

Firmware Specificatie voor 10G XAUI en 10GbE

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

© ASTRON 2011
All rights are reserved. Reproduction in whole or in part is prohibited without written consent of the copyright owner.

Distribution list:

Group:	Others:
Andre Gunst Daniel van der Schuur Harm Jan Pepping	Gijs Schoonderbeek

Document history:

Revision	Date	Author	Modification / Change
0.1	2011-12-12	Eric Kooistra	Creatie
0.2	2011-12-15	Eric Kooistra	Klaar voor review
1.0	2011-12-22	Eric Kooistra	<ul style="list-style-type: none">- Review commentaar verwerkt van AG, GS, DS, HJP.- 10G validatie met PC nu via Vitesse PHY loopback in plaats van via XGMII loopback in VHDL.- DP header insert / remove component ook nodig voor 1GbE ETH module.

Table of contents:

1	Introductie.....	6
1.1	Doel.....	6
1.2	Achtergrond.....	6
1.2.1	10G en 10GbE.....	6
1.2.2	IP versie.....	7
1.3	Overzicht.....	7
2	Functionele specificatie.....	9
2.1	TR_XAUI module.....	9
2.1.1	Initialisatie.....	9
2.1.2	Error checking.....	9
2.1.3	XGMII control en data.....	9
2.1.4	MDIO support.....	10
2.1.5	MM control.....	10
2.2	ETH10 module.....	11
2.2.1	UDP offload.....	12
3	Implementatie specificatie.....	14
3.1	TR_XAUI module.....	14
3.1.1	Reference design unb_tr_xaui.....	14
3.1.2	PHY loopback.....	15
3.2	ETH10 module.....	16
3.2.1	MAC woordbreedte.....	16
3.2.2	Payload woordalignment.....	16
3.2.3	Header mapping.....	16
3.2.4	Reference design unb_eth10.....	16
3.3	Wrappers.....	17
4	Test specificatie.....	18
4.1	Verificatie van TR_XAUI.....	18
4.1.1	Testbench tb_phy_xaui voor de XAUI IP.....	18
4.1.2	Testbench tb_tr_xaui for TR_XAUI module.....	18
4.1.3	Testbench tb_tr_xaui_mdio voor TR_XAUI met MDIO.....	19
4.1.4	Testbench tb_node_unb_tr_xaui.....	19
4.1.5	Testbench tb_unb_tr_xaui.....	19
4.2	Verificatie van ETH10.....	20
4.2.1	Testbench tb_mac10 voor de MAC10 IP.....	20
4.2.2	Testbench tb_eth10 voor de ETH10 module.....	20
4.2.3	Testbench tb_dp_header_insert_remove.....	21
4.2.4	Testbench tb_eth10_udp_offload voor UDP offload.....	21
4.2.5	Testbench tb_node_unb_eth10.....	21
4.2.6	Testbench tb_unb_eth10.....	21
4.3	Validatie.....	22
5	Stappenplan.....	23
5.1	TR_XAUI ontwikkeling.....	23
5.2	UDP offload ontwikkeling.....	23
5.3	ETH10 ontwikkeling.....	23
6	Eindresultaat.....	24
7	Appendix: Verificatie en validatie methode.....	25

Terminology:

10G	10 Gigabit
10Gbps	10 Giga bit per second
10GbE	10 Gigabit Ethernet
1GbE	10M/100M/1G Ethernet
BIST	Built In Self Test
BN	Back Node
DP	Data Path module [14, 15], DP = ST + timestamp information
DP packet	Eigen protocol om DP signalen via pakketten te transporteren [11]
DUT	Device Under Test
eop	End Of Packet
ETH	Ethernet
ETH10	Firmware module voor 10 Gigabit Ethernet
FN	Front Node
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HW	Hardware
IO	Input Output
IP	Intellectual Property, Internet Protocol
IPv4	Internet Protocol version 4
Lane	Een transceiver in een link
LASI	Link Alarm Status Interrupt
LCU	Local Control Unit
Link	Een groep lanes met dezelfde begin en eind
LOFAR	Low Frequency Array
MAC	Media Access Controller
MAC10	Media Access Controller voor 10GbE
MDIO	Management Data IO
MISO	Master In Slave Out [14]
MM	Memory-Mapped
MOSI	Master Out Slave In [14]
NIOS	Softcore microprocessor in FPGA
PHY	Physical layer
RTL	Register Transfer Level
SFP	Small Form Factor Pluggable (for up to 4.25 Gbps)
SFP+	Enhanced Small Form Factor Pluggable (for up to 10 Gbps)
SISO	Sink Out Source In [14]
ST	Streaming
sop	Start Of Packet
SOSI	Source Out Sink In [14]
SW	Software
TR_XAUI	Firmware module voor XGMII interface naar XAUI interface
TSE	Triple Speed Ethernet = ETH1
UDP	User Datagram Protocol
UNB	UniBoard
Unbos	UniBoard Operating System on the NIOS
Uthernet	Eigen point-to-point protocol dat lijkt op Ethernet maar dan zonder adressering [10]
XAUI	10-Gigabit Attachment Unit Interface (4 lanes @ 3.125 Gbps)
XFI	10 Gigabit Small Form Factor Pluggable (single lane @ 10.3125 Gbps)
XGB	10G Break-out board
XGMII	10 Gigabit Media Independent Interface

References:

1. "UniBoard V1.0 Board Description", ASTRON- PDT-00001, G. Schoonderbeek, S. Zwier
2. "Digital Beamformer System for APERTIF", ASTRON-RP-320, A. Gunst, E. Kooistra
3. "Digital Correlator System for APERTIF", ASTRON-RP-331, A. Gunst, E. Kooistra
4. "1Gb Ethernet Module Description", ASTRON-RP-396, E. Kooistra
5. "Testing 10 Gigabit links between UniBoard and a PC", nov 2010, JIVE, J. Hargreaves
6. Vitesse: "VSC8486 LAN/WAN PHY and VSC8476 LAN PHY Design Guide"
7. Vitesse: "VSC84-86-11 10 Gbps XAUI to XFI LAN/WAN Transceiver Datasheet"
8. Altera: "10-Gbps Ethernet MAC MegaCore Function User Guide", 10Gbps_MAC.pdf
9. Altera: "Transceiver PHY IP Core User Guide", xcvr_user_guide.pdf
10. "Uthernet (UTH) Module Description", ASTRON-RP-871, E. Kooistra
11. "Data Path Packet Module Description", ASTRON-RP-873, E. Kooistra
12. "UniBoard transceiver module", ASTRON-RP-449, D. van der Schuur
13. "UniBoard DDR3 firmware module", ASTRON-RP541, D. van der Schuur
14. "Specification for module interfaces using VHDL records", ASTRON-RP-380, E. Kooistra
15. "DP Streaming Module Description", ASTRON-RP-382, E. Kooistra
16. <http://en.wikipedia.org/wiki>

