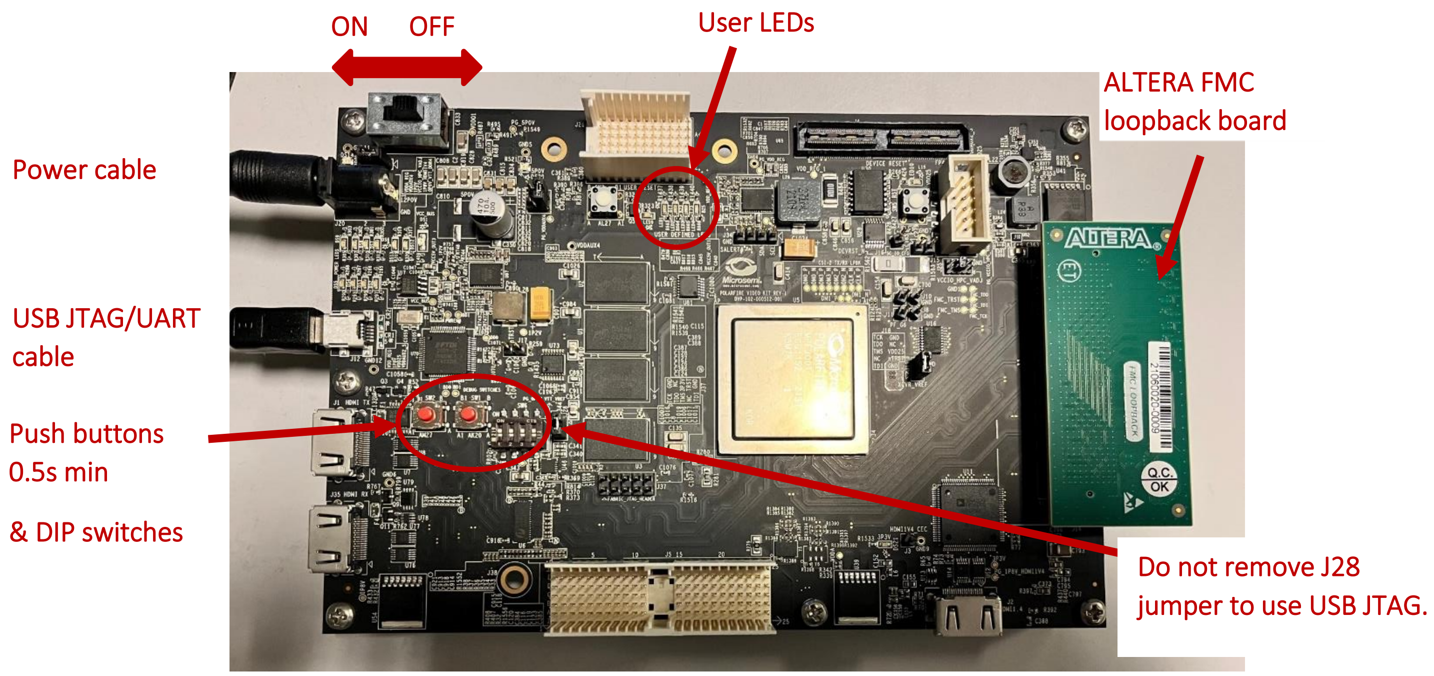
1 HARDWARE

1.1 MATERIAL LIST

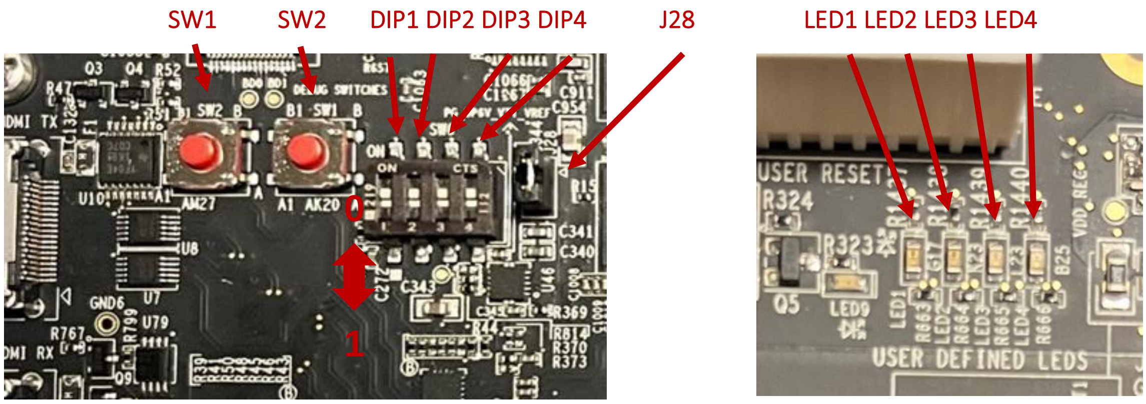
- MPF300-VIDEO-KIT-NS
  - ✓ Product web page: https://www.microsemi.com/existing-parts/parts/150804
- XM107 FMC loopback board
  - ✓ Product web page: https://www.whizzsystems.com/loopback-card/
  - ✓ Schematic: https://www.xilinx.com/content/dam/xilinx/support/documents/boards_and_kits/xtp090.pdf
  - ✓ User guide: https://docs.xilinx.com/v/u/en-US/ug539
- ALTERA FMC loopback board
  - ✓ Mouser web page: https://eu.mouser.com/ProductDetail/Terasic-Technologies/S0485?qs=81r%252BiQLm7BT%2FhHjxmUuTyA%3D%3D
  - ✓ Schematic: https://www.intel.com/content/www/us/en/support/programmable/articles/000086949.html

1.2 USER INTERFACE

- Connect Power cable
- Connect USB JTAG/UART cable
- Connect ALTERA FMC loopback board



1.3 User LEDs, DIP switches and Push buttons



User interface	Description
SW1	Reset (0.5s min)
SW2	SYNC (0.5s min)
DIP1	d_ctrl(0). Default: 0 for ramp test pattern. (*)
DIP2	d_ctrl(1). Default: 1 for ramp test pattern. (*)
DIP3	ESIstream RX and TX scambling enable. Default: 1
DIP4	ESIstream RX disparity processing enable. Default:1
LED1	UART RX is ready when ON.
LED2	ESIstream RX and TX modules are ready when ON.
LED3	ESIstream RX lanes are ready when ON.
LED4	Clock bit and ESIstream data bits error status. When there is one bit error detected on one ESIstream frame transmitted then LED is turned ON. Push reset button to clear the error.

(*) If d_ctrl = “00” then all zeroes test pattern.
(*) If d_ctrl = “01” or “10” then ramp test pattern.
(*) If d_ctrl = “11” then all one’s test pattern.

3 LIBERO PROJECTS

The package contains two projects, a 32-bit and a 64-bit.

Each project will generate a design implementation with a Gigabit Transceiver (GT) serialization and deserialization factor of 32-bit or 64-bit.

It means that the raw data logic vector at Gigabit Transceiver (GT) outputs for receivers (RX) and inputs for transmitters (TX) is configured with a size of 32-bit or 64-bit.

32-bit or 64-bit implementation selection is a trade-off between minimum link latency, minimum logic resources and frames frequency.

32-bit implementation reduces link latency, uses less logic resources but it increases frame frequency

64-bit implementation increases link latency, uses more logic resources but it reduces frame frequency, it can help to relax FPGA design timing constraints.

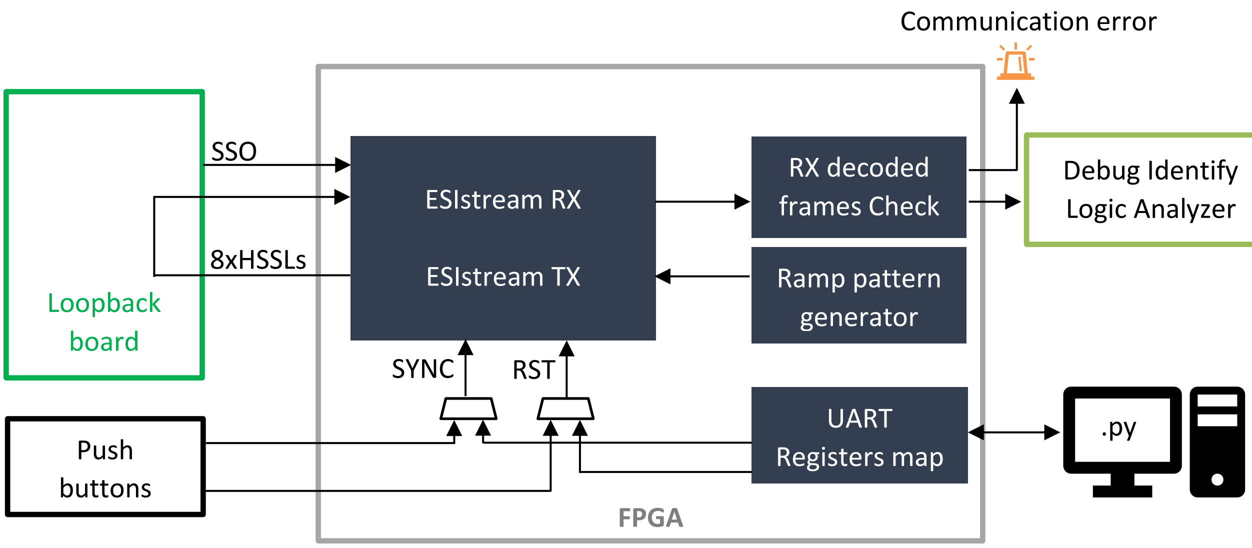
$$f_{frame32\text{-}bit} = 2^*f_{frame\_64\text{-}bit})$$

This project offers a VHDL design example to test the ESIstream serial link using both ESIstream TX and RX modules and a loopback board connecting FPGA HSSLs TX outputs on FPGA HSSLs RX inputs.

