ASTRON
Netherlands Institute for Radio Astronomy

UniBoard tranceiver module: tr_nonbonded

	Organisatie / Organization	Datum / Date
Auteur(s) / Author(s): Daniel van der Schuur	ASTRON	23 September 2010
Controle / Checked: Eric Kooistra	ASTRON	23 September 2010
Goedkeuring / Approval: Andre Gunst	ASTRON	20 October 2010
Autorisatie / Authorisation: **Handtekening / Signature** [illegible] Andre Gunst	ASTRON	20 October 2010

Doc.nr.:	ASTRON-RP-405
Rev.:	1.0
Date:	20-10-2010
Class.:	Public

Distribution list:

Group:	Others:
Andre Gunst Eric Kooistra Jonathan Hargreaves (JIVE) Salvatore Pirruccio (JIVE)	

Document history:

Revision	Date	Chapter / Page	Modification / Change
0.1	2010-08-19	-	Draft
0.2	2010-09-27	-	Removed section on reset sequence, added sections on clock domains and valid signals
0.3	2010-10-01	-	Removed section on FIFO valid signals
1.0	2010-10-10	-	Final

Table of contents:

List of figures:

Terminology:

BN	Back Node
BN_BI	Back node – Backplane Interface
CMU	Clock Multiplier Unit
FIFO	First In First Out
FN	Front Node
FN_BN	Front Node – Back Node
FPGA	Field Programmable Gate Array
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)
TX	Transmit

References:

1. ‘Design considerations for UniBoard’s Stratix IV Transceivers’, ASTRON-RP-386, Daniel van der Schuur
2. ‘Quartus II Handbook’, quartusii_handbook.pdf, www.altera.com
3. ‘Altera Stratix IV Device Handbook’, July 2010, , www.altera.com
4. ‘UniBoard Board Design’, ASTRON RP-316, Gijs Schoonderbeek, Sjouke Zwier

1 Introduction

The UniBoard uses high speed Stratix IV transceivers to interface between FPGAs using the mesh (front node - back node or FN_BN) interconnect, and to interface to hardware via the back node to backplane interface (BN_BI). Details and background information can be obtained by reading [1].

This document describes a VHDL transceiver module, tr_nonbonded, that can be used on the UniBoard to implement up to twelve full-speed transceivers in non-bonded mode per FPGA side, providing a theoretical throughput of 81,6 Gbps (12 x 8.5Gbps x [8/10]) per FPGA. It is designed to support hardware (other VHDL components) and software (NIOS II) control, as is demonstrated in two designs: *unb_mesh* and *unb_mesh_nios*. Although these two reference designs are used to test the mesh interconnect, the tr_nonbonded module can also be used on the BN_BI.

2 Functional description

2.1 Overview

An instance of the tr_nonbonded module can contain any number of transceivers supported by its host FPGA. On UniBoard, currently one to twelve (0..11) transceivers (see Figure 1) are supported for both sides of the FPGA, so this range is maintained throughout this document. The tr_nonbonded module provides a parallel interface (*tx_datain*, *rx_dataout*) between the fabric and transceivers of 8 to 32 bits wide, depending on the desired data rate.