1 Introductie

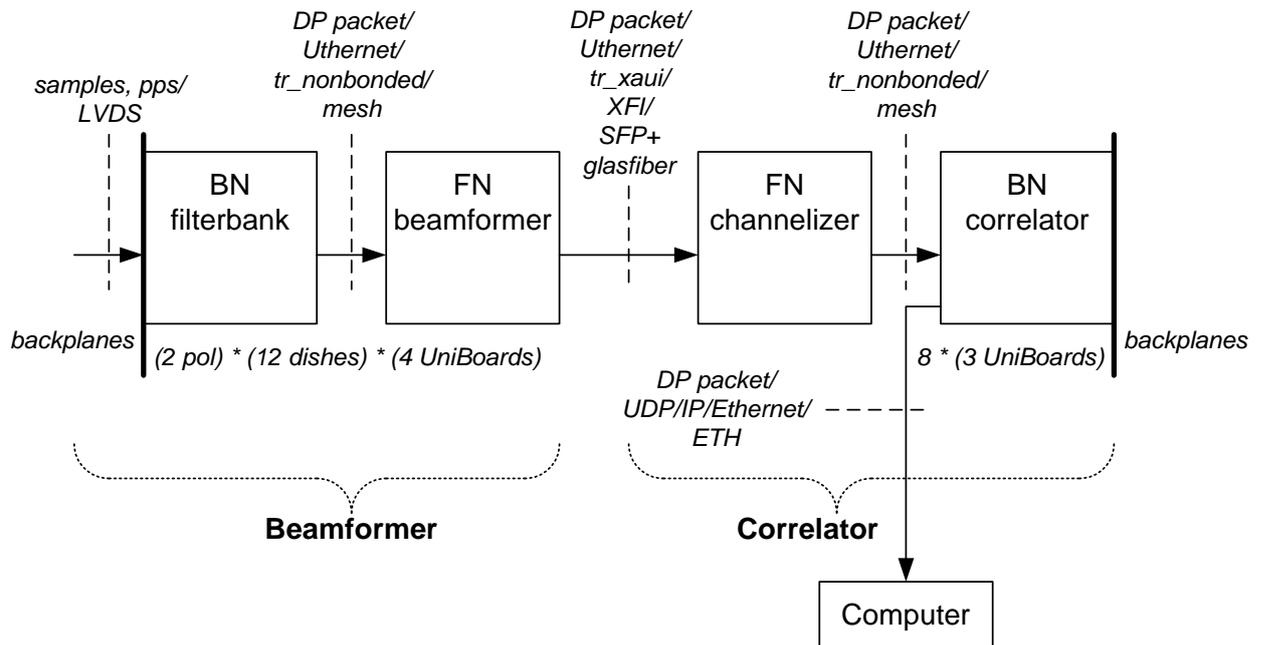
1.1 Doel

Dit document specificeert twee 10Gbps VHDL firmware modules. De ene biedt data communicatie op XGMII niveau via XAUI en heet TR_XAUI. De andere biedt data communicatie op 10 Gigabit Ethernet (10GbE) niveau en heet ETH10. De ETH10 module maakt gebruik van de TR_XAUI module. De TR_XAUI module is voldoende voor de APERTIF applicatie, de 10GbE functionaliteit hebben we voor APERTIF niet nodig, daarom moeten we eerst de TR_XAUI module maken. Dit document biedt ook een stappenplan voor de ontwikkeling van eerst de TR_XAUI module en vervolgens de ETH10 module.

1.2 Achtergrond

1.2.1 10G en 10GbE

Het 10Gbps interface op de front node (FN) FPGAs van UniBoard [1] is een XAUI interface. In het XAUI interface worden 4 gigabit transceivers gebundeld tot 1 samengestelde 10 Gbps link. Figuur 1 toont een overzicht van het APERTIF beamformer [2] en correlator [3] systeem op basis van UniBoard. Voor APERTIF is het voldoende om alleen communicatie op XAUI niveau te ondersteunen, omdat in APERTIF de 10 Gbps links gebruikt worden tussen FN en niet tussen FN en een 10GbE netwerkkaart in een PC of via een 10GbE switch. De output van de APERTIF correlator kan namelijk via de UDP offload poort van de 1GbE links gebeuren [3].



Figuur 1: UniBoard interfaces en de APERTIF beamformer and correlator

UniBoard is een generiek DSP board voor meerdere astronomische toepassingen, daarom moet het 10 Gbps XAUI interface ook gevalideerd worden voor communicatie met een 10GbE netwerkkaart in een PC. De link tussen een FN en een netwerkkaart in een PC kan ook geverifieerd worden door de PC de 10GbE data te laten genereren en deze data direct terug te retourneren in de FN, dus zonder de 10GbE data te interpreteren in de FN. Om daadwerkelijk in een applicatie op 10GbE niveau te kunnen communiceren met

een FN is de 10GbE Media Access Controller (MAC) IP van Altera nodig. Deze 10GbE MAC IP communiceert via XGMII met het XAUI interface.

Merk op dat 4 losse gigabit transceivers op zich sneller kunnen opereren dan 10 Gbps. Echter de XAUI standaard en de 10GbE standaard definiëren de data rate vast op 10 Gbps. Hierbij is deze 10 Gbps de 'user' data rate dus zonder 8B10B coding of 64B66B coding. De werkelijke bitrate op de link is hoger ten gevolge van deze codingfactor, daarom moet een XAUI transceiver ten minste 3.125 Gbps aankunnen en moet een enkele 10G transceiver ten minste 10.3125 Gbps aankunnen.

Merk op dat 10G of 10Gbps wat anders is dan 10GbE. Met 10GbE wordt 10 Gigabit Ethernet communicatie aangeduid en dit kan dus over een 10G link. Echter over een 10G link kan ook een ander protocol gecommuniceerd worden, zoals bijvoorbeeld het Uthernet protocol [10] of diagnostics test data [12]. Zolang je niet hoeft te communiceren via een switch of met een PC is het niet nodig om het Ethernet protocol te gebruiken. Voor point-to-point links is het Uthernet protocol [10] voldoende. In APERTIF verstuurt de FN in een beamformer subrek de beamlet data naar de FN in het correlator subrek. Net als voor de communicatie tussen back nodes via het backplane zullen we voor de communicatie tussen front nodes via glasvezel het DP packet protocol [11] over het point-to-point Uthernet protocol [10] gaan gebruiken, zie Figuur 1.

De communicatie tussen FN in verschillende subrekken of tussen FN en een PC netwerkkaart gebeurt veelal over grotere afstanden (> 10 m). Daarom behoeven het FN XAUI interface en het FN 10GbE interface alleen gevalideerd te worden voor glasvezel links, dus niet ook voor koper kabels.

1.2.2 IP versie

Het huidige *fn_base* BIST design voor de front nodes (FN) maakt gebruik van de v9.1 Altera 10GbE IP in:

```
$UNB\Firmware\modules\MegaWizard\eth10g\v91
```

De v9.1 Altera 10GbE IP omvat de MAC10 IP, de XAUI IP en de MDIO geïntegreerd in 1 component. De Management Data IO (MDIO) funktie is nodig in geval van seriële control van een PHY chip. Daarnaast is er ARP en ping code beschikbaar in VHDL die door JH [5] is gemaakt op basis van CASPER code. Deze v9.1 Altera 10GbE IP en ARP en ping code werkt wel, echter we kunnen er beter niet mee verder gaan omdat:

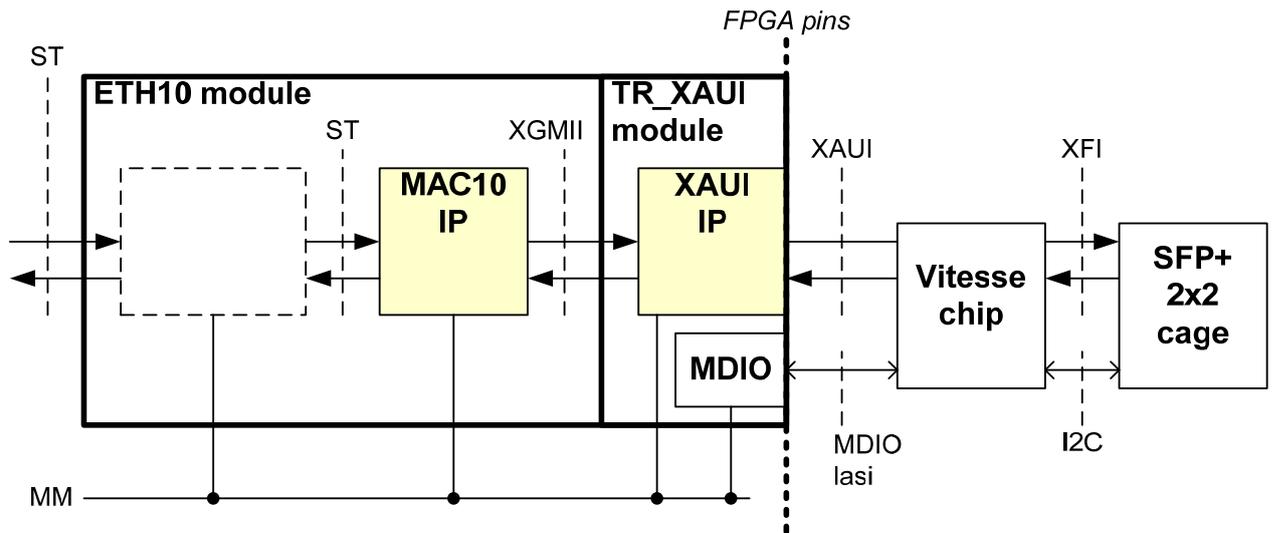
- het bevat de 10 GbE MAC inclusief de XAUI, terwijl APERTIF ook genoeg heeft aan alleen de XAUI
- het is gebaseerd op v9.1 en Altera beveelt aan om > v10 voor nieuw designs te gebruiken
- we hebben er nog geen stress tests mee gedaan
- het is lastig te onderhouden

De huidige v10 Altera 10GbE IP wordt niet meer als een compleet geïntegreerd blok aangeboden, maar moet zelf samengesteld worden uit de losse IP componenten voor de MAC10, de XAUI en de MDIO. De MAC10 [8] en de XAUI [9] zijn als MegaWizard IP beschikbaar. Dit is belangrijk voor ons, omdat we niet aan de SOPC Builder of QSys system flow van Altera vast willen zitten. De MDIO is niet beschikbaar als MegaWizard component maar gelukkig hebben we deze zelf en is deze ook al overgezet vanuit LOFAR naar UniBoard. JH heeft deze MDIO begin 2010 ook al eens succesvol toegepast op UniBoard [5].

```
$UNB\Firmware\modules\Lofar\mdio
```

1.3 Overzicht

Figuur 2 geeft een overzicht van de ETH10 firmware module en de TR_XAUI firmware module op een front node FPGA in combinatie met de Vitesse PHY chip en de SFP+ 2x2 cage. De Vitesse PHY chip zorgt voor de conversie van XAUI naar XFI richting een 10Gbps poort op de SFP+ 2x2 cage. De ETH10 module omvat de MAC10 IP van Altera en eigen RTL en de TR_XAUI instantie. De TR_XAUI module omvat de XAUI IP van Altera en de MDIO (van LOFAR). De XGMII loopback is een nog te maken loopback component.