DIP switches, push-buttons, LEDs allow to quickly synchronize the high-speed serial link, to start data transfer of a known ramp pattern and to check that there is no communication error.

Synchronization signal, errors status, received frames, or other signals can be directly mapped to Libero Identify Debug to be analyzed.

A simple UART interface allows accessing FPGA registers through read and write operations. A simple frames layer protocol has been defined to communicate with the register map.



0

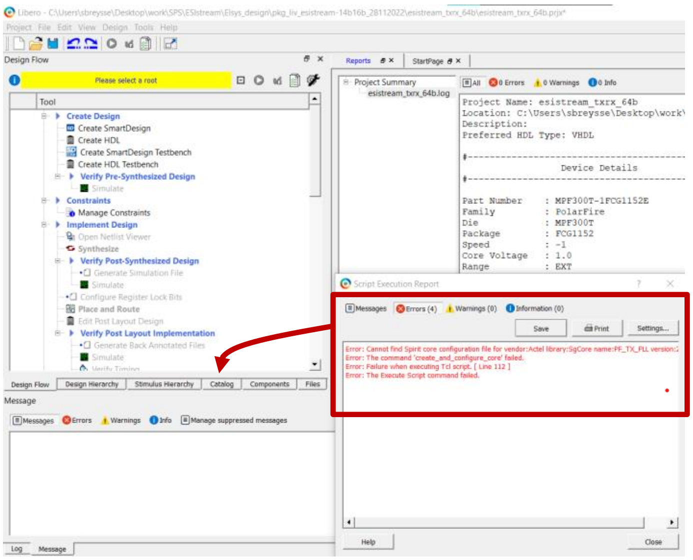
Figure 1: ESIstream TXRX with loopback board project overview.

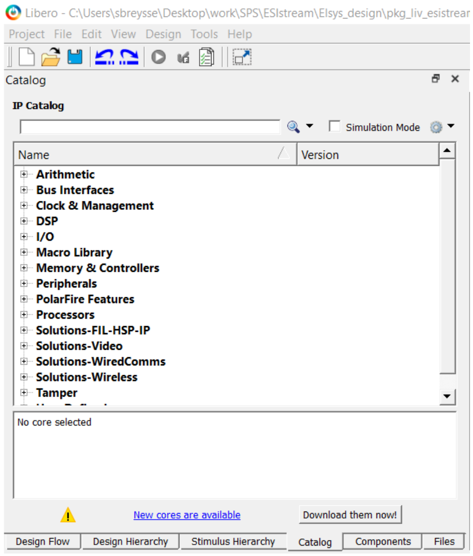
ESI stream

3.1 How to download and to install Libero IPs from catalog?

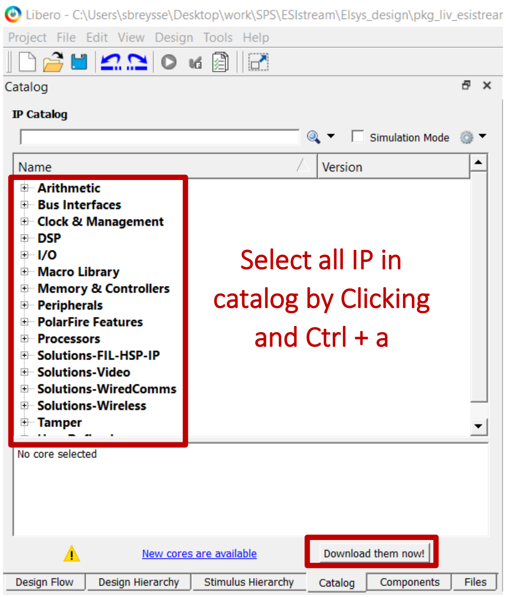
If this is the first time after Libero install, then this error appears.

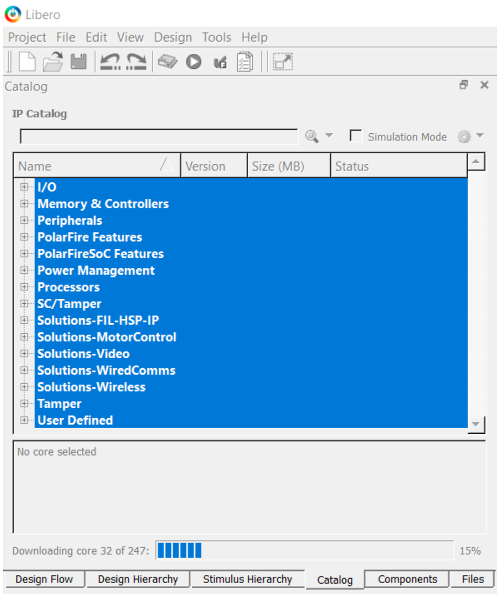
- Click on Catalog tab
- Click on Download them now! To download new available cores.





- In Catalog tab, select all IP categories and click on Download them now!



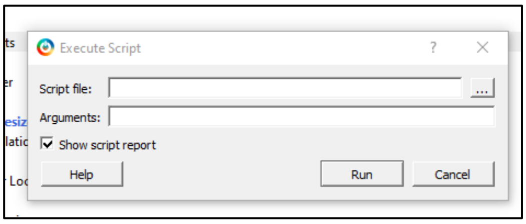


3.2 HOW TO GENERATE LIBERO PROJECT?

- Open Libero:

Package name	Version to use
Package_ESIstream _MPF_300T-1FCG1152E	Libero 2022.1

- Project > Execute Script…



- Select the tcl script that fits the requirements of the design and click on Run
  - /scripts/create_project_32b.tcl or /scripts/create_project_64b.tcl

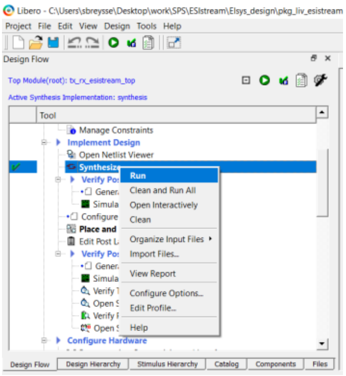
- When the project is built, a Libero project file (.prjx) is available in the corresponding directory (~/esistream_txrx_[32b/64b]).

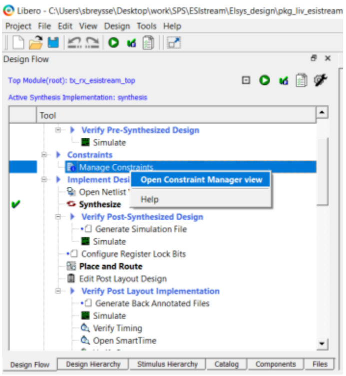
- Project > Open Project

It is now possible to compile, simulate or modify the example design using Microchip Libero toolchain.

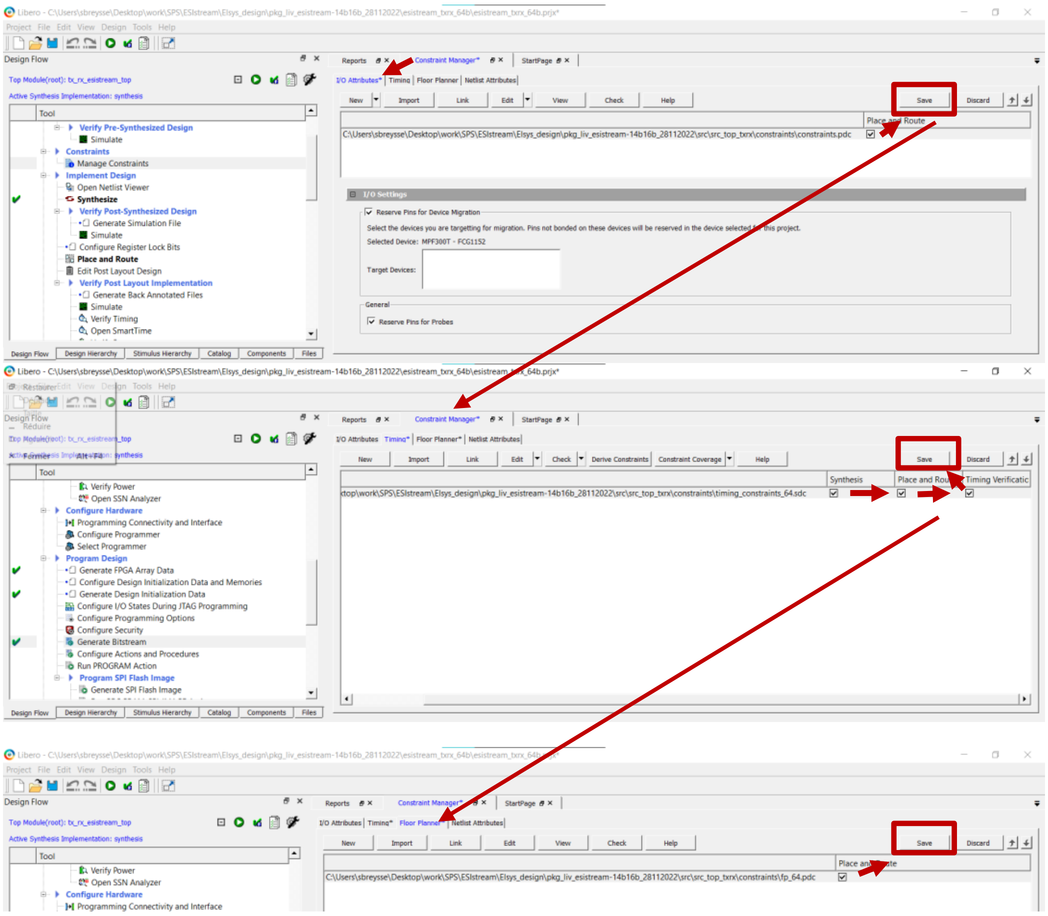
3.3 How to generate bitstream?

