QPACE system software: torus

A. Nobile, D. Pleiter

Id: syssw-torus.tex 3739 2010-07-17 17:50:20Z pleiter

Contents

1 Introduction

1.1 Terminology

Packet :	Within the PowerXCell 8i processor and in the torus network data is transported in packets of 128 Bytes.
Message :	Set of data sent from one node to another node. Must be a multiple of 128 Bytes. Note that message should be considered as a user concept as the QPACE torus network only communicates packets.

1.2 Software stack

The torus system software stack consists of the following layers: low-level library `tnw`, mid-level library `torus`. Additionally, high-level layers are planned.

`tnw` library	This library implements a low-level API which comprises a simple software layer to access the torus network with minimal abstraction.
`torus` library	The main functionality added by this mid-level API is the mapping from colour to Cartesian coordinates.

2 Topology

2.1 Definitions/conventions

Absolute coordinates	Absolute coordinates define a node-card position by backplane number $\mathtt{bp} = 0, \ldots, 7, 8, \ldots$ and the node-card's relative position on the backplane $\mathtt{nc} = 0, \ldots, 31$. The following ordering principle applies: $(\mathtt{bp}_0, \mathtt{nc}_0) < (\mathtt{bp}_1, \mathtt{nc}_1)$ if $\mathtt{bp}_0 < \mathtt{bp}_1$ or $\mathtt{bp}_0 == \mathtt{bp}_1 \&\& \mathtt{nc}_0 < \mathtt{nc\_1}$.
Colour coordinates	The 3-dimensional colour coordinate system defines the position of the node-card within the partition. Directions are defined according to the hardware links. By convention blue is the slowest and red the fastest running index.[1]
Partition	A partition is defined by providing the absolute coordinates of the node with the smallest absolute coordinate $(\mathtt{bp}_0, \mathtt{nc}_0)$, which we define to be the partition origin, and the absolute coordinate of the node in the opposite corner $(\mathtt{bp}_1, \mathtt{nc}_1)$ such that $\mathtt{bp}_0$ $(\mathtt{bp}_1)$ and $\mathtt{nc}_0$ $(\mathtt{nc}_1)$.
Blue cabling	In a case where the number of nodes in blue direction $N_b > 1$ additionally the blue cabling has to be defined:

Cabling	Description
`r1`	Single rack configuration
`r2`	Double rack configuration
`r4`	Quad rack configuration

It is assumed that cabling $\mathtt{r}i$ $(i = 1, 2, 4)$ is such that in rack k with $k \bmod i = 0$ and $k \bmod i = i - 1$ the backplanes in the front and back side of the racks are connected.

Partitioning rule: The following partitioning rules apply:

1. The partition corners are defined such that $\mathtt{bp}_0 \leq \mathtt{bp}_1$ and $\mathtt{nc}_0 \leq \mathtt{nc}_1$.
2. It is assumed that nodes are always allocated such that in any colour direction the torus is closed. Special cases:

Trivial closure :	Extension in one or more directions may be 1.
False two :	Extension is 2 without both nodes being connected by 2 links.

Mapping: The following rules apply when mapping absolute and colour coordinates:

Partition origin :	By convention the node-card with the smallest absolute coordinates is taken as origin with colour coordinates $(0, 0, 0)$.
Red direction :	In red direction the nodes are by conventioned arranged from left to right (machine front/back view), i.e. if `nc-<bp>-<nc>` $= (b, g, 0)$ then `nc-<bp>-<nc+i>` $= (b, g, i)$ with $i = 0, \ldots, N_r - 1$ and $N_r = 1, 2, 4, 8$.
Green direction :	In green direction the nodes are by conventioned arranged clockwise (machine front/back view). An example is shown in Fig. 2.1.
Blue direction :	In blue direction the nodes are by conventioned arranged clockwise (machine top view).

[1] E.g., a partition comprising one fully populated backplane has size $1 \times 4 \times 8$.

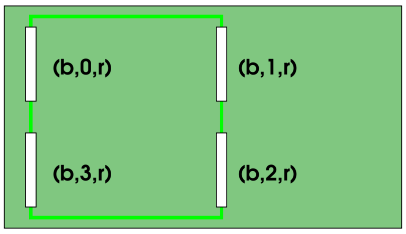

Figure 2.1: Colour coordinates of partition with $N_g = 4$.

