UniBoard XAUI firmware module

	Organisatie / Organization	Datum / Date
Auteur(s) / Author(s): Daniel van der Schuur	ASTRON	11 September 2012
Controle / Checked: Eric Kooistra	ASTRON	
Goedkeuring / Approval: Andre Gunst	ASTRON	
Autorisatie / Authorisation: **Handtekening / Signature** Andre Gunst	ASTRON	

Distribution list:

Group:	Others:
Andre Gunst Eric Kooistra Harm-Jan Pepping	Gijs Schoonderbeek Sjouke Zwier Harro Verkouter (JIVE) Jonathan Hargreaves (JIVE) Salvatore Pirruccio (JIVE)

Document history:

Revision	Date	Author	Modification / Change
0.1	2012-09-11	Daniel van der Schuur	Draft

Table of contents:

Terminology:

BN	Back Node
BN_BI	Back node – Backplane Interface
CMU	Clock Multiplier Unit
FN	Front Node
FN_BN	Front Node – Back Node
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
Nof	Number of
PCS	Physical Coding Sublayer
PHY	Physical layer
PMA	Physical Media Attachment
RX	Receive
SI_FN	Serial Interface – Front Node
SOPC	System On a Programmable Chip (Altera)
TR	Transceiver
TX	Transmit
XGB	10 Gigabit Breakout board
XGMII	10 Gigabit Media Independent Interface
XAUI	10 Gigabit Attachment Unit Interface

References:

1. 'DP Streaming Module Description', ASTRON-RP-382, Eric Kooistra
2. $UNB/Firmware/modules/diagnostics
3. https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk, the UniBoard FP7 SVN repository ($UNB)
4. 'Firmwarespecificatie voor 10G XAUI en 10GbE', ASTRON-SP-048, Eric Kooistra
5. "DIAG module description", ASTRON-RP-1313, Harm Jan Pepping
6. $UNB/Software/python/README.txt

1 Introduction

1.1 Purpose

The tr_xaui module provides the user with up to 3 duplex XAUI links with an equal number of 64-bit streaming [1] TX inputs and RX outputs on the user side. In addition, an MDIO PHY can be instantiated for each XAUI core, in case the XAUI cores interface to MDIO-compliant PHY chips – such as the Vitesse transceiver chips on the front-node side of the UniBoard.
An internal mms_diagnostics instance is used to allow the user to send diagnostics data across the XAUI links instead of user data. This mms_diagnostics instance is controlled with an MM interface.

1.2 Limitations

The number of XAUI PHY cores is limited to 3 per FPGA and can only be used on the front SI_FN interfaces and the back BN_BI interfaces.

1.2.1 Only SI_FN and BN_BI

The FN_BN interface cannot be used because only 3 out of 4 lanes per bundle in the UniBoard mesh (FN_BN) connect to hard-PCS transceivers. Altera's XAUI IP currently only supports 4 hard-PCS transceivers to be combined into one XAUI PHY. Also, one cannot combine randomly chosen hard-PCS transceivers into one XAUI PHY; additional limitations exist with respect to the transceiver blocks the hard-PCS transceivers belong to.

1.2.2 Soft-PCS XAUI IP

A quarter of the UniBoard FPGA transceivers lack a hard PCS. The VHDL supports the instantiation of a fourth, soft-PCS XAUI core that might work with the fourth, PMA-only (no PCS) transceiver bundles, such as [SI_FN_3_0 .. SI_FN_3_3] and [BN_BI_3_0 .. BN_BI_3_3]. This however has not been tested.