- Click on synthesize. Once the project is synthetized, open "manage constraints",

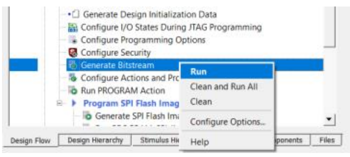




- Then add and select all the existing constraints, IOs (constraint.pdc), fp_32/64.pdc and timing_constraint_32/64.sdc, located in src\src_top_txrx\constraints directory of the package.
  Click on import if the file is missing to add it then tick each checkbox.

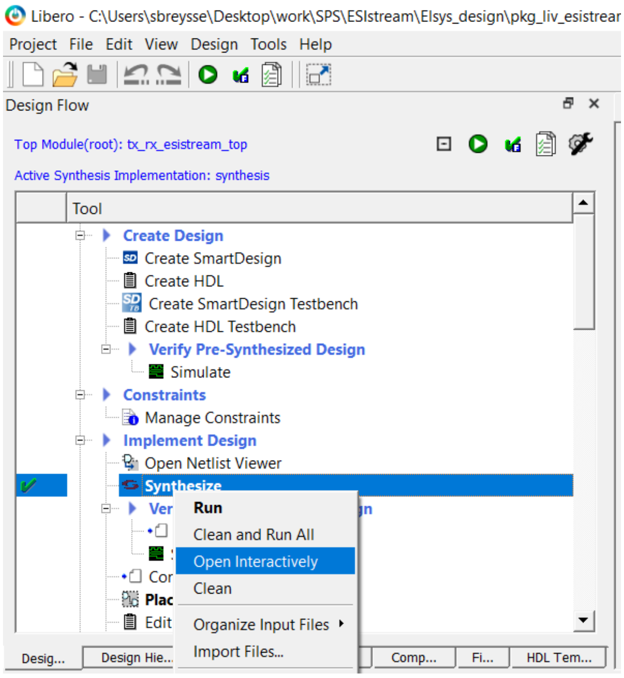


- Right-click on generate bitstream and on Run.

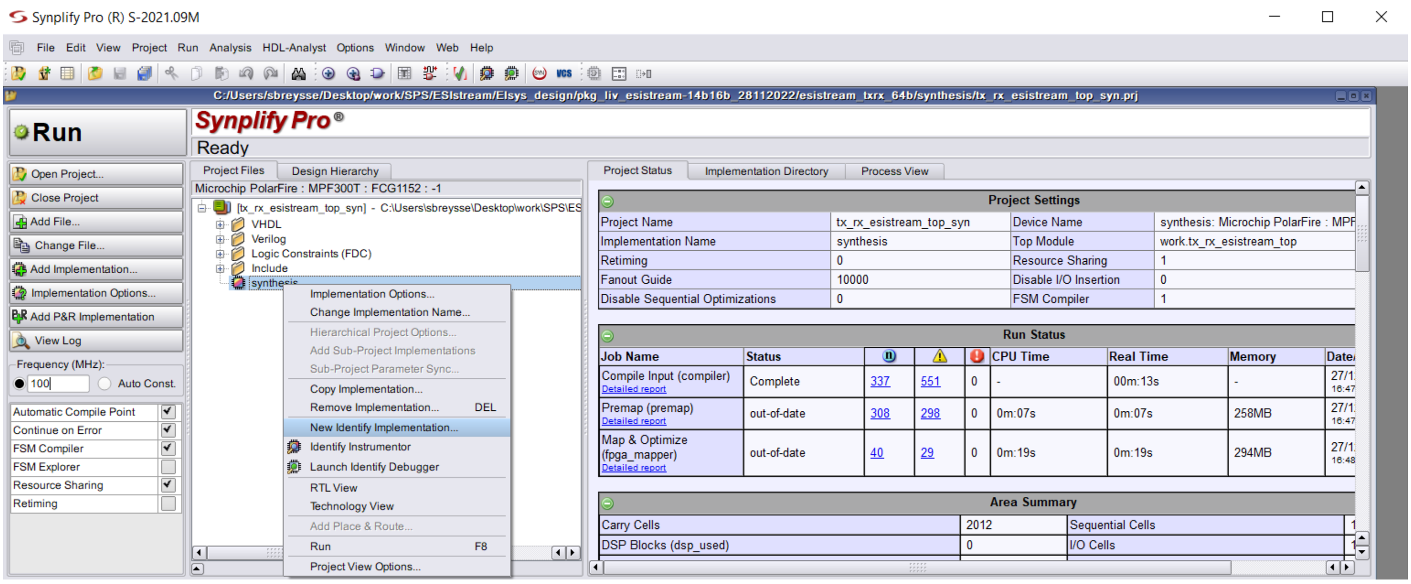


3.4 How generate bitstream and program FPGA?

- Open the project.
- Click on synthesize.
- Right-click on "synthesize "and select "open interactively" to launch the synthesis tool Synplify Pro:



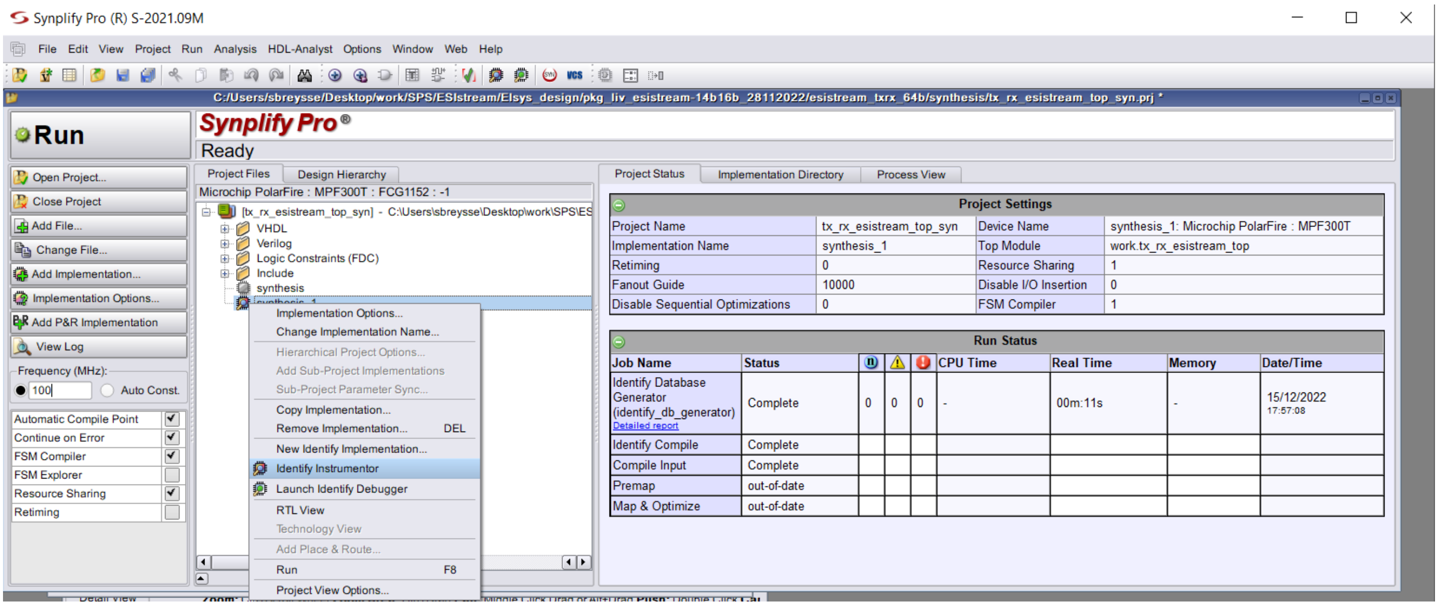
- Create a New Identify Implementation (see the figure below)



