Notes on the torus link logic

version 0.15
April 1, 2010

Contents

1 Introduction

This is an incomplete and preliminary collection of design considerations and implementation details of the torus network (TNW) developed at the Universities of Milano-Bicocca and Ferrara. The VHDL sources can be downloaded from

`http://moby.mib.infn.it/~simma/tnw`

under the conditions specified there.

1.1 Revision History

date	QPACE functionality	notes
04.12.2008	"torus2" final + OWC tests	0.8+
24.12.2008	"torus3" preliminary + torus4 tests Packet format changed!	0.9−
21.01.2009	"torus3" version 3102 1210 Added `mRe` in packetBuffer	0.9
04.02.2009	"torus3" version 3200 2040 New MDIO slave Changed almost full of txFifo Changed exceptions and reset Changed address calculation	0.10
19.03.2009	"torus3" version 3202 3190 Added `mRe` from OWC	0.11
20.03.2009	"torus3" version 3203 3200 Fixed wrong delay of `mRe` in packetBuffer	0.11
26.03.2009	"torus3" version 3204 3260 Fixed wrong credit update in P24 of rxBuffer Added `RX_EXC_OWCF`	0.12

24.07.2009	"torus4" version 4101 7240 Changed address handling in rxBuffer Changed implementation of packetBuffer Replaced chArbiter2 by chArbiter3 Synthesis fixes to rxLink and txLink	0.13
24.07.2009	"torus4" version 4201 8140 Moved rxLink and rxBuffer to txClk-domain Added phyInt for clock transition	0.13
20.08.2009	"torus4" version 4202 8200 Changed clock generation for phyInt from DCM to PLL Returned to chArbiter2	0.13
06.10.2009	"torus4" version 4301 a060 Changed notify mechanism (toggling bits)	0.14
16.11.2009	"torus4" version 4302 b160 Fixed double Feedback in case of errorT Fixed readout of DCR registers of rxBuffer Attached TXCLK (towards PHY) to PLL	0.14
04.12.2009	"torus4" version 4303 c040 Added flag1K in rxLink	0.14
13.01.2010	"torus4" version 4304 1130 Added flag1T and cntRxe in rxLink	0.15
26.01.2010	"torus4" version 4305 1260 Changed dirtyI rxLink Changed reset value of rExEn in txLink and rxLink	0.15
09.02.2010	"torus4" version 4306 2090 Changed condition for RX_EXC_CBA/NBA Changed reset value of rExEn in txLink and rxLink	0.15
12.03.2010	"torus4" version 4307 3120 Changed in txLink and rxLink reset value of rExEn and dropped offline from rrExc	0.15
01.04.2010	"torus4" version 4308 4010 Increased txFifo and rxBuffer to 16 KB Fixed handling of rComReady in rxLink	0.15

1.2 Notations

- S_{msg} maximal message size guaranteed by NWP (2 KB)
- N_{msg} maximal number of pending messages per channel and link (16 or 64)
- S_{pkt} fixed size of a packet (128 B)
- N_{ch} number of (virtual) channels per link (8)
- A_{ch} number of bits for addressing all channels within link (3)
- N_{notify} number of notify bits per channel and link (16)
- A_{notify} number of bits for addressing the notify bit per channel and link (4)
- A_{LS} number of bits for addressing all Bytes within LS (18)
- A_{msg} number of bits for addressing all Bytes within S_{msg} (11)
- A_{pkt} number of bits for addressing all Bytes within S_{pkt} (7)
- $/x/$ code-group (10b/8b encoded octet)
- $\|x\|$ column (code-groups on all 4 lanes, not necessarily equal)
- entName denotes the name of an entity (VHDL)
- `sigName` denotes the name of a signal (VHDL)
- `CONST_NAME` denotes the name of a constant (VHDL)
- `state` denotes a state of a FSM

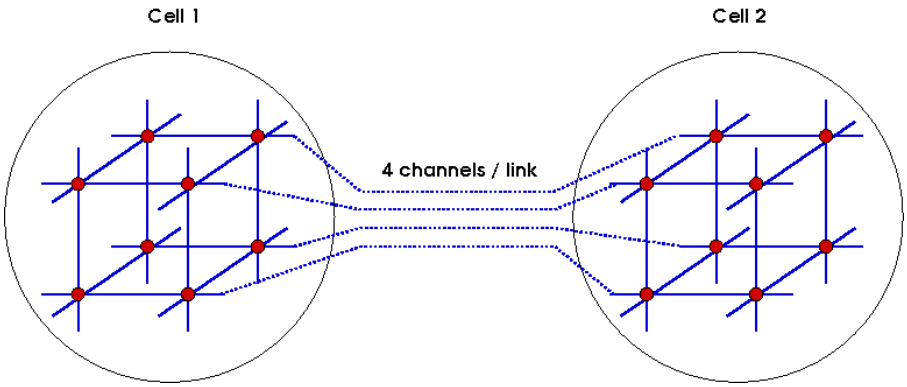
2 Communication Model

2.1 Logical Communication Channels

A single physical link between two Cell processors typically has to be shared for communications between more than one pair of SPEs. In absence of extra synchronisation mechanisms between the SPEs on each Cell, the time order among the send operations on one Cell may be different from the order among the corresponding receive operations on the other Cell.

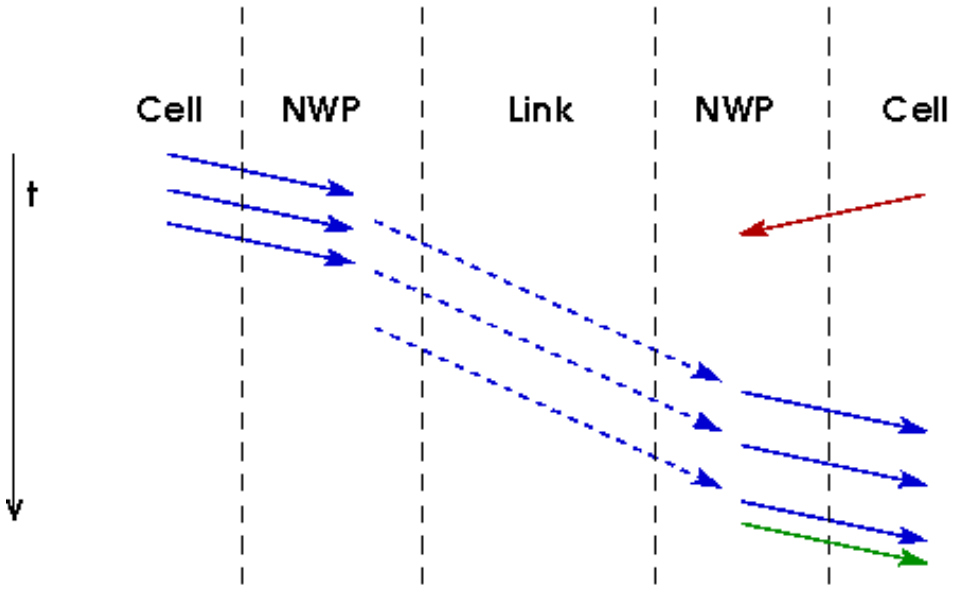
If the data transfer from the NWP to the destination Cell is based on credits from the SPEs, we need to implement N_{ch} separate logical communication channels to avoid delays (if the NWP has only a single Rx fifo and the credit for the first data packet is not yet available) or even deadlocks (if the credits could not be re-ordered according to the sequence of received data packets).

N_{ch}= 4 is sufficient for a 3-d torus topology with the SPEs within each Cell logically arranged as $2 \times 2 \times 2$ and with separate physical links in positive and negative directions



while N_{ch}= 8 is required to allow also logical $8 \times 1 \times 1$ arrangement of the SPEs or to support network topologies with a "false" torus of size 2 (using a single link for periodic closure).

2.2 Data Flow



Sender:

- DMA from processor device to NWP (memory mapped)

Receiver:

- Processor sends credit to NWP
- NWP trasfers received data to processor device
- MWP notifies processor when done (interrupt or polling?)

GBIF:

- inbound write: Tx data + credit
- outbound write: Rx data + notify

2.3 Communication Rules

1. At most $N_{ch} \equiv 8$ **channels per link** are available (for system- or user-defined communication topologies)
2. The **order** of send and receive operations to match if these operations correspond to data transfer in the same directions and across the same link and channel
3. The **size of a transfer** (`length`) must be a multiple of 128 B
   (NWP supports only 128-byte packets)
4. The **start address of transmitted data** (`txbuf`) must be aligned to **128 B**
   (Otherwise fragmentation of the EIB transfer yields invalid packet size for the NWP)
5. The **start address for received data** (`rxbuf`) must be aligned to **16 B**
   (The 7 LSB of the address are not included in the packet header by the NWP)
6. The **maximal size of a single transfer**[1] is

   $$\texttt{length} \;\leq\; S_{msg} = 16 \cdot S_{pkt} \equiv 2 \text{ KB}$$

   (This is a soft limit of the NWP: larger transfers may work, but can lead to a full Tx or Rx fifo, which may cause deadlocks or exceptions. Independent of the NWP, the limitation `length` ≤ 16 KB is imposed by the maximal DMA length supported by the MFC.)

[1] S_{msg} is large enough to hold at least all projected spinors (96 B aligned to 128 B) from one SPE of the surface of a 3d-slice with maximal local lattice, i.e.

$$S_{msg} \geq 4^2 \text{ sites} \times 128\ B = 2\ KB \tag{1}$$

7. Credits should be provided (e.g. through `torus_recv_*`) **before sending** the data to the NWP (to avoid possible deadlock situations due to lack of GBIF tags)

8. Maximal **number of pending**[2] **receive operations** per link and channel is $N_{msg} = 16$ or 64
   (This is the depth of the credit fifo for one channel. Exceeding N_{msg} may cause a fatal exception during DCR write.)

9. The total amount of data in pending transfers to/from the NWP is not limited
   (The above constraints allow accumulation of data transfers with a total size of up to $6 \cdot N_{ch} \cdot N_{msg} \cdot S_{msg} = N_{msg} \cdot 96$ KB to the NWP of one Cell. Of course, the pending DMA operations for credits and data might finally exceed the limit of pending DMA operations in the MFC)

2.4 Configuration of the Torus Topology

Given a machine partition (i.e. a set of physical nodes), the four steps listed below need to be performed in a consistent way in order to fully define and configure a torus communication topology. The first two steps define (and depend only on) the "node-level" topology, while the last two steps define (and also depend on) the "SPE-level" topology:

1. Selection of physical links, defined by a PHY ID ($0, \ldots, 5$) and a corresponding port (primary or redundant) on a pair of nodes

   The PHY IDs and ports depend on the node ID, in order to allow closing of the communications according to the requested node-topology (see also `notes-root.pdf`)

2. Assignment of logical coordinates (X, Y, Z) to each node

   This defines also a mapping of the logical directions $\pm X, \pm Y, \pm Z$ on each node to physical links, and must be consistent with the connectivity of the nodes via physical links.

3. Assignment of logical coordinates (x, y, z) to each SPE

   If the coordinates are local, i.e. labeling only the SPEs within a node, their assignment is arbitrary. Otherwise, if the coordinates are global, i.e. within the entire

[2]Starting when the credit is issued (explicitly by `tnw_credit` on the receiver, or implicitly by homogeneous send-receive operations on all nodes) and ending with the arrival of the notify word on the receiver.

topology (or even machine partition), their assignment must be analogous to and consistent with (X, Y, Z) of the nodes.

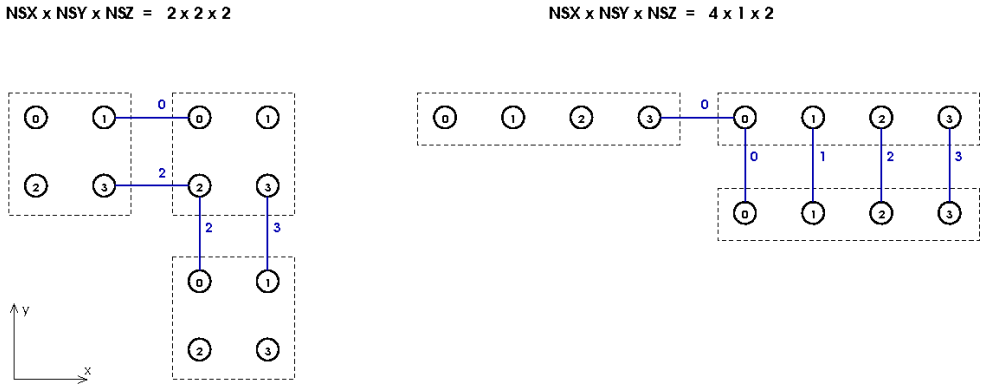
4. Assignment of virtual channels for each logical direction $(\pm x, \pm y, \pm z)$

   The assignment of the channels depends on the local SPE position within the node. If $i = 0, \ldots, 7$ is an arbitrarily assigned ID of the SPEs within each node, and $\mu = \pm x, \pm y, \pm z$, denote the logical directions, the channel assignment must satisfy

   $$c(\mu, i) = c(-\mu, T_\mu(i))$$

   where $T_\mu(i)$ is the ID of the SPE next to i in logical μ direction. For instance, one may choose $c(\mu, i) = \pi(i)$ for all $\mu = -x, -y, -z$, with $\pi(i)$ any permutation of $\{0, \ldots, 7\}$, and then determine $c(-\mu, i)$ from the above relation. Of course, those channels $c(\mu, i)$ for which $T_\mu(i)$ lies on the same SPE, do not correspond to "external" communications and will not be used.

   Note that in case of a "false-2" topology in a direction μ with $T_\mu(i) = i$, the above scheme assigns the same channel to the directions $\pm\mu$. Therefore, rule 2 of section 2.3 applies to transfers in directions $+\mu$ and $-\mu$ (and not only separately for each direction).



2.5 Low-Level API (Communication Primitives)

For the implementation of high-level communication functions and for low-level tests the following communication primitives, might be considered. They have explicit arguments to specify a physical link and a virtual channel (i.e. only step 1. of the previous section is assumed, while all other steps to define the topology are left to the user).

- `tnw_put ( plink, chid, &txbuf, offset, length, dmatag )`
  send `length` (in units of 128 B) data, starting from `&txbuf`, by DMA put to

the txFifo of a virtual channel `chid` and physical link `plink`. The `dmatag` (= 0 ... 31) specifies the DMA tag group that can be used to test for completion on the sending SPE, and `offset` specifies the remote offset that can be included in the packet header (the current address mapping for the txFifo only allows `offset`=0)

- `tnw_test_put ( dmatag )`

  test completion status of DMA tag group `dmatag` (corresponding to a data transfer to the TNW) by `spu_mfcstat`

- `tnw_credit ( plink, chid, &rxbuf, length, nindex )`

  pass credit to TNW in order to set up a transfer of `length` (in units of 128 B) data from virtual channel `chid` and physical link `plink` of the TNW to LS at address `&rxbuf`.

- `tnw_test_notify ( plink, chid, nindex )`

  test completion status of data reception by reading the bit specified by `nindex` at the notify location (a 16 B word within the notify area which must have been allocated by `tnw_notify_base`, see below, and which must be 128B aligned)

- `tnw_credit_base ( plink, chid, addr )`

  set credit base address register (in rxBuffer) to use `addr` for data reception via physical link `plink` and virtual channel `chid`

- `tnw_notify_base ( plink, chid, &nbuf )`

  set notify base address register (in rxBuffer) tp use notification location at `&nbuf` for physical link `plink` and virtual channel `chid`. Also initializes internal variables (state and address) needed by `tnw_test_notify`

- `tnw_reg_read ( addr )`

  read DCR register at address `addr` (note that the address bits which specify the physical link depend on whether the register is in rxLink or txLink)

- `tnw_reg_write ( addr, value )`

  write 32-bit `value` to DCR register at address `addr`

- `tnw_mdio_read ( plink, addr )`

  read MDIO register `addr` of the PHY corresponding to specified physical link `plink`

- `tnw_mdio_write ( plink, addr, value )`

  write 16-bit `value` to MDIO register `addr` of the PHY corresponding to specified physical link `plink`

- `tnw_config_*` for various other configuration operations ...

Open issues:

✘ It might be convenient to specify `plink` in terms of symbolic names which are independent of the topology (e.g. reserved variables, referring to link "colors"),

The actual values of these variables depend on the node-level topology and on the node ID. In particular, they depend on the periodicity needed for the links of each color (but not on all details of the node-level topology, e.g. on whether the logical x direction is mapped onto red or green links).

The variables might be chosen and initialized (in step 1. of section 2.4) according to the following scheme (for the definition of the topology names see `notes-root`):

variable	value	nodes	topology
`tnw_plink_rp`	2	$NID\%4 \neq 0$	r4, r8
	5	$NID\%4 = 0$	r4, r8
	5	$NID\%4 \neq 2$	r2'
	2	$NID\%4 = 2$	r2'
`tnw_plink_rm`	5	$NID\%4 \neq 0$	r4, r8
	2	$NID\%4 = 0$	r4, r8
	`tnw_plink_rp`		r2'
`tnw_plink_gp`	3	$NID = 0, \ldots, 7, 24, \ldots, 31$	g4
	0	$NID = 8, \ldots, 23$	g4
	3		gh
	0		gv
`tnw_plink_gm`	0	$NID = 0, \ldots, 7, 24, \ldots, 31$	g4
	3	$NID = 8, \ldots, 23$	g4
	`tnw_plink_gp`		gv, gh
`tnw_plink_bp`	1	???	
`tnw_plink_bm`	4	???	

In the above variable names, the first suffix letter refers to the "colors" (red, green, blue), and the second suffix letter to the orientation of the closed PB topologies (where positive orientation is defined by passing from node 0 to the node with lowest NID).

✘ Configuration of the physical links (i.e. selection of primary or redundant PHY ports, reset of rxLink and txLink, and assignment of the values of `tnw_plink_*`) should be done by a lower software level (e.g. the OS) prior to the start of the user program (taking into account the necessary information about the requested node-level topology).

✘ The mapping between the logical directions ($\pm x$, $\pm y$, $\pm z$) and `plink` and `chid` may be handled at a higher software level (using information about the requested node- and SPE-level topology).