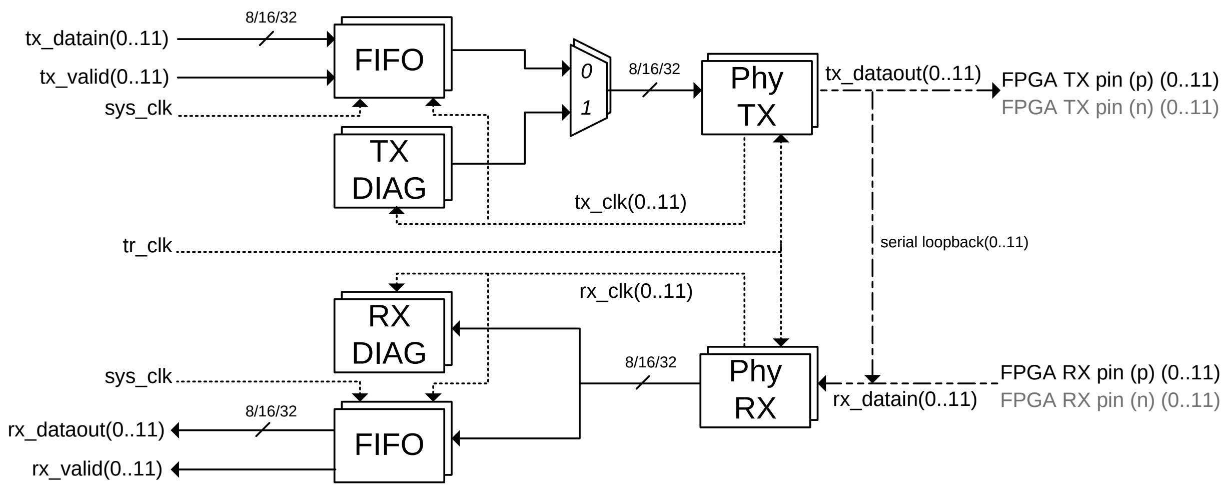


Figure 1 – Functional block schematic of tr_nonbonded

The serial connections (*tx_dataout*, *rx_datain*) of *Phy TX* and *Phy RX* can be connected to any FPGA transceiver pin, or may be looped back. The actual physical TX and RX and the serial loopback connection are contained within instances of an Altera Megafunction called Altera GigaBit Transceiver, or ALTGX.

FIFOs (optional) provide the necessary means of passing the parallel data between the fabric clock domain and the transceiver clock domains. The (also optional) DIAG modules may be used to test the transceiver functionality and physical interconnections.

2.2 Parameters

Parameters are defined as constants in a package file, or can be passed as a generic. After changing some of the parameters described in this chapter, the ALTGX and corresponding ALTGX_RECONFIG must be regenerated. Paragraph 3.2 elaborates further on these two Megafunctions.

2.2.1 Parallel data width

The parallel data width of the transmitter inputs and receiver outputs is dependent on the input transceiver clock frequency and the desired maximum data rate:

*Data width = data rate / transceiver_clk * (8/10)*

By default, the data width is set at 32 bits. The UniBoard uses a 156.25MHz transceiver clock, thus allowing a data rate of 6250Mbps.

2.2.2 Number of transceivers

The maximum number of transceiver instances is only limited by the number of physical transceivers available in the FPGA. As the currently used FPGA provides twelve full (PMA+PCS) transceivers per FPGA side, this is the default value. The tr_nonbonded module supports any number of transceivers between 1 and 12.

2.2.3 Duplex mode, receivers only, transmitters only

By passing values to generics, one can instantiate a transceiver module that contains a number of duplex transceivers, transmitters only, or receivers only.

2.2.4 Instantiate FIFOs

Using the *g_fifos* generic, one can set whether FIFOs should be instantiated or not. If FIFOs are instantiated, the tx_datain and rx_dataout can be clocked in and out using the system (fabric) clock (see Figure 1). If no FIFOs are instantiated (Figure 2), the tx_datain must be clocked using tx_clk, and rx_dataout must be clocked out using rx_clk.
Besides the *g_fifos* generic, generics are provided to set the FIFO depth (*g_fifo_depth)* and the number of words at which the FIFO's TX side buffer is considered to be almost full (*g_fifo_tx_almost_full_nof_words).*

