FTF | FREESCALE TECHNOLOGY FORUM 2015

Utilizing PCI Express in QorIQ LS Series Processors

FTF-SNT-F1121

Richard Nie | SMTS, Systems and Application Engineer

JUNE.2015

freescale™

External Use

Session Introduction

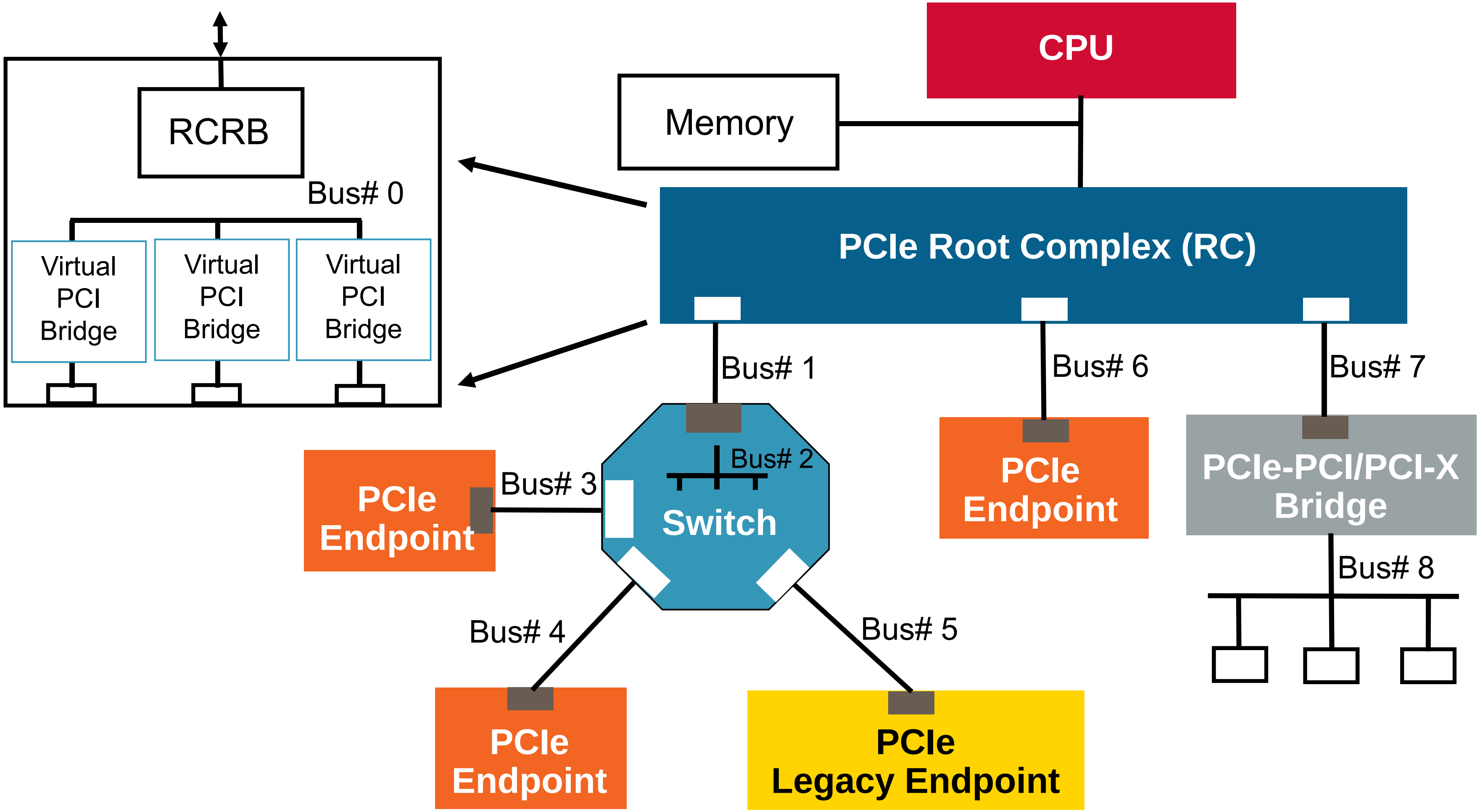
- Session Length: 2 hours
- This session will introduce the features and programming model of the new PCI Express controllers integrated in the Freescale QorIQ LS series processors. It will start with some key feature highlights, then dive into how to initialize the controller with the new programming model, establish the PCI Express link, and master outbound configuration and memory transactions to the downstream EPs.

Agenda

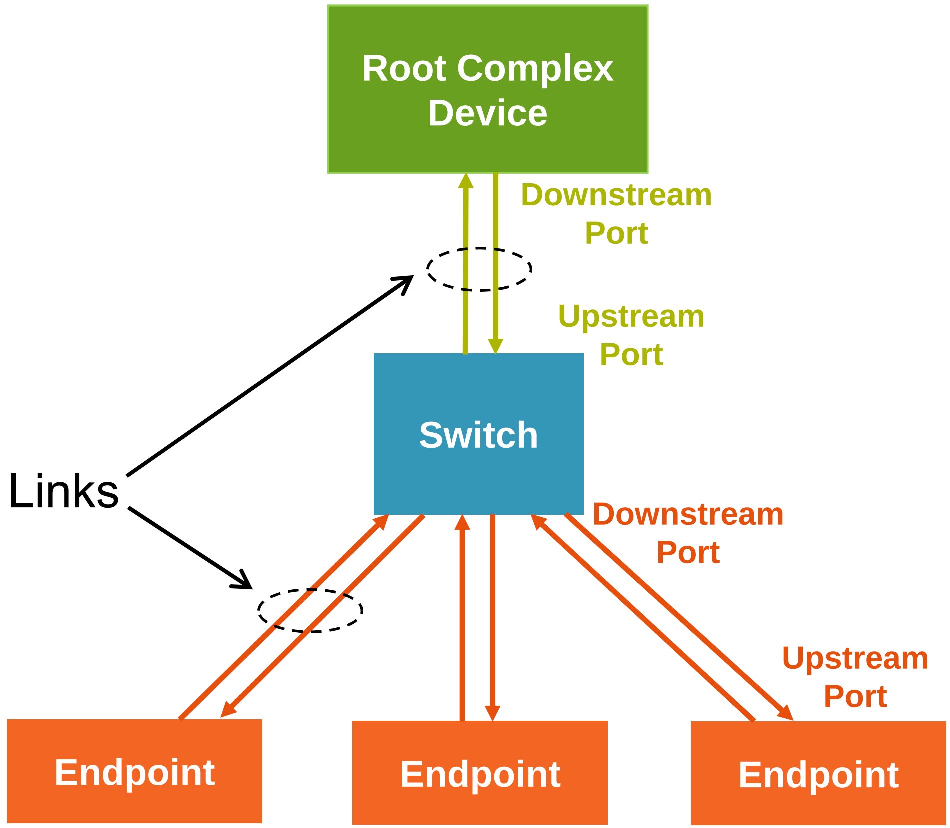
- PCI Express Architecture and Protocol Brief Overview
- LS2085A PCI Express Controller
  - Design Consideration Factors
  - RCW Configuration
  - Architecture Overview
  - Initialization and iATU Programming
- Summary
- Reference

