WWW.ESISTREAM.COM

ESIstream

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

ESIstream package V3 – User guide

ESIstream - 2020

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 generating VHDL design examples of the ESIstream serial interface.
The package ESIstream provides VHDL sources, constraints files, TCL scripts for project generation, VHDL testbench files for simulation and python scripts.

Although the different VHDL design examples are specific to a FPGA reference, the modular architecture allows an easy migration to a different FPGA reference replacing the FPGA manufacturer 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

Package_ESIstream_Xilinx_xc7vx690tffg1761-2
Package_ESIstream_Xilinx_xcku040-ffva1156-2-e
Package_ESIstream_Xilinx_xcku060-ffva1517-1-c
Package_ESIstream_Xilinx_xcvu9p-flga2104-1-e

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
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
HSSL	High Speed Serial Lane
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 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 ≥ 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.

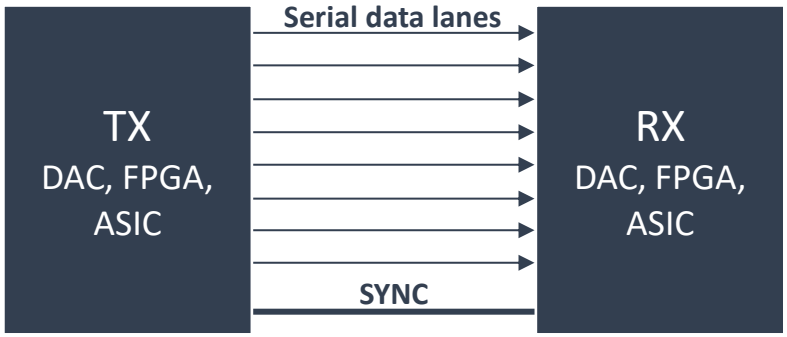


Figure 1: Basic ESIstream system

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.

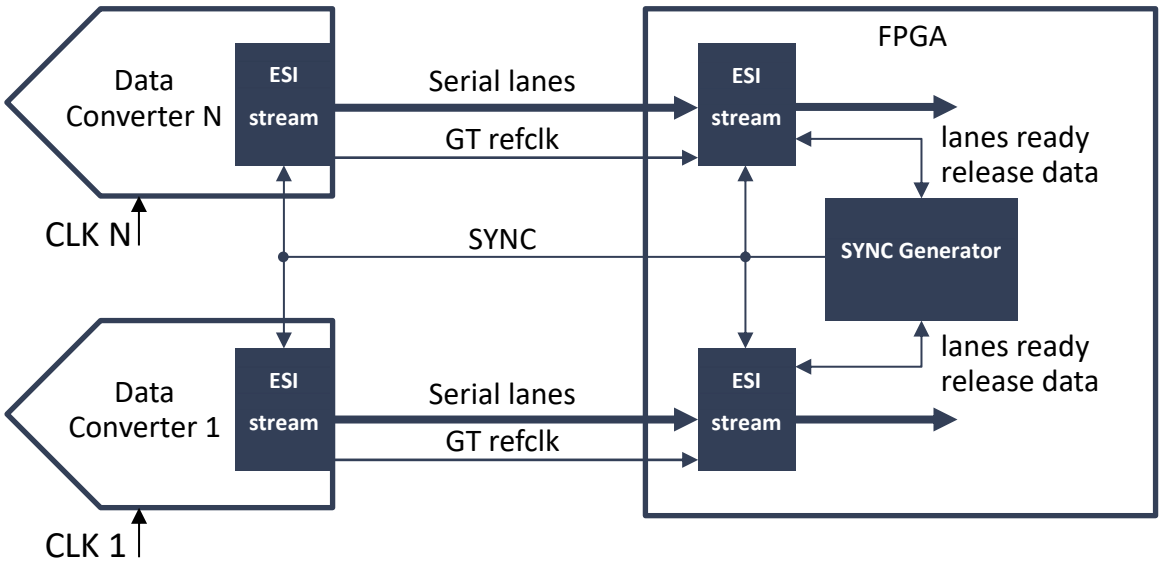


Figure 2: Multiple ADCs ESIstream system

License free

Anyone can download ESistream IP package on www.esistream.com/ip-package.
Users can send their contribution through ESistream contact web page.

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.

ESI
stream

TABLE OF CONTENTS

FIGURES

Documents Amendment

Issue	Date	Comments
A.0	March 2021	Creation

1. USER GUIDE

1.1 Package organization

All ESIstream packages are organized as follows:

Figure 3: ESIstream package organization.

<table>
<tr><th>Folder name</th><th>Description</th></tr>
<tr><td>python</td><td>
<table>
<tr><th>Sub-folder name</th><th>Description</th></tr>
<tr><td>api</td><td>Contain python scripts used to run ESIstream TX and RX through UART interface.</td></tr>
<tr><td>bat</td><td>Execute vivado from Tcl scripts.</td></tr>
<tr><td>build.py</td><td>In a cmd prompt:
- Use 'python build.py prj' to create vivado project.
- Use 'python build.py sim' to launch testbench simulation.
- Use 'python build.py gen' to launch bitstream generation.
- Use 'python build.py all' to create vivado projects, launch testbench simulations and generate Bitstream.</td></tr>
</table>
</td></tr>
<tr><td>src_[…]</td><td>Contain Vhdl source files (.vhd) and Vivado IP .xci files.</td></tr>
</table>

<table>
<tr><th>Folder name</th><th>Description</th></tr>
<tr><td>vivado_rx_ev12aq60x</td><td>ESIstream RX vhdl design example project dedicated for EV12AQ60x ADC.

<table>
<tr><th>Sub-folder name</th><th>Description</th></tr>
<tr><td>src_tb_top</td><td>Contain VHDL testbench files and vivado simulator waveforms file.</td></tr>
<tr><td>src_top</td><td>Contain project VHDL top file.</td></tr>
<tr><td>xdc</td><td>Contain project constraint file.</td></tr>
<tr><td>script_32b_dl.tcl</td><td>Script to build the project with an ESIstream RX using a 32-bit frames path width for each HSSL.</td></tr>
<tr><td>script_64b_dl.tcl</td><td>Script to build the project with an ESIstream RX using a 64-bit frames path width for each HSSL.</td></tr>
</table></td></tr>
<tr><td>vivado_txrx_xm107</td><td>ESIstream TX and RX vhdl design example project using XM107 loopback board

<table>
<tr><th>Sub-folder name</th><th>Description</th></tr>
<tr><td>src_tb_top</td><td>Contain VHDL testbench files and vivado simulator waveforms file.</td></tr>
<tr><td>src_top</td><td>Contain project VHDL top file.</td></tr>
<tr><td>xdc</td><td>Contain project constraint file.</td></tr>
<tr><td>script_16b_dl.tcl</td><td>Script to build the project with an ESIstream RX and ESIstream TX using a 16-bit frames path width for each HSSL.</td></tr>
<tr><td>script_32b_dl.tcl</td><td>Script to build the project with an ESIstream RX and ESIstream TX using a 32-bit frames path width for each HSSL.</td></tr>
<tr><td>script_64b_dl.tcl</td><td>Script to build the project with an ESIstream RX and ESIstream TX using a 64-bit frames path width for each HSSL.</td></tr>
</table></td></tr>
</table>

1.2 Vivado projects

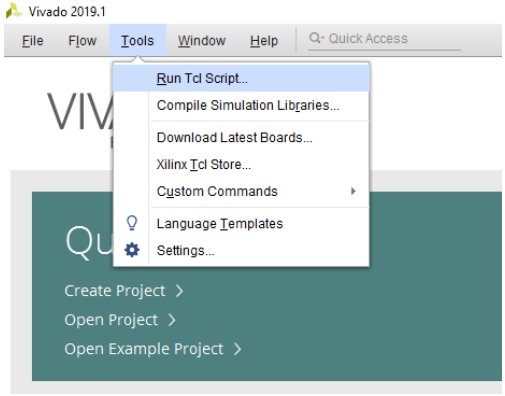
1.2.1 Build Vivado project

1.2.1.1 Method 1: Using Vivado GUI

- Open Vivado:

Package name	Vivado version to use
Package_ESIstream_Xilinx_xc7vx690tffg1761-2	Vivado 2019.2
Package_ESIstream_Xilinx_xcvu9p-flga2104-1-e	Vivado 2019.2
Package_ESIstream_Xilinx_xcku040-ffva1156-2-e	Vivado 2019.1
Package_ESIstream_Xilinx_xcku060-ffva1517-1-c	Vivado 2019.1

- Tools > Run Tcl Scripts…

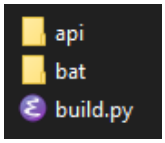


- Select the tcl script that fit your design requirements.