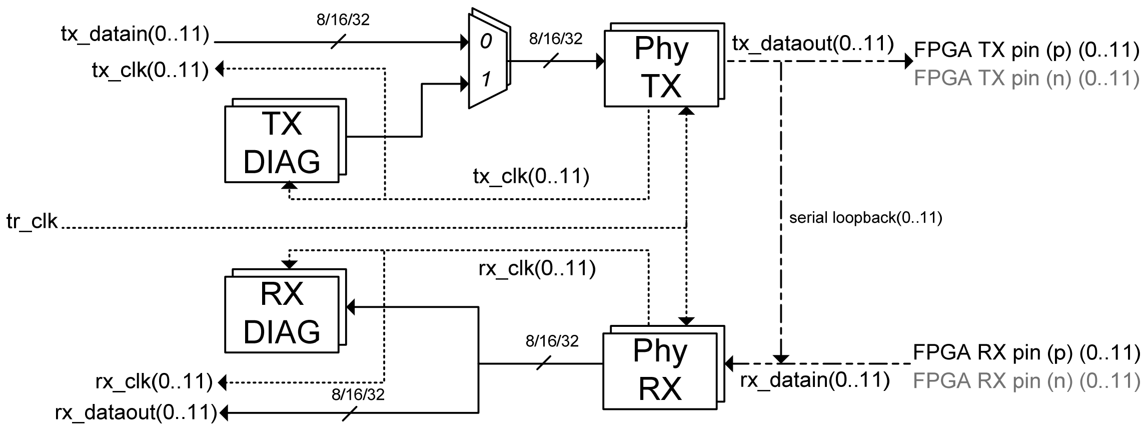


Figure 2 – tr_nonbonded instantiated without FIFOs

2.2.5 Serial loopback mode

Selecting serial loopback instantiates a direct connection between the serial input and the serial output of each transceiver. This mode is useful for functionally testing a design without having to connect it to other FPGAs.

2.2.6 Instantiate DIAG modules

The DIAG modules, originating from LOFAR, are diagnostics modules located at the transmitter and the receiver. The transmitter input is connected to a counter output or a pseudo random bit sequence (PRBS) generator. The receiver output is fed into a DIAG verifier that outputs a running diagnostics result.

2.3 Clock domains

Figure 3 shows a more detailed schematic of the tr_nonbonded module, in which one can distinguish the following three clock domains:

- System (fabric) clock (*ssc*)
- Receiver clock (*rxc*)
- Transmitter clock (*txc*)

To facilitate using the module in a design, these clock domains are abbreviated in the VHDL signal prefixes. This not only avoids unintended clock domain crossings, but is also necessary because of the optional

FIFOs: when instantiating the module with FIFOs, the parallel input and output data is in the system clock domain and the *rxc* and *txc* prefixed signals are not used. Without FIFOs, one must discriminate between data in the transmitter clock (*txc*) domain and data in the receiver clock (*rxc*) domain, leaving the *ssc* prefixed signals unused.