1.3 Module overview

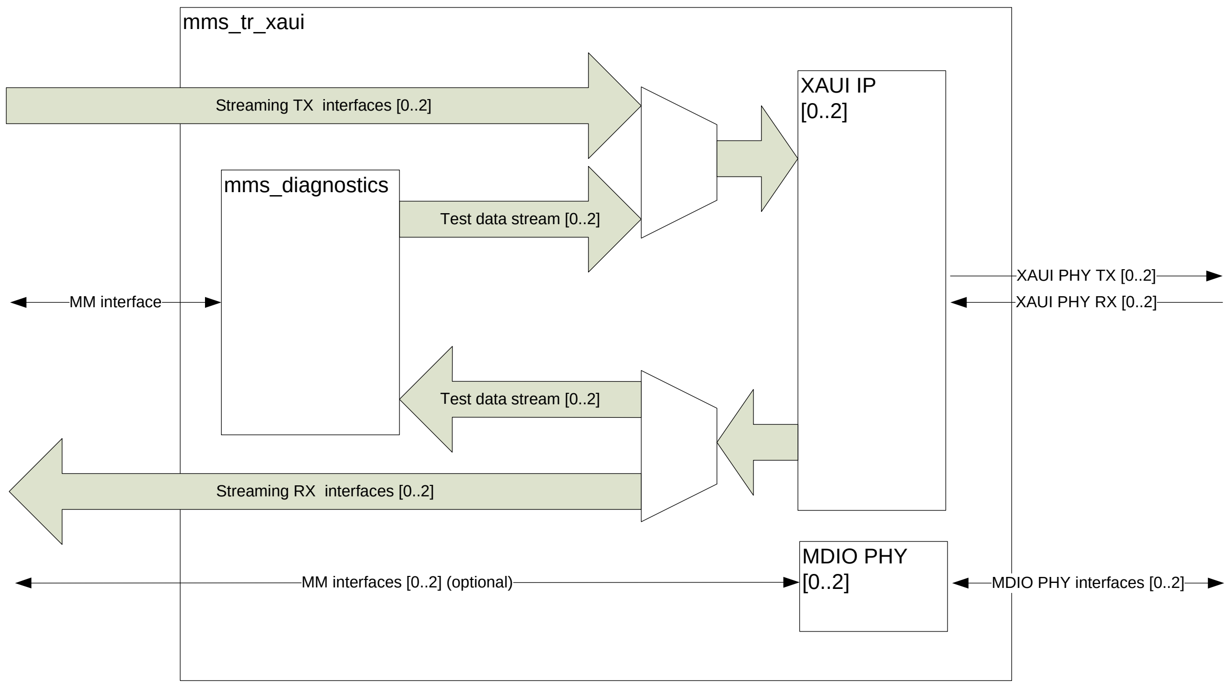


Figure 1 – mms_tr_xaui overview

Figure 1 shows the top-level of mms_tr_xaui and its main components. The XAUI data paths are self-regulating, i.e. there is no explicit control and/or monitoring required other than the flow control and status signals within the streaming interfaces. The MDIO cores, re-used from LOFAR, are controlled automatically (by VHDL) by an internal MDIO master, or externally via an optional array of MM interfaces.
The instantiated XAUI IP cores are generated by Altera's MegaWizard, but could be replaced with IP from any vendor.

2 Hardware interface

2.1 Clock and reset signals

Figure 2 shows the clock and reset signals that exist in the tr_xaui module. The mm_rst signal is user provided, the rx_rst and tx_rst signals are generated within mms_tr_xaui.

