OpenSPARC™ T2 Behavioral Model Specification

Sun Microsystems, Inc.
www.sun.com

Part No. 820-6778-10
November 2008, Revision A

Submit comments about this document at: `http://www.sun.com/hwdocs/feedback`

Contents

Figures

Tables

Preface

The OpenSPARC T2 Behavioral Model Specification gives information on the NIU SystemC model serving as an interface between the OpenSPARC T2 RTL and the SAM NIU C++ model. The PCI Express (PCIe) interface integrates the functionality of a host to PCI Express bridge onto the OpenSPARC T2 system-on-a-chip.

How This Document Is Organized

Chapter 1 provides information on the NIU SystemC model. The NIU SystemC model serves as an interface between the OpenSPARC T2 RTL and the SAM NIU C++ model.

Chapter 2 provides information on the PCI Express (PCIe) SystemC model. The PCI Express (PCIe) interface model integrates the functionality of a host to PCI Express bridge onto the OpenSPARC T2 system-on-a-chip.

Using UNIX Commands

This document might not contain information about basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. Refer to the following for this information:

- Software documentation that you received with your system
- Solaris™ Operating System documentation, which is at:

  `http://docs.sun.com`

Shell Prompts

Shell	Prompt
C shell	*machine-name*`%`
C shell superuser	*machine-name*`#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell superuser	`#`

Typographic Conventions

Typeface[*]	Meaning	Examples
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`

`AaBbCc123`	What you type, when contrasted with on-screen computer output	`%` **`su`**
		`Password:`

AaBbCc123	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the *User's Guide*. These are called *class* options. You *must* be superuser to do this. To delete a file, type `rm` *filename*.

* The settings on your browser might differ from these settings.

Documentation, Support, and Training

Sun Function	URL
Documentation	`http://www.sun.com/documentation/`
Support	`http://www.sun.com/support/`
Training	`http://www.sun.com/training/`

Related Documentation

The documents listed as online or download are available at:

`http://www.opensparc.net/`

Application	Title	Part Number	Format	Location
Documentation	*OpenSPARC T2 Core Microarchitecture Specification*	820-2545	PDF	Online
Documentation	*OpenSPARC T2 System-On-Chip (SoC) Microarchitecture Specification, Part 1 of 2*	820-2620	PDF	Online
Documentation	*OpenSPARC T2 System-On-Chip (SoC) Microarchitecture Specification, Part 2 of 2*	820-5090	PDF	Online
Documentation	*OpenSPARC T2 Processor Megacell Specification*	820-2728	PDF	Online
Documentation	*OpenSPARC T2 Processor Design and Verification User's Guide*	820-2729	PDF	Online
Documentation	*OpenSPARC T2 Behavioral Model Specification*	820-6778	PDF	Online

Third-Party Web Sites

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

`http://www.sun.com/hwdocs/feedback`

Please include the title and part number of your document with your feedback:

OpenSPARC T2 Behavioral Model Specification,
part number 820-6778-10

CHAPTER 1

OpenSPARC T2 NIU SystemC Model

1.1 Overview

NIU SystemC model serves as an interface between the OpenSPARC T2 RTL and the SAM NIU C++ model. NIU is implemented as a device in SAM model and implemented as a dynamic library, which is loaded into SAM environment dynamically during system simulation. The SystemC model makes use of the same dynamic library to incorporate the NIU functionality into the OpenSPARC T2 RTL simulation. The SAM NIU model is a zero-time model and is not aware of actual interfaces available in OpenSPARC T2.

NIU interfaces with OpenSPARC T2 through two separate interfaces.

- NIU interfaces with NCU to provide access to its configuration registers. NCU communicates with NIU through a 32-bit data bus and some control signals. Four 32-bit data chunks make up the 128-bit UCB (unit control block) packet send between the two units. The UCB packet include 64-bit payload data, 40-bit physical address, 4-bit packet type, cpu id, thread id, buffer size, request size, and byte mask. Please refer to *NCU MAS* document for details of UCB packet format and NIU interface.
- NIU interfaces with SIU to achieve read and write access to the system memory. This interface includes a 128-bit data bus and additional control signals. A 128-bit header is followed by four 128-bit chunks, which makes up 64-bytes of data. The header includes the physical address, a request id, access type, various error, ecc, and parity bits. For details of this interface please refer to *SIU MAS* document.

NIU interfaces with the outside world through two 10-GB XAUI interfaces. The SAM implementation of the NIU model communicates with the outside world at the packet level and include most of the functionality of the MAC. However, physical layer of networking interface is not implemented. In order to be able to utilize SAM model in RTL simulation, the SystemC model includes the functionality required to interface the SAM model to the physical layer. This functionality in SystemC includes 8b-10b coding/decoding and serialization/deserialization of the 10b codes.

