WWW.ESISTREAM.COM

ESIstream

The **E**fficient **S**erial **I**nterface

ESIstream XILINX V2-2

Version 2.2

ESIstream - 2020

ESIstream is 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 ESIstream IP package allows generating a VHDL project that implements a RX ESIstream serial interface supporting lane rates up to :

- 12.5 Gbps for the Kintex Ultrascale KU060 Xilinx FPGA (xcku060-ffva1517-1-c) for ESIstream_Xilinx_KU060_V2-2 package.
- 12.8 Gbps for the Kintex Ultrascale KU040 Xilinx FPGA for ESIstream_Xilinx_KU060_V2-2 package.
- 11.3 Gbps for the Virtex7 7VX690 Xilinx FPGA for ESIstream_Xilinx_7VX690_V2-2 package.

This document aims at explaining the VHDL project and the RX ESIstream IP architecture.

To simplify and to speed up ESIstream IP implementation, this package provides:

- VHDL sources.
- Fully implementable projects.
- Test-bench for simulation.

Although ESIstream IP is specific to a Xilinx FPGA, the modular architecture allows an easy migration to a different Xilinx target replacing the FPGA specific IPs.

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

Package supported by this document

ESIstream_Xilinx_KU060_V2-2 (ADAS-SDEV-KIT2 evaluation kit, all scripts available)
ESIstream_Xilinx_KU040_V2-2 (KCU105 evaluation kit, script_32b.tcl & script_64b.tcl only)
ESIstream_Xilinx_7VX690_V2-2 (VC709 evaluation kit, script_64b.tcl only)

Reference documents

ESIstream Protocol specification V2.0: www.ESIstream.com
EV12AQ600 datasheet: website link, download link

Terminology

ADC	Analog to Digital Converter
ASIC	Application-Specific Integrated Circuit
CDR	Clock and Data Recovery
DAC	Digital to Analog Converter
ESIstream	the Efficient Serial Interface
FAS	Frame Alignment Sequence
FPGA	Field Programmable Gate Array
ILA	Integrated Logic Analyzer (a Vivado feature)
LFSR	Linear Feedback Shift Register
PAS	PRBS Alignment sequence
PL	Programmable Logic
Phase Locked Loop	Phase Locked Loop
PRBS	Pseudo-Random Binary Sequence
RX	Receiver

TX	Transmitter
UI	Unit Interval: time to send a bit through the serial interface.
Xcvr	Transceiver

ESIstream Overview

ESIstream protocol initiated by Teledyne-e2v is born from a severe need of the following combination:

- Increase rate of useful data, when linking data converters operating at GSPS speeds with FPGAs on a serial interface, reducing data overhead on serial links, as low as possible.
- Simplified hardware implementation, simple enough to be built on RF SiGe technologies.

ESIstream provides an efficient High-Speed serial 14B/16B interface. It is **license-free** and supports in particular serial communication between FPGAs and High-Speed data converters.

An ESIstream system is made up of the following elements.

- A transmitter can be an ADC or an FPGA or an ASIC
- A receiver can be a DAC or an FPGA or an ASIC
- A number of lanes ($L \geq 1$) to transmit serial data
- A synchronization signal (sync) to initialize the communication.

There is no clock lane in a serial interface. For each lane, the receiver should recover the clock from the data.

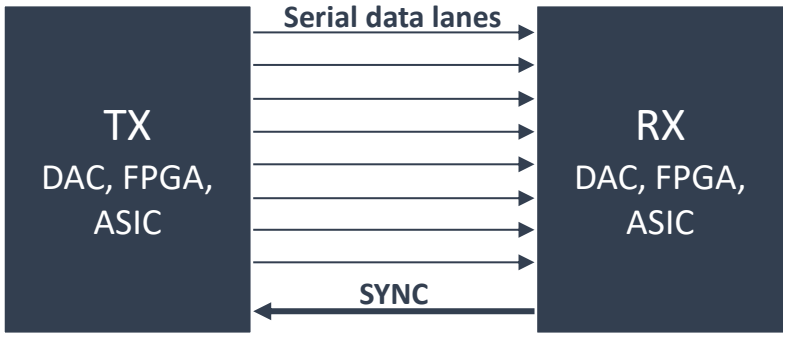


Figure 1: Basic ESIstream system

The main key benefits are:

- FLEXIBILITY: License free
- EFFICIENCY: 87.5%, with 14-bit of useful data and 2-bit overhead (clock bit and disparity bit).
- SIMPLICITY: Minimal hardware implementation.

The main key features of ESIstream are:

- Deterministic latency
- Multi-device synchronization
- Demonstrated lane rate up to 12.8Gbps (on Kintex Ultrascale KU040), depending on device abilities
- Multi-lanes synchronization
- Guaranteed DC balance transmission, ±16 bit running disparity
- Synchronization monitoring, using the clock bit (overhead bit)
- Sufficient number of transitions for CDR, max run length of 32.

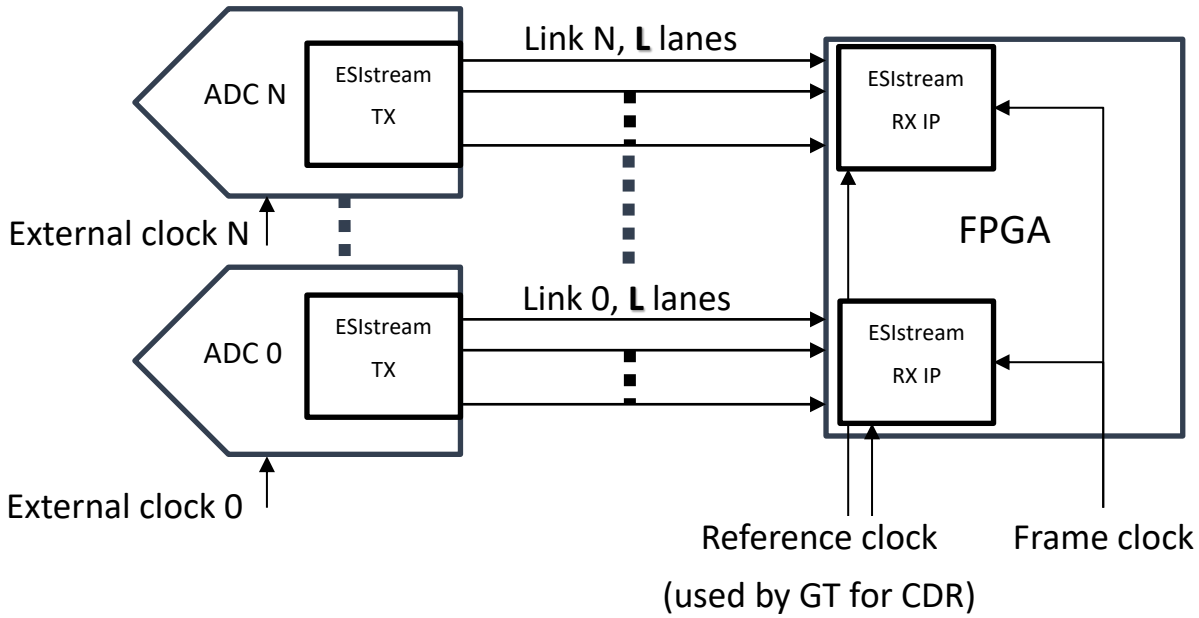


Figure 2: Multiple ADCs ESIstream system

License free, what does that mean?

- Who can download an ESIstream IP package including all sources (vhdl, xdc, xci, tcl)?

Anyone can download ESistream IP package on www.esistream.com/ip-package.

- Who can upload a new ESIstream IP package on www.esistream.com.or make changes on an existing ESIstream IP package?

Only Teledyne-e2v ESIstream developers.

- Can a user contribute to ESIstream?

Yes, users can send their contribution through ESistream contact web page.

- Is there a revision control?

For revision control, each package is referenced with a version number, which is incremented at each new release of a ESIstream IP package. For instance, ESISTREAM_XILINX_**V2-1**_32b.
Internally, a subversion client software allows to restore each modified file, and to compare it to previous revisions.
Finally, a repository maintains each released package (.zip files).

Disclaimer

ESI stream

TABLE OF CONTENTS

FIGURES

Documents Amendment

Issue	Date	Comments
1.0	June 2019	Creation
2.1	October 2019	Publication
2.2	April 2020	Lower RX decoding latency Lower RX logic resources utilization.

1. ESISTREAM RX IP (RECEIVER)

1.1 Principle

After a SYNC pulse, the RX IP waits for the synchronization sequence (FAS – Frame Alignement Sequence & PAS – PRBS Alignment Sequence) sent by the transmitter to align the received frames and to initialize its LFSR. The LFSR generates the PRBS sequence required to descramble the received frame. The receiver decodes the aligned raw data frames according to the ESIstream protocol specification applying descrambler processing and disparity bit processing to get the data.

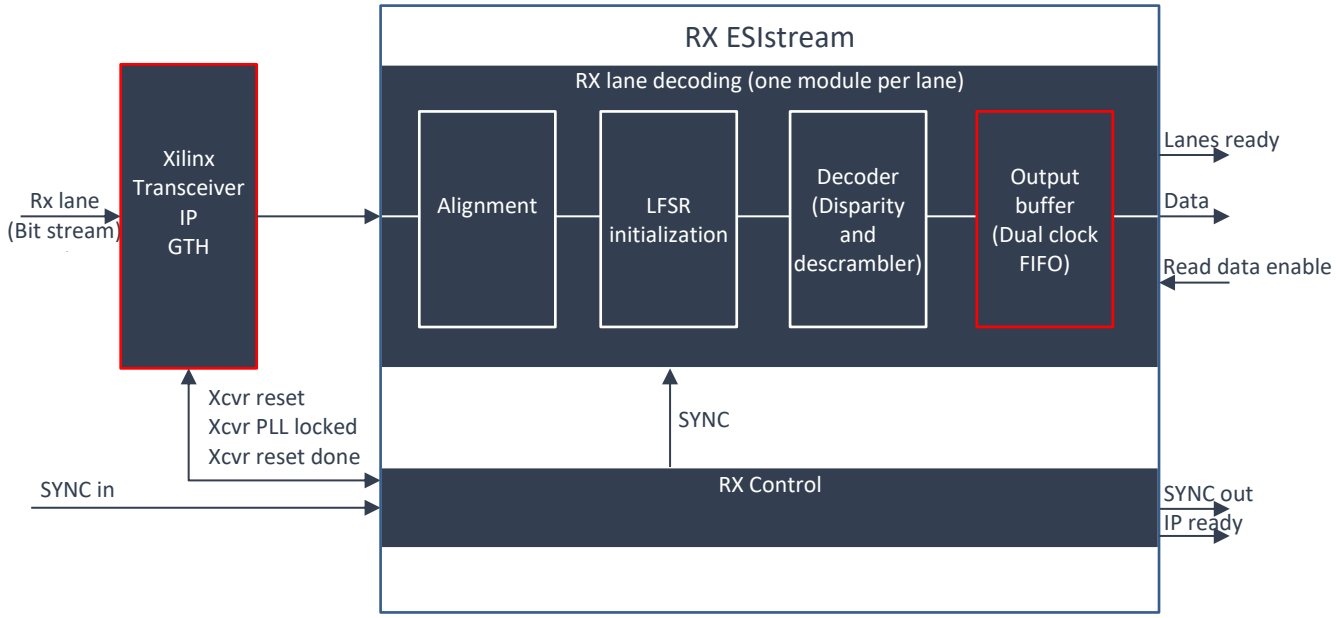


Figure 3: ESIstream RX IP block diagram, principle.

The output buffer allows managing of multiple lanes data alignment. For each lane, data are buffered in the corresponding output buffer as soon as a valid data is decoded. Then, when all lanes contain valid data (lanes_ready high), data can be released (read_data_en high).

The output buffer also allows crossing between the “recovered” frame clock domain (rx_usrclk) and the “acquisition” frame clock domain (clk_acq), see Figure 4 and Figure 5 for clock distribution details.
For non-deterministic latency applications, both clocks can be connected together using the rx user clock (rx_usrclk) provided by the GTH transceiver IP.
For deterministic latency applications, the “acquisition” frame clock domain (clk_acq) must be connected to a deterministic clock; for instance from a MRCC pin or from the reference clock through IBUFDS_GTE3 ODIV2 output.

In some cases and depending on the application, the dual clock FIFO implemented in the output buffer can be replaced by a single clock FIFO or by shift registers to reduce the amount of logic resources used by RX IP. For more information, please contact us using ESIstream contact web page or at GRE-HOTLINE-BDC@Teledyne.com.

Modules depicted with **red outlines** in the block diagrams (namely the transceiver IP GTH and the output buffer) are provided by Xilinx

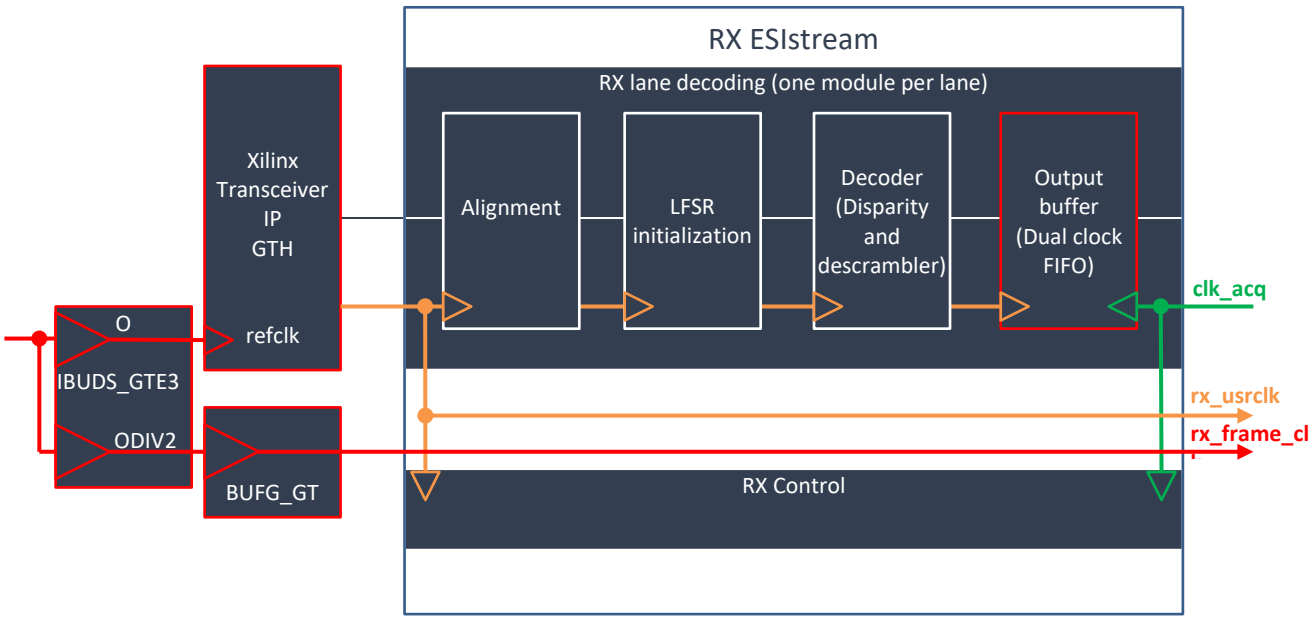


Figure 4: ESIstream RX IP block diagram, clock domains (using script_64b_dl.tcl to create the project).