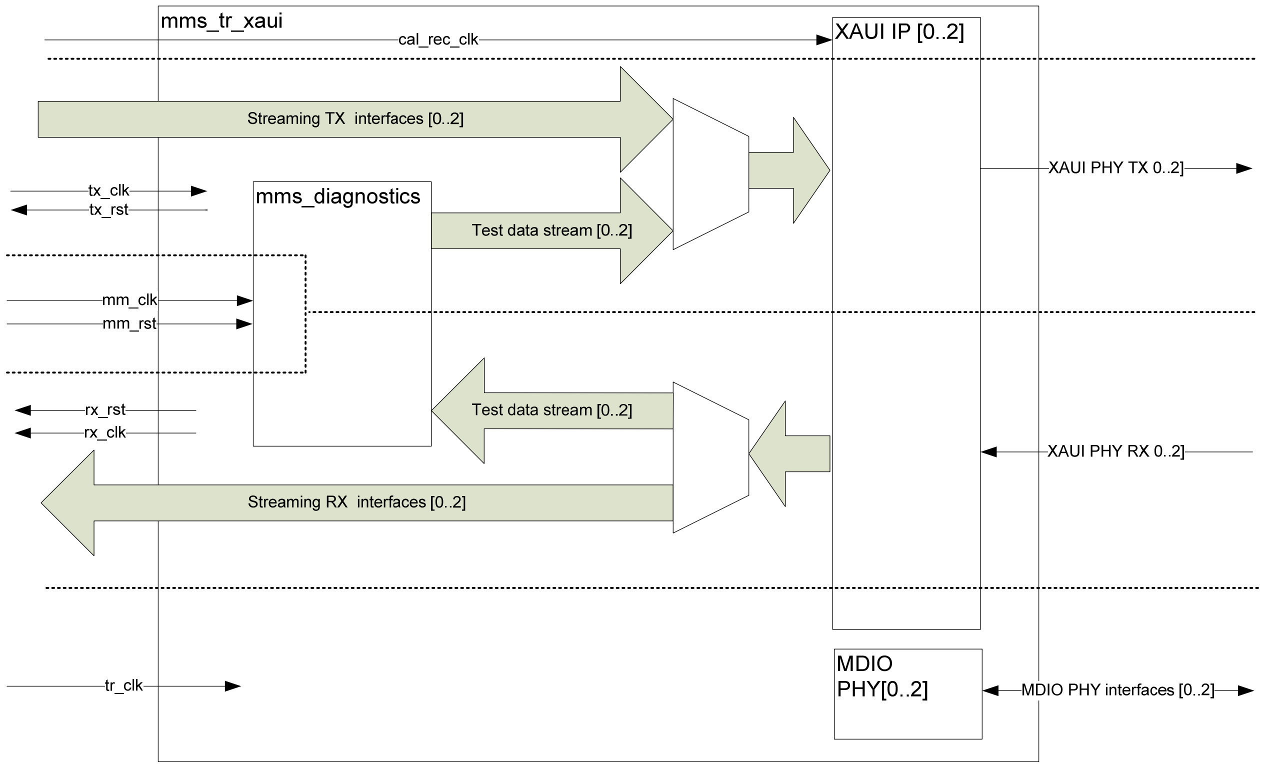


Figure 2 – mms_tr_xaui clock clock and reset signals

Table 1 summarizes the clock signals for tr_xaui.

Name	Frequency (MHz)	Description
tr_clk	156.25	Reference clock for XAUI PHY and MDIO PHY
tx_clk	156.25	User-provided clock to clock in parallel TX data
rx_clk	156.25	Generated clock to clock out parallel RX data
mm_clk	50	MM clock for the memory-mapped bus shared with e.g. a NIOS II processor
cal_reconf_clk	37,5 - 50	Used to clock the calibration block inside the XAUI PHY components. The frequency must be between 37.5 and 50MHz.

Table 1: tr_xaui clock signals

2.2 Parameters

Available parameters when instantiating the tr_xaui module are listed in Table 2.

Generic	Type	Description
g_nof_xaui	NATURAL	The number of XAUI cores / interfaces
g_mdio	BOOLEAN	TRUE instantiates an MDIO core for each XAUI PHY
g_mdio_mm_ctrl	BOOLEAN	TRUE enables external MM control of the MDIO cores (only when g_mdio=TRUE); prevents the default auto MDIO master mdio_ctrl.vhd from being instantiated.

Table 2: tr_xaui parameters

2.3 Interface signals

The interface signals of tr_xaui are listed in Table 3.

Interface	Type	Description
tx_sosi_arr	t_dp_sosi_arr	Array (g_nof_gx-1..0) of streaming TX data ports, source to sink (tr_xaui module). Data width = 64.
tx_siso_arr	t_dp_siso_arr	Array (g_nof_gx-1..0) of flow control signals from tr_xaui TX ports, sink to source.
rx_sosi_arr	t_dp_sosi_arr	Array (g_nof_gx-1..0) streaming RX data ports, source (tr_xaui module) to sink.
rx_siso_arr	t_dp_siso_arr	Array (g_nof_gx-1..0) flow control signals to tr_xaui RX ports, sink to source. Data width = 64.
xaui_rx	t_xaui_arr	Array of g_nof_xaui times 4 XAUI serial receiver lanes
xaui_tx	t_xaui_arr	Array of g_nof_xaui times 4 XAUI serial transmitter lanes
diagnostics_mosi	t_mem_mosi	MOSI interfaces to the mms_diagnostics instance
diagnostics_miso	t_mem_miso	MISO interfaces from the mms_diagnostics instance
mdio_mosi_arr	t_mem_mosi_arr	Optional (if g_mdio_mm_ctrl=TRUE) array of g_nof_xaui MOSI interfaces to the MDIO cores.
mdio_miso_arr	t_mem_miso_arr	Optional (if g_mdio_mm_ctrl=TRUE) array of g_nof_xaui MISO interfaces from the MDIO cores
mdio_mdc	STD_LOGIC_VECTOR	SL array of g_nof_xaui Management Data Clock outputs
mdio_mdat_in	STD_LOGIC_VECTOR	SL array of g_nof_xaui Management Data inputs
mdio_mdat_out	STD_LOGIC_VECTOR	SL array of g_nof_xaui Management Data outputs

Table 3: interface signals

3 Software interface

An mms_tr_xaui instance provides MM buses to control its internal mms_diagnostics module, and allows the user to manually control the internal MDIO cores.