When the project is built, 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.

1.2.1.2 Method 2: Using python script and Vivado without GUI

- Open ESIstream package python folder:



- Open build.py
  - Configure the project(s) to build:
    - With the lines below 5 projects will be generated:
      - Vivado_txrx_xm107 script_16_dl.tcl
      - Vivado_txrx_xm107 script_32_dl.tcl
      - Vivado_txrx_xm107 script_64_dl.tcl
      - Vivado_rx_ev12aq60x script_32_dl.tcl
      - Vivado_rx_ev12aq60x script_64_dl.tcl

```
65 #
66 # Valid implementation:
67 hw_project_list = ["vivado_txrx_xm107", "vivado_rx_ev12aq60x"]
68 implementation_list = [["script_16b_dl.tcl", 10, 0],
69                        ["script_32b_dl.tcl", 10, 10],
70                        ["script_64b_dl.tcl", 10, 10]]
71 #
```

    - With the lines below 1 project will be generated:
      - Vivado_rx_ev12aq60x script_64_dl.tcl

```
72 hw_project_list = ["vivado_rx_ev12aq60x"]
73 implementation_list = [["script_64b_dl.tcl", 10, 0]]
```

Note: Value 10 indicates the simulation time in µs when executing ‘python build.py sim’. You can change this value to increase the simulation time. If this value is set to 0 then the project is not built when executing ‘python build.py prj’.

- Open a cmd prompt here:
    - python build.py need an argument (prj, sim, gen or all). See arguments description below:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.18363.1379]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Travail\ESIstream\Package_ESIstream_Xilinx_xcku060-ffva1517-1-c v3.0\python>python build.py
------------------------------------------------------------
-- START of PYTHON BUILD.PY ARG without argument...
--
-- use 'python build.py prj' to create vivado project
-- use 'python build.py sim' to launch testbench simulation
-- use 'python build.py gen' to launch bitstream generation
-- use 'python build.py all'  to create vivado projects, launch testbench simulations and generate bitstream
--
------------------------------------------------------------
-- exit on error: python script argument is missing...