The Figure 4 uses a 64-bit deserialization data path compare to the Figure 5, which uses a 32-bit deserialization data path.

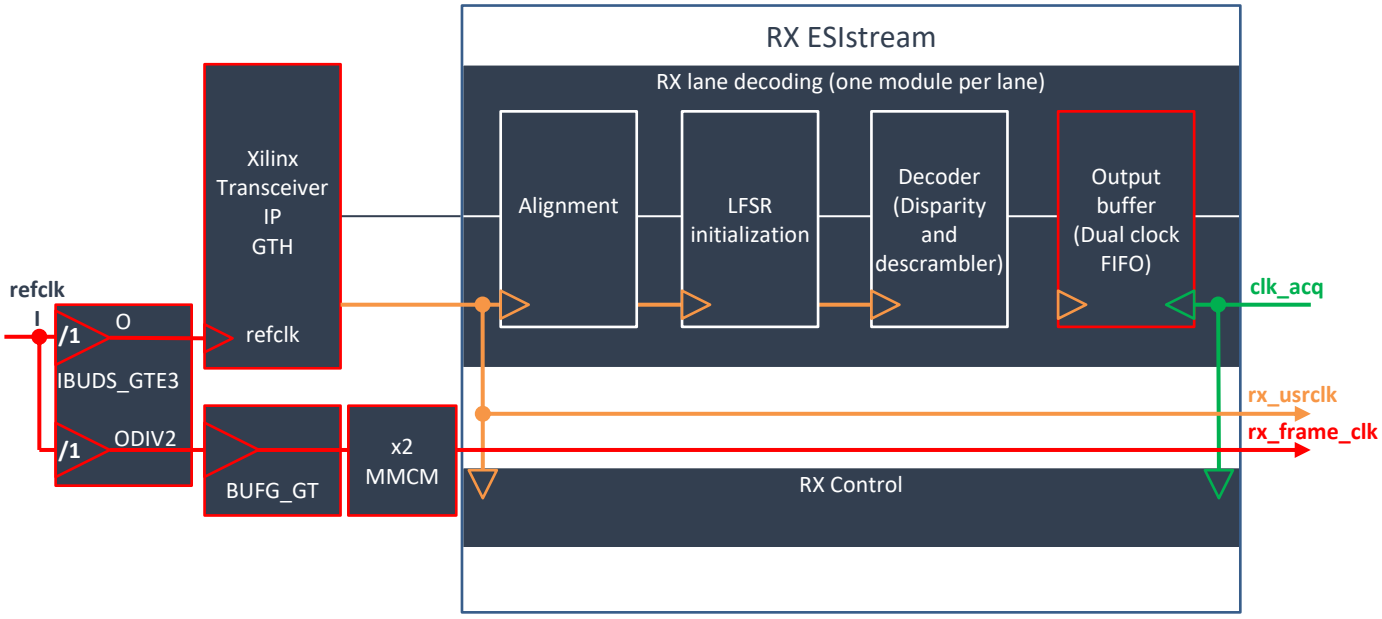


Figure 5: ESIstream RX IP block diagram, clock domains (using script_32b_dl.tcl to create the project).

ODIV2 seems to suggest that in UltraScale FPGA this clock output is only refclk input divided by two, but here it is configured to be the buffered refclk input (refclk divided by one).
Please check Figure 2-1: Reference Clock Input Structure of the Xilinx documentation, UG576 UltraScale Architecture GTH Transceivers

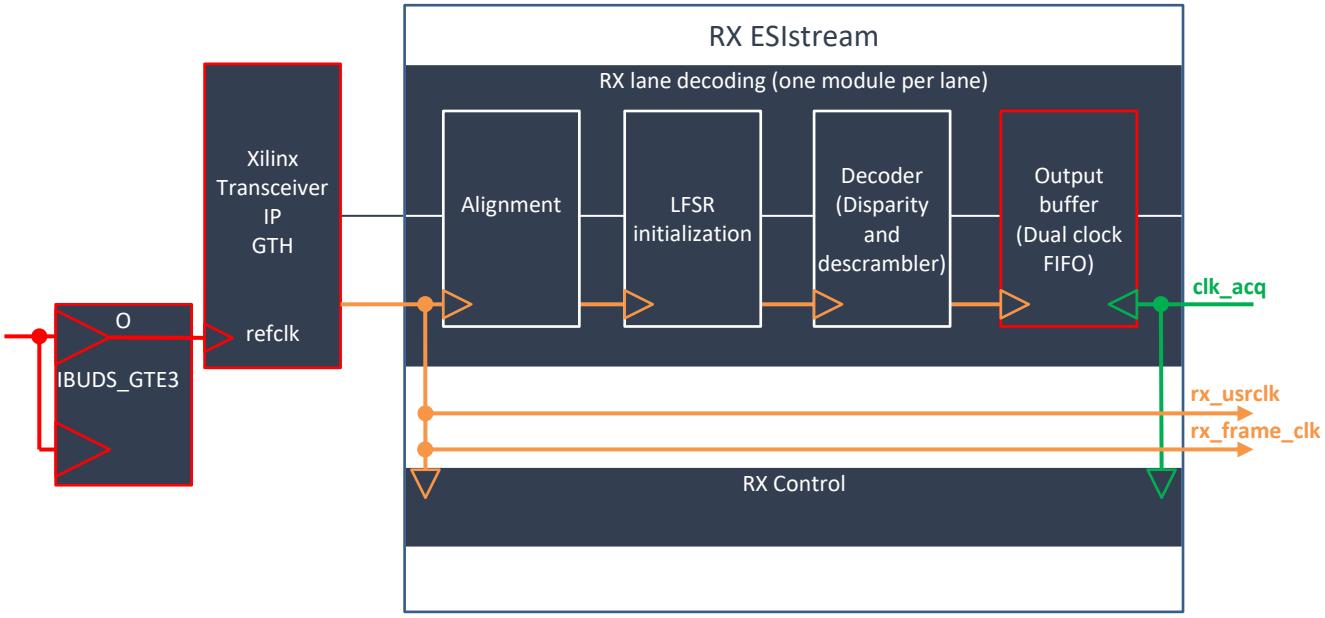


Figure 6: ESIstream RX IP block diagram, clock domains (using script_32/64b.tcl to create the project).

Using the creation project scripts script_32b or script_64b, the frame clocks rx_usrclk and rx_frame_clk are both generated by the transceiver IP either from the reference clock or from the recovered clock. The clock source should be selectable in the Vivado transceiver wizard interface.

1.2 IP Description

1.2.1 File tree

The diagram below shows the files and folders organization in the IP package for the ESIstream RX IP only.

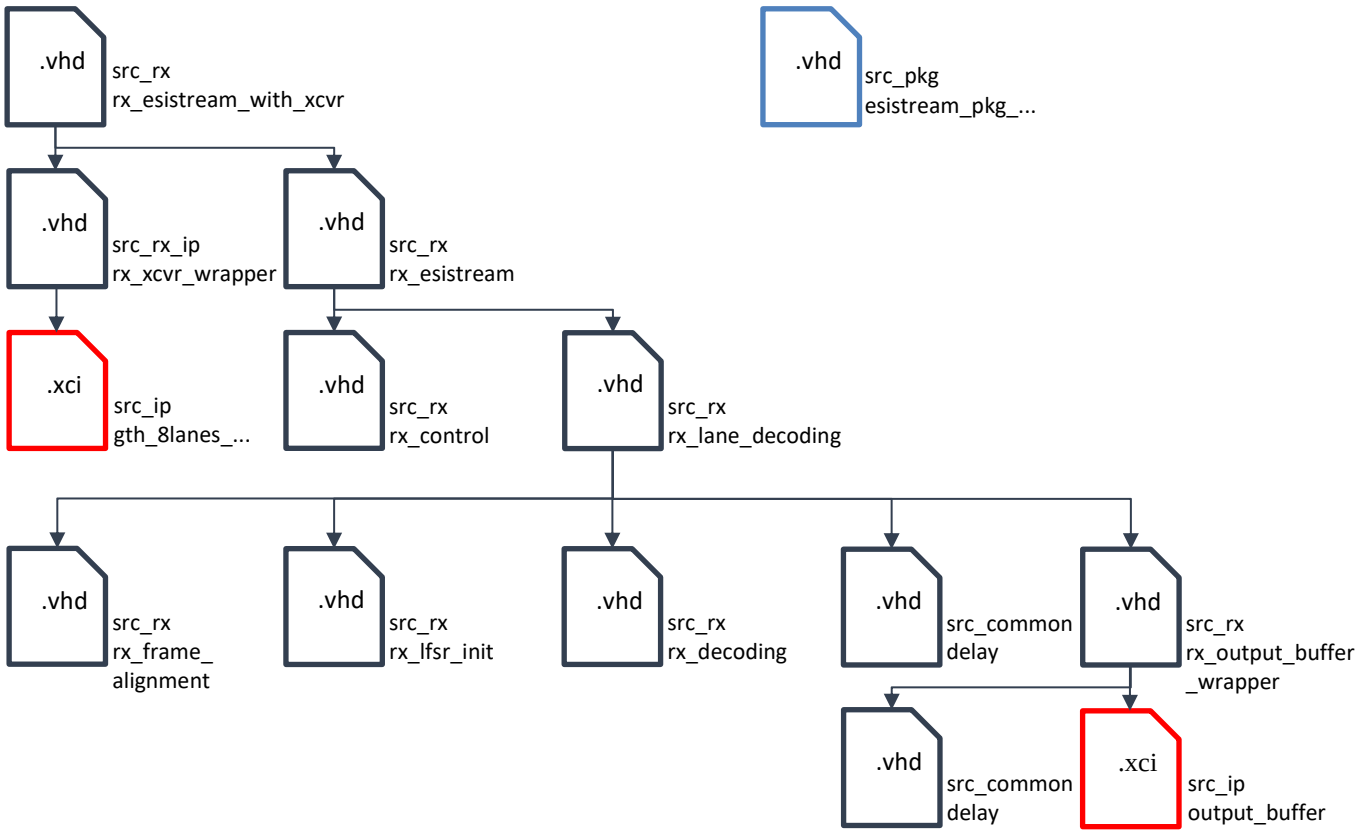


Figure 7: ESIstream RX IP file tree

1.2.2 ESIstream package

F name	Description
esistream_pkg_32b.vhd esistream_pkg_64b.vhd	Almost all files described in section 1.2 File tree use a common package *esistream_pkg_32/64b.vhd* to share common signal types, functions (for instance: LFSR), and constants (for instance: DESER_WIDTH, SER_WIDTH).

1.2.3 RX ESIstream IP top file

File name	Description
rx_esistream_with_xcvr.vhd	**This is the top wrapper file of the ESIstream RX IP.** It instantiates the transceiver wrapper and clock device primitives to generate the frame clock from the reference clock input or from the transceiver user clock output. It also instantiates the rx_esistream sub-module which manages the transceiver and which contains the link synchronization and decoding modules according to the ESIstream protocol V2.0. For each lane, the rx_esistream module decodes received raw data from the transceiver

	IP. • When DESER_WIDTH is 32, raw data contain unaligned 2 x 16-bits ESIstream encoded frame. • When DESER_WIDTH is 64, raw data contain unaligned 4 x 16-bits ESIstream encoded frame. The rx_esistream modules provides valid decoded data on its outputs frame_out and data_out when valid_out is high.

1.2.4 Transceiver wrapper

File name	Description
rx_xcvr_wrapper.vhd	Wraps the transceiver (GTH Xilinx IP) and clock device primitives to provide a generic entity interface depending on the project used. The rx_control module instantiated in the rx_esistream module manages the reset of the transceiver. For each lane, the transceiver delivers deserialized raw data in a logic vector of width 32-bit or 64-bit depending again on the project used.

1.2.5 Transceiver IP

File name	Description

gth_8lanes_32b.xci
gth_8lanes_64b.xci
	Transceiver default configuration (parameters value can be modified) :
	• Transceiver configuration preset: Start from scratch
	• Transceiver type: GTH
	• Receiver Line rate (Gb/s): 12.5
	• PLL type: QPLL0
	• Actual Reference Clock (MHz): 195.3125
	• Decoding: Raw (no encoding)
	• User data width:
	  ○ 32, when gth_8lanes_32b.xci
	  ○ 64, when gth_8lanes_64b.xci
	• Internal data width: 32
	• Buffer: Enable (1)
	• RXOUTCLK source: RXOUTCLKPMA
	• Insertion loss at Nyquist (dB): 20
	• Equalization mode: DFE
	• Link coupling: AC
	• Termination: Programmable
	• Programmable termination voltage (mV): 800
	• PPM offset between receiver and transmitter: 0
	• Channel Quad X1Y3 in SLR 0 Bank 227: 4/4 channels used
	• Channel Quad X1Y2 in SLR 0 Bank 226: 4/4 channels used
	• MGTREFCLK0 of Quad X1Y3: used for both Bank 227 & Bank 226
	• rxpd_in: enabled
	qpll0lock_out: enabled

1.2.6 RX ESIstream

File name	Description
rx_esistream.vhd	rx_esistream manages control signals in rx_control: • reset of the transceiver (rst_xcvr). • reset of the lane decoding module (rst_esistream). • synchronization signal of the lane decoding module (sync_esistream). rx_esistream instantiates one rx_lane_decoding sub-module for each serial lane: • When DESER_WIDTH is 32, it decodes 32-bits of raw data (xcvr_data(index)) received from the transceiver at each clock period of the frame clock. • When DESER_WIDTH is 64, it decodes 64-bits of raw data (xcvr_data(index)) received from the transceiver at each clock period of the frame clock. ○ index is the number of the lane. rx_esistream generates lane**s**_ready signal using **and bitwise operation** on lane_ready logic vector. Each bit of the lane_ready vector is addressed by one lane_ready output of each rx_lane_decoding sub-module. When high, the lane**s**_ready signal indicates that all lanes are synchronized.