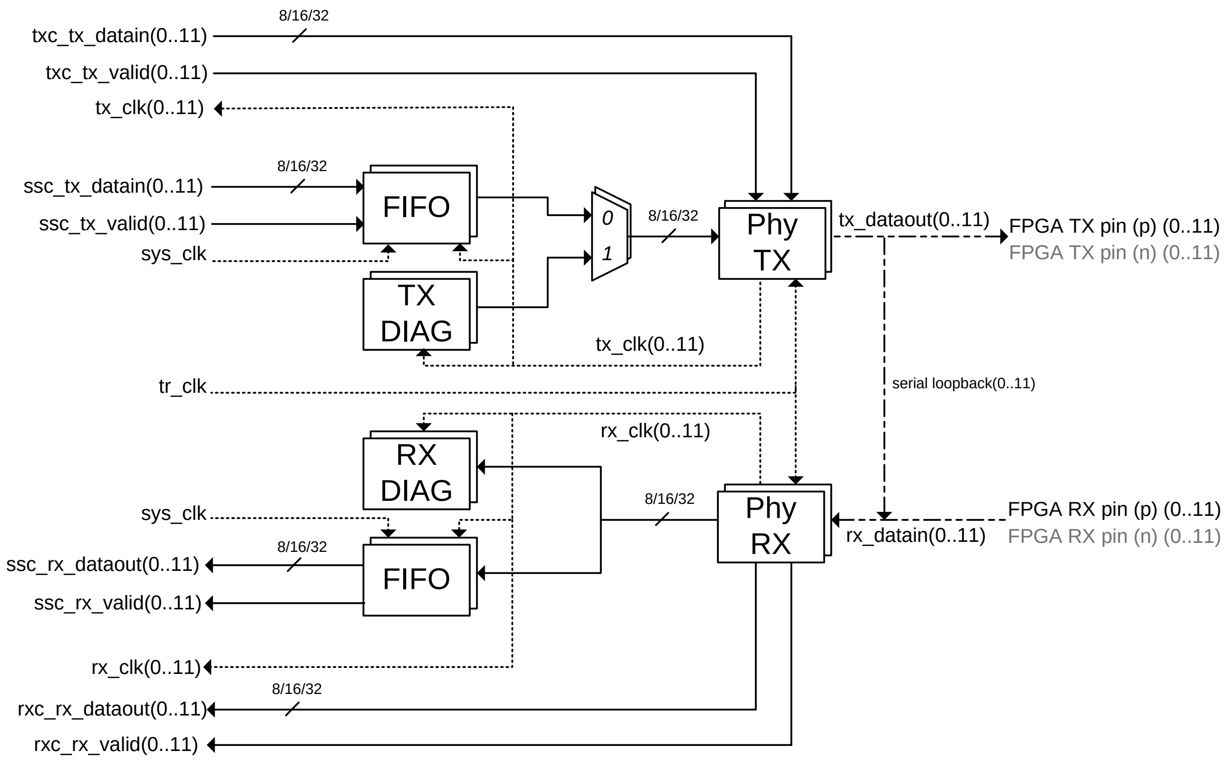


Figure 3 – Clock domains used in the tr_nonbonded module

3 Implementation

This chapter gives an overview of the VHDL architecture and hierarchy (Figure 4) of the tr_nonbonded module and its components.

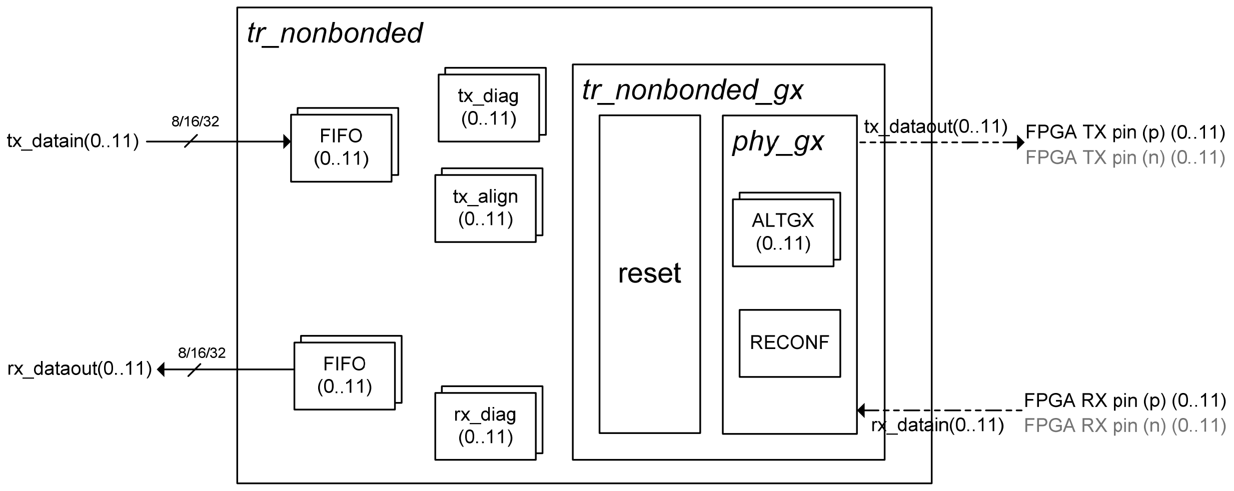


Figure 4 – Hierarchy of tr_nonbonded

3.1 Alignment of serially received data

Upon serialization of parallel data, the word boundary is lost. To restore this word boundary, an alignment block is used on the receiver side. Besides that, depending on when the system comes out of reset, the byte order may not match the original byte order. This problem is addressed by the *byte ordering block*. This paragraph describes the alignment procedure and these two necessary blocks.

