

Data Path Packet Module Description

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

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

UniBoard

DESP

Doc.nr.: ASTRON-RP-873

Rev.: 0.2

Date:

Class.: Public

Distribution list:

Group:	Others:
Andre Gunst (AG, ASTRON) Eric Kooistra (EK, ASTRON) Daniel van der Schuur (DS, ASTRON) Harm-Jan Pepping (HJP, ASTRON)	Gijs Schoonderbeek (GS, ASTRON) Jonathan Hargreaves (JH, JIVE) Salvatore Pirrucci (SP, JIVE)

Document history:

Revision	Date	Author	Modification / Change
0.1	2011-11-19	Eric Kooistra	Draft.
0.2	2012-06-29	Eric Kooistra	Redefined SOSI frame sync. Added g_use_sosi_err for dp_packet_dec. Added dp_packet_enc_channel_lo component. Added dp_packet_dec_channel_lo component.

Table of contents:

1	Introduction.....	4
1.1	Purpose	4
1.2	Scope	4
2	Interface, design and implementation of the components	5
2.1	dp_packet_enc – Encode DP SOSI to DP packet	5
2.2	dp_packet_dec – Decode DP packet to DP SOSI	7
2.3	dp_packet_enc_channel_lo - Encode channel low bits into the CHAN field.....	9
2.4	dp_packet_dec_channel_lo - Decode channel low bits from the CHAN field.....	10
3	Verification of the components.....	11
3.1	DP packet level test bench.....	11
3.2	PHY level test bench	12

Terminology:

BSN	Block Sequence Number
CHAN	Channel
CRC	Cyclic Redundancy Check
DP	Data Path
eop	end of packet
ERR	Error
ETH	Ethernet
LSBit	Least Significant bit
MSBit	Most Significant bit
PHY	Physical interface
RL	Ready Latency
SFD	Start of Frame Delimiter
SISO	Source in Sink Out
sop	start of packet
SOSI	Source Out Sink In
TLEN	Type/Length
UTH	Uthernet

References:

1. "Data Path Packet Interface Specification", ASTRON-SP-042, E. Kooistra
2. "Specification for module interfaces using VHDL records", ASTRON-RP-380, E. Kooistra
3. "DP Streaming Module Description", ASTRON-RP-382, E. Kooistra
4. "Data Path Interface Description", ASTRON-RP-394, E. Kooistra
5. "Timing in the Streaming interface", ASTRON-RP-481, E. Kooistra
6. "Uthernet Interface Specification", ASTRON-SP-041, E. Kooistra
7. "Uthernet (UTH) Module Description", ASTRON-RP-871, E. Kooistra

1 Introduction

1.1 Purpose

This document describes the DP packet encoding and decoding part of the data path (DP) module [3]. The DP contains components to encode the SOSI signals into a DP packet and to decode a DP packet into the SOSI signals. The DP module uses the streaming SOSI and SISO records that were defined in [2]. The DP packet has been specified in [1]. The SISO backpressure ready signal is not (yet) supported at DP packet level.

1.2 Scope

The DP packet encoding and decoding components make the similar DP packetizing components that were defined in [4] obsolete.

2 Interface, design and implementation of the components

2.1 dp_packet_enc – Encode DP SOSI to DP packet

2.1.1 Interface

The dp_packet_enc encodes the SOSI signals into a DP packet as shown below in respectively Figure 1 and Figure 2. The DP packet structure with the CHAN, BSN, DATA and ERR fields has been specified in [1].

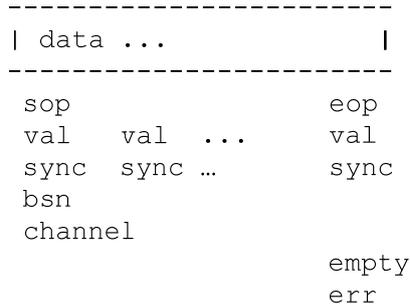


Figure 1: dp_packet_enc snk_in

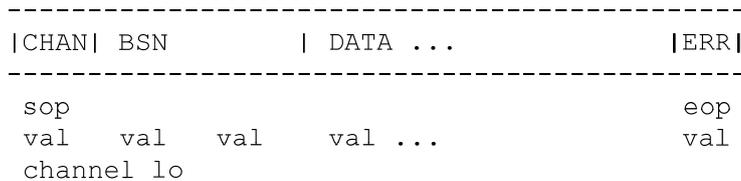


Figure 2: dp_packet_enc src_out

The dp_packet_enc interface parameters and ports are given in respectively Table 1 and Table 2.