1.2.6.1 Output data organization

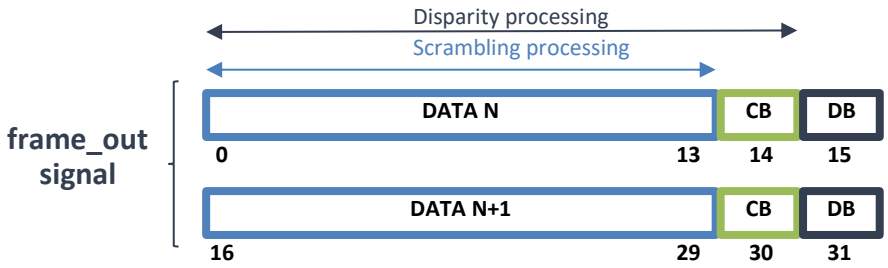


Figure 8: 32-bit output data organization

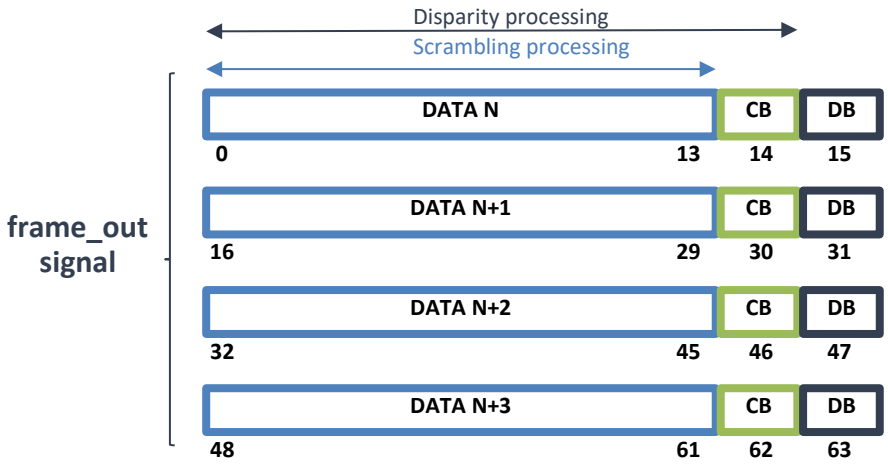


Figure 9: 64-bit output data organization

1.2.6.2 Entity generic parameters

rx_esistream.vhd module entity generic paremeters description.

Generic	Type	Values	Description
NB_LANES	Natural	1 to (FPGA maximum number of transceivers)	Number of lanes
COMMA	std_logic_vector	x"00FFFF00" or x"FF0000FF"	Frame Alignment Sequence synchronization pattern.

1.2.6.3 Entity port signals

rx_esistream.vhd module entity port description.

Port	Type	Width	Clock domain	Description
rst	Input	1	clk_acq	Active high transceiver PLL asynchronous reset
rst_xcvr	Output	1	rx_usrclk	Active high transceiver asynchronous reset
rx_rstdone	Input	NB_LANES array of 1-bit	-	Active high transceiver reset done.
xcvr_pll_lock	Input	NB_LANES array of 1-bit	-	Active high transceiver PLLs lock.
rx_usrclk	Input	1	rx_usrclk	Transceiver user clock (frame clock) from transceiver.
xcvr_data_rx	Input	DESER_WIDTH x NB_LANES bit vector	rx_usrclk	Transceiver user data from transceiver.
sync_in	Input	1	clk_acq	Active high pulse. Generates the synchronization sequence for the receiver, frame alignment and PRBS initialization.
prbs_en	Input	1	Async	Active high, enables scrambling processing.
lanes_on	Input	NB_LANES array of 1	Async	Active high, enable lane. If lane disabled, then no data is available in corresponding lane output buffer.
read_data_en	Input	1	clk_acq	Active high, enables read of data at buffers outputs. When enabled and when valid_out output is high, received data is streamed on frame_out and data_out.
clk_acq	input	1	clk_acq	Acquisition clock, output buffer read port clock, should be the same frequency with no phase drift with respect to receive clock (default: clk_acq should take rx_clk).
sync_out	Output	1	rx_usrclk	Active high pulse. Connect to transmitter sync_in to generate the synchronization sequence for the receiver, frame alignment and PRBS initialization
frame_out	Output	NB_LANES array of 16 x DESER_WIDTH/ 16	clk_acq	Decoded ESIstream frame: disparity bit (15) & clk bit (14) & data (13 down to 0) = frame_out(15 downto 0).
data_out	Output	NB_LANES array of 14 x DESER_WIDTH/ 16	clk_acq	Decoded useful data. data_out = frame_out(13 downto 0).
valid_out	Output	NB_LANES array of 1	clk_acq	Active high one rx_clk period later than read_data_en, frame_out and data_out stream valid output.
ip_ready	Output	1	Async	Active high, when transceiver PLL(s) locked and transceiver reset done. Indicates that IP is ready to receive a sync pulse.
lanes_ready	Output	1	clk_acq	Active high, indicates lanes synchronization status. When high, all enabled lanes are synchronized and data are available at output_buffer(s) outputs.

1.2.7 RX Control

File name	Description
rx_control.vhd	rx_control manages clock domains crossing of control signals between “recovered” frame clock (rx_usrclk) and “acquisition” clock domain (clk_acq). rx_control generates reset of the transceiver when global asynchronous system reset is activated or when at least one of the transceiver PLLs are unlocked. rx_control generates IP ready signal when transceiver reset procedure done and transceiver PLLs locked. IP ready informs that the rx_esistream module is ready to receive a synchronization pulse from the client application. rx_control generates the synchronization pulse of the lane decoding module. rx_control generates reset of the lane decoding module when at least one the transceiver PLLs is unlocked.

1.2.8 RX lanes decoding

File name	Description
rx_lane_decoding.vhd	One rx_lane_decoding must be instantiated per lane. • When DESER_WIDTH is 32, it decodes 32-bits of raw data (xcvr_data(index)) received from the transceiver at each clock period of the frame clock. That means two ESIstream frames are decoded every frame clock cycle. • When DESER_WIDTH is 64, it decodes 64-bits of raw data (xcvr_data(index)) received from the transceiver at each clock period of the frame clock. That means four ESIstream frames are decoded every clock cycle. When a synchronization pulse is received, it initializes the frame alignment module waiting for the COMMA alignment pattern to align the received frames. Once the received frames are aligned, the internal LFSR is initialized, using the PRBS words sent during the PRBS alignment sequence just after the Frame Alignment Sequence. This to synchronize the receiver LFSR and the transmitter LFSR which will both generates the same PRBS sequence. Then the decoder module can use the PRBS words generated by the rx_lfsr_init module to descramble the data. The decoder module also monitor the disparity bit of each frame to decode the disparity processing.

1.2.9 RX frame alignment

File name	Description
rx_frame_alignment.vhd	After a synchronization pulse, this module waits and detects the COMMA pattern repeated sixteen times during the Frame Alignment Sequence to align the received frames. Once the COMMA pattern is detected, it indicates to the rx_lfsr_init module to start waiting for the PRBS Alignment Sequence.

The transmitter (TX) sends a known 32 frames 'comma' sequence, 0x00FFFF00 or 0xFF0000FF, to the receiver (RX).

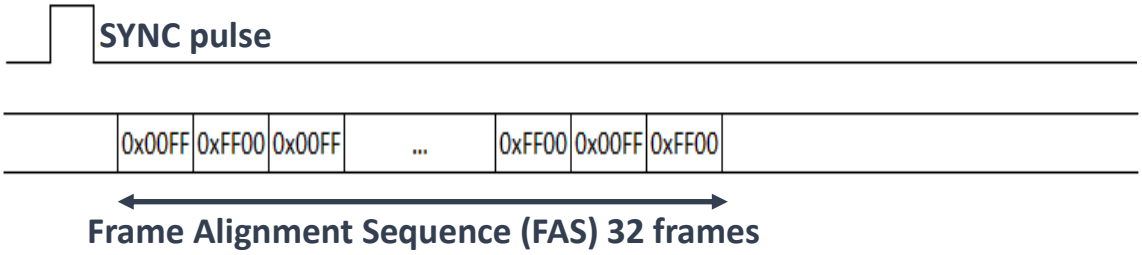


Figure 10: Link synchronization, Frame Alignment Sequence

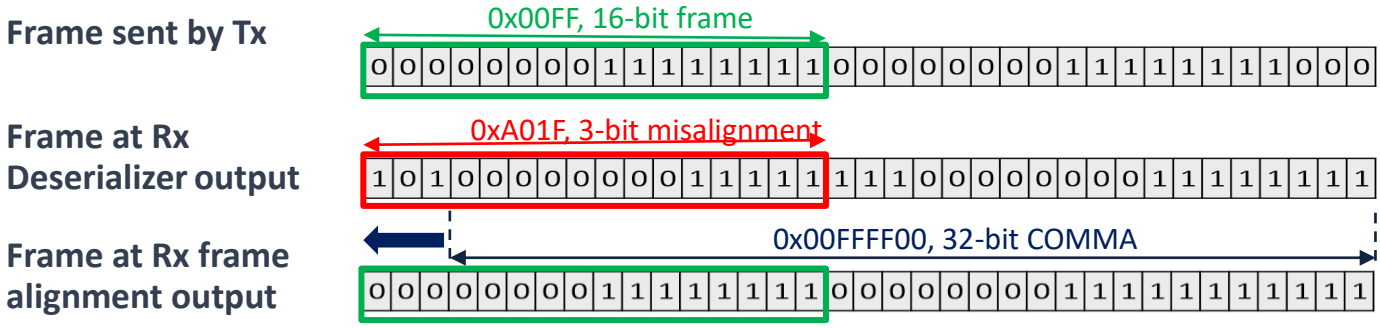


Figure 11: RX Frame alignment module, principle

1.2.10 RX LFSR init

File name	Description
rx_lfsr_init.vhd	After a synchronization pulse, waits for the frame alignment flag indicating that the COMMA pattern has been detected to start waiting for the PRBS Alignment Sequence (PAS). When the PRBS Alignment Sequence starts, the rx_lfsr_init module uses the received PRBS words to synchronize its internal LFSR with the transmitter LFSR which will both generates the same PRBS sequence.

Following the FAS and to initialize the Rx LFSR, the Tx sends 32 frames PRBS values in the data field.

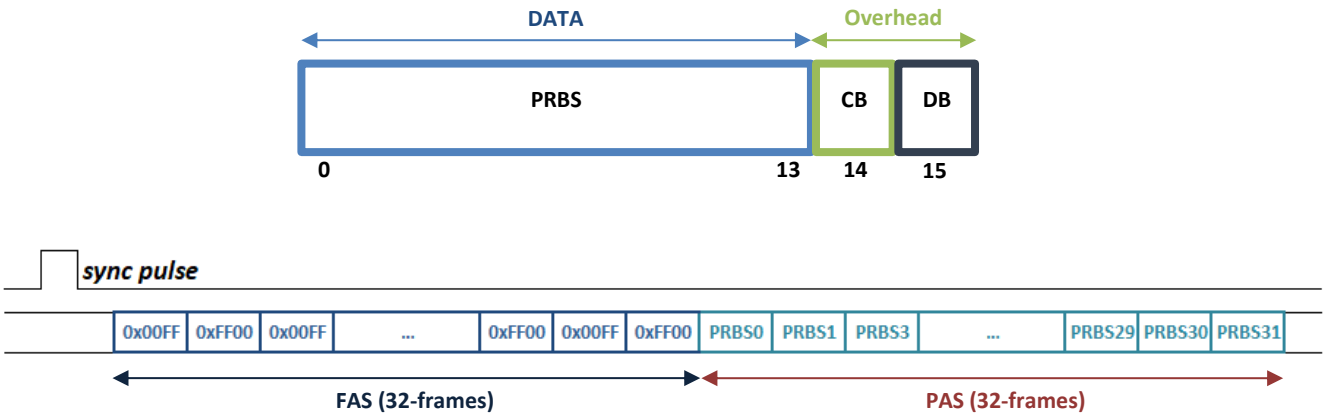


Figure 12: Link synchronization, PRBS Alignment Sequence

After both sequences sent on each lane, Tx and Rx are fully synchronized.

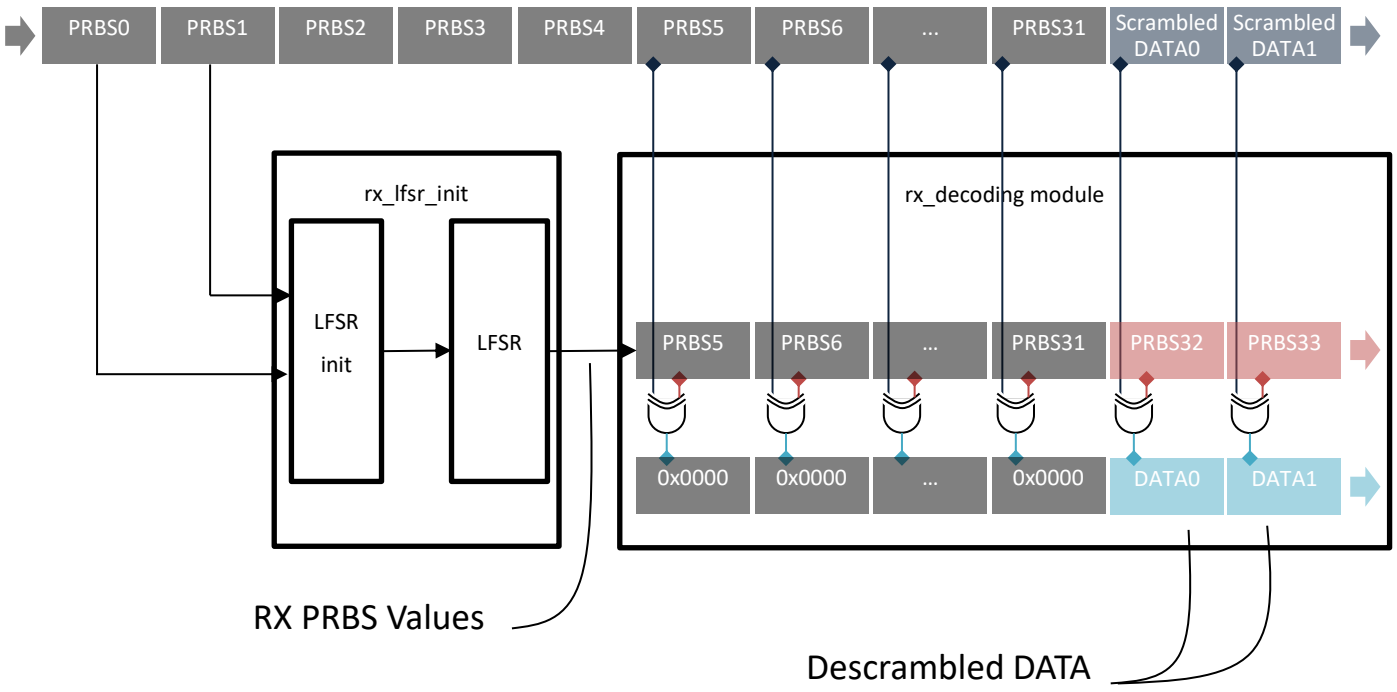


Figure 13: Descrambling principle

1.2.11 RX decoding

File name	Description
rx_decoding.vhd	For each ESIstream frame: First reverse disparity processing applied on the data and the clock bit, then descramble the data. Note: Scrambling not applied on overhead bits.

1.2.12 Delay in RX ESistream

File name	Description
delay.vhd in rx_lane_decoding.vhd	Create a ready data flag to be able to detect the first valid data received just after the PRBS alignment sequence. It is a delayed image of the “PRBS Alignment Sequence started” flag generated in the rx_lfsr_init module. This flag authorizes data writing in the output buffer (dual clock FIFO). Therefore, the frames of the Frame and PRBS Alignment Sequences are not written in the output buffer.

1.2.13 RX output buffer wrapper

File name	Description
rx_output_buffer_wrapper.vhd	The output buffer allows managing of multiple lanes data alignment. For each lane, data are buffered in the corresponding output buffer as soon as a valid data is decoded. Then, when all lanes contain valid data (lanes_ready high at rx_esistream module level), data can be released applying a logic high on read_data_en signal. The output buffer also allows crossing between the "recovered" frame clock domain (rx_usrclk) and the "acquisition" frame clock domain (clk_acq). Output buffer not empty indicates synchronized data are available at buffers outputs. Therefore, lane_ready is just a not operation of FIFO empty flag.