3.1 mms_diagnostics

[4] lists the register layout and contents of mms_diagnostics. Generic g_nof_streams equals the number of instantiated XAUI modules: g_nof_xaui.

3.2 MDIO

The MM registers available for optional manual control of the MDIO instances are listed in Table 4. The mdio_phy_reg provides an MM interface for the mdio_phy module and, when instantiated by setting generic g_mdio_mm_ctrl to TRUE, prevents instantiation of the default MDIO controller (mdio_phy_ctrl.vhd) which auto executes a predefined list of MDIO commands.

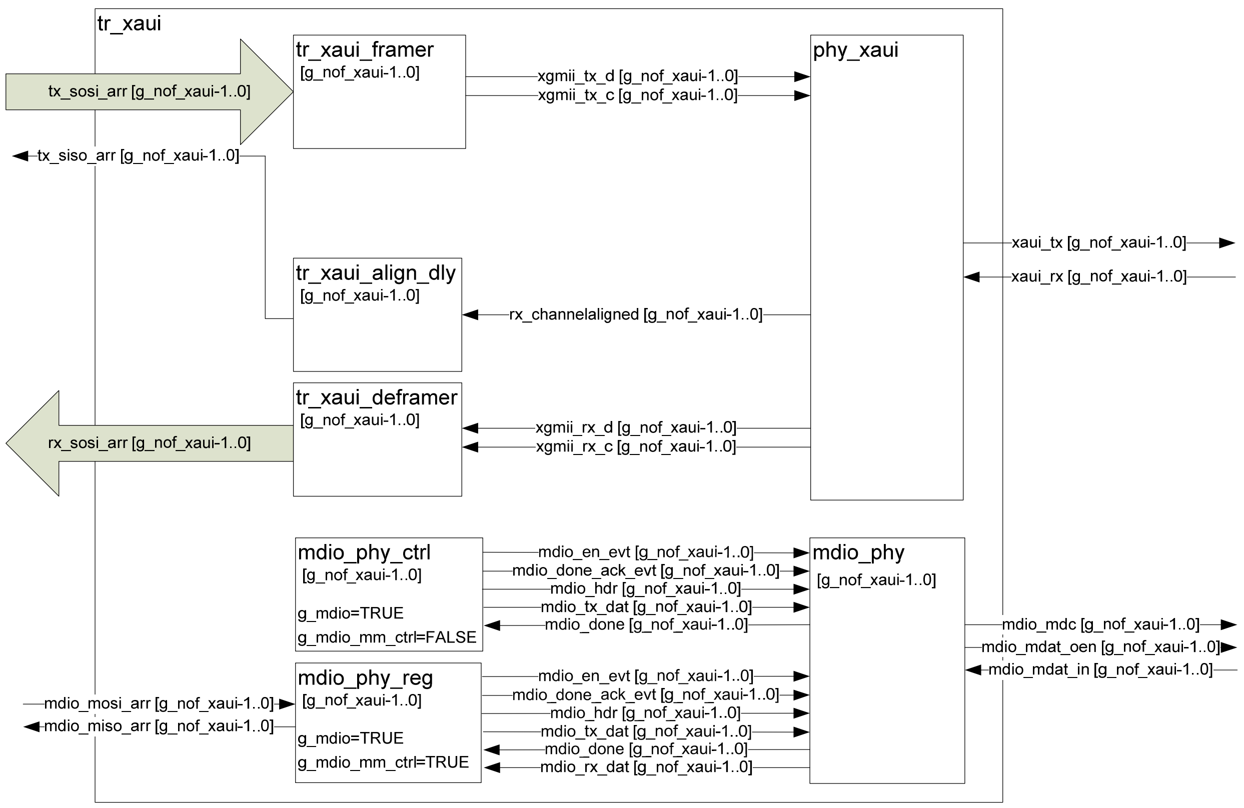
Name	Address (words)	Size (words)	Read/ Write	Description
HDR	0	1	W	TX header
EN_EVT	0	1	W	Implicit write enable when TX header is written.
TX_DAT	1	1	W	TX data
RX_DAT	2	1	R	RX data
DONE	3	1	R	Done
DONE_ACK_EVT	4	1	W	Done acknowledge

Table 4: mdio_phy_reg

More information on MDIO access sequences can be found in paragraph 5.3.

