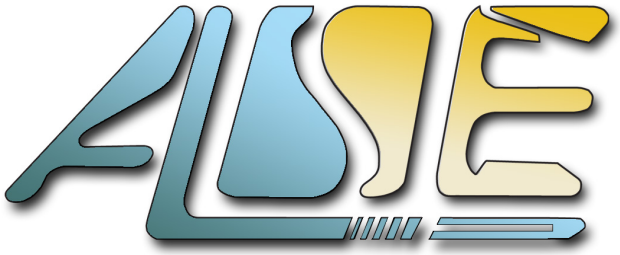
Aurora 8b/10b on Efinix

Example Design User Guide



A.L.S.E
8, Passage Barrault
75013 PARIS
France
Tel : +33 1 84 16 32 32

FPGA.fr

Advanced Logic Synthesis for Electronics

1. REVISION HISTORY

The following table shows the revision history for this document and the associated IP.

Version	Date	Revision
1.0	Mar 2025	Initial Version
1.1b	Apr 2025	Add STLV7325 support for Xilinx side

Table 1: Revision history

2. GENERAL INFORMATION

This document describes the ALSE Aurora 8b/10b Reference Design for Efinix Titanium.
This design has been tested on the Titanium Ti375 N1156 Development Kit board.

It comes in two parts :

- First : a pre-compiled bitstream for the Efinix board featuring a 1 lane ALSE Aurora 8b/10b IP attached to a loop-back design inside the FPGA logic area. Every frame received on PDU/UFC/NFC ports is simply sent-back to the sender. The Aurora IP is connected to the **J11 SFP+ connector**.
- Second : a full source code Master design targeting running on a Xilinx FPGA (built using Vivado v2023.2), featuring a Xilinx Logicore Aurora 8B/10B IP configured to match the Efinix reference design.
  This project has been implemented on **two Xilinx boards :**
  - AMD Virtex-7 Evaluation Kit (VC707) or
  - STLV7325 Kintex 7

  This design, based on the *Xilinx* Aurora 8B/10B IP, is sophisticated : it can exercise the Aurora link in a lot of ways and perform exhaustive testing of the *ALSE* Aurora 8b/10b IP running on the Efinix board.

Connect the Titanium Dev kit to the Xilinx VC707 or STLV7325 board to demonstrate the ALSE Aurora 8b/10b IP solution on Efinix.

2.1 IP Configuration

In the **current** version, the Aurora 8B/10B IP is configured as follows :

- ✓ Full-Duplex
- ✓ 1 x transceiver lane at 3,125 Gbit/s through SFP+ connector.
- ✓ 16 bits (2 bytes width) user Datapath.
- ✓ Framing Interface with no CRC.
- ✓ User Flow Control (UFC) in direct mode (Xilinx-IP style).
- ✓ Native Flow Control (NFC) in completion mode.
- ✓ Clock Compensation sequence generation enabled.
- ✓ 156.25 MHz reference clock for transceivers.

These settings could be modified to match specific customer requests. Contact ALSE.

2.2 Delivery Contents

The example design delivery includes :

- **Bitstream and flash image** for Efinix **Ti375 N1156 Development Kit**.
  This programs the FPGA with the Aurora 8b10b IP and the loopback reference design.
- A full Master Reference Design for **Virtex 7 Evaluation Kit (VC707)**, built using Vivado 2023.2.
  This project is provided as full source code !
- A bitstream for the same Master Reference Design, but ported to the **Kintex7 STLV7325 board.**

xil_ref_design/	Xilinx Reference Design Folder
fit/VC707_SFP_x1_16b_master	Vivado Project Files, compilation scripts and pre-compiled binary image.
src/	
boards/	Xilinx specific source files
common/	Common source files
test_modules/	Test source files
efinix_ref_design/	
Tx_FIFO_Rx_fifo.bit/hex	Efinix Reference Design Binaries
debug_profile.json	Efinix debug profile