Serial devices often use multiple lanes to transmit or to receive data. For each lane a different delay is introduced when the data propagate between the transmitter (TX) and the receiver (RX). This delay is composed of the TX latency, of the lane propagation delays and of the RX latency. The lane propagation delay will depends on the PCB trace length. The TX and RX latencies will depend on the use of elastic buffer introducing variable latencies on each lane.

To compensate the skew between lanes and to ensure that all lanes are aligned at the ESIstream RX IP outputs. The ESIstream RX IP implements an output buffer for variable delays compensation. This buffer can be either a FIFO or a shift register.
The idea is that on each lane, once the link is synchronized using the FAS and the PAS synchronization sequences, descrambled data start to be written in the output buffer. Then, when all output buffers are not empty the lanes_ready signal goes up authorizing the release of data from the buffer output. The client application should detect the lanes_ready signal rising and should apply a high logic level on read_data_en input to release the valid data. The application can also just loopback lanes_ready on read_data_en if there is no need for deterministic latency.
This mechanism ensures that all lanes are synchronized at ESIstream RX IP outputs.

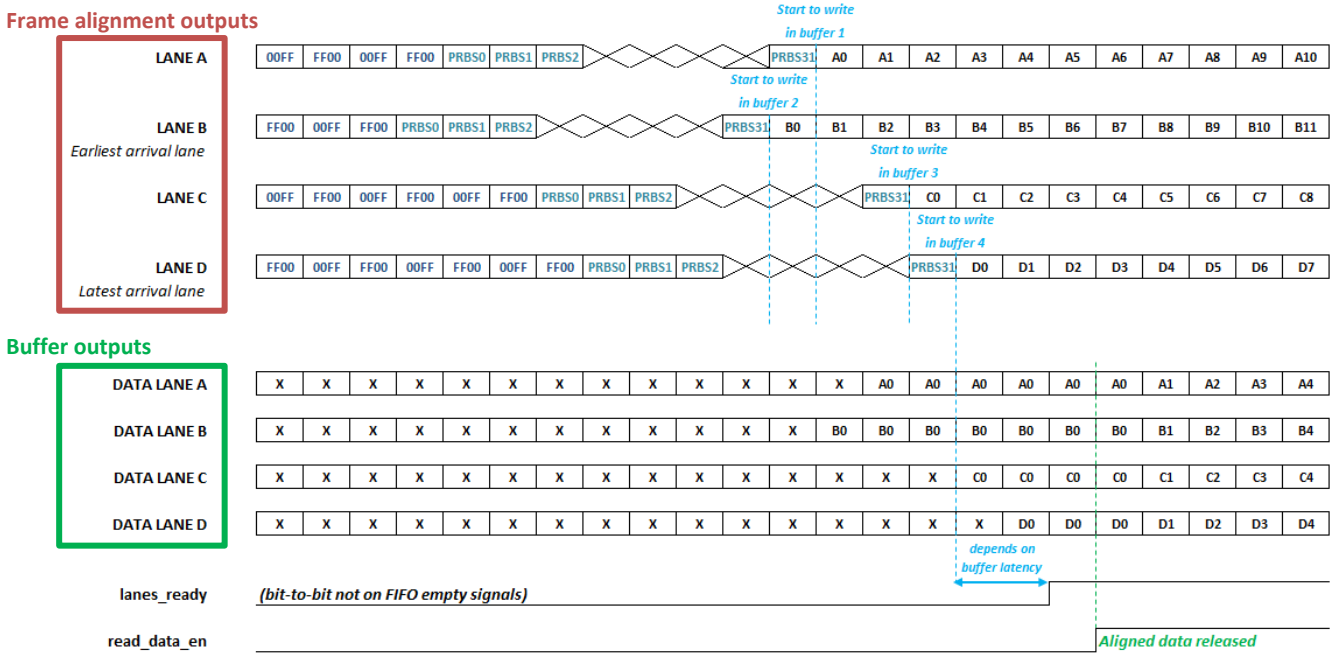


Figure 14: Multiple lane Synchronization, principle

1.2.14 Output buffer IP

File name	Description
output_buffer.xci	FIFO default configuration (parameters value can be modified) : • Interface Type: Native • FIFO implementation: Independent Clocks Builtin FIFO. • Read Mode: Standard FIFO • Write width: 16, means one FIFO per parallel ESIstream frame received every frame clock cycle. • Write Depth: 512, minimum value for Builtin FIFO. **Can be optimized using a Block RAM FIFO or a Distributed RAM FIFO depending on trade-offs on programmable logic resources utilization.** • Read Latency: 1. • Read Clock Frequency (MHz): 391 (> 390.625) • Write Clock Frequency (MHz): 391 (> 390.625) The write clock is in the “recovered” frame clock domain, rx_usrclk. The read clock is in the the “acquisition” frame clock domain, clk_acq. In some cases and depending on the application, the dual clock FIFO implemented in the output buffer can be replaced by a single clock FIFO or by shift registers to reduce the amount of logic resources used by RX IP. For more information, please contact us using ESIstream contact web page or at GRE-HOTLINE-BDC@Teledyne.com.

1.2.15 Delay in RX output buffer wrapper

File name	Description
delay.vhd in rx_output_buffer_wrapper	Synchronize valid data out flag signal with decoded data and decoded frame signals

2. USER GUIDE

2.1 Package overview

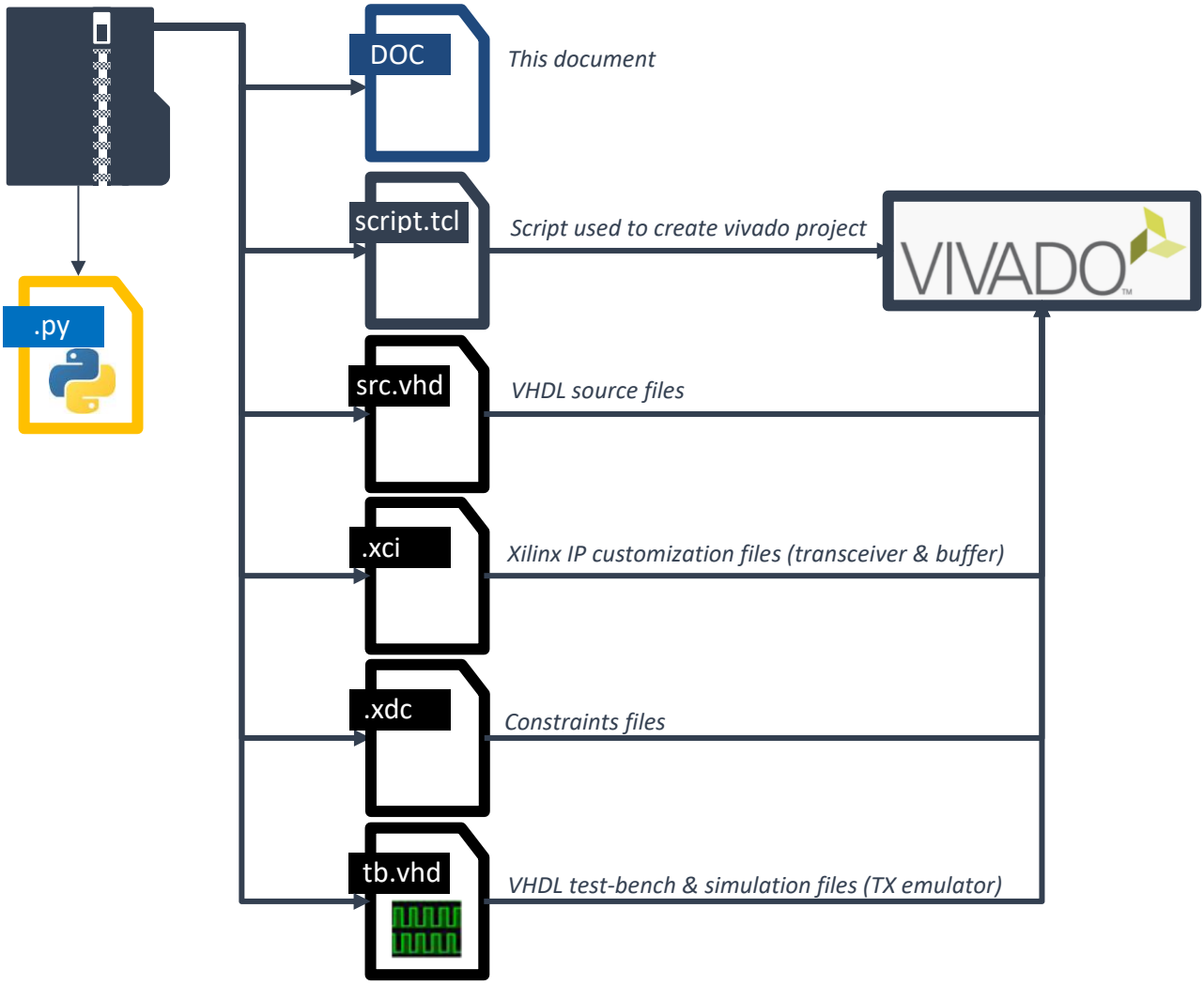


Figure 15: ESIstream package content overview

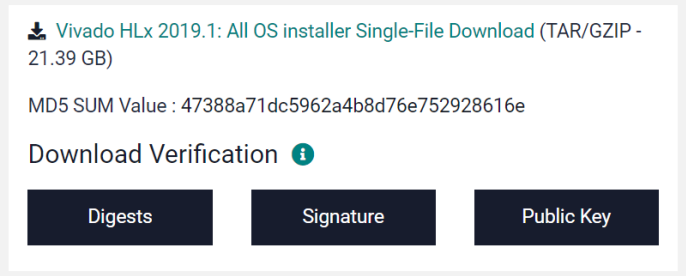
The table below provides an overview of folders organization in the package:

ESIstream package folder name	Description
src_common	Common VHDL source file: sync_generator, delay, synchronizer, spi_master, axi4_lite_master, uart
src_ip	FPGA IP files, generated with Vivado 2019.1 (ip-name.xci), gth transceiver, output buffer FIFO, adder, axi_uartlite, clocking wizards.
src_pkg	Common VHDL package files (esistream_pkg_32b /64b).
src_rx	ESIstream RX IP VHDL files.
src_rx_ip	ESIstream RX transceiver IP wrapper VHDL file.
src_tx	Reserved for future use.
src_tx_emulator	TX emulator VHDL files (no .xci transceiver IP). Only suitable for simulation purpose. Used in test-bench top file.
vivado_rx_aq600	<table><tr><th>Sub-folder name</th><th>Description</th></tr><tr><td>src_tb_top</td><td>VHDL Test-bench top file and vivado simulator wave file.</td></tr><tr><td>src_top</td><td>Project VHDL top files.</td></tr><tr><td>xdc</td><td>Project constraint files.</td></tr></table>
python	Software user interface using python scripts.

2.2 Vivado project

2.2.1 Vivado version and download link

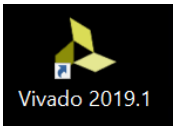
- Donwload and install **Vivado Design Suite – HLx Editions - 2019.1 Full product Installation.**
  - Use the link: https://www.xilinx.com/support/download.html



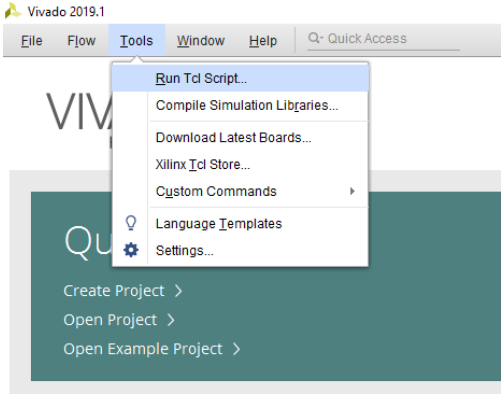
2.2.2 Generate the Vivado project

Tcl scripts (script_name.tcl) are provided and should be used to generate the project.
We recommend using Vivado version 2019.1. Updating the IP (in particular the GTH) to a more recent version may cause issues. Follow the procedure below to create automatically the Vivado project:

- Open Vivado 2019.1



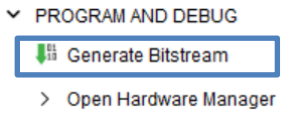
- Tools > Run Tcl Scripts…
  - Launch the package tcl script located in the directory vivado_rx_aq600/
  - There are four different projects (see section VHDL projects).
    -



When the project has been created and the IP generated, you can start compiling, simulating or modifying the example design like any Vivado project.
Note: If the TCL script is moved from its original folder it will not work as it uses relative path.

2.2.2.1 Generate the bitstream

- Click on *Generate Bitstream* in the *Flow Navigator* panel (*PROGRAM AND DEBUG*).



- Wait end of bitstream generation.
  - Check Synthesis, Implementation end without critical warnings.
  - Check there is no Timing error.

2.3 VHDL projects

2.3.1 Principle

All four projects use the same architecture:

- script_32b.tcl
- script_32b_dl.tcl
- script_64b.tcl
- script_64b_dl.tcl

See the section “Choosing the right project option” to understand projects differences.

The VDHL top file, rx_esistream_top.vhd, is located in vivado_rx_aq600/src_top folder:

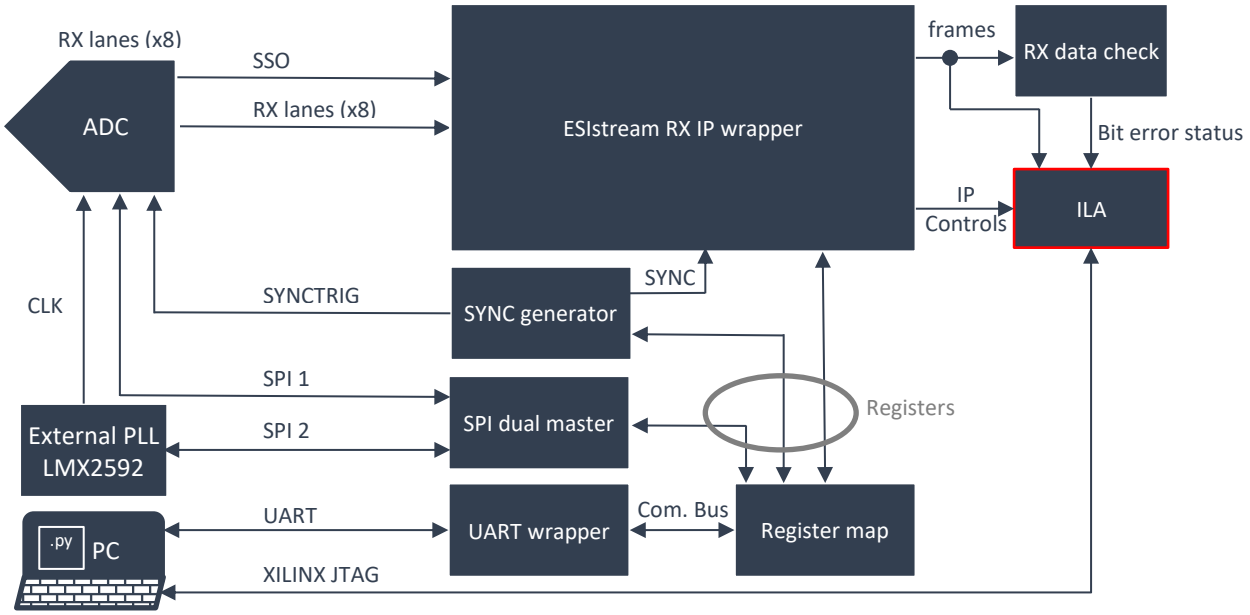


Figure 16: ESIstream RX IP project block diagram, principle.