Example: A partition extending from `nc-17-04` to `nc-21-13` has size $(2, 2, 2)$ and the node's absolute coordinates are mapped to colour coordinates as follows:

Absolute coordinate	Colour coordinate
`nc-17-04`	`(0,0,0)`
`nc-17-05`	`(0,0,1)`
`nc-17-12`	`(0,1,0)`
`nc-17-13`	`(0,1,1)`
`nc-21-04`	`(1,0,0)`
`nc-21-05`	`(1,0,1)`
`nc-21-12`	`(1,1,0)`
`nc-21-13`	`(1,1,1)`

For more examples see B.

2.2 Information

Low-level API At the level of the low-level API the following information is provided:

- Absolute position of the partition in terms of the absolute coordinates of nodes at two adjecent corners of the partition plus a blue cabling flag.
- Absolute node-card position.[2]
- Relative node-card position of the node-card within this partition in colour coordinates.
- Map of colour direction as well as PHY number and interface configuration[3]

[2] Information is derived from MAC address.

[3] Information is derived from knowledge of the absolute position of the partition and the node-card and the blue cabling flag.

3 Communication attributes

3.1 Addressing

The address where a packet will be written on the receiver side is given by the sum of the following addresses:

Name	Size	Alignment	Description
Base address	32 bit	128 Byte	Address of Local Store or start address of receive buffer in main memory.
Local offset	23 bit	128 Byte	Local offset provided with credit by receiver.
Remote offset	18 bit	128 Byte	Remote offset defined by sender. If a message of size $n \cdot 128$ Bytes with $n > 1$ is sent then the hardware automatically increments the remote offset for each packet.

4 Low-level API: tnw

4.1 Reference Guide: User functions

4.1.1 Initialisation

`tnw_init` Initialise the `tnw` library. Note that on PPU function must be called before starting any SPU threads.

PPU version:

`int tnw_init()`

SPU version:

`int tnw_init(id)`	
`unsigned long long id`	Thread identifier

`tnw_init_rx` Initialize receive handler.

`void tnw_init_rx(rx, phy, channel)`	
`tnw_rx_t *rx`	Pointer to receive handler
`unsigned int phy`	PHY index
`unsigned int channel`	Virtual channel

`tnw_finalize` Clean-up routine.

Provided on SPU only.

`int tnw_finalize`

4.1.2 Buffer management

`tnw_credit_base` Set base address of receive buffer. If receive buffer is in main memory then `addr` refers to page address. If receive buffer is in local store then `addr` is an offset with respect to address 0x0 of the local store. `addr` must be 128-byte aligned.

`void tnw_credit_base(phy, channel, addr)`	
`unsigned int phy`	PHY index
`unsigned int channel`	Virtual channel
`unsigned int addr`	Page address or offset

4.1.3 Communication operations

`tnw_credit` Provide credit to NWP.

`void tnw_credit(rx, addr, size, ctag)`	
`tnw_rx_t* rx`	Pointer to receive handler
`unsigned int addr`	Receive buffer address relative to base address
`unsigned int size`	Size of credit
`unsigned int msgtag`	Message tag

`tnw_test_notify` Non-blocking check whether notification has arrived.

`void tnw_test_notify(rx, ctag)`	
`tnw_rx_t* rx`	Pointer to receive handler
`unsigned int msgtag`	Message tag

`tnw_wait_notify` Blocking check whether notification has arrived.

void tnw_wait_recv(rx, ctag)	
tnw_rx_t* rx	Pointer to receive handler
unsigned int ctag	Communication tag

`tnw_put` Send message.

Provided on SPU only.

void tnw_put(phy, channel, txbuf, offset, size, dmatag)	
unsigned int phy	PHY/link number
unsigned int channel	Virtual channel ID
void* txbuf	Pointer to TX buffer
unsigned int offset	Remote offset
unsigned int size	Message size
unsigned int dmatag	DMA tag

`tnw_wait_put` Function only returns if no DMA operation with given DMA tag has not yet completed.

