Application Note

ALSE Aurora 8B/10B IP
taking advantage of the
Lattice Avant-G Transceivers

A.L.S.E
36b Av René Coty - 75014 Paris France
Tel: +33 1 84 16 32 32

Aurora 8B/10B on Avant-G
Application Note

Table of Contents

Aurora 8B/10B on Avant-G
Application Note

1. Revision History

Author : François BATARD - ALSE

Version	Description
v1.0	Initial Version
V 1.1	This release. Renesas Clock Generator issue solved
V1.2	Proof-reading updates

2. Archive

This document is part of an archive, which you can download*, containing :

doc/	This document
fit/lattice	Lattice Avant-G Bit-Stream and Reveal Files – generated with Radiant 2025.1.1 (latest version at the day of the writing). A valid configuration for the RC38612A clock generator is available in this folder as well.
fit/xilinx	Xilinx Reference Design Project File (for Vivado 2023.2). A bit-stream and a Virtual I/O debug files are available as well.
src/	Full Xilinx Aurora Reference Design Source Code

* : contact ALSE to get the up-to-date link.

3. Introduction

The latest **Lattice Avant™** 16nm FinFET platform is now available. Taking advantage of their know-how in terms of low cost and low power consumption, Lattice now enters the mid-range FPGA market with state-of-the-art silicon.
It offers a faster fabric and numerous high-speed transceivers channels (**Up to 28 channels at 25 Gbps for the largest Avant-X**) that are reaching FPGA NRZ transceiver's high boundary.

At ALSE we have a long history of working with practically all the FPGA vendors and families, and by consequence with all types of FPGA transceivers. We have developed many transceiver-based IPs including : **HDMI**, **SDI**, **JESD204b**, **Gedek** (100M/1G Ethernet), **10GEDEK** (10G Ethernet), **Aurora 8B/10B, Aurora 64B/66B** !
Therefore, having gained access to a brand new **Lattice** **Avant-G Versa board**, we could not resist to tryout this new FPGA family by porting our Aurora IPs on this platform.

In this Application Note we demonstrate a Reference Design based on our **Aurora 8B/10B IP ported on the Lattice Avant families**.

3.1. Introducing the Aurora 8B/10B Protocol

If you aren't yet familiar with the Aurora 8B/10B protocol, here is a brief introduction. You can find all the details in the full documentation available (refer to our Bibliography).

Aurora 8B/10B is an open-source, interconnect, framing protocol, published by Xilinx in 2002.

As stated in the name, it encodes data frames using the standard 8B/10B scheme and transmits them serially through SERializers/DESerializers (SERDES) aka Transceivers. The Aurora 64B/66B is similar (with one extra data interface and a few more behavioral differences) but it is designed for higher data-rates, using a 64B/66B encoding and wider data-paths.
Aurora 8B/10B transmits user **data** through two interfaces :

- the user **Protocol Data Unit** (**PDU**) interface is the main data interface and is used to transfer the main Data.
- the **User Flow Control** (**UFC**) interface is a higher-priority interface that is typically used to transmit Control information.

A third type of frame is transmitted upon user request :

- **Native Flow Control** (NFC) packets.
  It is used to send pause requests to the Aurora partner. When an NFC request is received by an Aurora, it stops transmitting data for a given amount of time, stalling user interface with back-pressure. Data transfer resumes after a specified delay.

The Aurora protocol can be used in **Duplex mode** (both RX and TX) or in **Simplex** mode (RX-only or TX-only). In our Reference Design, **we use the Duplex Mode**.

Up to 16 serial links (transceivers) can be used in parallel to augment the bandwidth accordingly. In our Reference Design we will use a single lane to simplify the interconnection between various boards (not many boards offer multiple lanes, like those using QSFP “quad” connectors).

Xilinx (AMD) Aurora IPs (both 8B/10B and 64B/66B) are included at no extra cost with Vivado subscription editions, and it is available on many AMD FPGAs that have transceivers.

It is common to use our Aurora IPs to connect an AMD FPGA to another brand FPGA (or an ASIC).
In this application note, we will use a Xilinx/AMD VC707 board as a link partner to also demonstrate the full compatibility and interoperability between ALSE Aurora IP and Xilinx/AMD Aurora IP.
But boards from any other vendors (Lattice, Altera, Microchip, Efinix) can be used as Link Partner.

It’s interesting to note that we have cases where we port our Aurora 8B/10B IP on some of Xilinx/AMD latest FPGAs, where the Aurora 8B/10B IP is no longer available, probably due to the fact that the latest and fastest AMD transceivers don’t offer any more 8B/10B encoding ! This is not a show stopper for us since our IP includes our own 8B/10B soft-PCS.

3.2. Requirements

To use our Reference Design ,you must have access to the following tools :

- **Lattice Radiant** 2025.1.1 **Programmer** and **Reveal** tools. Both are available as Standalone tools (at no cost) and can be downloaded from the Lattice website, if you do not want to download and install the full Radiant software package.
- Access to **Vivado 2023.2** and a License for Virtex-7 device if you want to compile the full source project that we deliver in source code. This is optional : our delivery contains a bitstream file and a debug file which can be used as is.
- **Renesas Timing Commander** free-software for RC38612A clock programming (see Appendix A Programming the RC38612A Clock Generator for programming procedure) – available for Windows only.

4. The Hardware

We use the feature-rich Lattice board reference **LAV-G70-VERSA-EVN-AS2 kit**.
This board features an Avant-G FPGA (exact reference is LAV-AT-G70-3LFG1152C).

Note : AVANT-G and AVANT-X FPGAs use **the same transceiver**.

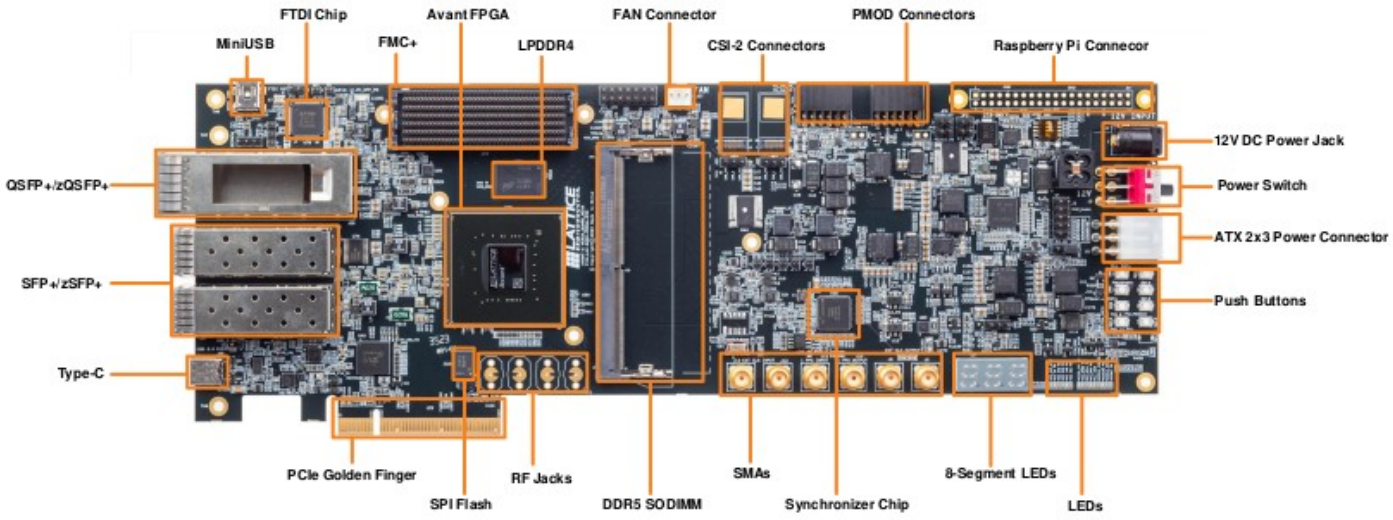


Figure 1.1. Top View of Avant-G/X Versa Board

4.1. Transceiver architecture

As it is often the case (but not always!), the Avant-G (and Avant-X) transceivers lanes are grouped by 4 to form what we commonly call **transceivers Quads**.
In a Quad, the four lanes are closely related. They have an exclusive access to a set of common resources and can share the same clocks for emission or reception if it is required by a protocol. They can, of course, work independently as well. This will be configured through the vendor transceiver IP that we use in our Reference Design.
In case of a multiple lanes Aurora configuration, we would use lanes from a common Quad to exploit these close relationships at our advantage. This would be optimal but not compulsory : with more than four lanes (up to 16 is possible) we would use lanes from several Quads.

On this board, we can access the 28 transceivers lanes through four types of connectors :

- one **x8 PCIe connector** dedicated to the PCIe protocol (unusable for our test-case), connected to SerDes Quads Q4 and Q5,
- 4 *unmounted* "**RF Jacks**" connectors (SerDes Quad Q6),
- 3 Quad transceivers (Q0, Q1 and Q2) accessible through the **FMC** connector.
- **2 zSFP+** cages (Serdes Q6) and **1 zQSFP+ cage** (SerDes Q3) for a total of 6 lanes, supporting up to 25G per link.