C:\Travail\ESIstream\Package_ESIstream_Xilinx_xcku060-ffva1517-1-c v3.0\python>python build.py prj
```

- Open a cmd prompt here:

1.2.2 Vivado projects Tcl scripts differences

The raw data logic vector at Gigabit Transceiver (GT) output when ESIstream RX, and GT input when ESIstream TX can be configured to be 16-bit, 32-bit or 64-bit.

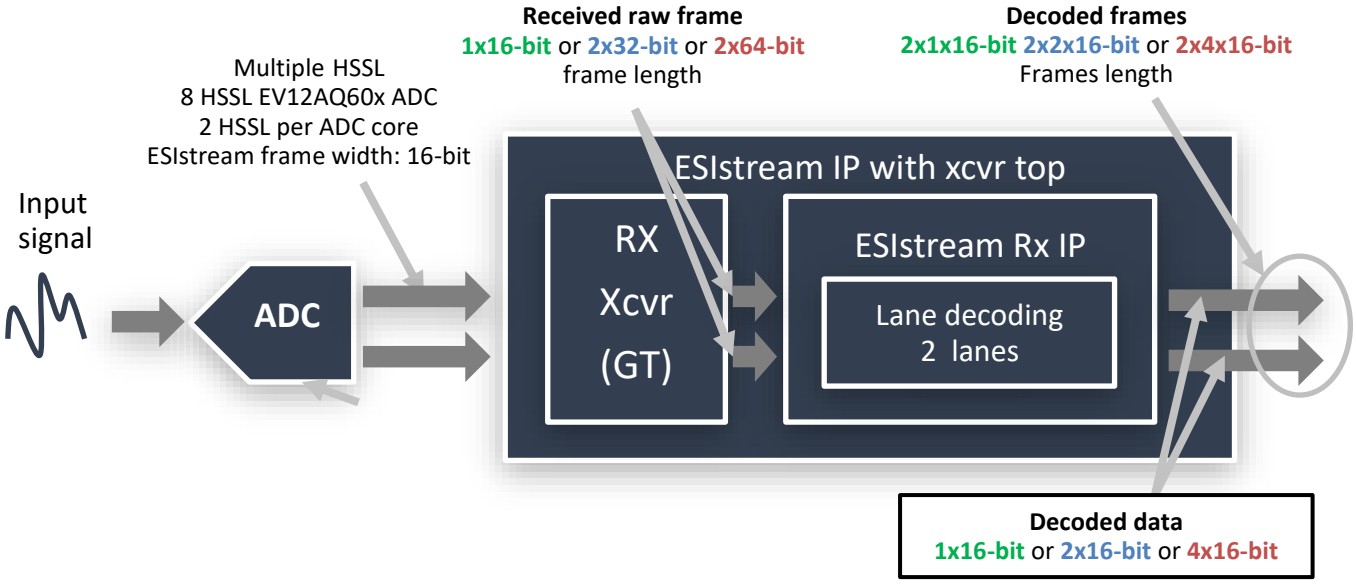


Figure 4: ESIstream RX IP data framing 16b or 32b or 64b, example with 2 HSSL.

The selection of the frame path width is a trade-off between minimum link latency, minimum logic resources, frames frequency in the FPGA allowing relaxing design timing constraints and maximum HSSL rate [Gbps].

The table below shows the logic resources utilization of the ESIstream RX IP per projects (16b or 32b or 64b) implemented on a Kintex Ultrascale xcku060-ffva1517-1-c.

Frames	Device	# HSSL	Max HSSL rate [Gbps]	FPGA frames frequency at Max HSSL rate [MHz]	LUTs KU060 (331680)	FFs KU060 (663360)	KU060 LUTs%	KU060 FFs%
16b	xcku060-ffva1517-1-c	8	6.25	390.625	1803	1854	0.54	0.28
32b	xcku060-ffva1517-1-c	8	12.5	390.625	3551	2514	1.07	0.38
64b	xcku060-ffva1517-1-c	8	12.5	195.3125	5291	4222	1.60	0.64

The Figure below shows the latency introduced by the ESIstream lane decoding module per projects (16b or 32b or 64b) implemented on a Kintex Ultrascale xcku060-ffva1517-1-c.

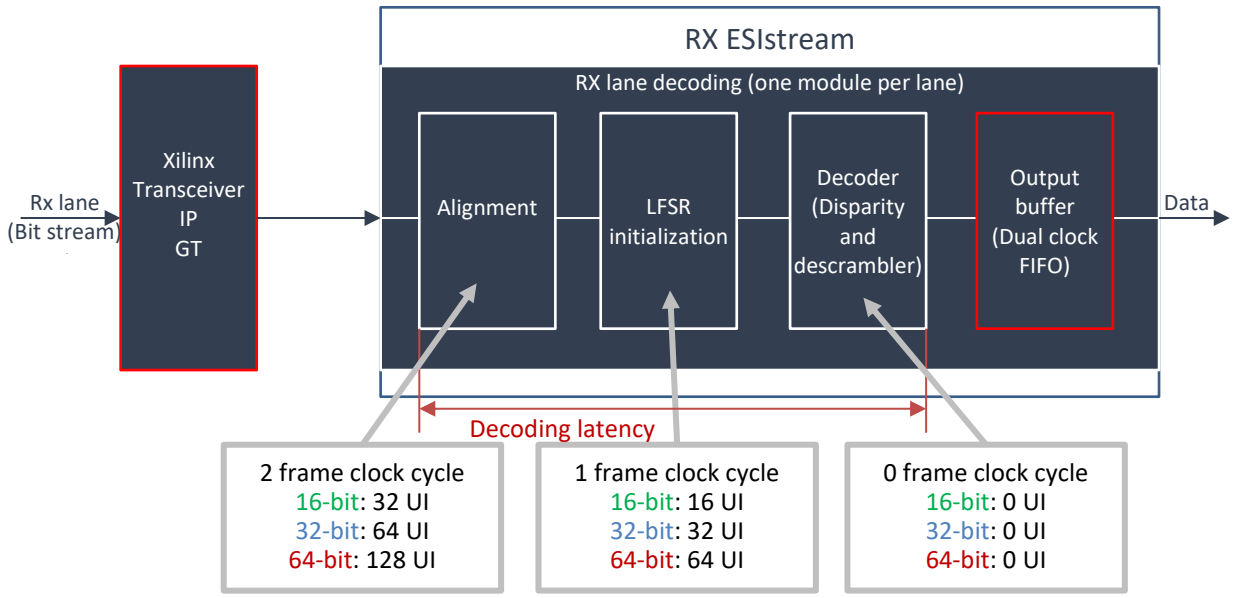


Figure 5: frames path, lane-decoding latencies per sub-modules.

The output buffer FIFO can use either the built-in FIFO, the distributed RAM or the shift-register implementation. Each implementation introduce a different latency.

1.3 Vivado_rx_ev12aq60x project

This project offers a dedicated vhdl design example to interface the EV12AQ60x FMC board with a FPGA.

A simple UART interface (8-bit, 115200) allows accessing FPGA registers through read and write operations.
A simple frames layer protocol has been defined to communicate with the register map through the UART communication.
Python scripts allow controlling the FPGA design through using the UART frames layer protocol.

The register map contain all registers to control the design (ESIstream, SPI Masters, SYNC generator…).

A module allows checking the decoded frames in order to find data bit error or clock bit erro. Error status, received frames, and some ESIstream controls signals are directly mapped to the Integrated Logic Analyzers (ILA) to be analyzed in Vivado hardware manager using JTAG port.