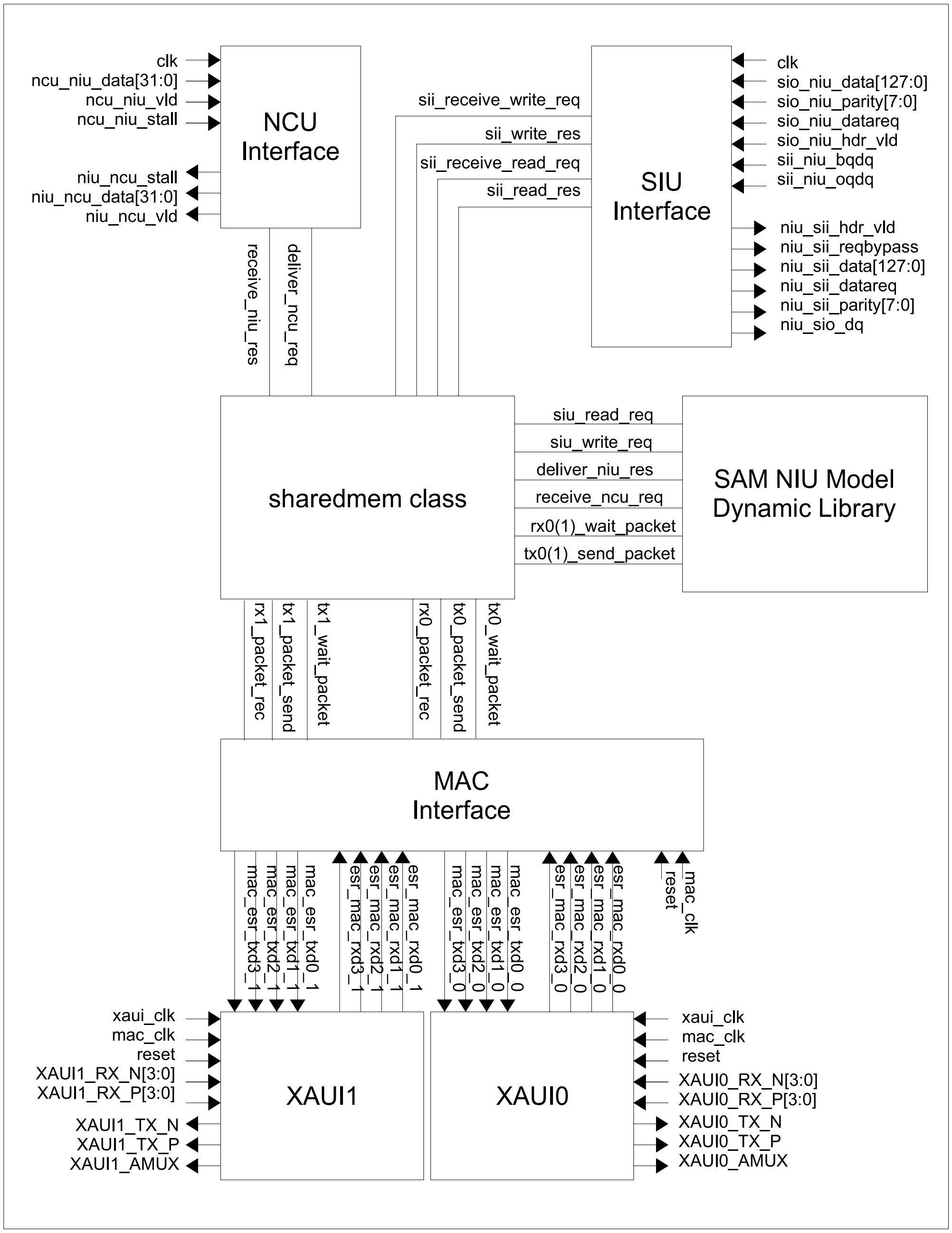
SAM devices, including the NIU device, interact with the system through a predefined interface, which includes a set of function calls. The interface functions are implemented in SystemC and interface with the OpenSPARC T2 RTL model.

FIGURE 1-1 shows the overall architecture of the SystemC model for NIU, the interface signals to the OpenSPARC T2 and the function calls to the SAM NIU model. There are four SystemC models:

- NCU interface
- SIU interface
- MAC interfaces
- XUAI model

Details of these SystemC models and SAM interface functions are provided in the following sections

FIGURE 1-1 Overall Architecture of SystemC Model



1.2 SAM NIU Interface Functions

SAM NIU model interfaces with SAM simulator through a standard interface, which includes a set of predefined function calls.

SAM NIU model implements the following functions to allow SAM to access its functionality. In this case, these functions are implemented to interface with OpenSPARC T2 RTL.

```
int mmi_access (uint32_t cpuid, void* obj, uint64_t paddr,
mmi_bool_t wr, uint32_t size, uint64_t* buf, uint8_t bytemask)
```

Is responsible for IO accessing NIU registers through NCU interface. NCU requests are received by the NCU Interface SystemC module and are passed to the SAM NIU through mmi_access function calls. The output of the SAM model is also send back to NCU through the same SytemC module.

```
void mmi_memread(uint64_t paddr, uint8_t * data, uint64_t size)
```

Is responsible for accessing the system memory for read operations. The system memory is accessed through the SIU interface. This call is implemented through SIU Interface SystemC module.

```
void mmi_memwrite(uint64_t paddr, const uint8_t * data, uint64_t
size)
```

Is responsible for accessing the system memory for write operations. The system memory is accessed through the SIU interface. This call is implemented through SIU Interface SystemC module.

```
int netsim_getmsg (int fd, char * buf, int maxlen, swtchdr * hdr)
```

Responsible for receiving the networks packets that arrive through the XAUI interface. XAUI and MAC Interface SystemC modules are responsible for receiving the packets from the physical layet and passing them to the SAM NIU interface through this function call. The packets received by SAM NIU model are later written into the system memory through SIU interface.

```
int netsim_putmsg (int fd, char * buf, int len, swtchdr * hdr)
```

Responsible for transmitting packets through the XAUI interface. The packets are formed by reading the packet data from the system memory through the SIU interface. The packets formed by SAM NIU model are sent to the physical later through MAC Interface and XAUI Systemc modules.

The SAM NIU model was initially executed as a separate process and was communicating to the RTL simulation through a shared memory implementation. Later, for performance reasons, the SAM NIU model is implemented as a separate thread and the shared memory semantics of the interface kept in place.

1.2.1 Interface to NCU

For OpenSPARC T2, the NCU is the focal point where PIO requests will be dispatched to the NIU and where PIO read returns and interrupts are processed. It serializes the PIOs from different CPU threads to the NIU. It also has an internal table where, based on the system interrupt data, it looks up the CPU thread number and the interrupt number used.

TABLE 1-1 NCU Interface Signals

Signal	Direction	Description
clk	input	clock
ncu_niu_data[31:0]	input	NCU to NIU data bus
ncu_niu_vld	input	NCU to NIU data valid
ncu_niu_stall	input	NCU back pressure control signal to NIU
niu_ncu_stall	output	NIU back pressure control signal to NCU
niu_ncu_data[31:0]	output	NIU to NCU data bus
niu_ncu_vld	output	NIU to NCU data valid

NCU Interface SystemC module interfaces between NCU and the SAM NIU model. The UCB packets received from the NCU are passed to the SAM NIU model. The response received from the NIU model is send back to the NCU. Furthermore, any interrupt that is generated by the SAM NIU model is sent to NCU through UCB packets. Please refer to *NIU MAS* for details of the UCB packet format.

1.2.2 Interface to SIU

SIU Interface SystemC module interfaces between SIU and the SAM NIU model. The interface signals for this module is given in TABLE 1-2.