We will use one zSFP+ cage for our Aurora 8B/10B demonstrator.
We will actually use **J27** (bottom SFP cage on the picture), linked to **Quad Q6**.

4.2. RC38612A Clock Generation

If you are familiar with transceivers you know that you always *need* a high quality **Reference Clock** to generate the set of clocks required for all the SerDes functions (CDR, Serial transmission, RX and TX user interface clocking...). This Reference Clock can often come from different sources : dedicated FPGA ports specific to a Quad, sometimes from the Upper or Lower Quad as well and sometimes even from the FPGA fabric itself. In our case, each Avant-G/X transceiver Quad **has only one possible clock source, through a dedicated FPGA port**. This must be considered early if you design your own board.

Beware : On the board, the clock port feeding Quad Q6 comes from an Renesas external multi-channel clock synchronizer RC38612A. By default (Kit User Guide), the clock is supposed to run at 156.25 MHz. It is the frequency that we need to run the Reference Design.
Our Reference Design does not reprogram the clock generator : you must use the free Renesas software (available here - Windows only) to program the board so that it generates the required 156.25 MHz clock if it is not doing it by default.

To access the RC38612A (and its configuration EEPROM) through the I2C interface (MINI USB connector), **some jumpers must be present on the board : JP10, JP11 and JP12**. See Appendix A Programming the RC38612A Clock Generator for all the details.

BEWARE: the three jumpers **JP10, JP11 and JP12 must be OPEN** at power-up when you want to use the PLL configuration stored in its associated EEPROM.

→ Be sure to remove the jumpers once the EEPROM is configured correctly.

4.3. Aurora Partner : Xilinx Virtex-7 Evaluation Board VC707

Every Duplex configuration needs a partner : here we use the **Xilinx Virtex-7 Evaluation Board**.

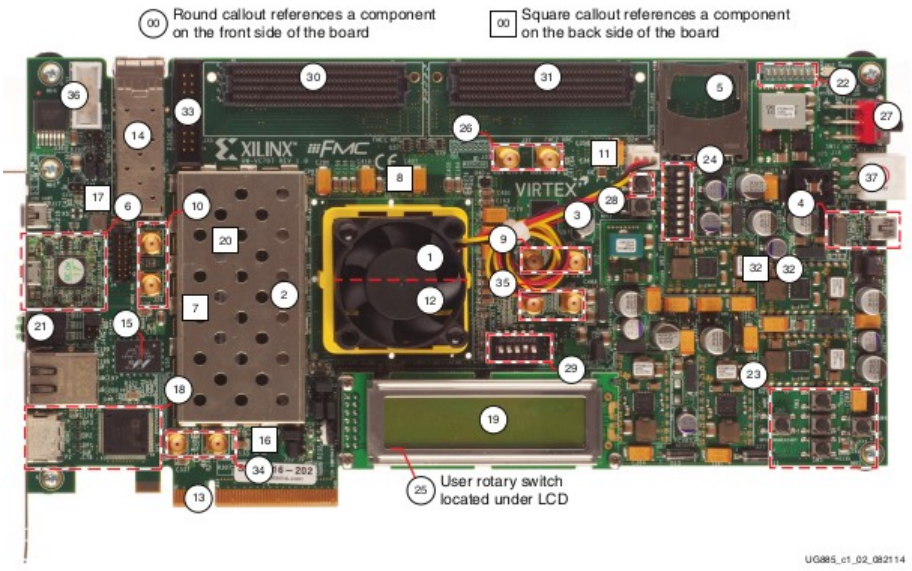


Figure 1-2: **VC707 Board Component Locations**

On this board, we use the only available SFP connector (component 14) to connect to our Avant-G Kit.

The choice of a Xilinx partner demonstrates that the ALSE Aurora 8B/10B IP is perfectly compatible and interoperable with the Xilinx IP.

Note : If you do not have access to this (very expensive !) Xilinx-AMD board, you can still test our Avant-G Reference Design :

- using an SFP loop-back module,
- using a second Lattice Avant-G Versa Reference Kit,
- Porting the Xilinx reference design that we deliver in source code to another Xilinx board.
- Asking ALSE a Reference Design for a specific target.

5. The Reference Design(s)

Let's talk now about the Reference Design we are going to build around our Aurora IP.

Most of the blocs are not Lattice specific and are common to all our Aurora Reference Designs for various FPGA device vendor and families.

All the Generators and Checkers block are present on both Xilinx and Lattice Reference Designs.

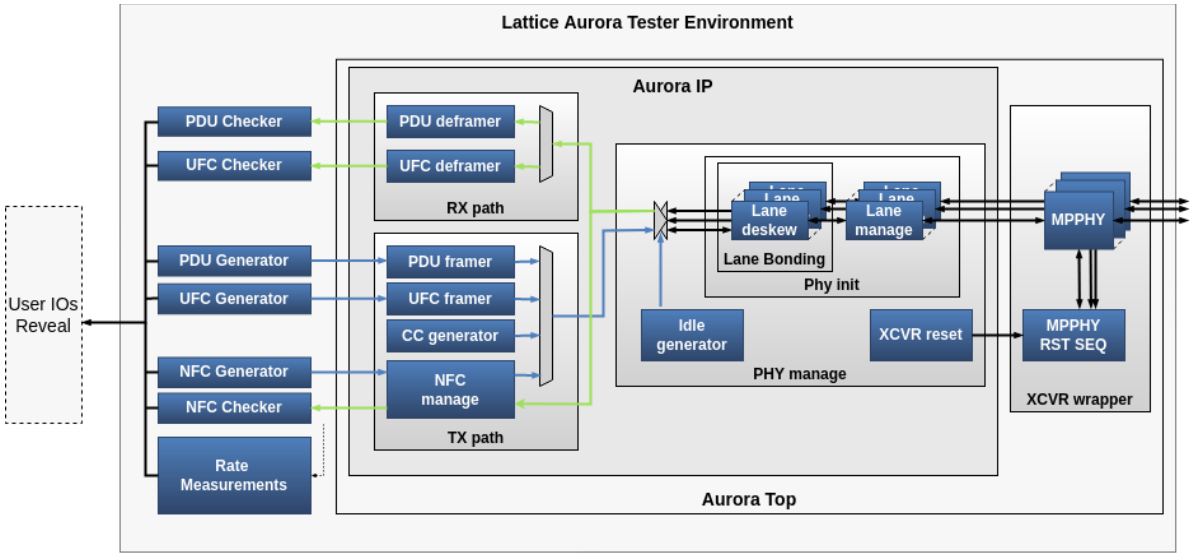


Figure 1: Lattice Avant-G/X Reference Design

5.1. Data Generators and Checkers

Most of the data volume is generated towards the PDU interface, which is the main Data interface of the IP. The **PDU Generator** builds data packets of various length, filled by a byte counter.
This data is output through a streaming interface towards the transmit user interface of the Aurora IP.

A bit-accurate verification is performed on the receiving side by the **PDU Checker**.
Three counters are made observable in real time to the user : the number of frames generated, the number of frames received, and the number of errors detected by the checker.

It is more or less the same as far as the **UFC interface** is concerned. Three equivalent counters are available for this interface as well.

The third interface : **NFC**, is different. To demonstrate and test this feature, the generator creates pause requests to the remote Aurora (not too often to not stall the bus for too long). Of course, it has direct consequence on the transfer rate of the Aurora partner.

On the **Lattice device**, all the **Generators** are **enabled statically** and cannot be disabled in the current implementation. However on the **Xilinx FPGA**, **each interface can be enabled or disabled** through a Virtual Input/Output (VIO) module.

Several generation options are actually available to customize the PDU generator but for the sake of simplicity, they are fixed in the Lattice. These options are available in the Xilinx design through the VIO module. They are discussed later on the Xilinx side.

Note that Lattice does now offer the equivalent of the Xilinx' VIO or Altera's Source & Probes.

5.2. Top-Level

Let us review both sides of the Reference Design, starting with the **Avant-G**.