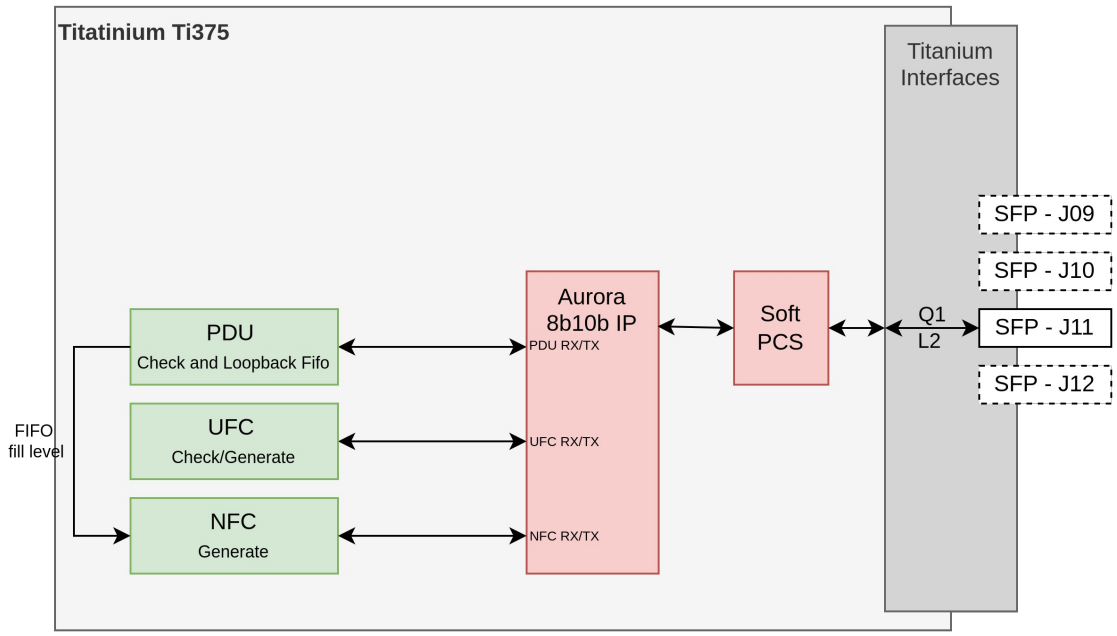
3. EFINIX LOOPBACK REFERENCE DESIGN

As explained in the introduction, the Efinix bitstream targets the **Efinix Ti365 N1156 Development Kit** and contains an ALSE Aurora 8b/10b IP that implements a data loopback : the data received on each port (PDU or UFC) is simply sent back to the sender on the same interface, unmodified, through a FIFO. The Native Flow Control (NFC) block implements throttling commands to the sender to avoid overflowing the loopback FIFO.

The Aurora IP is connected to **SFP J11** (**Quad 1, Lane 2**) of the board.

The bitstreams are named *Tx_FIFO_Rx_fifo.bit* and *Tx_FIFO_Rx_fifo.hex*.

Here is a simplified schematic view of the loopback design :



3.1 Protocol Data Unit (PDU)

The PDU Interface is the main data interface. The received data stream (see Xilinx section for frame format) is driven into a checker module that pushes valid data into a dual clock FIFO. The DCFIFO performs clock domain crossing between RX and TX clock domains. The FIFO output is then looped back to the TX streaming interface.

The FIFO level is monitored to avoid overflow. When reaching predetermined fill levels, instructions are sent to the sender through the NFC interface to delay the sending of new frames.

3.2 User Flow Control (UFC)

The UFC interface is a data interface with higher priority than PDU. As for PDU, UFC frames go through a checker and are sent back to the Aurora.

The UFC interface has a higher priority than the PDU interface in the Aurora protocol, meaning it is always first even if some PDU requests are waiting in the FIFO for a long-time. Therefore the UFC FIFO does not fill up, contrarily to the PDU Fifo, which may potentially fill up.

3.3 Native Flow Control (NFC)

Native flow control (NFC) is used to regulate the traffic sent by the Aurora partner. We use it in the design to avoid PDU FIFO overflow. When receiving UFC and PDU FIFO at a full rate, because of the small delay due to checking logic, the PDU FIFO tends to fill up. The design generates NFC request to pause the sending of the Aurora partner to keep the FIFO filling level under a certain limit.
However the system is not built to take into account RX NFC request : if we receive NFC requests as well as PDU and NFC FIFO at the same time, the pause request mechanism will not be sufficient to compensate and the PDU FIFO will overflow : some frames will be lost in the process. This situation should reflect on the RX counters and RX error counters on the partner side.

4. XILINX REFERENCE DESIGN

The Xilinx Reference design targets the Virtex-7 Evaluation Kit. All the sources to compile the delivery are provided as well as a pre-compiled bitstream (xil_top_vc707.bit/xil_top_stlv7325.bit) with its associated debug file (xil_top_vc707.ltx/xil_top_stlv7325.ltx).
The project, scripts and IPs are built using Vivado v2023.2.