Memory read requests from SAM NIU model (mmi_memread function calls) are converted into SIU requests. For a read operation only a header is sent to the SIU. A single memory request from the SAM NIU model can generate a number of SIU read requests. Each SIU read request returns 64-byte of data from the system memory.

Therefore, a read request larger than 64-bytes needs to be fragmented and sent to SIU. In addition to fragmentation, SIU interface module also deals with requests that are not aligned to 64-byte boundaries. While all requests from NIU RTL are 64-byte aligned, the SAM NIU model does generate unaligned request. The response from all fragments are collected and sent back to SAM NIU model as a single response. The thread requesting the read from SAM NIU model is blocked until the read request is completed.

Memory write requests from SAM NIU model (mmi_memwrite function calls) are converted into SIU requests as well. For a write operation, a header and four consecutive data cycles are generated. A single write request, similar to the read requests, may end up generating a number of SIU requests if the data size is larger than 64-bytes. The thread requesting the write operation from SAM NIU model is blocked until the write request is completed.

The SIU Interface SystemC module can handle one write and one read operation at a time. All other SAM NIU thread requesting a read or write operation are blocked until the current read/write operation is completed.

TABLE 1-2 SIU Interface Signals

Signal	Direction	Description
clk	input	clock
sio_niu_data[127:0]	input	Contains the header in the header cycle and the payload in the following data cycles. Data is in big endian format.
sio_niu_parity[7:0]	input	Contains the parity for each 16 bit of data Indicates that header has a payload following the header valid cycle, this is also asserted in the header cycle.
sio_niu_datareq	input	Indicates that header has a payload following the header valid cycle, this is also asserted in the header cycle.
sio_niu_hdr_vld	input	Asserted for one cycle (header cycle) to indicate that the header packet is being sent on the data pins
sii_niu_bqdq	input	Dequeue signal for the bypass queue.
sii_niu_oqdq	input	Dequeue signal for the ordered queue.
niu_sii_hdr_vld	output	Asserted for one cycle (header cycle) to indicate that the header packet is being sent on the data pins.

TABLE 1-2 SIU Interface Signals *(Continued)*

Signal	Direction	Description
niu_sii_reqbypass	output	Indicated that NIU is sending packet to SII's 'bypass' Queue. Asserted in the header valid cycle.
niu_sii_data[127:0]	output	Contains the header in the header cycle and the payload in the following data cycles. Data is in big endian format.
niu_sii_datareq	output	Indicates that header has a payload following the header valid cycle, this is also asserted in the header cycle.
niu_sii_parity[7:0]	output	Contains the parity for each 16 bit of data
niu_sio_dq;	output	Dequeue signal for inbound queue; data flow from sio to smx is flow control with 4 credits.

1.2.3 Interface to MAC

While SAM NIU model implements complete NIU functionality, the access to the network does not include physical layer. The SAM NIU model accesses the network through **`netsim_putmsg`** and **`netsim_getmsg`** functions. These two functions simply pass the packet data and the length to the network layer as implemented in SAM simulation environment.

The MAC interface implemented in SystemC model performs necessary coding/decoding work to transfer the packets between the XAUI interface and the SAM NIU model. The interface signals are given in TABLE 1-3. MAC interface performs 8b-10b coding for packets that are transmitted and performs 8b-10b decoding for packets that are received from XAUI interface. A simple first-in first-out queue is implemented on the **`sharedmem`** class to queue the packets received from the XAUI interface until they are processed and written to the system memory by SAM NIU model. If the queue is filled, the MAC interface starts dropping the packets. In the transmit path, there is no queue implementation. For each packet to be transmitted a new thread is created. Each transmit-thread waits until the XAUI interface is available to transmit the packet. Once the packet is transmitted by XAUI interface, the transmit-thread terminates.

The MAC interface uses **`rx0/1_packet_rec`** function to add the packets received to the packet queue. The **`tx0/1_wait_packet`** function is used to check availability of packets from the NIU SAM model. When the packets are transmitted, the completion is signal to the transmit-thread with **`tx0/1_packet_send`** function call.

TABLE 1-3 MAC Interface Signals

Signal	Direction	Description
mac_clk	input	Clock
reset	input	Reset
esr_mac_rxd0_0[9:0]	input	Lane0 10b code word from xaui0
esr_mac_rxd1_0[9:0]	input	Lane1 10b code word from xaui0
esr_mac_rxd2_0[9:0]	input	Lane2 10b code word from xaui0
esr_mac_rxd3_0[9:0]	input	Lane3 10b code word from xaui0
mac_esr_txd0_0[9:0]	output	Lane0 10b code word to xaui0
mac_esr_txd1_0[9:0]	output	Lane1 10b code word to xaui0
mac_esr_txd2_0[9:0]	output	Lane2 10b code word to xaui0
mac_esr_txd3_0[9:0]	output	Lane3 10b code word to xaui0
esr_mac_rxd0_1[9:0]	input	Lane0 10b code word from xaui1
esr_mac_rxd1_1[9:0]	input	Lane1 10b code word from xaui1
esr_mac_rxd2_1[9:0]	input	Lane2 10b code word from xaui1
esr_mac_rxd3_1[9:0]	input	Lane3 10b code word from xaui1
mac_esr_txd0_1[9:0]	output	Lane0 10b code word to xaui1
mac_esr_txd1_1[9:0]	output	Lane1 10b code word to xaui1
mac_esr_txd2_1[9:0]	output	Lane2 10b code word to xaui1
mac_esr_txd3_1[9:0]	output	Lane3 10b code word to xaui1

1.2.4 XAUI SystemC Module

The XAUI SytemC module mimics the SERDES functionality. The module does not perform any clock recovery. It does work on two separate input clocks, **`mac_clk`** and **`xaui_clk`** and **`xaui_clk`** runs 10 times faster than the mac clock. The interface defined by this module works synchronously as opposed to the real asynchronous implementation of SERDES. A brief description of the module input/output signals are provided in TABLE 1-4

The 10b code word received from MAC interface is serialized and put out as **`XAUI_TX_N`** and **`XAUI_TX_P`** signals. The differential signals received from **`XAUI_RX_N`** and **`XAU_RX_P`** are deserialized and sent to the MAC Interface. This module perform comma detection and correctly identifies the 10b code words.