Here is the top-level declaration (VHDL) of our Lattice Reference Design :

```
---------------------------------------
entity LAVG_VERSA_top is
---------------------------------------

  generic (
    vendor_name    : string    := "Lattice"; -- Vendor Name
    device_family  : string    := "LAVG";    -- Lattice Device Family
    nb_lanes       : integer   :=   1;       -- Number of XCVR lanes
    lane_width     : integer   :=  16;       -- 16 or 32 bits lane
    streaming_mode : std_logic := '0';       -- Framing Mode if '0'
    crc_enable     : std_logic := '0';       -- CRC Enabled if '1'
    simplex_tx     : std_logic := '0';       -- Simplex Tx only if '1'
    simplex_rx     : std_logic := '0';       -- Simplex Tx only if '1'
    DEBUG          : boolean   := False;     -- True for simulation
    ext_clk_freq   : positive  := 100e6      -- External Clock Frequency
  );

  port (

    -- ---------------------------------------------------------------
    --  Reset and Clocks
    -- ---------------------------------------------------------------
    arst_n           : in  std_logic;   -- Asynchronous active low reset
    clk              : in  std_logic;   -- Free-running clock (100 MHz)
    cnt_rstn         : in  std_logic;   -- Counter reset button (active low)

    -- ---------------------------------------------------------------
    --  zSFP J27 Cage Interface
    -- ---------------------------------------------------------------
    sfp_refclk_p   : in    std_logic; -- SFP Transceiver Reference Clock (P)
    sfp_refclk_n   : in    std_logic; -- SFP Transceiver Reference Clock (N)
    --
    sfp_rx_p : in  std_logic_vector(nb_lanes - 1 Downto 0); -- SFP Rx (P)
    sfp_rx_n : in  std_logic_vector(nb_lanes - 1 Downto 0); -- SFP Rx (N)
    sfp_tx_p : out std_logic_vector(nb_lanes - 1 Downto 0); -- SFP Tx (P)
    sfp_tx_n : out std_logic_vector(nb_lanes - 1 Downto 0); -- SFP Tx (N)

    -- ---------------------------------------------------------------
    --  User Leds (Active high)
    -- ---------------------------------------------------------------
    user_led : out std_logic_vector(2 downto 0)

  );
```

In the **Generic** declaration we find the Aurora configuration used here : a **1 lane duplex** instance, **2 bytes** per lane, in **framing** mode (PDU packetizer activated), **without CRC**.

The serial lane speed is **5 Gbit/s** – it is close from the maximum use case for 8B/10B encoding. For speeds higher than 6 Gbit/s, 64B/66B encoding format is used instead and we have an Aurora 64B/66B IP available.

Obviously, the Aurora configuration of **both sides** of the link **must exactly match** (see later for Xilinx IP configuration).

The external clock (*clk*) is a 100 MHz free running clock, used to generate the Power-On Reset and to measure the transceiver clocks frequencies and data rates.

arst_n and *cnt_rstn* **Reset** ports are connected to active-low push buttons, respectively **SW13** and **SW12**.

User LEDs are used to output status information.

5.3. User I/Os

The *arst_n* input is used to reset the whole design, including counters and transceivers.
Pressing the SW13 push-button causes the Aurora link to go down.

The *cnt_rstn* button is used to reset the **counter values** only.

The three User LEDs are used to provide status information :

- User LED D47 displays a heartbeat, indicating that the bit-stream is well programmed : it uses the free-running 100 MHz clock and is reset when pressing *arst_n*.
- User LED D45 is ON when the **Aurora Link is up**.
- User LED D46 is ON when an **Error** is detected by the PDU RX checker.

5.4. Avant-G Design Reveal Description

Live access to the counters as well as other computed values is achieved through the Lattice Radiant **Reveal** tool. Here is the full list of signals available in Reveal :

Name	Size	Description
link_up	1	'1' means Aurora Link is Up
PDU Interface		
cnt_pdu_frame_tx	32	Number of PDU frames sent
cnt_pdu_frame_rx	32	Number of PDU frames received
cnt_pdu_frame_err	32	Number of errors detected in received frames.
UFC Interface		
cnt_ufc_frame_tx	32	Number of UFC frames sent
cnt_ufc_frame_rx	32	Number of UFC frames received
cnt_ufc_frame_err	32	Number of errors detected in received frames.
NFC Interface		
cnt_nfc_wait_cnt_tx	32	Sum of idle cycles requested sent
cnt_nfc_wait_cnt_rx	32	Sum of idle cycles requested received
Frequencies and Data Rates		
tx_clkout_freq	32	Measured frequency of transmit data interface (~250 MHz)
rx_clkout_freq	32	Measured frequency of received data interface (~250 MHz)
tx_pdu_rate	32	TX PDU speed in MBytes/s
rx_pdu_rate	32	RX PDU speed in MBytes/s

Two types of triggers are available :

- The **link-up** signal, verifying that the link never goes down apart from reset activation.
- Error counter values, if you want to run a test over a long period of time, triggering if an error is detected (setting condition to *error counter > 0*).

5.5. MPPHY IP configuration

Using the transceiver lanes implies using the **MPPHY** IP.
As it is usually the case, the configuration of the IP is done in two parts :

- Configuration of the **PMA** – we use here one lane, running at 5 Gbit/s, with a reference clock running at 156.25 MHz.
  Since we want a 2 bytes interface, we set the PMA to use a 20 bit wide data-path.
- Configuration of the **PCS** – the Avant-G PCS supports 8B/10B encoding.
  When it is available, it is always a good idea to use the vendor's PCS (Hard or Soft) IP, and that's what we did in this Reference Design. When it's a Hard IP, this also saves some logic resources.
  Note that we have designed our own PCS, including the word aligner, which can be tricky, but we didn't have to use it in this case.

Note that with a data-path of 20 bits (16 bits after 8b/10b decoding) running at 5 Gbit/s, the data interface is running at 250 MHz! This can be challenging depending on the FPGA. However we found that timing convergence was reached quite easily on the Avant-G FPGA (see later Fmax result) : a good point for this family !

Note : Our Aurora 8B/10B PCS supports also a wider data-path (32 bits encoded as 40 bits) to ease the timings.

5.6. The Xilinx Reference Design

As said before, on Xilinx/AMD side, Checkers and Generators are exactly the same as on the Lattice side. A counter reset (and a System Reset) is available providing the same functionality as in the Lattice Design on SW3 (and SW6). Five User LEDs are used for :

- USER_LED 0: OFF when global reset is active
- USER_LED 1: ON when **Link is UP**
- USER_LED 2: ON when **lane(0) is UP** (equivalent to USER_LED 1 in our case)
- USER_LED 3: ON when a **PDU Error** has been detected
- USER_LED 4: ON when a **UFC Error** has been detected

Again, the Xilinx Reference Design is available as copyrighted Source Code as well as a pre-compiled bitstream. You will find a TCL script in the *fit/xilinx* folder. **Beware : the script has been designed for a (Vivado) Linux environment!**
For Windows you will have to adjust relative data paths.
Run the script in Vivado 2023.2 to generate the full project.
From there, launch the full compilation and generate the bitstream.

However, instead of ALSE Aurora IP, we use the Xilinx Free Aurora IP.

Here is a schematic view of the Xilinx design :

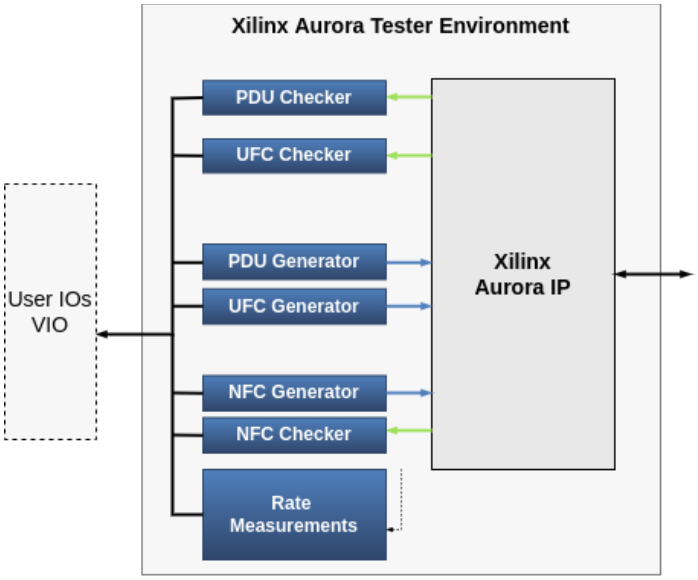


Figure 2: Xilinx Reference Design

In the schematic you may notice that we do not need to use Xilinx transceiver IP! Indeed, the Xilinx Aurora IP offers the possibility (that we use here) to generate automatically the transceiver IP, saving some time and trouble and making it really simple to use!

Note : When purchasing an Aurora IP from ALSE, it is delivered with a transceiver *wrapper*, so you don't have to manually configure and instantiate the vendor's transceiver(s). But the transceivers remain always accessible and you can tweak them or control them as you wish. This is important especially when our IP is purchased as a netlist.