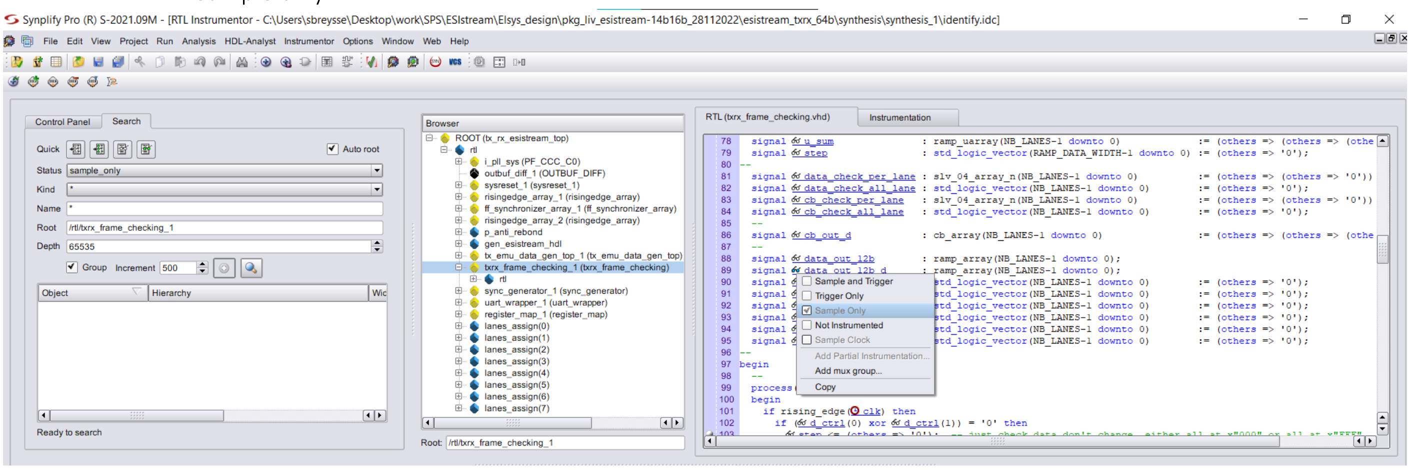
SynplifyPro creates a new Identify implementation with the default location at \synthesis\synthesis_1.

You can edit the implementation name in the Implementation Results tab. Do not change the path of the Results Directory.

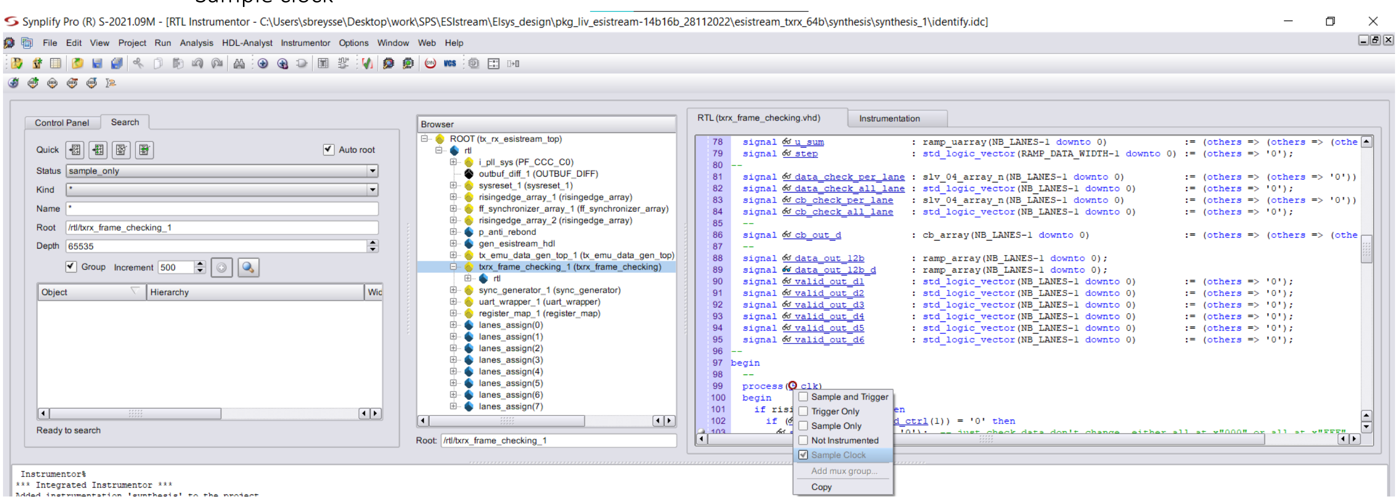
- Launch “Identify Instrumentator” (Right-click on synthesis_1)



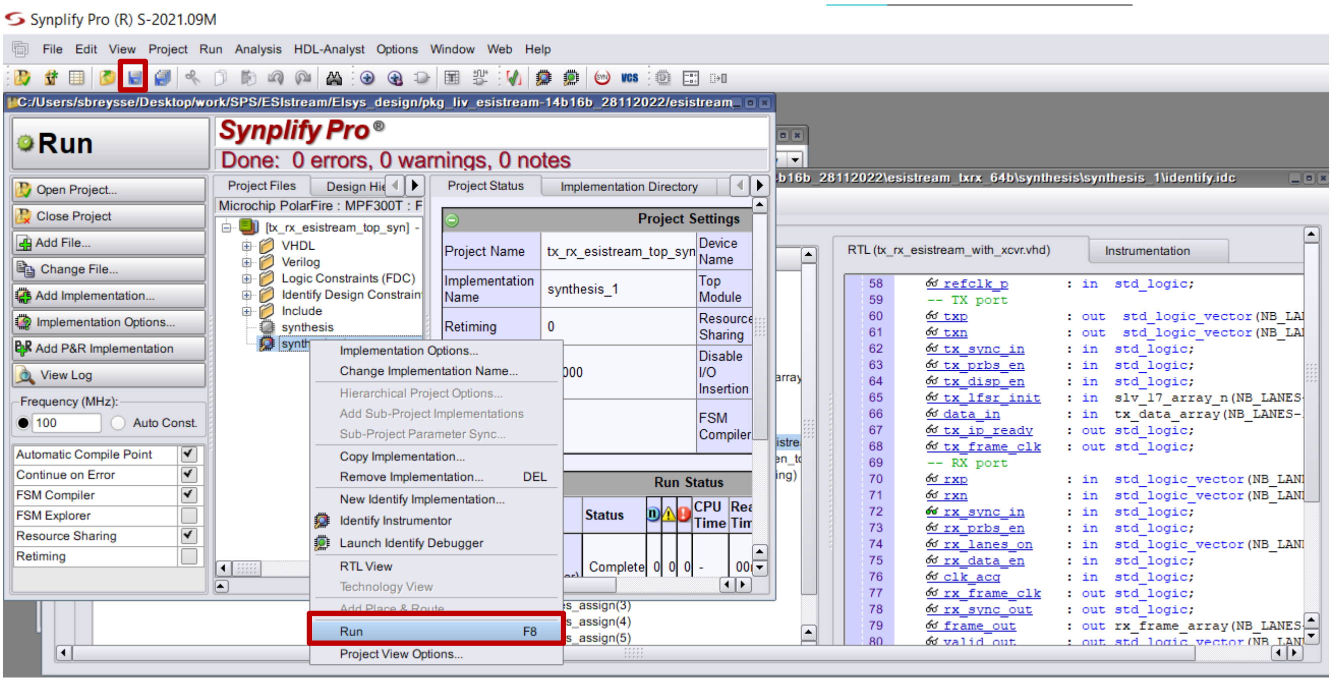
- Select debug signals and a sampling clock.
  - ✓ Sample only