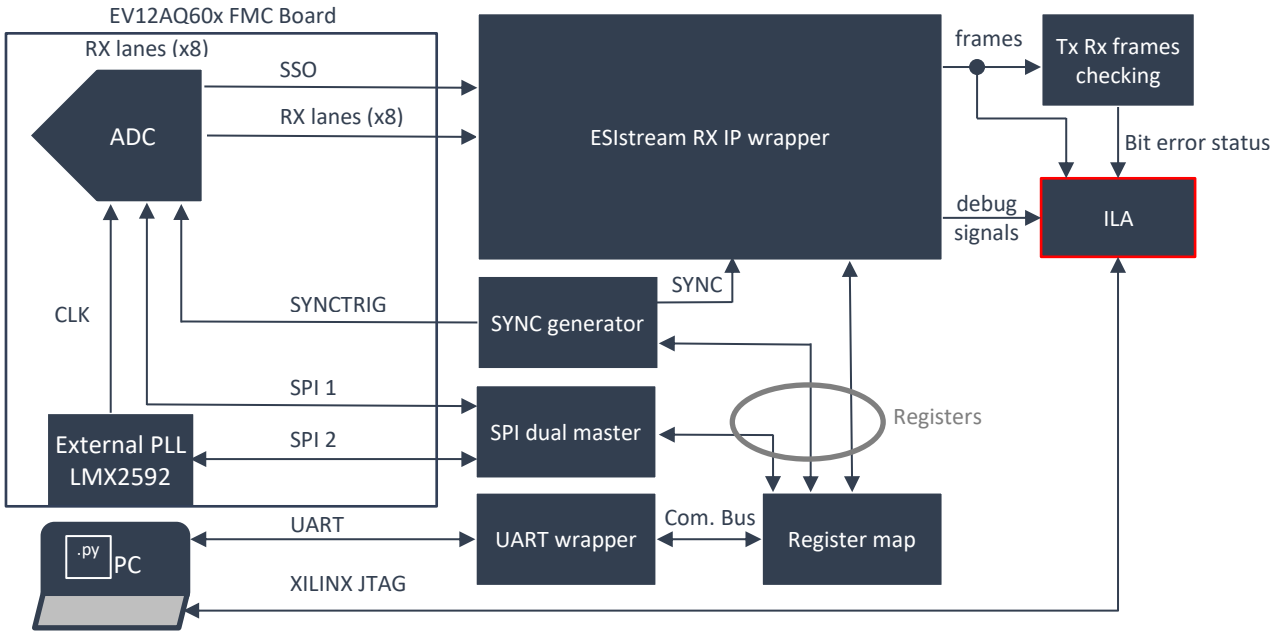


Figure 6: ESIstream RX IP project block diagram, principle.

1.3.1 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.

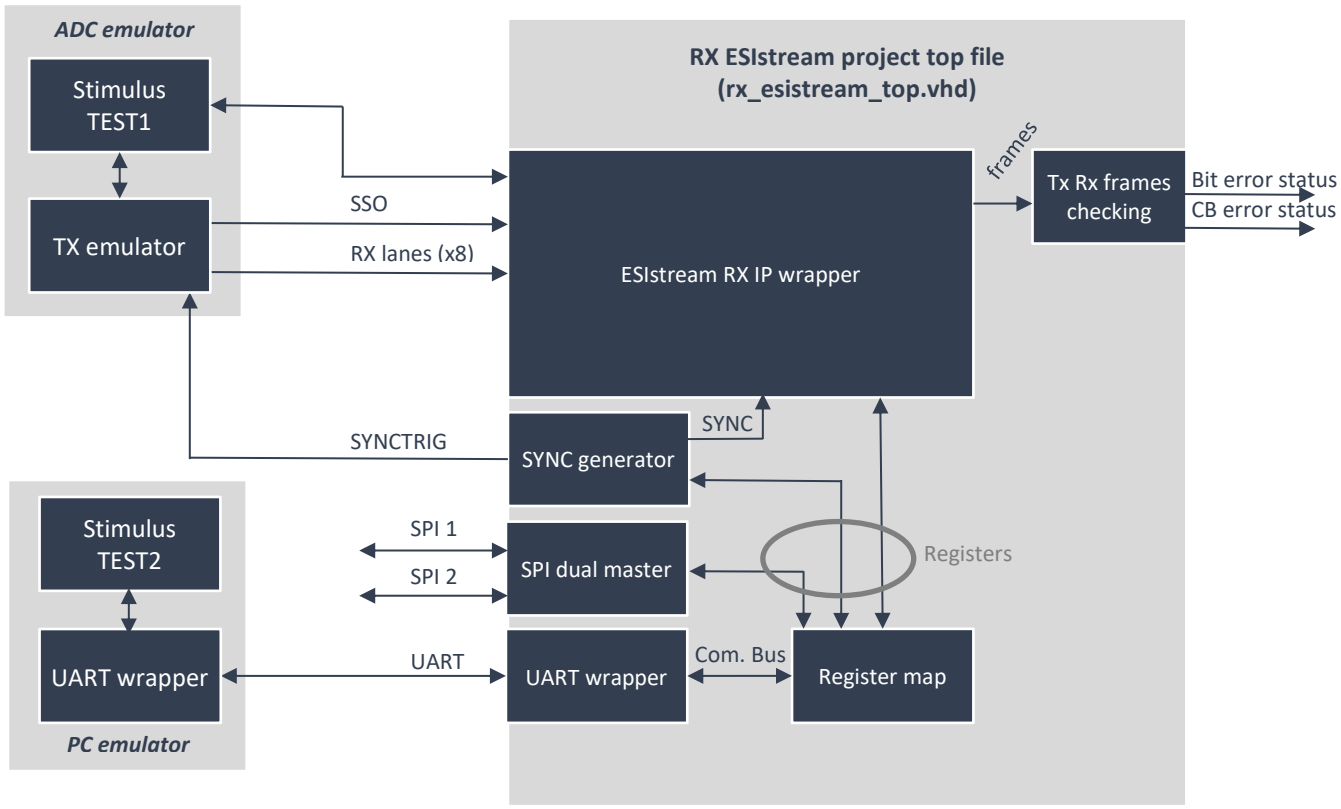


Figure 7: Test-bench Block diagram, principle

1.3.2 TEST 1

1.3.2.1 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 (txrx_frames_checking) 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

1.3.2.2 Flow chart

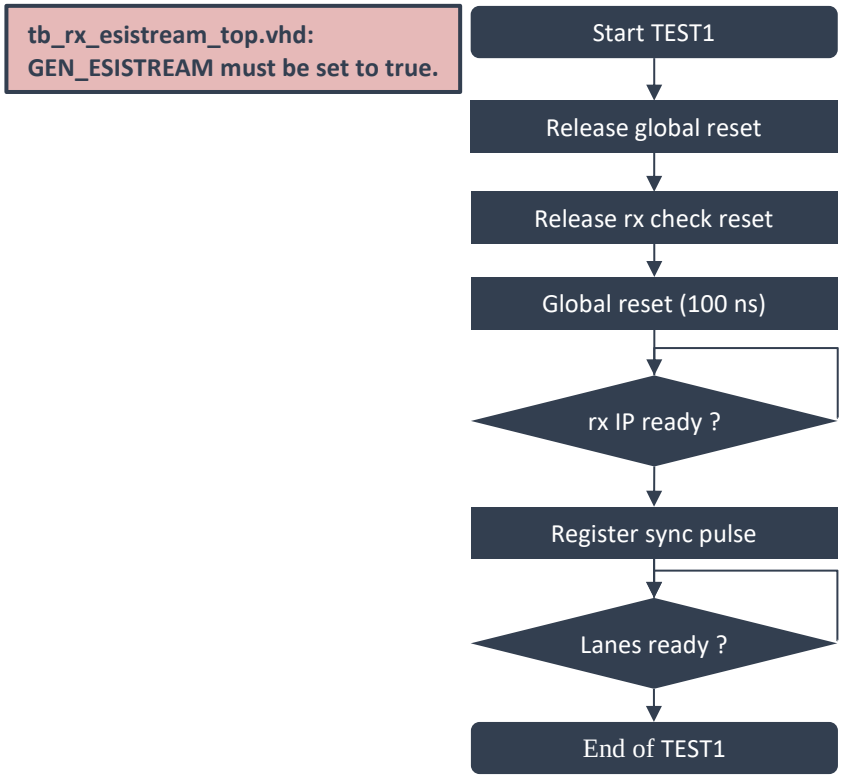


Figure 8: Test-bench flow chart, TEST 1

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

1.4 Vivado_txrx_xm107 project

This project offers a vhdl design example to test the ESIstream serial link using both ESIstream Tx and Rx IPs and a loopback board XM107 connecting TX outputs on RX inputs.

A simple UART interface (8-bit, 115200) allows accessing FPGA registers through read and write operations.
A simple frames layer protocol has been defined to communicate with the register map through the UART communication.
Python scripts allow controlling the FPGA design through using the UART frames layer protocol.

The register map contain all registers to control the design (ESIstream, SYNC generator, Tx Rx frames checking …).

A module allows checking the decoded frames in order to find data bit error or clock bit erro. Error status, received frames, and some ESIstream controls signals are directly mapped to the Integrated Logic Analyzers (ILA) to be analyzed in Vivado hardware manager using JTAG port.

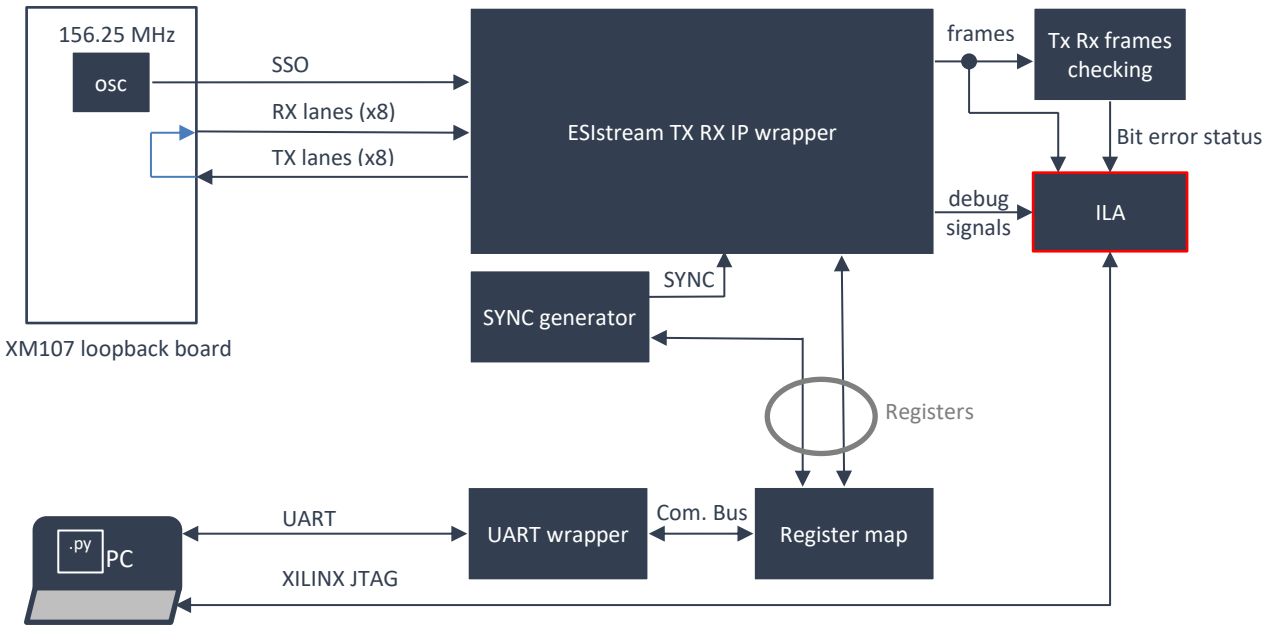


Figure 9: ESIstream RX IP project block diagram, principle.

1.5 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.

Project:

- vivado_rx_ev12aq60x (1)
- vivado_txrx_xm107 (2)
- both: (12)

Signal name	Register address	Register Bits	Type	Size	Project
tx_emu_d_ctrl(0)	0	0	W	Tx emulator data control:	12
tx_emu_d_ctrl(1)	0	1	W	Select the data sent by the Tx emulator. Tx_emu_d_ctrl: 00 – Constant: x"000" 01 – 12-bit ramp pattern, see EV12AQ60x datasheet 10 – 12-bit ramp pattern, see EV12AQ60x datasheet 11 – Constant: x"FFF"	12
rx_prbs_en	1	0	W	Enable ESIstream RX IP PRBS decoding (descrambling) when '1'.	12
tx_prbs_en	1	1	W	Enable ESIstream TX IP PRBS encoding (scrambling) when '1'.	2
tx_disp_en	1	2	W	Enable ESIstream TX IP disparity processing when '1'	2
reg_rst	2	0	W	Active high ('1') global software reset.	12
reg_rst_check	2	1	W	Active high ('1') Tx Rx frames checking module reset.	12