5.7. Xilinx Aurora IP Configuration

Here is the Xilinx Aurora IP configuration used in the Xilinx FPGA, compatible with the ALSE settings :

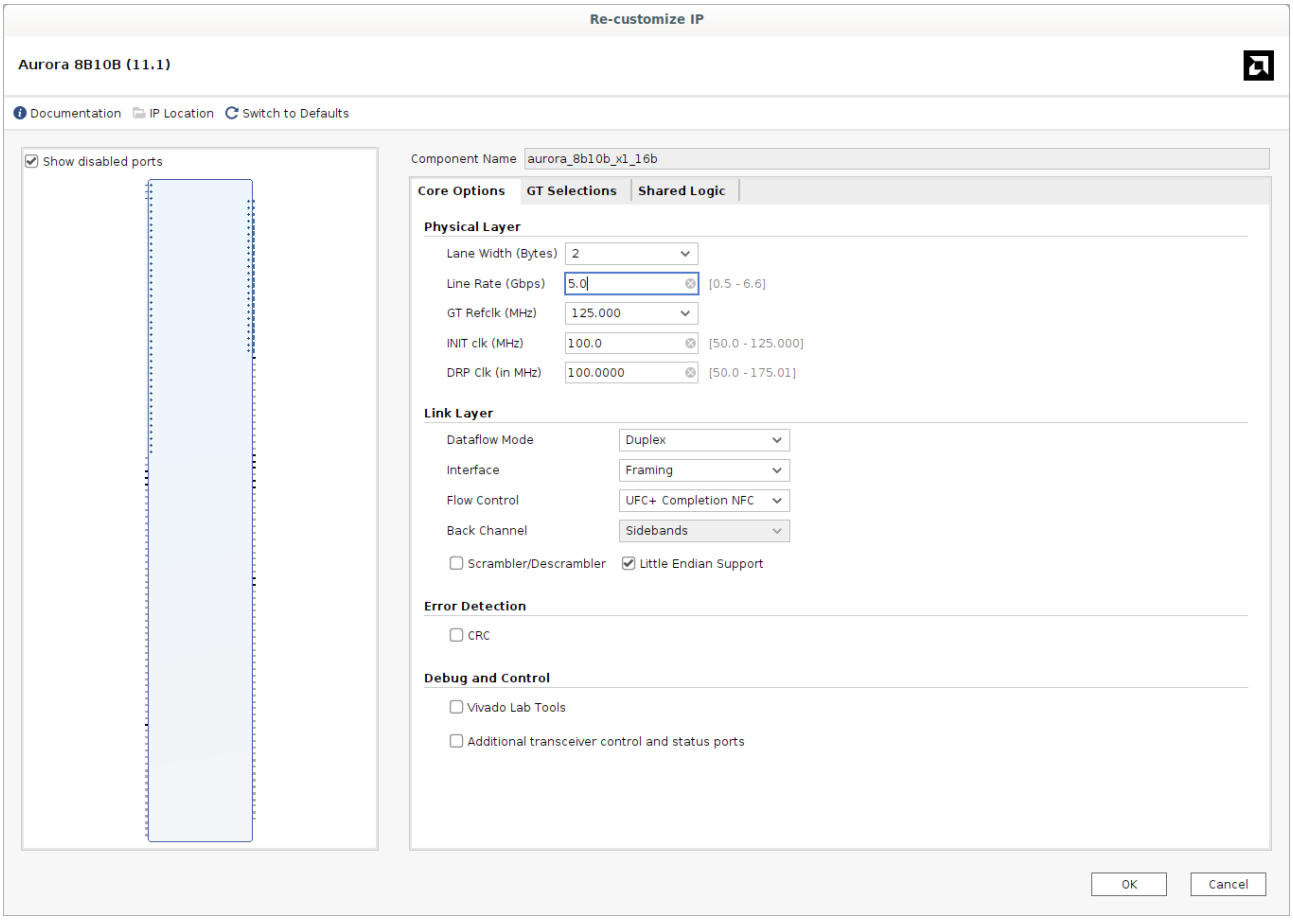


Figure 3: Xilinx-AMD Aurora IP Configuration

5.8. Xilinx Virtual Input/Output

Xilinx/AMD Virtual Input/Output module is a very handy way of getting control (sources) and status (probes) inside the FPGA through the JTAG port. It is similar to Altera's Sources and Probes.
We use it here to configure the generator module and to observe the live counters values from frame generators and checkers.

Name	Size	VIO port	Description
reset_cnt	1	**probe_out12**	Count Reset
Generation Enabled			
tx_pdu_enabled	1	**probe_out0**	PDU Gen Enabled
tx_ufc_enable	1	**probe_out2**	UFC Gen Enabled
tx_nfc_enable	1	**probe_out3**	NFC Gen Enabled
PDU Gen Config			
tx_pdu_underflow	1	**probe_out1**	PDU Gen Inserts Idle cycles if '1'
tx_pdu_endless	1	**probe_out5**	PDU Gen Endless Frames (no EOP)
tx_pdu_fixed_size_on	1	**probe_out4**	PDU Gen Fixed Frame Mode
tx_pdu_fixed_size_max	32	**probe_out6**	PDU Gen Fixed Frame Mode Max Size
tx_pdu_fixed_size_min	32	**probe_out7**	PDU Gen Fixed Frame Mode Min Size
tx_pdu_fixed_size_incr	32	**probe_out8**	PDU Gen Fixed Frame Mode Size Increment
tx_pdu_fixed_size_ifgap	32	**probe_out9**	PDU Gen Fixed Frame Mode Inter-Frame Gap
tx_pdu_fixed_cut_size	32	**probe_out10**	PDU Gen Fixed Frame Mode Cut Size
tx_pdu_fixed_cut_gap	32	**probe_out11**	PDU Gen Fixed Frame Mode Cut Gap
PDU Frame Count			
cnt_pdu_frame_tx	32	**probe_in0**	Count PDU Frame Transmitted
cnt_pdu_frame_rx	32	**probe_in1**	Count PDU Frame Received
cnt_pdu_frame_err	32	**probe_in2**	Count PDU Frame Error
tx_pdu_frame_size	32	**probe_in6**	PDU Gen Fixed Frame Mode Current Size
UFC Frame Count			
cnt_ufc_frame_tx	32	**probe_in3**	Count UFC Frame Transmitted
cnt_ufc_frame_rx	32	**probe_in4**	Count UFC Frame Received
cnt_ufc_frame_err	32	**probe_in5**	Count UFC Frame Error
NFC Wait Count			
cnt_nfc_wait_tx	32	**probe_in7**	Count NFC Wait Requested
cnt_nfc_wait_rx	32	**probe_in8**	Count NFC Wait Received
Computed Rates and Frequency			

user_clk_freq	32	**probe_in9**	Aurora User Clock Computed Frequency
tx_pdu_rate	32	**probe_in10**	TX PDU Transmit Rate (MBytes/s)
rx_pdu_rate	32	**probe_in11**	TX PDU Received Rate (MBytes/s)

About the Fixed Frame Mode

Instead of using frames of pseudo-random length, we can generate PDU frames of known length using the ***Fixed Frame Mode***.

To enable this generation mode, the **tx_pdu_fixed_size_on** must be set to '1'.

Once enabled, frame length are computed using the **tx_pdu_fixed_size_min/max/incr** values: frame length starts at **tx_pdu_fixed_size_min** and increases by **tx_pdu_fixed_size_incr** each frame until reach the maximum **tx_pdu_fixed_size_max** size.
Then, the process starts over.

When active, the current frame length used is available as **tx_pdu_frame_size**.
The **tx_pdu_fixed_size_ifgap** defines the inter-frame gap in clock cycles.
tx_pdu_fixed_cut_size/gap are used to insert idle clock cycles inside a frame generation – every **tx_pdu_fixed_cut_size** valid data cycles, **tx_pdu_fixed_cut_gap** idle cycles are inserted by the generator.

This mode is very useful to model frames from a specific application.
When **tx_pdu_fixed_size_on** is set to '0', parameters described above are not used.

6. Testing Procedure and Results

6.1. Avant-G Compilation Results

Full project compilation summary of the reference design (IP + generators, checkers and Reveal):

LAVG_VERSA_x1 Project Summary			
Implementation Name:	impl_1	Strategy Name:	Strategy1
Part Number:	LAV-AT-G70-3LFG1156C	Performance Grade:	3
Synthesis:	Synplify Pro	Top Module:	LAVG_VERSA_top
Timing Errors:	Place & Route, 0 (Setup), 0 (Hold)	IP Number:	1
Project File:	/home/fbatard/__SSTR/ALSE/Aurora/aurora_8b10b/fit/lattice/avantg_versa_x1_16b/LAVG_VERSA_x1.rdf		
Implementation Location	/home/fbatard/__SSTR/ALSE/Aurora/aurora_8b10b/fit/lattice/avantg_versa_x1_16b/impl_1		

Resource Usage			
LUT4:	1216/397440 (0.306%)	IO Buffers:	6/558 (1.075%)
PFU Register:	1659/397440 (0.417%)	EBR:	0/990 (0.000%)

Figure 4: Full compilation summary

Here are the compilation result for the Aurora IP configured in one lane, running at 5 Gbit/s on the Aurora Avant-G (including transceiver IP):

impl_1	LUT4	Logic	Distributed RAM	Ripple Logic	PFU Registers	IO Registers
gen_ip_framing.i_ip_framing	702(0)	616(0)	6(0)	80(0)	633(0)	0(0)
i_core	639(0)	559(0)	6(0)	74(0)	548(0)	0(0)
i_xcvr_wrapper	63(6)	57(6)	0(0)	6(0)	85(12)	0(0)

Figure 5: Aurora 8B/10B P&R resource usage

The IP and the transceiver (embedded in *i_xcvr_wrapper*) fit in just **700 LUT4s** which represents less than 0,2% of the full device (400**K** LUT4s available). **Our Aurora 8B/10B IP is very compact !**
And based on our experience it takes typically half a day to integrate it into a project.