The FPGA project is managed using a simple UART interface (8-bit, 115200) to access FPGA registers through read and write operations.
Python scripts examples are provided with the ESIstream IP package including:

- read uart function
- write uart function
- configure the external PLL to generate a 6.25 GHz CLK example
- link configuration example
- link synchronization example
- configure ADC in ramp mode example
- configure ADC in normal mode example
- configure ADC in pattern0 mode example
- get ADC ID example
- get VHDL version example

A small and easy to use frames layer protocol has been defined to communicate with the register map through the UART communication.
The register map contain all registers to control the design (ESIstream RX IP, SPI Masters & SYNC generator).
A module allows to scan the decoded data decoded in order to check if there is bit error. Bit error status, received frames, and ESIstream RX IP controls signals are mapped to Integrated Logic Analyzers IP to be analyzed in Vivado hardware manager using JTAG port.

2.3.2 Choosing the right project option

2.3.2.1 What does 32b or 64b indicate?

It indicates that, for each lane, the raw data logic vector received at the transceiver output will have a width of 32-bit when choosing a “32b script” or 64-bit when choosing a “64b script”.
At the ESIstream RX IP outputs, the decoded data will be distributed in parallel 16-bit frames

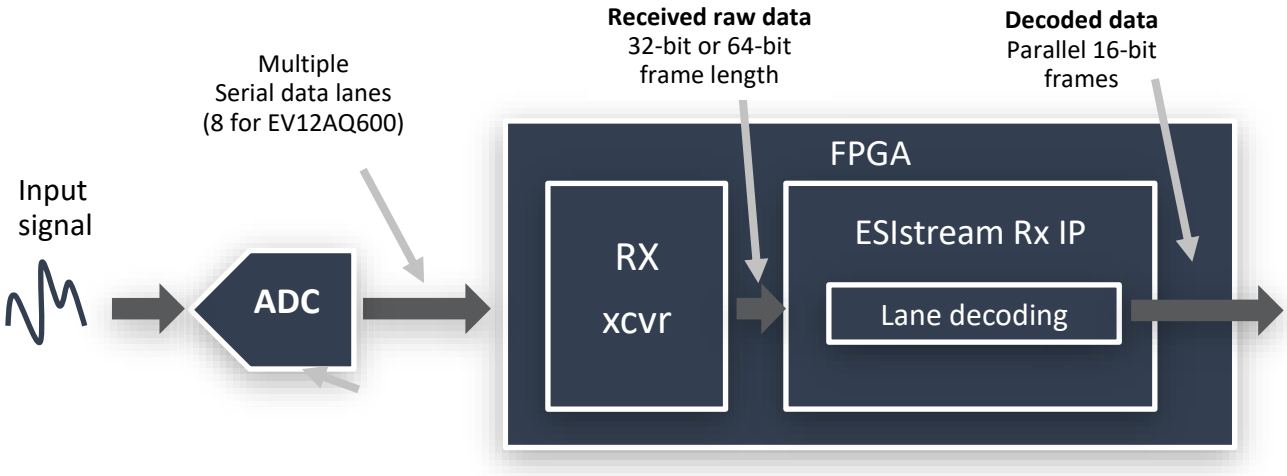


Figure 17: What does 32b or 64b indicate?

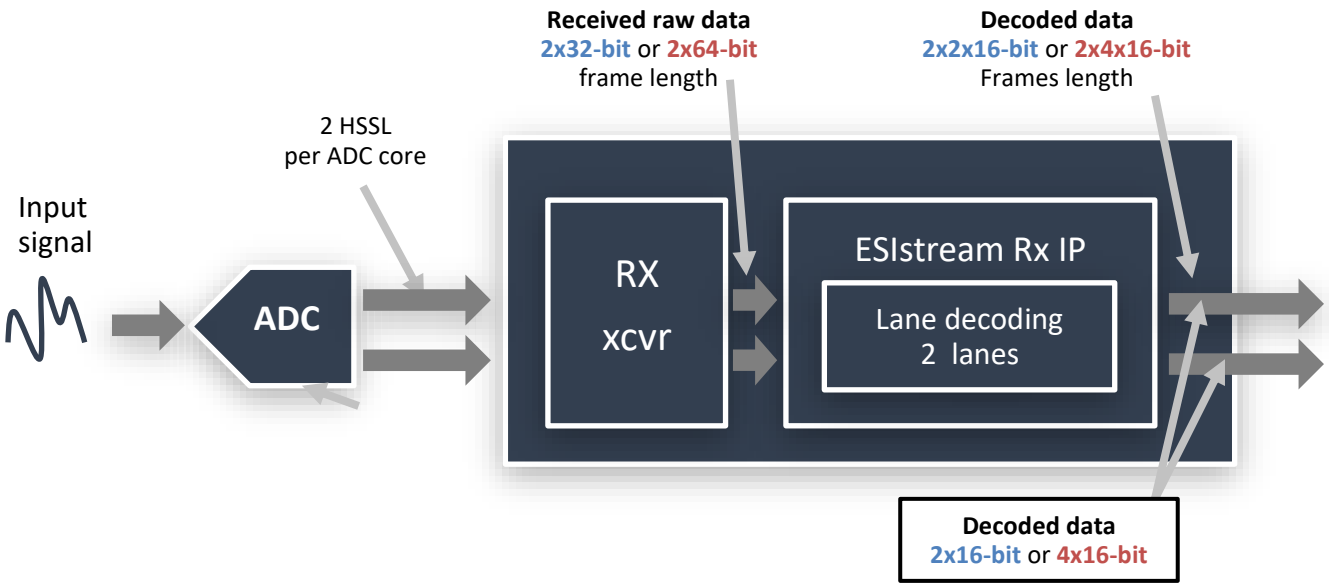


Figure 18: ESIstream RX IP data framing 32b or 64b, example with 2 lanes.

2.3.2.2 A trade-off between frame clock rate, logic resources and link latency

Choosing the right IP option is a trade-off between minimum link latency, minimum logic resources and lower frame clock rate in the FPGA allowing relaxing RX end design in terms of timing constraints.

- 32-bit wide configuration allows a design requiring a lower amount of logic resources and minimum link latency, in return the design will run at a higher frame clock rate than the 64-bit design. (Twice higher than the 64-bit design frame clock rate).

- 64-bit configuration allows a design less stressful with a frame clock rate running twice lower than the 32-bit design. This can be useful for FPGA with a small speed grade. In return, the amount of logic resources and the link latency are higher than the 32-bit design.

The table below shows the logic resources utilization of the ESIstream RX IP per projects (rx_esistream_with_xcvr.vhd) implemented on a Kintex Ultrascale xcku060-ffva1517-1-c (see Annex A: Logic resources utilization for more details).

Project	Device	Package	Speed grade	Number of Lanes	Max lane rate [Gbps]	LUTs KU060 (331680)	FFs KU060 (663360)	KU060 LUTs%	KU060 FFs%
16b	xcku060	FFVA1517	-1	8	6.25	1803	1854	0.54	0.28
32b	xcku060	FFVA1517	-1	8	12.5	3551	2514	1.07	0.38
32b_dl	xcku060	FFVA1517	-1	8	12.5	3543	2510	1.07	0.38
64b_dl	xcku060	FFVA1517	-1	8	12.5	5291	4222	1.60	0.64

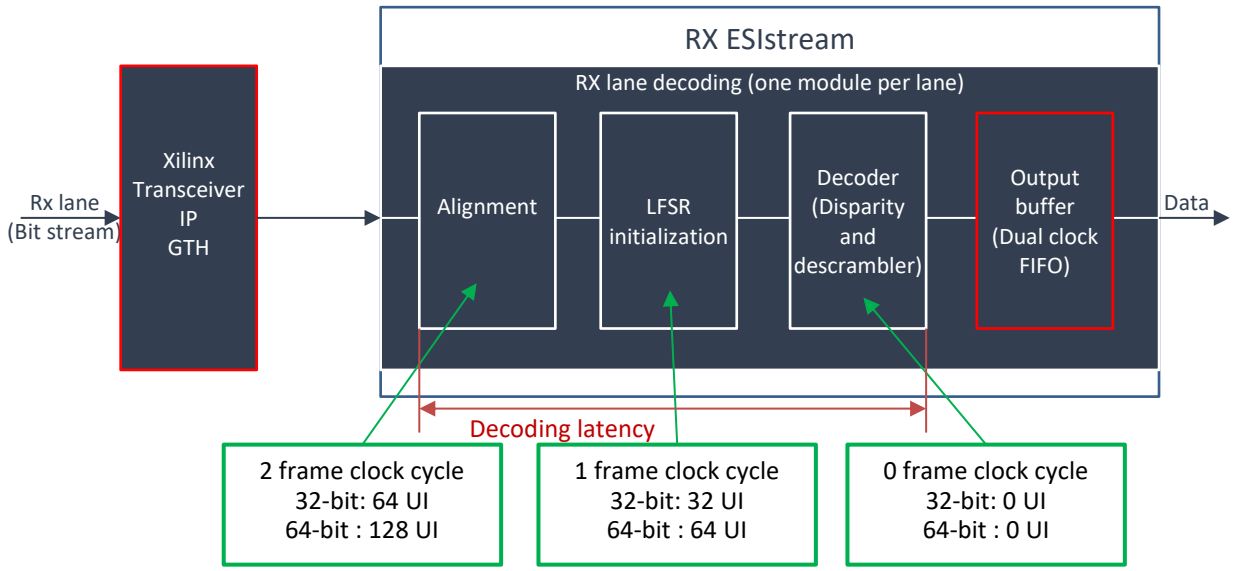


Figure 19: Data path, lane-decoding latencies per sub-modules.

For instance, at 12.5 Gbps per lane:

- Using a 64-bit implementation, the frame clk frequency is 195.3125 MHz and the decoding latency is 15.36 ns.
- Using a 32-bit implementation, the frame clk frequency is 390.625 MHz and the decoding latency is 7.68 ns.

When the application requires a 32-bit implementation with a deterministic low latency, driving the transceiver RXUSRCLK2 input with a synchronous frame clock allows replacing the dual clock FIFO output buffer by a single clock FIFO. The single clock FIFO can uses either built-in FIFO, distributed RAM or shift-register implementation to further reduce the total RX latency. The synchronous frame clock must be set at the right frequency, for instance 400 MHz @ 12.8 Gbps, and derived from the reference clock or from a global clock capable input.

2.3.3 UART frames layer protocol

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.

2.3.3.1 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 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).

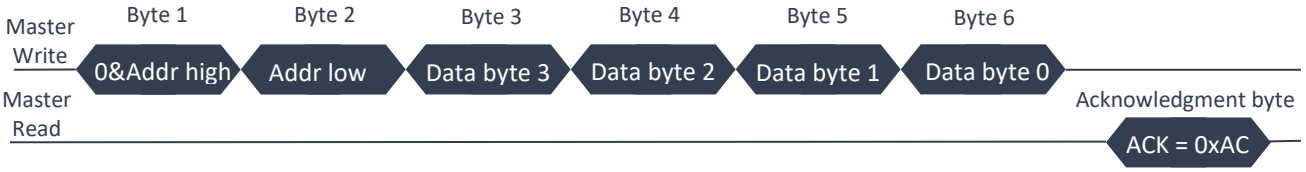


Figure 20: UART frames layer protocol, write operation

2.3.3.2 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 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.

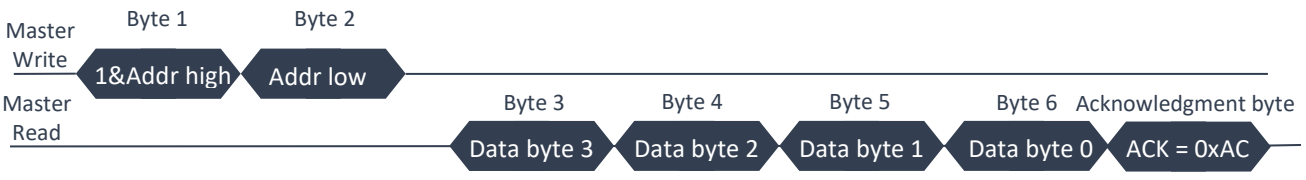


Figure 21: UART frames layer protocol, read operation

2.3.4 FPGA Register map

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

Signal name	Register address	Register Bits	Type	Size
-	0	31 - 0	W	Reserved
prbs_en	1	0	W	Enable ESIstream RX IP PRBS decoding when '1'.
reg_rst	2	0	W	Active high ('1') global software reset.
reg_rst_check	2	1	W	Active high ('1') RX data check module reset.
reg_aq600_rstn	2	2	W	Active low ('0') EV12AQ600 ADC reset.
rx_sync_rst	2	3	W	Active high ('1') SYNC generator module reset.
spi_ss	3	0	W	SPI slave select: - EV12AQ600 ADC when '0' - External PLL when '1'
spi_start	3	1	W	When '1' all SPI commands, pre-loaded in the SPI Master input FIFO (FIFO IN), are sent to the selected SPI slave.
fifo_in_din	4	23 – 0	W	SPI Master input FIFO data port. To write data through SPI. SPI commands must be pre-loaded in the SPI Master input FIFO. Then spi_start bit to send all commands through SPI.
sync_mode	5	0	W	SYNC Counter mode: normal: 0; training :1.
send_sync	6	0	W	When send_sync is set to high, the SYNC generator module detects the rising edge of the send_sync signal and starts sending the SYNC pulse both to the ESIstream RX IP and to the ADC. The SYNC pulse also starts the SYNC counter. The SYNC pulse allows synchronizing the serial link.
sync_delay	6	7 - 4	W	Apply delay, in number of clk_acq clock cycle between the synchronization signal sent to the ADC and the synchronization signal sent to ESIstream RX IP.
sync_wr_counter	7	7 - 0	W	Pre-load SYNC generator module counter value. In normal mode the SYNC counter starts to count when the SYNC pulse is generated and waits to reach this value to release the data from the output buffer.
sync_wr_en	7	8	W	Load sync_wr_counter from register to the counter end value logic. '1' must be apply to the pre-loaded value in sync_wr_counter register.
reg_8 (0x00000220)	8	31 - 0	R	FPGA firmware version
fifo_in_full	9	0	R	SPI Master input FIFO full flag. Full when '1'
fifo_out_empty	9	1	R	SPI Master output FIFO empty flag. Empty when '0'
fifo_out_dout	10	23 - 0	R	SPI Master output FIFO data port. Data read from the SPI are stored in this FIFO. After a SPI read operation data should be flushed performing "UART" read operation on this register until the fifo_out_empty register goes low.
sync_rd_counter	11	7 - 0	R	SYNC generator module memorized counter value after a SYNC pulse has been sent. Indicates the number of frame clock cycles, in the clk_acq domain, between the SYNC pulse and the instant the lanes_ready signals went up. In training mode: to evaluate the first data latency value. In normal mode: to check that the counter stops on the value defined in the sync_wr_counter register and release the date from the output buffer at the right instant.
sync_counter_busy	11	8	R	SYNC generator module counter is busy. Busy when '1'. Can be monitored to wait for the valid counter value.
-	255	31 - 0	R	Reserved

2.4 Testbench

The test-bench instantiates the project VDHL top file, rx_esistream_top.vhd, to be able to simulate the whole FPGA design detailed in this document.
The test-bench instantiates a TX emulator replacing the EV12AQ600 ADC and which acts like a simplified model of the ADC, configurable both in ramp mode or in pattern0 mode, allowing generating data through the simulated serial link.
The link synchronization can be fully simulated.
A UART wrapper instance also allows simulating the PC to FPGA communication. Two VHDL procedures, one for read operation and one for write operation, encapsulate UART frames protocol layer.
To simulate the UART slow communication compare to the ESIstream high-speed serial interface, it is recommended to deactivate the simulation of ESIstream in the test-bench setting the constant GEN_ESISTREAM to false. This allows speeding the simulation run time. Do not to forget to reset GEN_ESISTREAL to true for serial link simulation.