TABLE 1-4 XAUI Signals

Signal	Direction	Description
XAUI_RX_N[3:0]	input	High speed serial signal
XAUI_RX_P[3:0]	input	High speed serial signal
xaui_clk	input	xaui clock; 10x mac_clk frequency
mac_clk	input	mac clock
reset	input	Reset
XAUI_AMUX	output	
XAUI_TX_N[3:0]	output	High speed serial signal
XAUI_TX_P[3:0]	output	High speed serial signal
esr_mac_rxd0[9:0]	output	Lane0 10b code word to mac
esr_mac_rxd1[9:0]	output	Lane1 10b code word to mac
esr_mac_rxd2[9:0]	output	Lane2 10b code word to mac
esr_mac_rxd3[9:0]	output	Lane3 10b code word to mac
mac_esr_txd0[9:0]	input	Lane0 10b code word from mac
mac_esr_txd1[9:0]	input	Lane1 10b code word from mac
mac_esr_txd2[9:0]	input	Lane2 10b code word from mac
mac_esr_txd3[9:0]	input	Lane3 10b code word from mac

CHAPTER 2

PIU Behavioral Model Specification

2.1 PIU Overview

The PCI Express (PCIe) interface integrates the functionality of a host to PCI Express bridge onto the OpenSPARC T2 system-on-a-chip. In PCI Express PCI Express terminology this is called a PCI Express Root Complex. This allows an I/O subsystem based on PCI Express to be connected to an OpenSPARC T2 processor in flexible configurations using standard PCI Express components such as:

- PCI Express to PCI Express bridges and switches
- PCI Express to PCI and/or PCI Express to PCI-X bridges
- Native PCI Express devices

The use of PCI Express allows the use of inexpensive commodity components while supporting excellent bandwidth to high throughput I/O devices. Interoperability with PCI and PCI-X devices is achieved through bridges, and compatibility with existing software is achieved by using standard PCI abstractions. The signalling technology of PCI Express has high bandwidth per pin, giving a low pin count and reduced system cost.

TABLE 2-1 Abbreviation List

Misc.	Core	Sub-core	Block	Sub-block	Description
PCIe					PCI Express
	PEU				PCI Express Core Behavioral Model
			ILU-Intf		Ingress Layer SystemC Bus Interface Model
			CSR		Behavioral Model for CSR

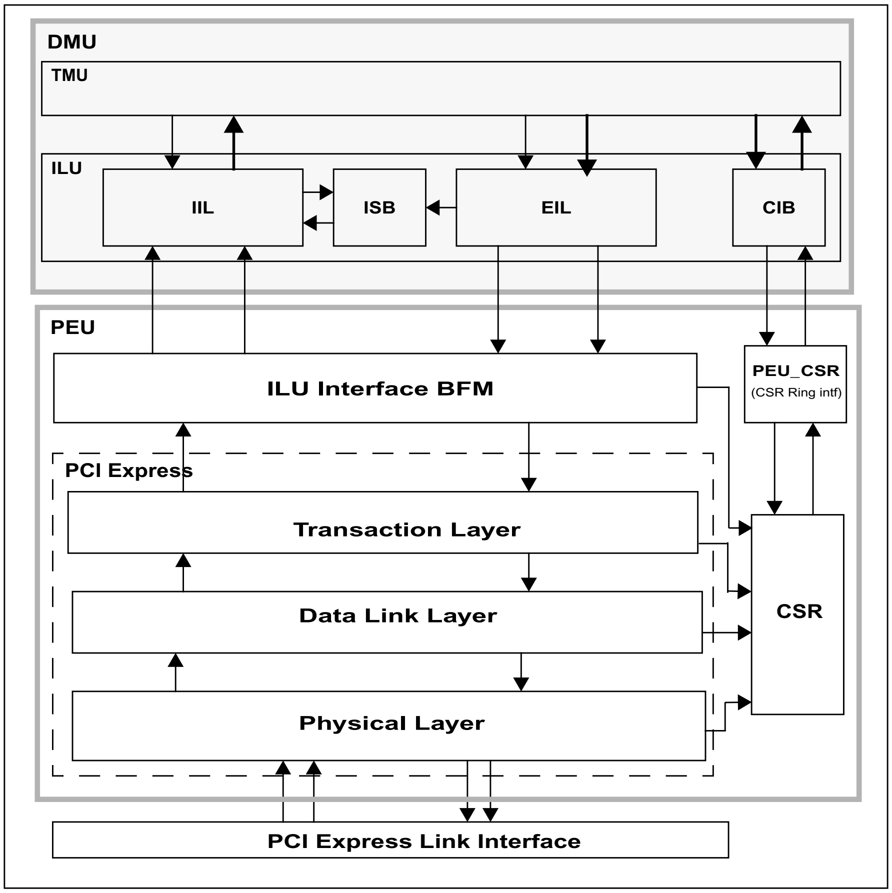
TABLE 2-1 Abbreviation List *(Continued)*

Misc.	Core	Sub-core	Block	Sub-block	Description
			PEU_CSR		CSR Ring Interface layer, behavioral model
		PCIe	TL		Transaction Layer Unit
				ITL	Ingress Transaction Layer
				ETL	Egress Transaction Layer
				RSB	Request Scoreboard
			DLL		DataLink Layer Unit
				PL Consumer	Ingress packet handler
				TL Consumer	Egress packet handler
				PL Producer	Egress traffic producer
				TL Producer	Forwarding ingress traffic
				Retry Buffer	Replay buffer manager
				FC Init	Flow Control Initialization
			PL		Physical Layer Unit
				LTSSM	Link Training and Initialization State Machine
				SDL	Scrambler/Descrambler logic
				EDL	Encode Decode function
				SKEW	Skew generator
				SERDES	Functional Serializer and Deserializer
	DMU				Data Manager Core
			TMU		Transaction Manager Unit
				DIM	Data Ingress Manager
				DEM	Data Egress Manager
			ILU		Transaction Manager Unit
				IIL	Ingress Interface Layer
				EIL	Egress Interface Layer
				ISB	Interface Score Board
				CIB	CSR Interface Block

This document describes the functional specifications and features of the PEU (PCI Express Unit) behavioral model for the OpenSPARC T2.