It is small, but is it fast? Let us remember that for a 2 bytes data-path with a serial link running at 5 Gbit/s, we need the user RX and TX interface clocks (coming from the transceiver) to run at 250 MHz. Here are the slack and Fmax reports on both clocks:

```
------------------------------------------------------------------------------------------------
           Clock tx_outclk            |                    |       Period       |     Frequency
------------------------------------------------------------------------------------------------
 From tx_outclk                       |             Target |          4.000 ns |        250.000 MHz
                                      | Actual (all paths) |          3.200 ns |        312.500 MHz
```

Figure 6: TX output clock timing

```
------------------------------------------------------------------------------------------------
           Clock rx_outclk            |                    |       Period       |     Frequency
------------------------------------------------------------------------------------------------
 From rx_outclk                       |             Target |          4.000 ns |        250.000 MHz
                                      | Actual (all paths) |          3.241 ns |        308.547 MHz
```

Figure 7: RX output clock timing

The answer is : **yes**, The Avant-G is fast enough and our Aurora IP meets the timings with a very comfortable margin !

6.2. Testing Procedure: Setup the boards

Configuring the VC707 board

- Power cycle the board.
- Connect the JTAG cable to your PC with Vivado Debug tools installed.
- Open Vivado Hardware Manager and click on Open Target: if connected properly, the Virtex-7 device (xc7…) should be detected and appear in the Hardware Manager window.
- Right-click on the device and select the provided bit-stream (*.bit*) and the Virtual I/O debug file (*debug_nets.ltx*). If you recompiled the project, use your own!

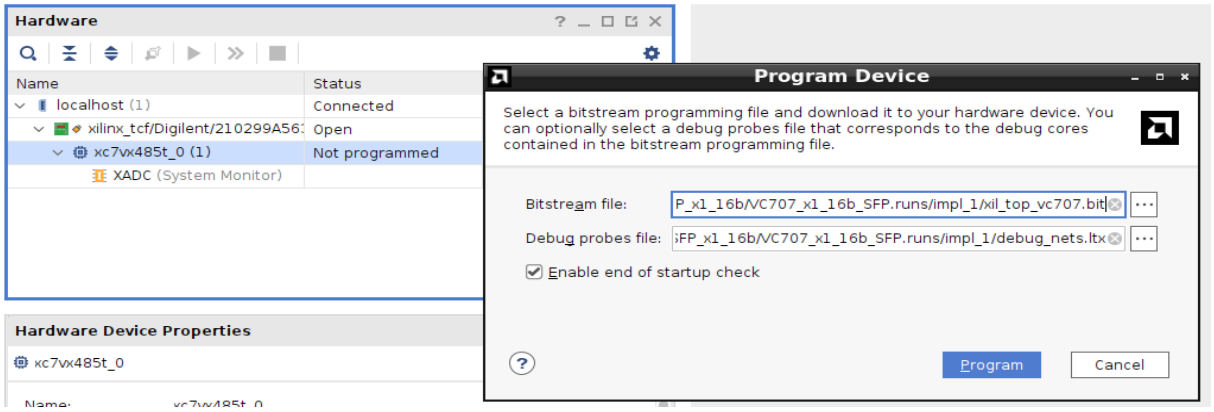
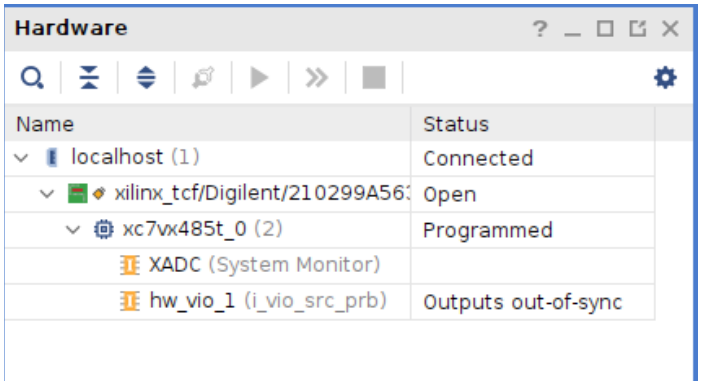


Figure 8: Vivado Hardware Manager

Figure 9: VIO is now detected

- Once programmed, the *hw_vio_1 (i_vio_src_prb)* should now appear in the hardware manager window:



- Add all the VIO source and probes, and rename probes using the correspondence table given above.
- Changing the source type to Toggle Button (*reset_cnt*) or *Active-High Button* (*PDU/UFC/NFC_enable*, *underflow*, *endless*, *fixed_size_on*) and renaming the probe for clarity is a good idea.
  Doing so you get something like this:

Name	Value	Activity	Direction	VIO
reset_cnt	0		Output	hw_vio_1
tx_pdu_enable	1		Output	hw_vio_1
tx_pdu_underflow	0		Output	hw_vio_1
tx_ufc_enable	0		Output	hw_vio_1
tx_nfc_enable	0		Output	hw_vio_1
tx_pdu_endless	0		Output	hw_vio_1
tx_pdu_fixed_size_on	0		Output	hw_vio_1
tx_pdu_fixed_size_max[31:0]	[H] 0004_E200		Output	hw_vio_1
tx_pdu_fixed_size_min[31:0]	[H] 0004_E200		Output	hw_vio_1
tx_pdu_fixed_size_incr[31:0]	[H] 0000_0000		Output	hw_vio_1
tx_pdu_fixed_size_ifgap[31:0]	[H] 0000_9C40		Output	hw_vio_1
tx_pdu_fixed_cut_size[31:0]	[H] 0000_0320		Output	hw_vio_1
tx_pdu_fixed_cut_gap[31:0]	[H] 0000_004B		Output	hw_vio_1
cnt_pdu_frame_tx	[H] 0000_0000		Input	hw_vio_1
cnt_pdu_frame_rx	[H] 0000_0000		Input	hw_vio_1
cnt_pdu_frame_err	[H] 0000_0000		Input	hw_vio_1
cnt_ufc_frame_tx	[H] 0000_0000		Input	hw_vio_1
cnt_ufc_frame_rx	[H] 0000_0000		Input	hw_vio_1
cnt_ufc_frame_err	[H] 0000_0000		Input	hw_vio_1
tx_pdu_frame_size	[H] 0000_0000		Input	hw_vio_1
cnt_nfc_wait_tx	[H] 0000_0000		Input	hw_vio_1
cnt_nfc_wait_rx	[U] 0		Input	hw_vio_1
user_clk_freq	[U] 250007084		Input	hw_vio_1
tx_pdu_rate	[U] 0		Input	hw_vio_1
rx_pdu_rate	[H] 0000_0000		Input	hw_vio_1

Figure 10: VIO windows with renamed sources and probes

At this stage, *user led 0* should be the only led ON. You should see a *user_clk_freq* close to 250MHz.

Programming the Lattice LAV-G70 Versa Evaluation Board

It is now time to program the Avant-G FPGA.

- **The first step is to enable programming through JTAG by closing Jumper JP1**.
  When unconnected, the FPGA is programmed only from the configuration flash and is not accessible through JTAG !
- Make sure that jumpers **JP10**, **JP11**, and **JP12** are **OPEN** so that the external PLL configures correctly.

Here, we assume that the Renesas RC38612A associated EEPROM **contains a valid configuration**, generating the **156.25 MHz Quad transceiver reference clock** (see Appendix A Programming the RC38612A Clock Generator).
If you are unsure of the PLL configuration, run the following steps and check using Reveal tool transceiver generated clocks frequencies. Depending on the results, you will know if the configuration is valid or not.

- Connect using a **5G capable SFP cable**, J27 SFP+ cage to the VC707 SFP+ cage.
- Connect the MINI USB (**not** USB-C) to your PC equipped with Radiant Programming and Reveal Software to access FPGA through JTAG.
- Lattice Programmer should detect the FTDI chip present on the board : the FPGA is located on the second port of the chip. Select the correct Port in the software and add the bit-stream for configuration.