The *my_tb* process generates the test-bench stimulus. It is composed of two parts:

- TEST 1: to simulate the serial link
  - GEN_ESISTREAM must be set to true.
  - To speed-up simulation, registers are not used to configure controls signals. FPGA GPIOs are used in simulation instead of register accesses.
- TEST 2: to simulate the serial communication.
  - To speed-up the simulation run time:
    - GEN_ESISTREAM should be set to false.
    - TEST 1 should be commented.

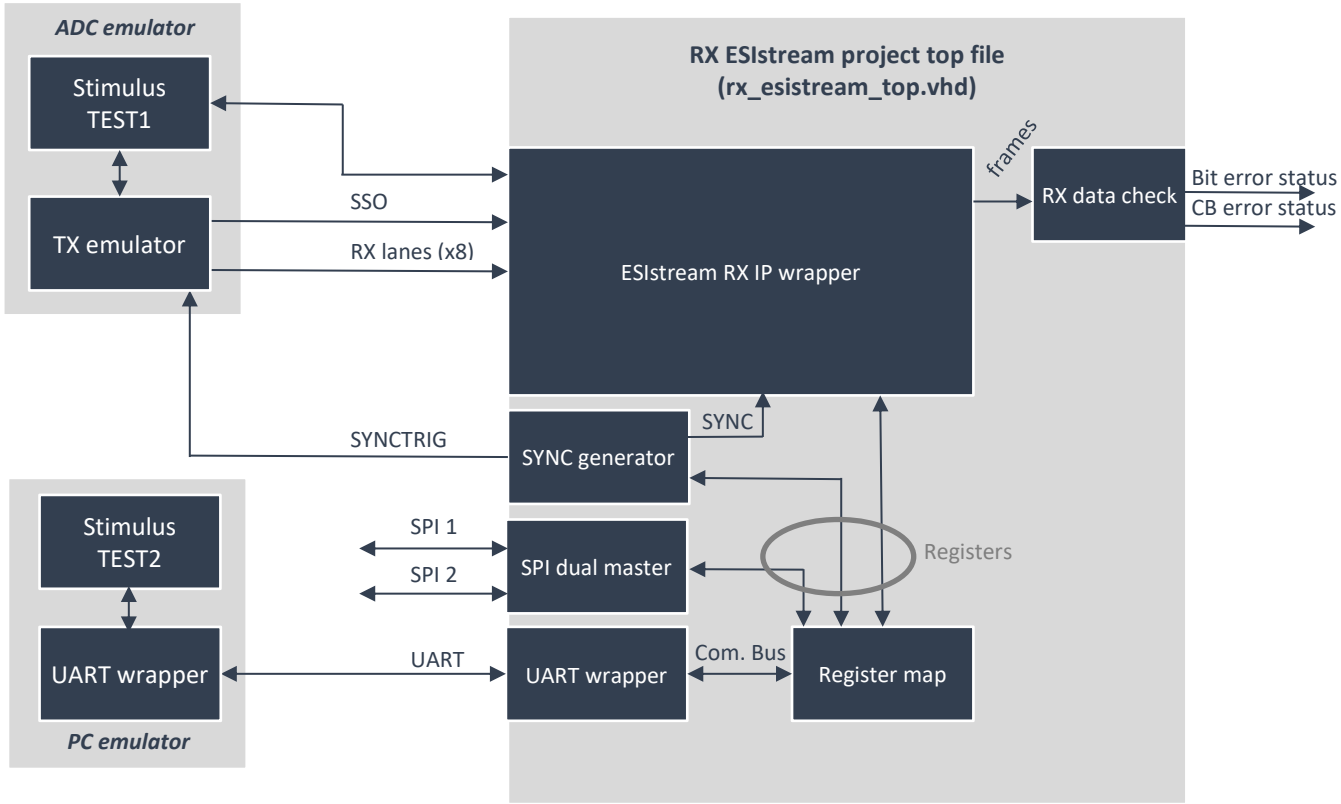


Figure 22: Test-bench Block diagram, principle

2.4.1 TEST 1

2.4.1.1 TX emulator

The TX emulator embeds a data generator module generating known data pattern at the transmitter emulator outputs. According to the value of the data generator control input signal *tx_d_ctrl,* the data to encode can be a 12-bit positive ramp, a pattern0 or a pattern1.

tx_d_ctrl value	Waveform encoded by the TX emulator module
"00"	Pattern0: "00000000000000"
"01"	12-bit positive ramp.
"10"	Reserved
"11"	Pattern1: "11111111111111"

The validation module (rx_check) allows checking received data values. It reports an error when there is a bad value in one of the received data field or in one of the clock bit field of the received ESIstream frames.

Status name	Description
Bit error status	'0': No error '1': Data bit error
CB error status	'0': No error '1': Clock bit error

2.4.1.2 Flow chart

Figure 23: Test-bench flow chart, TEST 1

At the end of the TEST 1, check bit error status, clock bit status and frames values.

2.5 Hardware project setup

- **Warning: To avoid power supplies hot plug, check all power supplies and power switches positions are OFF before connecting any FMC Mezzanine board on a FMC carrier board.**

- **Warning: Use the correct hardware configuration for each bitstream.**

For more information on the Teledyne-e2v high-speed ADC EV12AQ600 FMC board or on the hardware setup, please contact Teledyne-e2v support at GRE-HOTLINE-BDC@Teledyne.com.

2.5.1 Kintex Ultrascale - KU060

The design is compatible with and tested on the FPGA evaluation kit ADA-SDEV-KIT2, a development platform for space grade FPGA systems from ALPHA DATA.

2.5.2 Kintex Ultrascale - KU040

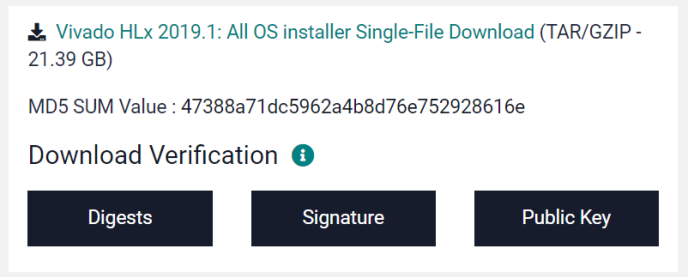
The design is compatible with and tested on the FPGA evaluation kit KCU105, a development environment for evaluating Kintex UltraScale FPGAs from Xilinx.

2.5.3 Virtex 7 - 7VX690

The design is compatible with and tested on the FPGA evaluation kit VC709, a development environment for evaluating Virtex7 FPGAs from Xilinx.

2.6 Software project setup

- Donwload and install **Vivado Design Suite – HLx Editions - 2019.1 Full product Installation.**
  - Use the link: https://www.xilinx.com/support/download.html
  - Vivado Design suite requires a Vivado license.



- Install python environment following the latest version of the document ESIstream_Install_Python_Environment available on www.ESIstream.com. Python scripts offer a better user experience compared to push-buttons, switches and led only.

- Create vivado project using one of the tcl scripts available in each package.
  - Create the bitstream.
  - Load bitstream through JTAG interface using Vivado hardware manager and Xilinx Platform Cable USB II.
- Download and unzip the ev12aq600_python_v1-0 python scripts package.
- Go to the unzipped folder location and open a cmd or a Powershell prompt:
  - Launch python scripts in this order:
    - Python> python .\get_status.py

```
Python> python .\get_status.py


------------------------------------------------------------
-- Serial communication opened... True
-- Status: 0x20152018
-- HDL version: 0x220
-- Get Chip ID (2324):
addr = 17
-- (v) Chip ID valid
-- Serial communication closed... True
------------------------------------------------------------
```

    - Python> python .\ev12aq600_external_pll_6250.py
      - To configure start the external PLL providing the ADC master clock.
- Re-program FPGA from Vivado Hardware Manager to use Vivado Integrated Logic Analyzer (ILA) which needs a free running clock. Then, re-launch python scripts in this order:
  - Python> python .\get_status.py
  - Python> python .\ev12aq600_link_config.py
    - Synchronize ESIstream link between the FPGA and the ADC EV12AQ600.
  - Python> python .\ev12aq600_ramp-mode.py
    - Configure the ADC EV12AQ600 in ramp mode to generate known patterns (see EV12AQ600 datasheet).

Using ILA in hardware manager, received data can be monitored. Checking signals ber_status and cb_status allow ensuring there are no errors on received data.

Siognal	Description
ber_status	'0': No error '1': Data bit error
cb_status	'0': No error '1': Clock bit error

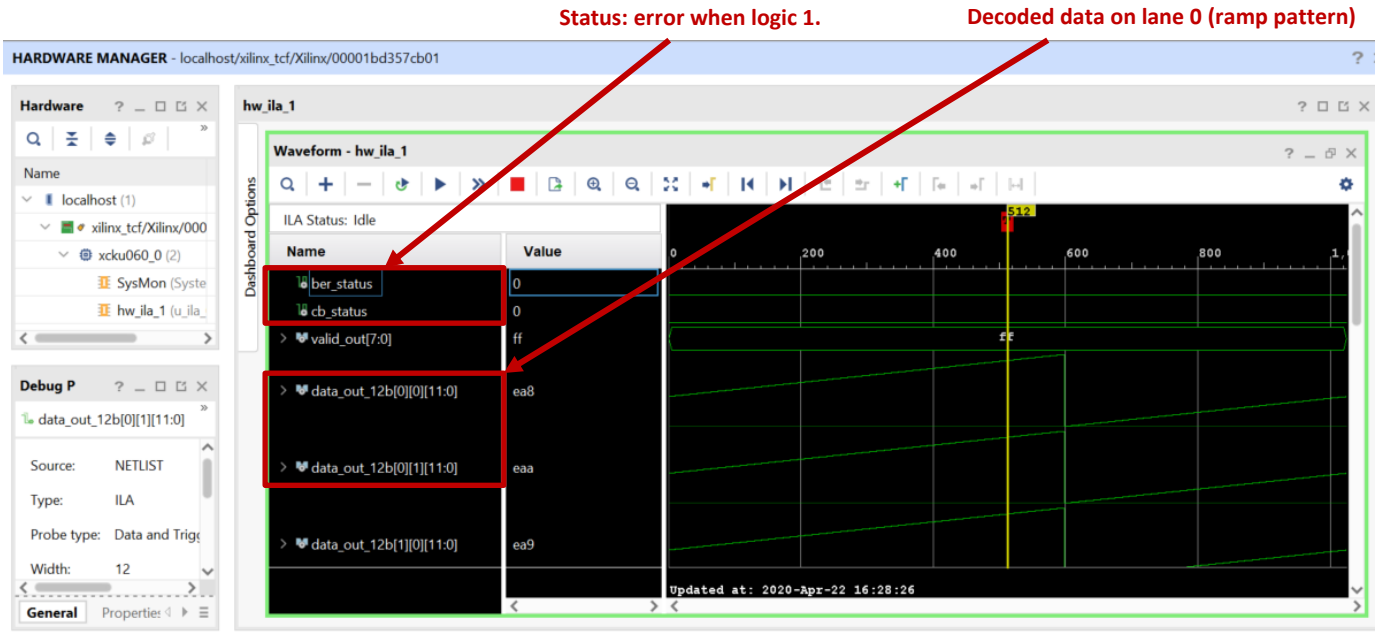
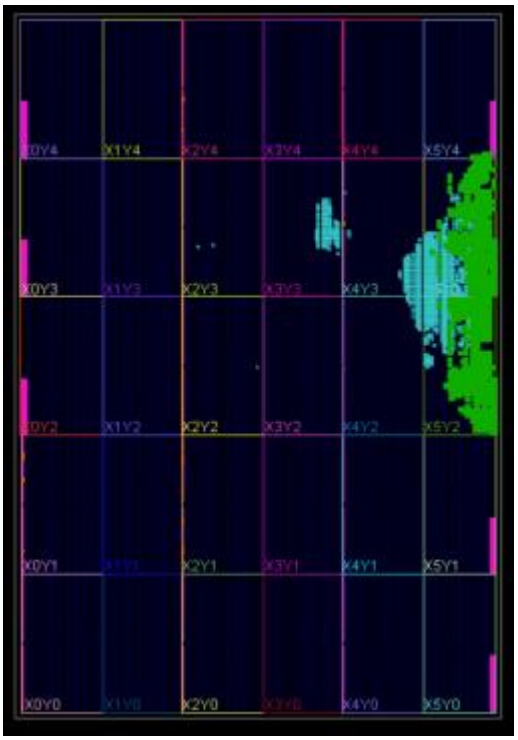
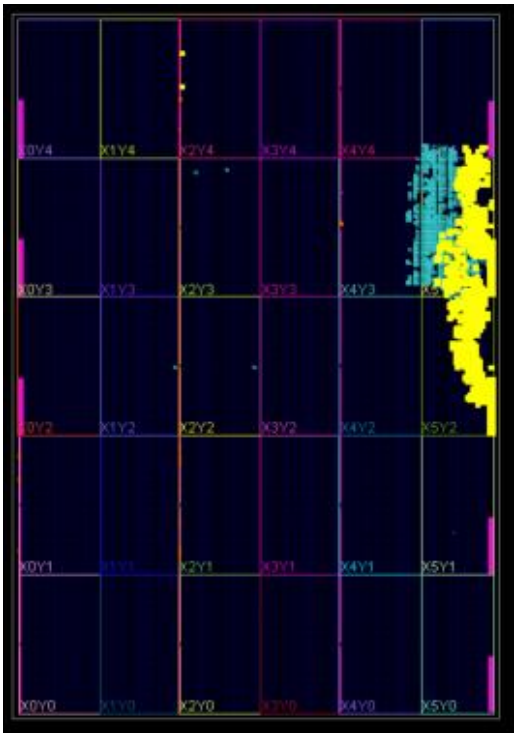