Figuur 2: De ETH10 module inclusief de TR_XAUI module in een FN

Het eigen RTL gedeelte in de ETH10 module (aangegeven door het gestreepte blok in Figuur 2) zorgt voor de communicatie van UDP pakketjes en voor het afhandelen van speciale pakketjes zoals ARP en ping. De Media Access Controller (MAC) zorgt dat de Ethernet frames als streaming pakketjes kunnen worden gecommuniceerd met het XGMII interface. Het XGMII interface is een parallel interface. De TR_XAUI module zorgt dat deze signalen serieel over 4 transceiver lanes gecommuniceerd kunnen worden met de Vitesse PHY chip. De Vitesse PHY chip zorgt dat de informatie over 1 lane gecommuniceerd kan worden, zodat er dan nog maar 1 transceiver module nodig is per 10G link. De 2x2 cage biedt vier 10G transceiver modules en er zijn ook vier Vitesse PHY chips per front node.

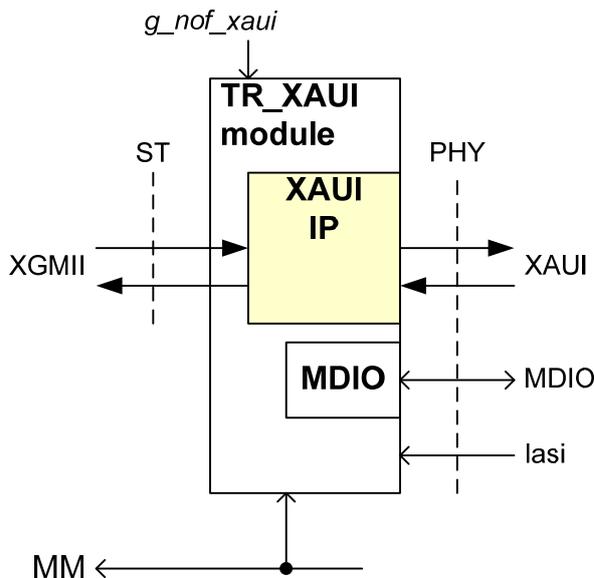
Per front node zijn er maximaal 12 full-featured FPGA transceivers beschikbaar, daarom worden drie 10 Gbps links gebruikt en zijn er maximaal 3 instantanties van de TR_XAUI module of van de ETH10 module (met daarin de TR_XAUI module) nodig. Het vergt extra ontwikkeltijd om de 4 less-featured transceivers te kunnen gebruiken voor de vierde (soft) XAUI, daarom doen we dat eerst niet. Echter zonder deze vierde (soft) XAUI 10G link hebben we mogelijk steeds 4 in plaats van 3 UniBoards nodig in de APERTIF correlator. Dit komt omdat het aantal benodigde links tussen de APERTIF beamformer en de APERTIF correlator niet zo zeer wordt bepaald door de data rate maar vooral door het aantal poorten. Uit Figuur 1 volgt dat de totale APERTIF beamformer (2 pol) * (12 dishes) * (4 UniBoards) * (4 FN/UniBoard) = 384 FN heeft. Elke beamformer FN gebruikt 1 10G link voor output naar een correlator FN. Op de totale APERTIF correlator komen dus 384 10G links binnen. Met 4 10G links per correlator FN zijn er hier dan dus 24 UniBoorden voor nodig (zoals aangegeven in Figuur 1) en met 3 10G links per FN zijn er dan dus 32 UniBoorden nodig.

De MDIO is nodig om de Vitesse 8486 PHY chip [6, 7] in te kunnen stellen. Via de MDIO kan indien nodig ook de I2C master in de Vitesse PHY chip bedient worden en met deze I2C master kan dan een 10G transceiver module in de SFP+ 2x2 cage ingesteld worden. De Vitesse PHY chip output ook het LASI (Link Alarm Status Interrupt) signaal naar de FN, dit signaal kan beschikbaar gemaakt worden als een bit in een MM status register. De Vitesse PHY chip geeft ook een RX_ALARM en TX_ALARM output, maar deze zijn verbonden met LEDs op de SFP+ 2x2 cage en niet verbonden met de FN FPGA.

2 Functionele specificatie

2.1 TR_XAUI module

Aangezien de XAUI functionaliteit ook zonder 10 Gigabit Ethernet toepasbaar is is het logisch om de TR_XAUI als een aparte module te definiëren. Figuur 3 toont de TR_XAUI module bestaande uit de Altera XAUI IP en de MDIO.



Figuur 3: TR_XAUI module inclusief MDIO en register voor LASI input

Via generic *g_nof_xaui* is het mogelijk om 1 of meer XAUI interfaces te instantiëren. De XAUI IP heeft waarschijnlijk geen MM control maar een externe PHY chip mogelijk wel en dat wordt dan verzorgd door de MDIO component en een parallel IO register voor signalen zoals LASI [1, 6, 7]. De Management Data IO (MDIO) functie moet selecteerbaar zijn middels een generic *g_use_mdio*.

2.1.1 Initialisatie

De TR_XAUI hoeft geen status informatie aan te geven naar de user. Voor Tx is het voldoende om aan te nemen dat de XAUI link vrij snel na node power up actief is. Daarvoor wordt valid Tx user data genegeerd en daarna geaccepteerd. Voor Rx is het voldoende dat de TR_XAUI via een valid signaal aangeeft dat er valid user data beschikbaar is.

2.1.2 Error checking

De TR_XAUI module zelf heeft geen data error detectie te doen. Data error detectie gebeurt op een communicatie niveau hoger bijvoorbeeld in de *diagnostics* test data monitor of via de CRC-check in het Ethernet pakket [10] of in een Ethernet packet [16].

2.1.3 XGMII control en data

De XAUI IP moet toegankelijk zijn via XGMII in de vorm van een SOSI signaal. Het XGMII interface bestaat uit 64 bit data *txd[63:0]* en 8 bit control *txc[7:0]*. Omdat we het Ethernet protocol hebben hoeven we de sop en eop mogelijkheden van XGMII niet te gebruiken. De SOSI valid inactive wordt gecodeerd als *txc = 0xFF* en *txd* als 8 IDLE = 0x07 bytes en de SOSI valid active wordt gecodeerd als *txc = 0x0* met de *txd* is de SOSI data[63:0]. We gebruiken het XGMII interface in eenheden van 64 bit woorden, met SOSI valid maar zonder SOSI sop en SOSI eop.

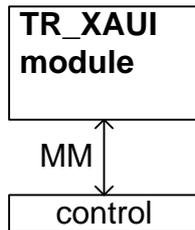
2.1.4 MDIO support

De Management Data IO functie in de TR_XAUI module is nodig om de Vitesse PHY chip te kunnen bedienen voor bijvoorbeeld:

- om enkele 10G PHY link parameters nader in te stellen (zie code van JH [5])
- om de Vitesse PHY in loopback te kunnen zetten (zie sectie 3.1.2).