PCI Express Architecture & Protocol Brief Overview

PCI Express Architecture Overview

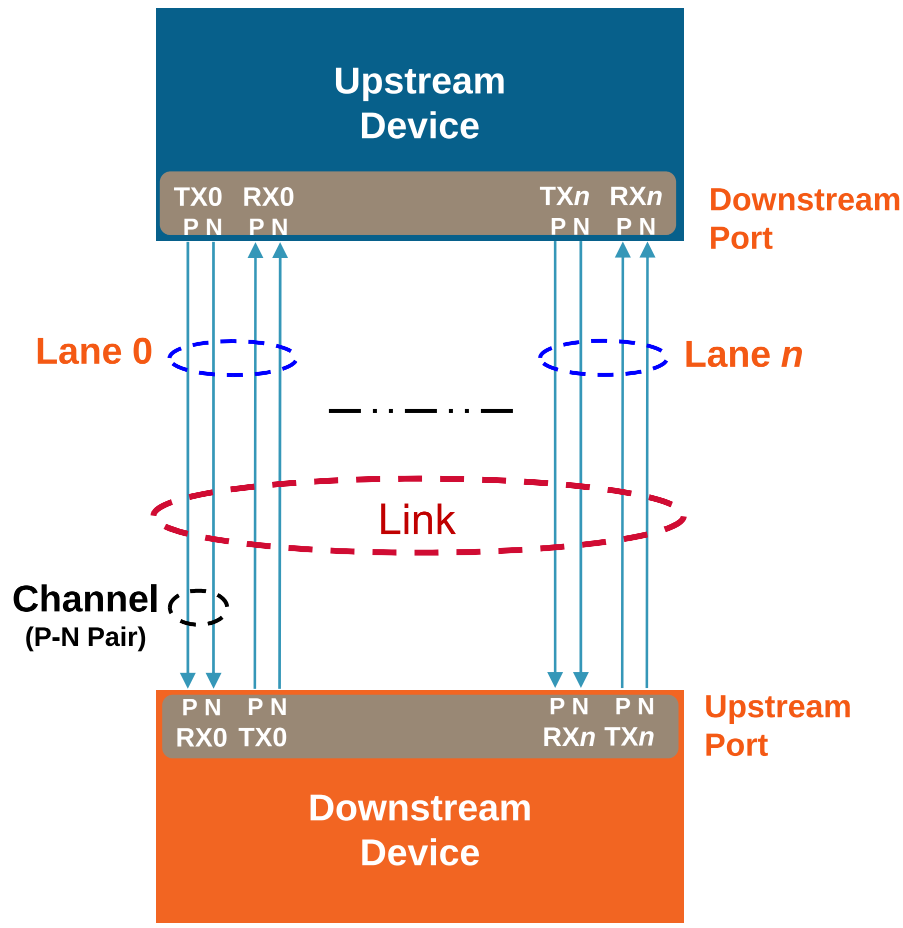


PCI Express Topology Related Terminology

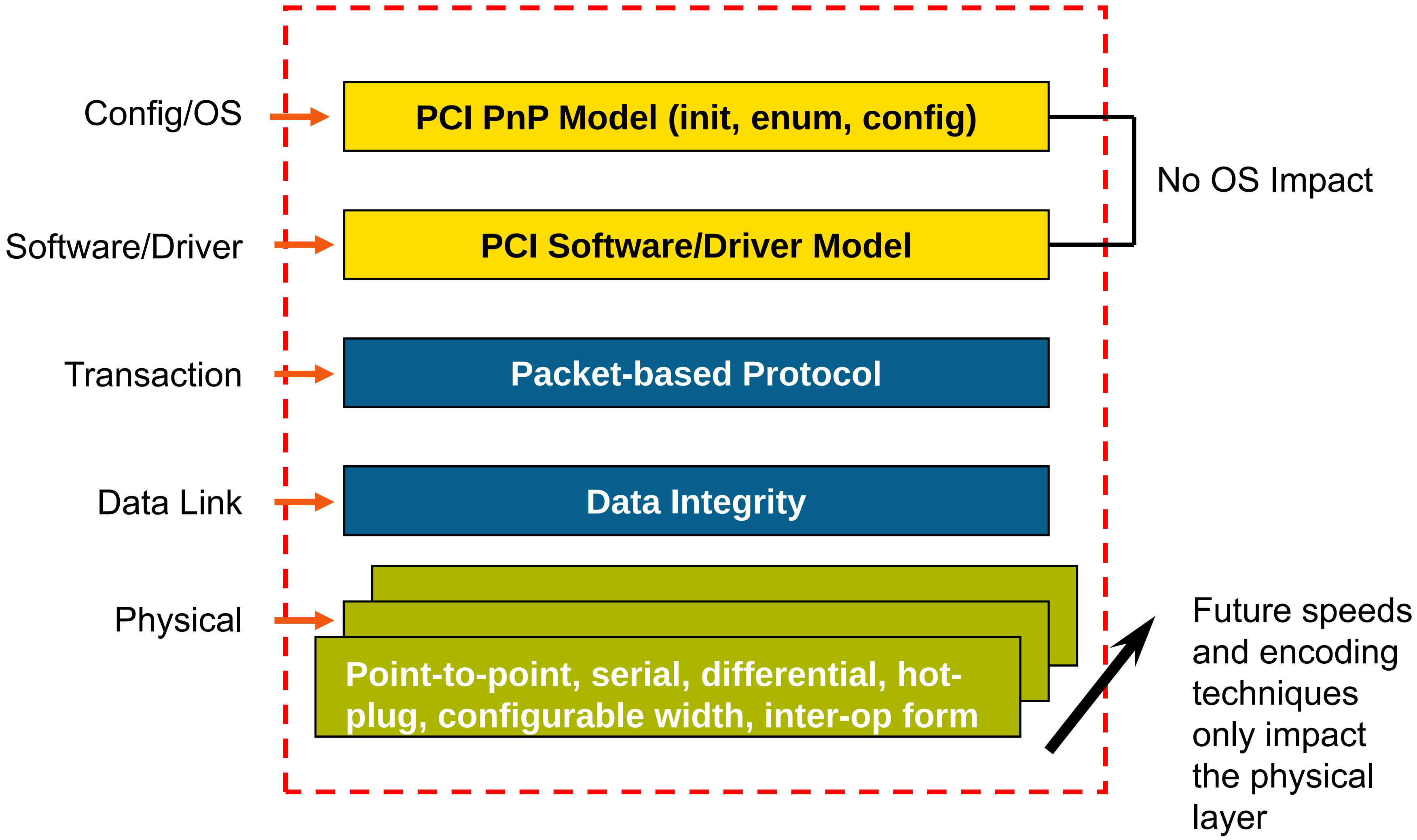


PCI Express Physical Layer Terminology

- Link
  - Collection of two ports and their interconnecting ***lanes***
- Lane
  - A set of differential signal pairs: one pair for Tx and another for Rx.
- Port
  - Physically, ***a group of transmitters and receivers*** located on the ***same chip*** that ***define a link***
  - Logically, an interface between a component and a PCI Express Link
- x1, x2, x4, x8, x16, ….. xN (Link)
  - ***A by-N link is composed of N lanes***



PCI Express – Layered Architecture (Software and Hardware)

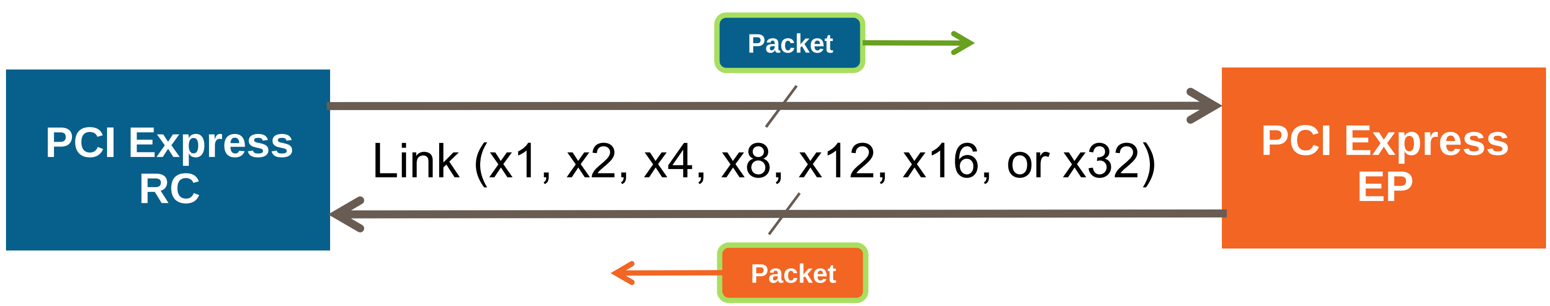


PCI Express Transaction Bandwidth

- Point-to-point connection
- Serial bus significantly reduces number of pins → requires only 4 pins for a x1 link
- Scalable: x1, x2, x4, x8, x16, x32
- Dual-simplex connection
- 2.5 GT/s per direction (***Gen 1***)
- Packet-based transaction protocol

Link Width	**x1**	**x2**	**x4**	**x8**	**x12**	**x16**	**x32**
RAW Bandwidth ***per direction***, Tx or Rx (Gbits/sec)	2.5	5	10	20	30	40	80
Effective Bandwidth ***per direction***, Tx or Rx (Gbits/sec)	2	4	8	16	24	32	64