Generic	Type	Description
g_data_w	natural	>= 1, the data width of the packet
g_channel_lo	natural	>= 0, the snk_in.channel[g_channel_lo-1:0] bits are not encoded into the CHAN field, but passed on to src_out.channel instead. This allows using these LSbit channel fields to define different groups of DP packet streams.

Table 1: dp_packet_enc parameters

Signal	IO	Type	Description
rst	IN	std_logic	Reset
clk	IN	std_logic	Clock
snk_out	OUT	t_dp_siso	SISO: ready
snk_in	IN	t_dp_sosi	SOSI: sync, bsn, data, valid, sop, eop, channel, empty, err
src_in	IN	t_dp_siso	SISO: ready
src_out	OUT	t_dp_sosi	SOSI: data, valid, sop, eop, channel_lo

Table 2: dp_packet_enc

2.1.2 Design

The CHAN, BSN and ERR field have a specified fixed width of respectively 16 bit, 48 bit and 16 bit. The packet data width is set by g_data_w and can have any value ≥ 1 . Therefore the number of packet data words that is needed to encode the CHAN, BSN and ERR field depends on g_data_w . For example for $g_data_w = 16$ the CHAN takes 1 data word, BSN takes 3 data words and the ERR field takes 1 data word [1]. If the number of data words in the input data block marked by the $snk_in.valid$, $snk_in.sop$ and $snk_in.eop$ is N then the total src_out output packet will have $N+5$ data words.

The $snk_in.channel$ field is read at the $snk_in.sop$. The LSBits $snk_in.channel[g_channel_lo-1:0]$ are passed on directly to $src_out.channel$. The rest of the MSBits of $snk_in.channel$ are transported via the CHAN field.

For framed data the $snk_in.sync$ is active during the entire frame (so for all active $snk_in.valid$ from $snk_in.sop$ to $snk_in.eop$) and applies to the next frame [5]. The dp_packet_enc reads the $snk_in.sync$ at the $snk_in.sop$ and transports the $snk_in.sync$ as MSBit of the BSN field. The $snk_in.bsn$ is read at the $snk_in.sop$ and transported via the BSN field. If the BSN field just fits in an integer number of packet data words then the input sync will be transported instead of the MSbit 47 of the $snk_in.bsn$ value.

The $snk_in.data$ is transported via the DATA field. The $snk_in.data$ is copied to $src_out.data$ as it is, so any resizing to g_data_w is left to do by the downstream blocks.

The $snk_in.err$ field is read at the $snk_in.eop$ and transported via the ERR field.

The $snk_in.empty$ is valid at the $snk_in.eop$ but ignored, because it is assumed to be implicitly known per channel. Hence the recipient of the packet will know it based on the CHAN field.

In case the CHAN, BSN and/or ERR field do not fit in an integer number of packet data words (of width g_data_w) then the unused MSbits of the MSword are set to 0.

2.1.3 Implementation

Figure 3 shows the RTL block diagram of the dp_packet_enc implementation. The dp_packet_enc uses dp_hold_input to maintain the input RL of 1 and the output RL of 1 [3].