2.1.5 MM control

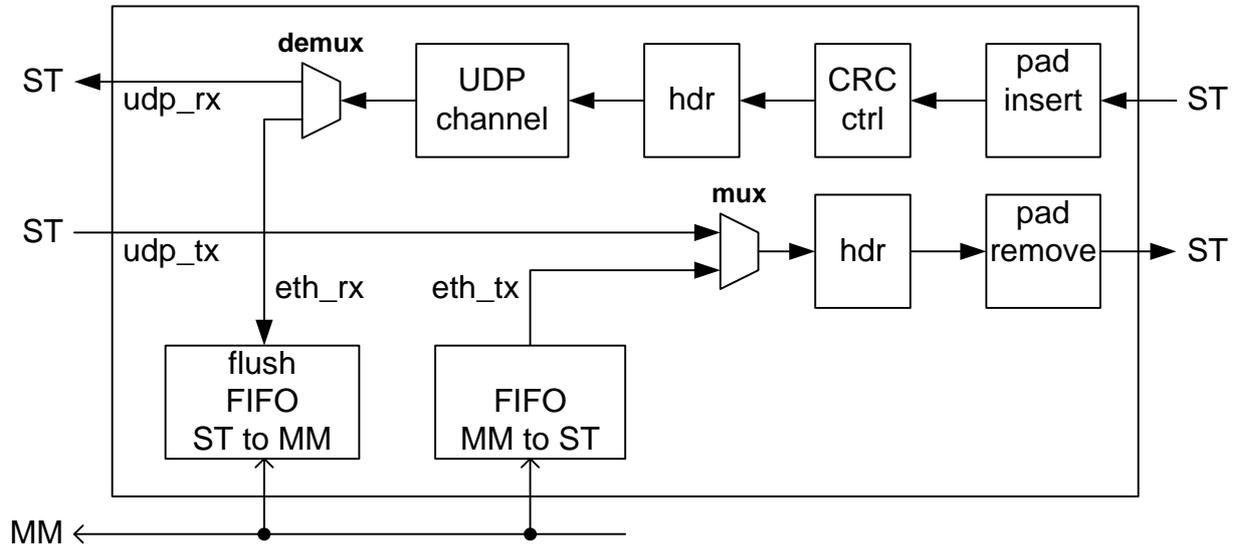
De MM control voor de TR_XAUI module kan gebeuren via de NIOS als taak in *unbos* of vanuit de PC via MM access schrijf- en leesberichtjes. Echter aangezien de benodigde MM control waarschijnlijk eenmalig kan deze PHY control beter als een state machine component in VHDL beschikbaar zijn zoals aangegeven in Figuur 4.



Figuur 4: TR_XAUI MM control in VHDL

2.2 ETH10 module

Qua architectuur kan de ETH10 module sterk lijken op de ETH module [4] voor 1 GbE die we al hebben. In plaats van de Altera TSE MAC IP hebben we nu dan de Altera 10GbE MAC IP. Figuur 5 toont het blokdiagram van de ETH10 RTL code (passend in het gestreepte blok in Figuur 2).

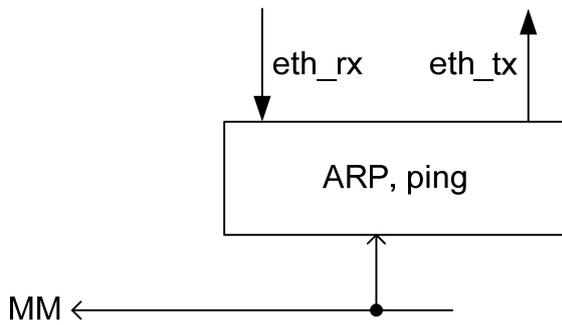


Figuur 5: Block diagram van de ETH10 RTL code

Voor de 1GbE ETH module op een UniBoard FPGA node is de hoofdtaak het transporteren van de UDP node-control frames van en naar de NIOS microprocessor. De ETH module heeft ook een UDP offload poort voor Tx en voor Rx maar die worden nog niet gebruikt. Voor ETH10 zijn juist die UDP offload poorten het belangrijkste, want via de Tx poort kan de streaming data dan over de 10GbE link gestuurd worden zonder tussenkomst van de NIOS microprocessor via het MM interface. De NIOS wordt dan alleen nog gebruikt om de IPv4 link te onderhouden, dus voor ARP en ping.

In de ETH module [4] bestaat het MM interface voor de UDP control pakketten uit een MM Tx packet register en een Rx packet register, alle data is dus direct leesbaar en schrijfbaar voor de NIOS en in willekeurige volgorde. In plaats van dit Tx en Rx packet register interface is het simpeler om een FIFO plus ST-to-MM adapter te gebruiken voor Rx en een MM-to-ST adapter plus FIFO te gebruiken voor Tx, zoals in Figuur 5. Een soortgelijk MM-ST interface passen we ook toe in de DDR3 module [13]. De Rx FIFO heeft een DP flusher aan de input nodig die zorgt dat er steeds maar 1 pakket de FIFO in gaat. De Tx FIFO moet eerst gevuld worden met een pakket voordat het verstuurd kan worden, omdat de NIOS het versturen niet bij kan houden en er anders dan dus gaten in het pakket ontstaan wat niet mag gebeuren.

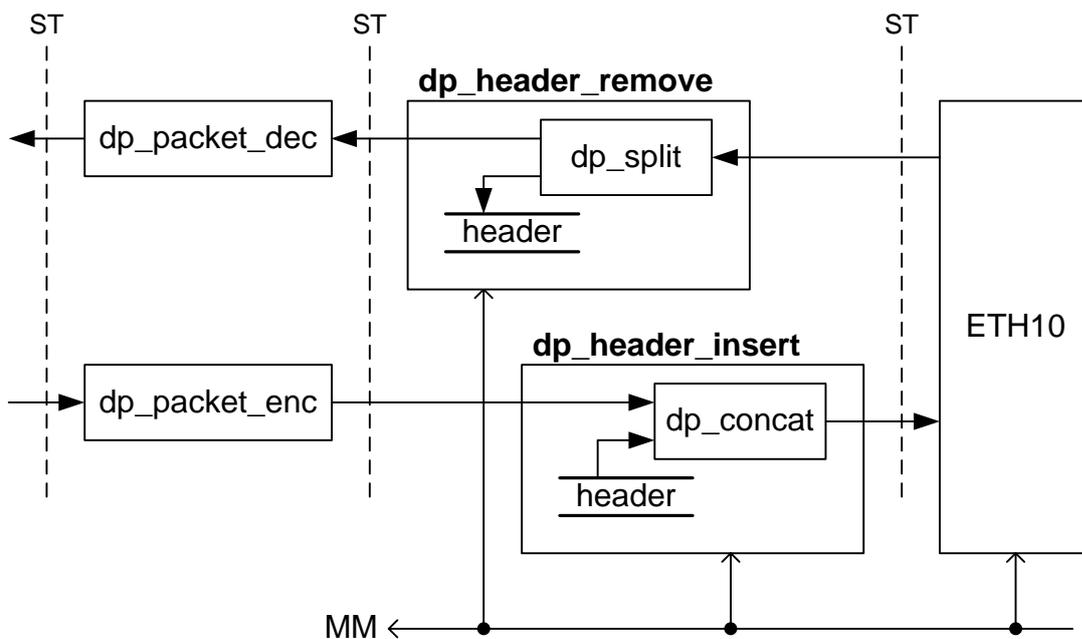
Als de NIOS dan ARP en ping [16] verzorgt voor de 10GbE linken, dan moet dat als extra software taak aan *unbos* toegevoegd worden, zodat *unbos* ook 1, 2 of 3 ETH10 linken kan onderhouden. De huidige *unbos* software verzorgt al ARP en ping voor de 1GbE link, dus de benodigde functionaliteit is bekend. Alternatief is om de ETH10 link control taak in VHDL te implementeren, deze VHDL component kan dan direct aansluiten op de demux en mux in Figuur 5 dus zonder de FIFOs, zie Figuur 6. Voordeel is dan dat we *unbos* hiermee niet belasten en ook niet hoeven aan te passen.



Figuur 6: ARP and ping control in firmware

2.2.1 UDP offload

Figuur 7 laat zien hoe de UDP offload poort van de ETH10 module gebruikt gaan worden. De UDP offload poort is een enkele streaming poort, maar dankzij het SOSI channel veldje kunnen toch meerdere UDP stromen via deze offload poort bediend worden. Het UDP-port nummer mapt hierbij op het SOSI channel nummer.



Figuur 7: Stream pakket codering en decodering voor de UDP offload poort

Om de streaming data aan de UDP offload poort aan te kunnen bieden hebben we twee extra functies nodig:

- Een component die de streaming data in de payload van een UDP pakket kan stoppen en er een ETH/IP/UDP header voor plakt. Die ETH/IP/UDP header is schrijfbaar via de MM bus en wordt vervolgens voor ieder pakket geplakt, zodra er weer een blok streaming data klaar zijn voor Tx. Zonodig kan de header ook nog uitgebreid worden met applicatie specifiek deel. Een blok streaming data wordt gemarkeerd door de sop en de eop. Het aan elkaar plakken van een header en een payload kan mbv *dp_header_insert* en het opsplitsen van een header en een payload kan mbv *dp_header_remove*.
- Om de streaming control informatie over te sturen (dwz channel nummer, BSN, sync, en error status) is het nodig om de streaming data eerst in te pakken. Dit inpakken gebeurt met *dp_packet_enc* en het uitpakken gebeurt met *dp_packet_dec* [11].