***Note: Raw** bandwidth reflects the maximum theoretical physical link speed with 8b/10b encoding*



More About PCI Express Throughput

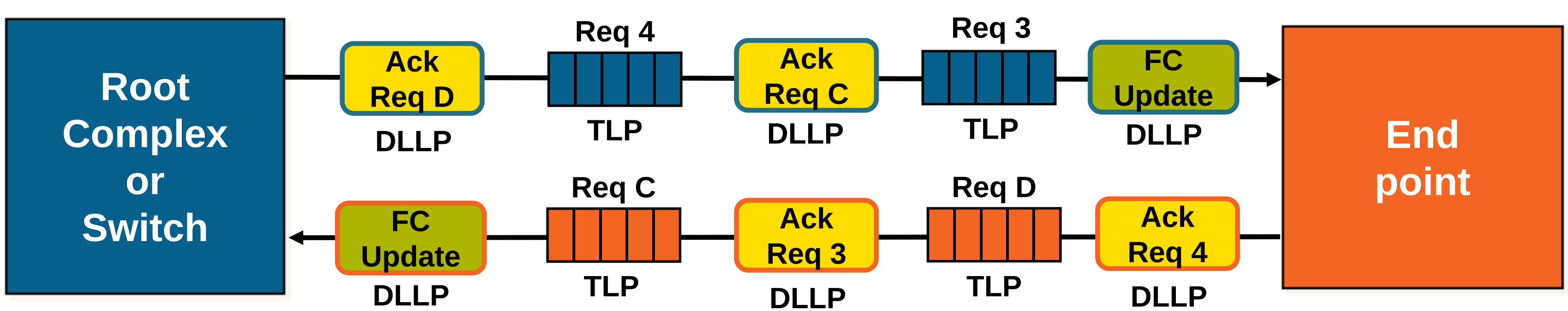
Effective Bandwidth (*per direction*)	Link Width						
	x1	**x2**	**x4**	**x8**	**x12**	**x16**	**x32**
PCIe ***Gen 1*** Bandwidth (Gbits/sec)	2	4	8	16	24	32	64
PCIe ***Gen 2*** Bandwidth (Gbits/sec)	4	8	16	32	48	64	128
PCIe ***Gen 3*** Bandwidth (Gbits/sec)	8	16	32	64	96	128	256

Note:

1) *These are the **Effective** bandwidth **per direction** excluding the 8b/10b encoding for Gen1 and Gen2, and 128b/130b encoding for Gen3*
2) *Gen1 x1 link raw data rate = 2.5 GT/s*
3) *Gen2 x1 link raw data rate = 5.0 GT/s*
4) *Gen3 x1 link raw data rate = 8.0 GT/s*

PCI Express Transaction Layer Highlights

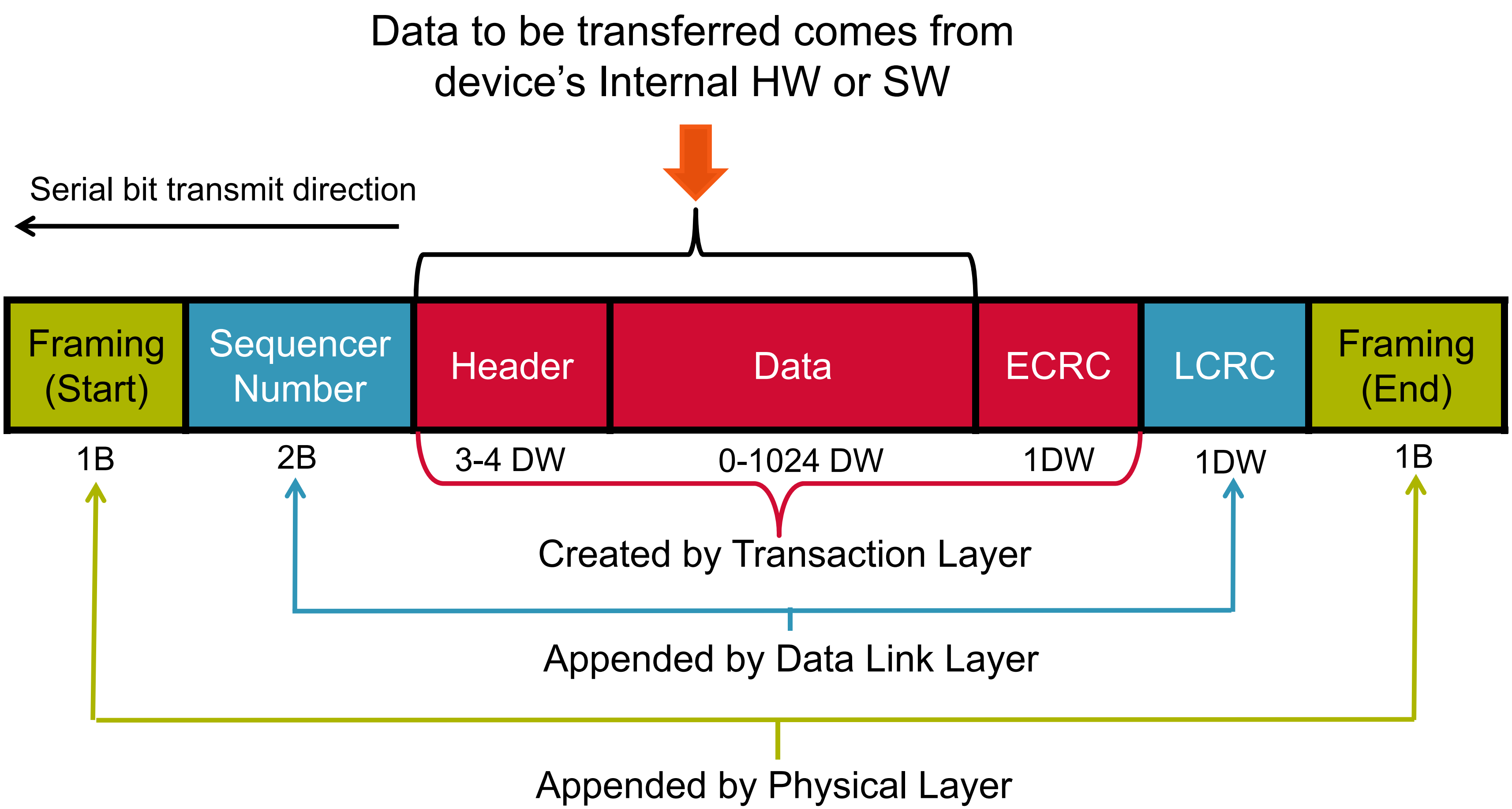
- Packet-based split-transaction protocol
- Provides R/W logical transactions to software
- 4 basic transaction types: memory, I/O, configuration and message
- 32-bit and 64-bit memory addressing
- Three routing methods
  - Address routing (memory and I/O)
  - ID routing (configuration)
  - Implicit routing (messages)
- Transactions are carried by Transaction Layer Packets (TLPs)