2.2 PIU Block Diagram

FIGURE 2-1 PIU Block Diagram



2.3 OpenSPARC T2 PEU Root Complex Feature Summary

TABLE 2-2 PEU Root Complex Summary

Feature	SystemC Model Supported
PCI Express Specification Compliance	1.0a+
IO Virtualization IOMMU (handles using granularity by bus id); Number of PCI Express functions supported	No Yes 1
Link at 2.5Gb/sec: 2.0GByte/sec per direction Link Width; Max_Payload_Size [Device Capability -> Device Control]	 x8 max; 512B

Flow control Credit:
Posted Header;
Posted Data;
Non-Posted Header;
Non-Posted Data;
Completion Header
Completion Data
	(Default Values)
	32 credits;
	192 credits;
	16 credits;
	infinite (not used as RC)
	infinite (16 credits);
	infinite (64 credits)

Power Management:
Power States;
Link States;
 	
Active State Power Management (Receive Support);
PME Message Receive Support;
Set_Slot_Power_Limit Transmit Support;
Wake from PME (Vaux Support);
	
	D0
	L0, L0s
	
	No
	No
	No
	No

PCI Express Configuration Register Support:
PCI Express Root Complex Compliant Configuration Space;
Parameters loadable via SPROM (sw-initiated);
Implemented as part of CSR tool Ring
	Single Function;
	No;
	Yes;
	Yes

Traffic Class / Virtual Channels TC0-TC7 on VC0 supported (Completions returned w/same TC as req); All egress TLPs sent w/TC0	 Yes; Yes
ECRC Generation & Checking Supported	No

TABLE 2-2 PEU Root Complex Summary *(Continued)*

Feature	SystemC Model Supported
Hot Plug Support (instead relies on external Switch)	No
Interrupts: Legacy interrupt support with INTX emulation; Translates MSI/MSI-X interrupts into mondo interrupts	 Yes; Yes
Error Handling: Sun-Extended PCI Express Advanced Error Handling;	 Yes
Ordering Rules: Sun4u/Sun4v compliant; Device Control Reg: Relaxed ordering bit	 Yes; No

2.4 PCI Express (PEU) Behavioral Model

The PCI Express behavioral model is a transaction level, non-synthesizable model. This transaction level model behaves as a Root Complex device on the PCI Express link. It models logical state machine of the Link Initialization and link status state machine. It does not model electrical timings of the actual design.

2.5 PEU Port Interface

The PEU model interfaces with the PCI Express link and models the complete logical behavior of the link interface including the initialization state transitions, framing, serialization and deserialization. The link clock is supplied externally, and is not auto extracted from the link.

The model has bus interface with the ILU/DMU of the OpenSPARC T2. The ILU (Interface Layer Unit) can read and write the ingress/egress transactions from the TL. This interface is modelled into the ILU interface BFM, to connect the behavioral model to the T2 design.

TABLE 2-3 provides the list of signal interfaces for the PEU model.

TABLE 2-3 PEU Interface Signals

Signal	Direction	Description
	PCI Express Link Interface	
link_clk	Input	OpenSPARC T2 Signal
link_in[7:0]	Input	OpenSPARC T2 Signal
link_in_bar[7:0]	Input	OpenSPARC T2 Signal
link_out[7:0]	Output	OpenSPARC T2 Signal
link_out_bar[7:0]	Output	OpenSPARC T2 Signal
	PEU-DMU Interface	
d2p_csr_ack	Input	OpenSPARC T2 Signal
d2p_csr_rcd[95:0]	Input	OpenSPARC T2 Signal
d2p_csr_req	Input	OpenSPARC T2 Signal
d2p_cto_ack	Input	OpenSPARC T2 Signal
d2p_csr_ack	Input	OpenSPARC T2 Signal
d2p_csr_rcd[95:0]	Input	OpenSPARC T2 Signal
d2p_csr_req	Input	OpenSPARC T2 Signal
d2p_cto_ack	Input	OpenSPARC T2 Signal
d2p_ech_wptr[5:0]	Input	OpenSPARC T2 Signal
d2p_edb_addr[7:0]	Input	OpenSPARC T2 Signal
d2p_edb_data[127:0]	Input	OpenSPARC T2 Signal
d2p_edb_dpar[3:0]	Input	OpenSPARC T2 Signal
d2p_edb_we	Input	OpenSPARC T2 Signal
d2p_ehb_addr[5:0]	Input	OpenSPARC T2 Signal
d2p_ehb_data[127:0]	Input	OpenSPARC T2 Signal
d2p_ehb_dpar[3:0]	Input	OpenSPARC T2 Signal
d2p_ehb_we	Input	OpenSPARC T2 Signal
d2p_erh_wptr[5:0]	Input	OpenSPARC T2 Signal
d2p_ibc_nhc[7:0]	Input	OpenSPARC T2 Signal
d2p_ibc_pdc[11:0]	Input	OpenSPARC T2 Signal
d2p_ibc_phc[7:0]	Input	OpenSPARC T2 Signal
d2p_ibc_req	Input	OpenSPARC T2 Signal

TABLE 2-3 PEU Interface Signals *(Continued)*

Signal	Direction	Description
d2p_idb_addr[7:0]	Input	OpenSPARC T2 Signal
d2p_ihb_addr[5:0]	Input	OpenSPARC T2 Signal
d2p_spare[4:0]	Input	OpenSPARC T2 Signal
p2d_ce_int	Output	OpenSPARC T2 Signal
p2d_csr_ack	Output	OpenSPARC T2 Signal
p2d_csr_rcd[95:0]	Output	OpenSPARC T2 Signal
p2d_csr_req	Output	OpenSPARC T2 Signal
p2d_cto_req	Output	OpenSPARC T2 Signal
p2d_cto_tag[4:0]	Output	OpenSPARC T2 Signal
p2d_drain	Output	OpenSPARC T2 Signal
p2d_ecd_rptr[7:0]	Output	OpenSPARC T2 Signal
p2d_ech_rptr[5:0]	Output	OpenSPARC T2 Signal
p2d_erd_rptr[7:0]	Output	OpenSPARC T2 Signal
p2d_erh_rptr[5:0]	Output	OpenSPARC T2 Signal
p2d_ibc_ack	Output	OpenSPARC T2 Signal
p2d_idb_data[127:0]	Output	OpenSPARC T2 Signal
p2d_idb_dpar[3:0]	Output	OpenSPARC T2 Signal
p2d_ihb_data[127:0]	Output	OpenSPARC T2 Signal
p2d_ihb_dpar[3:0]	Output	OpenSPARC T2 Signal
p2d_ihb_wptr[6:0]	Output	OpenSPARC T2 Signal
p2d_mps[2:0]	Output	OpenSPARC T2 Signal (Minimum of all EP functions’ mps)
peu_dmu_epmode	Output	EP mode bit: 1 = in EP mode, 0 = in RC mode
peu_dmu_ro_en	Output	1 = Relaxed ordering enabled
p2d_oe_int	Output	OpenSPARC T2 Signal
p2d_spare[4:0]	Output	OpenSPARC T2 Signal
p2d_ue_int	Output	OpenSPARC T2 Signal