4 Design

This chapter describes the structure of tr_xaui (Figure 3). As mms_tr_xaui only instantiates the mms_diagnostics in addition to tr_xaui, this chapter focusses on the tr_xaui entity only.
The following chapter covers the components mentioned here in more detail.



tr_xaui
tx_sosi_arr [g_nof_xaui-1..0]
tr_xaui_framer
[g_nof_xaui-1..0]
xgmii_tx_d [g_nof_xaui-1..0]
xgmii_tx_c [g_nof_xaui-1..0]
phy_xaui
tx_siso_arr [g_nof_xaui-1..0]
xaui_tx [g_nof_xaui-1..0]
xaui_rx [g_nof_xaui-1..0]
tr_xaui_align_dly
[g_nof_xaui-1..0]
rx_channelaligned [g_nof_xaui-1..0]
tr_xaui_deframer
[g_nof_xaui-1..0]
rx_sosi_arr [g_nof_xaui-1..0]
xgmii_rx_d [g_nof_xaui-1..0]
xgmii_rx_c [g_nof_xaui-1..0]
mdio_phy_ctrl
[g_nof_xaui-1..0]
g_mdio=TRUE
g_mdio_mm_ctrl=FALSE
mdio_en_evt [g_nof_xaui-1..0]
mdio_done_ack_evt [g_nof_xaui-1..0]
mdio_hdr [g_nof_xaui-1..0]
mdio_tx_dat [g_nof_xaui-1..0]
mdio_done [g_nof_xaui-1..0]
mdio_phy
[g_nof_xaui-1..0]
mdio_mdc [g_nof_xaui-1..0]
mdio_mdat_oen [g_nof_xaui-1..0]
mdio_mdat_in [g_nof_xaui-1..0]
mdio_phy_reg
[g_nof_xaui-1..0]
g_mdio=TRUE
g_mdio_mm_ctrl=TRUE
mdio_mosi_arr [g_nof_xaui-1..0]
mdio_miso_arr [g_nof_xaui-1..0]
mdio_en_evt [g_nof_xaui-1..0]
mdio_done_ack_evt [g_nof_xaui-1..0]
mdio_hdr [g_nof_xaui-1..0]
mdio_tx_dat [g_nof_xaui-1..0]
mdio_done [g_nof_xaui-1..0]
mdio_rx_dat [g_nof_xaui-1..0]

Figure 3 - tr_xaui design

4.1 XAUI IP and XGMII

The XAUI IP cores are located in phy_xaui. These XAUI IP cores accept TX data and control signals in the form of a 72-bit wide XGMII interface – 64 bits of data and 8 control bits. The tr_xaui_framer converts its streaming input data to separate XGMII data and control signals. In addition, it adds the required START, TERMINATE and IDLE control words that the XAUI IP cores require. The XAUI cores transmit the START and TERMINATE words as they are, but converts the IDLE words to XAUI-compliant control characters to manage the XAUI bus in terms of e.g. XAUI lane alignment.

4.2 RX channel alignment

A status signal emerging from each XAUI core, rx_channelaligned, indicates whether the incoming RX data is properly locked on to. This signal is delayed in tr_xaui_dly by roughly one second to indicate when a connecting receiver can be assumed to be rx_channelaligned as well. This delayed signal is fed back to the TX stream as XON and overrides the READY TX flow control signal until high.

4.3 MDIO

As mentioned in the previous chapters, MDIO PHY cores are instantiated if the user sets g_mdio to TRUE. If this is the case, automatic MDIO controllers, mdio_phy_ctlr instances, are used by default. After powerup, these auto execute a list of MDIO commands passed as a generic to initialize MDIO devices such as the Vitesse chips used on the UniBoard. Alternatively, the user can decide to instantiate manually MM controllable mdio_phy_reg instances instead of mdio_phy_ctrl.

5 Implementation

5.1 tr_xaui_framer

The tr_xaui_framer component passes the incoming streaming data to its XGMII data output, but inserts an XGMII START word (**S**= 0x00000000000000FB) to the output XGMII data at the transition from invalid input data to valid input data, where 0xFB stands for Frame Begin. At the transition from valid to invalid, it adds an XGMII TERMINATE word (**T**=0x07070707FD000000), where 0xFD stands for Frame Delimiter. If the streaming input remains invalid for longer than one cycle, its puts IDLE words (**I**=0x0x070707070707070) on the XGMII data bus.

In parallel to the assignment of XGMII data words, XGMII control bits are put on the xgmii_tx_c bus. For each of the 8 bytes on the XGMII data bus, an XGMII control bit indicates whether or not the corresponding byte should be interpreted as a control byte. For instance, in parallel to the 64-bit XGMII START word 0x00000000000000FB, control bits b'00000001 indicate that only byte 0 is a control byte.