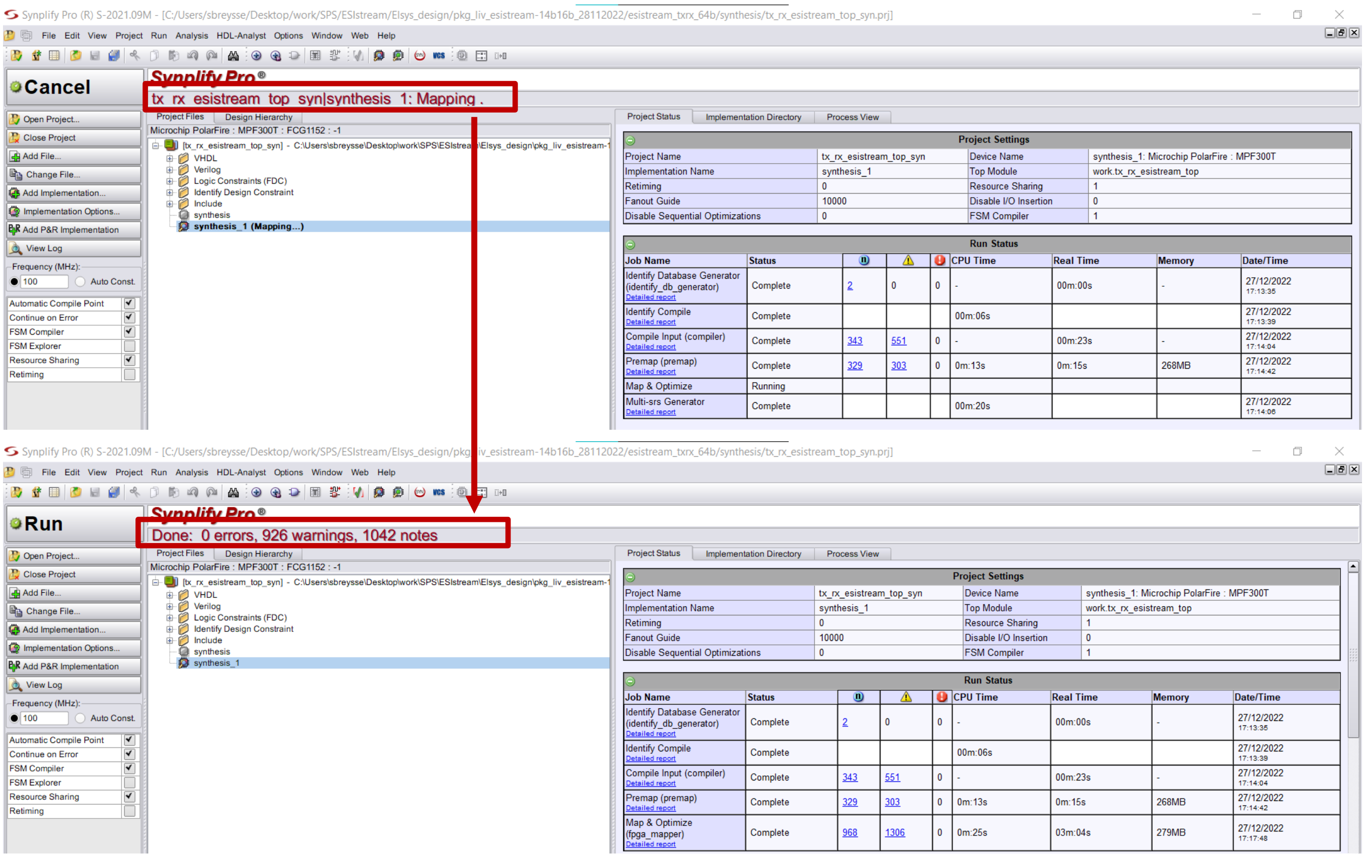
  - ✓ Sample clock



- Save the project then click on run.

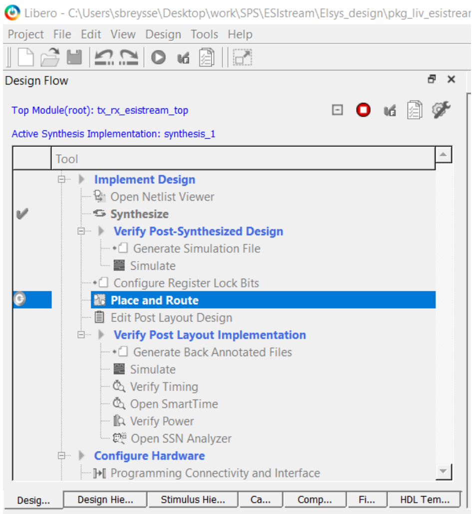


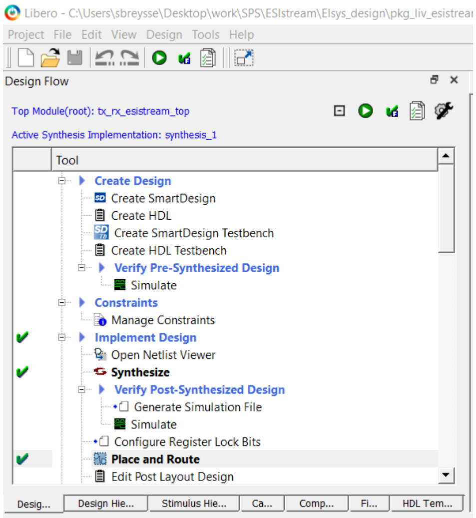
- Wait for end of mapping...



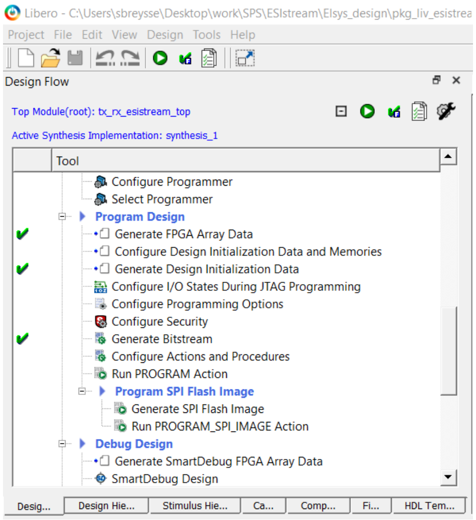
- Close Symplify Pro.