SPU only.

void tnw_wait_put(dmatag)	
unsigned int	dmatag

`tnw_test_put` Check whether DMA operation to move data to TX link completed.

unsigned int tnw_test_put(dmatag)	
unsigned int dmatag	DMA tag

4.1.4 Memory management

`tnw_rx_alloc` Allocate RX handler.

tnw_rx_t* tnw_rx_alloc()

`tnw_rx_free` Free RX handler previously allocated using `tnw_rx_alloc()`.

void tnw_rx_free(rx)	
tnw_rx_t* rx	RX handler

`tnw_lock_page` Pin given page.

Available on PPU only.

void tnw_lock_page(addr)	
unsigned int addr	Page address

`tnw_unlock_page` Remove lock for given page.

Available on PPU only.

void tnw_unlock_page(addr)	
unsigned int addr	Page address

4.1.5 Topology

`tnw_get_topology` Initialises content of struct instantiated by user with topology information.

Available on PPU only.

void tnw_get_topology(tnw_topo_t* topo)	
tnw_topo_t *topo	Reference to struct containing topology information

`tnw_get_nc_pos` Returns information on absolute node-card position.

Available on PPU only.

`tnw_nc_pos_t tnw_get_apos()`

`tnw_get_spe_phys_id` Return physical ID of SPU.

Available on SPU only.

`unsigned int tnw_get_spe_phys_id()`

4.2 Reference Guide: Data structures in user functions

4.2.1 `tnw_topo_t`

The data structure containing the topology information is defined as follows:

```
struct {
  tnw_nc_pos_t apos0;     /* partition root (abs. coordinates) */
  tnw_nc_pos_t apos1;     /* opposite partition corner */

  unsigned int bc;        /* blue cabling flag */

  tnw_nc_pos_t apos;      /* node-card position (abs. position) */

  int r_size;             /* partition size (col. coordinates) */
  int g_size;
  int b_size;

  int r_coord;            /* node-card position (col. coordinates) */
  int g_coord;
  int b_coord;

  int rp_lnk;             /* index link in plus red direction */
  int rm_lnk;             /* index link in minus red direction */

  tnw_if_mode_t rp_ifm;   /* redundant/primary interface */
  tnw_if_mode_t rm_ifm;

  int gp_lnk;             /* index link in plus green direction */
  int gm_lnk;             /* index link in minus green direction */

  tnw_if_mode_t gp_ifm;   /* redundant/primary interface */
  tnw_if_mode_t gm_ifm;

  int bp_lnk;             /* index link in plus blue direction */
  int bm_lnk;             /* index link in minus blue direction */

  tnw_if_mode_t bp_ifm;   /* redundant/primary interface */
  tnw_if_mode_t bm_ifm;
}
```

4.3 Reference Guide: System/internal functions

The following functions are used within the library or by system tools and should not be called by user application programs.

4.3.1 DCR registers interface

`tnw_reg_read` Read from DCR register.

`unsigned int tnw_reg_read(addr)`	
`unsigned int addr`	DCR register address

`tnw_reg_write` Write to DCR register.

`void tnw_reg_write(addr, data)`	
`unsigned int addr`	DCR register address
`unsigned int data`	Value

`tnw_{tx|rx}_read` Read from DCR register of a particular TX or RX link.

`unsigned int tnw_{tx|rx}_read(link, reg)`	
`unsigned int link`	Link index
`unsigned int reg`	DCR register address

`tnw_{tx|rx}_write` Write to DCR register of a particular TX or RX link.

`void tnw_{tx|rx}_write(link, reg, data)`	
`unsigned int link`	Link index
`unsigned int addr`	DCR register address
`unsigned int data`	Value

4.3.2 PHY control and monitoring

`tnw_mdio_read` Read from PHY register via MDIO bus

`unsigned int tnw_mdio_read(phy, reg)`	
`unsigned int phy`	PHY index
`unsigned int reg`	PHY register

`tnw_mdio_write` Write to PHY register via MDIO bus

`void tnw_mdio_write(phy, reg, data)`	
`unsigned int phy`	PHY index
`unsigned int reg`	PHY register
`unsigned int data`	value

`tnw_reset_phy` Perform reset of PHY. Returns ’0’ if DCM locked or ’1’ otherwise.