As START and TERMINATE words need to be inserted, gaps in the TX input stream are necessary. In addition to this, XAUI packets require a minimum inter packet gap that allow XAUI control characters to be sent in between packets to maintain the XAUI link. To provide TX data in a user selectable data-to-gap ratio, the input TX stream is fed into a dp_gap instance first. An FSM uses these invalid cycles to put the correct XGMII data words and control bits on the output buses. The following table illustrates how this is performed.

gap_sosi	G	G	D0	D1	D2	D3	G	G	G	G	G	D4	D5	D6
prev_gap_sosi	G	G	G	D0	D1	D2	D3	G	G	G	G	G	D4	D5
nxt_tx_data	**I**	**I**	**S**	D0	D1	D2	D3	**T**	**I**	**I**	**T**	**S**	D4	D5
state	G	G	G	D	D	D	D	G	G	G	G	G	D	D

Table 5: tr_xaui_framer FSM replacing gaps with XGMII words

The FSM has only two states: Gap or Data. Registered output nxt_tx_data either equals the registered input prev_gap_sosi, an IDLE word or a START or TERMINATE word based on the transition between the two states.

5.2 tr_xaui_deframer

In addition to performing the reverse of tr_xaui_framer, the tr_xaui_deframer compensates for the misalignment that might occur on the RX side. The XAUI word boundary is 32 bits, which means that even though the TX inputs and RX outputs are 64 bits wide, it is processed by the XAUI IP core as two interleaved 32-bit XAUI interfaces. This in turn means that, depending on the moment that the RX locks to the incoming signal and starts de-interleaving the 32-bit bus into one 64-bit bus at half the speed, the 32-bit LS word in the 64-bit received word might have been transmitted as the 32-bit MS word. The possibility of the 32-bit word boundary being incorrect is accounted for and corrected in tr_xaui_deframer, ensuring that the 64 data bits emerge as RX stream exactly as they have been input as TX stream. The following tables show how this correction operates. First, Table 6 shows the simple FSM function when misalignment is not accounted for, meaning nothing more than replacing IDLE, START and TERMINATE cycles with gaps (de-asserting rx_sosi.valid).

rx_data	I	I	S	D0	D1	D2	D3	T	I	I	T	S	D4	D5
nxt_rx_sosi	**G**	**G**	**G**	D0	D1	D2	D3	**G**	**G**	**G**	**G**	**G**	D4	D5
state	G	G	G	D	D	D	D	G	G	G	G	G	D	D

Table 6: tr_xaui_deframer FSM, not accounting for word misalignment

Table 7 shows the situation where the RX data is interpreted respecting its 32-bit word boundary, and the alignment happens to be correct: on a START, the LS RX data word matches the LS portion of the START word (0x000000FB).

rx_data_hi	I_hi	I_hi	**S_hi**	D0_hi	D1_hi	D2_hi	D3_hi	T_hi	I_hi	I_hi	T_hi	S_hi	D4_hi
rx_data_lo	I_lo	I_lo	**S_lo**	D0_lo	D1_lo	D2_lo	D3_lo	T_lo	I_lo	I_lo	T_lo	S_lo	D4_lo
nxt_rx_sosi_hi	**G**	**G**	**G**	D0_hi	D1_hi	D2_hi	D3_hi	**G**	**G**	**G**	**G**	**G**	D4_hi
nxt_rx_sosi_lo	**G**	**G**	**G**	D0_lo	D1_lo	D2_lo	D3_lo	**G**	**G**	**G**	**G**	**G**	D4_lo
state	G	G	G	D	D	D	D	D	G	G	G	G	D

Table 7: tr_xaui_deframer FSM, RX data aligned

Table 8 show the necessary steps when the incoming RX data is misaligned. Misalignment is determined when the LS portion of the START word occurs on the MS RX data word. As the desired high and low data words are not present at the same cycle, the received high portion is buffered (prev_rx_data_hi) so it can be assigned to the low portion of nxt_rx_sosi along with the unbuffered rx_data_lo which will be assigned to the high portion of nxt_rx_sosi.