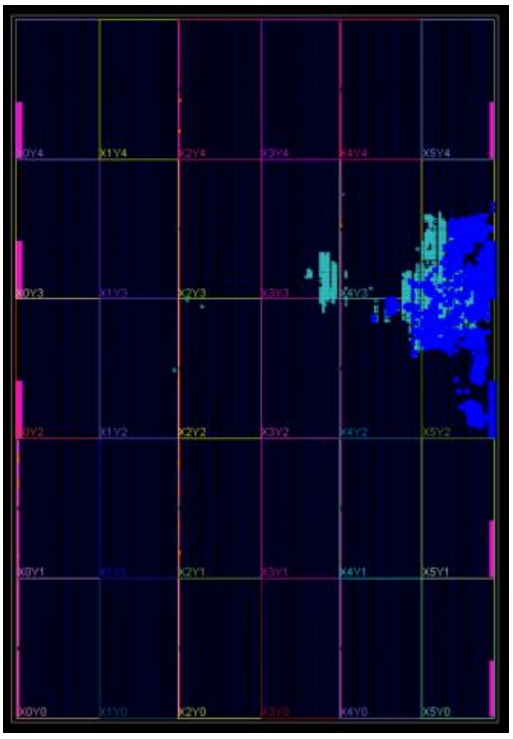
Figure 24: ILA waveforms – script_32b.tcl



script_64b_dl .tcl



script_32b_dl .tcl



script_32b .tcl

Figure 25: Device floorplan utilization per project

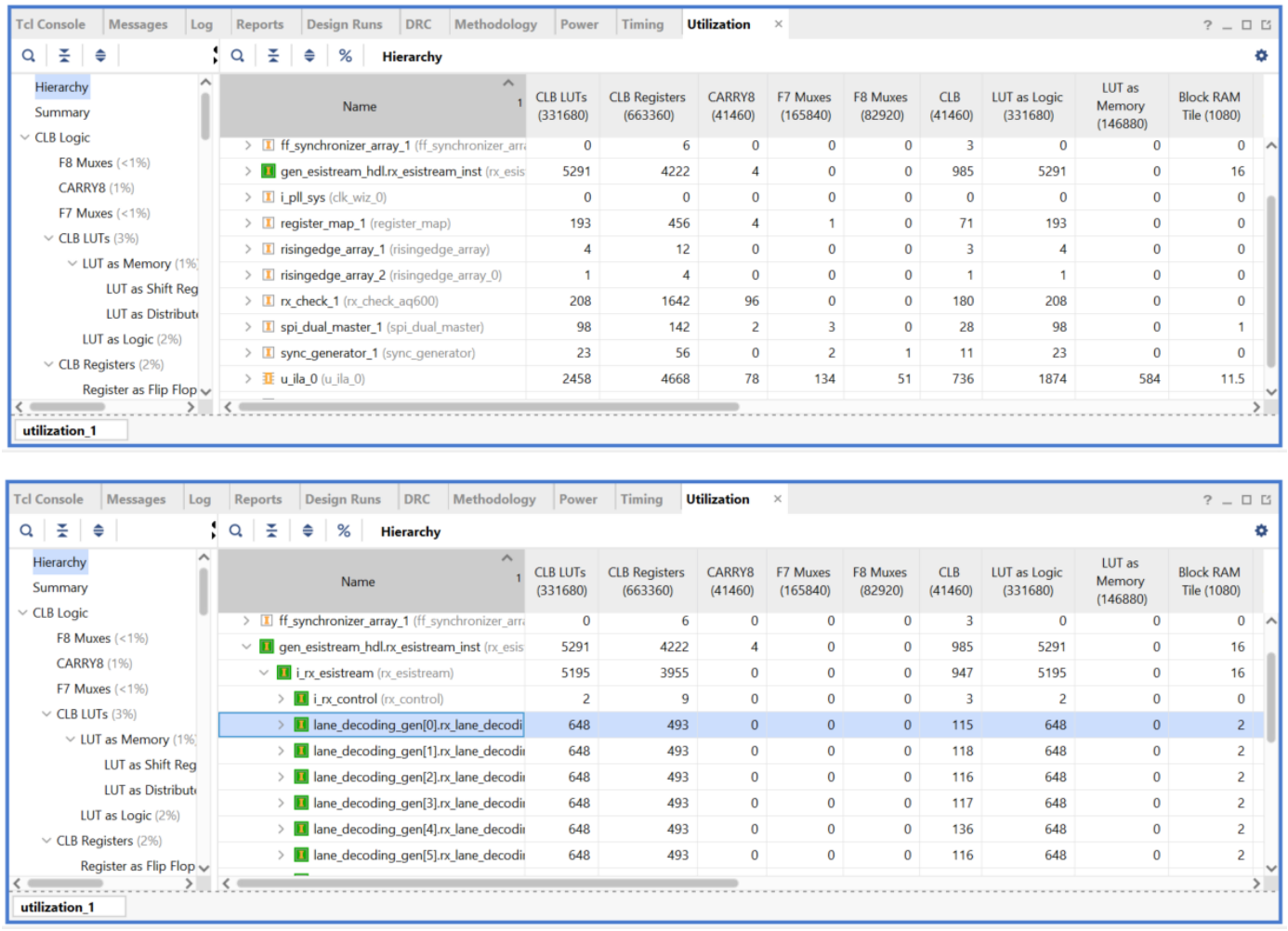
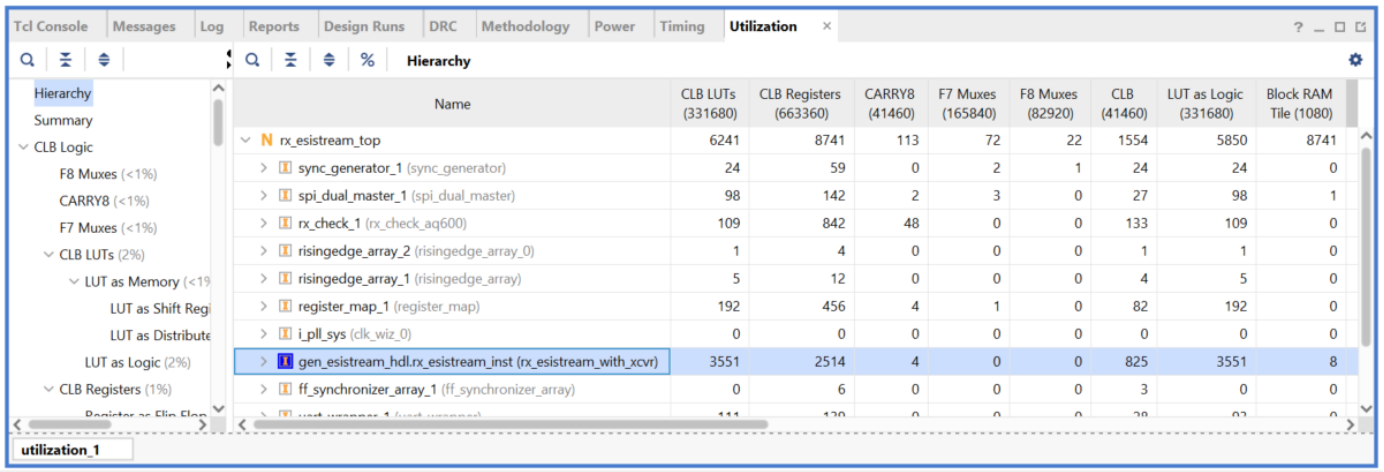


Figure 26: script_64b_dl.tcl, Hierarchy logic resources utilization



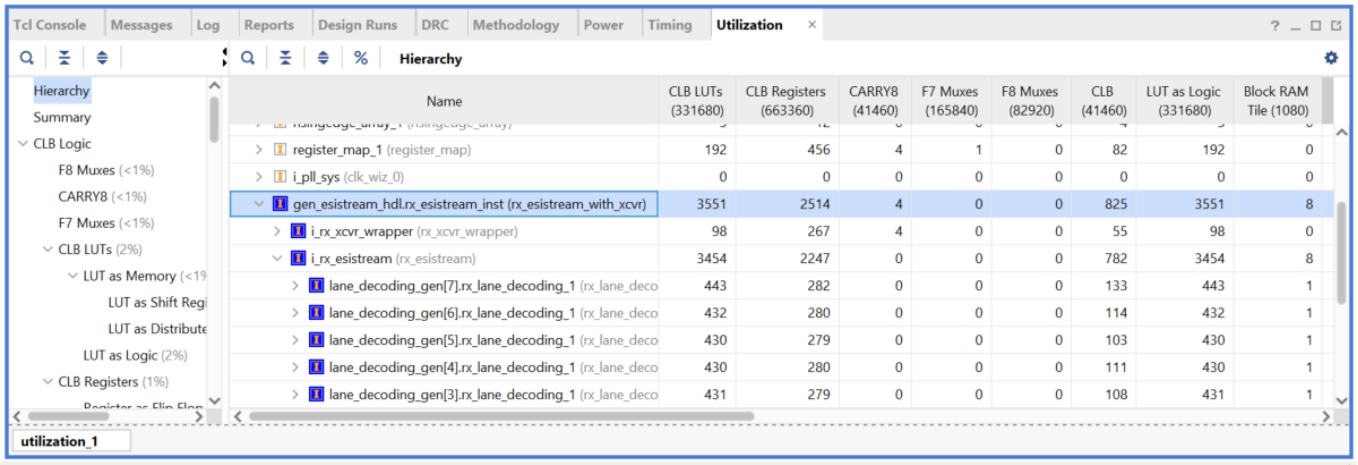
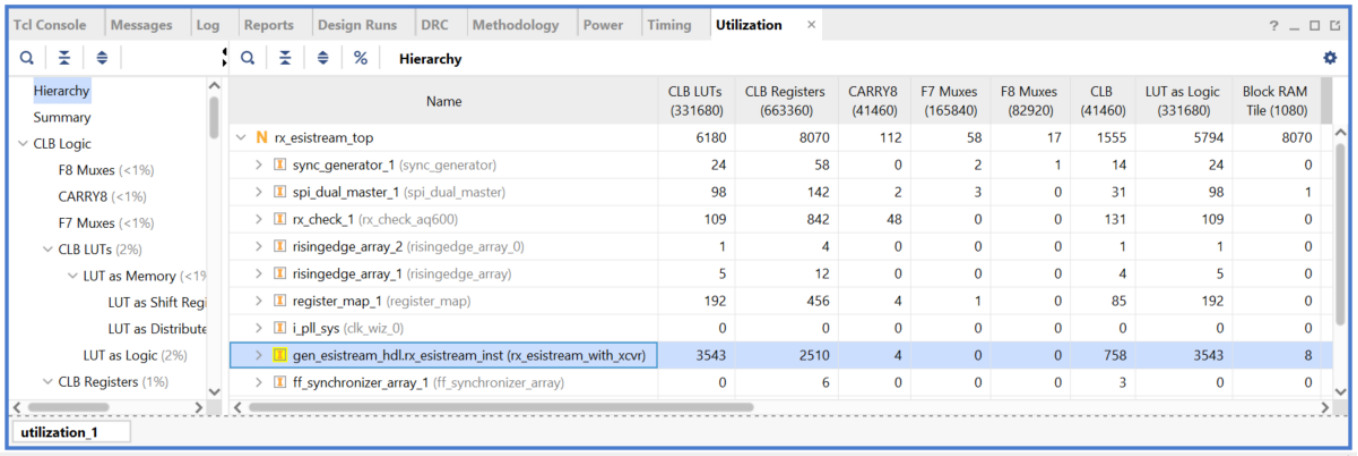


Figure 27: script_32b.tcl, Hierarchy logic resources utilization



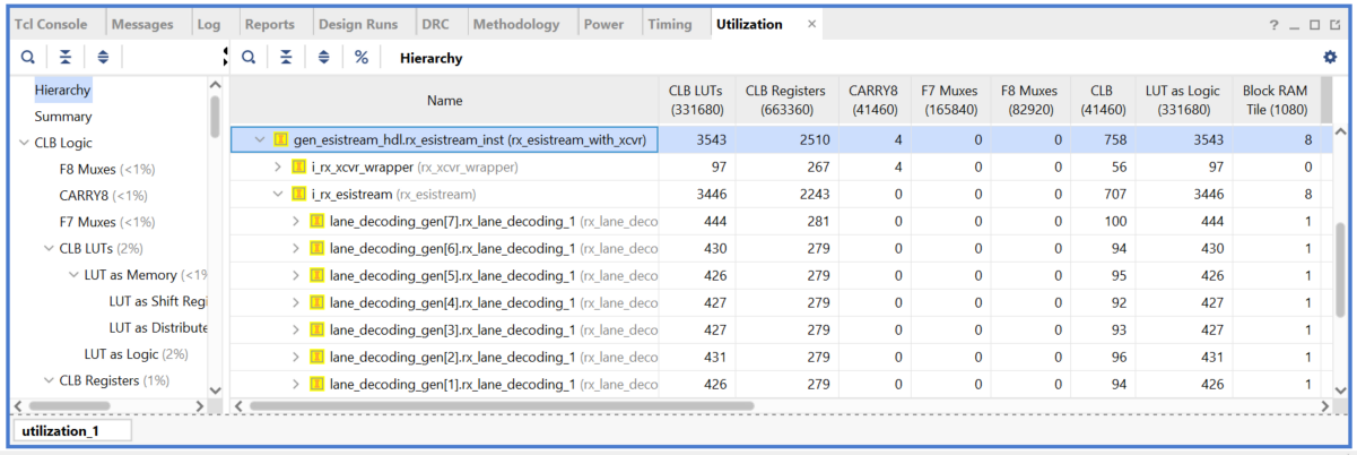


Figure 28: script_32b_dl.tcl, Hierarchy logic resources utilization

1- Create the Vivado project using one of the project script, see section Generate the Vivado project). Check Simulation Sources and make sure *tb_rx_esistream_top* is active.
2- In the *Flow Navigator*, right click on *SIMULATION\Run Simulation*
3- Click on *Run behavioral Simulation*.