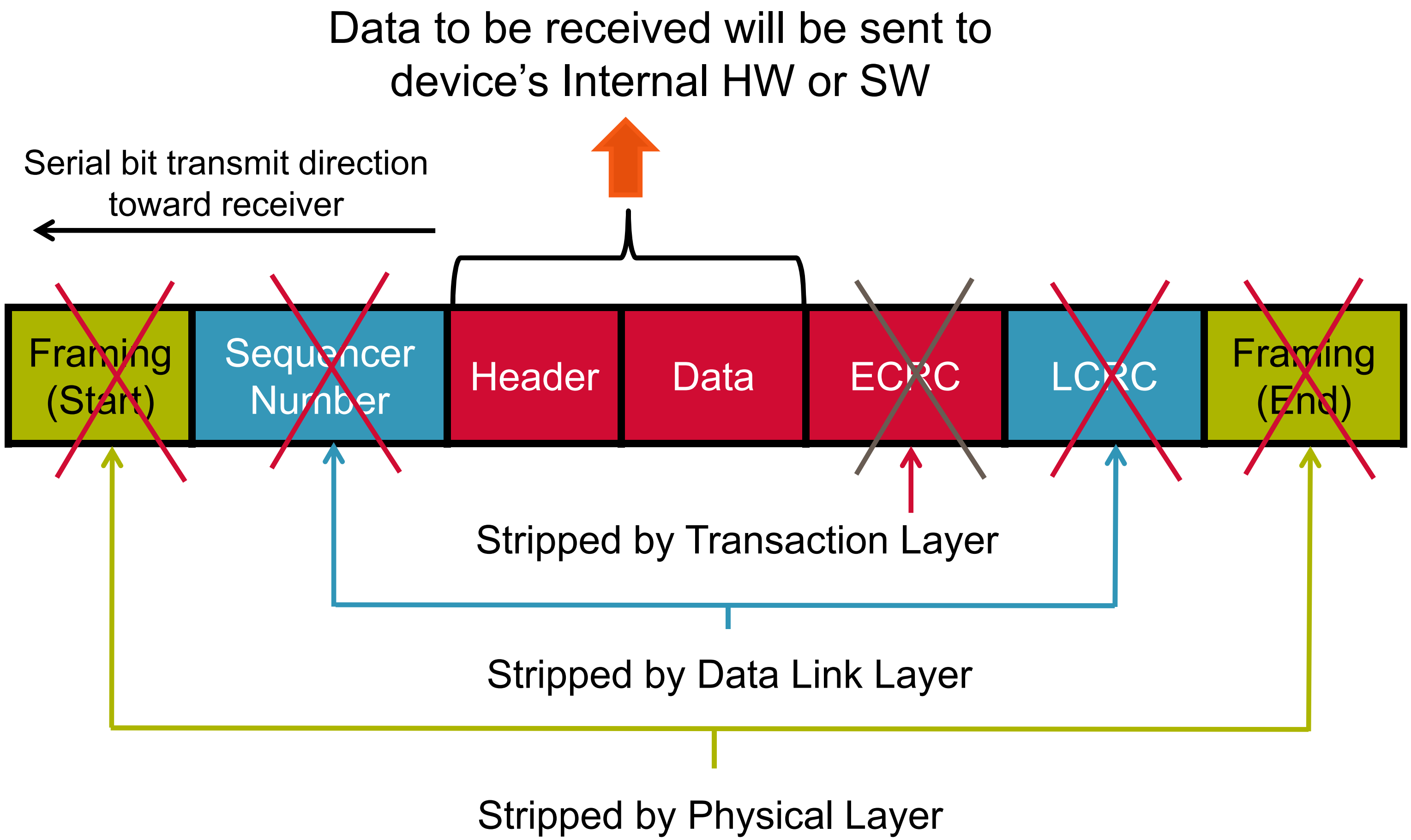
PCI Express Transaction Types

Transaction			Completion	
Request Type	Request TLP	Non-Posted or Posted	Required	Packet Type
Memory Read	MRd	Non-Posted	Yes	CplD, Cpl (error)
Memory Write	MWr	Posted	NO	
Memory Read Lock	MRdLk	Non-Posted	Yes	CplD, Cpl (error)
IO Read	IORd	Non-Posted	Yes	CplD, Cpl (error)
IO Write	IOWr	Non-Posted	Yes	Cpl
Configuration Read	CfgRd0, CfgRd1	Non-Posted	Yes	CplD, Cpl (error)
Configuration Write	CfgWr0, CfgWr1	Non-Posted	Yes	Cpl
Message w/o Data	Msg	Posted		
Message w/Data	MsgD	Posted		

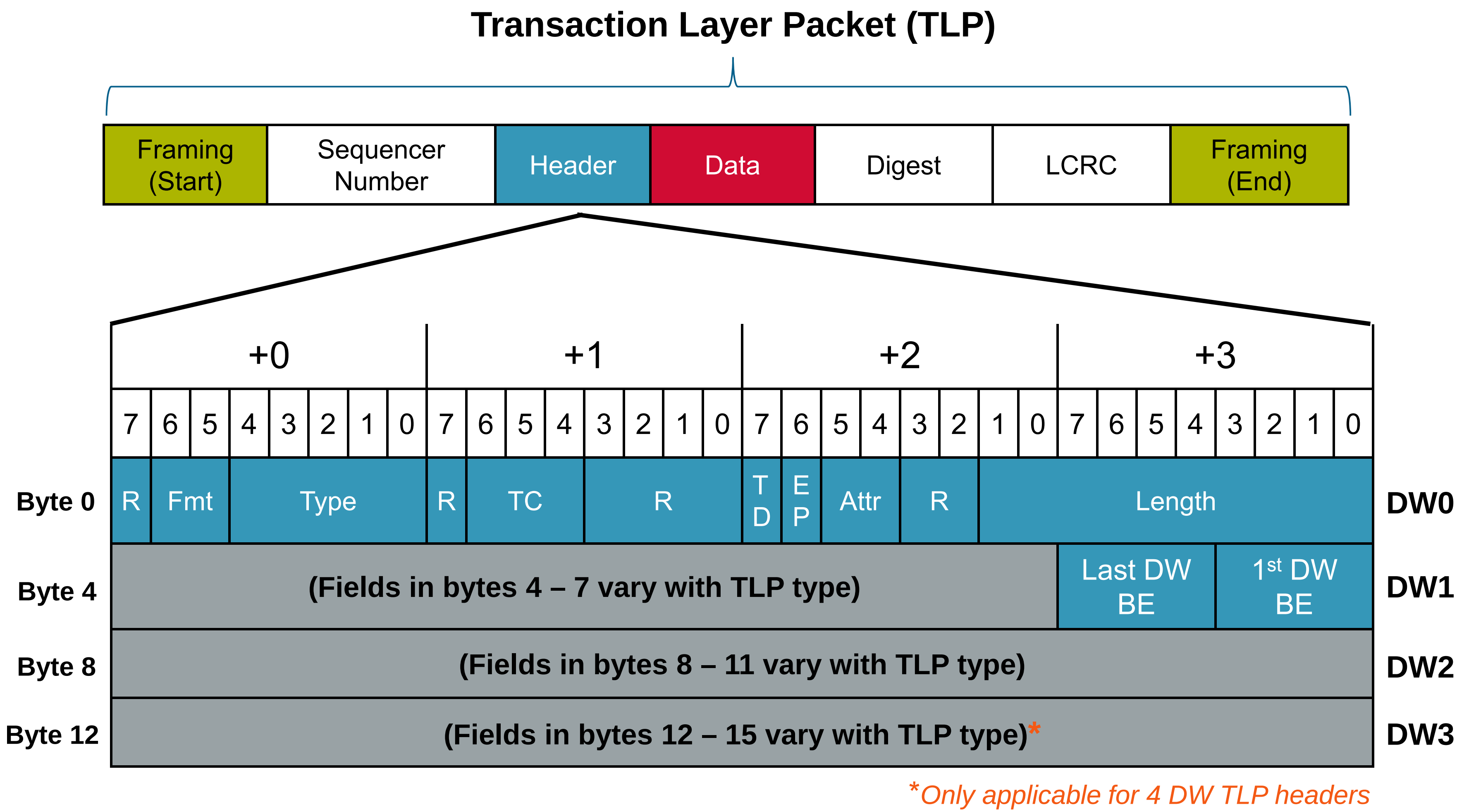
PCI Express TLP (Transaction Layer Packet) Assembly



PCI Express TLP Disassembly



PCI Express TLP Generic Header Fields



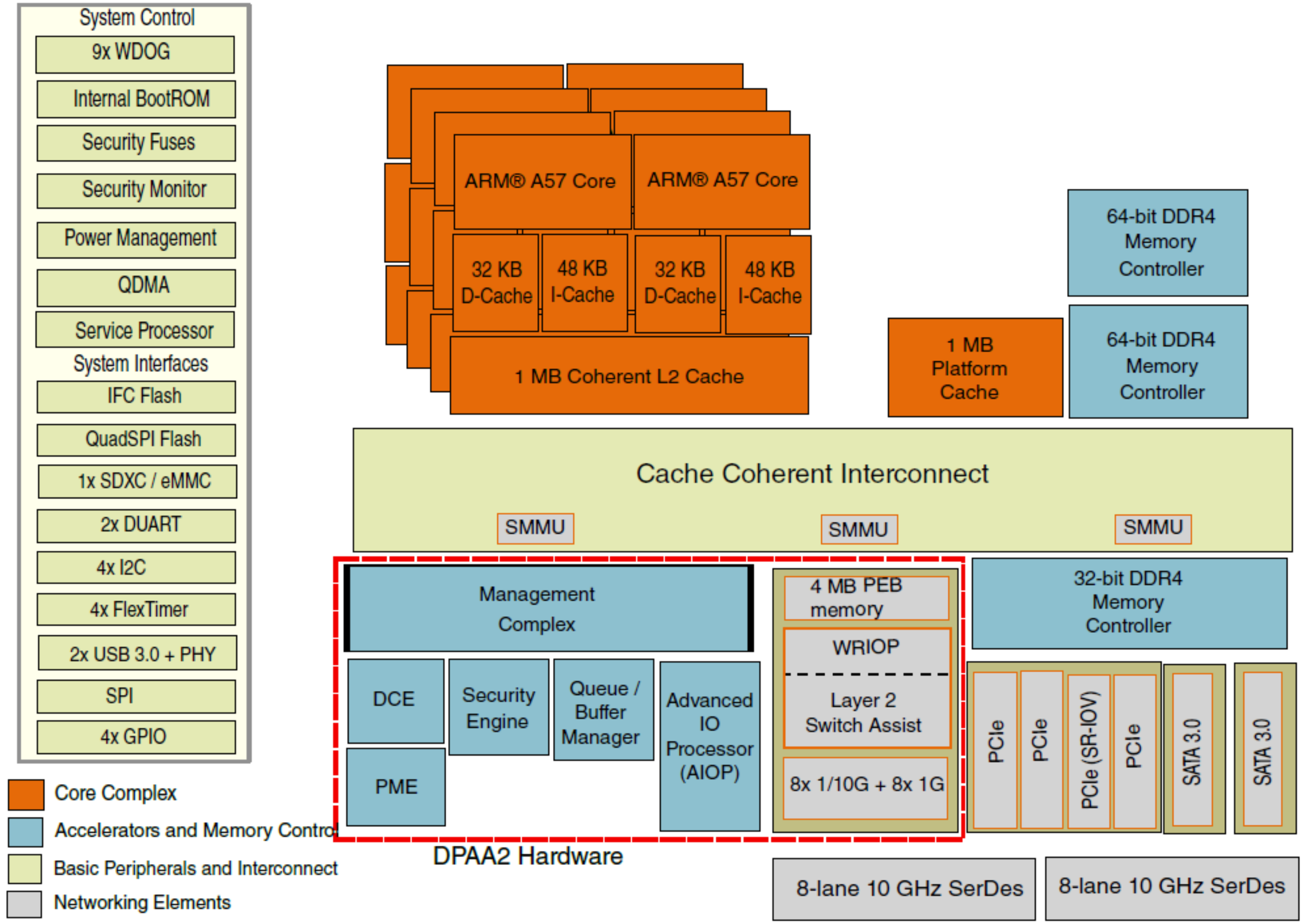
PCI Express TLP Fmt[1:0] and Type Encoding