rx_data_hi	I_lo	**S_lo**	D0_lo	D1_lo	D2_lo	D3_lo	T_lo	I_lo	I_lo	T_lo	S_lo	D4_lo
rx_data_lo	I_hi	I_hi	**S_hi**	D0_hi	D1_hi	D2_hi	D3_hi	T_hi	I_hi	I_hi	T_hi	S_hi
prev_rx_data_hi		I_lo	S_lo	D0_lo	D1_lo	D2_lo	D3_lo	T_lo	I_lo	I_lo	T_lo	S_lo
nxt_rx_sosi_hi	**G**	**G**	**G**	D0_hi	D1_hi	D2_hi	D3_hi	**G**	**G**	**G**	**G**	**G**
nxt_rx_sosi_lo	**G**	**G**	**G**	D0_lo	D1_lo	D2_lo	D3_lo	**G**	**G**	**G**	**G**	**G**
state	G	G	D	D	D	D	D	G	G	G	G	G

Table 8: tr_xaui_deframer FSM, RX data misaligned

5.3 mdio_phy_ctlr

This MDIO PHY controller takes an MDIO command list (t_mdio_cmd_arr) as generic parameter and auto executes all of these commands in sequence after powerup. This MDIO command array is an array of t_mdio_cmd, a record type consisting of the following MDIO access parameters:

- Write or read;
- Device address;
- Device register;
- Write data.

6 Reference design

A reference design, unb_tr_xaui, instantiates mms_tr_xaui along with yet another mms_diagnostics module (besides the mms_diagnostics module inside mms_tr_xaui). This second mms_diagnostics module emulates an external user data stream, whereas the internal mms_diagnostics is able to override this user data stream with a test data stream that can be verified independently from the user stream.

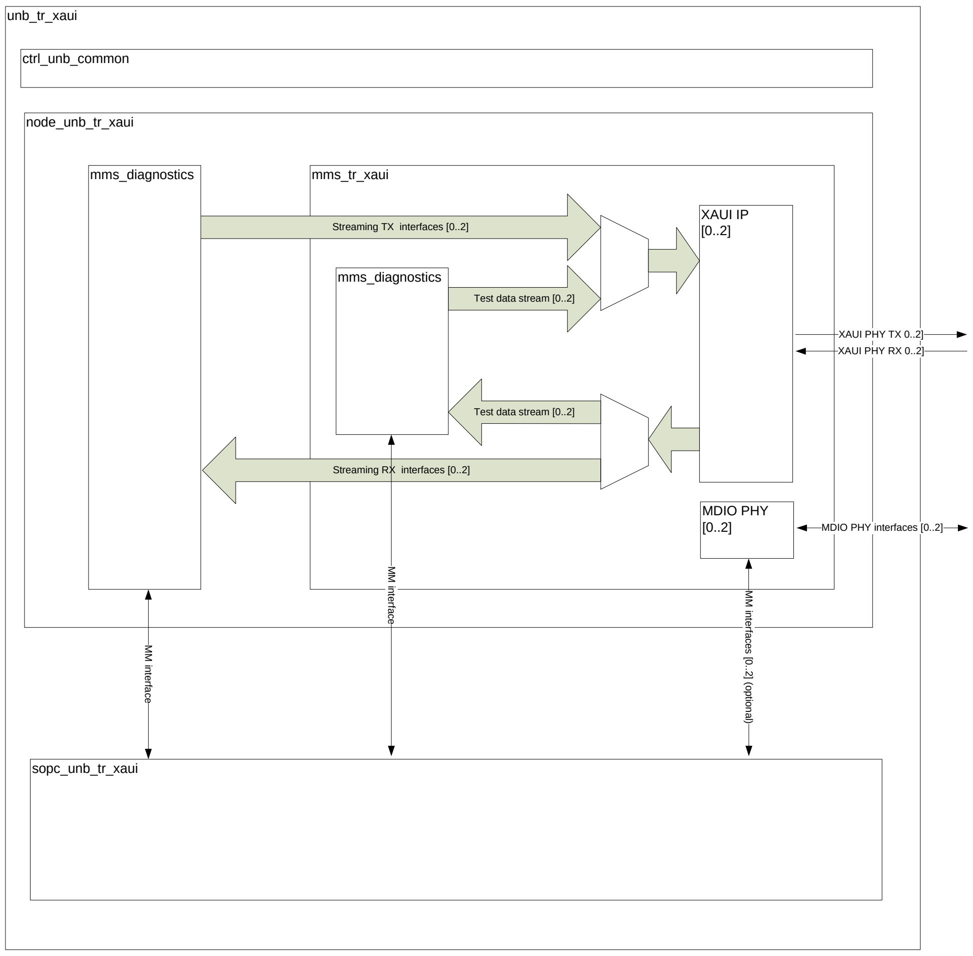


Figure 4 - design unb_tr_xaui

The second instance of mms_diagnostics operates in exactly the same way as the internal instance, with the exception that the internal one, when enabled, overrides the external diagnostics stream.