TABLE 2-3 PEU Interface Signals *(Continued)*

Signal	Direction	Description
	Global Signals	
gclk	Input	global cmp clock grid, 1.4GHz
pc_clk	Input	OpenSPARC T2 Signal
rst_wmr_	Input	OpenSPARC T2 Signal
rst_por_	Input	OpenSPARC T2 Signal

2.6 10,000 feet

The PCI Express behavioral model, models a transaction level model for the PCI Express block. It implements the physical layer, data link layer and transaction layer functionality. For the OpenSparc T2, it includes the ILU Bus Interface Model and models the Header and Data Buffer interfaces to the ILU.

Listed here are the main functions of the PEU Behavioral Model at 10,000 feet level.

Physical Layer:

- Link Training and Initialization
- Link Retrain/Reconfig and Drain State
- Packet framing
- 8b/10b Encoding and Decoding
- Packet Scrambling and Descrambling

Data Link Layer:

- Flow Control Initialization
- LCRC calculation
- Ack/Nak TLP packet
- Retransmit packet for reliable communication

Transaction Layer:

- Ingress Transaction Packet Header Parsing and Data Checking
- Egress Transaction Request Scoreboarding
- Drain State handling

CSR:

- CSR Ring Interface

ILU_Intf BFM:

- Model Ingress Header Buffer (IHB) and Ingress Data Buffer (IDB) Data Memory Interface
- Model Egress Header Buffer (EHB) and Egress Data Buffer (EDB) Data Memory Interface

2.7 ILU Interface BFM (ILU_intf)

The ILU Interface BFM responsible for interfacing the SystemC model with the ILU RTL signal level interface.

Main functions of the ILU interface are:

- Forward ingress packet from ITL to the ILU/DMU.
- Capture and forward the complete egress packet to ETL and RSB.
- Emulate the transaction layer data buffers, IHB, IDB, EHB, and EDB.
- Insert and check parity on egress header/data.
- Update interrupt and timeout signals to the ILU

ILU Interface BFM interfaces the DMU/ILU block. The Transaction Layer packets are exchanged over the buffer interface between the ILU and the BFM. The BFM emulates the read/write behavior of four data RAMs:

- Ingress header buffer (IHB)
- Ingress data buffer (IDB)
- Egress header buffer (EHB)
- Egress data buffer (EDB)

2.8 IHB and IDB

IHB and IDB are treated as a single circular buffers.

As a consumer of IHB, ILU would detect IHB's emptiness.

The IHB's emptiness detection is through its read/write pointers. There are 64 entries in IHB. Therefore, it's a six-bit IHB read/write address (d2p_ihb_addr). However, the IHB write pointer passed from the BFM to ILU is seven-bit (p2d_ihb_wptr) with the MSB as a roll-over bit. ILU keeps its own seven-bit IHB read pointer with the MSB as a roll-over bit too. It's empty if the seven-bit read/write pointers are the same.

The IHB's fullness detection is through global PCI Express flow control credit mechanism.

There is no need for IDB emptiness detection because it's guaranteed that the transaction associated payload is ready to pull in IDB when the transaction header is processed down the pipeline.

2.8.1 EHB and EDB

EHB and EDB are treated as two circular buffers (half/half). The low address space (one half) is partitioned for completion (DMA Cpl/CplD) records and their associated payload (named as ECH & ECD buffer); the high address space (the other half) for request (PIOs) records and their associated payload (named as ERH & ERD buffer).

For each header circular buffer, ILU passes its write pointer to the BFM (d2p_ech_wptr for ECH and d2p_erh_wptr for ERH); The BFM passes its read pointer to ILU (p2d_ech_rptr for ECH and p2d_erh_rptr for ERH). The MSB in these read/write pointers is a roll-over bit.

Similarly, for the data circular buffers the BFM passes read pointer to ILU (p2d_ecd_rptr for ECD and p2d_erd_rptr for ERD). The MSB in these read pointers is a roll-over bit.

As a producer of EHB and EDB, ILU detects their fullness for both circular buffers. ILU keeps its own set of write pointers to ERD and ECD with the MSB as a roll-over bit. A circular buffer is full if their roll-over bits in read/write pointers vary and the rest are the same.

As a consumer of EHB and EDB, the BFM detects the emptiness for the two header circular buffers. However, there is no need to detect the emptiness for the two data circular buffers because it's guaranteed that the transaction associated payload is ready to pull in EDB when the transaction header is processed in ETL. A circular buffer is empty if their read/write pointers are the same.

2.8.2 Link and Transaction Status

The interface BFM also updates the interrupts to the DMU/software in case transaction errors, or uncorrectable link errors.

2.9 Transaction Layer Unit (TL)

The TL contains three major blocks:

- Ingress Transaction Layer (ITL)
- Request Scoreboard (RSB)
- Egress Transaction Layer (ETL)

2.9.1 Ingress Transaction Layer (ITL) Functionality

Ingress Transaction Layer parses the ingress transaction header to identify any error in the transaction, and would flag any unsupported and malformed transactions.

As the functionality of the root complex (RC) would identify following type of header types as unsupported requests.