2.6 High-level Communication Functions

At a higher level one might introduce the following functions to perform communications via the torus network (a `_*` in the function name refers to one of the suffixes `_xp`, `_xm`, `_yp`, `_ym`, `_zp`, and `_zm` for the logical directions):

- `torus_init` checks and initializes the external communication topology (via the network links within a partition of `NCX` × `NCY` × `NCZ` Cells and with the SPEs on each Cell arranged as `NSX` × `NSY` × `NSZ`). In particular:
  - assignment of a physical link and virtual channel (Cell- and SPE-dependent) for each external logical direction
  - allocation and initialisation of internal variables (e.g. for the mapping of logical directions to physical links, virtual channels, and memory addresses of the corresponding txFifo in the NWP)
  - initialisation of the credit base address registers (e.g. equal to LS offset) for all required links and channels (using e.g. `tnw_credit_base`)
  - allocation and initialization of notify locations in LS for all required links and channels
  - initialization of notify base address registers for all required links and channels (using e.g. `tnw_notify_base`)
- `torus_send_* ( &txbuf, length, nindex )`
  start DMA transfer to NWP by calling `tnw_put` for corresponding `plink` and `channel`
- `torus_recv_* ( &rxbuf, length, dmatag )`
  pass credit to NWP by calling `tnw_credit` for corresponding `plink` and `channel`
- `torus_wait_send ( dmatag )`
  poll on completion of DMA transfer by calling `tnw_test_put`
- `torus_wait_recv_* ( nindex )`
  poll on completion of data reception by calling `tnw_test_notify` for corresponding `plink` and `channel`

Optional additional functions:

- `torus_sendrecv_*` combined versions of `torus_send_*` and `torus_recv_*` (from opposite logical directions)

- `torus_test_recv` non-blocking version of `torus_wait_recv_*`)
- `torus_test_send` non-blocking version of `torus_wait_send`)
- `torus_cell_{x,y,z}` return logical coordinates of Cell within the machine partition (or topology???)
- `torus_spe_{x,y,z}` return logical coordinates of SPE within Cell

Moreover, the high-level communication functions might be extended to handle also internal communications (between SPEs within the same Cell) via DMA plus suitable ready and notify operations (see e.g. Dirac kernel by Andrea Nobile).

An SPU code using these high-level functions might then look as follows:

```
#include <torus.h>
volatile MyType txbuf[LBUF] __attribute__ ((aligned(128)));
volatile MyType rxbuf[LBUF] __attribute__ ((aligned(16)));
int dmatag, nindex;
...

torus_init(NSX,NSY,NSZ);                // initialize SPE topology
...

torus_recv_xm(\&rxbuf,length,nindex);  // credit BEFORE transmission!
torus_send_xp(\&txbuf,length,dmatag);  // data transmission
...

torus_wait_send(dmatag);                // wait for completed DMA
...                                     // before overwriting txbuf

torus_wait_recv(nindex);                // wait for notify
...                                     // before using received data
```

2.7 Implementation of QMP

Capability Requirements: (see [4])

- ✔ 1. Barriers → global signals
- ✘ 2. Send contiguous message to arbitrary node → multi-hops only by software
- ✔ 3. Send message to neighbouring nodes → genuine TNW

- ✘ 4. Transfer non-contiguous message of strided blocks → alignment constraints?
- ✘ 5. Transfer to machine specified by communication map → depends on 2.
- ✘ 6. Transfer to machine specified by physical node ID → depends on 2.
- ✔ 7. Broadcast → by software via TNW
- ✔ 8. Global sum → by software via TNW
- ✔ 9. Global max → by software via TNW
- ✔ 11. Global reduction with arbitrary binary function → by software via TNW
- ✔ 12. Machine configuration discovery and control → support by system software

Performance Requirements: (see [4])

- ✔ 1. Overlapping of computation and communications
- ✔ 2. Multiple sends without waiting for the first to complete → fifo sizes?
- ✔ 3. Initiation of sends in all directions by a single call → by software
- ✔ 4. Wait on message completion without barrier
- ✔ 5. Expensive operations performed ahead of multiple ("persistent") communications
- ✔ 6. Minimal bookkeeping overhead on host and interfaces

Communication API: (see [4])

- Configuration and layout of machine
  → machine = "partition" + "topology"
- Initialization and finalization of QMP resources
- Discovery of machine capabilities at run time
  → using system support (e.g. access to registers of NC and RC)
- Creation of logical (allocated) machine
  → assignment of physical links and virtual channels (see section 2.4)
- Allocate communication buffers in memory
  → LS only?
- Declare endpoints of message channels
- Communications (start send and receive, test, wait)
  → mapping on torus functions

QMP	TNW
`QMP_start`	`torus_send`
	`torus_recv`
`QMP_is_complete`	`torus_test_*`
`QMP_wait`	`torus_wait_*`

3 Control Information Flow

3.1 Arguments of High-level Function `torus_send_*`

argument	information	values	comments
`txbuf`	Address of Tx data	$k \cdot 128$ B, $k < 2^{11}$	full LS range, 128 B aligned
`length`	Length of Tx data	$l \cdot 128$ B, $l \leq S_{msg}/128$	multiple of 128 B ($l \leq 16$ for S_{msg}=2 KB)
`dmatag`	Tag group of DMA	$0 \ldots 31$	used for `spu_readch`
implicit	Link ID	$0 \ldots 5$	assigned by `torus_init`
implicit	Channel ID	$0 \ldots N_{ch} - 1$	assigned by `torus_init`

3.2 Arguments of High-level Function `torus_recv_*`

argument	information	values	comments
`rxbuf`	Address for Rx data	$k \cdot 128$ B, $k < 2^{14}$	full LS range, 16 B aligned
`length`	Length of Rx data	$l \cdot 128$ B, $l \leq S_{msg}/128$	multiple of 128 B ($l \leq 16$ for S_{msg}=2 KB)
`nindex`	Index for notify location	$0 \ldots N_{notify} - 1$	allocated by `torus_init`
implicit	Link ID	$0 \ldots 5$	assigned by `torus_init`
implicit	Channel ID	$0 \ldots N_{ch} - 1$	assigned by `torus_init`

✘ Maybe the choice N_{notify}=16 is not optimal. Alternatively, N_{notify}= 8 would allow to reduce the storage requirement for the notify locations to 128 B per channel and link, while N_{notify}=32 would match the number of tag groups available for `torus_send_*`

3.3 Credits and Notify

information	bits	comments
Link ID	— (3)	implicit (in memory map/FIFO)
Channel ID	— (A_{ch})	implicit (in memory map/FIFO)
Write Address Offset	14 ($A_{LS} - 4$)	full LS range, 16 B aligned
Notify Index	4 (A_{notify})	N_{notify} bits in 128 B aligned word
Size s	≥ 4 ($A_{msg} - A_{pkt}$)	in units of 128 B packets (assuming value $s = 0$ for 1 packet)
Repeat	≤ 10	
total	≤ 32	data width of DCR
	e.g. 14+4+7+7	(allows 128×8 S_{msg})

The write address for the i-th 128 B data packet (with $0 \leq i < s$ where s is the size specified in the credit) is 128 B aligned and given by[3]

$$\begin{aligned} & 2^{41} \cdot M_c \quad +' \quad 128 \cdot \Big(\langle B_c \rangle_{25} \oplus \langle R_i \rangle_{11} \oplus \langle C \rangle_{16} \Big) \\ = \texttt{0x20000000000} & \cdot M_c \quad +' \quad \texttt{0x80} \cdot \Big(\langle B_c \rangle_{25} \oplus \langle R_i \rangle_{11} \oplus \langle C \rangle_{16} \Big) \end{aligned} \tag{2}$$

where

- M_c is the MSB of the credit base address DCR register
- $\langle B_c \rangle_{25}$ is the 25-bit value specified in the LSB of the credit base address DCR register of the corresponding channel
- $\langle R_i \rangle_{11}$ is the 11-bit value of the remote offset received in header of packet i
- $\langle C \rangle_{16}$ is the 16-bit value of the address offset specified in the credit

i.e. the data address is determined according to the following scheme

	MSB	LSB
M_c	m0 0000 0000 0000 0000 0000 0000 0000 0000 0000	
B_c	bbbb bbbb bbbb bbbb bbbb bbbb b000 0000	
C	ccc cccc cccc cccc c000 0000	
R_i	rr rrrr rrrr r000 0000	

[3]Here, a $+'$ indicates that the summation can be performed by simple concatenation of the bits, while $\oplus$ requires a true adder device.

A 16B notify word after $s+1$ data packets (each with 128 B payload) is written to the 128B-aligned address

$$\begin{aligned} & 2^{41} \cdot M_n \quad +' \quad 128 \cdot \langle B_n \rangle_{25} \\ = \texttt{0x20000000000} & \cdot M_n \quad +' \quad \texttt{0x80} \cdot \langle B_n \rangle_{25} \end{aligned} \tag{3}$$

where

- M_n is the MSB of the notify base address DCR register
- $\langle B_n \rangle_{25}$ is the 25-bit value specified in the LSB of the notify address offset DCR register of the corresponding channel

i.e. the notify address is determined according to the following scheme

	MSB	LSB
M_n	m0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	
B_n	bbbb bbbb bbbb bbbb bbbb bbbb b000 0000	

Each time, when a notify word for a given channel is generated and written to the Cell, one of its 16 LSB (as specified by the "notify index" in the credit) is toggled. A well-defined initial state `0x0000` is enforced by `rxRst` as well as by each write-access to the notify base address register `rNba` of the corresponding channel.

If the optional "repeat" is implemented ... xxx

- ✘ address used for data should be the same for (the first packet of) reach repetition
- ✘ data used in notify should also contain info about the number of the current repetition (to check RAW dependencies for each individual and specific transfer)
- ✘ it is not clear whether a multi-buffering scheme allows to guarantee WAR dependencies

3.4 Attributes of Inbound Write

information	bits	comments
Address offset	$A_{msg} \ldots A_{LS}$	
Link ID	3	from memory map
Channel ID	A_{ch}	from memory map
total	≤ 42	`al_in_wr_addr`

3.5 Control to txLink (from IWC)

information	bits	comments
Address offset of packet	11 $(A_{LS} - A_{pkt})$	full LS range, 128 B aligned
Length of packet	— (6)	fixed: 128 B
Channel ID	A_{ch}	
Link ID	— (3)	implicit
total	≤ 16	extra bits of BRAM
	e.g. $14 = 11 + 3$	(N_{ch}= 8)

3.6 Header of Data Packet (on Physical Link)

information	bits	comments
Address offset of packet	11 $(A_{LS} - A_{pkt})$	from IWC
Length of packet	—	fixed (or implicit in packet type)
Channel ID	A_{ch}	from IWC
Link ID	— (3)	implicit
total	≤ 32	separate column
	≤ 20	separate column enforcing error flush
	≤ 24	same column as /S/ symbol
	≤ 16	same column as /S/S/ symbols
	≤ 15	free parity bits in txFifo
	e.g. 14 = 11+3	

3.7 Control from rxLink (to rxBuffer)

information	bits	comments
Address offset of packet	11 $(7 \ldots 14)$	from header
Channel ID	A_{ch}	from header (implicit only with separate FIFOs)
Link ID	— (3)	implicit

3.8 Attributes for Outbound Write

information	bits	comments
Address	≤ 42	`out_wr_addr` (EA or RA?)
Notify Flag	1	select between Data and Nofify
Length of GBIF transfer	≤ 13 (4K)	implicit in Notify if packet size fixed
		`out_wr_size` (> 0)
		`out_wr_max_size` $= 256$ (SGB)
		multiple of 16 B (128 B aligned by IOIF?)

4 Link Protocol

4.1 Packet Format

"column"	code-groups (lane 0...3)				comment
0	h	h	h	h	16 bit header
1	d	d	d	d	data
⋮					
32	d	d	d	d	data
33	c	c	c	c	32-bit CRC
	$/I/$	$/I/$	$/I/$	$/I/$	idle, IPG

Rules:

- A column with a feedback command (ACK or NACK) can be inserted between any of the above columns
- One or more extra idle columns (possibly equal to $\|I\|$) can be inserted between any of the above columns
- One RESTART column is sent before any data is resent (i.e. whenever the resend level increases, see sect. 4.4)
- No feedback is sent for any packet after the first NACK, until a RESTART is received.
- No RESTART column is sent before the first packet after reset
- 2 idle columns[4] (or more) are sent after at most consecutive 256 packets (to allow for sufficient clock rate compensation)

[4]Guarantees transmission of one $\|A\|$ or $\|K\|$, followed by one $\|R\|$ (see Fig. 48-6 of [3]).

4.2 Command Encoding

Tentative proposal [fe 16.5.2008]:

	lane 0	lane 1	lane 2	lane 3
RESTART	$/K_1/$	$/K_1/$	$/K_1/$	$/K_1/$
ACK	$/K_2/$	$/K_2/$	$/K_2/$	`cnt`
NACK	$/K_3/$	$/K_3/$	$/K_3/$	`cnt`

- requires 3 free K-codes $K_1, \ldots, K_3$
- delayed disparity errors or code violations are flushed by RESTART
- header column should be encoded to flush delayed disparity errors and code violations
- extra data byte `cnt` in ACK and NACK can be used for a counter to implement a weak protocol check (causing a fatal error only after 3 consecutive mismatches)
- detection of all commands is robust by majority vote. Nevertheless, a triple $/k/$ $/d/$ $/E/$ can not be unambiguously identified as a command or data column (it might arise from $/k/$ $/k/$ $/k/$ or from $/d/$ $/d/$ $/d/$ due to a single-lane error plus a delayed code violation or disparity error?).

Alternative coding [fe 16.5.2008]:

START	$/K_1/$	$/K_1/$	h	h
RESTART	$/K_1/$	$/K_1/$	$/K_1/$	$/K_1/$
ACK	$/K_2/$	$/K_2/$	$/K_1/$	$/K_1/$
NACK	$/K_2/$	$/K_2/$	$/K_2/$	$/K_2/$

- might need only 3 free K-codes
- allows no counter for protocol checks
- needs extra colums to guarantee flushing of delayed code violations or disparity errors, e.g. after each data packet (to avoid propagation of errors between packets) and after commands in lanes with a K-code (to enforce error detection)

Open issues:

- ✘ Choice of the values of free bits in header column
  (to enforce flushing of delayed code violations or disparity errors)

4.3 Transmission Errors

Discussion [dp+ts 9.5.2008]

- The network protocol should be robust wrt either of the following situations (i.e. not a combination of such situations):
  - 1 bit per packet (i.e. up to 136 Bytes) flips on serial XAUI link (i.e. encoded data stream)
  - 1 bit error on parallel XGMII interface
- After each packet it should be ensured that next 4 bytes on the 4 lanes are coded such that disparity errors are discovered (i.e. RXH[0..3] is on). This is the case for any K-code.
- Monitoring relation between CRC errors and disparity errors may be interesting as a CRC error without disparity error would indicate errors on the parallel XGMII interface.
- Any error detected by PHY (coding violation, disparity, etc.) and generating a $/E/$ in data stream should be handled in the same way as a CRC error and lead to NACK [fe 16.5.2008]

Questions:

- Is transmission of K-symbols on 2 lanes sufficient to protect against the above error cases?
- Maximum number of flipped bits after decoding of code-group with 1-bit error?

  If `RXC` is ignored, worst case is a one-bit error during transmission of D1.0 (0x01) causing $/E/$ = K30.7 (0xfe), i.e. an 8-bit error on `RXD`.

  Even if `RXH`=0 is required, a one-bit error during transmission of D3.1 (0x23), which has the same encoding for both running disparities, may be decoded as a valid K28.6 (0xdc), i.e. cause an 8-bit error.

  If `RXC`=0 is required, the worst case is a 4-bit error, e.g. for a one-bit error during transmission of D1.1/RD- (0x21) becoming D14.1 (0x2e).
- Can disparity errors escape the CRC [hs]?

4.4 Resend Mechanism

Delays:

The time delay between transmission of a packet and receiving of the corresponding feedback is a-priori not known, and may vary from packet to packet. For the worst case, in units of the time to transmit one packet, $T_{pkt} \approx 34$ cycles, we estimate

Origin	Delay $[T_{pkt}]$
passing packet to PHY	1
Tx latency of PHY	1
delay of cable (5m)	<1/4
Rx latency of PHY	1
availability of CRC	1
arbitration of feedback	$\leq 1/16$
Tx latency of PHY	1
delay of cable	<1/4
Rx latency of PHY	1
Total	≤ 7

The time delay between subsequent feedbacks can be guaranteed to be at least $O(10)$ cycles (typically $\geq T_{pkt}$).

Resend Buffer:

We consider a Resend Buffer which is logically a cyclic buffer of variable length. Whenever a data packet is sent (or re-sent) over the link, it is pushed into the Resend Buffer. Packets are removed from the Resend Buffer only upon receiving a corresponding ACK. When a packet is resent, it is removed from the Resend Buffer and re-inserted after the last packet.

Resend Level:

Resending of data packets is recursive, i.e. we need to handle situations when a resent data packet itself causes a NACK and some packets need to be resent several times.

When txLink (re-)sends a data packet which already has been sent L times, we say that the data packet has "resend level L" and that txLink is in resend level L. When txLink in a non-zero resend level, we also say that the link is in "resend mode".

Example:

The data movement during the transmission of the packet sequence $d_1, d_2, \ldots$ (with d_1 the first packet in txFifo) might occur as in the following example:

txFifo last ... first	Resend Buffer last ... first	`fbIn`	Tx to PHY	L	comment
$\ldots d_2, d_1$	—	—	—		
$\ldots d_2$	—	—	d_1	0	send d_1
$\ldots d_4, d_3$	d_1	—	d_2	0	
$\ldots d_5, d_4$	d_2, d_1	—	d_3	0	
$\ldots d_6$	d_3, d_2, d_1	a_1	d_4	0	ACK for d_1
$\ldots d_6$	d_4, d_3, d_2	n_2	d_5	0	NACK for d_2
$\ldots d_6$	d_5, d_4, d_3, d_2	(n_3)	RESTART		enter resend mode
$\ldots d_6$	$*, d_5, d_4, d_3$	(n_4)	d_2	1	send d_2 from Resend Fifo
$\ldots d_6$	$d_2, *, d_5, d_4$	(n_5)	d_3	1	d_2 re-pushed on Resend Fifo
$\ldots d_6$	$d_3, d_2, *, d_5$	—	d_4	1	
$\ldots d_6$	$d_4, d_3, d_2, *$	a_2	d_5	1	ACK for resent d_2
$\ldots d_7, d_6$	d_5, d_4, d_3	a_3	—		
$\ldots d_7, d_6$	d_5, d_4	n_4	—		NACK for resent d_4
$\ldots d_7, d_6$	d_5, d_4	(n_5)	RESTART		
$\ldots d_7, d_6$	$*, d_5$	—	d_4	2	
$\ldots d_8, d_7, d_6$	$d_4, *$	—	d_5	2	
$\ldots d_8, d_7, d_6$	d_5, d_4	—	—		
$\ldots d_8, d_7, d_6$	d_5, d_4	a_4	—		
$\ldots d_8, d_7, d_6$	d_5	a_5	—		exit resend mode
$\ldots d_8, d_7, d_6$	—	—	—		resume normal mode
$\ldots d_8, d_7$	—	—	d_6	0	send next data from Tx Fifo
$\ldots d_8$	d_6	—	d_7	0	

Legend:

- The content of the Resend Buffer is shown as if it was a fifo (other implementations are discussed below).
- A "$*$" in the Resend Buffer indicates a (logical or true) "marker" which is pushed onto the fifo when the restart level L increases (and hence, when viewing the Resend Buffer as a cyclic buffer, separates the oldest from the newest packet)
- a_i and n_i denotes an ACK or NACK feedback, respectively, refering to d_i
- A NACK in parenthesis, (n_i), might be suppressed by rxLink (in "quiet mode")

Synchronisation between txLink and rxLink:

To simplify the synchronisation between txLink and rxLink, all packets which follow a corrupted packet (which raised the first NACK) are discarded by rxLink until that packet has been resent and arrives again at rxLink.