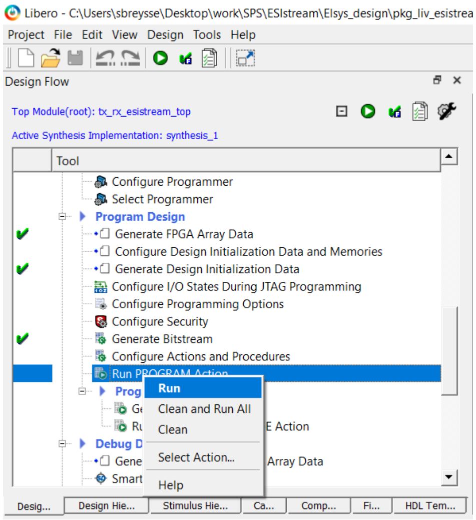
- Run Place and Route



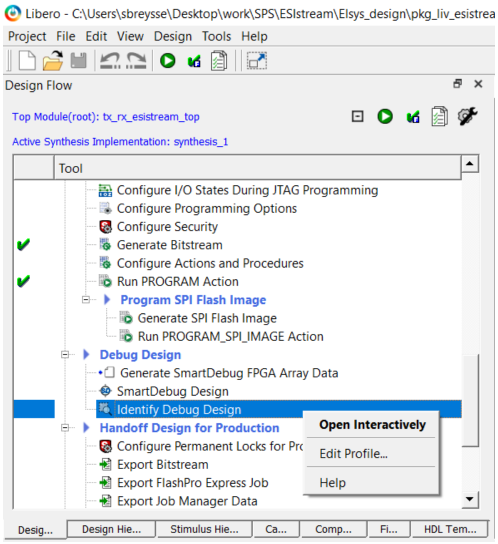


- Generate Bitstream and Run Program Action

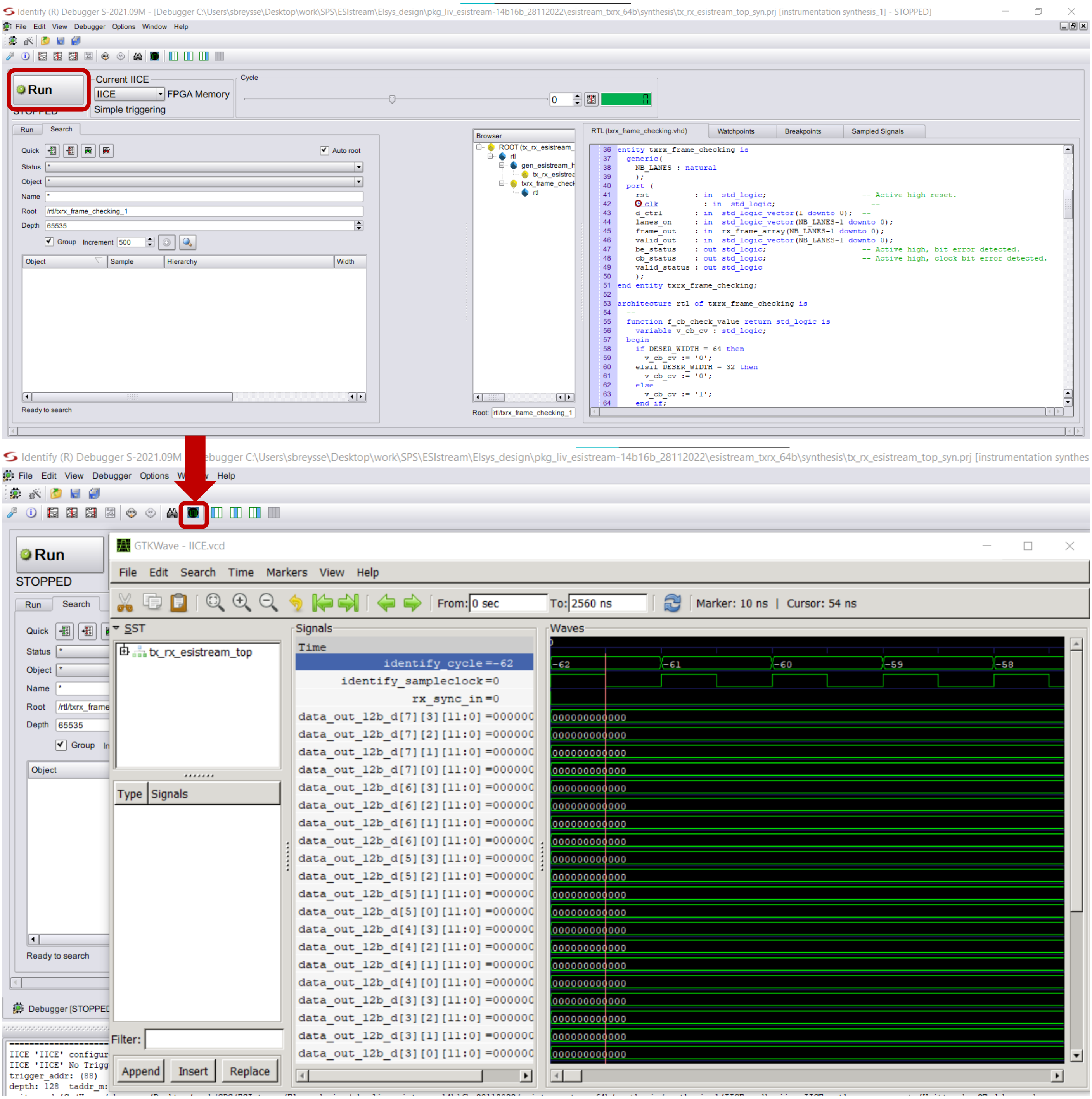




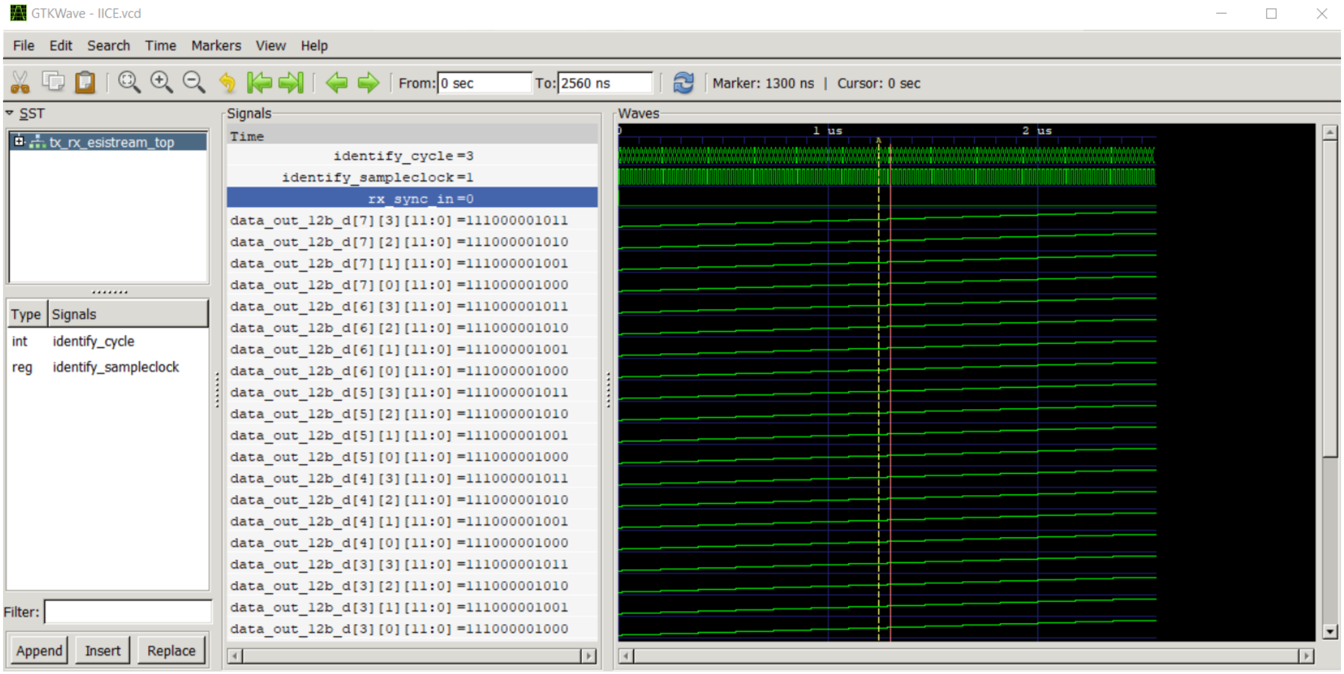
- Wait for end of programming…
- All debug signals can be analyzed by launching the tool Identify Debug Design.

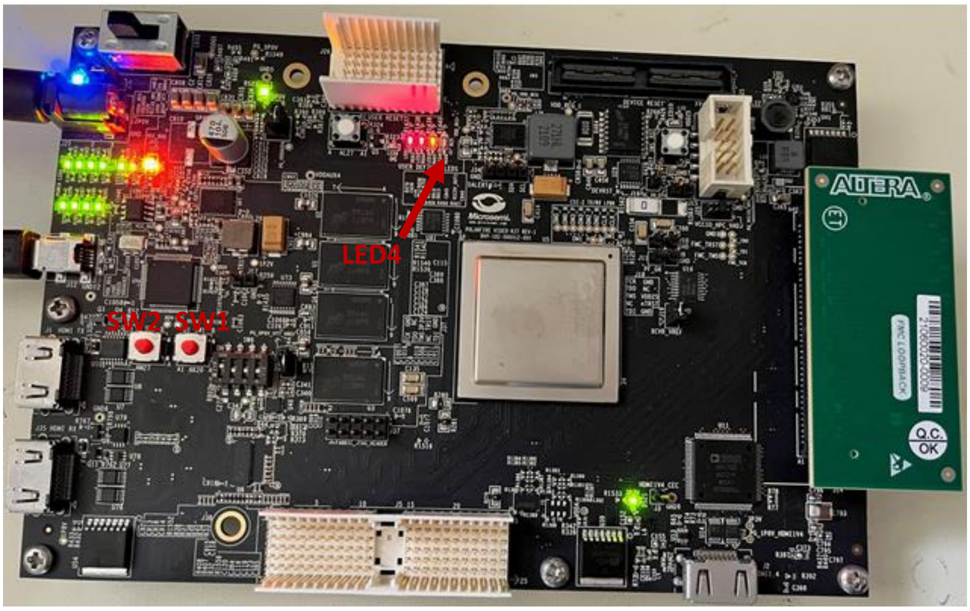


- Click on Run and Waveform viewer.



- Push Reset button (SW1) and SYNC button (SW2), then click on Run and open Waveform viewer again:

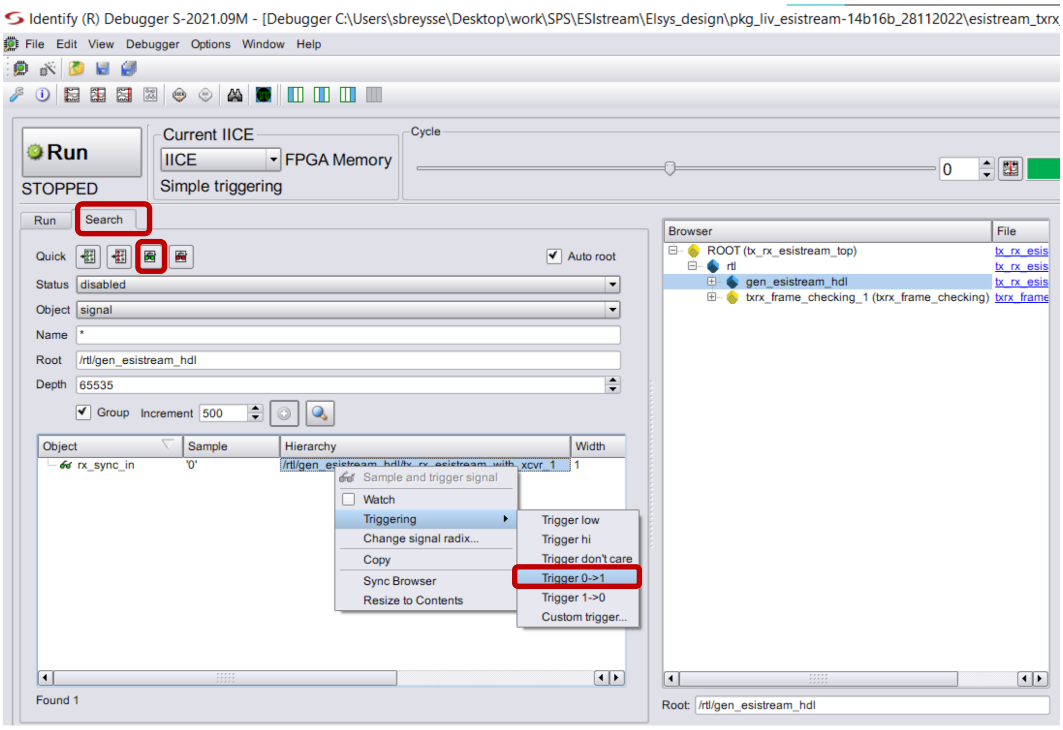




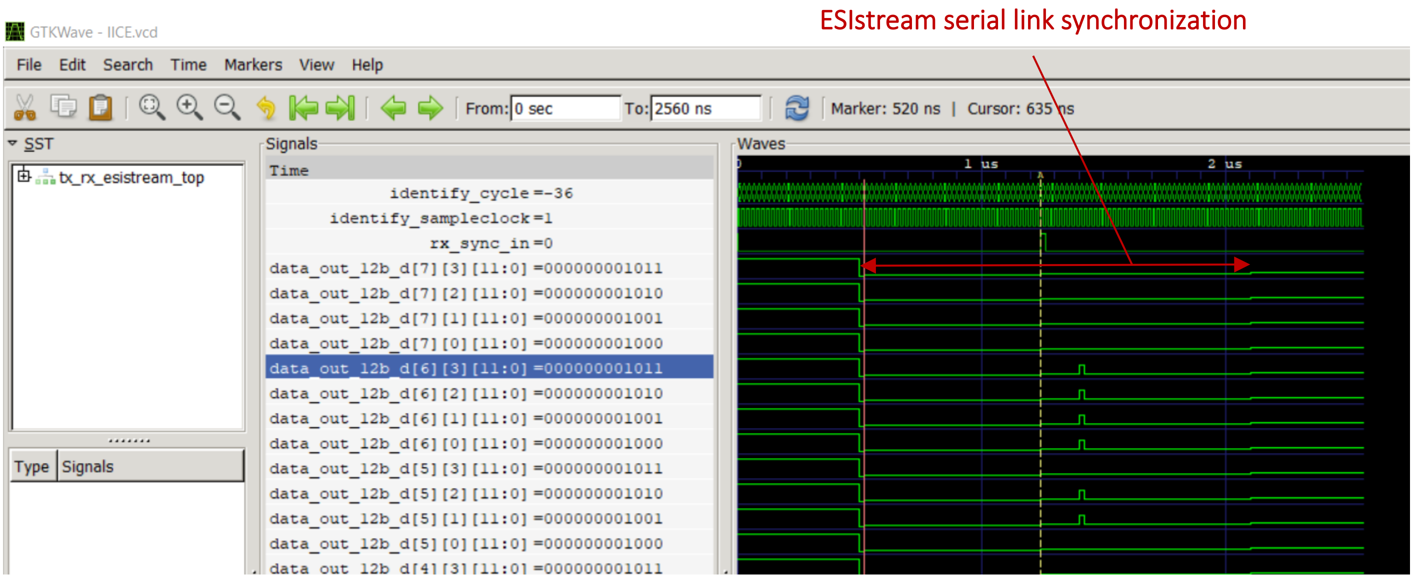
LED4 is OFF indicating there is no communication error.

3.5 How to set trigger on SYNC pulse?

- Click on Search tab
- Click on Search for disabled watchpoints
- Right-click on rx_sync_in then Triggering then Trigger 0-> 1 to select rx_sync_in rising edge.

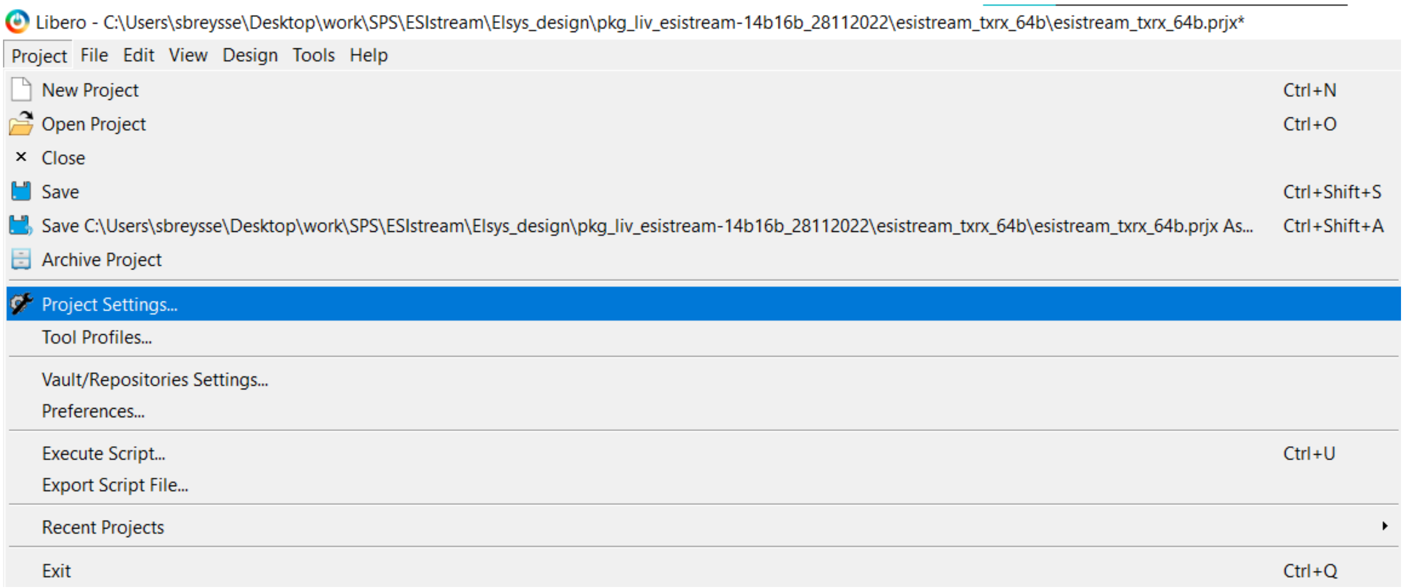


- Click on Run
- Push SYNC button (SW2)
- Open waveform viewer.

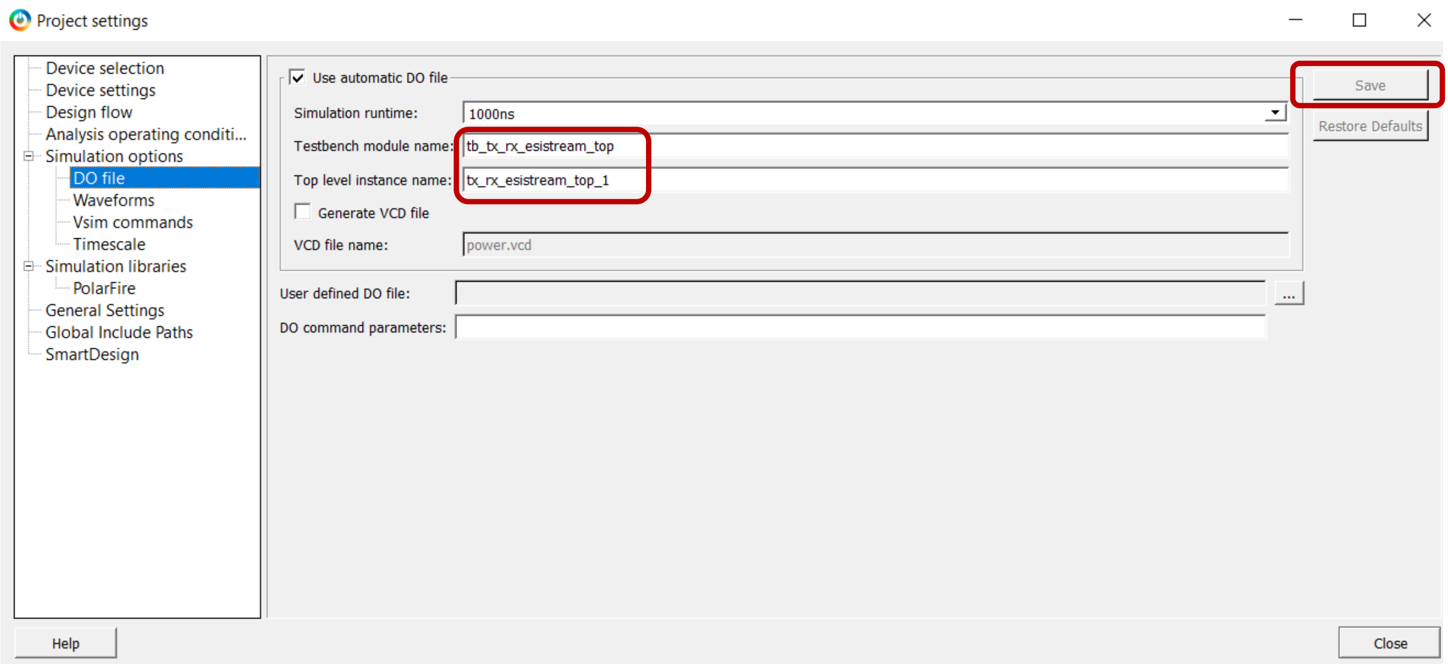


3.6 How to simulate FPGA design example?

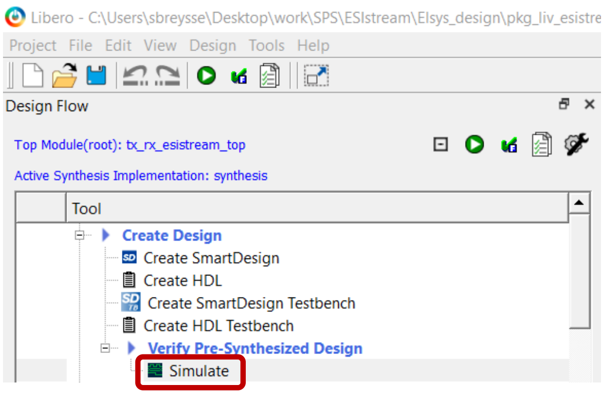
- Click on Project and Project Settings…