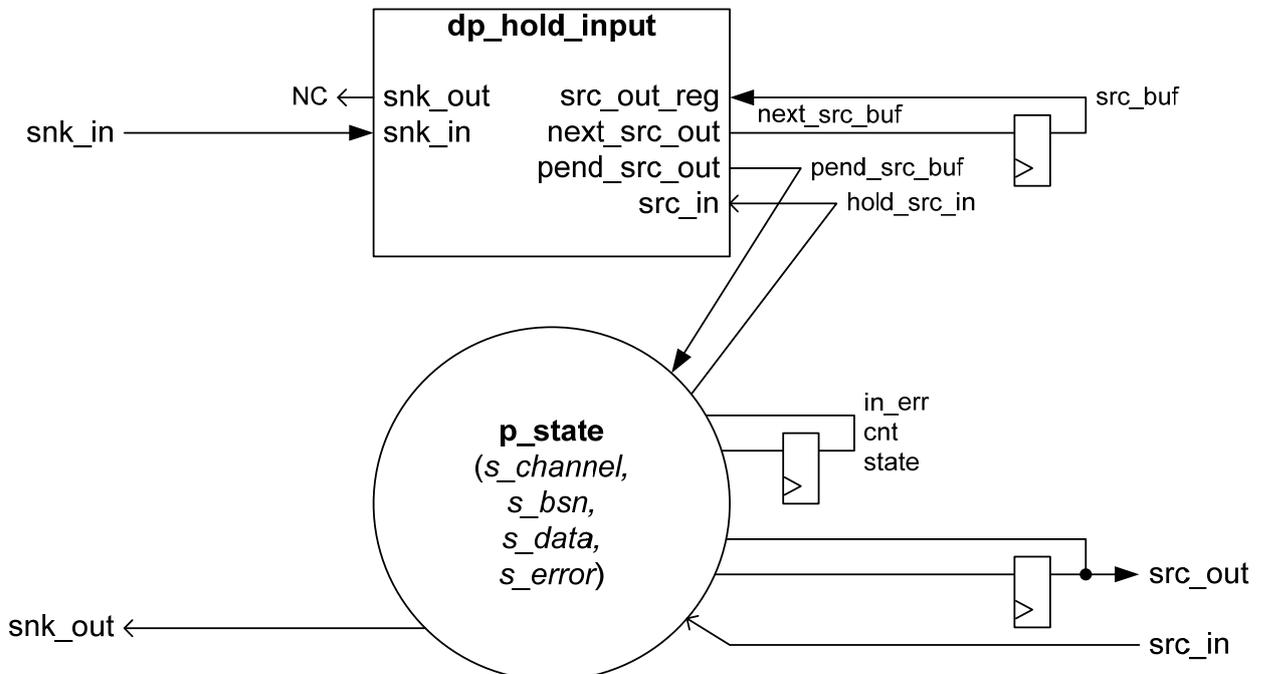


Figure 3: dp_packet_enc implementation

2.2 dp_packet_dec – Decode DP packet to DP SOSI

2.2.1 Interface

The `dp_packet_dec` decodes the SOSI signals from a DP packet as shown below in respectively Figure 4 and Figure 5.

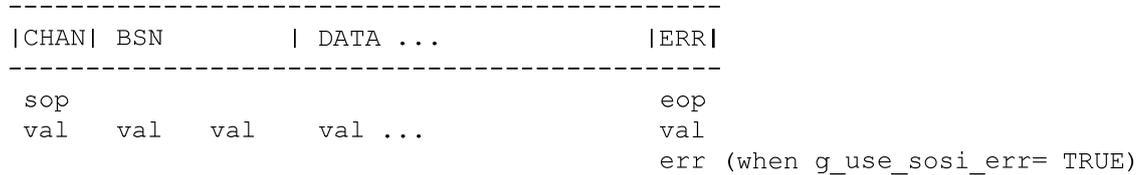


Figure 4: `dp_packet_dec snk_in`

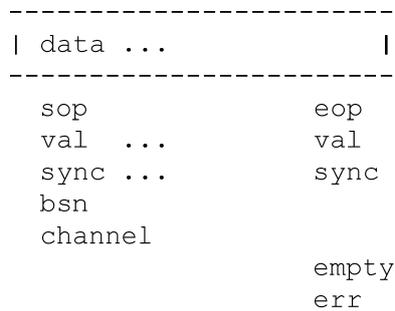


Figure 5: `dp_packet_dec src_out`

The `dp_packet_enc` interface parameters and ports are given in respectively Table 1 and Table 2.