Due to the unknown time delay, this must be explicitly indicated to rxLink by a RESTART command. At this point, rxLink knows (or can determine) the number of packets, $N_{pkt}(1)$, which will be resent by txLink.

There are two ways to control the behaviour of rxLink when handling further errors that may occur while receiving the resent packets:

- "Active Rx": When a further error occurs in the n_1-th resent packet, rxLink discards a total of $N_{pkt}(2) = N_{pkt}(1) - n_1 + 1$ packets (of resand level $L = 1$) and then autonomously resumes receiving of the remaining $N_{pkt}(2)$ packets of resend level $L = 2$. Further errors during their receiving can be handled in an analogous way, until all $N_{pkt}(1)$ packets have been received without error.
- "Passive Rx": No use is made of the possible knowledge of $N_{pkt}(1)$, and rxLink discards all received packets it until receives a further RESTART command (analogous to the behaviour after an error which has occured at resend level 0).

In the following we consider only a "Passive Rx". In this case, txLink has to send a RESTART command before any data packet which has a higher[5] resend level than the previous ones. For this purpose txLink must keep track of the resend level.

This can be realized by implementing the Resend Buffer as a RAM with three pointers.

- `waddr` pointing to the next address where new data can be written
- `start` pointing to the *oldest* data written to the buffer (and not yet removed due to a corresponding ACK)
- `current` pointing to the next data to be resent

When not in resend mode, only `start` and `waddr` are used to manage the buffer as a fifo. When resend mode is entered, or whenever the resend level is increased, `current` is set to `start`. Then, `current` is incremented for each packet which is resent, until it reaches `waddr`, i.e. $N_{pkt}(L)$ packets have been resent. txLink can start to consume feedbacks from the resent packets already while still resending the packets: When an ACK is received, the oldest packet is removed from the buffer by simply incrementing `start`), otherwise, a NACK requires to enter a higher resend level (transmitting a RESTART and setting `current` to `start`).

Compared to the fifo based implementations, this has the advantage that removing of a packet requires only the time[6] to increment `start` (1 cycle).

Note that rxLink may

[5] After an error all packets, which have already been (re-)sent by txLink, need to be resent again. Therefore, the resend level can only increase, or the resend mode terminates (returing to resend level 0).

[6] The overall time for resend may thus be reduced by $O(33) \cdot L_{max}$ cycles, where L_{max} is the maximal resend level which occured.

- either continue to send (redundant) NACKs for each packet which is received after the first NACK and before receiving a RESTART ("verbose mode")
- or suppress any feedback after the first NACK until a RESTART is received ("quiet mode")

However, the time spent in resend mode may be reduced[7] in the second case (suppression of repeated NACKs by rxLink upon errors), because resending at level L can be immediately terminated (to enter at level $L+1$) when a NACK is received.

In the following we consider an implementation with "quiet mode", because this seems simpler for both, rxLink (NACK needs to be generated only in a single state and no bookkeeping of incoming data is required), as well as for txLink (handling of the redundant NACKs would have to be done in correspondence with resending a packet, possibly delaying the latter, or requiring extra bookkeeping).

5 txLink

5.1 Interface between Control Block and txLink

dir	bits	signal	comments
in	1	`rst`	reset FSM (not PHY)
in	1	`offline`	disconnect inputs
out	≥ 9	`exc`	exceptions

5.2 Interface between IWC and txLink

Proposal Regensburg 23.4.2008 + update 18.6.2008

dir	bits	function	comments
in	128	`data`	
in	42	`addr`	unchanged for 8 words
in	1	`first`	1 clk every 8 words
in	1	`we`	
out	1	`almostFull`	3 words before Full

Handling of length information from GBIF:

[7] This may not be relevant, because the first feedback of a packet resent at level L typically arrives after transmission of all $N_{pkt}(L)$ packets.

- All bits provided by IWC (also needed by Ethernet)
- IWC raises signal to exception handler block if lentth $\neq$ 128 B

Handling of Fifo Full:

- IWC checks in advance for `almostFull`
- What response (to MGB) should be created in case of timeout?
  - pull and dump remaining data
  - raise `in_wr_done_error` $\Rightarrow$ FIR $\Rightarrow$ optional machine check
  - raise signal to exception handler block

  Until FIR handling is understood, just dump and raise error!

NOTE: `almostFull` is suppressed when `offline` is asserted to txLink

5.3 Interface between PHY and txLink

dir	bits	function	comments
out	1	`TX_CLK`	
out	32	`TXD[31:0]`	
out	4	`TXC[3:0]`	
in	1	`TX_FAULT`	(asynchronous)

5.4 FIFO and RAM resources in txLink

- txFifo: $d \approx 512$, $w = 128$ bit (4×32)

$$S_{\text{txFifo}} = d \times w \geq 64 \cdot S_{pkt} = 4 \cdot S_{msg} = 8 \text{ KB}$$

  ($w' \leq 16$ bit extra bits of txFifo are used for header information)

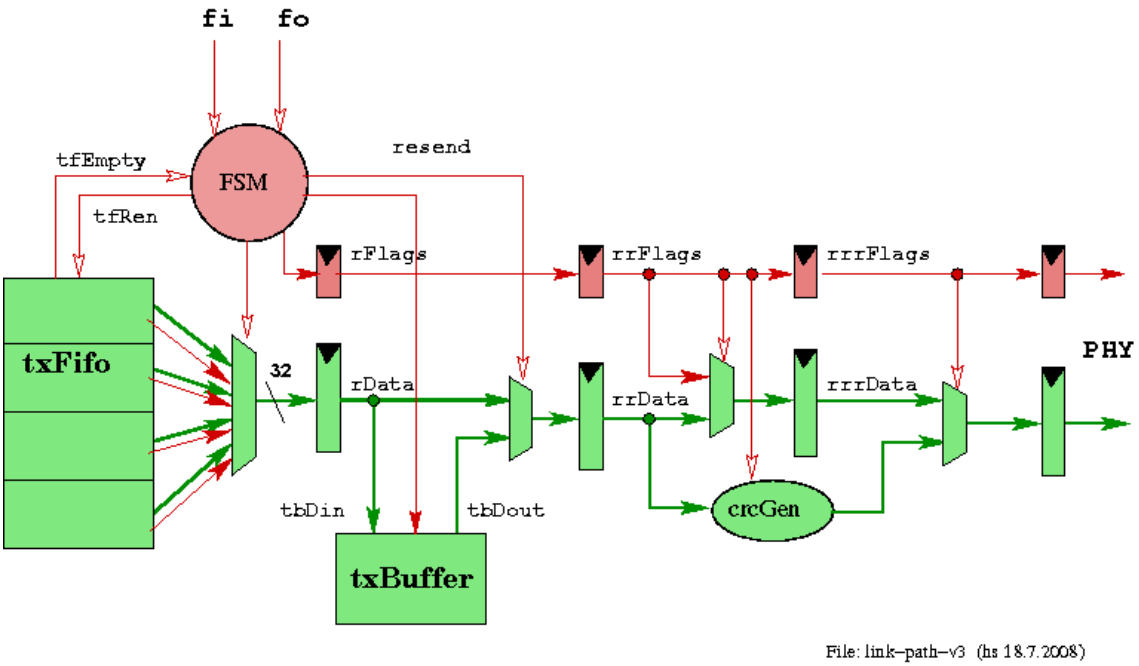
$$\longrightarrow 2\times \texttt{FIFO36\_72}$$

  ✘Size has been doubled (to $S_{\text{txFifo}} = 16$ KB, $d = 1024$) in torus4 version 4308 4010

- txBuffer (Resend Buffer): 8 packets $\times$ 128 B = 1 KB

$$\longrightarrow \lceil 0.5 \rceil \texttt{ RAM18K} \simeq 0.5\times \texttt{ FIFO36\_72}$$

5.5 Data Paths of txLink



5.6 Control Logic of txLink

Open issues:

- ✘ `fbIn` might need a tiny fifo (e.g. just toggling of two registers)
- ✘ Do we need an exception if `iwcIn.we` is asserted during `offline`?

General Remarks:

- A single FSM handles the following tasks
  - read and serialize header info and data from txFifo
  - read or flush data from Resend Buffer
  - insert feedback, idle, restart columns into data stream
- Assuming that txFifo is written faster than read, it should never become empty during transmission of a packet. If it happens, an exception is raised.
- The Resend Buffer is called txBuffer in the following. It implements a RAM with three pointers, as discussed above. The pointers refer to data blocks of 64×32 bit. Read- and write-access to individual 32-bit words within each

block is explicitly handled by two 6-bit addresses, which both can have the same values, because reading and writing can never occur at the same time (reading only when in resend mode, and writing only otherwise). The values of these addresses are decoded from the states of the FSM as

$$\texttt{addr} = \begin{cases} \langle \texttt{count} \rangle \times 4 + \langle n \rangle & \text{in } \mathsf{DATA}\langle n \rangle \\ \texttt{0x20} & \text{in } \mathsf{HEAD} \end{cases}$$

- Assuming that txBuffer is large enough for the maximal feedback delay, it should never become full. If it happens, e.g. because feedbacks were lost, an exception is raised.
- Resend mode continues until all data in txBuffer is resent (`tbEmptyS`), but can be (and typically is) left *before* all feedbacks for the resent packets are received
- Data propagates together with flags (valid bit and further control bits) in order control down-stream manipulations on the data (e.g. CRC)
- The input `offline` allows to effectively decouple txLink from all other inputs to allow a clean reset without side-effects to or from elsewhere. In particular, when `offline` is asserted, the signals `iwcIn.we`, `fbInReq` are ignored, while `iwcOut.almostFull` is held low.
- For handling of incoming and outgoing feedbacks if if `offline` is asserted, see below.

Handling of Outgoing Feedback Commands:

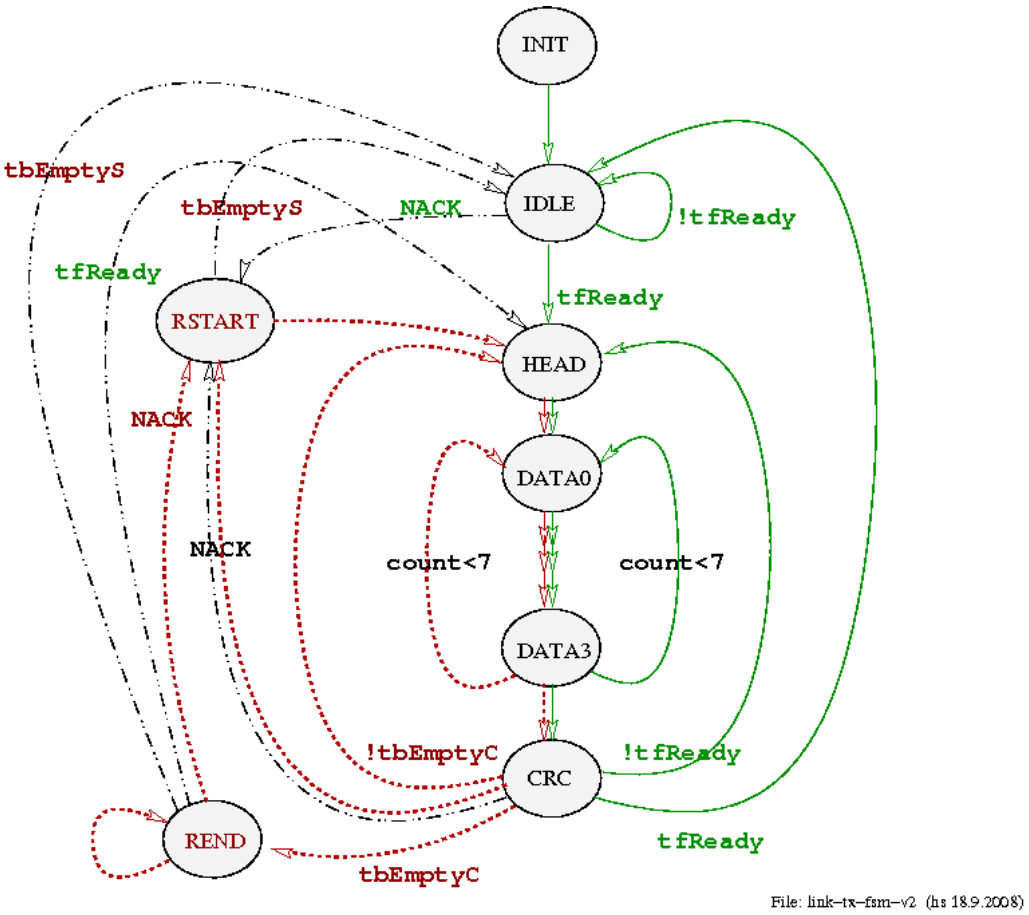
- Outgoing feedback commands (generated by rxLink upon receiving a packet, and to be sent accross the physical link) are passed from rxLink by the two signals, `fbOutReq` (NRZ encoded), and `fbOutVal`.
- `fbOutReq` is ignored if `rxOff` (rather than `offline`) is asserted. This allows unidiractional data transmission even if txLink on the receiver node is offline. Although rxLink will usually not generate outgoing feedback commands when it is offline (by `rxOff`), the suppression of `fbOutReq` by `rxOff` may be needed to avoid generation of spurious outgoing feedbacks when rxLink has no clock (e.g. during reset of DCM). Therefore, the suppression is done by `rxOff` in the clock domain of txLink.
- The resulting signal after NRZ decoding, `foAct` (and `foVal`), is active (within txLink) only for 1 cycle. It immediately suspends the FSM for 1 cycle to create a "hole" that allows insertion of the feedback command. In the corresponding state (one cycle after `foAct`) the signal `OE` is de-asserted.
- The signals `foAct` (and `foVal`) are also handled corrctly when `offline` is asserted.

- A counter `foRefCnt` is used to enumerate each outgoing feedback

Handling of Incoming Feedback Commands:

- Incoming feedback commands (received by rxLink from the physical link) are passed to txLink by two signals, `fbInReq`, `fbInVal`, plus the 8-bit counter value `fbInCnt`. The signal `fbInReq` indicates a valid feedback values on the other lines when ever it toggles (NRZ encoding).
- Incoming feedback commands are ignored (and not captured) if `offline` is asserted.
- All information from incoming feedback commands is captured in the registers `fiReq`, `fiVal`, and `fiCnt`. A request, indicated by `fiReq`=1, is only cleared upon assertion of `fiDone`.
- An incoming ACK is handled immediately (by asserting `tbIncS` to increment the start pointer in txBuffer by 1 packet).
- An incoming NACK may only be handled, after transmission of a packet is completed (or resending the entire content of txBuffer), hence `fiDone` is only asserted when the FSM reaches RSTART
- A counter `fiRefCnt` is used as a reference counter. It is compared with the counter value received together with each incomming feedback command. Occurrence of 3 consecutive mismatches will raise an exception

Main Control FSM of txLink:



The complete state of the FSM is defined by three registers:

- `state` corresponds to the states of the above diagram
- `resend` indicates resend mode and is set in RSTART and unset in REND. In resend mode, only the dotted red transitions in the left half of the above diagram are possible, hence only the states RSTART, HEAD, DATA0, DATA1, DATA2, DATA3, CRC, and REND are accessible. Otherwise, if not in resend mode, only the green transitions in the right half of the above diagram are possible. The dash-dotted black transitions in the above diagram correspond to a change of `resend`.
- `count` counts and controls the 8 repetitions of the sequence DATA0, DATA1, DATA2, and DATA3 (and is irrelevant in any other state)

Note that in the same clock cycle, t, in which FSM is in state HEAD, DATA0, ..., CRC, the corresponding data is assumed to be at the *input* of register `rData`. One cycle later, at $t + 1$, the corresponding data appears at the output of register

rData, and is written to txBuffer when not in resend mode. Otherwise, when in resend mode, the data does *not* pass through rData, but is read from txBuffer in clock cycle $t+1$ (and input to the next register rrData).

States and transitions of the main control FSM:

- INIT is the default state enforced by rst or offline. Transition to IDLE occurs when offline is de-asserted.
- IDLE is left to HEAD when tfReady (see below) is asserted or to RSTART when an incoming NACK feedback is received.
- HEAD corresponds to transmission of the header column and is left to DATA0 immediately, unless delayed by foAct.
- The states DAT0, DATA1, DATA2, DATA3, are cyclically passed 8 times to transmit all data columns of a packet. Each state is left immediately, unless delayed by foAct.

  If not in resend mode, tfRen must be asserted in DATA3 (and when count$\neq$7) to read txFifo.

  If in resend mode, tbIncC must be asserted during DATA3 and count=7 to have the updated value of tbEmptyC available in CRC.
- CRC corresponds to transmission of the CRC, i.e. the last column of of a complete packet. Assertion of offline always forces transition to INIT (not shown in the figure).

  An incomming NACK feedback always yields to a transition to RSTART (independent of resend and tbEmptyS, i.e. a NACK during resend mode can terminate the resending at the current level before being complete).

  Otherwise, if resend=0, a transition to IDLE or directly to HEAD occurs depending on availability of data from txFifo indicated by tfReady.

  If resend=1, a transition to REND or HEAD occurs depending on whether all data from txBuffer has been resent as indicated by tbEmptyS.
- ✘ IPG can be reached from CRC to enforce an IPG with idle columns after a complete packet and is not shown in the figure
- RSTART initiates a new resend sequence and is left to HEAD after 1 cycle, unless delayed by foAct. During RSTART a RESTART command column is transmitted over the link, and assertion of tbSetC sets the read pointer to the oldest data packet in txBuffer.

  If there is no data to be resent (tbEmptyS is asserted), e.g. because the incoming NACK was spurious, RSTART is left to IDLE and resend is set to 0. If this NACK was due to timeout of rxLink upon a spurious data, all subsequent incoming feedbacks will have a counter mismatch.