4.1 Compilation

Note that this step is *optional* since the delivery includes a compiled and ready to use bitstream.

A compilation script is available for Linux. Simply open a shell in the directory :
xil_ref_design/fit/xilinx/<board_name>_SFP_x1_16b_master/
then launch the *do_compile.sh* script.
It creates a Vivado project using Tcl scripts, compiles the full project and generates a bitstream.

If you are not under Linux, call the *<board_name>_SFP_x1_16b_master.tcl* script directly from the Vivado GUI.

4.2 Reference Design Contents

The Reference Design contains a Generator and a Checker module for each interface (PDU, UFC, and NFC).

Each Generator is enabled and disabled through the a Xilinx **VIO** module instantiated in the design.

In the following, in *italic* you will find the corresponding signal module names in the VIO debug module.

- **PDU Frame Generation (pdu_gen.vhd)**
  The PDU Frame Generator is enabled/disabled through it's **enable** signal : ***s_pdu_tx_ena***.
  By default, it generates frames of variable length filled with pseudo-random data created by a Linear Feedback Shift Register (LFSR).
  The generator also allows the user to generate custom PDU frames by asserting the **fixed_size_on** (***s_pdu_fixed_size_on***) signal (to value 1). In this mode, the user specifies the following 32 bits positive values to set the data-stream as desired :
  - Consecutive frames will have incremental size, between **fixed_size_min** (***s_pdu_fixed_size_min***) and **fixed_size_max** (***s_pdu_fixed_size_max***) words.
  - The frame size increment is **fixed_size_incr** (***s_pdu_fixed_size_incr***).
  - Each frame are split by a configurable gap **fixed_size_ifgap** (***s_pdu_fixed_ifgap***) (in clock cycles).
  - A frame can be split with Idle cycles by setting the **fixed_cut_size** (***s_pdu_fixed_cut_size***) value to set the gap position inside a frame (in word) and **fixed_cut_gap** (***s_pdu_fixed_cut_gap***) to set its width (in clock cycles).

  The module contains two more parameters.
  - First the **underflow** (***s_pdu_underflow***) that inserts dead cycles at random clock cycles, thus decreasing a bit the PDU bandwidth.
  - Second the **endless_mode** (***s_pdu_endless_mode***). As long as it stays high, no more EOP are generated.

  The module updates a frame counter (***s_pdu_tx_frame_cnt***) that counts the number of frames sent.
  When enabled and not in endless mode, you should see it increasing.
  When in fixed mode, the current output frame size can be read on **frame_size** (***s_pdu_tx_frame_size***).
- **UFC Generation (ufc_gen.vhd)**
  UFC frames generation process has the same mechanism as the PDU module but without the fixed mode.
  It only uses an **enable** signal (***s_ufc_tx_ena***) for activation and deactivation.

- **NFC Generation (nfc_gen.vhd)**
  NFC generation is **disabled by default** on the master side and not accessible in the VIO module.
  When NFC generation is enabled, it slows down the loop-back side causing frame losses when enabled in parallel of the PDU side.
  It is enabled on loop-back side to serve as a back-pressure mechanism avoiding fifo overflows.

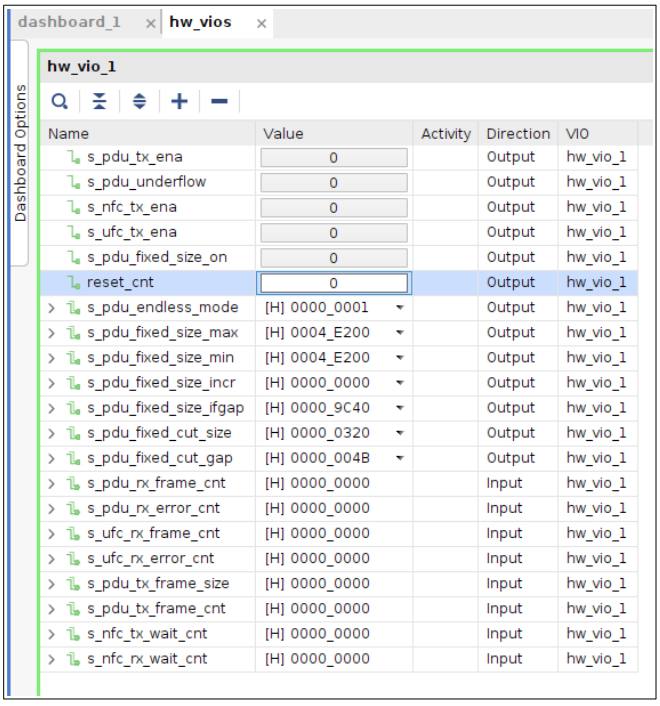
Each interface contains also a Checker, that counts the number of received frames as well as errors.
Here is the description of the available counters on RX side :