Generic	Type	Description
<code>g_data_w</code>	natural	≥ 1 , the data width of the packet
<code>g_channel_lo</code>	natural	≥ 0 , the <code>snk_in.channel[g_channel_lo-1:0]</code> bits are not encoded into the CHAN field, but passed on to <code>src_out.channel</code> instead. This allows using these LSbit channel number to define different groups of DP packet streams.
<code>g_use_sosi_err</code>	boolean	When the DP packet comes from a PHY link then it may have gotten corrupted contents so then it is useful to set <code>g_use_sosi_err</code> to TRUE to get the error status from the PHY link instead of from the received ERR field.
<code>g_use_this_siso</code>	boolean	Default use TRUE for best throughput performance. When FALSE then <code>dp_packet_dec</code> does not need to control <code>snk_out</code> and it is ready when the downstream sink is ready, this may ease achieving timing closure. When TRUE then in addition <code>dp_packet_dec</code> can also be ready when it is receiving inter frame gaps or the header to increase the throughput.

Table 3: `dp_packet_dec` parameters

Signal	IO	Type	Description
rst	IN	std_logic	Reset
clk	IN	std_logic	Clock
snk_out	OUT	t_dp_siso	SISO: ready
snk_in	IN	t_dp_sosi	SOSI: data, valid, sop, eop, channel_lo
src_in	IN	t_dp_siso	SISO: ready
src_out	OUT	t_dp_sosi	SOSI: sync, bsn, data, valid, sop, eop, channel, empty, err

Table 4: dp_packet_dec ports

2.2.2 Design

The DP packet overhead words are stripped.

The *src_out.sync* is derived from the BSN MSbit and output before the *src_out.sop*. The *src_out.bsn* is derived from the BSN field and is valid at the *src_out.sop*.

The *src_out.channel* is derived from the CHAN field and is valid at the *src_out.sop*.

The first *src_out.data* word is the first DATA field word and is also valid at the *src_out.sop*. The last *src_out.data* word is the last DATA field word and is valid at the *src_out.eop*. The *dp_packet_dec* can handle DP packets with only one DATA field word, so then the *src_out.valid*, *src_out.sop* and *src_out.eop* are all active in the same clock cycle.

The *src_out.empty* is set to 0. The assumption is that the downstream recipient of the *src_out* will know the actual empty value based on the *src_out.channel* identifier.

The *src_out.err* is derived from the ERR field if *g_use_sosi_err* = false and is valid at the *src_out.eop*. In case the DP packet was transported over some PHY link with cyclic redundancy check (CRC) protection, then a CRC error at the receiver may be reported into the *src_out.err* signal instead by using *g_use_sosi_err* = true.

2.2.3 Implementation

Figure 6 shows the RTL block diagram of the *dp_packet_dec* implementation. The *dp_packet_dec* uses *dp_hold_input* to maintain the input RL of 1 and the output RL of 1 [3].

The *dp_packet_dec* uses *dp_shiftreg* to strip the packet ERR field words from the *src_out* stream. Therefore the *dp_shiftreg* contains *c_error_len* + 1 number of data words, where *c_error_len* is the number of packet data words used for the ERR field. When the *blk_sosi.eop* is there the process *p_src_out* strips these words by setting the *src_out.eop* at the last payload data word that is just still in the shift register. Together with that *src_out.eop* it also sets the *src_out.err* based on the ERR field value that was aggregated by process *p_src_err*.