- REND is left when further data is ready to be sent from txFifo (assertion of `tfReady`) or when for all resent data an incoming ACK feedback has been received (assertion of `tbEmptyS`). A direct transition to RSTART occurs if an incoming NACK feedback is received (to enter a new resend sequence at the next higher level).

Effective FSM to read txFifo:

Reading of txFifo is controlled by register `tfRen`, which must be asserted for exactly 1 clock cycle and early enough to have the data available at the input mux Mux1 when the main FSM reaches the corresponding state (HEAD or DATA0). Moreover, `tfRen` must not be asserted too early, such that the new data does not arrive before the entire 144-bit word is actually consumed (in the first cycle of DATA3 , which however may be entered with a delay of 1 cycle, if a `foAct` is handled during DATA2).

Reading of txFifo is therefore controlled by an effective FSM (separate states from the main FSM, but of course, transitions are mutually conditioned), which is encoded in the registers `tfRen` and `tfReady`.

effective state	`tfReady`	`tfRen`
"WAIT"	0	0
"READ"	1	1
"LOCK"	1	0

The effective state "WAIT" is the default (e.g. after `rst`) to wait until `tfRen` can be asserted for 1 clock cycle in the effective state "READ" (which can not be reached directly from "LOCK" to avoid repeated assertion of `tfRen` while the data is not yet completely consumed).

The transition from "WAIT" to "READ" occurs if txFifo is not empty (`tfEmpty`=0) and if the transition of the main FSM into the next (possibly data-consuming) state is not delayed by `foAct`.

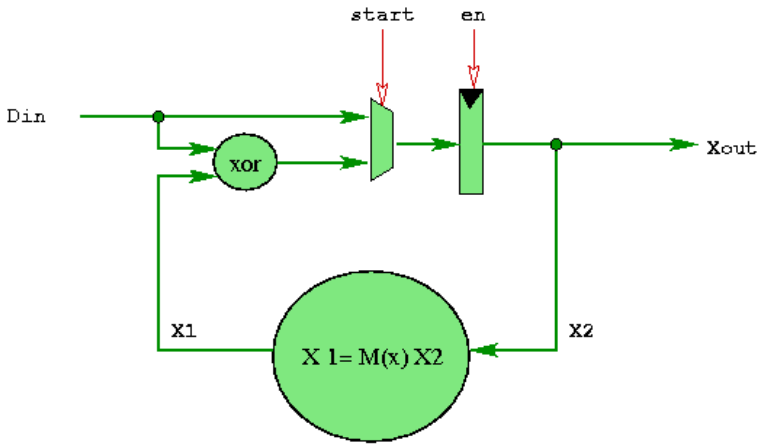
The transition from "READ" to "LOCK" (de-assertion of `tfRen`) occurs always after 1 clock cycle.

The transition from "LOCK" to "WAIT" (de-assertion of `tfReady` occurs when a sufficient part of the data has been consumed, such that the subsequent transition to "READ" can not occur before all data has been consumed. This is achieved by performing the transition from "LOCK" to "WAIT" at the same clock edge, when the main FSM leaves DATA1 .

The register `tfReady` indicates that data for Mux1 is already available or is guaranteed to be available in the next clock cycle. When not in resend mode, all transitions of the main FSM to HEAD are conditioned by `tfReady`.

5.7 CRC

- ✘ ??
- ✘ Current polynomial: `0x04c11db7`
- ✘ Current implementation:



File: crc-gen-v0 (hs 18.7.2008)

5.8 Error Handling in txLink

source	severity	handling	comment
TX_FAULT		EXC	TX_EXC_FAULT (free running) *asynchronous* input from PHY indicating: • internal transmit FIFO error • loss of PLL lock in CSU (always enabled and independent of SD) Cleared by reading Reg 0x0008/0xC008
NACK in fbIn	minor	FSM DIAG	enter start resend mode increment NACK counter
NACK Ovfl	fatal	EXC	TX_EXC_NACK overflow of NACK counter
RESEND Ovfl	fatal	EXC	TX_EXC_RESTART too many NACKs while in resend mode
Feedback Error	fatal	EXC	TX_EXC_FBCNT ≥ 3 consecutive feedbacks with counter mismatch
	minor	DIAG	isolated feedback with counter mismatch
Header Error	fatal	EXC	TX_EXC_HEAD missing first at start of packet
txFifo Empty	fatal	EXC	TX_EXC_EMPTY unexpected during packet (or timeout)
	—	—	otherwise normal idle situation
txBuffer Full	fatal	EXC	TX_EXC_FULL feedbacks lost (or txBuffer too small)
fbIin Ovfl	fatal	EXC	TX_EXC_FBIN accumulation of un-handled incoming feedbacks (i.e. design error in FSM)
fbOut Ovfl	fatal	EXC	TX_EXC_FBOUT accumulation of un-handled outgoing feedbacks

6 rxLink

6.1 Signals from PHY

- RX_CLK
- RXD[31:0]
- RXC[3:0]

- RXH[3:0]
- RX_FAULT (asynchronous)
- INTB (asynchronous, active-low)

6.2 Interface between rxDecoder and rxLink

dir	bits	signal	comments
in	32	data1	unchanged from RXD
in	1	valid	asserted also during Start and Restart?
in	1	flag1A	decoded ACK feedback column
in	1	flag1N	decoded NACK feedback column
in	1	flag1R	decoded RESTART command column
in	1	flag1I	decoded IDLE column

6.3 Interface between rxBuffer and rxLink

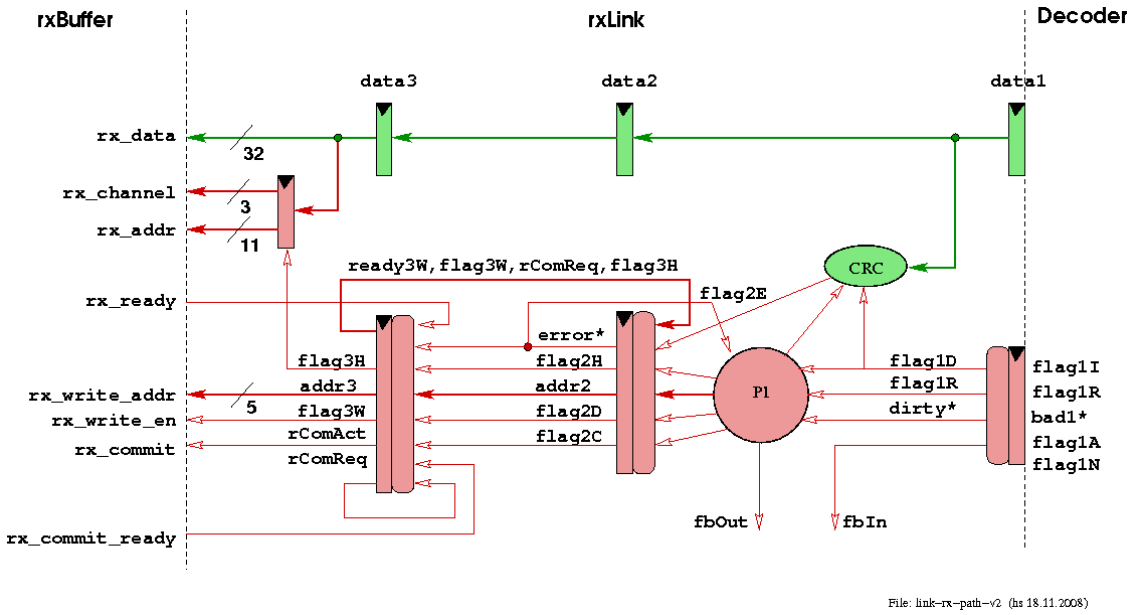
According to owc_link_module_description.pdf [email sh 21.5.2008] (and signal names according to recpack.vhd):

dir	bits	function	comments
out	32	data	
out	1	writeEn	
out	5	writeAddr	32-bit chunk within 128B packet
in	1	ready	RAM storage available
out	11	addr	packet address offset (from header)
out	3	channel	channel ID (from header)
out	1	commit	proceed to next packet
in	1	commitReady	addr_multfifo not full

Remarks:

- channel might be changed to a channel selector (decoded to N_{ch} bits), because in rxLink there is lots of time for decoding
- channel must be valid during 2 cycles (before and during commit)
- commitReady takes into account channel and allows rxLink to wait with commit (of the last packet already written to BRAM and with successful CRC check) until success is guaranteed. If necessary, only subsequently received packets need to be discared and NACK'ed.

6.4 Data Paths in rxLink



6.5 Input Decoder of rxLink

In the first two pipeline stages the following flags are generated:

- `flag1A` if for any 2 of the 3 lanes 0, ..., 2 (`RXD,RXC`) matches ACK
- `flag1N` if for any 2 of the 3 lanes 0, ..., 2 (`RXD,RXC`) matches NACK
- `flag1R` if for any 3 of the 4 lanes 0, ..., 3 (`RXD,RXC`) matches RESTART
- `flag1I` if for any 3 of the 4 lanes 0, ..., 3 (`RXD,RXC`) matches IDLE
- `flag1K` if for any 3 of the 4 lanes 0, ..., 3 `RXC` is asserted
- `flag1E` if any of the 4 lanes 0, ..., 3 (`RXD,RXC`) matches $/E/$

and

- `bad1A` if *not* in all 3 lanes 0, ..., 2 (`RXD,RXC`) matches ACK or if `RXC(3)`=1
- `bad1N` if *not* in all 3 lanes 0, ..., 2 (`RXD,RXC`) matches NACK or if `RXC(3)`=1
- `bad1R` if *not* in all 4 lanes 0, ..., 3 (`RXD,RXC`) matches RESTART
- `bad1I` if *not* in all 4 lanes 0, ..., 3 (`RXD,RXC`) matches IDLE
- `bad1D` if `RXC`=1 in any of the 4 lanes 0, ..., 3

In the next pipeline stage (the first stage of the rxLink logic, hence denoted by 1) a data column is assumed and signaled by `flag1D` if none of the above flags (`flag1A or flag1N or flag1I or flag1R or flag1K`) are asserted.

Error detection and diagnostics uses the 5 flags (X = `A, N, R, I, D`)

$$\texttt{dirty}X \texttt{ <= flag1}X \texttt{ and bad1}X$$

6.6 Conditions for Decoding Errors

With decreasing probability, we can have the following decoding error cases

Error	Condition
Spurious Data	Lost CMD*) (but no spurious CMD)
Lost $\|\|A\|\|$ or $\|\|R\|\|$	At least 1 of the 4 lanes 0, ..., 3 has an error on (`RXD,RXC`)
Lost IDLE or RESTART	At least 2 of the 4 lanes 0, ..., 3 have an error on (`RXD,RXC`)
Lost ACK or NACK	At least 2 of the 3 lanes 0, ..., 2 have an error on (`RXD,RXC`)
Lost Data	Spurious CMD (but no lost CMD)
Spurious ACK or NACK	At least 2 of the 3 lanes 0, ..., 2 have an error on (`RXD,RXC`) *and* match condition for `flag1A` or `flag1N`
Spurious IDLE or RESTART	At least 3 of the 4 lanes 0, ..., 2 have an error on (`RXD,RXC`) *and* match condition for `flag1I` or `flag1R`

*) In the above table CMD denotes any of ACK, NACK, RESTART, or IDLE.

6.7 Robustness of the Protocol against Errors

Error	Context	Link Behaviour
Modified Data	packet	`errorC` or data corruption[0)]
Lost $\|\|A\|\|$ or $\|\|R\|\|$	IDLE	ignored or PHY error
Spurious Data	IDLE	`errorT`
	packet	`errorC` or data corruption[0)]
Lost IDLE	any	see other spurious cases
Lost RESTART	RESTART	rxLink frozen in RESTART, eventually `TX_EXC_BFULL`
Lost ACK or NACK		eventually `TX_EXC_FBCNT`
Lost Data	isolated packet	`errorT`
	packet stream	`errorC` or data corruption[1)]
Spurious ACK or NACK	any	eventually `TX_EXC_FBCNT` ✘ CHECK VHDL 2do
Spurious RESTART	RESTART	`errorC` or data corruption[2)]
	other	ignored
Spurious IDLE	packet	see other lost cases

[0)] if error is not detected by CRC

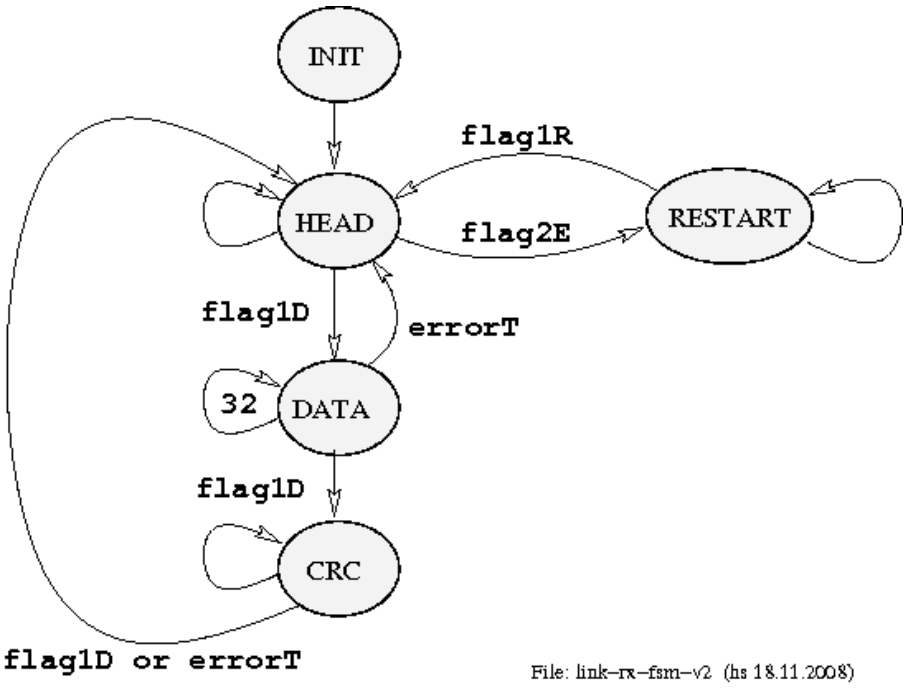
[1)] eventually `errorC` will occur for some later packet, but error in current packet might not be detected by CRC

[2)] if the spurious RESTART is followed by a full packet (with good CRC) that has been sent by txLink before receiving the NACK which caused rxLink to enter RESTART

For details on the handling of `errorT` see section 6.10.

6.8 Control Logic of rxLink

Details of FSM (P1):



XXX THE FOLLOWING DESCRIPTION NEEDS TO BE UPDATED!!!

- In case of consecutive packets without IPG, commit must be performed within one cycle (in commit) and simultaneous with the registration of the next header. If new data can not be written to the BRAM in rxBuffer already during the cycle immediately following commit, an extra register might be inserted to delay the path of `rx_data`.
- In start and commit any incoming data (i.e. a potential header) is registered. At some later time, but before crc, this register must be forwarded to a second register, from where the address and channel ID for the commit is extracted.
- In start and commit any incoming data and absence of `ready` (no space in RAM) must be registered as an error (and suppress any write-enable to the RAM) or lead to a direct transition (not shown) to nack (possibly with some delay before generating the NACK)
- errors from PHY decoding during data (or during commit or start if `data`. i.e. if it is considered a header) must be accumulated until crc or lead to a direct transition (not shown) to nack (possibly with some delay before generating the NACK)
- In commit an ACK is transmitted (only when entering from crc)
- wait and the dotted transitions are a possible optimisation to allow one pending commit in case of `full` (no space in `addr_fifo`). When entering wait an

ACK for the pending commit can already be generated in the first cycle. When leaving `wait` to `nack` (because further data arrives before the commit was compleded) a some dalay might be needed to avoid a too short time intervals between subsequent feedbacks.

- In `restart` no output signals are asserted (in particular, no feedbacks), and `restart` is only left upon receiving a RESTART

6.9 Error Handling in rxLink

source	severity	handling	comment
RX_FAULT		EXC	RX_EXC_FAULT (free running)
			configurable by Reg 0xd003 (default=enabled):
INTB		EXC	RX_EXC_INT (free running)
			configuration see sect. 6.13 (default=disabled)
			cleared by reading Reg 0xd$i$0b
Restart Timeout	fatal	EXC	Not (yet) implemented
			✘ no restart received after error
Packet Timeout	fatal	EXC	RX_EXC_TP
			no valid data while within good packet
		DIAG	increment errorT counter
RXH			8b/10b code violation or disparity
			✘ not yet handled explicitly
			(should always lead to some dirty$X$ error)
	minor	FSM	recoverable by resend
	fatal	EXC	✘ if corrupt feedback or RESTART?
		DIAG	increment cntRxh counter
/E/			PHY generated /E/ (implies RXH???)
			✘ not yet handled explicitly
			(should always lead to some dirty$X$ error)
	minor	FSM	recoverable by resend
	fatal	EXC	✘ if corrupt feedback or RESTART?
		DIAG	increment cntRxe counter
Decode Error	minor	FSM	suspend data Rx, wait for RESTART
	fatal	EXC	✘ if risk of protocol corruption?
		DIAG	increment errorD counter
CRC Error	minor	FSM	suspend data Rx, wait for RESTART
		DIAG	increment errorC counter
RESTART	minor	FSM	resume suspended data Rx
		DIAG	✘ counter (see remark below)?
Data FIFO Full	—	FSM	normal back-pressure:
			suspend data Rx, wait for RESTART
			feedback NACK (or BUSY?)
		DIAG	increment errorW counter
Header FIFO Full	—	FSM	normal back-pressure:
			suspend data Rx, wait for RESTART
			feedback NACK (or BUSY?)
		DIAG	increment errorH counter

Remark: The number of RESTART is less or equal to

$$\texttt{cntErrC} + \texttt{cntErrD} + \texttt{cntErrH} + \texttt{cntErrW} + \texttt{cntErrT}$$

(since `errorT` does not cause a NACK while waiting for first data)

6.10 Handling of Packet Timeout (`errorT`)

The counter `cntTO` is set to `RX_TO_CNT_MAX` during HEAD and decremented every clock cycle without `flag1D` or `flag1A` or `flag1N`. The signal `errorT` is asserted when `cntTO` becomes zero. However, we distinguish the following three cases (since rxLink.vhd v0.16 in torus 4304 1130):

`errorT` during	VHDL signal	NACK	exception	comment
first DATA	—	no	no	spurious data during IDLE
later DATA	`flag1Tb`	yes	yes	
CRC	`flag1Tc`	yes	no	lost data during packet

while in all three cases the next state is HEAD and `cntErrT` is incremented.