De *dp_header_insert* en *dp_header_remove* componenten staan los van de ETH10 module, omdat ze ook voor bijvoorbeeld de UDP offload poorten van de ETH module [4] gebruikt kunnen worden. Daarom horen deze componenten in de DP module [15]. De componenten moeten geschikt zijn voor gebruik in combinatie met de ETH module en met de ETH10 module, het verschil zit in de data woordbreedte van respectievelijk 32 bit en 64 bit. Deze componenten hebben we nog niet, maar kunnen gebruik maken van *dp_concat* en *dp_split* zoals aangegeven in Figuur 7.

3 Implementatie specificatie

3.1 TR_XAUI module

De TR_XAUI module komt in deze directory in de UniBoard SVN repository:

\$UNB/Firmware/modules/tr_xaui/

De TR_XAUI module is vergelijkbaar met de *tr_nonbonded* module [12]. De *tr_nonbonded* module biedt de mogelijkheid om de 12 transceivers aan de mesh kant in de FN en BN FPGA of aan de backplane kant in de BN FPGA onafhankelijk van elkaar te gebruiken. De TR_XAUI biedt de mogelijkheid om de 12 transceivers aan de frontpanel kant van de FN of aan de backplane kant van de BN als 3 XAUI bundels van 4 transceivers elk te gebruiken. Dankzij het XAUI protocol werken deze 4 transceivers dan als 1 samengestelde seriële link, echter wel beperkt tot 10 Gbps. De TR_XAUI module is waarschijnlijk eenvoudiger dan *tr_nonbonded*, omdat de XAUI IP het opzetten van de link zelf al verzorgt. In het simpelste geval bestaat de TR_XAUI module alleen uit *g_nof_xaui* instanties van de XAUI IP en de MDIO.

3.1.1 Reference design unb_tr_xaui

Het *unb_tr_xaui* design in Figure 8 is qua opzet gelijk aan het design *unb_tr_nonbonded* [12]. Het design omvat de TR_XAUI module te samen met de test functionaliteit uit de *diagnostics* module, de standaard componenten uit *unb_common* en een SOPC Builder system. Zo vormt *unb_tr_xaui* een compleet design dat werkt op een FPGA en bestaande uit:

- *ctrl_unb_common* → met o.a. de TSE
- *sopc_unb_tr_xaui* → met o.a. de NIOS en de MM bus
- *node_unb_tr_xaui* → met de DUT

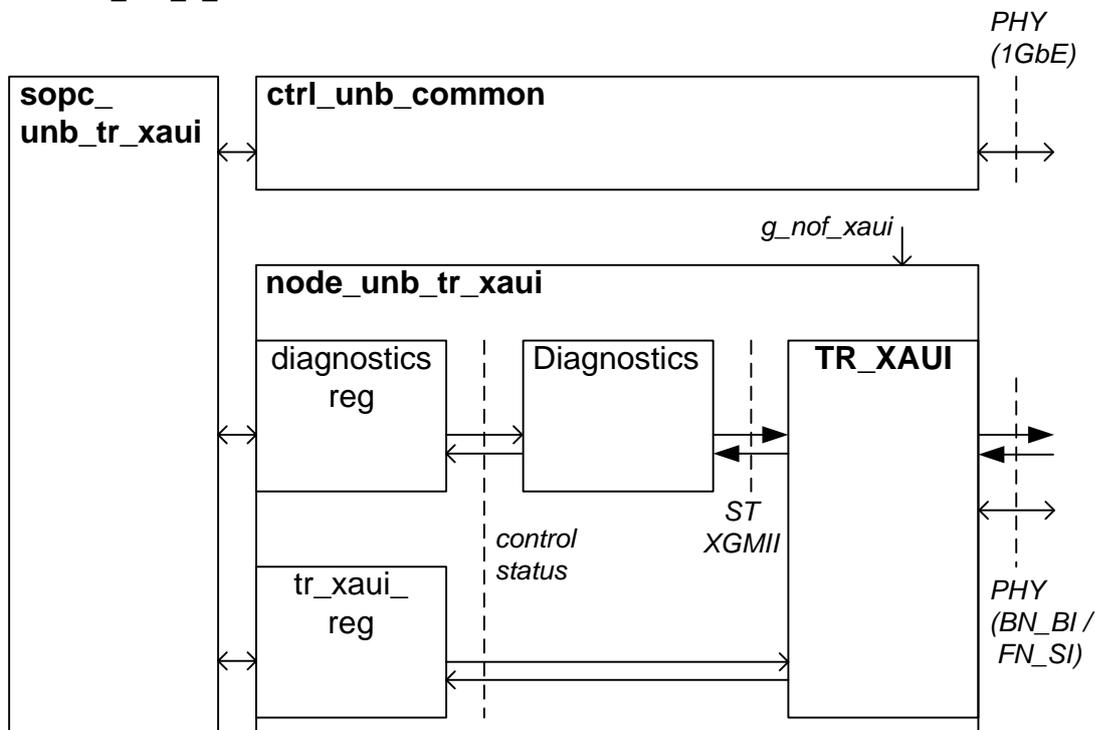
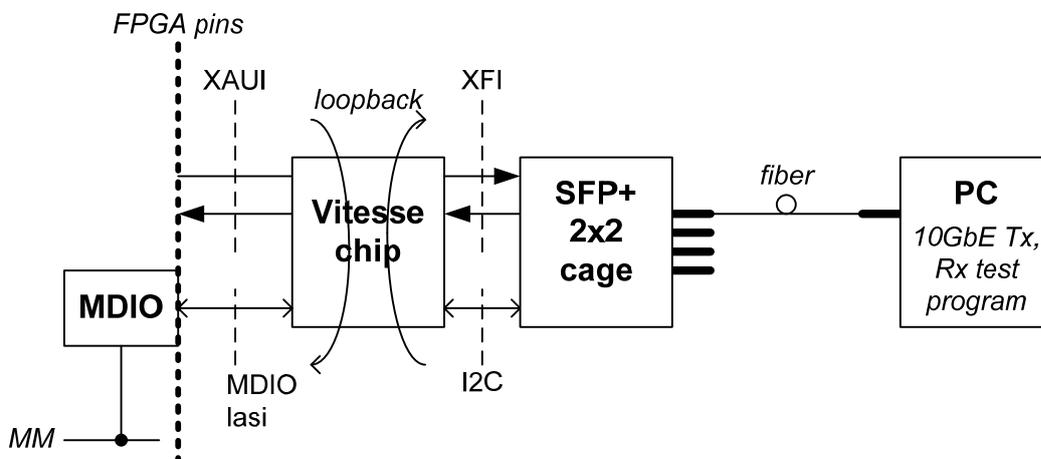


Figure 8: Design unb_tr_xaui

Aangezien op de back nodes ook 12 transceivers beschikbaar zijn en dus 3 XAUI poorten is het mogelijk om *unb_tr_xaui* zowel op de FN als op de BN te draaien en daarom heet het *unb_tr_xaui* (en niet *fn_tr_xaui*). Op de FN kan mbv het MDIO interface de Vitesse PHY chip nader ingesteld worden. Op de BN wordt het MDIO interface niet aangesloten, omdat in dat geval de communicatie op XAUI niveau gaat, bijvoorbeeld via CX4 kabels via XGB of via een backplane naar een BN op een ander UniBoard.

3.1.2 PHY loopback

Voor het valideren van de communicatie tussen een FN en een netwerkkaart in een PC is het voldoende om de 10G PHY op UniBoard in de Vitesse chip terug te koppelen. Het FN-Vitesse PHY interface wordt namelijk middels de TR_XAUI 10G linken tussen FN - FN al gevalideerd. Figuur 9 toont een blokdiagram een 10G link test tussen een 10GbE netwerkkaart in een PC en de Vitesse PHY chip in loopback. De aanname is dat op de PC een test programma draait om 10GbE data verkeer te genereren en te monitoren of alle pakketten foutloos weer ontvangen worden. Om de Vitesse PHY chip in loopback te kunnen zetten is een FPGA design nodig met daarin de MDIO VHDL firmware module.



Figuur 9: Opzet voor 10GbE test tussen FN – PC netwerkkaart middels Vitesse PHY loopback

3.2 ETH10 module

De ETH10 module komt deze directory in de UniBoard SVN repository:

\$UNB/Firmware/modules/eth10/

De volgende secties duiden enkele specifieke implementatie aspecten en het benodigde reference design.