To align the data on the receiver side, the transmitter must send a specific alignment pattern. This is performed by the *tx_align* entity. This entity is responsible for the selection of two types of data that can be fed into the transmitter:

- The user data
- The alignment pattern

During initialization after power up, the *tx_align* entity continuously outputs the alignment pattern to the parallel input of the transmitters. The enable input of *tx_align* and the duration of the assertion of this signal are controlled by the reset state machine. When enabled, the input user data is not forwarded to the transmitter. Instead, the alignment pattern is sent. This pattern (*c_alignment_pattern*) is set in the tr_nonbonded.pkg file. The actual alignment to this pattern on the receiver side is performed by the *Word Alignment Block*. The same pattern is also used by the byte ordering block to restore the byte order. Word alignment and byte ordering blocks are available as options in the ALTGX Megafunction.

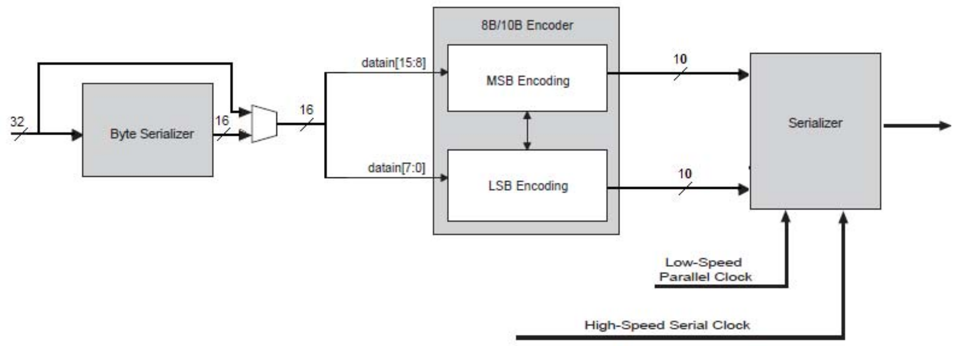
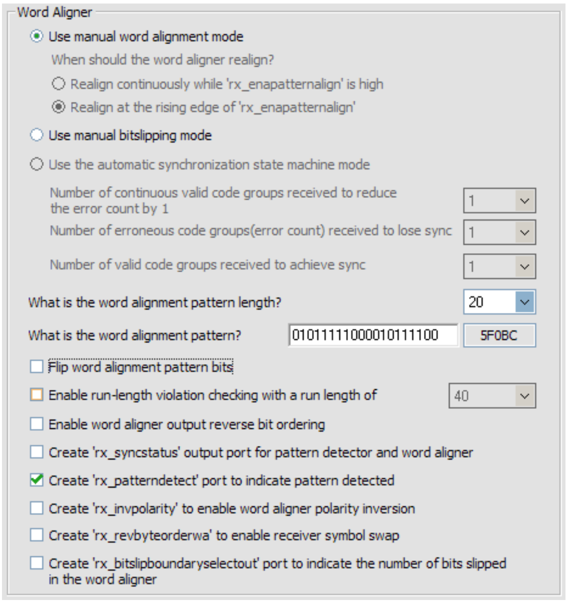


Figure 5 – Transmitter data path

3.1.1 Word alignment

When a 32-bit width is used for the fabric-transceiver interface, as illustrated in Figure 5, the 32-bit data is byte serialized into half the width at twice the frequency. The resulting 16 bits are fed into the 8b/10b encoder that encodes the MSB and LSB into 10-bits each and outputs these 20 bits to the serializer. On the receiver side, the word aligner is located before the 8b/10b decoder and thus looks for word alignment patterns that are 8b/10b encoded. This word alignment pattern, 10 (single width) or 20 (double width) bits wide, is entered as a parameter in the corresponding ALTGX Megafunction field (see Figure 6).