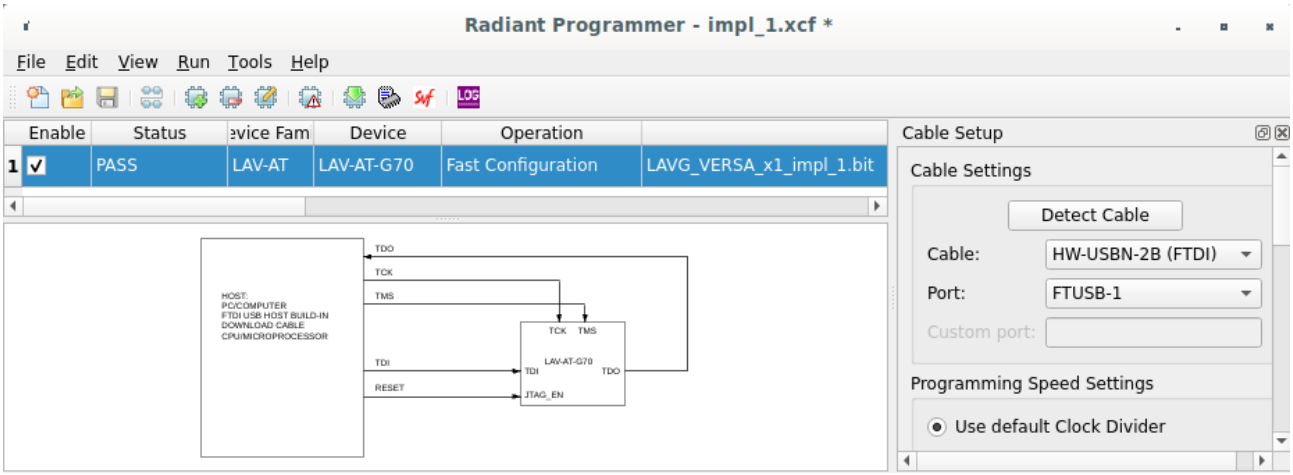


Figure 11: Lattice Programmer connect to second FTDI (FTUSB-1)

Program the FPGA with ALSE ***LAVG_VERSA_x1_impl_1.bit*** pre-compiled bitstream with following consequences :

- **LED D47** of Lattice Kit **beats slowly** indicating that the binary was programmed correctly,
- **LED D45** of Lattice Kit **is ON**, indicating ALSE Aurora **Link Up** on Avant-G side,
- **User LEDs 1 and 2 are ON** on VC707 board, indicating Aurora **Link Up** on AMD side.

LED D46 (Lattice) and User LEDs 3 and 4 (AMD) should stay OFF : they are ON only in case of error detection.

If you do not have a **link-up** at this stage, you probably have one of these two issues :

- you **did not close Jumper JP1** of Versa Kit which enables the Avant-G programming,
- you **did not configure the Renesas clock generator** correctly.

In both cases, you can use the Reveal Analyzer tool : if it does not find an instance to connect, the bit-stream was not programmed correctly.

If *rx_clkout_freq* and *tx_clkout_freq* values do not indicate 250Mhz, you have a frequency issue on RC38612A clock generator! Go to the Appendix A Programming the RC38612A Clock Generator and follow the EEPROM configuration procedure.

If you have a link-up, you are now ready for testing ! All three generators (PDU, UFC, NFC) are active in Lattice design. On AMD side, they are active by default as well, but can be enabled and disabled dynamically using the Virtual I/O utility.

Received and transmitted PDU data rates are displayed on both debugging tools (VIO, Reveal) as Mbytes/s. Check that they match, and how enabling/disabling each interface influence the effective data transfer rates !

6.3. Testing Procedure: Base test at start-up

When link-up if done, the setup should look like this:

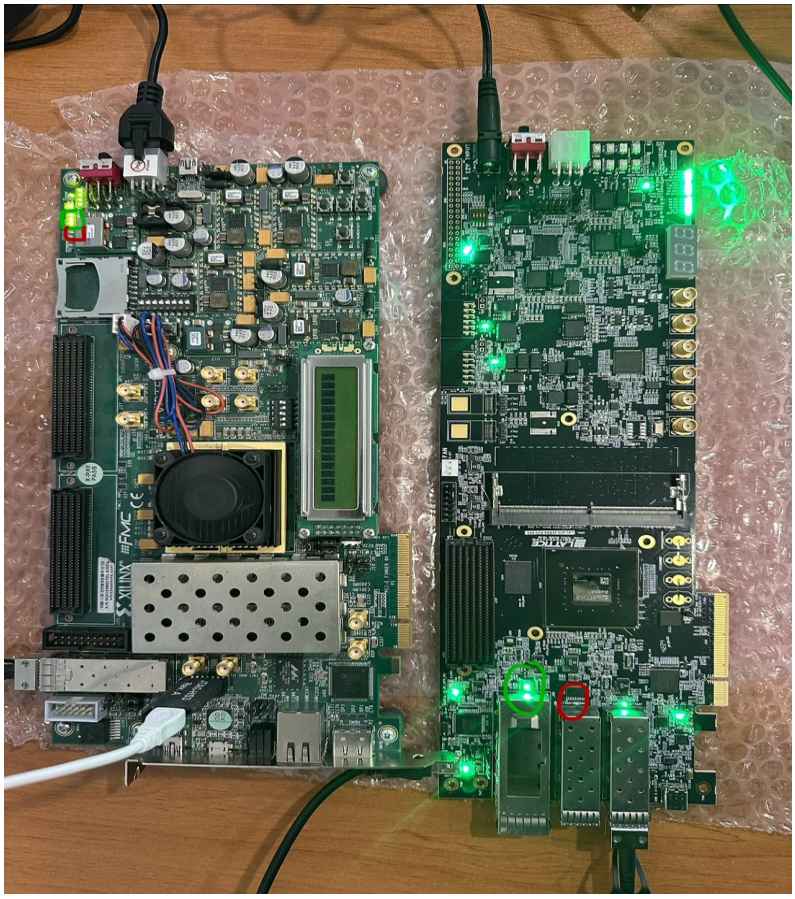


Figure 12: Avant-G Versa (Left) and VC707 (Right) after link-up

Link LEDs (circled in green) are ON and error LEDs (circled in red) are OFF.

On both side, frames counters increment :

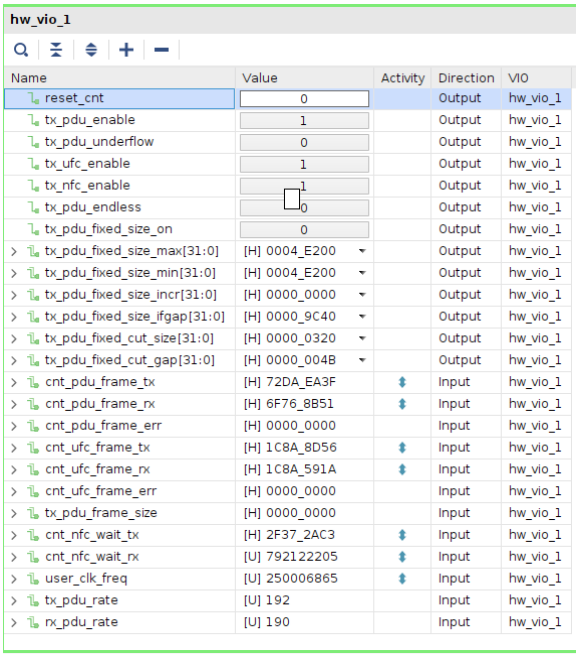


Figure 13: VIO all generator ON

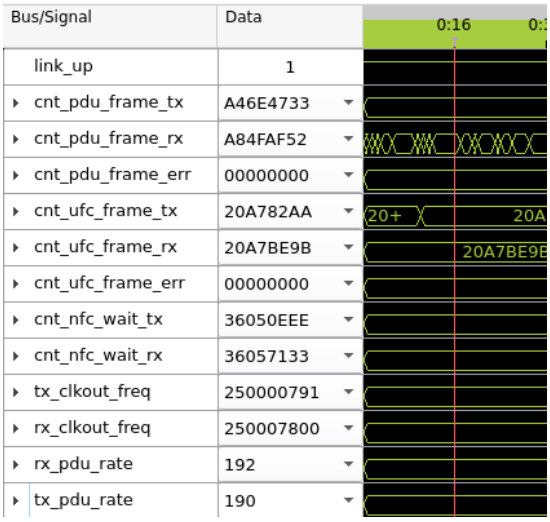


Figure 14: Reveal all generator ON

We notice that the TX PDU rate on one side matches the RX PDU rate of the other side.

The rate is quite low compared to the maximum theoretical data rate (4 Gbit/s).
The Avant-G Aurora is sending and receiving PDU data at 1,5 Gbit/s in parallel.
It would be a lot higher if we disable UFC and NFC sending as well as NFC sending from the Xilinx partner (causing transmit pauses on Lattice side).

Interface clocks (*user_clk* on AMD, *rx/tx_clkout*) are all running at 250 MHz, as expected.