6.11 Behaviour and Configuration of RX_FAULT

- RX_FAULT_P/R are *asynchronous* output pins of the PHY
- Assertion is persistently registered in Reg 0x0008/0xc008 (MDIO).
  This MDIO register is cleared only by reading it (if WICMODE).
- Assertion is persistently registered in exception bit RX_EXC_FAULT (DCR).
  This DCR register is cleared only by issuing rxRst or by writing it.
- RX_FAULT is the logical OR of the following conditions
  - deasserted SD pin (if SD_INCL)
  - transition detect (D_TRANS_V if D_TRANS_INCL)
  - loss of syncronisation in DRU (RX_SYNC_V if RX_SYNC_INCL)
  - RX FIFO error (always)
- RX_FAULT is deasserted as soon as none of the above conditions holds
- Conditions are configured by *_INCL bits in MDIO registers
  (values in parenthesis refer to the redundant link)

Register 0xd003:

mask (1 bit per *link*)	name
0x0040 (0x0004)	D_TRANS_INCL_P
0x0020 (0x0002)	SD_INCL_P
0x0010 (0x0001)	RX_SYNC_INCL_P

Register 0xd004 (0xd005):

mask (1 bit per *lane*)	name
0x00f0	D_TRANS_INCL_P<l>
0x000f	RX_SYNC_INCL_P<l>

- By default *all* sources are enabled (i.e. all INCL bits are 1 after phyRst)

6.12 Sources of INTB

In the following table $i = 1, 2$ is to be used in register addresses for primary/redundant links, respectively, and analogous $j = 3, 4$.

source	status	enable	description
LOCKI	0xd0b1	y?	CSU lock to REFCLK (also included in TX_FAULT)
SD_*_I	0xd006	n	SD pin (see Reg 0xd004 for inclusion in RX_FAULT)
D_TRANS_DET_PR[*]_I	0xd006	n	transition detect (see Reg 0xd004 for inclusion in RX_FAULT)
SYNC_ERRI[3:0]	0xdi0b	n	loss of byte alignment (see Reg 0xd004 for inclusion in RX_FAULT)
ALIGN_ERRI	0xdi0b	y?	loss of lane alignment (trunked mode only)
SDI[3:0]	0xdi0b		signal detect input
HICRI[3:0]	0xdi0b	y?	overflow of 8b/10b error counter (see Reg 0xdi04 for thresholds)
LF_DETI	0xdi0b	?	Local Fault $\|\|LF\|\|$
RF_DETI	0xdi0b	?	Remote Fault $\|\|RF\|\|$
UNDERRUNI[3:0]	0xdi0c	y?	FIFO underrun
OVERRUNI[3:0]	0xdi0c	y?	FIFO overrun
PGC_FRAME_SYNCI[3:0]	0xdi0d	n	frame synchronisation (only lane 0 in trunked mode)
PGC_SYNCI[3:0]	0xdi0d	n	pattern synchronisation (only lane 0 in trunked mode)
PGC_ERRI[3:0]	0xdi0d	n	expected pattern value (only lane 0 in trunked mode)
PC_ERRI[3:0]	0xdi0d	n	packet counter rules (see 14.4.5 of [1]) (only lane 0 in trunked mode)
LOST_SYNC_I	0xdj04	y?	FIFO read FSM in re-sync state
Q_DET_INT_I	0xdj04	n	primitive sequences detected on XGMII

6.13 Behaviour and Configuration of INTB

- INTB is an *asynchronous* and *active-low* output pin of the PHY (and not directly mapped to any MDIO register)
- INTB may be asserted $O(17)$ cycles before the corresponding column appears on RXD
- INTB = or_reduce(I and E)
- The I-bits listed in subsection 6.12 are
  - asserted at a rising edge of the corresponding V-bit
  - persistently registered
  - cleared only by reading (if WICMODE)
- To guarantee deassertion of INTB the MDIO registers Reg 0xd*i*0b, ..., 0xd*i*0d need to be read
- By default *all* sources are disabled (all E-bits are 0 after phyRst)

The V-, I-, and E-bits of the interrupt sources are mapped to the following MDIO registers (a "+" indicates 1 bit per *lane*, register addresses in parenthesis refer to the redundant link):

source	mask	registers		
		V (RO)	I (RO)	E (RW)
D_TRANS_DET	*)	0xd003	0xd006	0xd003
SD_<link>	*)	"	"	"
XGMII_FIFO0_ERR	*)	—	"	—
XGMII_FIFO1_ERR	*)	—	"	—
D_TRANS_DET_P	*) +	0xd004 (0xd005)	0xd007 (0xd008)	0xd004 (0xd005)
RF_DET	0x4000	0xd10e (0xd20e)	0xd10b (0xd20b)	0xd108 (0xd208)
LF_DET	0x2000	"	"	"
ALIGN_ERR	0x1000	"	"	"
HICER	0x0f00 +	"	"	"
SD	0x00f0 +	"	"	"
SYNC_ERR	0x000f +	"	"	"
OVERRUN	0x00f0 +	0xd10f (0xd20f)	0xd10c (0xd20c)	0xd109 (0xd209)
UNDERRUN	0x000f +	"	"	"
PC_ERR	0xf000 +	0xd110 (0xd210)	0xd10d (0xd20d)	0xd10a (0xd20a)
PGC_ERR	0x0f00 +	"	"	"
PGC_SYNC	0x00f0 +	"	"	"
PGC_FRAME_SYNC	0x000f +	"	"	"

*) The corresponding registers have an irregular layout as follows:

Register 0xd003:

mask	name	comment
0x8000 (RW)	D_TRANS_DET_P_E	E primary
0x4000 (RW)	D_TRANS_DET_R_E	E redundant
0x2000 (RW)	SD_P_E	
0x1000 (RW)	SD_R_E	
0x0800 (RO)	D_TRANS_DET_P_V	V primary
0x0400 (RO)	D_TRANS_DET_R_V	V redundant
0x0200 (RO)	SD_P_V	
0x0100 (RO)	SD_R_V	
0x0080 (RW)	SD_INV_P	INV/INCL primary
0x0040 (RW)	D_TRANS_INCL_P	
0x0020 (RW)	SD_INCL_P	
0x0010 (RW)	RX_SYNC_INCL_P	
0x0008 (RW)	SD_INV_R	INV/INCL redundant
0x0004 (RW)	D_TRANS_INCL_R	
0x0002 (RW)	SD_INCL_R	
0x0001 (RW)	RX_SYNC_INCL_R	

Register 0xd006:

mask	name	comment
...		
0x0008 (RO)	D_TRANS_DET_P_I	I primary
0x0004 (RO)	SD_P_I	I primary
0x0002 (RO)	D_TRANS_DET_R_I	I redundant
0x0001 (RO)	SD_R_I	I redundant

Register 0xd004 (0xd005):

mask	name	comment
0xf000 (RW)	D_TRANS_DET_P<lane>_E	+) E
0x0f00 (RO)	D_TRANS_DET_P<lane>_V	+) V
0x00f0 (RW)	D_TRANS_INCL_P<lane>	+) INCL
0x000f (RW)	D_SYNC_INCL_P<lane>	+) INCL

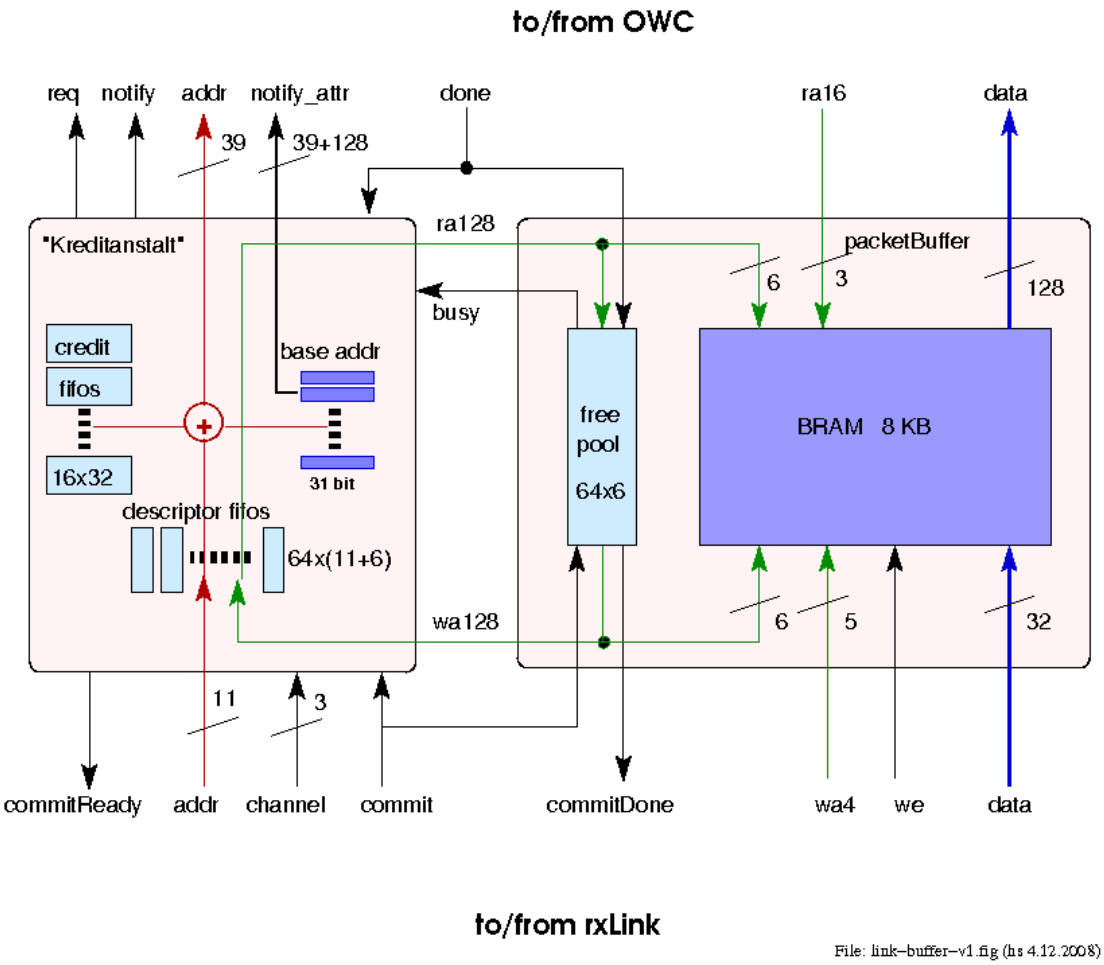
Register 0xd007 (0xd008):

mask	name	comment
...		
0x000f (RO)	D_TRANS_DET_P<lane>_I	+) I

7 rxBuffer

✘Size of packetBuffer has been doubled (to $S_{\text{packetBuffer}} = 16$ KB, $d = 1024$) in torus4 version 4308 4010

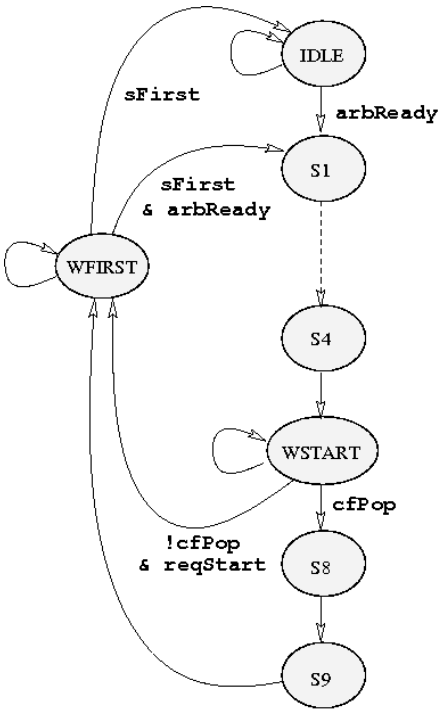
7.1 Data Paths of rxBuffer



7.2 Control Logic of rxBuffer

Control of rxBuffer is performed by 2 coupled FSMs

- PFSM (P20) to arbitrate and prefetch the next request
- OFSM to handle the handshake with OWC to issue the request



The states of PFSM are shown in the above figure. PFSM is controlled by

- `arbReady` (combinatorial) is generated by the arbiter whenever a channel has a request to the arbiter asserted (i.e. data and a corresponding credit for any channel is available)
- `reqStart` (1-cycle pulse) is generated when OFSM is in state START. This is the first cycle when `owcReq` has toggled to a new value, and is equivalent to the last (and possibly only) cycle, when PFSM is in state WFIRST
- `sFirst` (1-cycle pulse) is generated when OWC asserts "first"

It must be guaranteed that `sFirst` is *not* asserted before WFIRST is reached, i.e. during the first 3 cycles after `owcReq` has toggled, otherwise `sFirst` is lost and PFSM is stuck in WFIRST.

The channel-wise counters `chCnt` define a sub-state machine of PFSM, and are

- read during S2 to update `cgCnt` (the counter of the current granted channel)
- updated with `cgCnt-1` when `reqStart` is asserted (i.e. during the last cycle in WSTART)

OFSM is encoded implicity in the signals `reqStart` and `reqActive`

state	encoding		transitions
	`reqStart`	`reqActive`	
IDLE	0	0	START when `flag2`
START	1	0	ACTIVE after 1 cycle
ACTIVE	0	1	IDLE when `sDone` and `not(flag2)`
			START when `sDone` and `flag2`

and controlled by

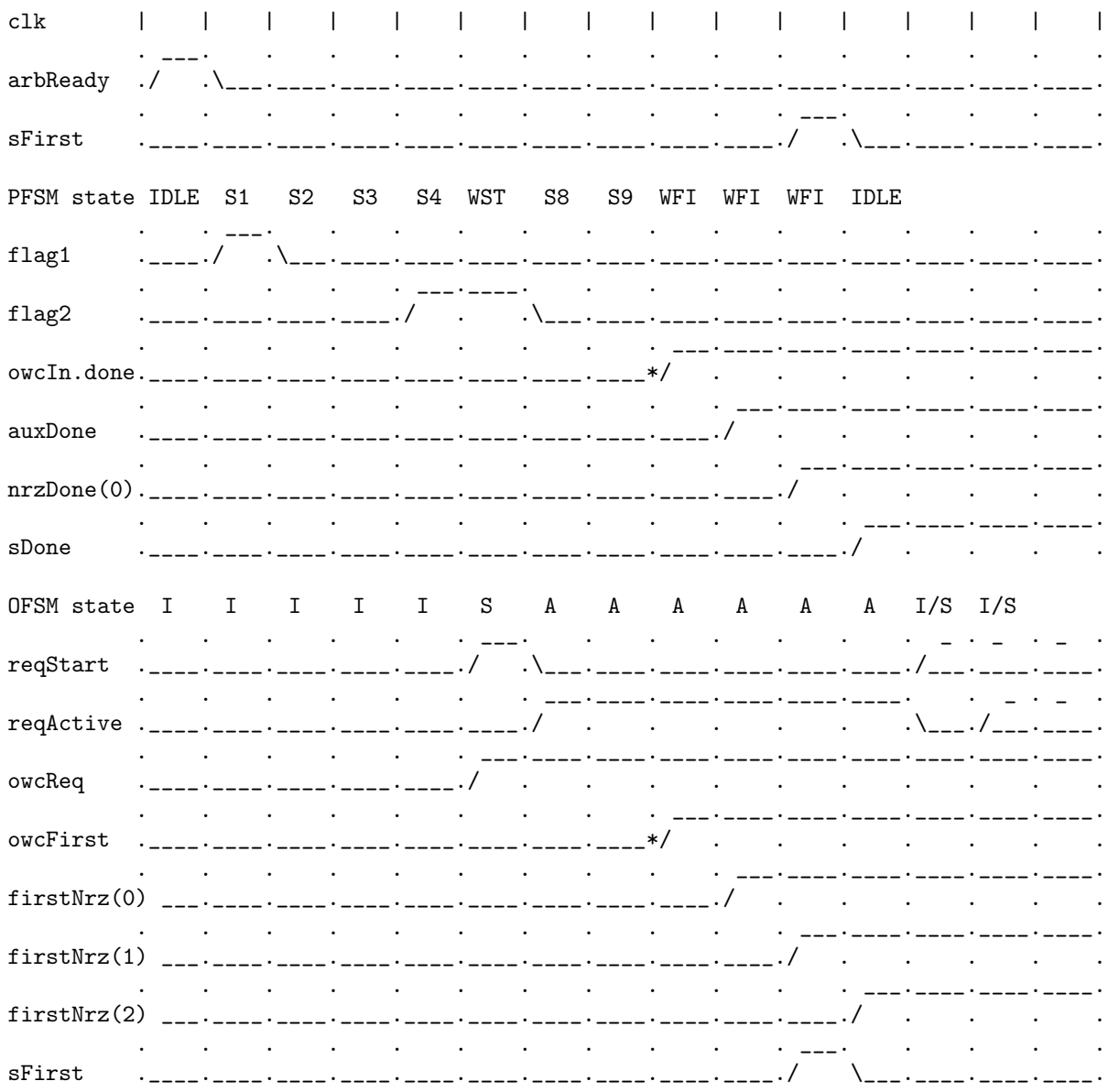
- `flag2` which is asserted when PFSM is in state S4 or WSTART
- `sDone` which is asserted whenever the "done" output of OWC is equal to the request input (i.e. the "done" corresponding to the last request has occured)

It must be guaranteed that `sDone`

- is asserted after each toggling of `owcReq`
- is *not* asserted during the first cycle after `owcReq` has toggled

but there is no timing constraint between `sDone` and `sFirst` (i.e. assertion of `sFirst` after `sDone` does no any harm, although it does not make much sense).

Minimal timing (direct transition from S4 to WSTART, no cfPop)



The $*$ at the rising edges of `owcFirst` (= `owcIn.firstNRZ`) and `owcIn.done` indicates the assumption that these two signals can not be asserted less than 3 cycles after toggling of `owcReq`. For `owcFirst` this is safe if $5T_{\texttt{gbif\_clk}} > 3T_{\texttt{rxClk}}$, because `owcReq` needs to pass 2 synchronisation registers and takes 1 cycle to start the FSM of OWC. Then, `first` is asserted after 1 non-idle cycle, and (currently) `owcFirst` is toggled after 1 further cycle. If OWC generates `first` as NRZ signal and the latter register is dropped, we need $4T_{\texttt{gbif\_clk}} > 3T_{\texttt{rxClk}}$.