Word Aligner

- [✓] Use manual word alignment mode
  - When should the word aligner realign?
  - [ ] Realign continuously while 'rx_enapatternalign' is high
  - [✓] Realign at the rising edge of 'rx_enapatternalign'
- [ ] Use manual bitslipping mode
- [ ] Use the automatic synchronization state machine mode
  - Number of continuous valid code groups received to reduce the error count by 1: 1
  - Number of erroneous code groups(error count) received to lose sync: 1
  - Number of valid code groups received to achieve sync: 1

What is the word alignment pattern length? 20

What is the word alignment pattern? 01011111000010111100 5F0BC

- [ ] Flip word alignment pattern bits
- [ ] Enable run-length violation checking with a run length of 40
- [ ] Enable word aligner output reverse bit ordering
- [ ] Create 'rx_syncstatus' output port for pattern detector and word aligner
- [✓] Create 'rx_patterndetect' port to indicate pattern detected
- [ ] Create 'rx_invpolarity' to enable word aligner polarity inversion
- [ ] Create 'rx_revbyteorderwa' to enable receiver symbol swap
- [ ] Create 'rx_bitslipboundaryselectout' port to indicate the number of bits slipped in the word aligner

Figure 6 – Word alignment

The 20-bit default alignment pattern for the tr_nonbonded module is:

0011110100	0011111010	= 8b/10b encoded, LS Bits right
K28.0	K.28.5	= Comma symbol followed by control word
0xBC	0x1C	= 8b/10b decoded

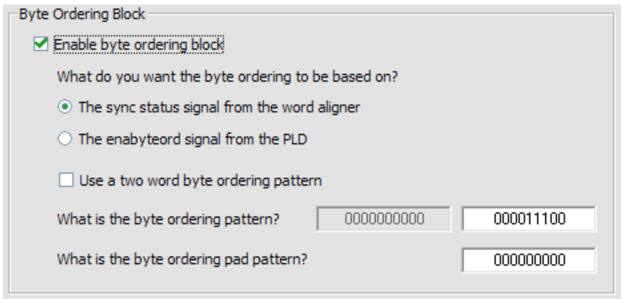
This corresponds to 0xBC1C after 8b/10b decoding at the receiver end and before 8b/10b encoding at the transmitter end. The comma symbol (K.28.5 or 0xBC) differs from the second byte so their places cannot be swapped during alignment. The second byte (0x1C) is also used in the byte ordering block described in the next paragraph.

The alignment pattern is input in the Megafunction in reverse, the way it is sent after serialization:

0101111100 0010111100 (0x5F0BC) = 8b/10b encoded, LS Bits left (reversed)

3.1.2 Byte ordering

As described in the previous paragraph, the 0x1C byte is used for byte ordering. The byte ordering block (available as option in the ALTGX Megafunction, see Figure 7) uses the synchronization status output of the word aligner to start the byte ordering – this is necessary because the byte deserializer at the receiver could output the bytes in the wrong order. When it finds byte ordering pattern 0x1C at the LSB position, it considers the data to be aligned. Otherwise, it inserts a padding pattern (set to 0x00) to effectively shift the data onto the LSB position.


Byte Ordering Block

[✓] Enable byte ordering block

What do you want the byte ordering to be based on?

[✓ The sync status signal from the word aligner]

The enabyteord signal from the PLD

[ ] Use a two word byte ordering pattern

What is the byte ordering pattern? 0000000000 000011100

What is the byte ordering pad pattern? 000000000

Figure 7 – Byte ordering options

3.2 ALTGX and ALTGX_RECONFIG

There are three types of pre-generated ALTGX Megafunctions that can be instanced in *phy_gx (see* Figure 4 – Hierarchy of tr_nonbonded*)*, depending on the Boolean generics *g_tx* and *g_rx*:

- Duplex
- Transmitters only
- Receivers only

Also, one of two ALTGX_RECONFIG Megafunctions will be instanced:

- *stratix_iv_gx_reconfig* -> for synthesis
- *stratix_iv_gx_reconfig_sim* -> for simulation

The *stratix_iv_gx_reconfig_sim* instance will be used when the *g_sim* generic is set to TRUE, and will instance a reconfiguration module for two (duplex/transmitter/receiver) channels, yielding faster simulation times. When the synthesis module is used, the number of transceivers set in the package file will be used.