3.2.1 MAC woordbreedte

Groot verschil is dat intern de ETH module met 32b data woordbreedte werkt (dus 4 ethernet octets per woord) terwijl de ETH10 module met 64b data moet werken (dus 8 ethernet octets per woord). De componenten in Figuur 5 zijn al aanwezig in de ETH module, deze moeten dus omgezet worden naar soortgelijke componenten in de ETH10 module maar dan geschikt voor 64b data breedte.

3.2.2 Payload woordalignment

Het blijkt dat de totale ethernet header voor al de pakketten steeds 42 octets lang is [4]:

- ETH/ARP → 14 + 28 = 42
- ETH/IP/UDP → 14 + 20 + 8 = 42
- ETH/IP/ICMP → 14 + 20 + 8 = 42

De ETH module werkt met 32b woorden, dus de totale ethernet header past in 11 woorden. De TSE MAC IP heeft een optie die we ook gebruiken om de header te alignen op de 32b woordgrens door er 2 octets voor aan toe te voegen. Dit heeft dan als voordeel dat de header in 11 woorden past en dat de payload data precies op de woordgrens van het 12e woord begint. De ETH10 MAC heeft zo'n woord alignment optie niet, maar ook daar is het wenselijk om de payload op een 64b woord grens te laten vallen. Mbv *dp_pad_insert* (op basis van *dp_concat*) kunnen we 6 octets aan de header toevoegen om deze op 48 octets = 6 woorden te krijgen voor Rx en mbv *dp_pad_remove* (op basis van *dp_split*) kunnen we die 6 octets weer verwijderen voor Tx. Testbench *tb_dp_pad_insert_remove* laat zien hoe en dat dit werkt.

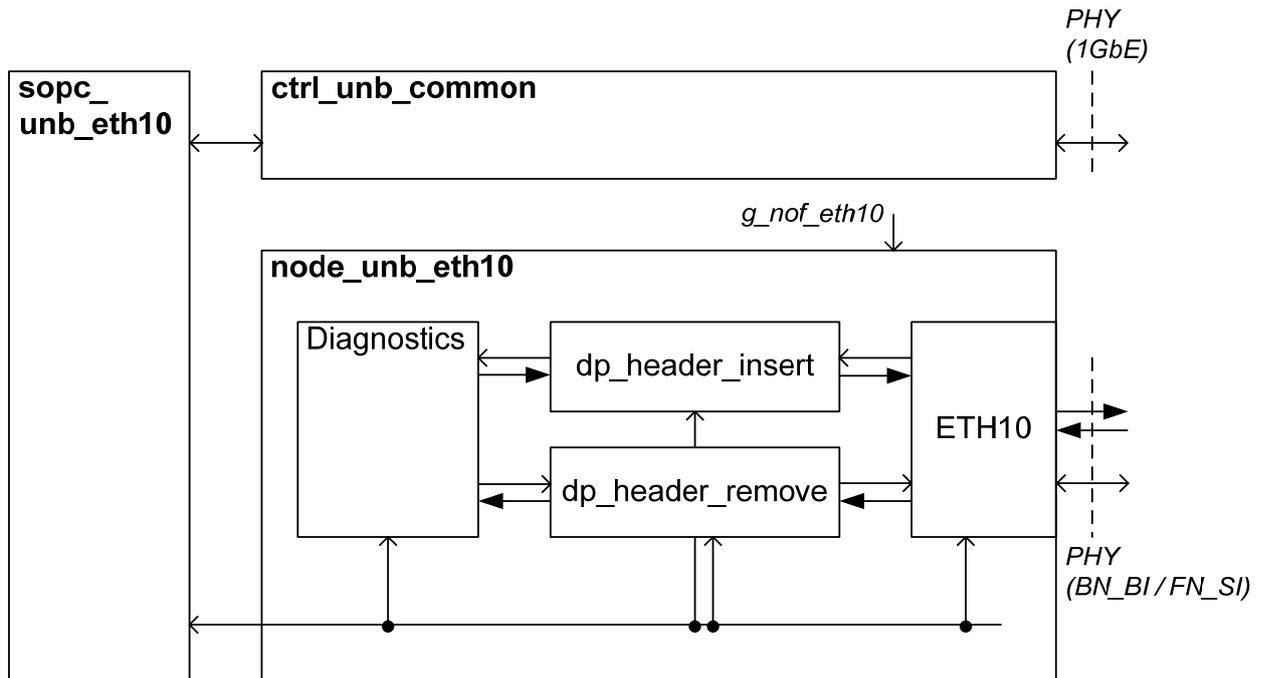
3.2.3 Header mapping

In de ETH module implementatie wordt her en der steeds de header opgeslagen in een 11 word buffer (11 x 32b = 44 octets). Mbv map functies, die in wezen resulteren in alleen draden, wordt deze header data dan geïnterpreteerd als ARP, of als IP/UDP of als IP/UDP/ICMP (ping). Dit is een mooie manier van werken. Zelfde idee moeten we ook gebruiken voor ETH10 maar dan hebben we 6 woorden nodig (6 x 64b = 48 octets).

3.2.4 Reference design unb_eth10

Een reference design voor de ETH10 module is ook toepasbaar op de back nodes, daarom heet het *unb_eth10* (en niet *fn_eth10*). Figuur 10 toont het reference design *unb_eth10* met *g_nof_eth10* = 1, 2, of 3. Zo vormt *unb_eth10* een compleet reference design dat werkt op een FPGA en bestaande uit:

- *ctrl_unb_common* → met o.a. de TSE
- *sopc_unb_eth10* → met o.a. de NIOS en de MM bus
- *node_unb_eth10* → met de DUT



Figuur 10: Design unblock_eth10

3.3 Wrappers

De MAC10 IP en de XAUI IP moeten omvat worden door respectievelijk een `mac10.vhd` en een `phy_xaui.vhd` VHDL wrapper component [14]. Deze VHDL wrapper zorgt bijvoorbeeld voor het mappen van de standaard logic IO van de IP op de SOSI en SISO record IO. Vergelijk de `tse.vhd` wrapper voor de 1GbE TSE IP in de ETH module.

4 Test specificatie

4.1 Verificatie van TR_XAUI

4.1.1 Testbench *tb_phy_xaui* voor de XAUI IP

De *tb_phy_xaui* is nodig omdat het rechtstreeks de XGMII IO van de XAUI IP aanstuurt. Hierdoor is er volledige vrijheid in het zetten van de stimuli.

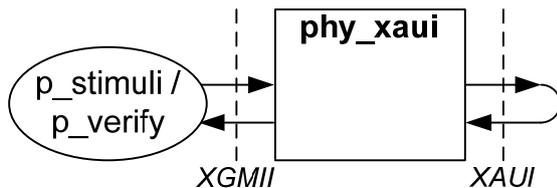
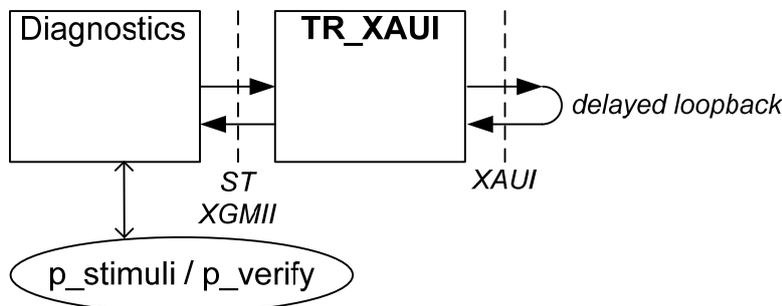


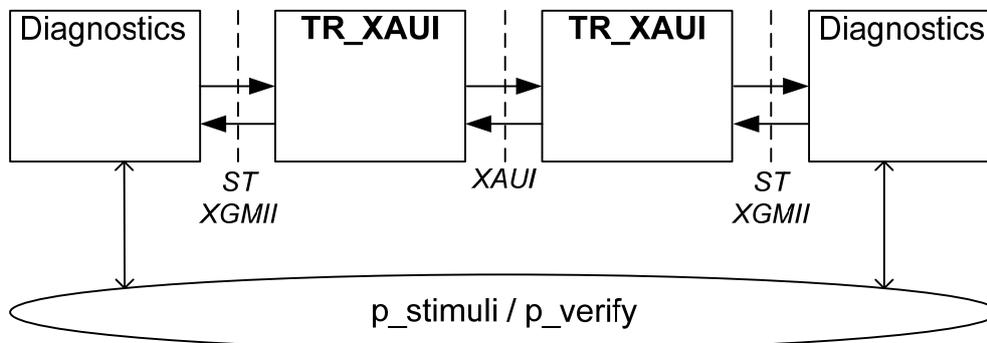
Figure 11: *tb_phy_xaui* met *phy_xaui* loopback

4.1.2 Testbench *tb_tr_xaui* for TR_XAUI module

De *tb_tr_xaui* kan simpelweg de Tx van het XAUI interface in loop back aansluiten op de Rx met een delay in VHDL ($Rx \leftarrow TRANSPORT Tx \text{ AFTER } 0 \text{ ps}$) zoals in Figuur 12 of eventueel kan *tb_tr_xaui* twee instanties van TR_XAUI gebruiken zoals in Figuur 13 (vergelijkbaar met *tb_tr_nonbonded* [12]). In beide gevallen moet de test een tijdje simuleren en dan self-checkend aangeven of er niks fout is gegaan en of de test wel gebeurd is.