7.3 packetBuffer

7.4 multiFifo

8 Reset, Initialisation, Erros, Diagnostics

In the following we use the shorthand notations

- `phyRst` for reset to PHY
- `dcmRst` for reset to DCM
- `rxRst` for reset to rxLink and rxBuffer
- `rxOff` for offline to rxLink and rxBuffer
- `txRst` for reset to txLink
- `txOff` for offline to txLink
- suffix [D] for a signal or entity on the "data path",
  e.g. txLink[D] on the sending node
  and rxLink[D] on the receiving node
- suffix [F] for a signal or endity on the "feedback path",
  e.g. rxLink[F] on the sending node
  and txLink[F] on the receiving node

8.1 Offline Behaviour

Open issues:

- ✘ Review behaviour of txLink for `offline`
  - writing of data from IWC is suppressed
  - data loss (if IWC writes during `offline`) is detected by `TX_EXC_OFF`
  - back-pressure (`AlmostFull`) to IWC is suppressed
  - outgoing feedbacks are handled (also when `offline` is asserted),
    but suppressed by `rxOff` (see also section 5.6)
  - handling of incoming feedbacks is suppressed
  - entry upon exception?
- ✘ Review behaviour of rxLink and rxBuffer for `rxOff`
  - data loss (if received from link while `rxOff`) is detected by `RX_EXC_OFF`
  - incoming feedbacks are always handled and forwarded to txLink
  - outgoing feedbacks should be suppressed (`flag2E` needs proper suppression of `error*`)
    ✘ CHANGE VHDL 2do

- new (and pending???) commit should be suppressed (`flag1C`, `flag2C` depend on FSM, but maybe also reset `rCom*`???)
  ✘ CHANGE VHDL 2do
- incoming data and/or WE to rxBuffer (e.g. `flag1D`) should be suppressed
  ✘ CHANGE VHDL 2do
- generation of `error*` should be suppressed
  ✘ CHANGE VHDL 2do
- rxLink FSM should be driven into INIT (from CRC and RESTART)
  ✘ CHANGE VHDL 2do
- rxLink should enter `offline` only from HEAD?
- suppress requests to OWC (by FSM?)
- rxBuffer should enter `offline` only after "done" and before reqest to OWC
- enter `offline` upon exceptions?

✘ Review $\|I\|$ transmission during reset

✘ detection of corrupted packets to OWC during reset

8.2 Rules for Reset and Initialisation

The following situations may cause undefined behaviour or errors and are **not allowed**:

(E) `rstDcm` or `rstPhy` while accessing DCR registers in rxLink
(clock in RX domain is not running)

(E) `rstDcm` or `rstPhy` while rxLink[F] is receiving incoming feedbacks
(clock in RX domain is not running)

(E) data transmission after `txRst`[D] without `rxRst`[D] on receiver node
(will eventually lead to `TX_EXC_FBCNT` after ≥ 3 packets)

(E) data transmission after `rxRst`[D] without `txRst`[D] on sender node
(will eventually lead to `TX_EXC_FBCNT` after ≥ 3 packets)

(E) `txRst`[F] while transmitting outgoing feedback from local rxLink[D]
(but OK inbetween outgoing feedbacks???)

(E) `rxRst`[F] while local txLink[D] is not offline
(may cause spurious feedback)
✘ POSSIBLY CHANGE VHDL (reset P40 in rxLink by `txRst`)

(E) rxRst[D] while data (or credits) in rxBuffer
(packet buffer and credit fifo are flushed)

(E) `rxOff`[D] de-assertion while receiving data from link
(timeout or incorrect CRC for first packet, OK after RESTART)

(E) `txOff`[D] de-assertion while OWC writes data
(is partially ignored and causes `TX_EXC_HEAD` during subsequent packet)

The following operations are **safe**:

(S) `txRst`[D] without txRst[F] on corresponding remote node
(reset `foRefCnt` in txRst[F] by `rxRst`[D])
✘ CHANGE VHDL 2do

8.3 Reset and Initialisation Sequence

Control Operations via DCR:

- reset PHY
- reset DCM
- configure PHY ports (denoted by `phyConfig` and described in section 8.4)
- reset txLink: FSM, FB counters, txFifo (both sides), txBuffer
- reset rxLink: FSM, FB counters, credits
- deassert offline of txLink: suppress all inputs from IWC or rxLink (and all exceptions)
- deassert offline of rxLink: suppress all outputs to OWC or txLink (and all exceptions)

This corresponds to the following DCR operations on the receiver node

```
w tx<n> 2  0x101X       # reset to PHY
r tx<n> 2  0x201X       # optional check
w tx<n> 2  0x011X       # reset to DCM
r tx<n> 2  0x021X       # optional check
w mdio<n> 0xd003 0x00PR # select SD:
                        #   PR = 0xF5 for primary
                        #   PR = 0x5F for redundant
w tx<n> 2  0x003X       # reset to rxLink
r tx<n> 2  0x005X       # optional check
w tx<n> 2  0x000X       # rxLink online
```

and on the transmitter node

```
w tx<n> 2  0xXXX3     # reset to txLink
r tx<n> 2  0xXXX5     # optional check
w tx<n> 2  0xXXX0     # txLink online
```

Note:

- Reset of Tx must be performed *before* Rx at other end of the link (on an other node) is put online
- Rx and Tx at *both* ends of a link must be online *before* data are transmitted in any of the two directions (because of feedbacks)

Thus, initialization of a link between 2 nodes can be performed by performing on each node the sequence

```
...                   # optional reset of PHY and DCM
w tx<n> 2  0x0033     # reset txLink and rxLink
# --- barrier ---
...                   # optionally switch PHY ports
phyConfig             # configure PHY ports
w tx<n> 2  0x0000     # switch rxLink and txLink online
# --- barrier ---
...                   # start data transmission
```

where the operations above a line indicated by "`--- barrier --`" must be performed on *all* nodes (with arbitrary order between the nodes) *before* an operation below that line is performed on *any* of the nodes.

Open issues:

- ✘ See open issues in txLink
- ✘ How are RAM and addr_multififo in rxBuffer reset (on both clock domains)? Possibly by an inhibit/disconnect command (to let IWC or OWC complete handling of pending packets) followed by the actual reset
- ✘ Does rxLink need to trigger the initialisation of the free_addr_pool in rxBuffer?
- ✘ How should "open" (Tx and Rx) links be configured?

8.4 PHY Configuration (Primary/Redundant Link)

- The pin `SD_*` of the PHY is fixed to 0, indicating that there is no meaningful input on the serial links, and forcing $\|LF\|$ on XGMII.
- To start data reception, the appropriate link must be enabled by configuring the switch and asserting `SD_INV_*`

The following sequence of MDIO operations (denoted by `phyConfig` in the previous section) has to be performed to configure the primary or redundant link of the PHY (see function `torus_select_primary` in `svn/drivers/linux/torus/tnw.c`)

op	address	value	comment
w	0xD081	0x000F	enable switch
w	0xD090	0x0100	map lanes 0 and 1 to 0 and 1
w	0xD091	0x0302	map lanes 2 and 3 to 2 and 3
w	0xD080	0x0002	switch event
w	0xD080	0x0000	restore default state
w	0xD004	0xF0FF	enable TRANS and SYNC monitor for primary
w	0xD005	0x0000	disable TRANS and SYNC monitor for redundant
w	0xD003	0x00F0	enable `SD_INV_P`

or the redundant link

op	address	value	comment
w	0xD081	0x000F	enable switch
w	0xD090	0x0504	map lanes 0 and 1 to 4 and 5
w	0xD091	0x0706	map lanes 2 and 3 to 6 and 7
w	0xD080	0x0002	switch event
w	0xD080	0x0000	restore default state
w	0xD004	0x0000	disable TRANS and SYNC monitor for primary
w	0xD005	0xF0FF	enable TRANS and SYNC monitor for redundant
w	0xD003	0x000F	enable `SD_INV_R`

8.5 Machine Configuration

- The machine partitions (handling of global signals) are configured through registers on the RC.
- The network topology is configured via DCR from Cell (or SPI backdoor)

For further details see sections 2.4 and 2.5, as well as `notes-root`.

8.6 Errors

Sources / Use Cases

- fatal link failures (t.b.d.)
- timeout or deadlock situations (t.b.d.)
- downward KILL

Mechanisms

- Interrupt via `INTB`
- FIR (Failure Identification Register)
- CS (Checkstop via CPLD) to Cell

8.7 Diagnostics

✘This subsection is just a collection of items to be rewritten or placed elsewhere!

- Correlation between disparity and CRC errors

8.8 Debugging

✘This subsection is just a collection of items to be rewritten or placed elsewhere!

- txFifo empty
  (should never happen during packet, unless data from GBIF is not contiguous)
- Tx length $\neq$ 128 B (handled by IWC)

9 DCR Registers

All register addresses in this section refer to C3PO space.

9.1 DCR Slaves

slave	link	bit mask	address	registers
		SSSS SSSS SRRR RRRR		
TNW_TX	0-5	0011 0110 0LLL RRRR	0x36 <l>f	$O(10) \times N_{link}$
TNW_TB	0-5	0011 0110 1LLL RRRR	0x36 <8+l>f	$O(10) \times N_{link}$
TNW_RX<l>	<l>	0011 0LLL 000R Rrrr	0x3<l> 1f	$O(10) + 3 \times N_{ch} \leq 32$

9.2 DCR Register Map

address	description	names (VHDL)	
0x36 <l>0	Tx Exception Status	rExc	TX_RA_EXC
0x36 <l>1	Tx Exception Enable	rExEn	TX_RA_EXEN
0x36 <l>2	Tx Control and Status	rCtrl	TX_RA_CTRL
0x36 <l>3	MDIO Read Access	rMdioR	TX_RA_MDIO_R
0x36 <l>4	MDIO Write Access	rMdioW	TX_RA_MDIO_W
0x36 <l>5	Tx NACK Counter	cntNack	TX_RA_CNT_NACK
0x36 <l>6	Tx Resend Counter	cntResend	TX_RA_CNT_RESEND
0x36 <l>f	TNW revision	VERSION	
0x36 <8+l>6	Test Bench Registers	see sect. 11	
0x3<l> 00	Rx Exceptions and Errors	rExc,rErr	RX_RA_EXC
0x3<l> 01	Rx Exception Enable	rExEn	RX_RA_EXEN
0x3<l> 02	Rx Command Error Count		RX_RA_CMD
0x3<l> 03	Rx Data Error Count		RX_RA_DATA
0x3<l> 04	Rx Flow Error Count		RX_RA_FLOW
0x3<l> 05	Rx PHY Error Count		RX_RA_PHY
0x3<l> 06	Rx Fifo Status		RX_RA_FIFO
0x3<l> 07	Rx Control		RX_RA_CTRL
0x3<l> 1<c>	Notify Base Addresses	rNba	RX_RS_NBA
0x3<l> 2<c>	Credit Fifos	cfIn	RX_RS_CF
0x3<l> 3<c>	Credit Base Addresses	rCba	RX_RS_CBA

where

- <l> = 0, . . . , 5 labels the (physical) links
- <c> = 0, . . . , 7 labels the (virtual) channels

9.3 DCR Registers in TNW_TX

0x36<l>0: Tx Exception Status (rExc)

Default: 0x00000000

bits	mask	description	name (VHDL)
1 RW	0x00000001	TX_FAULT_P or TX_FAULT_R from PHY	TX_EXC_FAULT
1 RW	0x00000002	NACK overflow	TX_EXC_NACK
1 RW	0x00000004	restart counter overflow	TX_EXC_RESTART
1 RW	0x00000008	feedback mismatchs (3 consecutive)	TX_EXC_FBCNT
1 RW	0x00000010	invalid header (missing first)	TX_EXC_HEAD
1 RW	0x00000020	txFifo written while full	TX_EXC_FFULL
1 RW	0x00000040	txFifo empty during packet	TX_EXC_EMPTY
1 RW	0x00000080	txBuffer full (> 1KB)	TX_EXC_BFULL
1 RW	0x00000100	Data written by IWC while offline	TX_EXC_OFF
1 RW	0x00000200	fbIn accumulation	TX_EXC_FBIN
1 RW	0x00000400	fbOut accumulation	TX_EXC_FBOUT
8 RW	0x00ff0000	last restart counter	cntRestart
1 RO	0x10000000	txFifo not empty	tfEmpty
1 RO	0x20000000	txFifo not almost empty (≥ 8 packets)	tfAempty
1 RO	0x40000000	txFifo almost full	tfAfull

0x36<l>1: Tx Exception Enable (rExEn)

Analog bits (and maximum maxRestart for restart counter)
Default: 0x00ff07f9

0x36<l>2: Tx Control and Activity Status (rCtrl, rActivity)

Default: 0x00000011

bits	mask	description	name (VHDL)
1 RW	0x00000001	Force txLink into offline mode	offline
1 RW	0x00000002	Reset to txLink logic (pulsed)	rst
1 RW	0x00000004	Done	
1 RW	0x00000008	reserved (for Dirty txRst)	✘ VHDL 2do
1 RW	0x00000010	Force rxLink into offline mode	rxOff
1 RW	0x00000020	Reset to rxLink logic (pulsed)	rxRst
1 RW	0x00000040	Done	
1 RW	0x00000080	reserved (for Dirty rxRst)	✘ VHDL 2do
1 RW	0x00000100	Reset to rxClk DCM (persistent)	rxDcmRst
1 RW	0x00000200	reserved (for Done)	✘ VHDL 2do
1 RW	0x00000400	Locked	dcmLock
1 RW	0x00001000	Reset to PHY (persistent)	phyRst
1 RW	0x00002000	reserved (for Done)	✘ VHDL 2do
1 RO	0x00100000	txLink FSM not idle	
1 RO	0x00200000	txFifo not empty	tfEmpty
1 RO	0x00400000	txBuffer not empty	tbEmptyS
1 RO	0x01000000	rxLink FSM not idle	
1 RO	0x02000000	rxBuffer FSM not idle	
1 RO	0x04000000	Packet buffer not empty (not reliable)	✘ VHDL 2do
1 RO	0x08000000	Descriptor fifo not empty	
1 RO	0x10000000	Credit fifo not empty	

0x36<l>3: MDIO Read Access (rMdioR)

Default: 0x00000000

bits	mask	description	name (VHDL)
16 RW	0x0000ffff	Data	
15 RW	0xfdff0000	Address	
1 RO	0x02000000	Busy	

0x36<l>4: MDIO Write Access (rMdioW)

Default: 0x00000000

bits	mask	description	name (VHDL)
16 RW	0x0000ffff	Data	
15 RW	0xfdff0000	Address	
1 RO	0x02000000	Busy	

0x36<l>5: Tx NACK Counter (cntNack)

Default: 0x00000000

bits	mask	description	name (VHDL)
32 RW	0xffffffff	Total count of received NACK	cntNack

0x36<l>6:Tx Resend Counter (cntResend)

Default: 0x00000000

bits	mask	description	name (VHDL)
32 RW	0xffffffff	Total count of entries into resend mode	cntResend

0x36<l>f:TNW revision (VERSION)

bits	mask	description
4 RO	0x0000000f	Time index (if several revisions on same day)
8 RO	0x00000ff0	Day of month
4 RO	0x0000f000	Month (hexadecimal)
8 RO	0x00ff0000	Minor revision number
4 RO	0x0f000000	Major revision number
4 RO	0xf0000000	Type of TNW

The value of this register is specified in the file tnwConstPack.VHD.

9.4 DCR Registers in TNW_RX*

0x3<l>00: Rx Exception Status (rExc,rErr)

Default: 0x00000003

bits	mask	description	name (VHDL)
1 RW	0x00000001	RX_FAULT_P from PHY*)	RX_EXC_FAULTP
1 RW	0x00000002	RX_FAULT_R from PHY*)	RX_EXC_FAULTR
1 RW	0x00000004	INTB from PHY (free running))	RX_EXC_INT
		(cleared by reading MDIO registers, see 6.13)	
1 RW	0x00000008	Timeout (waiting for data within packet)	RX_EXC_TO
1 RW	0x00000010	Credit fifo overrun (write while full)	RX_EXC_CFF
1 RW	0x00000020	Credit size zero	RX_EXC_CSZ
1 RW	0x00000040	rCba written while used	RX_EXC_CBA
1 RW	0x00000080	rNba written while used	RX_EXC_NBA
1 RW	0x00000100	Data received from link while offline	RX_EXC_OFF
1 RW	0x00000200	Protocol error with OWC (unexpected "done")	RX_EXC_OWCD
1 RW	0x00000400	Protocol error with OWC (unexpected "first")	RX_EXC_OWCF
1 RW	0x00000800	spare	RX_EXC_SPARE
1 RW	0x00001000	Invalid rCba arithmetics (carry into bit 32)	RX_EXC_CADD
5 RW	0x001f0000	Error status of last NACK	
	01	• failed CRC	errorC
	02	• dirty data decoding	errorD
	04	• missed header (pending commit)	errorH
	08	• failed write (data buffer full)	errorW
	10	• timeout	errorT
8 RW	0xff000000	cfNotEmpty frozen by RX_EXC_CBA/NBA	rCfNotEmpty

*) SD_INCL_X in Reg 0xd003 and 0xd005/0xd004 must be disabled if link X = R/P is disabled by SD_INV_X in Reg 0xd003 (otherwise RX_FAULT_X is always asserted).