4 Designs using tr_nonbonded instances

This chapter describes two designs, unb_mesh and unb_mesh_nios, that instantiate the tr_nonbonded module with up to twelve transceivers. Both are used to test the UniBoard mesh infrastructure and transceiver capabilities. Unb_mesh does not contain a NIOS II processor, which makes it more appropriate for simulation than unb_mesh_nios. The design consists of a pseudo random number generator connected directly to each transmitter and a verifier connected directly to each receiver. Unb_mesh_nios is intended to provide an example of a design using the tr_nonbonded module with a NIOS II processor. The unb_mesh_nios design uses tr_nonbonded module without the diagnostics modules enabled in tr_nonbonded itself (also an option in unb_mesh). Instead, diagnostics modules are used in the top-level entity, the same entity in which the tr_nonbonded module is instantiated. Also, the 'user data' (generated and verified by the diagnostics modules) crosses the system and transceiver clock domains by means of FIFOs.

4.1 unb_mesh

The unb_mesh design uses a tr_nonbonded instance with 12 channels, and uses the DIAG modules to generate a bit stream at the transmitter and to verify the data at the receiver end. Figure 8 shows how the transceivers connect to the mesh interconnect.

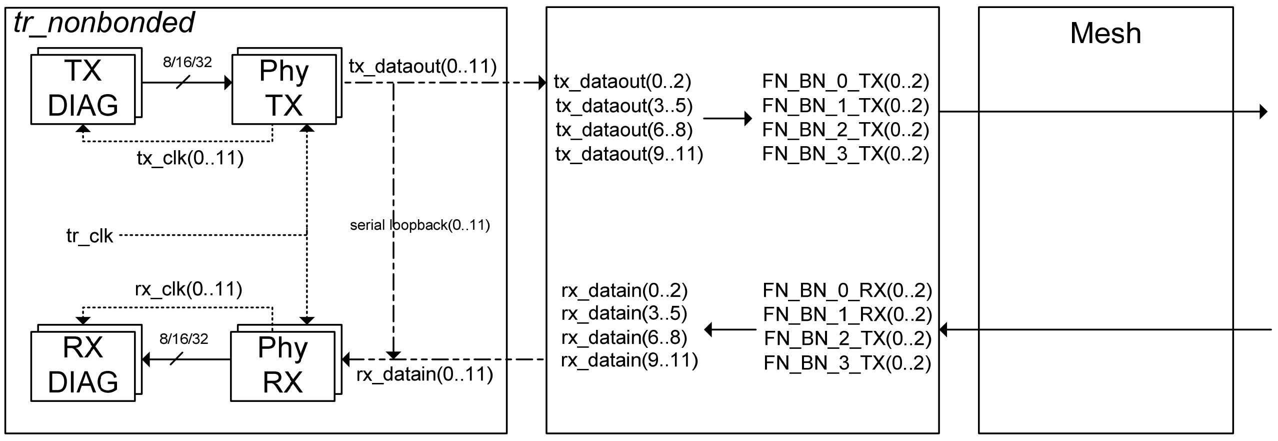


Figure 8 - Block schematic of node with duplex channels

The diagnostics result of each receiver is output by the DIAG verifiers at the receiver ends. Each bit of this [*c_data_with + 1*] bit value should be zero to indicate correct data is being received. This [*c_data_with + 1*] bit value is converted to a one-bit value via an OR operation, so one bit per receiver indicates whether or not correct data is received. These 12 (in a 12-channel design) individual bits are then again fed into an OR gate so a one-bit signal, connected to an LED, indicates the test result.

4.1.1 Simulation: duplex link

The test bench of unb_mesh instantiates two nodes and connects the FN_BN_#_RX signals of one node to the FN_BN_#_TX signals of the other, and vice versa.

4.1.2 Simulation: simplex link

There is a second option in the test bench, which can be selected to simulate two nodes, one with transmitters only and the second with receivers only.

4.1.3 Simulation: top-level diag modules and FIFOs

A third test bench option is to instantiate the tr_nonbonded module without diag modules. In this example, diag modules are instantiated in the top entity and interface to the tr_nonbonded module via FIFOs.

4.1.4 Synthesis

Synthesis of unb_mesh can be performed on a duplex design with any number of channels. Other modes are supported in tr_nonbonded but require modifications of unb_mesh.
This firmware is meant to be downloaded onto all eight FPGAs, as the test will fail if any receivers do not get (correct) input data.