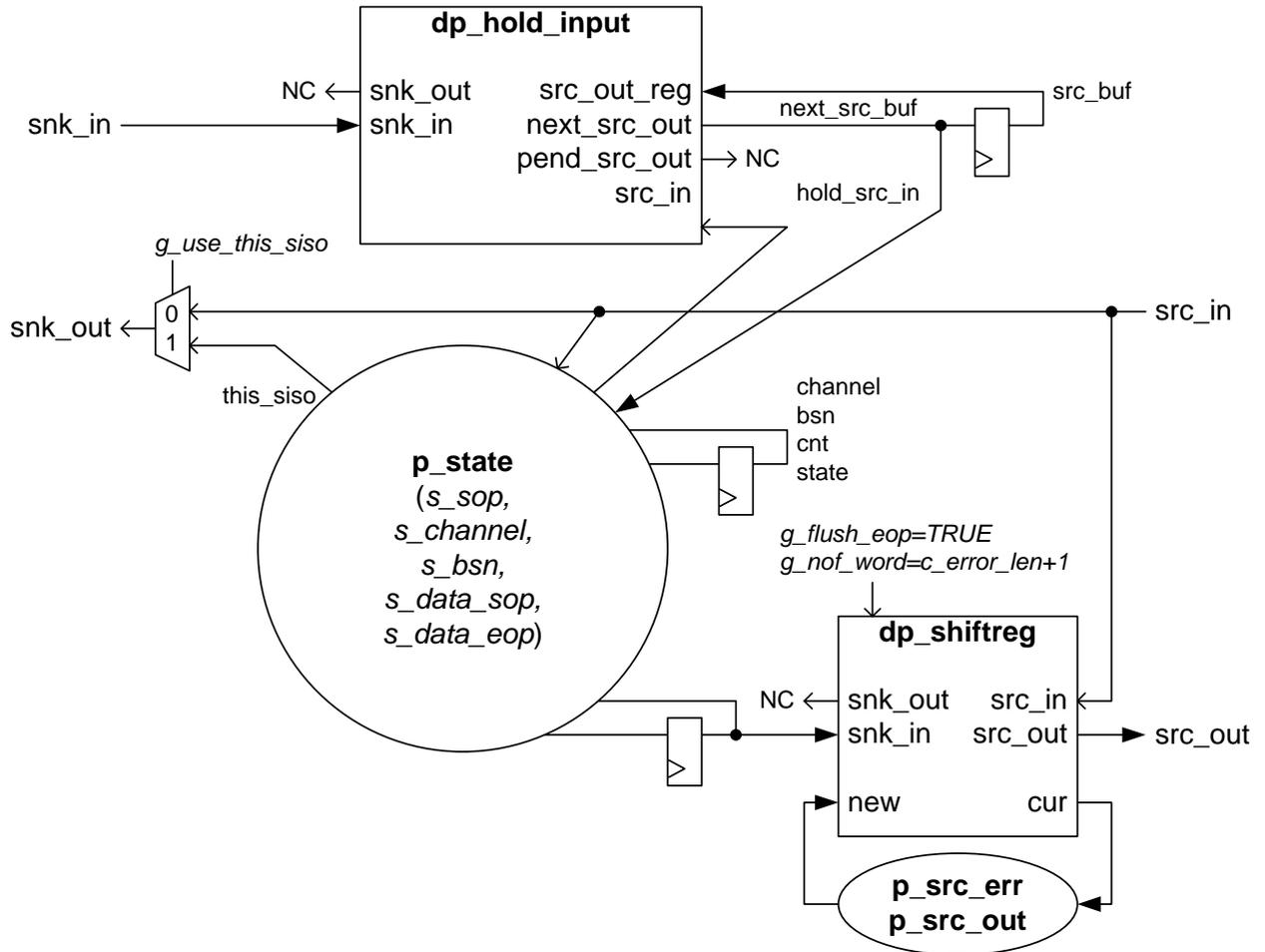


Figure 6: dp_packet_dec implementation

2.3 dp_packet_enc_channel_lo - Encode channel low bits into the CHAN field

2.3.1 Interface

The `dp_packet_enc_channel_lo` encodes the `sosi.channel` low bits into the high part of the CHAN field of a DP packet. The `snk_in.data` must be DP packet data. The `[g_channel_lo-1:0]` bits of the `sosi.channel` field get placed into the `[high-1:high-g_channel_lo]` bits of the CHAN field. The `[high]` bit of the CHAN field is not used because it is reserved for future use. The `dp_packet_enc_channel_lo` interface parameters and ports are given in respectively Table 5 and Table 6.

Generic	Type	Description
<code>g_data_w</code>	natural	≥ 1 , the data width of the packet
<code>g_channel_lo</code>	natural	Number of channel bits. Pre-conditions: - <code>g_channel_lo+1</code> \leq <code>g_data_w</code> - <code>g_channel_lo+1</code> \leq <code>c_dp_packet_channel_w</code> = 16 - <code>g_channel_lo+1</code> \leq <code>c_dp_packet_channel_w</code> - currently used CHAN width

Table 5: dp_packet_enc_channel_lo parameters

Signal	IO	Type	Description
snk_out	OUT	t_dp_siso	SISO: ready
snk_in	IN	t_dp_sosi	SOSI: data, valid, sop, eop, channel_lo
src_in	IN	t_dp_siso	SISO: ready
src_out	OUT	t_dp_sosi	SOSI: data, valid, sop, eop