1. Types (unsupported requests)
   a. Memory Read Request - Locked
   b. I/O Read Request
   c. I/O Write Request
   d. Configuration Reads
   e. Configuration Writes
   f. Message Requests with data payload
2. Message Codes (unsupported requests)
   a. PM_Active_State_Nak
   b. PME_Turn_Off
   c. Unlock

   d. Vendor_Defined Type 0
   e. All Hot Plug signaling messages
3. Malformed packet checks
   a. Reserved types
   b. Crossing 4 KB boundary
   c. Max payload size exceeded
   d. Any message which is not traffic class zero

2.9.2 Request Scoreboard (RSB) Processing

Request scoreboard validates all the PIO completion packets. It would identify the following six categories of the PIO completion errors:

1. Unsolicited Completion Error (as long as one of the following conditions meets)
   a. "tlp_tag" in completion header (Cpl/CplD) doesn't match any outstanding PIO request's "tlp_tag"
   b. "req_id" in completion header doesn't match its corresponding request's "req_id". Since OpenSPARC T2 PEU is a root complex, the "req_id" is 16'b0 for all the PIO requests. Thus, it's an unsolicited completion error as long as "req_id" in completion is not 16'b0
   c. It's a CplLk type
   d. It's a CplDLk type
2. Malformed Completion Error (as long as one of the following conditions meets)
   a. CplD with unsuccessful status
   b. Completion status is "configuration retry" for a non-configuration PIO request
   c. CplD associates with a PIO write request
   d. Successful Cpl associates with a PIO read request
   e. Any mismatches in the following fields between a completion and its corresponding PIO request:
      i. TC (OpenSPARC T2 PEU sets it to 3'b0 in the requests)
      ii. Attr (OpenSPARC T2 PEU sets it to 2'b0 in the requests)
      iii. Length (only check for CplD)

iv. Byte Count (it should be 12'h4 in Cpl/CplD resulted from PIO io/cfg rd/wr requests)

v. Lower Address (it should be 7'b0 in Cpl/CplD resulted from PIO io/cfg rd/wr requests)

3. Configuration Retry Error - completion status in Cpl header is "configuration retry" for a configuration PIO request
4. Unsuccessful Read Error - completion resulted from PIO read request, whose status in Cpl header is not "successful completion"
5. Unsuccessful Write Error - completion resulted from PIO write request, whose status in Cpl header is not "successful completion"
6. Time Out Error - no response within a programmed amount of time

2.9.3 Egress Transaction Layer (ETL) Header and Data Control

Transaction layer accepts the packet coming from the ILU and the request scoreboard and forwards them to the Data Link Layer for transmission if required credits are available.

All the posted transactions are posted with the request scoreboard for the completion checks. The functionality of request scoreboard, and associated errors are explained above.

2.9.4 Flow Control

Release records are used to flow control the issuing of transactions between the TMU and ILU. A Core will not issue transaction records if it has no credit to do so.

There are two credit bases which control transactions between the TMU and ILU, one for ingress and one for egress, which track requests and associated completions.

The TMU owns the credit base for TLP requests it issues to the ILU (egress). The ILU owns the credit base for TLP requests it issues to the TMU (ingress). The ILU will accept up to sixteen TLP non-posted read/write requests without flow controlling the TMU/ILU interface.

2.10 Control and Status Register (CSR) and CSR Ring Interface

The PEU CSR are implemented as part of the SystemC model, using STL map for CSR address and record. All subblocks in the model can connect directly to the CSR over the CSR interface port, and read/write to the CSR.

All the external access to the CSR over the CSR ring protocol are gated to the CSR through the peu_csr module. The peu_csr connects to rest of the design over the CSR ring interface.

Also for the test env, to have direct (signal cross referenced) interface to the CSR is provided through the csr omni interface. Omni interface provides the updated snapshot of the CSR data values to the Verilog signals, and vice versa.

2.11 Data Link Layer

2.11.1 Functionality

Data link layer is responsible for providing the necessary communication channels between TL and PL.

The major responsibilities of Data Link Layer (DLL) include:

- Send/Receive FC initialization/update packets
- Append Sequence numbers to the TL packets
- Perform LCRC and CRC checks on packets
- Send ACK/NACK for DLLPs received from PL
- Replay packets which were NAKed

2.11.2 Communication Channels

DLL contains the five TLM channels which interact with Transaction (TL) and Physical (PL) layers. There are two TLM channels which send and receive packets to/from TL. There are three TLM channels between DLL and PL. One of these channels receive packets from PL, the other two are used to send packets to PL for DLLP and TLP packets separately. This is done to ensure that no starvation of each type of packet will occur at the PL level.

2.11.3 DLL Architecture

There are four separate producer/consumer threads which interact with TL and PL:

- pl_consumer: Upon receiving a STP packet, lcrc and sequence numbers are checked and ACK/NAK is sent to PL accordingly. If SDP packets are received, flow control registers are updated provided packet has the correct CRC.
- pl_producer: This thread retrieves packets from TLP and DLLP queues and transfers the packet to the PL layer.
- tl_consumer: After receiving the TLP from TL, sequence number and LCRC are appended to the packet and put into the TLP queue.
- tl_producer: Packets from Tl queue are retrieved and sent to TL by this thread.

Additionally architecture is added to:

- calculate_lcrc: Calculates LCRC
- dll_ctrl_mgmt: Implements the state machine for DL state machine during link training
- fc_init: Sends and receives FC init packets during initialization
- replay_buffer: Implements the replay buffer which stores transactions to be replayed

2.12 Physical Layer

As discussed in the previous sections, the physical layer is the third among the three layers in the PCIe fabric. The Physical Layer gets the data from and sends it to the upper two layers (Data Link Layer and Transaction Layer).

In the SystemC architecture, the Physical Layer has been implemented in both SystemC and Verilog. It has been designed to follow the PCIe specification and performs the following functionality.

1. Performs link initialization and training
2. Performs 8b/10b decoding and encoding
3. Framing/Deframing of the encoded data

4. Clock based frame boundary calculation

5. Disparity Checks

In addition, the PL talks to a very basic SERDES interface. The basic unit of transaction is the packet. The Data Link Layer puts both DLL and TL packets in a queue. The enqueuing of these packets into the DLL queue and the TL queue are asynchronous. The PL then takes the necessary amount of time to send the contents of the entire packet (DLL or TL packet) over the link. Similarly, the PL gathers packets over the link and forwards them to the DLL.