Table 2-3: Fmt[1:0] and Type[4:0] Field Encodings

TLP Type	Fmt [1:0][2]	Type [4:0]	Description
MRd	00 01	0 0000	Memory Read Request
MRdLk	00 01	0 0001	Memory Read Request-Locked
MWr	10 11	0 0000	Memory Write Request
IORd	00	0 0010	I/O Read Request
IOWr	10	0 0010	I/O Write Request
CfgRd0	00	0 0100	Configuration Read Type 0
CfgWr0	10	0 0100	Configuration Write Type 0
CfgRd1	00	0 0101	Configuration Read Type 1
CfgWr1	10	0 0101	Configuration Write Type 1
Msg	01	$1\ 0r_2r_1r_0$	Message Request – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-11).
MsgD	11	$1\ 0r_2r_1r_0$	Message Request with data payload – The sub-field r[2:0] specifies the Message routing mechanism (see Table 2-11).
Cpl	00	0 1010	Completion without Data – Used for I/O and Configuration Write Completions and Read Completions (I/O, Configuration, or Memory) with Completion Status other than Successful Completion.
CplD	10	0 1010	Completion with Data – Used for Memory, I/O, and Configuration Read Completions.
CplLk	00	0 1011	Completion for Locked Memory Read without Data – Used only in error case.
CplDLk	10	0 1011	Completion for Locked Memory Read – otherwise like CplD.
			All encodings not shown above are Reserved.

LS2085A PCI Express Controller

PCIe Support on QorIQ LS2085A



LS2085A PCIe Controller Feature Highlights

- Four PCI Express Gen3-capable controllers :
  - One controller is SR-IOV EP capable (PEX3)
  - One controller supports up to Gen3 x8
  - Six system memory regions with programmable location and size
    - The size of each region can be from 4 KB to 4 GB

- Major changes compared to Power Architecture SoCs:
  - The PCI Express controllers feature an integrated AXI Bridge Module
  - Configuration space registers are memory mapped within CCSR space for easy access
  - All registers within the PCI Express controllers are little endian now!

LS2085A PCI Express Controller:

- Design Consideration Factors

PCIe Controller Design Considerations

- Major design consideration factors:
  - SerDes link width and lane mapping selection
  - Controller mode selection – RC or EP?
  - SerDes reference clock frequency and clock chip selection
  - Form factor related board design consideration
  - EP Card's power rail requirement

SerDes Link Width and Lane Mapping Selection

- Consideration factors:
  - x1, x4, x8?
  - How many and which PCI Express controllers to use?
  - Sharing with other SerDes protocol on the same SerDes?
- Refer to Table 25-1 and 25-2 to plan for the SerDes usage

Table 25-1. SerDes 1

SRDS_PRCTL_S1	H	G	F	E	D	C	B	A	PLL mapping
hex	SD1[0]	SD1[1]	SD1[2]	SD1[3]	SD1[4]	SD1[5]	SD1[6]	SD1[7]	H-A
03	*PCIe1*				*PCIe2*				22222222
05	SG1	SG2	SG3	SG4	*PCIe2*				22222222
07	SG1	SG2	SG3	SG4	SG5	SG6	SG7	SG8	22222222

Table 25-2. SerDes 2 (continued)

SRDS_PRCTL_S2	A	B	C	D	E	F	G	H	PLL mapping
1E	SG9	SG10	SG11	*SG12*	SG13	SG14	SG15	SG16	22212222
20	SG9	SG10	*SG11*	SG12	SG13	SG14	SG15	SG16	22122222
22	SG9	*SG10*	SG11	SG12	SG13	SG14	SG15	SG16	21222222
24	*SG9*	SG10	SG11	SG12	SG13	SG14	SG15	SG16	12222222
3D	*PCIe3*								22222222
3E	*PCIe3*								22222222
3F	*PCIe3*				*PCIe4*				22222222
40	*PCIe3*				*PCIe4*				22222222
41	PCIe3				PCIe4		SATA1	SATA2	11111122
42	PCIe3				PCIe4		SATA1	SATA2	11111122

PCIe Controller Mode Selection – RC or EP?

- Consideration factors:
  - Want the PCIe controller to act as host? → RC!
    - RC is responsible to bus enumeration, resource planning, power management control as well as interrupt handling
    - RC's configuration space contains Type 1 Header registers
  - Designing an add-in card to be plugged into a host system? → EP!
    - EP's configuration space contains Type 0 Header registers
  - Want some of the PCIe controllers as host/RC and others as EP?

PCIe SerDes Reference Clock Considerations

- Reference Clock Frequency consideration factors:
  - PCIe Base Spec and CEM Spec only call out 100 MHz!!!
  - QorIQ processors do support both 100 and 125 MHz
  - Use 100 MHz if possible!
    - For RC-mode design, think about the potential EP cards (especially off-the-shelf ones) to be plugged into your RC system's PCIe slot. What if that EP card follows PCIe CEM spec and only accepts 100-MHz Ref. Clock?
    - For EP-mode design, think about this consequence: your EP card is plugged into an unknown RC's (for example Windows PC's) PCIe slot offering only 100-MHz Ref. Clock by your customer, unfortunately, your EP card is designed to accept only 125-MHz Ref. Clock in POR or RCW and can't be changed due to the sharing with other SerDes protocol in the same SerDes bank!

PCIe SerDes Reference Clock Consideration (cont.)

- Using different SerDes Ref. Clock on two sides of the link?
  - Theoretically, it's okay to have 125-MHz +/-300ppm Ref. Clock at one end and 100MHz +/-300ppm Ref. Clock at the other end of the link
  - However, avoid this if possible to simplify the design
- PCI Express does support spread-spectrum clocking
  - Common clock must be used – exactly same clock used for both sides of the link!
- SerDes Ref. Clock Driver Chip selection
  - Refer to the HSSI chapter of each Freescale device's hardware specification for detailed DC and AC specification requirement
  - Pay attention to the DC amplitude spec. Refer to AN4311 application note for recommended connection scheme
  - Common Connection scheme: HCSL DC-coupled, LVDS AC-coupled
  - ***The ultimate connection scheme should comes from clock driver chip vendor!***

Form Factor Related Board Design Consideration

- For RC-mode design
  - Consider implementing PCIe slot if possible
  - If connecting RC and EP with direct PCB trace on the same board, consider implementing Mid-Bus Probe footprint for debug
  - Feed a common 100-MHz Ref. Clock to the slot to avoid surprise!
- For EP-mode design
  - Design for 100-MHz Ref. Clock if this is a PCIe plug-in card
- Follow PCI Express base and CEM specifications
  - Design and simulate the channel budget for transmitter and receiver
    - Jitter, insertion loss, skew (lane-to-lane and within a differential pair), crosstalk, equalization, trace impedance and propagation delay
  - Perform Tx and Rx eye measurement on the prototype board

EP Card's Power Rail Requirement

- Follow PCIe CEM Specification's requirement
  - Power consumption depends on card form factor
  - Refer to Section 4.2, "Power Consumption" of the PCIe CEM Spec Rev 3.0 for power dissipation and initial power draw requirement!