`void tnw_reset_phy(phy)`	
`unsigned int phy`	PHY index

`tnw_reset_xgmii_fifo` Perform reset of the PHY’s XGMII FIFO.

`unsigned int tnw_reset_xgmii_fifo(phy)`	
`unsigned int phy`	PHY index

`tnw_reset_phy_counter` Reset PHY’s performance counters.

`unsigned int tnw_reset_phy_counter(phy)`	
`unsigned int phy`	PHY index

`tnw_config_if` Configure PHY interface.

`void tnw_config_if(phy, pr)`	
`unsigned int phy`	PHY index
`tnw_if_mode_t pr`	primary/redundant interface selector `IF_PRIMARY` or `IF_REDUNDANT`

`tnw_set_pr` Select primary or secondary PHY interface.

`unsigned int tnw_set_pr(phy, pr)`	
`unsigned int phy`	PHY index
`tnw_if_mode_t pr`	primary/redundant interface selector `IF_PRIMARY` or `IF_REDUNDANT`

`tnw_set_preemphasis` Enable/disable pre-emphasis on high-speed transmit interface.

`unsigned int tnw_set_preemphasis(phy, en)`	
`unsigned int phy`	PHY index
`unsigned int en`	Selector (0=disabled, otherwise enabled)

`tnw_set_swing` Set differential swing on high-speed transmit interface[1]

`unsigned int tnw_set_swing(phy, val)`	
`unsigned int phy`	PHY index
`unsigned int val`	value

`tnw_set_equalization` Update receiver equalization settings for all lanes:[2]

- 00: no equalization
- 01: full equalization
- 11: half equalization

`unsigned int tnw_set_pr(phy, pr)`	
`unsigned int phy`	PHY index
`unsigned int eq`	Equalisation setting

`tnw_set_clkcfg` Set or unset PHY's `RX_CLK_CFG` configuration bit to switch off/on receiver clocks `RX_CLK_[1-3]`

`unsigned int tnw_set_clkcfg(phy, val)`	
`unsigned int phy`	PHY index
`unsigned int val`	0=on, 1=off

4.3.3 Link control and management

`tnw_reset_link` Reset link

`void tnw_reset_link(lnk)`	
`unsigned int lnk`	link index

`tnw_remove_offline` Move link out of offline state

`void tnw_remove_offline(lnk)`	
`unsigned int lnk`	link index

4.3.4 Topology

`tnw_set_topo` Write content of structure containing topology information to kernel module.

`void tnw_set_topo(topo)`	
`tnw_topo_t* topo`	Pointer to topology structure

`tnw_get_topo` Copy topology information stored in kernel module to topology information structure.

`void tnw_get_topo(topo)`	
`tnw_topo_t* topo`	Pointer to topology structure

`tnw_topo_create` Routine calculates colour coordinates from absolute node-card coordinates.

[1] See table 40 and 41 in [2].
[2] See p. 126 in [3]

void tnw_topo_create(apos0, apos1, bc, apos, topo)	
tnw_nc_pos_t apos0	Node-card with smallest absolute coordinates
tnw_nc_pos_t apos1	Node-card with largest absolute coordinates
int bc	Cabling index
tnw_nc_pos_t apos	Current node-cards absolute coordinate
tnw_topo_t* topo	Pointer to topology structure

`get_spu_phys_id` Return physical ID of SPU.

Available on SPU only.

unsigned int get_spu_phys_id(ctx)	
spe_context_ptr_t ctx	Reference to thread context

4.3.5 Other

`tnw_device_init` Internal routine to initialize device access.