reg_aq600_rstn	2	2	W	Active low ('0') EV12AQ60x ADC reset.	1
rx_sync_rst	2	3	W	Active high ('1') SYNC generator module reset.	12
spi_ss	3	0	W	SPI slave select: - EV12AQ60x ADC when '0' - External PLL LMX2592 when '1'	1
spi_start	3	1	W	When '1' all SPI commands that are pre-loaded in the SPI Master input FIFO (FIFO IN), are sent to the selected SPI slave.	1
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 input FIFO. - spi_start bit must be enabled to send all commands.	1
sync_mode	5	0	W	SYNC Counter mode: - normal: 0 - training: 1	12
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.	12
manual mode	6	1	W		12
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.	12
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.	12
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.	12

<table>
<tr><td>Firmware version</td><td>8</td><td>31 - 0</td><td>R</td><td>FPGA firmware version (x"00000300" for V3)</td><td>12</td></tr>
<tr><td>fifo_in_full</td><td>9</td><td>0</td><td>R</td><td>SPI Master input FIFO full flag. Full when ‘1’</td><td>12</td></tr>
<tr><td>fifo_out_empty</td><td>9</td><td>1</td><td>R</td><td>SPI Master output FIFO empty flag. Empty when ‘0’</td><td>12</td></tr>
<tr><td>fifo_out_dout</td><td>10</td><td>23 - 0</td><td>R</td><td>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.</td><td>12</td></tr>
<tr><td>sync_rd_counter</td><td>11</td><td>7 - 0</td><td>R</td><td>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.</td><td>12</td></tr>
<tr><td>sync_counter_busy</td><td>11</td><td>8</td><td>R</td><td>SYNC generator module counter is busy. Busy when ‘1’. Can be monitored to wait for the valid counter value.</td><td>12</td></tr>
<tr><td>sync_odelay_o</td><td>11</td><td>24 - 16</td><td>W</td><td></td><td>12</td></tr>
<tr><td>sync_get_odelay</td><td>12</td><td>15</td><td>R</td><td></td><td>12</td></tr>
<tr><td>sync_odelay_i</td><td>12</td><td>8 - 0</td><td>R</td><td></td><td>12</td></tr>
<tr><td>ref_sel_ext</td><td>14</td><td>6</td><td>W</td><td>ref clk source switch
<table>
<tr><th>ref_sel_ext</th><th>ref clk source</th><th></th></tr>
<tr><td>0</td><td>Internal</td><td>Default</td></tr>
<tr><td>1</td><td>external</td><td></td></tr>
</table></td><td>1</td></tr>
<tr><td>ref_sel</td><td>14</td><td>5</td><td>W</td><td>External ref clk switch
<table>
<tr><th>ref_sel</th><th>ref clk source</th><th></th></tr>
<tr><td>0</td><td>External ref clk
SMA EXT REF</td><td>Default</td></tr>
<tr><td>1</td><td>fpga_ref_clk</td><td></td></tr>