Table 6: dp_packet_enc_channel_lo ports

2.3.2 Design

The DP packet CHAN field occurs at the sop. By placing the channel_lo bits in the high part of the CHAN field this dp_packet_enc_channel_lo can combinatorially modify the CHAN data at the sop. For small g_data_w the CHAN field can cover multiple data words, therefore using the low part of the CHAN field would imply shifting in the g_channel_lo bits and would require more logic including registers to modify the entire CHAN field. The snk_in channel is passed on unmodified to the src_out.channel, because DP packets typically do not use the channel field, so the g_channel_lo bits are the only valid bits in the channel field and therefore they may as well remain, rather than being shifted out.

2.3.3 Implementation

The dp_packet_enc_channel_lo is a combinatorial component.

2.4 dp_packet_dec_channel_lo - Decode channel low bits from the CHAN field

2.4.1 Interface

The dp_packet_dec_channel_lo decodes the sosi.channel low bits from the high part of the CHAN field of a DP packet. The snk_in.data must be DP packet data. The [high-1:high-g_channel_lo] bits of the CHAN field get set to 0 and placed into the [g_channel_lo-1:0] bits of the sosi.channel field. The dp_packet_dec_channel_lo interface parameters and ports are given in respectively Table 7 and Table 8.

Generic	Type	Description
g_data_w	natural	>= 1, the data width of the packet
g_channel_lo	natural	Number of channel bits.

Table 7: dp_packet_dec_channel_lo parameters

Signal	IO	Type	Description
rst	IN	std_logic	Reset
clk	IN	std_logic	Clock
snk_out	OUT	t_dp_siso	SISO: ready
snk_in	IN	t_dp_sosi	SOSI: data, valid, sop, eop
src_in	IN	t_dp_siso	SISO: ready
src_out	OUT	t_dp_sosi	SOSI: data, valid, sop, eop, channel_lo

Table 8: dp_packet_dec_channel_lo ports

2.4.2 Design

The sosi.channel field must be valid during the entire frame, so not only at the sop. Therefore it is necessary to use channel_lo_hold and to have a clock and reset.

2.4.3 Implementation

The dp_packet_dec_channel_lo decodes combinatorially.

3 Verification of the components

3.1 DP packet level test bench

Figure 7 shows the `tb_dp_packet` testbench that verifies `dp_packet_enc` and `dp_packet_dec` together. The two main stimuli for a streaming interface component are:

- upstream source enable
- downstream sink ready.

These stimuli can active, random or pulsed. The `tb_dp_packet` uses the procedures described in [3] to generate blocks of data for the `dp_packet_enc` sink and to verify that these blocks of data arrive properly at the `dp_packet_dec` source. The `proc_dp_verify_data()` is used to verify the `sosi.data`, but also the `sosi.channel`, `sosi.bsn` and the `sosi.err`. The `expected_rx_*` signals in Figure 7 are used to verify that the test as run at all. When the stimuli have finished then signal `tb_end` stops the testbench clock to automatically stop the simulation.