- ***s_pdu_rx_frame_cnt***; counts received PDU frames
- ***s_pdu_rx_error_cnt***; number of invalid words in PDU frames after a bit check.
- ***s_ufc_rx_frame_cnt;*** number of ufc frame received.
- ***s_ufc_rx_error_cnt***; number of invalid words received on UFC interface
- ***s_nfc_rx_wait_cnt***; Sum of received NFC, increasing when PDU fifo from loop-back side requires back pressure.

4.3 VIO Module

Here is the a capture of the hardware manager connected to the VIO module of the design.

It contains all the signals listed above, as well as ***reset_cnt*** that allow the user to reset the data counters to 0.

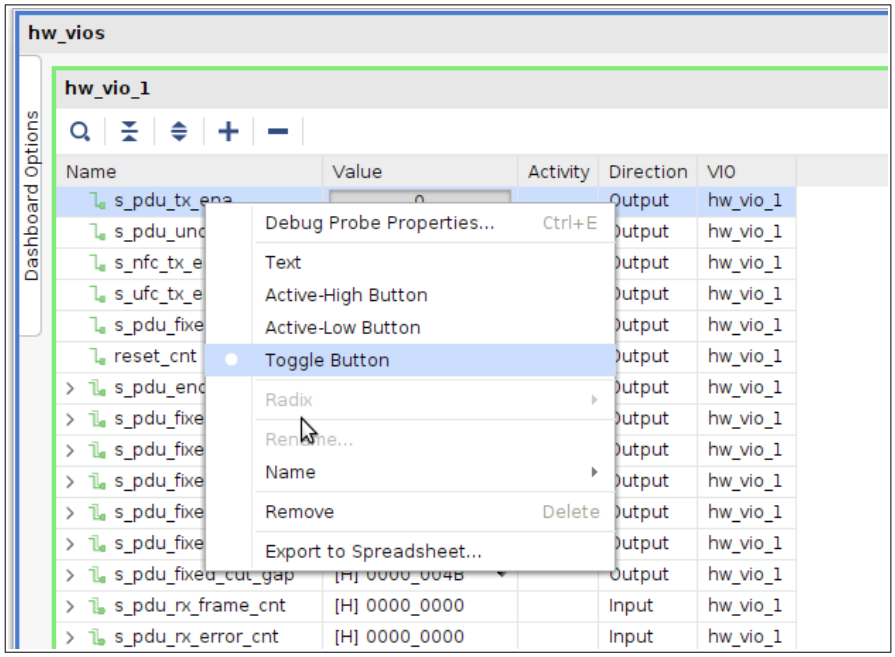


By default, the probes do not have the names displayed on the capture above. They are all named ***probe_nn*** or ***source_nn***, and we will change this.

Once the hardware manager is opened, the device programmed, and the debug core is found by Vivado : **source (run) the *rename_probes.tcl* script** to rename the probes as in the capture above.

Right-click on the enabled signals and select *Toggle Button* to change the button type to switch.

Set the *reset_cnt* to *Active-High Button*.