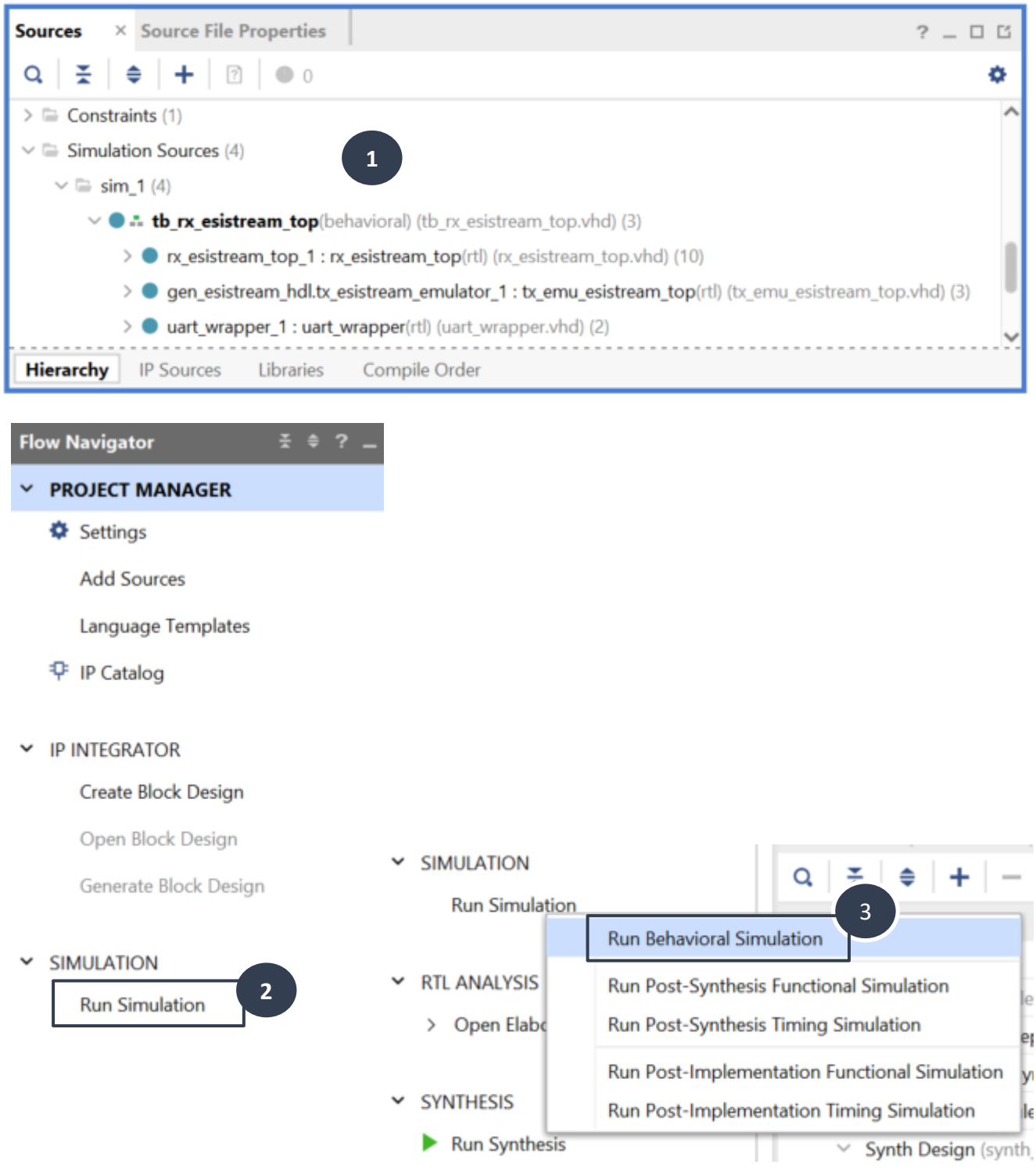


Figure 29: Steps to launch a behavioral simulation in Vivado

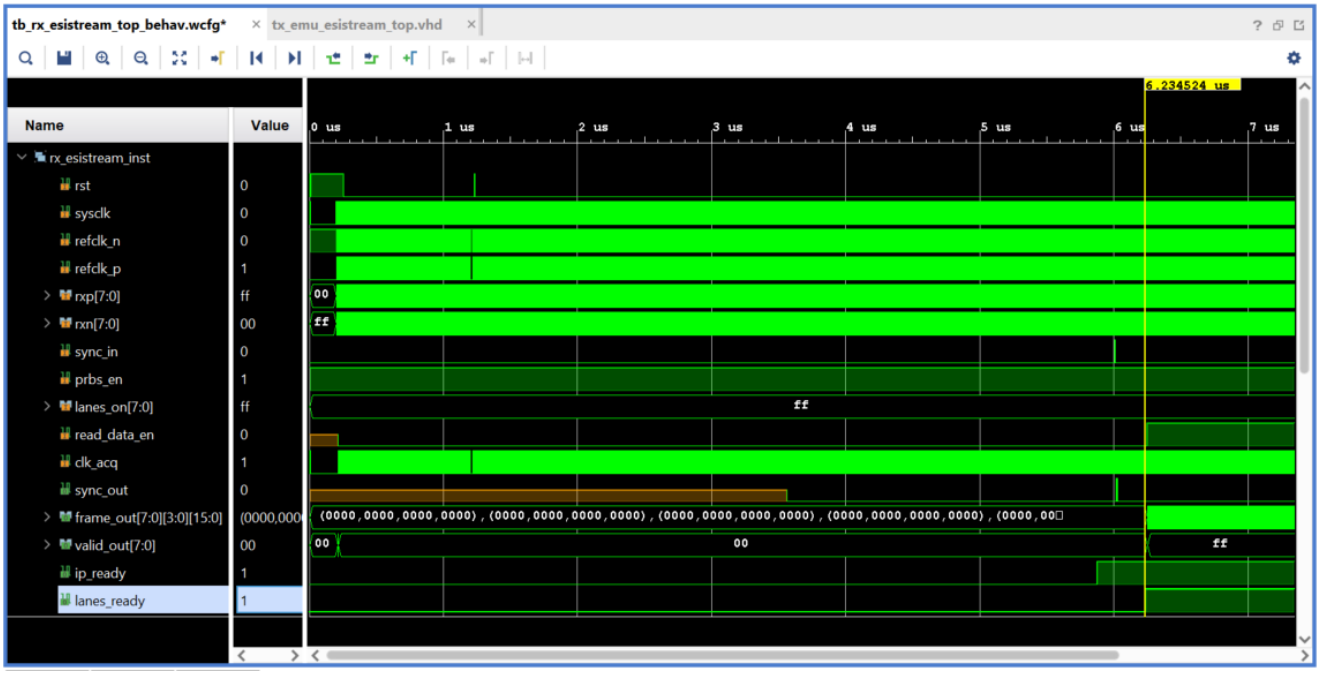


Figure 30: TEST 1 simulation minimum run time 6500 ns

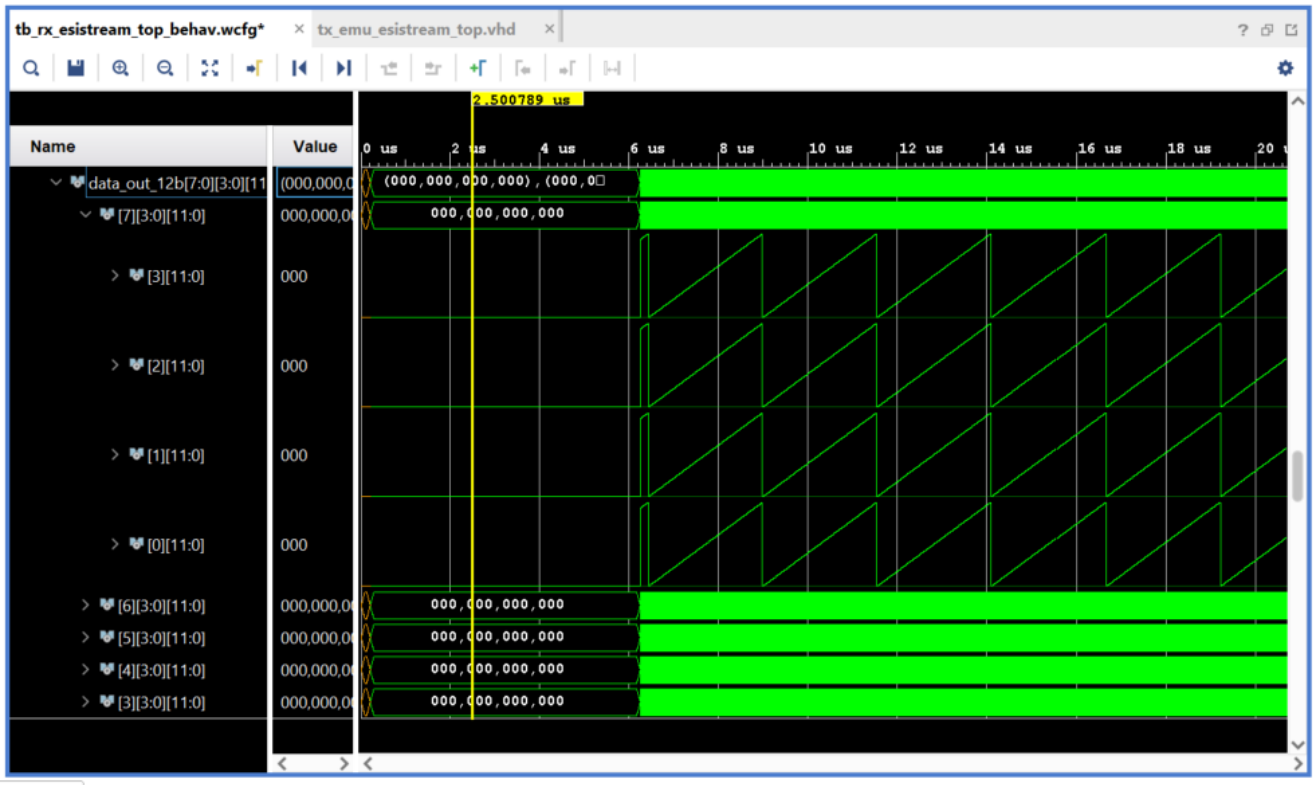


Figure 31: Ramp test mode & 64-bit project: Lane 7, RX data output frames sent to the ILA

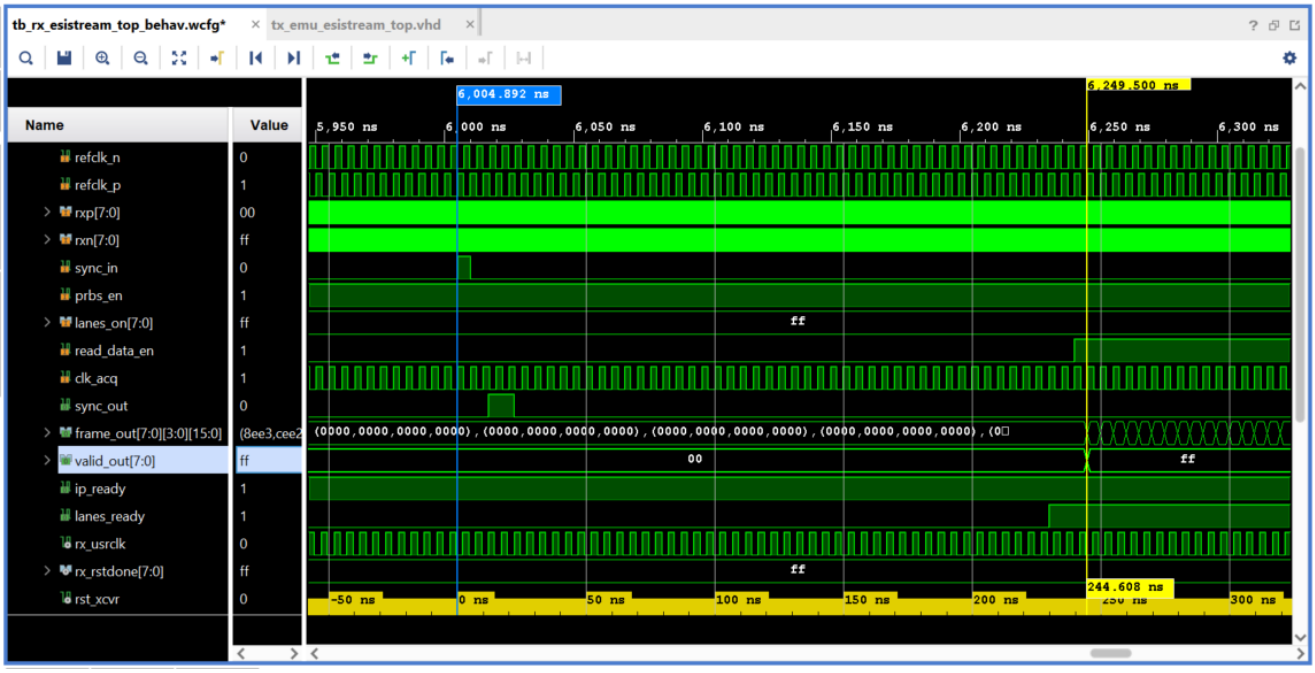


Figure 32: 64-bit RX ESIstream IP synchronization time: 245 ns from SYNC pulse to first valid decoded data.

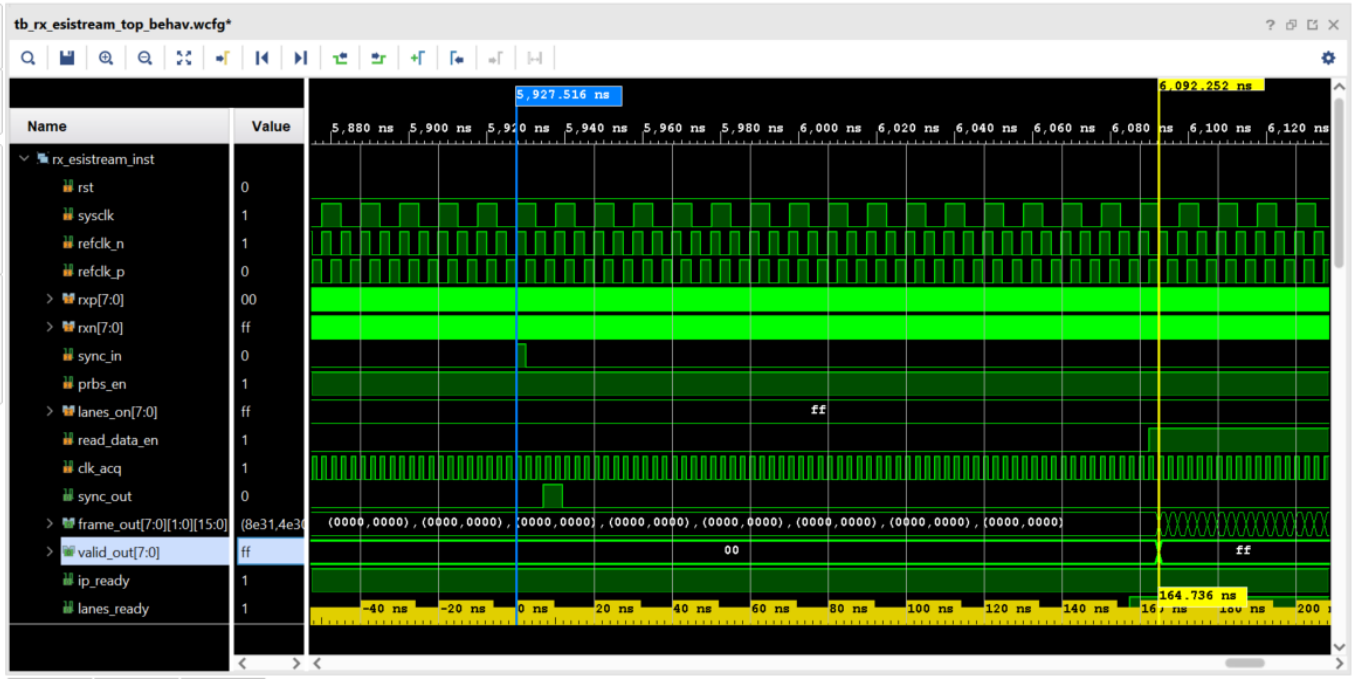


Figure 33: 32-bit RX ESIstream IP synchronization time: 165 ns from SYNC pulse to first valid decoded data.