Now, if we stop the generation of all frames on the Xilinx side (not configurable on Lattice side) we get:

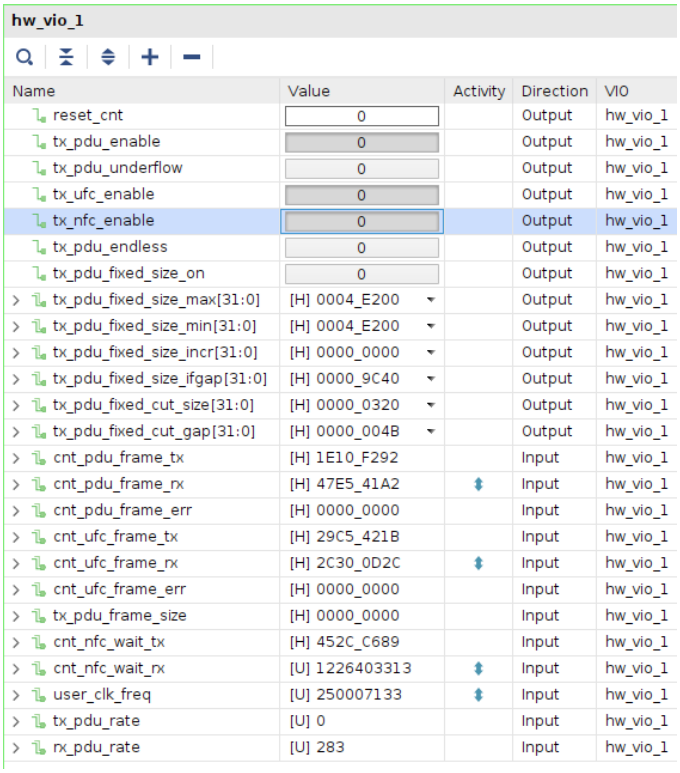


Figure 15: VIO all generator OFF

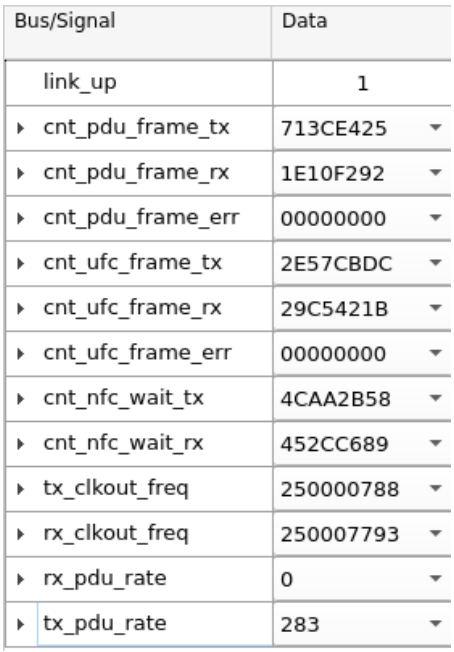


Figure 16: Reveal RX only

As we can expect, when we stop generators on Xilinx side and compare the number of frames transmitted and received on both sides we get the same values : no data was lost in the process!

Frame Type:	**PDU**	**UFC**	**NFC**
TX/RX frames	0x1E10F292	0x2E57CBDC	0x452CC689

6.4. Testing Procedure: Reach maximum transfer speed

As said before, to maximize the measured transfer speed (PDU rate) by our rate module we must maximize the number of valid PDU cycles. For this, three things must be done :

- Disabling any other types of frame transmission (UFC, NFC)
- Disable NFC request from remote side
- Minimizing overhead

With this version of the reference design, due to the first condition, we can only maximize the speed on the Xilinx → Lattice pipe : we need to enable only the PDU generator. Therefore we will focus on this link direction, however, note that it would be similar in the other direction.

By enabling only the PDU generation on the Virtex-7 board we get this:

hw_vio_1

Name	Value	Activity	Direction	VIO
reset_cnt	0		Output	hw_vio_1
tx_pdu_enable	1		Output	hw_vio_1
tx_pdu_underflow	0		Output	hw_vio_1
tx_ufc_enable	0		Output	hw_vio_1
tx_nfc_enable	0		Output	hw_vio_1
tx_pdu_endless	0		Output	hw_vio_1
tx_pdu_fixed_size_on	0		Output	hw_vio_1
tx_pdu_fixed_size_max[31:0]	[H] 0004_E200		Output	hw_vio_1
tx_pdu_fixed_size_min[31:0]	[H] 0004_E200		Output	hw_vio_1
tx_pdu_fixed_size_incr[31:0]	[H] 0000_0000		Output	hw_vio_1
tx_pdu_fixed_size_ifgap[31:0]	[H] 0000_9C40		Output	hw_vio_1
tx_pdu_fixed_cut_size[31:0]	[H] 0000_0320		Output	hw_vio_1
tx_pdu_fixed_cut_gap[31:0]	[H] 0000_004B		Output	hw_vio_1
cnt_pdu_frame_tx	[H] 8D23_E2F6		Input	hw_vio_1
cnt_pdu_frame_rx	[H] 130A_2635		Input	hw_vio_1
cnt_pdu_frame_err	[H] 0000_0000		Input	hw_vio_1
cnt_ufc_frame_tx	[H] 29C5_421B		Input	hw_vio_1
cnt_ufc_frame_rx	[H] C7A3_3C5E		Input	hw_vio_1
cnt_ufc_frame_err	[H] 0000_0000		Input	hw_vio_1
tx_pdu_frame_size	[H] 0000_0000		Input	hw_vio_1
cnt_nfc_wait_tx	[H] 47A1_68EE		Input	hw_vio_1
cnt_nfc_wait_rx	[U] 4056011399		Input	hw_vio_1
user_clk_freq	[U] 250006975		Input	hw_vio_1
tx_pdu_rate	[U] 201		Input	hw_vio_1
rx_pdu_rate	[U] 283		Input	hw_vio_1

Figure 17: VIO with PDU Generator only

Whoops, we notice that the Lattice board is sending PDU frames quicker than the VC707 (283 Mbytes/s > 201 MBytes/s) whereas all its generators are still active.. How is this possible?!

Because of the NFC interface! Remember, when an **NFC request is received** by the Aurora, it is processed **internally** and **pauses the transmitter** for a specific number of clock cycles.

If we re-enable the NFC sending, the *rx_pdu_rate* drops to 190 Mbytes/s. However it does not increase the TX rate – and has no reason to.
Actually the NFC generator does not generate a lot of transaction, and enabling it reduces the Virtex-7 sending rate by only a few bits/s (rate displayed is still 201 Mbytes/s).

Let us **disable it again** at it does not help up to reach our goal here!

Let us remember the two things we still need to do : disabling the “NFC effect” and minimizing frame the overhead. We are actually gonna solve both issue with one click, using a settings of the Auroras that we did not discuss yet : *completion* or *immediate NFC*.

If you go back to Xilinx Aurora IP Configuration you may notice that while configuring the aurora we chose *UFC + Completion NFC* (instead *UFC + Immediate NFC*). In this configuration, the NFC pauses are applied between packets, meaning after an EOP is received on the transmit PDU interface. It is opposed with the *immediate* behavior that pauses the frame sending even in the middle of a frame and resumes later once the pause is done.

We have a generation option that fits perfectly the request: when *tx_pdu_endless* we emulate the streaming mode and no longer generate any EOP! Doing so, NFC received request are never applied! Moreover, it minimizes the overhead due to Start and End of packet signaling of the protocol :

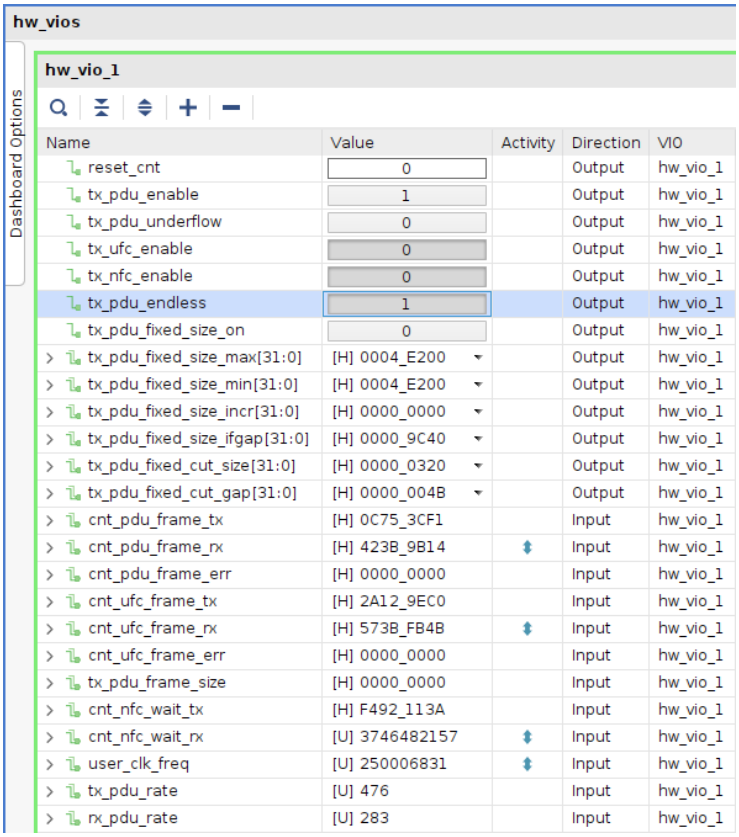


Figure 18: VIO Maximum Rate

link_up	1
cnt_pdu_frame_tx	79841A2F
cnt_pdu_frame_rx	0C753CF1
cnt_pdu_frame_err	00000000
cnt_ufc_frame_tx	67764858
cnt_ufc_frame_rx	2A129EC0
cnt_ufc_frame_err	00000000
cnt_nfc_wait_tx	FA273406
cnt_nfc_wait_rx	F492113A
tx_clkout_freq	250000763
rx_clkout_freq	250007778
rx_pdu_rate	476
tx_pdu_rate	283

Figure 19: Reveal Maximum Rate

We reach 476 Mbytes/s, so 3,8 Gbit/s : quite close to the maximum theoretical limit which is 4 Gbit/s!