</table></td><td>1</td></tr>
<tr><td>clk_sel</td><td>14</td><td>4</td><td>W</td><td>EV12AQ60x CLK source switch
<table>
<tr><th>sync_sel</th><th>synco fpga source</th><th></th></tr>
<tr><td>0</td><td>PLL LMX2592</td><td>Default</td></tr>
<tr><td>1</td><td>External SMA</td><td></td></tr>
</table></td><td>1</td></tr>
<tr><td>synco_sel</td><td>14</td><td>3</td><td>W</td><td>SYNCO Multiplexer CBTL01023 SEL-input
<table>
<tr><th>synco_sel</th><th>synco fpga source</th><th></th></tr>
<tr><td>0</td><td>SYNCO ADC</td><td>Default</td></tr>
<tr><td>1</td><td>SYNCO External SMA</td><td></td></tr>
</table></td><td>1</td></tr>
<tr><td>sync_sel</td><td>14</td><td>2</td><td>W</td><td>SYNC Multiplexer CBTL01023 SEL-input
<table>
<tr><th>sync_sel</th><th>sync adc source</th><th></th></tr>
<tr><td>0</td><td>SYNC FPGA</td><td>Default</td></tr>
<tr><td>1</td><td>SYNC External SMA</td><td></td></tr>
</table></td><td>1</td></tr>
<tr><td>hmc1031_d1</td><td>14</td><td>1</td><td>W</td><td rowspan="2">Low jitter clock generation with integer N PLL
<table>
<tr><th>D0</th><th>D1</th><th>State</th><th>Ref clock frequency</th><th>PLL Division ratio</th><th></th></tr>
<tr><td>0</td><td>0</td><td>OFF</td><td>N.A</td><td>Power-down</td><td>Default</td></tr>
<tr><td>1</td><td>0</td><td>ON</td><td>100 MHz</td><td>Divide by 1</td><td></td></tr>
<tr><td>0</td><td>1</td><td>ON</td><td>20 MHz</td><td>Divide by 5</td><td></td></tr>
<tr><td>1</td><td>1</td><td>ON</td><td>10 MHz</td><td>Divide by 10</td><td></td></tr>
</table></td><td>1</td></tr>
<tr><td>hmc1031_d0</td><td>14</td><td>0</td><td>W</td><td>1</td></tr>
<tr><td>-</td><td>255</td><td>31 - 0</td><td>R</td><td>Reserved</td><td>-</td></tr>
</table>

1.6 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.

1.6.1.1 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).

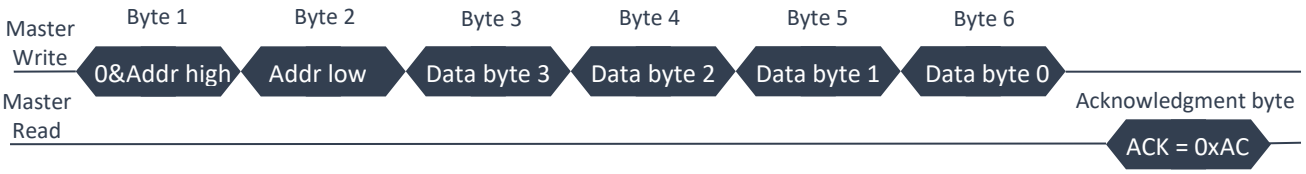


Figure 10: UART frames layer protocol, write operation

1.6.1.2 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.

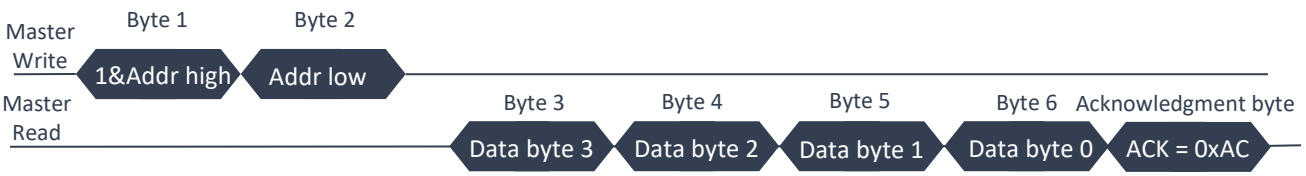


Figure 11: UART frames layer protocol, read operation

1.7 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 us https://www.esistream.com/contact.

1.7.1 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.

1.7.2 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.

1.7.3 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. ESISTREAM RX IP (RECEIVER)

2.1 Principle