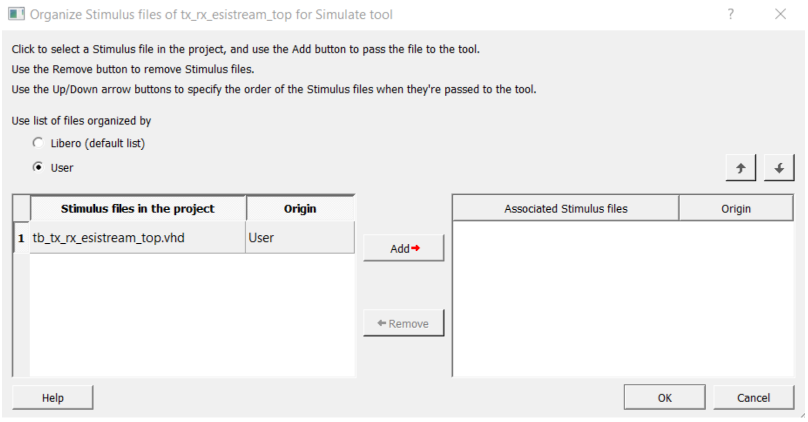


- Click on DO file
  - Fill Testbench module name and Top level instance name fields and click on Save button.

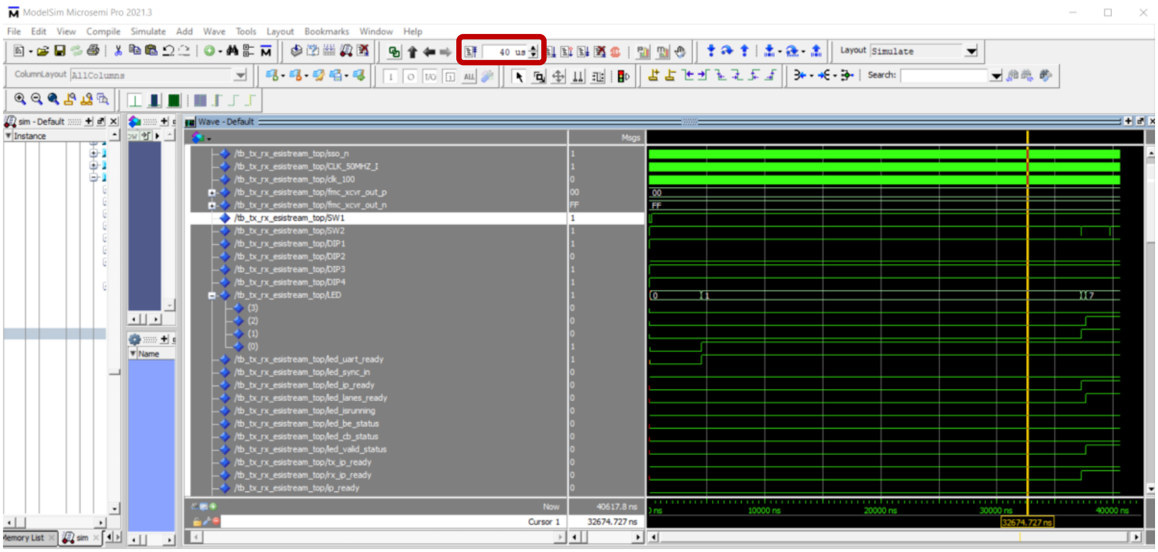


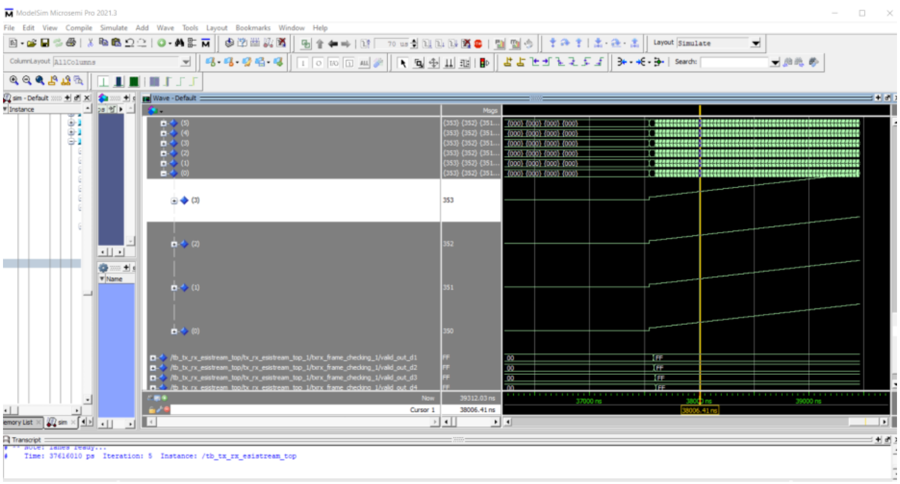
- Launch simulation:





- Run simulation for 40 µs.





4 UART FRAMES: LAYER PROTOCOL

The design embeds an interface 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.

4.1 REGISTER WRITE OPERATION

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

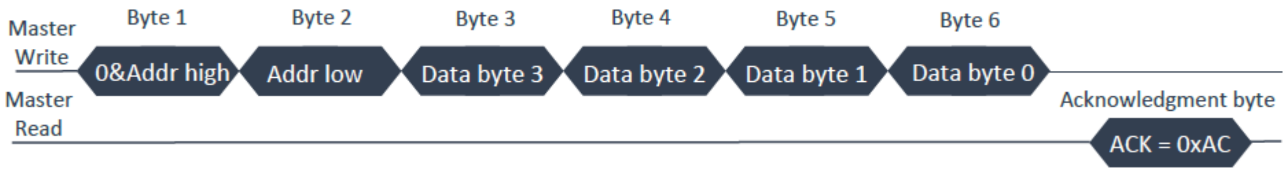


Figure 2: UART frames layer protocol, write operation

4.2 REGISTER READ OPERATION

The UART master must send the data in the order described on the figure below to be able to read a register value.
Firstly, the master sends 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.

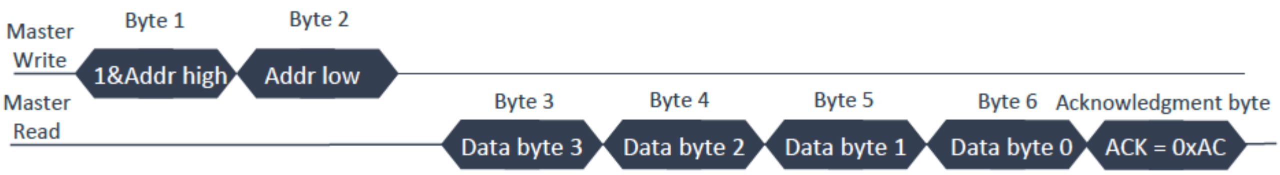


Figure 3: UART frames layer protocol, read operation

4.3 REGISTER MAP

All the registers have a size of 32-bit.
All the 32 bits are read when performing a read operation through the UART communication.
All the writable registers can also be read.

<table>
<tr><th>Signal name</th><th>Register address</th><th>Register bits</th><th>type</th><th>comment</th></tr>
<tr><td>tx_emu_d_ctrl(0)</td><td>0</td><td>0</td><td>W</td><td rowspan="2">Tx emulator data control:
Select the data sent by the Tx emulator.<table><tr><th>tx_emu_d_ctrl</th><th>Waveforn 12-bit</th></tr><tr><td>00</td><td>Constant: x”000”</td></tr><tr><td>01</td><td>12-bit ramp pattern</td></tr><tr><td>10</td><td>12-bit ramp pattern</td></tr><tr><td>11</td><td>Constant: x”FFF”</td></tr></table></td></tr>
<tr><td>tx_emu_d_ctrl(1)</td><td>0</td><td>1</td><td>W</td></tr>
<tr><td>rx_prbs_en</td><td>1</td><td>0</td><td>W</td><td>Enable ESIstream RX IP PRBS decoding (descrambling) when ‘1’.</td></tr>
<tr><td>tx_prbs_en</td><td>1</td><td>1</td><td>W</td><td>Enable ESIstream TX IP PRBS encoding (scrambling) when ‘1’.</td></tr>
<tr><td>tx_disp_en</td><td>1</td><td>2</td><td>W</td><td>Enable ESIstream TX IP disparity processing when ‘1’</td></tr>
<tr><td>reg_rst</td><td>2</td><td>0</td><td>W</td><td>Active high (‘1’) global software reset.</td></tr>
<tr><td>reg_rst_check</td><td>2</td><td>1</td><td>W</td><td>Active high (‘1’) Tx Rx frames checking module reset.</td></tr>
</table>

(*) not used in this project