4.2 unb_mesh_nios

This design (see Figure 9) uses a tr_nonbonded instance with FIFOs and without diagnostics modules. Instead, the diagnostics modules are instantiated in the top level entity and are controlled by a NIOS II processor which is part of an SOPC.

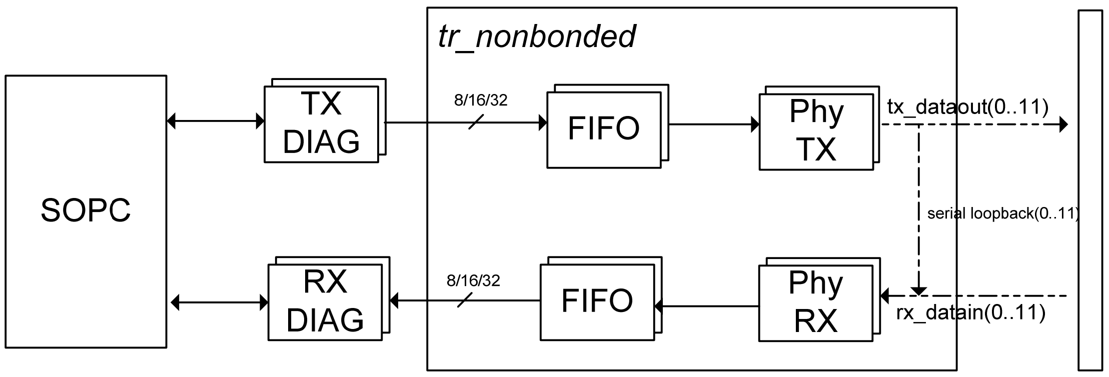


Figure 9 – Unb_mesh_nios: an SOPC controlling a tr_nonbonded instance

The SOPC contains PIO registers that control or read out the following signals:

DIAG: TX sequencer:
- Enable
- Mode: PRSG or counter

DIAG: RX Monitor:
- Enable
- Mode: PRSG or counter
- Diagnostics result
- Diagnostics result valid

Receiver buffers
- Take 100 samples of one of the receiver buffers

4.2.1 Software control

The PIO registers can be accessed by using the JTAG UART menu. This menu is displayed after start up, and requires that the user has opened a NIOS II terminal within a NIOS II command shell. A possible testing sequence could be the following:
- Download SOF file onto all 8 nodes
- Open 8 NIOS II terminals
- Enable all TX sequencers
- Enable all RX monitors
- Display the diagnostics result of each node

The functions that perform these actions are located and described in:

$*UNB/firmware/software/modules/tr_nonbonded/*

A main source file exists at the following location:

$*UNB/firmware/software/apps/unb_mesh_nios/main.c.*

4.2.2 Simulation

Simulation of this design is possible, however not recommended (or maintained) because the NIOS II processor combined with the transceiver instances make simulation very slow. In addition, the unb_mesh_nios design is made to control multiple or all nodes simultaneously, which is impossible in simulation in terms of speed and the lack of support for JTAG UART input in ModelSim.
For simulation, the unb_mesh design should be used.

4.2.3 Synthesis

This design can be run on any number of nodes. However, only the receivers connected to active (and booted at the same time) transmitters will succeed in aligning properly.
As the full diagnostics result is read from a register as a hexadecimal value, one can determine which receivers failed the test by converting the value to binary, e.g. the binary value *0000 0000 0010* indicates that the test failed on transceiver 1 (out of 12 -> 0..11).

5 Appendix – list of files

5.1 tr_nonbonded

The tr_nonbonded module:
https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk/Firmware/modules/tr_nonbonded

5.2 unb_mesh

The VHDL source code, Quartus project files and ModelSim files are located at:
https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk/Firmware/designs/unb_mesh

5.3 unb_mesh_nios

The VHDL source code, Quartus project files are located at:
https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk/Firmware/designs/unb_mesh_nios/

The matching C code for the NIOS II processor is located at:
https://svn.astron.nl/UniBoard_FP7/UniBoard/trunk/Firmware/software/modules/src/tr_nonbonded