After a SYNC, the RX IP waits for the ESIstream Synchronization Sequence (ESS) composed of the Frame Alignement Sequence (FAS) and of the PRBS Alignment Sequence (PAS). The transmitter sends the ESS to allow the receiver to recover the frames in the serial data stream and to initialize its LFSR. The LFSR generates the PRBS sequence required to descramble the frame. The receiver decodes the frames according to the ESIstream protocol specification (descrambler, disparity bit) to get the data.

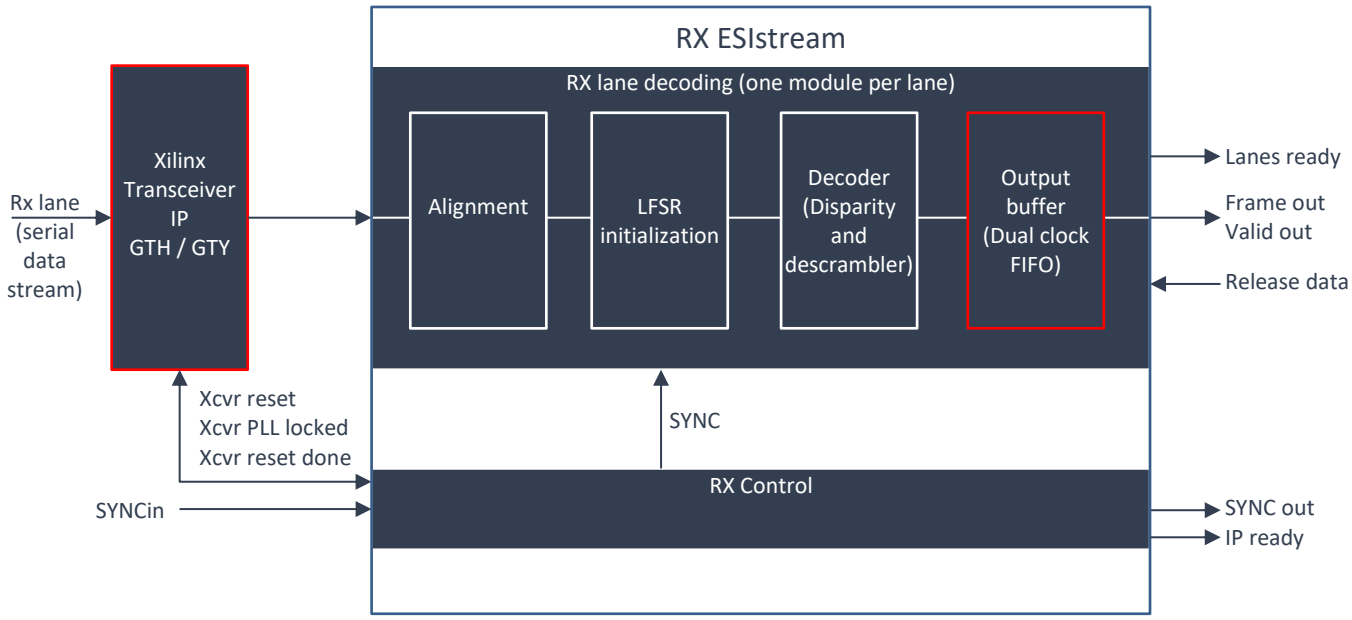


Figure 12: ESIstream RX IP block diagram, principle.

The output buffer enables data alignment on multiple lanes. For each lane, the frames are buffered, in the output buffer, as soon as the fisrt valid frame is decoded. The first valid frame is available just after the ESS. When all lanes contain a valid frame (i.e. when lanes ready signal is high), the frames can be released from the output buffers at the same time (i.e. by setting the release data signal high).
The output buffer also allows transferring the frames between the transceiver clock domain (rx_usrclk) to the acquisition 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 user clock provided by the transceiver IP (clk_acq <= rx_usrclk).
For deterministic latency applications, the acquisition clock domain (clk_acq) must be connected to a deterministic clock synchronous with the transmitter clock. For instance, from a FPGA Global Clock (GC) 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 a 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

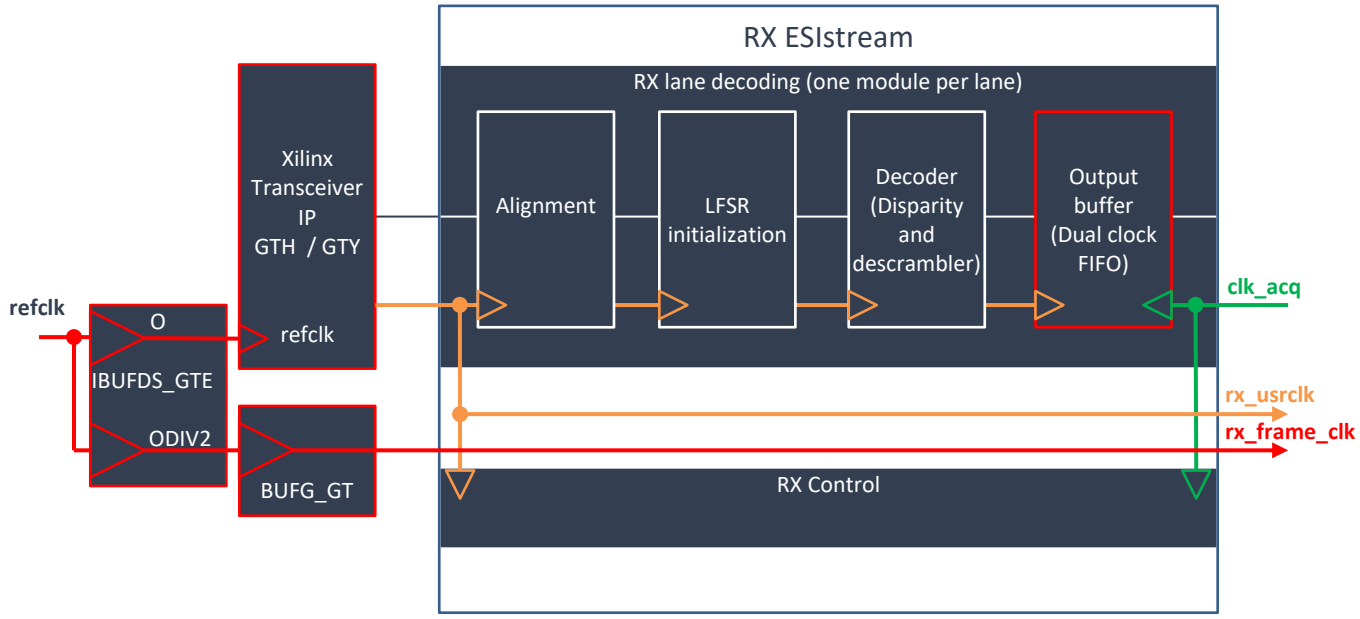


Figure 13: ESIstream RX IP block diagram, rx_frame_clk from IBUFDS_GT ODIV2 output no MMCM.

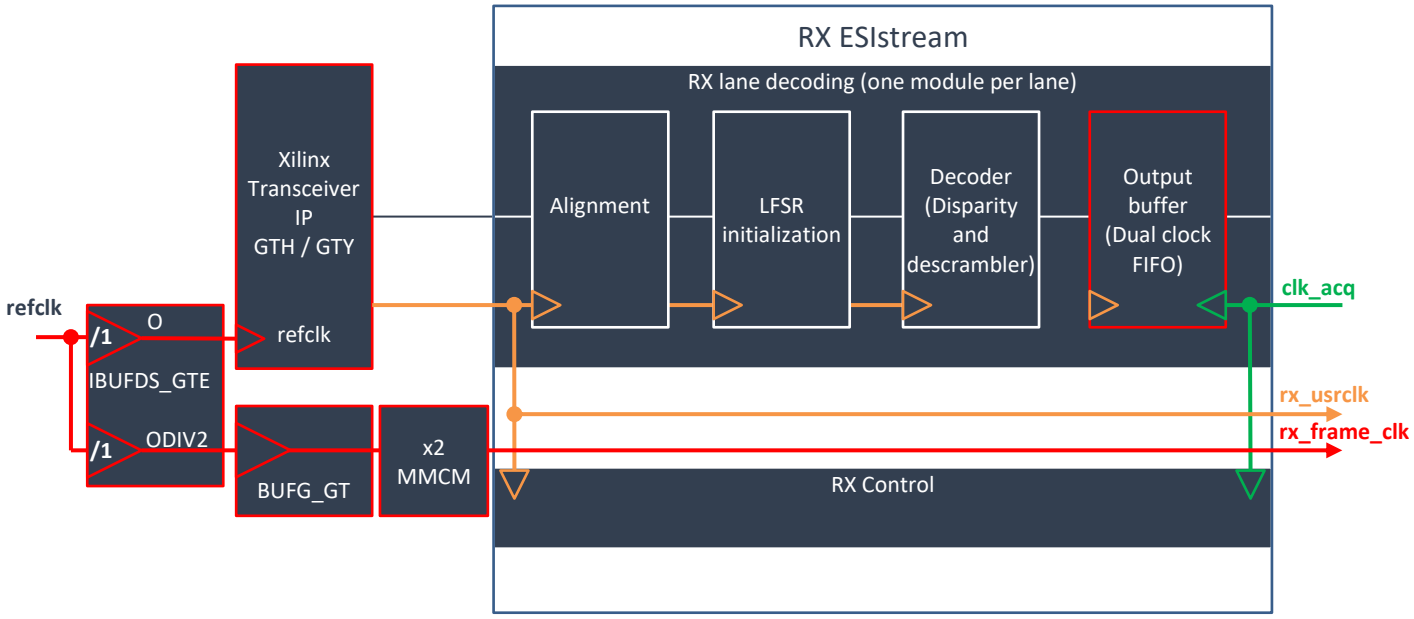


Figure 14: ESIstream RX IP block diagram, rx_frame_clk from IBUFDS_GT ODIV2 output with MMCM.

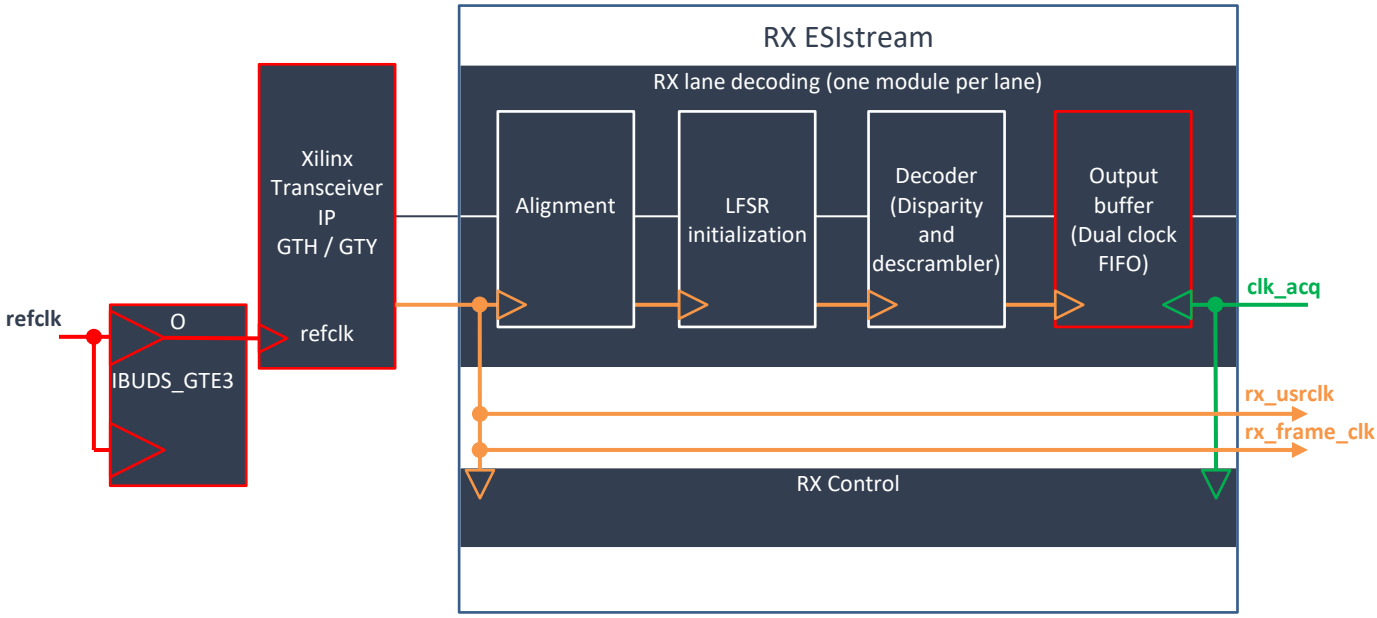


Figure 15: ESIstream RX IP block diagram, rx_frame_clk from rx_usrclk

ODIV2 is configured to buffer the refclk input (refclk divided by one), see "Figure 2-1: Reference Clock Input Structure" of the Xilinx documentation UG576 UltraScale Architecture GTH Transceivers.

2.1.1 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.

2.1.1.1 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.

2.1.1.2 Entity port signals

rx_esistream.vhd module entity port description.

Port	Type	Width	Clock domain	Description
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.

2.1.1.3 Frame Output signal description