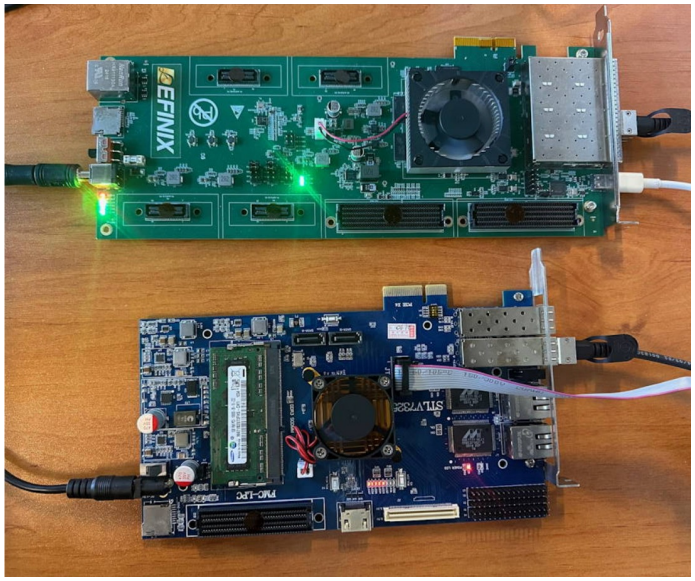


5. HARDWARE TEST

- Start by connecting both USB ports to a PC.
- Power-up both boards.
- Program the Titanium Kit with the pre-compiled bitstream. You can either program the on-board flash or the FPGA (sram). As long as the board SFP is not connected to a working Aurora, LED 2 should be blinking, LED 6 OFF and all other LEDs ON.

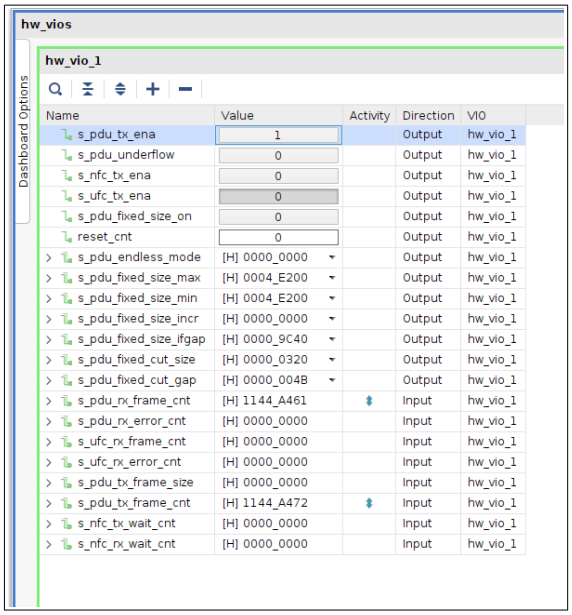
  Once connected (with a loopback SFP module for example), LED 6 turns ON.
- Program the Xilinx board (VC707 or STLV7325) with a bitstream **specifying the correct debug file** (*.ltx). Only LED 0 is ON, all other LEDs are OFF.
  Once connected (with a loopback SFP module for example) LEDs 1 and 2 turn ON.
- Connect SFP J11 of the Titanium Dev Kit to the only SFP (or SFP+ A) cage of the VC707 (STLV7325) with a DAC cable or a fiber as below :

VC707 ↔ Titanium (J11)



STLV7325 (SFP+ A) ↔ Titanium (J11)

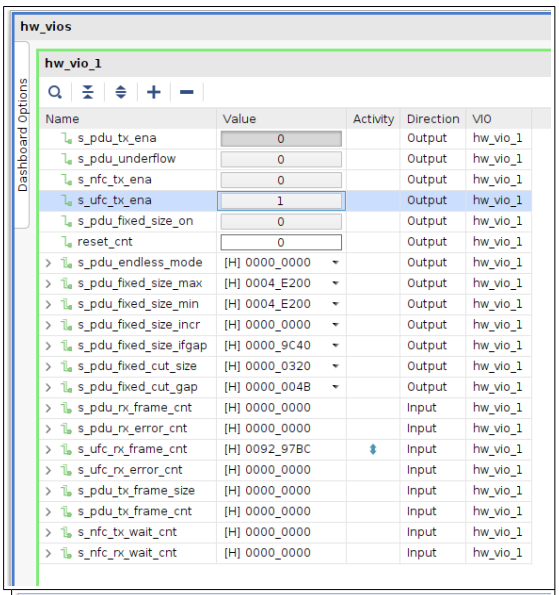
Connect the Vivado Hardware Manager to the VIO debug modules present in the Xilinx design and start sending frames using the **enable** signals.



Start for example by enabling the PDU interface. You now see the RX (received) and TX (sent) frame counters counting up quickly.

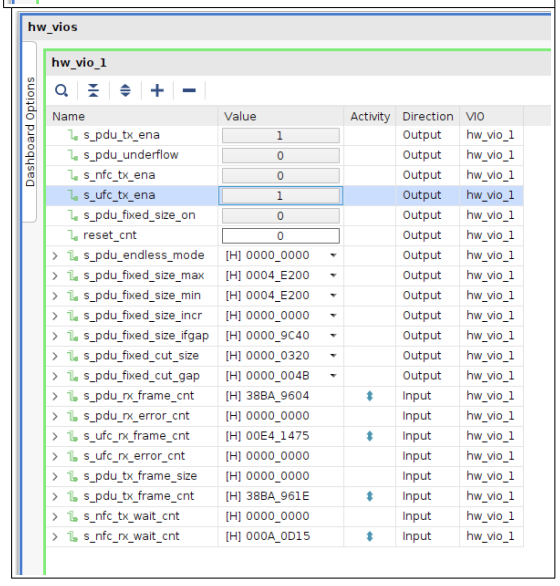
If you stop the sending by setting ***s_pdu_tx_ena*** to 0, both counters should display the exact same number, meaning that the Titanium board sent back the exact amount of frames received.