PCI EXPRESS CARD ELECTROMECHANICAL SPECIFICATION, REV. 3.0

Table 4-1: Power Supply Rail Requirements

Power Rail	10 W Slot	25 W Slot	150W-ATX Power Connector	75 W Slot
+3.3V				
Voltage tolerance	± 9% (max)	± 9% (max)	N/A	± 9% (max)
Supply Current	3.0 A (max)	3.0 A (max)		3.0 A (max)
Capacitive Load	1000 µF (max)	1000 µF (max)		1000 µF (max)
+12V				
Voltage tolerance	± 8%	± 8%	+5% / -8% (max)	± 8%
Supply Current	0.5 A (max)	2.1 A (max)	6.25 A (max)	5.5 A (max)
Capacitive Load	300 µF (max)	1000 µF (max)		2000 µF (max)
+3.3Vaux			N/A	
Voltage tolerance	± 9% (max)	± 9% (max)		± 9% (max)
Supply Current				
Wakeup Enabled	375 mA (max)	375 mA (max)		375 mA (max)
Non-wakeup Enabled	20 mA (max)	20 mA (max)		20 mA (max)
Capacitive Load	150 µF (max)	150 µF (max)		150 µF (max)

EP Card’s Power Rail Requirement (cont.)

- Consideration factor when designing x4/x8 EP Cards
  - There are far fewer 75W slots in x86 PC motherboards than 25W slots, even though PCIe CEM Spec Rev 3.0 increases the maximum power dissipation for x4/x8 cards from 25W to 75W.
  - Therefore, the following old requirement from the PCIe CEM Spec Rev 2.0 is safer, although more conservative.

PCI EXPRESS CARD ELECTROMECHANICAL SPECIFICATION, REV. 2.0

Table 4-2: Add-in Card Power Dissipation

	X1		x4/x8	x16	
Standard height	10 W[1] (max)	25 W[1] (max)	25 W (max)	25 W[2] (max)	75 W[2, 4] (max)
Low profile card[3]	10 W (max)		25 W (max)	25 W (max)	

LS2085A PCI Express Controller:

- RCW Configuration

RCW Configuration – Which SerDes PLL to Use

- Identify which PLL is used for the selected SRDS_PRTCL_Sn
  - Power down unused PLL.
  - For the PLL in use, select the valid SerDes Ref. Clock Frequency.

Table 25-1. SerDes 1 (continued)

SRDS_PRCTL_S1	H	G	F	E	D	C	B	A	PLL mapping
32	XAUI1				XAUI2				11111111
33	QSGa	QSGb	QSGc	QSGd	*PCIe2*				22222222
35	XFI1	XFI2	XFI3	XFI4	PCIe2	QSGb	QSGc	QSGd	11112222
37	XFI1	XFI2	QSGc	QSGd	PCIe2				11222222
39	*PCIe1*	SG2	SG3	SG4	*PCIe2*	SG6	SG7	SG8	22222222
3B	PCIe1	XFI2	XFI3	XFI4	PCIe2	XFI6	XFI7	XFI8	21112111

PLL1 is used on the selected lanes

PLL2 is used on the selected lanes

Table 25-2. SerDes 2 (continued)

SRDS_PRCTL_S2	A	B	C	D	E	F	G	H	PLL mapping
40	*PCIe3*				*PCIe4*				22222222
41	PCIe3				PCIe4		SATA1	SATA2	1111111[illegible]
42	PCIe3				PCIe4		SATA1	SATA2	11111122
43	*PCIe3*				X	X	SATA1	SATA2	11111122
44	*PCIe3*				X	X	SATA1	SATA2	11111122

PLL2 is used on the selected lanes

PLL1 is used on the selected lanes

RCW Configuration – SerDes Ref. Clock Freq. and Divider

- Follow the following three steps to configure:
  1) Select the SRDS_PRTCL_Sn based on desired SerDes lane mapping
  2) Select & ***Provide the SAME*** valid SerDes Ref. Clock Frequency for the PLL in use!!!
  3) Power down the unused PLL
  4) Select the valid SRDS_DIV_PEX_Sn for the PLL in use

Table 25-5. Valid SerDes Reference Clocks and RCW Encodings

SerDes Protocol (given lane)	Valid reference clock frequency	Valid setting as determined by SRDS_PRTCL_Sn	Valid setting as determined by SRDS_PLL_REF_CLK_SEL_Sn	Valid setting as determined by SRDS_DIV_[prot]_Sn
SGMII (1.25 Gbps)	100 MHz	SG @ 1.25 Gbps	0: 100 MHz	Don't Care
	125 MHz		1: 125 MHz	
2.5x SGMII (3.125 Gbps)	125 MHz	SG @ 3.125 Gbps	0: 125 MHz	Don't Care
	156.25 MHz		1: 156.25 MHz	
QSGMII (5.0 Gbps)	100 MHz	Any QSG	0: 100 MHz	Don't Care
	125 MHz		1: 125 MHz	
XAUI (3.125 Gbps)	125 MHz	XAUI @ 3.125 Gbps	0: 125 MHz	Don't Care
	156.25 MHz		1: 156.25 MHz	
XFI (10.3125 Gbps) or 10GBASE-KR	156.25 Mhz	XFI @ 10.3125 Gbps	0: 156.25 MHz	Don't Care
XFI (10.3125 Gbps)	161.1328125 MHz (hidden from cust)	XFI @ 10.3125 Gbps	1: 161.1328125 MHz	Don't Care
PCI Express 2.5 Gbps (doesn't negotiate upwards)	100 MHz[1]	Any PCIe	0: 100 MHz	2'b10: 2.5 G
	125 MHz[1]		1: 125 MHz	
PCI Express 5 Gbps (can negotiate up to 5 Gbps)	100 MHz[1]	Any PCIe	0: 100 MHz	2'b01: 5.0 G
	125 MHz[1]		1: 125 MHz	
PCI Express 8 Gbps (can negotiate up to 8 Gbps)	100 MHz[1]	Any PCIe	0: 100 MHz	2'b00: 8.0 G
	125 MHz[1]		1: 125 MHz	
SATA (1.5, 3 or 6 Gbps)	100 MHz	Any SATA	0: 100 MHz	Don't Care[2]
	125 MHz		1: 125 MHz	

If possible, **ALWAYS** choose 100 MHz

If possible, restrict the maximum link speed

Case Study – RCW SerDes Parameter Configuration

- Example: usage requires SerDes 2, SRDS_PRTCL_S2 = 0x42
  1) PEX3 is mapped to SerDes 2 Lane A-D, allowed to run up to Gen2 x4, using PLL1
  2) PEX4 is mapped to SerDes 2 Lane E-F, allowed to run up to Gen2 x2, using PLL1
  3) SATA1 or SATA2 is mapped to SerDes 2 Lane G or H respectively, allowed to run up to 6 Gbps, using PLL2
  4) In RCW (***little endian***), configure the parameters as below:
     a) RCW [488:487] = HOST_AGT_PEX = 2'b00; //assume PEX4 & PEX3 run in host/RC mode
     b) RCW [899:898] = SRDS_PLL_PD_PLL[4:3] = 2'b00; //SerDes 2 PLL2 & 1 not powered down
     c) RCW [931:930] = SRDS_PLL_REF_CLK_SEL_S2 = 2'b00; //Use 100 MHz for SD2 PLL2 & 1
     d) RCW [947:946] = SRDS_DIV_PEX_S2 = 2'b01; //PEX4 & PEX3 can run up to 5.0G

Table 25-2. SerDes 2 (continued)

SRDS_PRCTL_S2	A	B	C	D	E	F	G	H	PLL mapping
40	*PCIe3*				*PCIe4*				2222[illegible]22
41	PCIe3				PCIe4		SATA1	SATA2	111111[illegible]
42	PCIe3				PCIe4		SATA1	SATA2	111111[illegible]22
43	*PCIe3*				X	X	SATA1	SATA2	[illegible]11122
44	*PCIe3*				X	X	SATA1	SATA2	[illegible]1111122