0x3<l>01: Rx Exception Enable (rExEn)

Analog bits (except error status bits wErr*)
Default: 0x00001fff

0x3<l>02: Rx Command Error Count

Default: 0x00000000

bits	mask	description	name (VHDL)
8 RW	0x000000ff	dirty decoded ACK	cntDirtyA
8 RW	0x0000ff00	dirty decoded NACK	cntDirtyN
8 RW	0x00ff0000	dirty decoded RESTART	cntDirtyR
8 RW	0xff000000	dirty decoded IDLE	cntDirtyI

0x3<l>03: Rx Data Error Count

Default: 0x00000000

bits	mask	description	name (VHDL)
16 RW	0x0000ffff	packets with CRC error	cntErrC
8 RW	0x00ff0000	packets with dirty data (*without* CRC error)	cntErrD
8 RW	0xff000000	packets with timeout	cntErrT

0x3<l>04: Rx Flow Error Count

Default: 0x00000000

bit	mask	description	name (VHDL)
16 RW	0x0000ffff	packets rejected for pending commit	cntErrH
16 RW	0xffff0000	packets rejected for failed data write	cntErrW

0x3<l>05: Rx PHY Error Count

Default: 0x00000000

bit	mask	description	name (VHDL)
16 RW	0x0000ffff	counter of $/E/$ on any lane	cntRxe
16 RW	0xffff0000	counter of RXH on any lane	cntRxh

0x3<l>06: Rx Fifo Status

Default: 0x00000000

bit	mask	description	name (VHDL)
1 RO	0x00000001	Packet buffer not empty	fpFull
1 RO	0x00000002	Packet buffer $\geq$ half filled	fpAempty
1 RO	0x00000004	Packet buffer full	fpEmpty
8 RO	0x0000ff00	Descriptor fifos not empty	dfEmpty
8 RO	0x00ff0000	Credit fifos not empty	cfEmpty
8 RO	0xff000000	Credit fifos full	cfFull

0x3<l>07: Rx Control

Default: 0x00000000

bit	mask	description	name (VHDL)
1 RW	0x00000001	Credit Fifo hold	cfHold
1 RW	0x00000002	Credit Fifo reset	cfRst

0x3<l> 1<c>: Notify Base Addresses

Default: 0x81ffffff

bit	mask	description	name (VHDL)
25 RW	0x01ffffff	Notify base address B_n of channel <c> (LSB must be zero)	rNba
1 RW	0x80000000	Notify base segment S_n of channel <c>	rNba

Note that the registers rNba are only reset by power-on reset (poRst), but not by a reset to rxLink (from control register 0x36<l>2)

0x3<l> 2<c>: Credit Fifos / Notify Bits

Default: empty

bit	mask	description	name (VHDL)
32 WO	0xffffffff	Credit Fifo for channel <c>	cfIn
	0x0000ffff	Credit address C_d, see eq. (2)	cfOut
	0x0fff0000	Credit size (in units of 128B)	cfOut, chCnt
	0xf0000000	Notify index C_n, see eq. (2)	cfOut, chCnt
16 RO	0x0000ffff	State of Notify Handler for channel <c>	chNfy

Note that the credit fifos cfIn can *not* be read via DCR. A read access to the above registers will return the current state of the notify-handler registers chNfy (i.e. the value of the 16 LSB used in the last notify).

0x3<l> 3<c>: Credit Base Addresses

Default: 0x81ffffff

bit	mask	description	name (VHDL)
25 RW	0x01ffffff	Credit base address B_d of channel <c>	rCba
1 RW	0x80000000	Credit base segment S_d of channel <c>	rCba

Note that the registers rCba are only reset by power-on reset (poRst), but not by a reset to rxLink (from control register 0x36<l>2)

9.5 MDIO Registers

- Access to MDIO registers is performed via DCR register operations
- Automatic reading of MDIO is not required
- All MDIO registers have address with MSB 0x0, 0xC, or 0xD. Hence, the address bit corresponding to 0x2 is used to indicate that MDIO operation is still "busy" (alternatively one could use negation of the entire address).

address	busy
0x0XXX	0x2XXX
0xCXXX	0xEXXX
0xDXXX	0xFXXX

MDIO write access:

- DCR write operation to the MDIO-write register (mapped into DCR space).

16 bit	`0x0000ffff`	RW	data
15 bit	`0xdfff0000`	RW	address
1 bit	`0x20000000`	RW	busy

- It is asumed that a check for completion/success it is not necessary.

MDIO read access:

- DCR write operation to MDIO-read register
- DCR read operation(s) to the same MDIO-read register

16 bit	`0x0000ffff`	RO	data
15 bit	`0xdfff0000`	RW	address
1 bit	`0x20000000`	RW	busy

DCR read is assumed to always arrive after completion (some check could be provided by allowing the data field to be RW)

10 VHDL Sources

10.1 Files

- `tnwConstPack.vhd`: constants specific and common TNW
- `tnwRecPack.vhd`: records specific and common to TNW
- `tnw.vhd`: top of TNW module
- `txLink.vhd`: top of txLink
- `txFifo.vhd`: Fifo for txFifo
- `txBuffer.vhd`: BRAM with three pointers for resendBuffer
- `rxLink.vhd`: top of rxLink
- `rxBuffer.vhd`: link buffer and "Kreditanstalt"
- `packetBuffer.vhd`: BRAM and Fifo for data buffer and free pool
- `multiFifo.vhd`: Multiple Fifo used in rxBuffer for channel-wise book-keeping
- `chArbiter.vhd`: Arbiter between channels used in rxBuffer
- `phyInt.vhd`: Passing of signals between clock domains of PHY and rxLink
- `crcPack.vhd`: constants specific for CRC generation
- `crcGen.vhd`: CRC generation

The top-level wrapper `torus.vhd` instantiates

- the TNW module
- optional test-benches plus one additional DCR slave

and provides conversion of signal names and types (of the ports) from IBM/QPACE ("`to`") to our TNW-internal conventions.

10.2 TNW-internal Coding Conventions

- Reset is synchronous and active high
- Outputs of entities are usually registered
- Bits of signals are numbered according to the `downto` scheme
- Names of ports are "entity-centric" (e.g. `xxxIn` denotes the input from `xxx`)

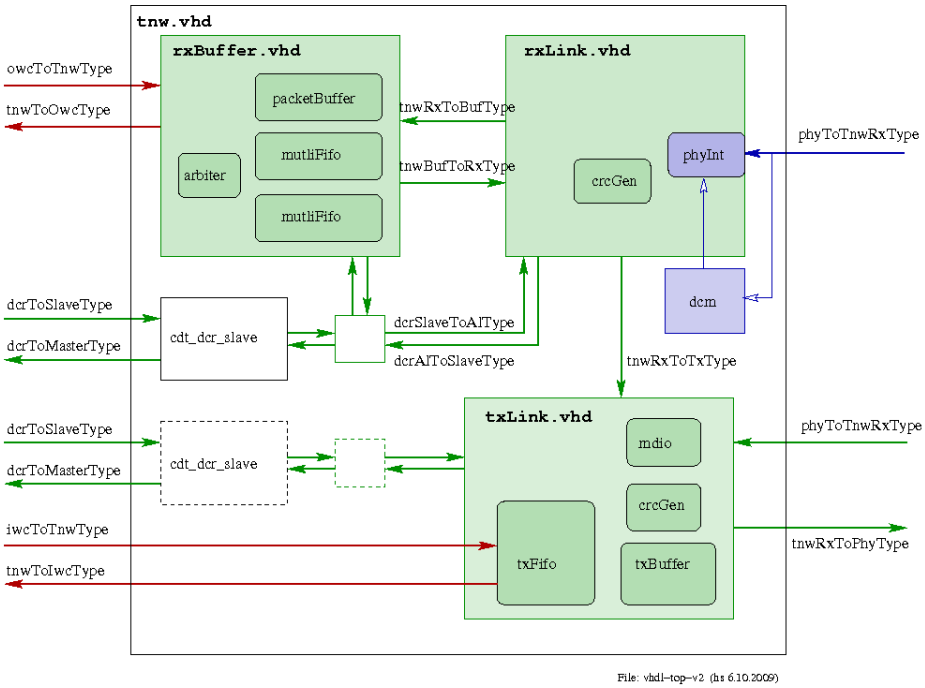
- Names of signals use "camel notation" (e.g. `someFunnySignal`)
- Names of constants are upper-case with underscores (e.g. `SOME_IMPORTANT_CONSTANT`)
- Names of states are upper-case (and short to avoid underscores)
- Names of processes are usually `P<number>`

10.3 Hierarchy of the TNW module

The TNW module `tnw.vhd` instantiates a parametric number of links, each composed of

- txLink
- rxLink
- rxBuffer
- one DCR slave and DCM for the Tx clock domain (used by all txLink modules)
- a DCR slave and DCM for each Rx clock domain (used by rxLink and rxBuffer)

All modules within TNW (except the DCR slaves) strictly use `downto` numbering of bits.



10.4 Clock Domains

- F2 = global FPGA clock 250 MHz
- L2 = 250 MHz from link-specific RX_CLK (generated by DCM)
- R2 = 250 MHz from REF_CLK (generated by DCM)

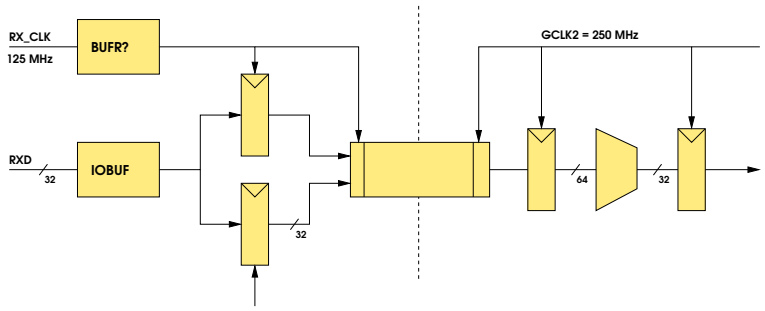
Option	txLink	rxDecoder	rxLink	rxBuffer
0	R2	L2	L2	L2
1	L2	L2	L2	L2
2	R2	L2	R2	R2

Preferable: Option 0 [ts,fs,hs 5.6.2008]

First impllementation: Option 1 (allows single DCR slave for all txLink) [10/2008]

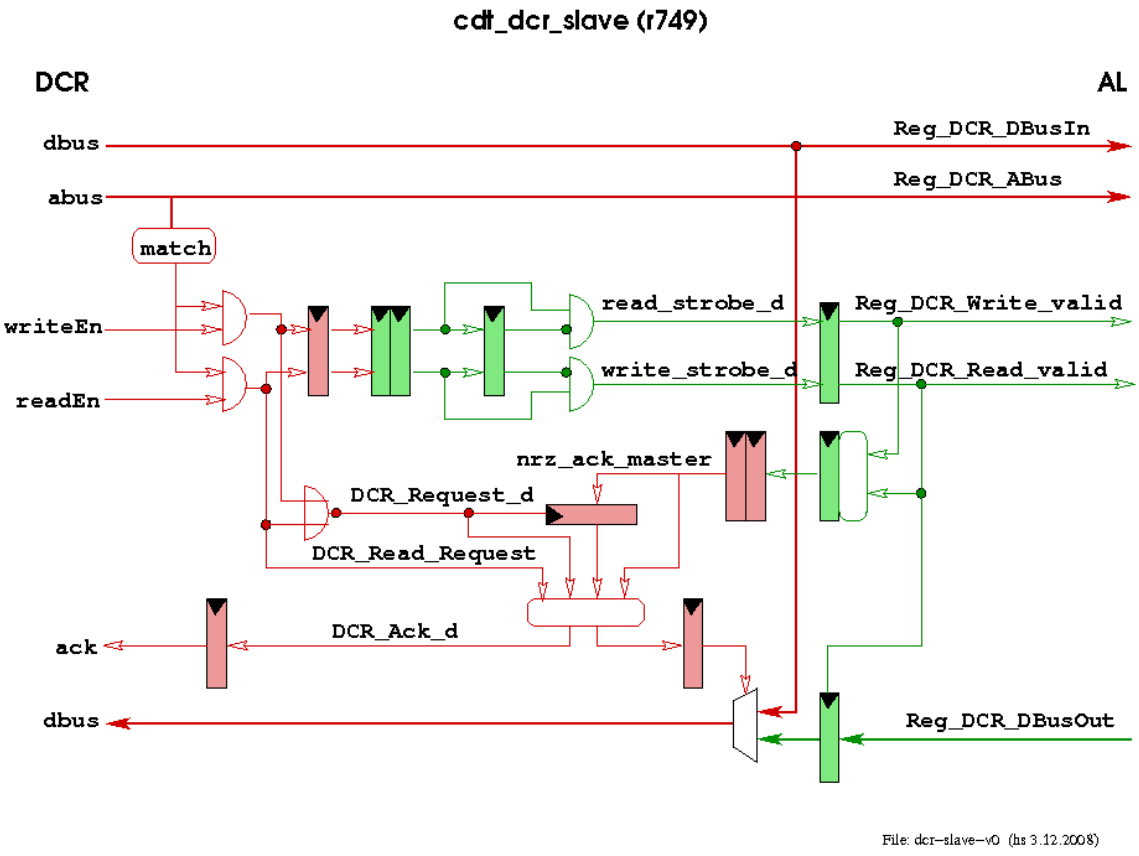
Final implementation: Option 2 [since version 4201 8140]

- The current implementation of phyInt uses an asynchronous Fifo with two input clocks of 250 MHz, derived from REF_CLK and from RX_CLK of the corresponding link, respectively.
- An alternative implementation which directly uses the 125 MHz RX_CLK of the PHY could be implemented, e.g., by the following logic (2 registers clocked on opposite edges plus a Fifo)



N.B.: Number of inserted/deleted /I/ columns can be monitored via registers 0xd$i$06 and 0xd$i$07

10.5 DCR Slave



Conversion from QPACE/IBM to TNW-internal convention (i.e. "`downto`") of the bit numbering in singals between DCR slaves and application logic is performed where the corresponding slave is instantiated.

11 PMC Testbench Fe

11.1 Control Registers

see also /home/fabio/homeraid/vhdl/qpace/tnw3/doc/registers.tx

0x3<8+l>00: Tx Test Control (TXCTRL)

Default: 0x00000000

bits	mask	description	name (VHDL)
1 RW	0x00000001	Data Generation enable	DATAGEN
1 RW	0x00000002	Data Generation mode (0=random,1-seq)	DATAGEN_FLAG
1 RW	0x00000010	IWC enable	IWCSWITCH
1 RW	0x00000100	Insert bit-error on TXD transmission	BACATXD
1 RW	0x00000200	Insert bit-error on TXC transmission	BACATXC
3 RW	0x00070000	Channel address used by TB	CHADDR
5 RW	0x1f000000	One-shot data generation (count and enable)	ONESHOTCNT

0x3<8+l>01: Rx Test Control (RXCTRL)

Default: 0x00000000

bits	mask	description	name (VHDL)
1 RW	0x00000001	Data Check enable	CHKEN
1 RW	0x00000002	Data Check mode	CHKFLAG
1 RW	0x00000010	OWC enable	OWCSWITCH

0x3<8+l>02: Tx Frame Counter (TXFRAMECNT)

Default: 0x00000000

bits	mask	description	name (VHDL)
32 RO	0xffffffff	TX Frame Counter (1 frame = 8 M packets = 1 GB)	TXFRAMECNT

0x3<8+l>03: Rx Error Counter (RXERRCNT)

Default: 0x00000000

bits	mask	description	name (VHDL)
32 RO	0xffffffff	RX Data-Check Error Counter	ERRCNT

N.B.: Values are valid only if `CHKEN` is asserted

`0x3<8+l>04`: Rx Frame Counter (`RXFRAMECNT`)

Default: `0x00000000`

bits	mask	description	name (VHDL)
32 RO	`0xffffffff`	RX Frame Counter	`RXFRAMECNT`

`0x3<8+l>05`: Last Data (`RXLASTDATA`)

Default: `0x00000000`

bits	mask	description	name (VHDL)
32 RO	`0xffffffff`	Last data column checked	`RXLASTDATA`

N.B.: Updated only by valid data (i.e. not free running)

11.2 Test Results and History

12.9.2008 Running loopback without DCR slaves and data checker

- Problem: No feedbacks generated by rxLink
  Fix: added signal `is_fb` in procedure decode_fb

16.9.2008 Running both boards with 2m cable

- Problem: Resend occurs every $O(30)$ sec and always enters deep levels
  Fix: added proper reset of `to_counter`

9.10.2008 Integration of re-written rxLink

31.10.2008 Test of DCR and MDIO access with 6 links on PB in Böblingen

10.11.2008 Rx sampling errors fixed by proper parameters for DCM

14.11.2008 TB running over cable without errors for 12 h

18.11.2008 Added pipeline stage in rxLink

27.11.2008 Test of 6 links in internal loopback on NC in Böblingen

- Problem: `TX_FAULT` on link 2 and 5
  Fix: swap pins `TX_FAULT_i` $\leftrightarrow$ `RX_FAULT_i` of link 2 and 5 in `torus_pins.ucf`
- Problem: CRC errors persistent on link 5
  Fix: swap pins `RXD(30)` $\leftrightarrow$ `RXD(31)` of link 5 in `torus_pins.ucf` (W6 for `RXD(30)` is correct)

6.1.2009 Test of torus3 logics (running PHY with internal loopback)

13.1.2009 Analysis of errors in 32-bit blocks when reading packetBuffer

- No improvement by generating addresses on falling clock edge
- Suppressed by driving read-enable (with strict timing or `sDone`)

22.1.2009 Analysis of DCM reset and OWC requests:

- after PHY-reset (without DCM reset, but no DCM clock input)
  DCM resumes automatically (maybe with asymmetric output signal?)
- after first poRst (which does not assert DCM reset, but PHY-reset)
  DCM resumes automatically
- after subsequent poRst (implies PHY-reset)
  DCM does NOT resume automatically, but only after explict DCM-reset.
- With modified VHDL, s.t. poRst implies DCM-reset (released 2009-01-22),
  DCM resumes automatically after repeated poRst
- No spurious OWC request was seen in any of the above cases

11.3 2do

- Monitor number of inserted/deleted $/I/$ columns via Reg 0xd$i$06 and 0xd$i$07
- Test sequence for keeping clean `RX_FAULT` of un-used port
- Tests with 3.125 GHz

12 Rialto Tests Bb

28.11.2008 Test of bidirectional transfer over 6 links between 4 NC
Problem: Redundant links 2r and 3r don't work (unless in IDLE)
(Tx has FB mismatch, Rx has CRC errors)

1.12.2008 Test of bidirectional transfer over 4 links (0, 3p) between 4 NC
(no NACK on 7 out of 8 directions for 60h)

11.12.2008 torus2 integrated into spine and controlled from C-program on PPE

18.12.2008 Packets injected via IWC and transmitted [Andrea]

19.12.2008 Packets and notify written via OWC [Thomas]

- Sampling of `dataAddress` occurs after $O(5)$ cycles
- Sampling of `notifyAddress` occurs after $O(11)$ cycles

20.1.2009 Latency measurements with torus2 [Simon]

- DCR read: $\approx$ 2930 cycles
- DCR write: $\approx$ 2770 cycles
- Setup of PPU controlled MFC get operations: $\approx$ 800 cycles
- Setup of PPU controlled MFC put operations: $\approx$ 770 cycles

20.1.2009 torus3 (only with 2 red links) integrated into spine

21.1.2009 Problem: BE checkstop occurs during each DCM reset

23.1.2009 Analysis with scope verifies

- explicit DCM reset (`w tx 2 0x100`) generates spurious OWC request
- multiple $1 \rightarrow 0$ transitions of reset signal to torus (from RRAC/GBIF)
thus explaining the need for explicit DCM reset (see tests Fe)

24.1.2009 Test of heavy data transfer (with DMA and Wait) from SPE to txLink

- Problem: wrong data and txLink exception "missing first" (`0x10`)

26.1.2009 DCM reset (`dcmRst`) together with `rxRst` (`w tx 2 0x100`)
does NOT cause BE checkstop

2.2.2009 Test of data transfer SPE-TNW-SPE

13 PMC Testbench Z

13.1 Initial Structure

- RXD sampled with RX_CLK, everything else running at 2×REF_CLK
- Access to MDIO registers
- Verification of synchronisation and alignment
- Manual transfer of sequences
- Automatic generation and check of sequences

13.2 Usage

Thomas!