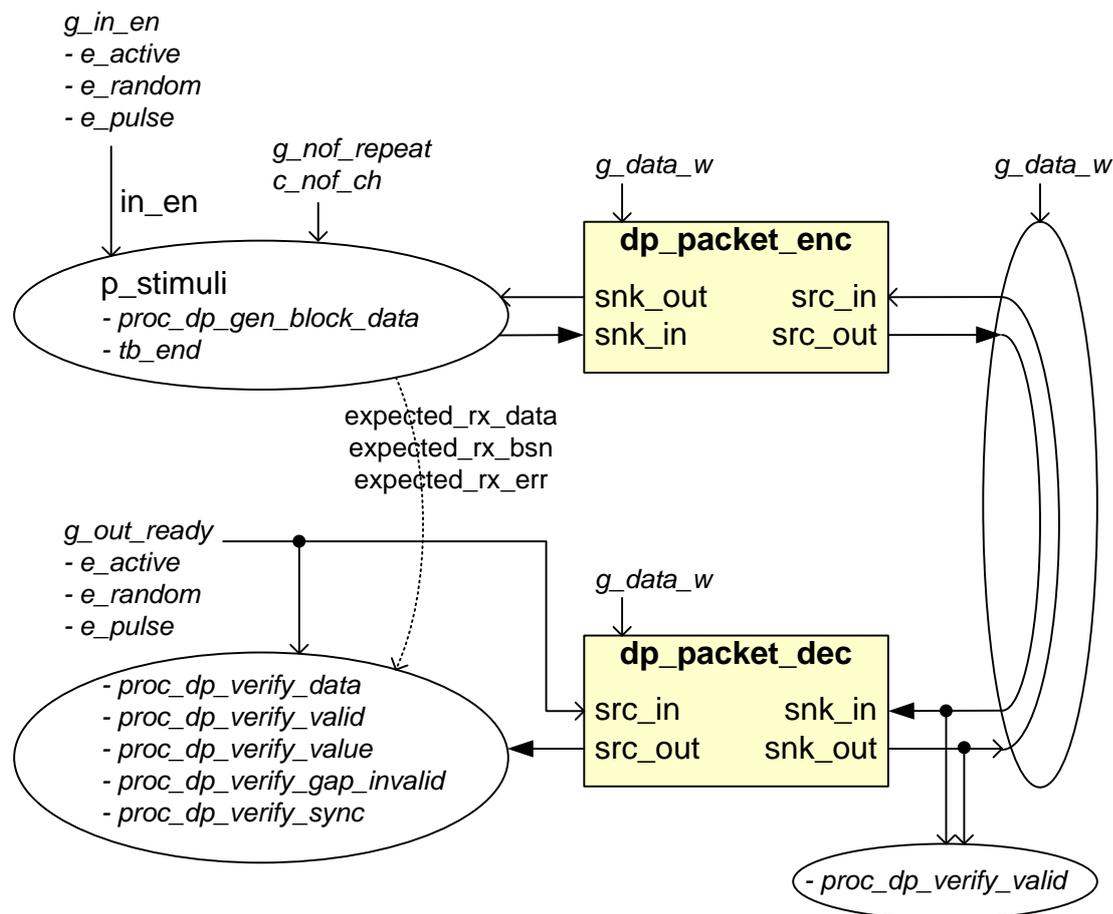


Figure 7: `tb_dp_packet` testbench for `dp_packet_enc` and `dp_packet_dec`

The `tb_dp_packet` multi-testbench instantiates testbench `tb_dp_packet` several times with different generic settings to perform regression tests on the `dp_packet_enc` and `dp_packet_dec`.

The `dp_packet_enc_channel_lo.vhd` and `dp_packet_dec_channel_lo.vhd` are verified in `tb_dp_distribute.vhd`.

3.2 PHY level test bench

Figure 8 shows the `tb_uth_dp_packet` testbench that verifies the transport of DP packets over an Uthernet [6] PHY link. Note that the scheme of Figure 8 models a practical use case for communicating multiple SOSI streams over a single PHY link.