Provided on PPU only.

int tnw_device_init(dcr, iwc)	
unsigned char **dcr	Pointer to pointer to DCR memory map
unsigned char **iwc	Pointer to pointer to IWC memory map

4.4 Examples

4.4.1 PPU program

```
                          #include <tnw_ppu.h>

                          int main(int argc, char *argv[])
                          {
                            tnw_topo_t topo;
                            ...

                            /* Initialization of libtnw *BEFORE* starting spu threads */
Library initialisation      tnw_init();

Topology setup              tnw_get_topology(&topo);
                            printf("Absolute NC position bp=%d nc=%d n",
                                   topo.apos.bp, topo.apos.nc);

Program execution           ...
                          }
```

4.4.2 SPU program

```
                  #include <tnw_spu.h>

                  /* must be global unless tnw_alloc_rx() is used */
RX handler        tnw_rx_t rx;

                  /* Must be global to fullfil alignment requirements */
Send buffer       unsigned int txBuf[4096] __attribute__((aligned(128)));

                  /* Must be volatile and global */
Receive buffer    volatile unsigned int rxBuf[4096] __attribute__((aligned(128)));
```

```
                         int main(unsigned long long id, unsigned long long argp,
                                  unsigned long long envp)
                         {
                           unsigned int lnk, ch, msgSize;
                           unsigned int dmaTag, msgTag;

         Initialisation    tnw_init(id);

    DMA tag reservation    dmaTag = mfc_tag_reserve();

    Define base address    tnw_credit_base(lnk, ch, 0);

   Initialize RX handler   tnw_init_rx(&rx, lnk, ch);

                           ...

         Provide credit    tnw_credit(&rx, rxBuf, msgSize, msgTag);

              Send data    tnw_put(lnk, ch, txBuf, 0, msgSize, dmaTag);

  Wait for notification    tnw_wait_notify(&rx, msgTag);
                           ...
                         }
```

4.4.3 Compiling the program

The `tnw` library comprising of libraries and include files is installed in a path defined in the environment variable `$QROOT`. All pathes should be defined relative to this path.

To compile the SPU program use the following compiler options:

- Include path: `-I$(QROOT)/include`
- Linker options: `-L$(QROOT)/lib/spu -ltnw_spu`

To compile the PPU program use the following compiler options:

- Include path: `-I$(QROOT)/include`
- Linker options: `-L$(QROOT)/lib -lqpace -ltnw`

5 Programming Rules

DMA tags	To avoid using the same DMA tags as the library the user must the MFC tag manager functions, e.g. `mfc_tag_reserve()`, to reserve tags (see [1], p. 83).
`tnw` library initialisation	On the PPU the routine for initialising the torus network library must be called before starting the SPU threads.
RX handler	The receive handler must either be defined as global variable or the function `tnw_rx_alloc()` must be used to instantiate the handler (see 4.1). Otherwise correct alignment will not be guaranteed.
TX buffer	The send buffer must be allocated 128 Byte aligned.
RX buffer	The send buffer must be allocated 128 Byte aligned using the attribute `volatile`.
Base address	The base address for a particular link and virtual channel can only be updated by the receiving node if there are no pending credits.
TX size	The maximum size used in a call of `tnw_put()` is restricted by the following rules: • Size must always be $\leq 16\,$kBytes • In case the number of sending devices $n = 1$: To avoid deadlocks the maximum amount of data in flight per link should be $\leq 16\,$kBytes. • If $n > 1$ sending devices (i.e. SPEs) use the same link (but different virtual channels) then the <u>total</u> amount of data in flight should not exceed the following limits: – $\leq 8\,$kBytes per virtual channel; – $\leq 16\,$kBytes per link.
Remote offset	The remote offset provided in the `tnw_put()` command must be in the range from 0 to (256 kBytes – message size).[1]
Credit size	The maximum credit size is 512 kBytes.
Number of credits	The number of unconsumed credits is limited to 64.
Local offset	The local offset provided with the credit must be in the range from 0 to $65535 \cdot 128\,$Bytes, i.e. `0x0` $\leq$ local offset $\leq$ `0x7fff80`.

[1] With other words: The remote offset of the last packet within the message is $\leq$ `0x3ff80`.

Appendix A Torus network initialisation

A.1 Overview

To reset and configure the TNW the following steps are performed:

- PHY reset
- PHY interface configuration
- Link reset and PHY soft reset
- Link off-line removal

Between each of these steps all nodes have to be synchronized.

A.2 PHY reset `rst-phy`

PHY reset is done when loading torus kernel module. The operations are implemented in routine `torus_phy_reset(u32 lnk)` in file `tnw.c`.