13.3 Test Results and History

- Patches:
  - disconnect pin 8 of U5 from GND (connect to 1 or leave disconnected) [28.5.2008]
  - disconnect pin 9 of U5 from VDD (has internal pull-down) [4.6.2008]
  - R31, R35, R54, R55 should be put directly on test-points (for good termination) [5.6.2008]

28.5.2008 First testboard available

- Tests:
  - OK: read and write access to MDIO registers
  - `rx_clk0` runs continuously
  - `INTB` is not asserted (remains high)
  - `RXH` is not asserted

29.5.2008 Second testboard available and connected by 0.5 m cable
  - Transfer of random data (4 $\|I\|$ after every 1 KB) without errors over $O(15)$ min
  - Tuning phase of `rx_clk2x` seems not necessary (but can create errors)

30.5.2008 Reached 20h without error

- Investigation of K-symbol handling in trunked mode:
  - $/Q/$ on lane 0 deletes entire column
  - $/T/$ on any lane triggers the `check_end` function, which leads to $/E/$ due to (unexpected) data d_i, e.g. as follows

$$\begin{array}{cccc} d & d & d & d \\ d & /\mathrm{T}/ & d_2 & d_3 \\ d_4 & & & \end{array} \longrightarrow \begin{array}{cccc} d & d & /\mathrm{E}/ & /\mathrm{E}/ \\ /\mathrm{E}/ & /\mathrm{T}/ & d_2 & d_3 \\ d_4 & & & \end{array}$$

    or with multiple $/T/$

$$\begin{array}{cccc} d & d & d & d \\ /\mathrm{T}/ & d & /\mathrm{T}/ & /\mathrm{T}/ \\ d & d & d & d \end{array} \longrightarrow \begin{array}{cccc} d & /\mathrm{E}/ & /\mathrm{E}/ & /\mathrm{E}/ \\ /\mathrm{E}/ & /\mathrm{E}/ & /\mathrm{E}/ & /\mathrm{T}/ \\ d & d & d & d \end{array}$$

  - Less than 4 $/K/$, $/A/$, or $/R/$ in a column are **not** replaced by $/I/$

2.6.2008 Tests in 10b mode (`mod_sel`=111):
  - When inserting $/Q/$ on lane 0, the column is not deleted (in particular, verified for /Q/0/0/0/, /Q/0/0/1/, /Q/0/0/2/). That means that the Q-columns that are deleted when using trunked mode on both sides are deleted by the tx PMC.

- Inserting one invalid 10b code (0x367) gives $/E/$ + `rxh`
- Inserting invalid 10b codes on three successive bytes in the same lane gives three successive $/E/$ (+`rxh`).
- Doing this on four successive bytes in the same lane gives four successive $/E/$ (+rxh), and after that the PMC looses alignment (since he looses byte synchronization) which is signalled by putting $\|LF\|$ (Q/0/0/1) columns on the XGMII output (in accordance with the 10GbE state diagrams)
- MDIO register 0x0018/0xc018 works as expected, i.e. reports the status of byte-synchronization (per lane) and lane alignment (across all four lanes).
- After loosing byte synchronization, synchronization and byte alignment are regained after sending enough $\|K\|$ and $\|A\|$-columns
- Sending the wrong disparity code for a symbol which has two different disparity-neutral encodings (e.g. 0x63), $/E/$ (+`rxh`) is produced. When doing this for four successive bytes on one lane (or other combinations, see FSM in Fig. 48-7 of [3]) alignment is lost (as above).
- Flipping one bit of the 10b code does not necessarily produce an $/E/$ in that column, but may result in an undetected error and produce an /E/ (+rxh) later, as expected.

- Test of "good" packet counters:

  Control by Reg 0xd105, 0xd140, and 0xd000
  Counters in Reg 0xd145, 0xd146, and 0xd147
  Sometimes errors are indicated, but origin is not understood

- Measurement of eye diagrams:

3.6.2008 Loopback latency measurements:

loopback path (see p.28 of [1])	clock cycles 250 MHz (0.5 m cable)
serial	60
system diagnostic	16
parallel diagnostic	54
metallic line	62
parallel line	103
inside FPGA	119

4.6.2008 Switching between primary and redundant link:

`rx_clk0` remains unchanged and in phase with `REF_CLK` during switching

- Tests of clock rate compensation and alignment loss:

  During contiguous $\|I\|$ transfer Reg 0xd206 indicates[8] no deleted and numerous inserted $\|I\|$ ($O(390)/s$, i.e. at rate of 1.5×10^{-6})

  Reg 0xc018 indicates synchronisation (bits 0:3) and alignment (bit 12)

  During contiguous data transfer without $\|I\|$
  - Reg 0xc018 continues to indicate synchronisation but alignment is lost (bit 12 becomes 0)
  - Reg 0xd10c indicates `UNDERRUN`

  When operating in opposite link direction, Reg 0x206 indicates deleted (and no inserted) $\|I\|$ and Reg 0xd10c indicates `OVERRUN` when no $\|I\|$ are transfered

5.6.2008 Stable running at 3.125 GHz (`REF_CLK` = 156.25 MHz)

(jumpers as in Table 2 ??? of [1])

- Verified that all Idle columns (also $\|K\|$, $\|A\|$) can be deleted for clock rate compensation (if at least two consecutive in a row)

6.6.2008 Tests of `RX_CLK` during PMC reset:
  - `RX_CLK` remains low during reset
  - stable clock *period* is reached $O(10)$ μs after reset

- Tests without termination (leaving termination registers at PMC side)

  When DCI is turned off at FPGA, transmission remains stable (tested for $O(1)$ min) Note that DCI must *not* be turned off on the line passing from `RX_CLK` (J21) to regional clock of FPGA, otherwise toggling data on `RXD` causes glitches on `rx_clk`.

  All tests so far have been performed with `PAD_TERMINATE` = 0 (default values of Reg 0xd00c, i.e. SSTL unterminated), which is *not* correct for current setup of testbench (SSTL_2 Class I, i.e. configuration #1 of p. 250 in [1] with external 50 Ω termination to VTT)

- Tests with 3 m cable at 3.125 GHz

  No errors during brief tests $O(1)$ min

[8] This register must be *written* to update it with the values of the internal PHY registers. Then, values remain constant when subsequently (re-)read.

loopback path	clock cycles 312.5 MHz	
(see p.28 of [1])	0.5 m	3 m
serial	60	62
system diagnostic	17	17
parallel diagnostic	54	54
metallic line	62	69
parallel line	102	109
inside FPGA	118	127

A Special Code-Groups for 10GbE

(see Table 48-1 and 48-2 of [3])

Free	XGMII D (C=1)		PCS code-group value	name	Encoding	Code	PCIe	Description
no	0x07	$\rightleftharpoons$	0xbc	K28.5 +)	001111 1010	$/K/$	COM	Idle in $\|I\|$ or $\|T\|$
no	0x07	$\rightleftharpoons$	0x7c	K28.3	001111 0011	$/A/$	IDL	Idle in $\|I\|$
no	0x07	$\rightleftharpoons$	0x1c	K28.0	001111 0100	$/R/$	SKP	Idle in $\|I\|$
yes	0xfb	$\leftrightarrow$	0xfb	K27.7	110110 1000	$/S/$	STP	Start
no	0xfd	$\leftrightarrow$	0xfd	K29.7	101110 1000	$/T/$	END	Terminate
no	0xfe	$\leftrightarrow$	0xfe	K30.7	011110 1000	$/E/$	ENB	Error
no	0x9c	$\leftrightarrow$	0x9c	K28.4	001111 0001	$/Q/$	res1	Sequence
yes	0xf7	$\leftrightarrow$	0xf7	K23.7	111010 1000		PAD	
yes?	0x3c	$\leftrightarrow$	0x3c	K28.1 +)	001111 1001		FTS	
yes	0x5c	$\leftrightarrow$	0x5c	K28.2	001111 0101	$/P/$	SDP	Primitive
yes	0xdc	$\leftrightarrow$	0xdc	K28.6	001111 0110		res2	
no?	0xfc	$\leftrightarrow$	0xfc	K28.7 +)	001111 1000		res3	
no	others	$\rightarrow$	0xfe	K30.7				Invalid

- In the second column, a +) indicates that these codes contain the sequence 0011111 or 1100000.
- PMC uses for Byte alignment K28.5 and K28.1, but not K28.7 (see 10.3.2 [1])
- The first column indicates codes which can be used for our custom protocol
- According to Enternet Standard one can *not* feely use [ts]:
  $/K/$, $/A/$, $/R/$, $/T/$, $/E/$, $/Q/$, and K28.7

In particular:

- $/K/$, $/A/$, $/R/$: obvious
- K27.7 ($/S/$): not used in 10GbE state machines [ts]
- K29.7 ($/T/$): interpreted in `check_end` function
  (see 48.2.6.1.4 and fig. 48-9 of [3], and 14.4.4 of [1])
- K30.7 ($/E/$): why not? (Might be distinguished from error through other indicators?)
- K28.4 ($/Q/$): used to within $\|LF\|$ (loss of alignment) and $\|RF\|$
- K23.7: not used in 10GbE state machines [ts]

- K28.1: why yes? (Used for Byte alignment, but not used once alignment is gained)
- K28.2: not used in 10GbE state machines [ts] (Any special role of $/P/$?)
- K28.6: not used in 10GbE state machines [ts]
- K28.7: why not? (Diagnostics according to 36.2.4.9, but not used by PMC?) May cause code-group realignment according to 36.2.4.9, but according to fig. 48-7, this can not happen unless we lose synchronization [ts].

Ordered sets and special code-groups (see Table 48-4 of [3])

Code	Ordered Set	Encoding
$\|\|K\|\|$	Sync column	all $/K/$
$\|\|A\|\|$	Allign column	all $/A/$
$\|\|R\|\|$	Skip column	all $/R/$
$\|\|S\|\|$	Start column	$/S/$ on lane 0
$\|\|T\|\|$	Terminate column	$/T/$ on any lane followed by $/K/$
$/E/$	Error code group	$/E/$ on any lane
$\|\|Q\|\|$	Sequence ordered set	$/Q/$ on lane 0
$\|\|LF\|\|$	Local Fault signal	/Q/0/0/1/
$\|\|RF\|\|$	Remote Fault signal	/Q/0/0/2/
$\|\|P\|\|$	Signal ordered set	$/P/$ on lane 0

B 10b/8b Coding

1-bit Errors (see Annex 36B of [3]) on XAUI link may lead to RD error detection in

- *same* code group as error
- n-th code group after error (if all $n-1$ following code groups, but not the n-th, have only a single encoding, like D3.1)
- *two subsequent* code groups

C PM8358 Behaviour in 10GE Operation Mode

Lane Synchronisation (Byte Alignment): (see 14.4.1 on p. 240 of [1])

- is gained after 4 $/K/$ symbols
- asserts `SYNC_ERRV` if lost

Lane Alignment: (see 14.4.1 on p. 240 of [1])

- is gained after 4 $/A/$ symbols (with minimal distance of 16, see `A_CNT` in [3])
- asserts `ALGN_ERRV` if lost
- drives $\|LF\|$ on `RXD` if lost

Clock-Rate Compensation: (see 14.4.1 on p. 240 of [1])

Idle ($\|A\|$, $\|K\|$, $\|R\|$) or $\|Q\|$ is deleted when Fifo gets full. In particular,

- Idle can be deleted if preceeded by Idle ⇒ IPG at least 1 Idle
- Idle can be deleted if preceeded by $\|Q\|$ ⇒ IPG at least 1 Idle
- $\|Q\|$ can be deleted if preceeded by identical $\|Q\|$ ⇒ one $\|Q\|$ preserved

$\|R\|$ can be inserted during IPG when Fifo gets empty

D PM8358 Pins

XGMII Interface		
`TX_CLK`	in (SSTL)	
`TXD[31:0]`	in (SSTL)	
`TXC[3:0]`	in (SSTL)	
`TXH[3:0]`	in (SSTL)	tied to 0 for 8b/10b encoding
`TX_FAULT_*` (P,R)	out (async)	see Reg 0x0008/0xc008
`RX_CLK_*` (0...3)	out (SSTL)	
`RXD[31:0]`	out (SSTL)	
`RXC[3:0]`	out (SSTL)	
`RXH[3:0]`	out (SSTL)	code violation or disparity error
`RX_FAULT_*` (P,R)	out (async)	see Reg 0x0008/0xc008

XAUI Interface	
`DL<n><x>_P/N` (0...3; P,R)	in (CML)
`SL<n><x>_P/N` (0...3; P,R)	out (CML)

Clock, Control, and Status[†)]				
REFCLK_P/N	in			
SYSCLK	out		locked to REFCLK	
HC[1:0]	in (pulled-up)	(11)*	Amplitude control	Table 40
EN_PRE_EMPHASIS	in (pulled-up)	1*		0xd021
EN_SLPBK	in (pulled-up)	0	Serial Loopback	0x0000
EN_EQUALIZATION	in (pulled-up)	1*	Equalization of DL*	0xd025
MODE_SEL[[2:0]	in (pulled-up)	(000)*	Mode selection	
RSTB	in (pulled-up)		Reset	
INTB	out (open drain)	pull-up	Interrupt	Table 8
FAILOVER	in (pulled-up)	0	Cross-Bar Control	0xd080
FAILOVER_IMM	in (pulled-up)	0	Cross-Bar Control	Table 7
Management Interface				
MDC	in		< 10 MHz	
PRTAD[4:0]	in (pulled-up)	(00000)*		
MDIO	bi (open drain)	pull-up		
MDIO_SEL	in (pulled-up)	1*	Voltage selection	

[†)] The 3rd column indicates fixed values (a * denotes values which can be changed by jumper settings). The last column refers to further details in [1] or corresponding configuration registers.

E PM8358 Registers

This table shows only the most relevant registers, for the complete list see [1].

primary	redundant	description	page [1]
		DTE XS Control and Status	
0x0000	0xc000	reset, loopback, speed	71
0x0001	0xc001	power, status, FAULT	72
0x0008	0xc008	TX_FAULT, RX_FAULT, loopback	80
0x0018	0xc018	Lane Status (ALIGN, SYNC)	85
		Master Reset and Configuration	
0xd001		configs related to RESET, CLK and MODE_SEL	91
0xd002		configs related to MODE_SEL	96
0xd003		SD, RX_FAULT config, TRANS status	100
0xd004	0xd005	line-wise fault config and monitor: TRANS and SYNC	103
0xd006		status SD_I, TRANS_DET_I, FIFO_ERR_I	111
0xd007	0xd008	line-wise status TRANS_DET_I and TX_DATA_MON	113
0xd009+a		MDIO device ID	117
0xd00b		HALF_RATE, NIBBLE_MODE	119
0xd00c		TRANS_DET_THRES, FIFO_RST, SSTL pad config	120
0xd020 ...0xd05f		Analog transceiver signal config	122
0xd080 ...0xd0af		Cross-Bar config	131
0xd0b0 ...0xd0ff		Clock Synthesis Unit config and status	151

REFX 10 GE Receiver

primary	redundant	description	page [1]
0xd100	0xd200	Global control	155
0xd101	0xd201	Global monitor	157
0xd102	0xd202	PGC thresholds	159
0xd103	0xd203	PGC lengths	160
0xd104	0xd204	8b/10b error thresholds	161
0xd105	0xd205	packet counter control	162
0xd106	0xd206	Trunked inserted column count	163
0xd107	0xd207	Trunked deleted column count	164
0xd108	0xd208	INTB enable 1	165
0xd109	0xd209	INTB enable 2	167
0xd10a	0xd20a	INTB enable 3	168
0xd10b	0xd20b	INTB status 1	169
0xd10c	0xd20c	INTB status 2	171
0xd10d	0xd20d	INTB status 3	172
0xd10e	0xd20e	lane-wise status 1	174
0xd10f	0xd20f	lane-wise status 2	176
0xd110	0xd210	lane-wise status 3	177
0xd111	0xd211	control characters: $/K/$	179
...	...	$/R/$, $/A/$, $/I/$, $/S/$, $/T/$, $/P/$	
0xd118	0xd218	$/CSP/$	
0xd119	0xd219	clock TIP mask	187
0xd140	0xd240	lane 0 control	188
0xd141	0xd241	lane 0 monitor	191
0xd142	0xd242	lane 0 PGC control	193
0xd143	0xd243	lane 0 PGC error count	195
0xd144	0xd244	lane 0 8b/10b error count	196
0xd145	0xd245	lane 0 packet error count	197
0xd146	0xd246	lane 0 total packet count MSW	198
0xd147	0xd247	lane 0 total packet count LSW	199
0xd148	0xd248	lane 0 LF character	200
0xd149	0xd249	lane 0 RF character	201
0xd14a	0xd24a	lane 0 PSC_S character	202
0xd14b	0xd24b	lane 0 PSC_T character	203
...	...		
0xd17b	0xd27b	lane 3 PSC_T character	

TEFX 10 GE Transmitter

primary	redundant	description	page [1]
0xd300	0xd400	Global control	204
0xd301	0xd401	Global diagnostics	207
0xd302	0xd402	Packet counter control	209
0xd303	0xd403	INTB enable	211
0xd304	0xd404	INTB status	212
0xd305	0xd405	INTB status V	213
0xd320	0xd420	lane 0 control	214
0xd321	0xd421	lane 0 control and diagnostics	218
0xd322	0xd422	lane 0 packet count LSW	221
0xd323	0xd423	lane 0 packet count MSW	222
0xd324	0xd424	lane 0 packet error count	223
0xd328	0xd428	lane 1 control	
...	...		
0xd33c	0xd43c	lane 3 packet error count	

References

[1] PMC Sierra, PM8358 datasheet, Document PMC-2020263, Issue 7, Nov 2005 [phy-PM8358-data.pdf]

[2] PMC Sierra, PM8358 errata, Document PMC-2030815, Issue 4, Dec 2006 [phy-PM8358-errata.pdf]

[3] IEEE Std 802.3ae – 2002 (SS94996 published 30 Aug 2002) [802.3ae-2002.pdf]
see http://standards.ieee.org/getieee802/download/802.3-2005_section4.pdf

[4] http://usqcd.jlab.org/usqcd-docs/qmp/QMP-2-0-Introduction.html