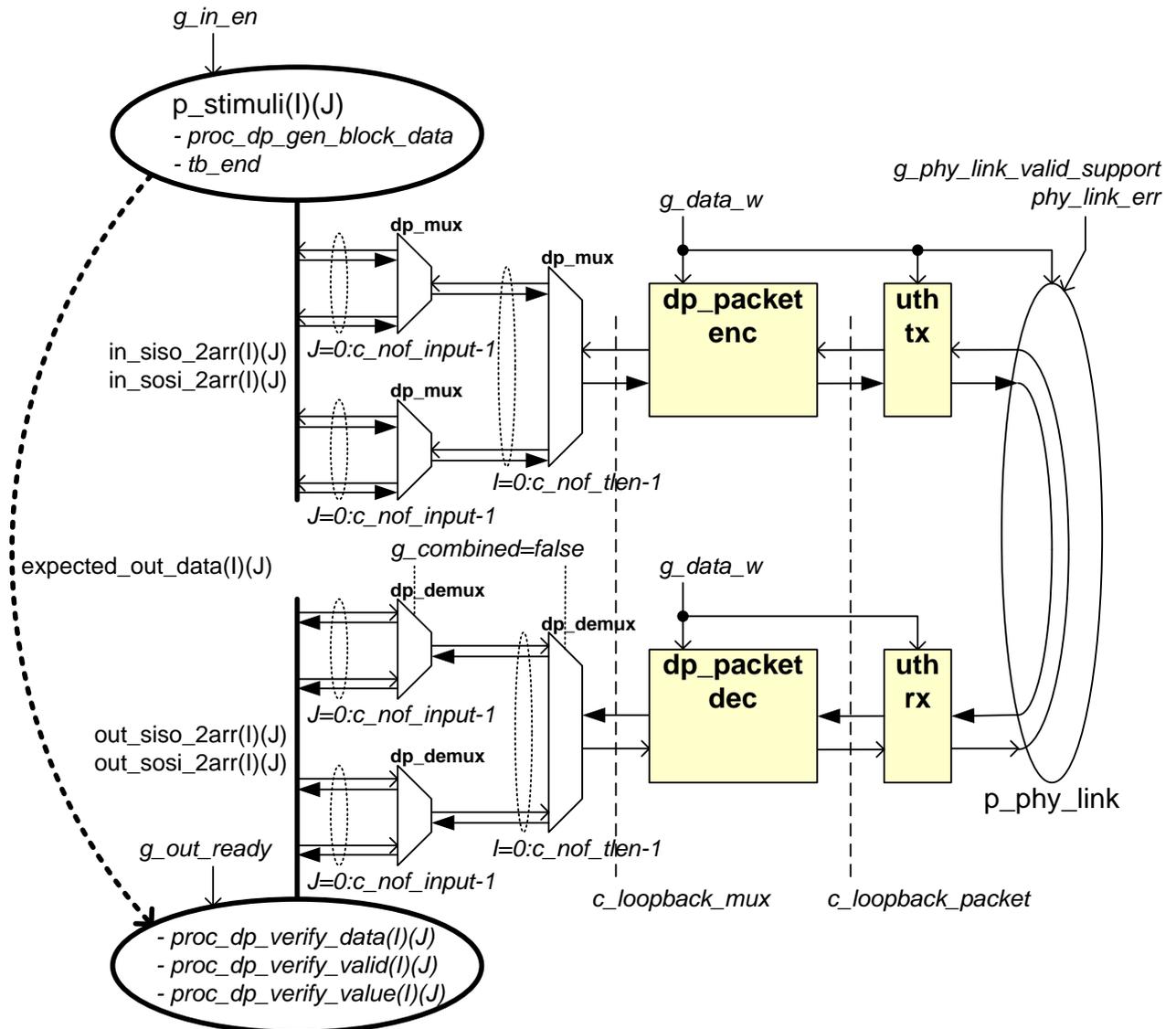


Figure 8: `tb_uth_dp_packet` testbench for `dp_packet` and UTH

The `tb_uth_dp_packet` testbench uses the `uth_tx` and `uth_rx` components from [7]. The `p_stimuli` process generates `c_nof_tlen` different types of packets. For each packet type the `p_stimuli` process generates `c_nof_input` parallel input streams. All input streams are identified by the channel number. The low part of the channel number reflects the different packet types and gets mapped on different TLEN type field values of the Uthernet protocol. The high part of the channel number gets mapped on the DP packet CHAN field. The distinction is set by the DP packet encoding and decoding parameter `g_channel_lo = c_nof_tlen_w`. The

testbench uses dp_mux and dp_demux to perform the multiplexing and de-multiplexing to and from a single stream.

All p_stimuli(I)(J) get the same input enable stimulus and all out_2arr(I)(J) get the same sink ready stimulus. These flow control stimuli can be active, random or pulsed dependent on *g_in_en* and *g_out_ready*. By means of setting *c_loopback_mux* = TRUE or *c_loopback_packet* = TRUE the same test bench can also run looping back already at this interfaces as shown in Figure 8. Default both loopback constants are FALSE. Using *c_loopback_mux* = TRUE and always active flow control stimuli via *g_in_en* and *g_out_ready* reveals that the multiplexers and demultiplexers can pass on the framed data without gaps.

The tb_tb_uth_dp_packet multi-testbench instantiates testbench tb_uth_dp_packet several times with different *g_data_w* and with different flow control.