The ***s_pdu_rx_frame_err*** should stay at 0.



If now we enable the UFC interface only, we see the RX count incrementing.

Again, the RX UFC error count should stay at 0.



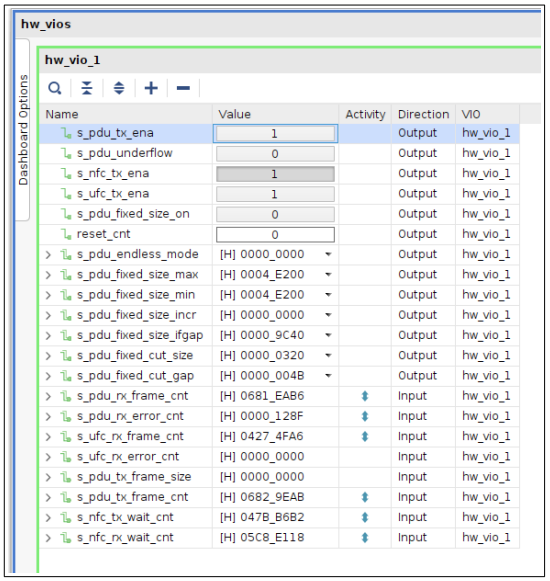
Now, if we enable both PDU and UFC interface, both RX (receive) counters counters go up, as well as the TX PDU counter.
Both error counters should stay at 0.

Another counter goes up : the RX NFC wait counter.
Indeed, on the Titanium board, when sending on both interfaces, the processing delay as well as the latency fill up the PDU FIFO. To compensate and to avoid a FIFO overflow, NFC requests are generated to reduce the sending speed of the Xilinx side.

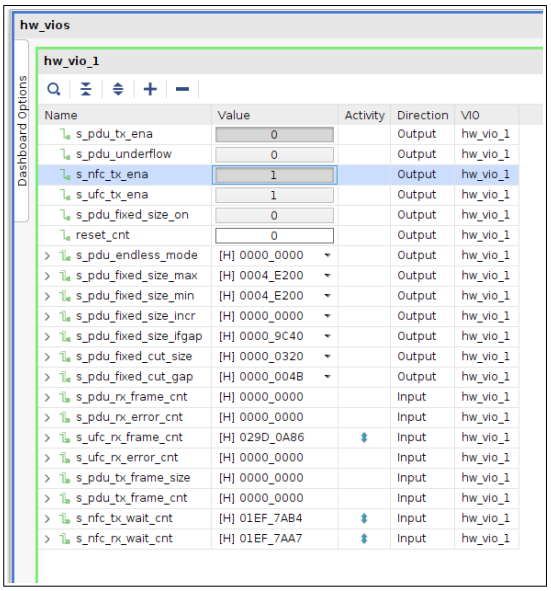
If you enable the *underflow* mode you will notice that the ***s_nfc_rx_wait_cnt*** goes up a lot slower : because we already reduce the PDU sending speed with this option.

Only if NFC generator is connected in the VIO (which is not the case by default) :

If we now enable all the interfaces, we get errors : the NFC generation mechanism on the RX side does not compensate enough and we get a PDU fifo overflow, loosing frames in the process. You should see the PDU RX error count go up.



When PDU is disabled, UFC does not generate traffic to get any overflow and we get the expected behavior : no errors with TX and RX counters increasing.



6. TECHNICAL SUPPORT

For any question about this IP, please contact ALSE Technical Support by **E-mail** at support@alse-fr.com or at a specific E-mail address that you may have received.

If a telephone contact is desired, our R&D office can be reached at +33 1 84 16 32 32, during office hours, CET.

You will be either answered directly or directed to an engineer available and competent, depending on the type of question or support.

Advanced Logic Synthesis for Electronics
A.L.S.E. The FPGA Experts www.FPGA.fr

-=oOo=-