Figuur 12: *tb_tr_xaui* met TR_XAUI loopback



Figuur 13: *tb_tr_xaui* alternatief met twee TR_XAUI instanties voor end-to-end test.

4.1.3 Testbench `tb_tr_xaui_mdio` voor TR_XAUI met MDIO

Figure 14 toont de testbench `tb_tr_xaui_mdio` met de MDIO in de TR_XAUI en een model van een MDIO slave in VHDL. Dit MDIO slave model modelleert het MDIO gedrag van de Vitesse PHY chip. De MDIO firmware module bevat al een model van een MDIO slave. Dit model kan een relevant Vitesse PHY register bevatten welke via de MDIO bus geschreven en gelezen kan worden. Het is dus beslist niet de bedoeling dat XAUI gedrag van de Vitesse PHY chip gemodelleerd wordt, het gaat hier alleen om het MDIO interface. De `tb_tr_xaui_mdio` testbench verzekert zo dat we de Vitesse chip in kunnen stellen.

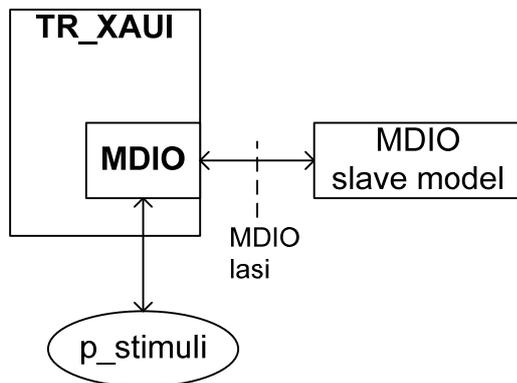


Figure 14: `tb_tr_xaui_mdio`

4.1.4 Testbench `tb_node_unb_tr_xaui`

De `tb_node_unb_tr_xaui` dient om de component `node_unb_tr_xaui` te verifiëren ter voorbereiding op de integratie van `node_unb_tr_xaui` in design `unb_tr_xaui`, zie Figure 8.

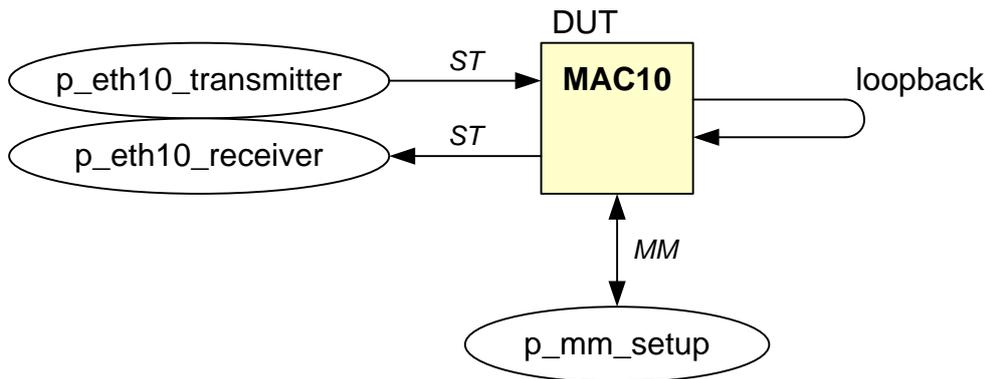
4.1.5 Testbench `tb_unb_tr_xaui`

De `tb_unb_tr_xaui` dient om design `unb_tr_xaui`, zie Figure 8, te verifiëren met een NIOS `main.c` software applicatie ter voorbereiding op de validatie van het `unb_tr_xaui` design op hardware. De NIOS `main.c` applicatie doet hetzelfde als wat voor `tb_node_unb_tr_xaui` in de vorm van VHDL processen wordt gedaan. De `main.c` is een relatief simpel C applicatie programma dat de middels de `diagnostics` component en de XAUI PHY in loopback het `unb_tr_xaui` design verifieert in simulatie en valideert op hardware.

4.2 Verificatie van ETH10

4.2.1 Testbench `tb_mac10` voor de MAC10 IP

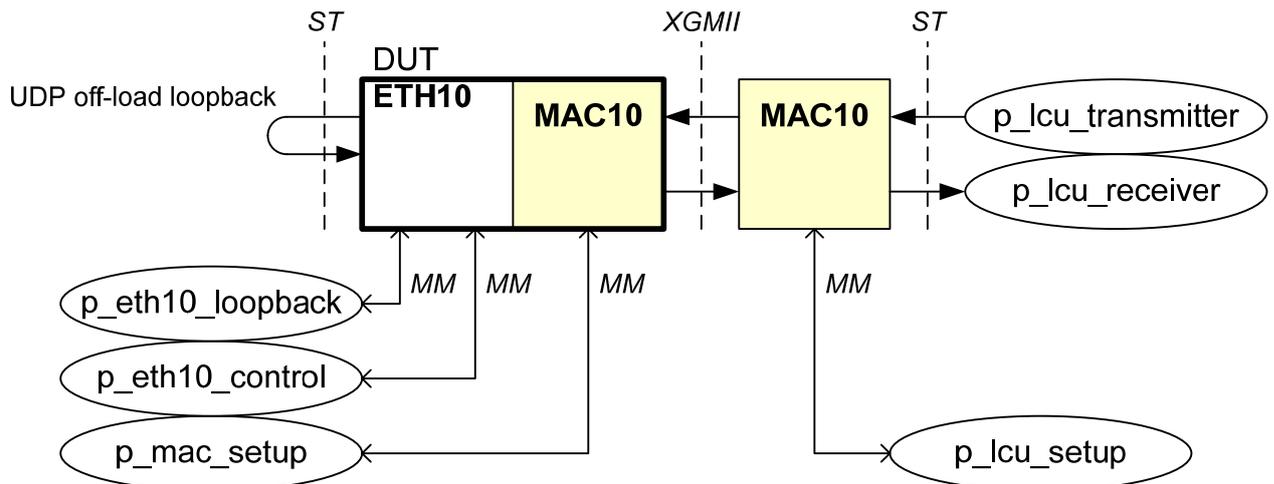
De `tb_mac10` testbench dient om de MAC10 IP te simuleren en lijkt op `tb_tse` zoals gebruikt voor de 1GbE TSE IP [4]. Na eerst een voorbeeld testbench of reference design van Altera te hebben gesimuleerd, dient deze `tb_mac10` testbench om de essentie van de MAC10 qua ST en MM interface te vatten in een compacte, meer generieke testbench. Net als met `tb_phy_xaui` grijpt ook `tb_mac10` direct in op de IO van de MAC10 IP om zo maximale vrijheid te hebben in het toedienen van de stimuli. Met testbench `tb_mac10` is het bijvoorbeeld eenvoudig om verschillende pakket groottes en inter-pakket gaps te simuleren of om diverse MM control settings van de MAC10 IP uit te proberen.



Figuur 15: `tb_mac10`

4.2.2 Testbench `tb_eth10` voor de ETH10 module

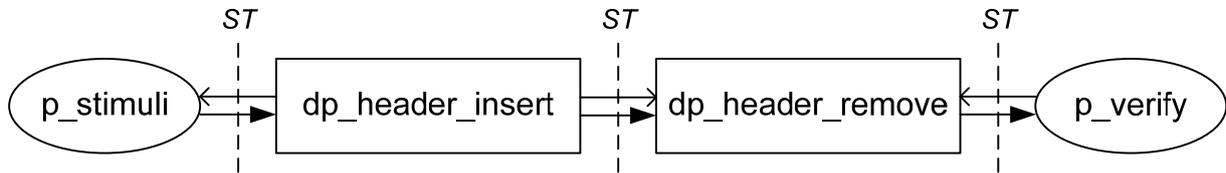
De `tb_eth10` testbench dient om de ETH10 module te simuleren en lijkt op `tb_eth` zoals gebruikt voor de 1GbE TSE IP [4]. Het genereren van Ethernet pakketjes en het bedienen van het MM interface in de testbench procedures en processen vergemakkelijkt het manipuleren van de stimuli. Met deze `tb_eth10` testbench kunnen ook ARP en ping geverifieerd worden.