The NIOS II processor in the SOPC can run a dedicated C application, see Chapter 9, or unb_osy to allow Python programs to target the design.

There are three revisions of this design, which are discussed in Chapter 8.

7 Verification

7.1 tb_phy_xaui

This test bench provides very basic, static XGMII data and control signals to the XAUI IP under test, and demonstrates the need for a sequence of IDLE words to be fed to the TX side before switching to user data. It also demonstrates lucky alignment or unlucky misalignment, depending on the number of cycles after which the XAUI IP is released from reset.

7.2 tb_tr_xaui_framer and tb_tr_xaui_deframer

Tb_tr_framer only feeds a generated, gapped DP stream to the framer after which the resulting XGMII output data and control bits can be observed in the wave window.
Tb_dp_deframer add a deframer to the above setup, and a verification process monitoring the resulting framed->deframed RX data stream. In addition, misalignment is introduced deliberately by swapping the LS and MS portions of the XGMII framed data (as described in Chapter 5) before feeding them to the deframer that takes care of re-alignment.

7.3 tb_tr_xaui

This test bench instantiates the higher level tr_xaui and interfaces it to a diagnostics module. This is not an MMS slave, so its control signals are asserted directly (non-MM) by the test bench.

7.4 tb_node_unb_xaui

The node function of design unb_tr_xaui is looped back on serial interface level and tested. The external (user) stream is enabled and monitored and a simulation model of an MDIO slave (mmd_slave) is written to and read back via the MM interface.

7.5 tb_unb_tr_xaui

This test bench instantiates the full design, including the SOPC with its NIOS and can run the dedicated C application.

7.6 tb_mmf_node_unb_xaui

This test bench instantiates a (variable) number of daisy-chained node_unb_tr_xaui instances, but uses *mm_file* instances to interface MM buses to files that can be accessed by Python. This allows the user to run the same test cases and utility Python programs for both verification and validation.

8 Validation

Design unb_tr_xaui contains three revisions based on type of nodes the design can run on. These revisions are discussed in this chapter. The MM register map per revision does not change, but beware that the MDIO MM buses remain unconnected if no mdio_reg is instantiated, such as in the non-FN revisions and in the default FN revision that uses the auto executing MDIO controller instead of manual MM control.
Python program tc_tr_xaui_dgn.py is used to target these designs. In addition, the dedicated C application main.c can be used.

8.1 unb_tr_xaui – revision fn_tr_xaui

This revision can be un on the front nodes only and target the SI_FN interface to and from the Vitesse XAUI<->XFI transceiver chips. The generic set in the Quartus settings file sets g_fn to TRUE which in turn instantiates an MDIO core for each XAUI core.

8.2 unb_tr_xaui – revision unb_tr_xaui

This revision is only provided for completeness, as the mesh FN_BN interconnect on the UniBoard does not allow the 4-lane XAUI protocol to be used as only 3 out of 4 lanes are usable because of 1 of 4 transceivers lacking a PCS layer. Future updates of Quartus might allow the PMA-only transceivers to be incorporated as part of a XAUI PHY.

8.3 unb_tr_xaui – revision bn_tr_xaui

The backplane transceiver lanes (BN_BI) have the same layout as the SI_FN transceiver lanes. This revision however does not instantiate the MDIO cores as MDIO compliant chips such as the Vitesse chips on the FN side are absent. This revision can be used with a UniBoard 10Gb breakout board (XGB) and CX4 cables or in a UniBoard-APERTIF rack system, interconnecting the back nodes via its backplane (AUB).

9 List of files

9.1 Module tr_xaui

All files of the tr_xaui module are located in:

$UNB/Firmware/modules/tr_xaui

9.2 Design unb_tr_xaui

Design unb_tr_xaui can be found at:

$UNB/Firmware/designs/unb_tr_xaui

9.3 Python test case

The test case that targets design unb_tr_xaui or one of its revisions, and also targets simulation tb_mmf_node_unb_tr_xaui, is located at:

$UNB/Software/python/peripherals/tc_tr_xaui_dgn.py

The main peripheral used by this script is found in:

$UNB/Software/python/peripherals/pi_diagnostics.py

More information on targeting simulations using Python can be obtained by reading [6]:

9.4 Main.c

A dedicated C application is located at:

$UNB/Firmware/software/apps/unb_tr_xaui