The following operations are performed:

- Hard reset of PHY by writing `0x1111` to TX DCR register 2
- DCM reset by writing `0x111` to TX DCR register 2
- Force link into off-line more by writing `0x11` to TX DCR register 2

FIXME: *There is a risk that forcing link into off-line mode is done before DCM has locked.*

After completion of the reset operations the TX DCR register 2 is read up to `TOUT` (currently 400) times to check whether DCM has locked.

The application `torus_ctl` (command “rst-phy”) additionally performs the following operations:

- Write `0x33` to TX DCR register 2, i.e. links are resetted and put into off-line mode.

A.3 PHY interface configuration and XGMII FIFO reset `cfg-ifp/cfg-ifr`

The operations are implemented in routine `tnw_config_if()` in the `tnw` library. The following operations are performed:

- Configure primary or secondary interface using kernel routine
- Set pre-emphasis
- Set swing
- Set receive equalization
- Switch off `RX_CLK_[1-3]` by asserting `RX_CLK_CFG` bit in register `0xd001`
- XGMII FIFO soft reset (see [3], p. 20):
  - Check PLL lock status (`LOCKV` = 1 in register `0xD0B0`)
  - Check TxClk presence (`TX_CLK_MON` = 1 in register `0xD006`)

  - Reset XGMII FIFO:
    * Set `XGMII_FIFO[0..1]_RST` in register `0xD00C`
    * Clear `XGMII_FIFO[0..1]_RST` in register `0xD00C`

A.4 Link reset and PHY counter reset `rst-lnk`

The operations are implemented in routine `torus_link_reset()` in file `tnw.c`. The following operations are performed:

- Read PHY MDIO register `0xD006`
- Depending on primary or secondary interface the registers `0xD10B`, `0xD10C`, `0xD007`, `0xC007` or `0xD20B`, `0xD20C`, `0xD008`, `0xC008` are read
- Links are resetted and fored into off-line mode by writing `0x33` to TX DCR register 2
- The PHY's performance monitor is reset by writing `0x8000` to the register `0xD000`

Typically operation is invoked via application `torus_ctl` (command "rst-lnk") which may perform additional checks (e.g. on link status registers).

A.5 Remove off-line mode

The operations are implemented in routine `torus_cdm_remove_offline()` in file `tnw.c`. The following operations are performed:

- Write `0x0` to TX DCR register 2

Appendix B Partitioning examples

Note that not all of the following examples may be support in a configuration with global signals:

#Nodes	Partition size	Blue cabling	Partition root	Other corner
1	$(1, 1, 1)$	any	`nc-00-00`	`nc-00-00`
2	$(1, 1, 2)$	any	`nc-00-00`	`nc-00-01`
	$(1, 2, 1)$	any	`nc-00-00`	`nc-00-08`
4	$(1, 1, 4)$	any	`nc-00-00`	`nc-00-03`
	$(1, 4, 1)$	any	`nc-00-00`	`nc-00-24`
8	$(1, 1, 8)$	any	`nc-00-00`	`nc-00-07`
	$(2, 2, 2)$	r1/r2/r4	`nc-00-00`	`nc-04-09`
16	$(1, 4, 4)$	any	`nc-00-00`	`nc-00-27`
32	$(1, 4, 8)$	any	`nc-00-00`	`nc-00-31`
64	$(2, 4, 8)$	r1/r2/r4	`nc-00-00`	`nc-04-31`
128	$(4, 4, 8)$	r2	`nc-00-00`	`nc-12-31`
256	$(4, 8, 8)$	r2	`nc-00-00`	`nc-13-31`
512	$(4, 16, 8)$	r2	`nc-00-00`	`nc-15-31`
1024	$(8, 16, 8)$	r2	`nc-00-00`	`nc-15-31`

Appendix C Pending tasks and issues

Description	Status
Specify and implement functions to read-out PHY counters	2do

Bibliography

[1] IBM, “Software Development Kit for Multicore Acceleration. Programming Tutorial,” Version 3.1 release 0.

[2] PMC Sierra, “PM8358 QuadPHY 10GX Data Sheet,” issue No. 7, November 2005.

[3] PMC Sierra, “PM8358 QuadPHY 10GX Errata,” issue No. 4, December 2006.