Figuur 16: `tb_eth10`

4.2.3 Testbench `tb_dp_header_insert_remove`

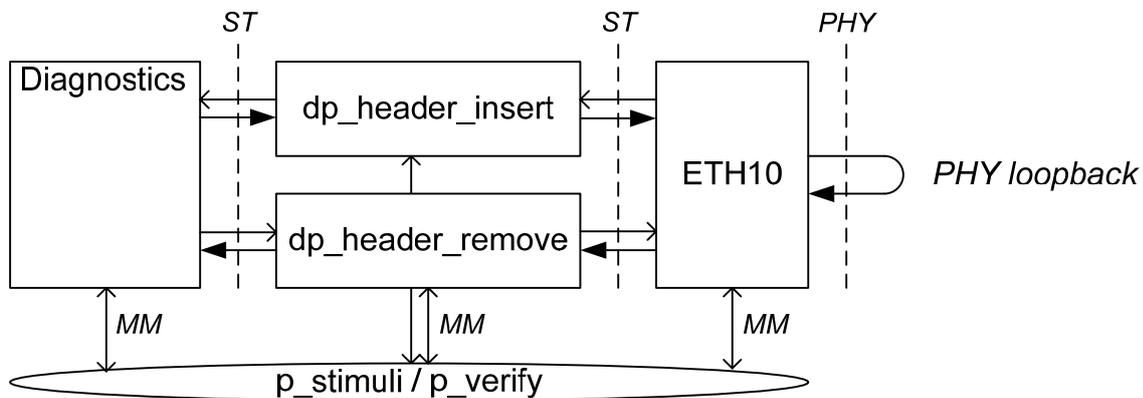
De testbench `tb_dp_header_insert_remove` lijkt op `tb_dp_pad_insert_remove` en dient om de functies die nodig zijn voor de UDP offload poort eerst los te verifiëren. Deze testbench maakt gebruik van de verify procedures uit `tb_dp_pkg.vhd` [15] om deze componenten met backpressure flow control te kunnen testen.



Figuur 17: `tb_dp_header_insert_remove`

4.2.4 Testbench `tb_eth10_udp_offload` voor UDP offload

De `tb_eth10_udp_offload` testbench in Figuur 18 dient om de UDP-port instelmogelijkheden van ETH10 te verifiëren.



Figuur 18: `tb_eth10_udp_offload`

4.2.5 Testbench `tb_node_unb_eth10`

De `tb_node_unb_eth10` testbench dient om het `node_unb_eth10` gedeelte in `unb_eth10` design eerst los te simuleren mbv VHDL process stimuli.

4.2.6 Testbench `tb_unb_eth10`

De `tb_unb_eth10` testbench dient om het design `unb_eth10` met een NIOS `main.c` software applicatie te simuleren ter voorbereiding op validatie op hardware. De ETH10 kan in loopback aangesloten worden in de testbench.

4.3 Validatie

De validatie van de TR_XAUI module en van de ETH10 module behelst de testen op de UniBoard hardware. In eerste instantie korte testen, maar uiteindelijk stress testen van ten minste 24 uren. Gedurende zo'n stress test mag er geen data verloren gaan en mogen er ook geen bitfouten optreden.

De testen omvatten Tx en Rx verkeer in de volgende situaties:

- tussen BN via XGB en CX4 kabels
- tussen FN via fiber
- tussen FN en PC netwerk kaart via fiber

De testen tussen FN kunnen al op XAUI niveau gevalideerd worden op een vergelijkbare manier als gedaan is voor *tr_nonbonded* op de back nodes [12]. Aan de hand van deze testen kunnen de instellingen voor de Vitesse PHY chips gevalideerd worden (middels het MDIO interface).

Voor de testen tussen FN en PC netwerk kaart is alleen de MDIO functie binnen de TR_XAUI module al voldoende, omdat hiermee de Vitesse PHY chip in loopback gezet kan worden, zoals aangegeven in Figuur 9. In dat geval moet de PC de Ethernet pakketjes genereren en via de loopback in de Vitesse PHY chip weer ontvangen.

Ten slotte moeten de stress tests gedaan worden met Ethernet pakketjes gegenereerd door de ETH10 module op de FN.

5 Stappenplan

Dankzij de v9.1 eth10g module en ARP en ping functionaliteit in het *fn_base* BIST design weten we dat de 10GbE interfaces op de front nodes van UniBoard werken. Het *fn_base* design is zien als een verkennend design dat relatief snel resultaat heeft bereikt. Met de TR_XAUI module en de ETH10 module willen we de 10G interfaces van UniBoard robuust en onderhoudbaar maken zodat we er plug-en-play gebruik van kunnen maken in onze applicaties zoals APERTIF. Dit stappenplan zorgt dat we alle facetten van de TR_XAUI module en de ETH10 module onder de knie krijgen en het geheel in afgebakende stappen opbouwen. Elke stap voegt een stukje toe, niet te veel zodat de ontwikkeling beheersbaar blijft en niet te weinig opdat elke stap toegevoegde waarde heeft. De stapjes zijn verder zo gekozen dat we het eerst doen wat we het eerst nodig hebben in een applicatie. Voor APERTIF is dat TR_XAUI.

5.1 TR_XAUI ontwikkeling

- *tr_xaui* simuleren mbv VHDL stimuli op alle niveaus van XAUI IP t/m FPGA test design
- *unb_tr_xaui* design tussen BN via XGB en CX4 kabels
- *unb_tr_xaui* design inclusief MDIO tussen FN via glasvezel
- *unb_tr_xaui* design inclusief MDIO en met Vitesse PHY in loopback zodat de PC de 10G link kan doorfluiten mbv 10GbE data verkeer
- TR_XAUI Module Description document

5.2 UDP offload ontwikkeling

- *dp_header_insert.vhd* en *dp_header_remove.vhd* en testbench

5.3 ETH10 ontwikkeling

- MAC10 IP reference design simuleren om te leren hoe het werkt
- MAC10 IP wrappen en simuleren in een eigen testbench en Tx en Rx procedures
- ETH10 module implementeren en simuleren mbv Tx en Rx procedures
- ETH10 module mbv synthetiseerbare UDP offload data simuleren
- *unb_eth10* design samenstellen en simuleren met eenvoudige app/*unb_eth10/main.c* en loopback
- *unb_eth10* design op UniBoard met eenvoudige app/*unb_eth10/main.c* tussen wee FN
- *unb_eth10* design op UniBoard met *unbos* en instellen mbv Python script
- *unb_eth10* design stress test tussen FN via glasvezel
- *unb_eth10* design stress test tussen FN en een PC via glasvezel
- ETH10 Module Description document

6 Eindresultaat

De belangrijkste resultaten zijn de twee 10G modules:

- TR_XAUI module geschikt voor 1, 2 of 3 links inclusief een module description document
- ETH10 module geschikt voor 1, 2 of 3 links inclusief een module description document

Indirect zijn hiervoor ook de volgende nieuwe DP componenten nodig:

- DP module componenten *dp_header_insert.vhd* en *dp_header_remove.vhd*, nodig om streaming data via een UDP offload poort van de ETH module en de ETH10 module te versturen

Mbv de TR_XAUI module dient het UniBoard FN_SI interface gevalideerd te worden voor links tussen FN en voor een link tussen FN en een PC.

7 Appendix: Verificatie en validatie methode

Voor de TR_XAUI en de ETH10 interface modules gebruiken we een verificatie en validatie methode die werkt van detail niveau naar applicatie niveau. Op het detail niveau kan snel gesimuleerd worden en is er veel vrijheid in het manipuleren van de stimuli en op het applicatie niveau wordt de DUT in normale mode gebruikt. Dankzij de specifieke *main.c* op de NIOS (en de JTAG_UART) kan het reference design voor de module zowel in simulatie als op hardware getest worden. Met behulp van *unbos* op de NIOS en een Python script op de PC kan hetzelfde (of meer) getest worden inclusief de control keten via het 1GbE interface (dus zonder de JTAG_UART). Ten slotte kunnen op hardware stress testen gedaan worden door een test vaak te herhalen of langer te laten duren.

1. Testbench voor de low level IP (XAUI, MAC10) met process voor stimuli en verificatie
2. Module (TR_XAUI, ETH10) testbench met process voor stimuli en verificatie
3. Module (TR_XAUI, ETH10) testbench met synthetiseerbare diagnostics test data
4. Node testbench met VHDL stimuli
5. Design testbench met *main.c* software stimuli
6. Validatie van design op FPGA met *main.c* software stimuli
7. Validatie met *unbos main.c* en een Python stimuli script