Note: Funny enough, there is still another type of frames circulating on the bus, Clock Compensation frames. However, Xilinx 8B/10B IP looses link when they are not present, whereas they should be use only when we want to perform clock compensation.. If we were using two auroras, and under correct circumstances we could disable them and earn a few more bit/s!

6.5. Testing Procedure: Testing hot-plug

It is always interesting to test the hot-plug robustness of transceiver protocols. In our case, both Xilinx and ALSE Aurora 8b10b IPs are robust to it.

You can connect and disconnect the SFP connectors as many times as you want, the link-up should re-establish once cables are properly connected. Both Reference Designs are built to reset after a loss of link, staying error free.

If you set Reveal trigger to trig on the link loss as in the capture below, you may see a few data errors caused by the disconnection.

Link disconnection can be tested as well by resetting one of the FPGA with user push-buttons, starting over the link-up process.

7. Conclusion

We have verified that the Avant-G transceivers (and Avant-X, remember they are identical !) **are performing optimally in our application.**

As often with Lattice, the challenge was more in accessing the actual information and working around a few unexpected and undocumented obstacles (one being JP10, JP11 and JP12 effect on RC38612A configuration) than in realizing the design in itself. Despite its length, the document hides several holes in the documentation, particularly for the MPPHY !

However the **goal is reached**: with this application note, we used both PMA and PCS layers of the SerDes IP with success, **validating our Lattice version of our Aurora IP and the Lattice transceiver IP for 8B/10B usage at 5 Gbit/s** !

We have actually tested successfully this transceiver at much for higher-speeds : **we ported our Aurora 64B/66B** on this same board with success, using only the PMA this time performing the 64B/66B encoding and decoding with our own soft PCS IP.

Hopefully you will have learned and maybe experimented something useful reading this document and playing with our Reference Design.

If you are interested in the IP do not hesitate to contact ALSE.
Aurora 8B/10B is only one of our many IPs and it is available on most of the FPGAs and boards provided they offer transceivers !

8. Bibliography

- **Aurora 8b10b Protocol** https://docs.amd.com/v/u/en-US/aurora_8b10b_protocol_spec_sp002
- **Lattice Avant Platform - Specifications Data Sheet** https://www.latticesemi.com/view_document?document_id=53700
- **Lattice Avant-G/X MPPHY Module User Guide** https://www.latticesemi.com/view_document?document_id=54139
- **Lattice Avant SerDes/PCS User Guide** https://www.latticesemi.com/view_document?document_id=54117
- **Avant-G Versa Board - User Guide** https://www.latticesemi.com/view_document?document_id=54169
- **VC707 Evaluation Board for the Virtex-7 FPGA** https://docs.amd.com/v/u/en-US/ug885_VC707_Eval_Bd
- **RC38612A Datasheet** https://www.renesas.com/en/document/dst/rc38612-datasheet?r=1506416
- **Reveal User Guide** https://www.latticesemi.com/~/media/328D471BF2C74EB1907832FAA6FB344B.ashx
- **Virtual Input/Output – LogiCORE IP Product Guide** https://docs.amd.com/v/u/en-US/pg159-vio

APPENDIX A
Programming the RC38612A Clock Generator

For our Reference Design, we need the Avant-G transceiver Quad 6 Reference Clock port to receive a 156.25 MHz LVDS clock (namely *MPQ6_REFCLK_P/N*).

This FPGA pin is sourced by the clock output **Q1** of the **RC38612A**.
By default, according to the **Avant-G/X Versa Board** documentation, this output should be active and running at 156.25 MHz.

The first undocumented element about this external PLL is the effect of jumpers J10, J11 and J12 on it : if they are **CLOSED**, the PLL **will not be able** to read its configuration E2PROM (U33, 24FC1025T from Microchip). Without this access, the output clocks will not be enabled.
However, these three jumpers are required for a user to access the I²C bus of the PLL.

The correctness of the Clock generator configuration can be checked using the first step of our design, and checking the frequencies of clocks generated by the transceivers (see Testing Procedure: Setup the boards). If you do not have clocks (null frequency) or if frequencies are not those expected, you need to reprogram the E2PROM before running tests and make sure the Jumpers are set as described here !

1. Prepare your PC

Before programming the board it is a good idea to check your default configuration! If your board is not affected by the problem and enjoy a valid default configuration, you will have nothing to do and will even be able to power-cycle the board and test at will !

- Download and install the Renesas Timing Commander software on a Windows PC. You will need access to the content of *fit/lattice/RC386128A* folder from this PC for an easy configuration.
- Close jumpers J10, J11 and J12 on the Versa board to connect the RC38612A I2C port to the FTDI chip.

2. Check the RC38612A current configuration

► Start by clicking on  in the top-right corner of the window.

► In the prompt windows, set the I2C board address to 0x**B0** :

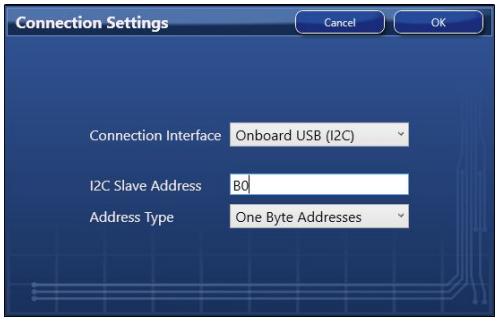


Figure 20: Set I2C address

► Click on the  icon to connect to the board through the FTDI. In the top-right corner you should now have  .

The tool may ask you to update the *Firmware Version* of the chip on the board.
IF this occurs, click on *Firmware Utility* and *Update RAM to Current Version Only.*

► Now, click on the  utility to read all the register values from the clock generator. If you see the output clock **Q1 enabled** and running at **156.25 MHz**, you can run the reference design and forget about this Appendix.

Otherwise, close the firmware and follow next steps to get a valid clock output.

3. Program RC38612A with the custom configuration (optional)

You can follow these steps for testing, however it is better to program the E2PROM to get a static configuration that will stay after a power-cycle.

► Open the firmware and click *Open Settings File*. Select provided configuration files and click *Open*:

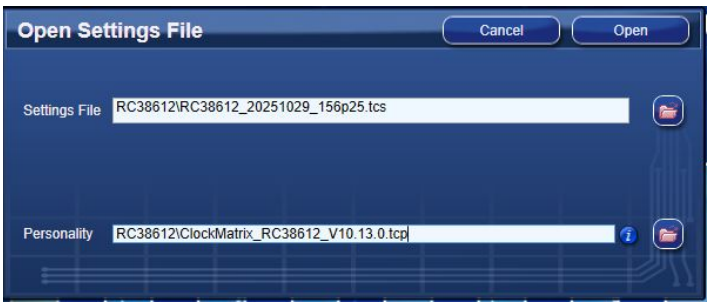


Figure 21: Load configuration file

► After loading (can take up to a minute), you should now see the configuration that we are going to apply to the clock generator : **clock output Q1** should be **enabled** and configured to run at **156.25 MHz**:

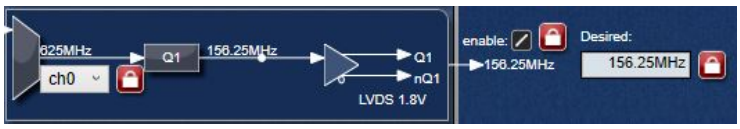


Figure 22: Valid clock output configuration

► Click on  to connect to the chip. The tool may ask you to update the *Firmware Version* of the chip on the board. IF this occur, click on *Firmware Utility* and *Update RAM to Current Version Only.*

► Click on  to write the configuration into the chip. A valid clock should now be available for our SerDes.

Done. **After a power-cycle, the board gets back in its previous step !**

4. Program the configuration E²PROM

On the same I²C bus as the clock generator we find a small (1 Mbits = 128k bytes) E²PROM memory : 24FC1025T from Microchip. As mentioned before, this memory is used by the Clock Generator at power up to get its configuration.

We explain here how to update this memory with the proper settings (to produce the 156.25 MHz clock needed by the transceiver).

- Start by disconnecting from the chip by clicking on  in the top right corner if you are connected to the Clock Generator.
- Click on *EEPROM Utility* and select the correct type of E²PROM in the prompt window: *24x1025 (@ I2C 0x50, 0x54)*.
- Now click on *Write Config to E2PROM* and select the *ClockMatrix_EEPROM_156p25.hex* file provided by ALSE. It contains the same configuration as the **.tcs/tcp* files but in hexadecimal format, suited for EEPROM configuration.
- Write the EEPROM : at next power-cycle, if the jumpers **J10, J11 and J12 are OPEN** (they should currently be closed) the PLL will be correctly configured.
- You can check that the E2PROM was correctly configured by connecting to the chip one-time, (procedure described previously) and pressing the SW6 switch *PLL_RSTn*.
  It should reset the PLL and start the reconfiguration through the E²PROM.
- Now read the chip registers and check that the 156.25 MHz clock is generated correctly.

Note: if you do not connect one time before to the chip, the clock generator will not try to configure through the I2C board, due to a conflict with the FTDI chip. Connecting it one time seems to solve the issue.
In practice, removing the three I2C jumpers solves this issue !