2.12.1 Link Initialization and Training

Link Initialization is done by the PL. This is done by a module called LTSSM (Link Training and Status State Machine). The following states have been implemented

1) DETECT_QUIET

2) DETECT_ACTIVE

3) POLLING_ACTIVE

4) POLLING_CONFIG

5) CFG_LINKWIDTH_START (config linkwidth start)

6) CFG_LINKWIDTH_ACCEPT (config linkwidth accept)

7) CFG_LANENUM_WAIT (config lanenum wait)

8) CFG_LANENUM_ACCEPT (config lanenum accept)

9) CFG_IDLE

10) L0

11) Disabled Entry

12) Disabled Idle

13) Disabled

14) Recovery Recover Lock

15) Recovery Recover Config

16) Recovery Idle

17) Hot Reset

18) L0s

Transitions among all of the states above are supported. The power management states L1, and L2 are currently not supported. The current implementation also accounts for a single channel (single link). Multiple lanes in a link are supported. LINK_WIDTH of 2, 4, 8, 16, and 32 are supported. The default LINK_WIDTH is 8. No electrical properties are checked. There is a default timeout value for the state machine to transition from the DETECT_QUIET to the DETECT_ACTIVE and from the DETECT_ACTIVE to the POLLING_ACTIVE states.

The LTSSM module does not perform actual clock extraction during link initialization. Instead, it samples initial data at both the positive edge and negative edge to understand the rate of change of data. Once it understands how the data changes, it locks onto either the positive or the negative edge of the clock and forms the frame boundary accordingly.

There are two separate engines running in the module, the receiver engine and the transmitter engine. Accordingly, there are two frame boundaries, the receiver frame boundary and the transmitter frame boundary. The link training state machine is dependent on both the receiver as well as the transmitter engines. Since the model is for a Root Complex, the LTSSM starts off with sending the first initialization data.

2.12.2 Scrambling and Descrambling

Scrambling and Descrambling is done on eight bits of data. The scrambler/descrambler module uses an LFSR logic to do this. The LFSR is set to an initial value and then with every advancing clock tick, it applies an XOR logic to the sampled bits with the current LFSR value and then shifts the LFSR value.

Special control (K) characters are never scrambled. The COM (comma) character resets the LFSR to the initial value. Intermittent SKP (skip) symbols are used to align the LFSRs of both the root complex and the end point.

Scrambling and Descrambling starts only after the PCI Express device reaches the CFG_IDLE state during initialization.

2.12.3 Decoding and Encoding

Decoding and Encoding is performed, based on a map table. There are four Hash Map tables. Two for special character encoding and two for data character encoding. Each type requires two encodings because the PCI Express specification allows for disparities in the link. Each 8b symbol has a (+) positive encoding and a (-) negative encoding. While getting data from the upper layers, it first scrambles the data and then encodes it. While receiving data from the link, it first decodes data and then descrambles before passing it on to the upper layers. Decoding and encoding is done continously.

2.12.4 Framing and Deframing

The PL is responsible for framing data while putting it onto the link and then deframing the data it receives from the link.

Framing is also responsible for sending data on multiple links at the same time. This takes advantage of multiple lanes in PCI Express. Typically, a frame consists of LINK_WIDTH number of columns and ten rows. Each symbol is transmitted or received on one column. Thus, it takes ten clock cycles to receive LINK_WIDTH number of symbols. The higher the number of lanes supported, the more the number of symbols transmitted or received in one frame boundary. Typically, every frame boundary is ten clock cycles.

2.12.5 Clock Based Frame Boundary Calculation

The concept of framing and deframing is dependent on the calculation of the frame boundary. This is calculated in the LTSSM module based on the arrival of the first COM (K28.5) character.

A counter is utilized to assert the frame boundary signal after every ten clock cycles. It is triggered by the arrival of the first COM character.

There are two frame boundaries, the transmitter and the receiver. The transmitter frame boundary is based on what is being transmitted by the LTSSM. The receiver frame boundary is calculated based on what the LTSSM receives from the link.

The frame boundary is continuously calculated based on the clock and the COM character. If the transmitter changes from driving at one edge to another, the frame boundary changes too. This makes sure that the symbols are aligned right after coming out of a reset sequence.

2.12.6 Disparity Checks

One characteristic of the PCI Express link is that it transmits equal number of 1s and 0s over a period of time. This keeps the potential of the link neutral. From the functional point of view, this gives rise to the two separate encodings for a 8b symbol. Different lanes can maintain different disparities. The (+) positive encoding is for a lane where the previous transaction for that lane had more 1s than 0s. The (-) negative encoding for a lane is where the previous transaction for that lane had more 0s than 1s. Some of the lanes can have neutral disparities due to having equal number of 0s and 1s. For those lanes, the existing disparity for that lane is retained during encoding.

There is a disparity checker in both the LTSSM and the PL modules to insure that the symbols received on the ingress side are encoded correctly and follow the current disparity for that lane.

2.12.7 Packetization

The basic unit of a transaction in the model is a packet. The DLL produces a packet and puts it into a queue for the PL to consume. The PL similarly produces a packet and puts it into the queue for the DLL to consume.

The transformation from a frame to a packet and vice versa is done in the PL Top module. A packet can be thought of as the logical equivalent of a frame. When the PL receives raw data from the link, it first decodes, descrambles and frames the data. It then strips each lane in the frame to form the packet. A packet consists of sequential symbols. These symbols could be special symbols or data symbols. For example, a DLL packet starts with a SDP symbol and ends with either the END or EDB symbol. Similarly, a TL packet starts with the STP symbol.

2.12.8 SERDES

The physical layer implements a very basic SERDES interface. Based on a frame boundary, it serializes and deserializes the incoming data. This frame boundary is supplied by the LTSSM module. On the egress side, the SERDES takes in a frame and then serializes it. Typically, it takes ten clock cycles for the SERDES to put one frame onto the link. Each serialized data is a concatenation of all the bits of the lanes. Thus, the size of a serialized data is LINK_WIDTH bits.

For example, if the LINK_WIDTH is 8, one frame of serialized data = {lane7[0], lane6[0], lane5[0], lane4[0], lane3[0], lane2[0], lane1[0], lane0[0]}. The next frame will contain bit one of all these lanes, the next will contain bit two and so on.

The SERDES does a reverse transformation while deserializing the incoming data.