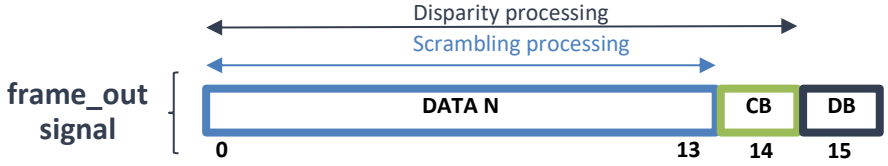


Figure 16: 16-bit output data organization

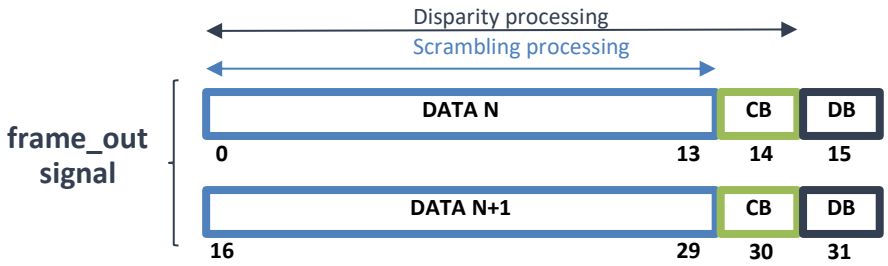


Figure 17: 32-bit output data organization

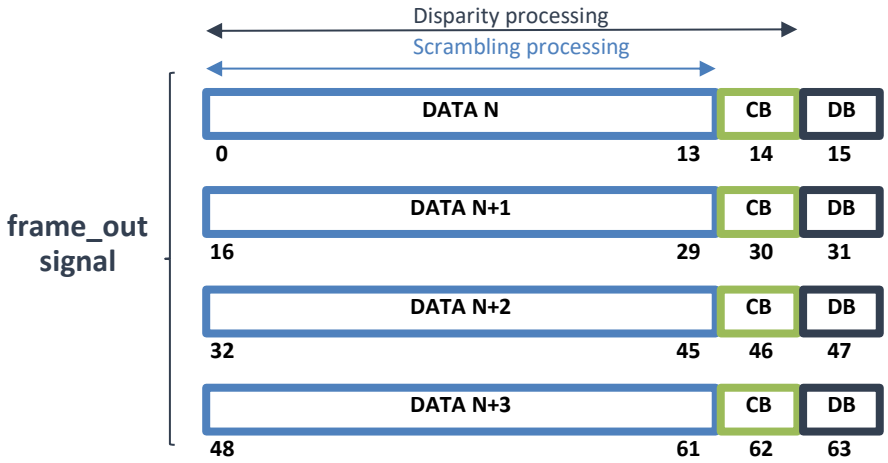


Figure 18: 64-bit output data organization

2.1.2 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.

2.1.3 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.

2.1.4 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).

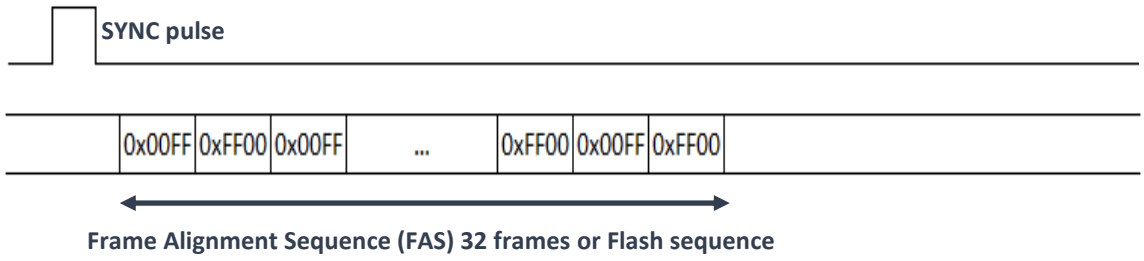


Figure 19: Link synchronization, Frame Alignment Sequence (FAS) or Flash sequence

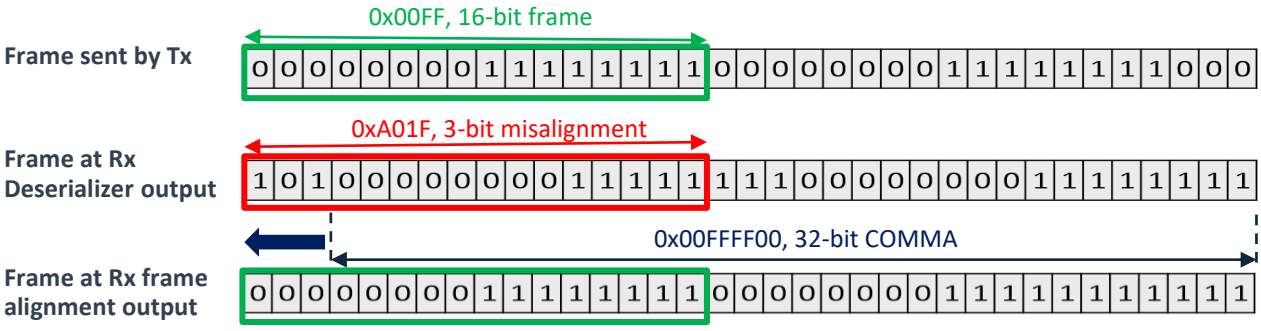


Figure 20: RX Frame alignment module, principle

2.1.5 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.

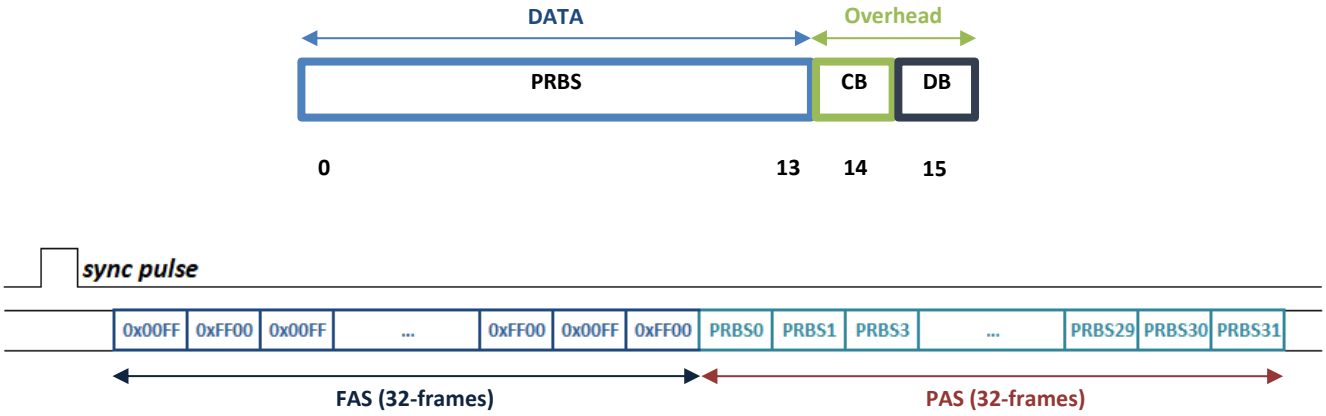


Figure 21: Link synchronization, PRBS Alignment Sequence

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

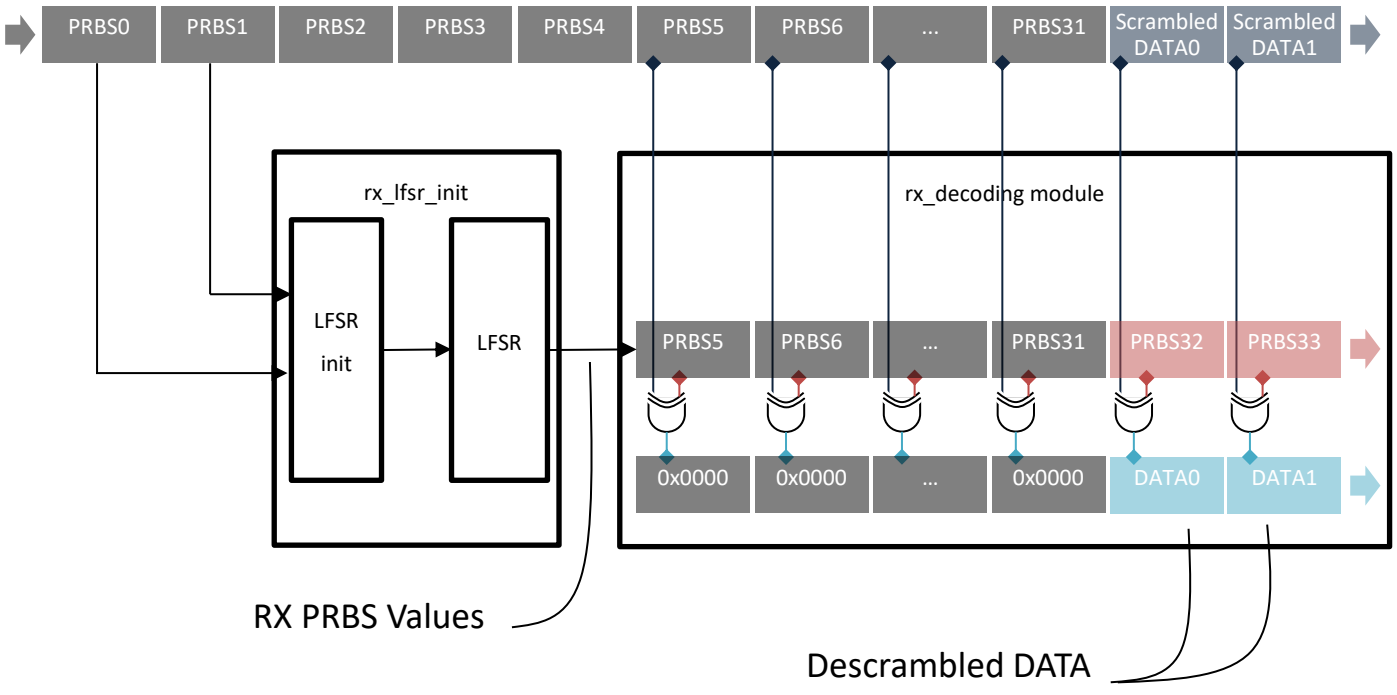


Figure 22: Descrambling principle

2.1.6 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.

2.1.7 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.

2.1.8 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.

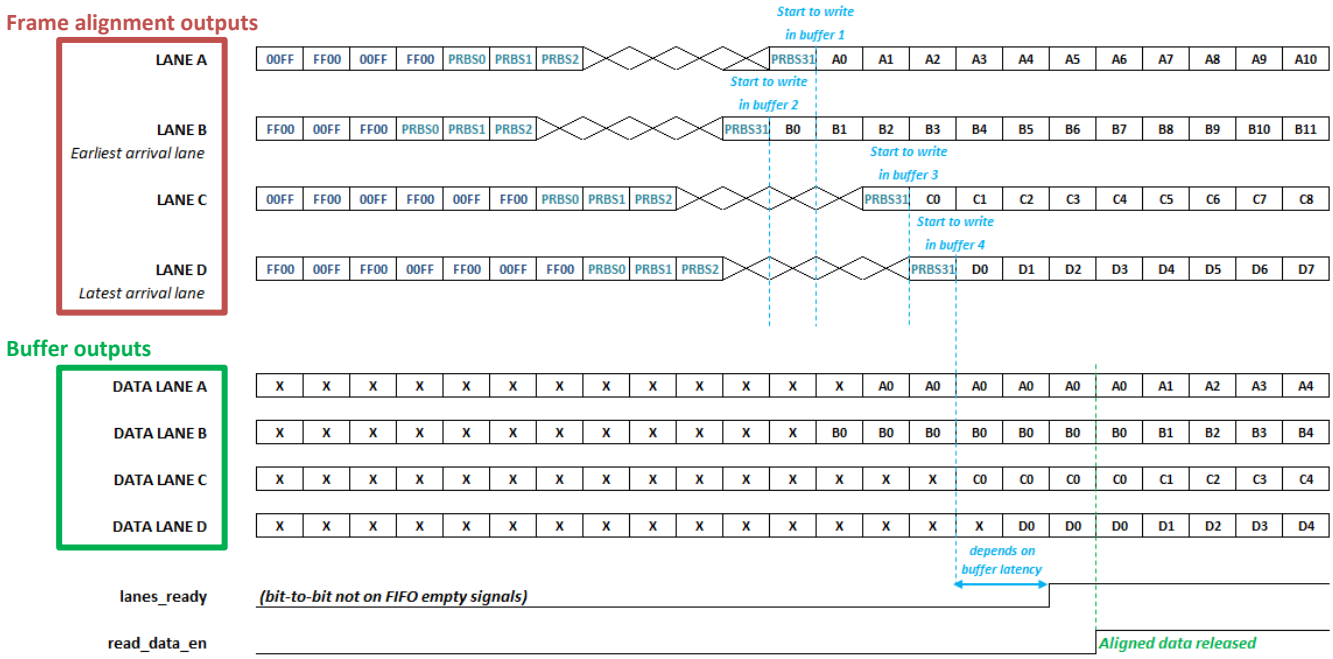


Figure 23: Multiple lane Synchronization, principle

2.1.9 Output buffer IP

File name	Description
output_buffer.xci	The write clock is in the GT rx_usrclk frame clock domain, The read clock is in the the “acquisition” frame clock domain, clk_acq. FIFO can be optimized using a Block RAM FIFO or a Distributed RAM FIFO depending on trade-offs on programmable logic resources utilization. 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 get the team involved using ESIstream contact web page or at GRE-HOTLINE-BDC@Teledyne.com.

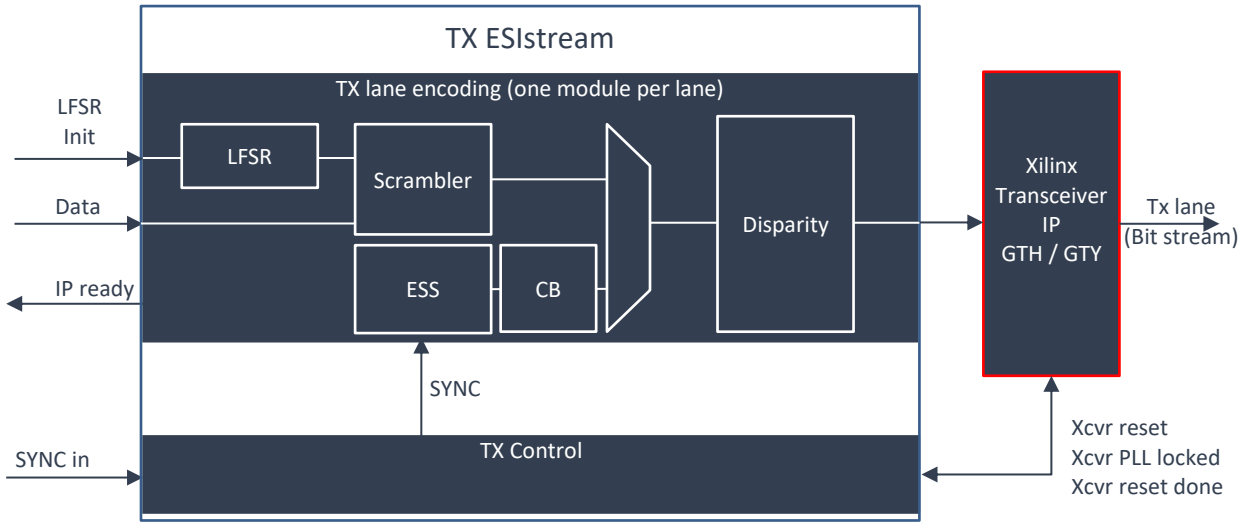
2.1.10 Delay in RX output buffer wrapper

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

3. ESISTREAM TX IP (TRANSMITTER)

3.1 Principle

After a SYNC, the transmitter sends the ESIstream Synchronization Sequence (ESS) to allow the receiver to recover the frames in the serial data stream and to initialize its LFSR. The LFSR generates the PRBS sequence required to descramble the frame. The receiver decodes the frames according to the ESIstream protocol specification (descrambler, disparity bit) to get the data.



In case of a multi-lane interface, in order to reduce correlation between lanes, each lane should have different initial values (LFSR Init) for the scrambling units.