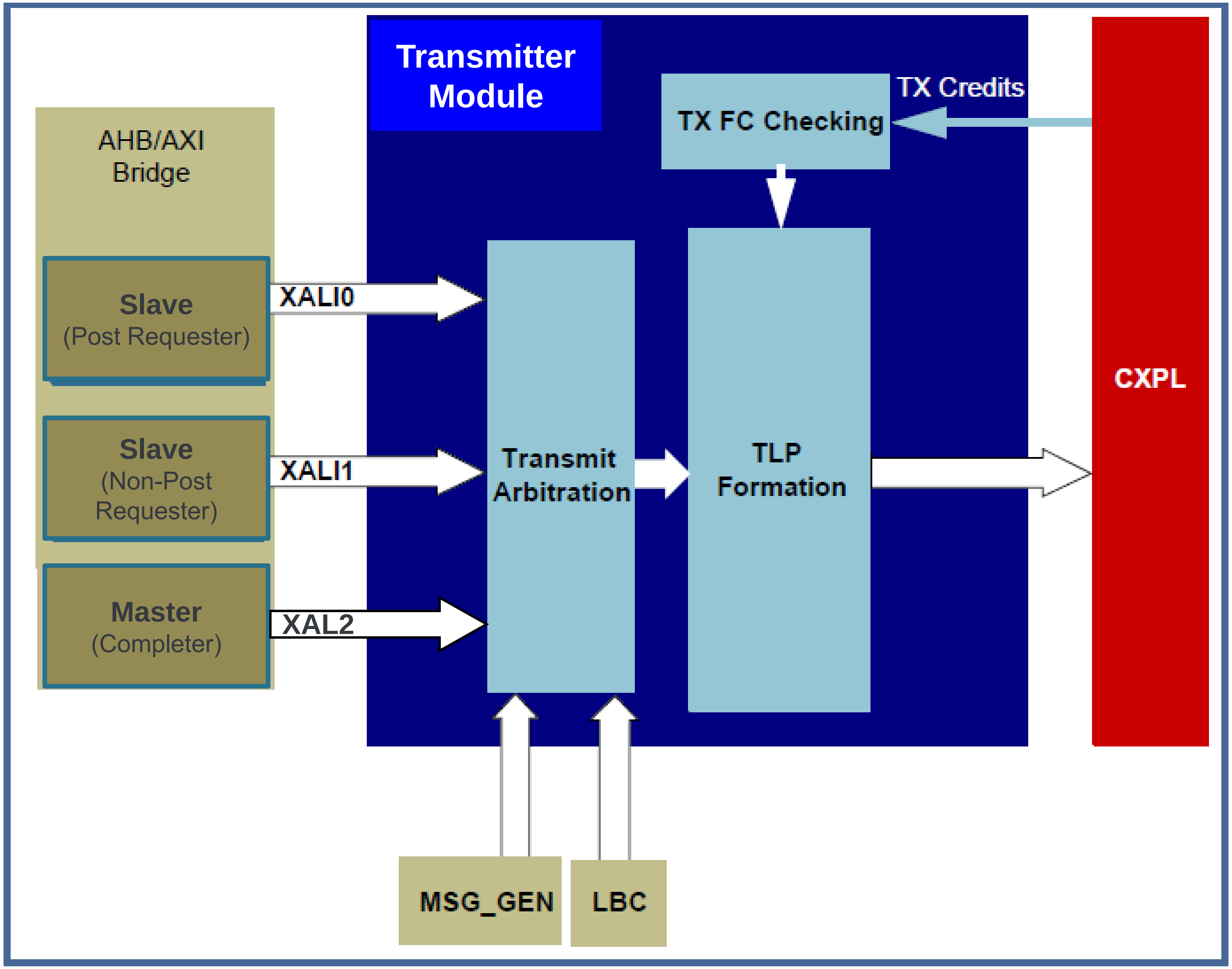
PLL2 is used on the selected lanes

PLL1 is used on the selected lanes

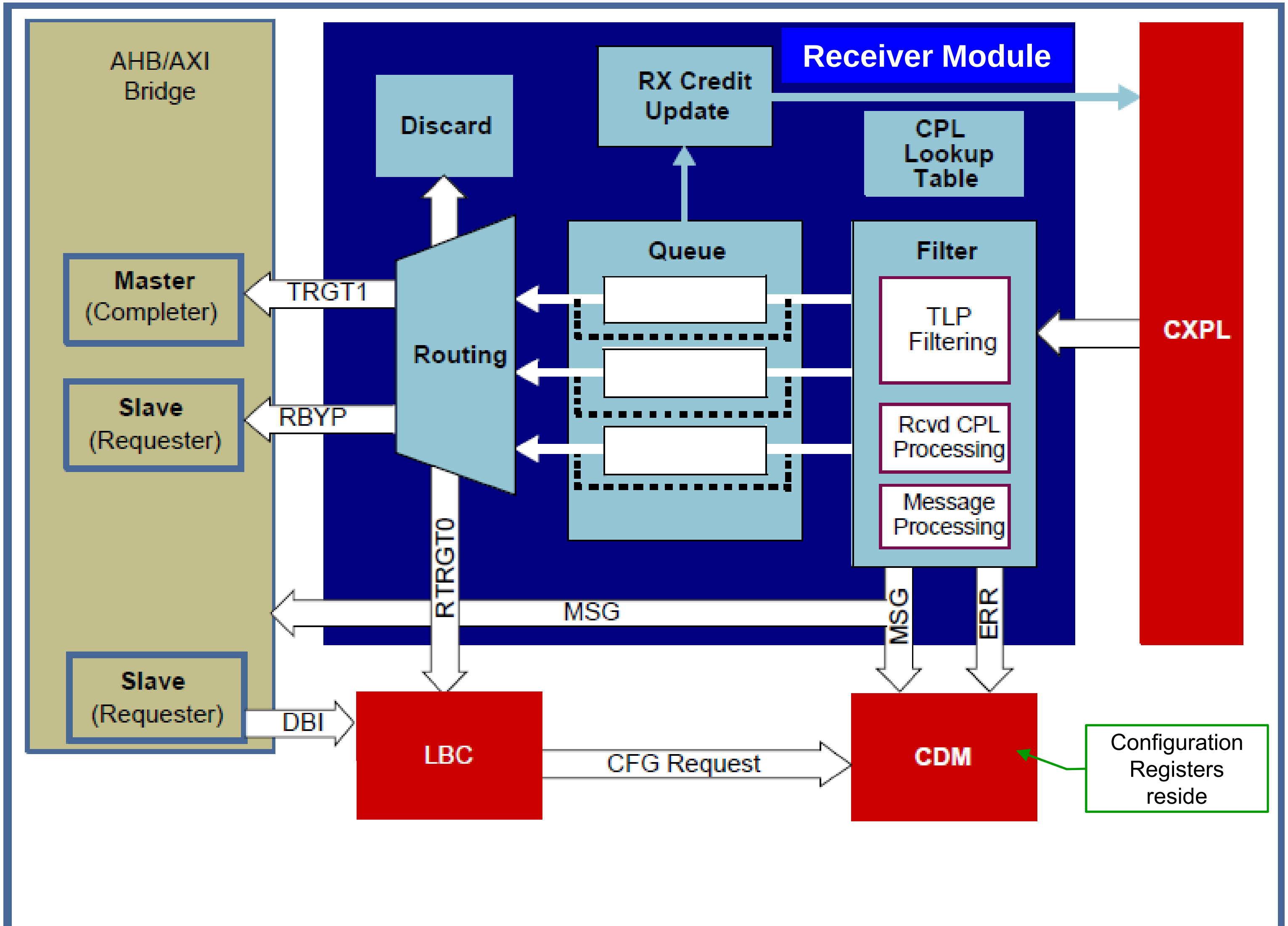
LS2085A PCI Express Controller:

- Architecture Overview

LS2085A PCIe Controller Transmit Path Block Diagram



LS2085A PCIe Controller Receive Path Block Diagram



iATU Address Translation Diagram

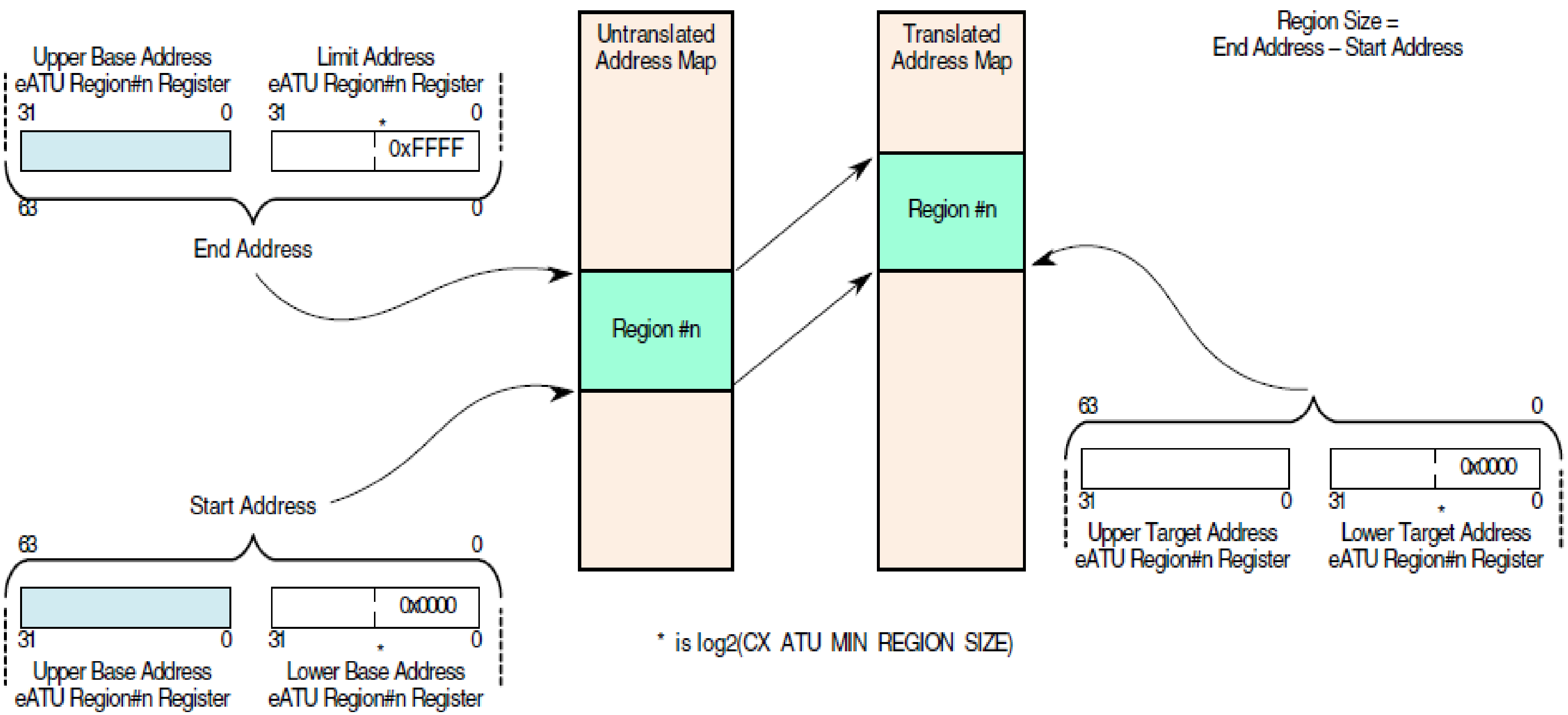


Figure 20-238. iATU Address Region Mapping: Outbound and Inbound (Address Match Mode), 64-bit Address

LS2085A PCIe Controller Register and Memory Space

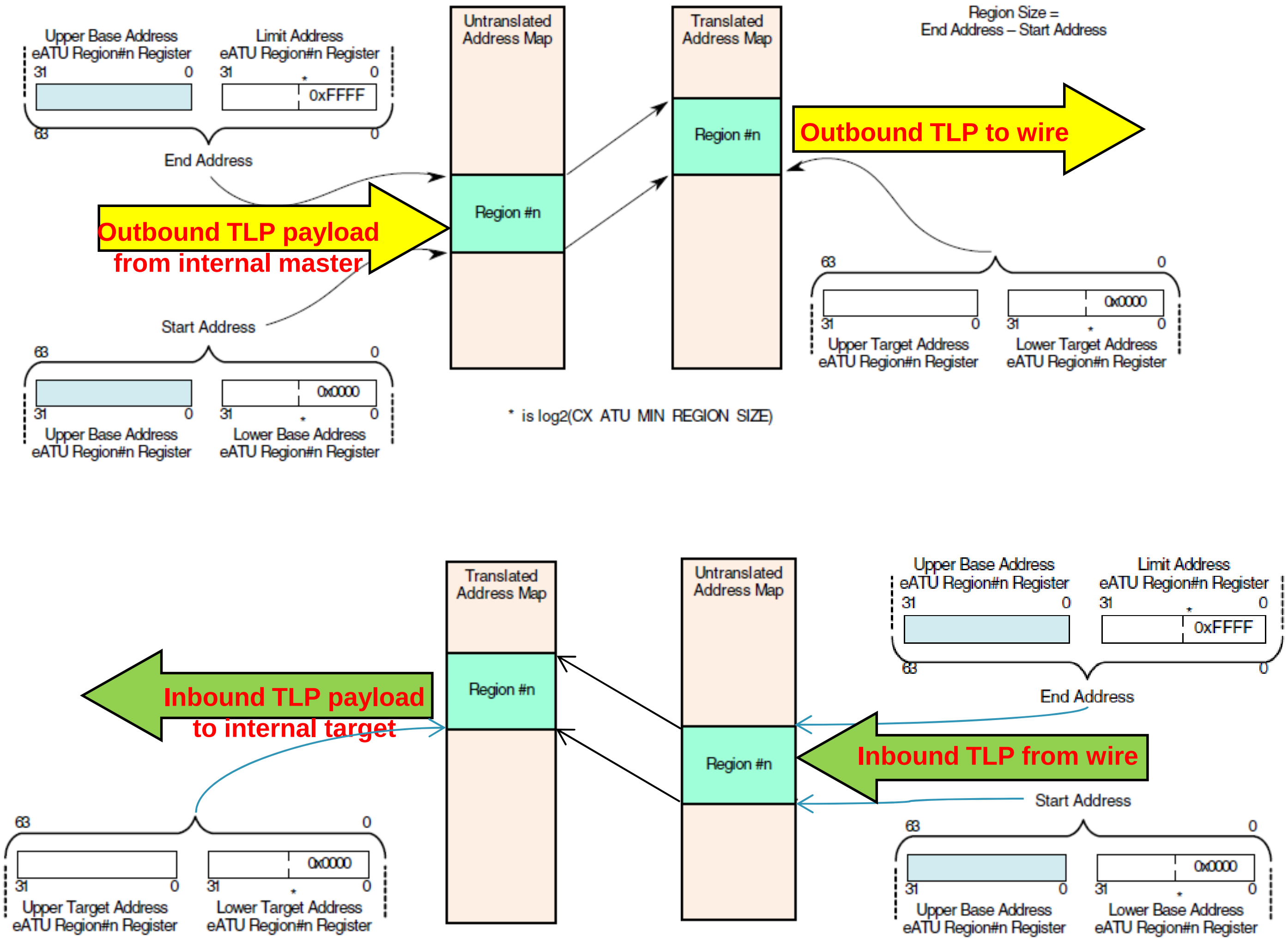
Controller	Usage	Address	Note
PEX1	PEX1 Configure Registers	0x340_0000 – 0x340_FFFF	CCSR space
	PEX1 Lookup Table	0x348_0000 – 0x348_FFFF	CCSR space
	Usable memory space (8 GB)	0x10_0000_0000 – 0x11_FFFF_FFFF	System Memory space
PEX2	PEX2 Configure Registers	0x350_0000 – 0x350_FFFF	CCSR space
	PEX2 Lookup Table	0x358_0000 – 0x358_FFFF	CCSR space
	Usable memory space (8 GB)	0x12_0000_0000 – 0x13_FFFF_FFFF	System Memory space
PEX3 (SR-IOV Capable)	PEX3, PF0, 1 Configure Registers	0x360_0000 – 0x360_FFFF	CCSR space
	PEX3 Lookup Table	0x368_0000 – 0x368_FFFF	CCSR space
	Usable memory space (8 GB)	0x14_0000_0000 – 0x15_FFFF_FFFF	System Memory space
PEX4	PEX4 Configure Registers	0x370_0000 – 0x370_FFFF	CCSR space
	PEX4 Lookup Table	0x378_0000 – 0x378_FFFF	CCSR space
	Usable memory space (8 GB)	0x16_0000_0000 – 0x17_FFFF_FFFF	System Memory space

LS2085A PCIe Internal System Memory Sample Usage

- Using PCIe Controller 1 (PEX1), RC Mode as example

Region Direction	Usable *Internal* System Memory	Possible Region Usage Example (up to 6 regions *per direction*)
Outbound Access	0x10_0000_0000 – 0x11_FFFF_FFFF (8 GB total, using PEX1 as example)	1 for generating CFG0 TLP
		1 for generating CFG1 TLP
		4 left for outbound MWr & MRd TLPs
Inbound Access	CCSR: 0x0100_0000 – 0x0FFF_FFFF (assume allowing inbound CCSR access)	1 for inbound MWr & MRd to BAR0
	Other internal system memory space (***Non-PEX!!!*** Example: DDR and etc.)	5 left for other inbound MWr & MRd access

iATU Address Match Mode Translation Example – for RC



LS2085A PCI Express Controller:
- Initialization and iATU Programming

LS2085A Major Power-On Reset Sequence & Stages

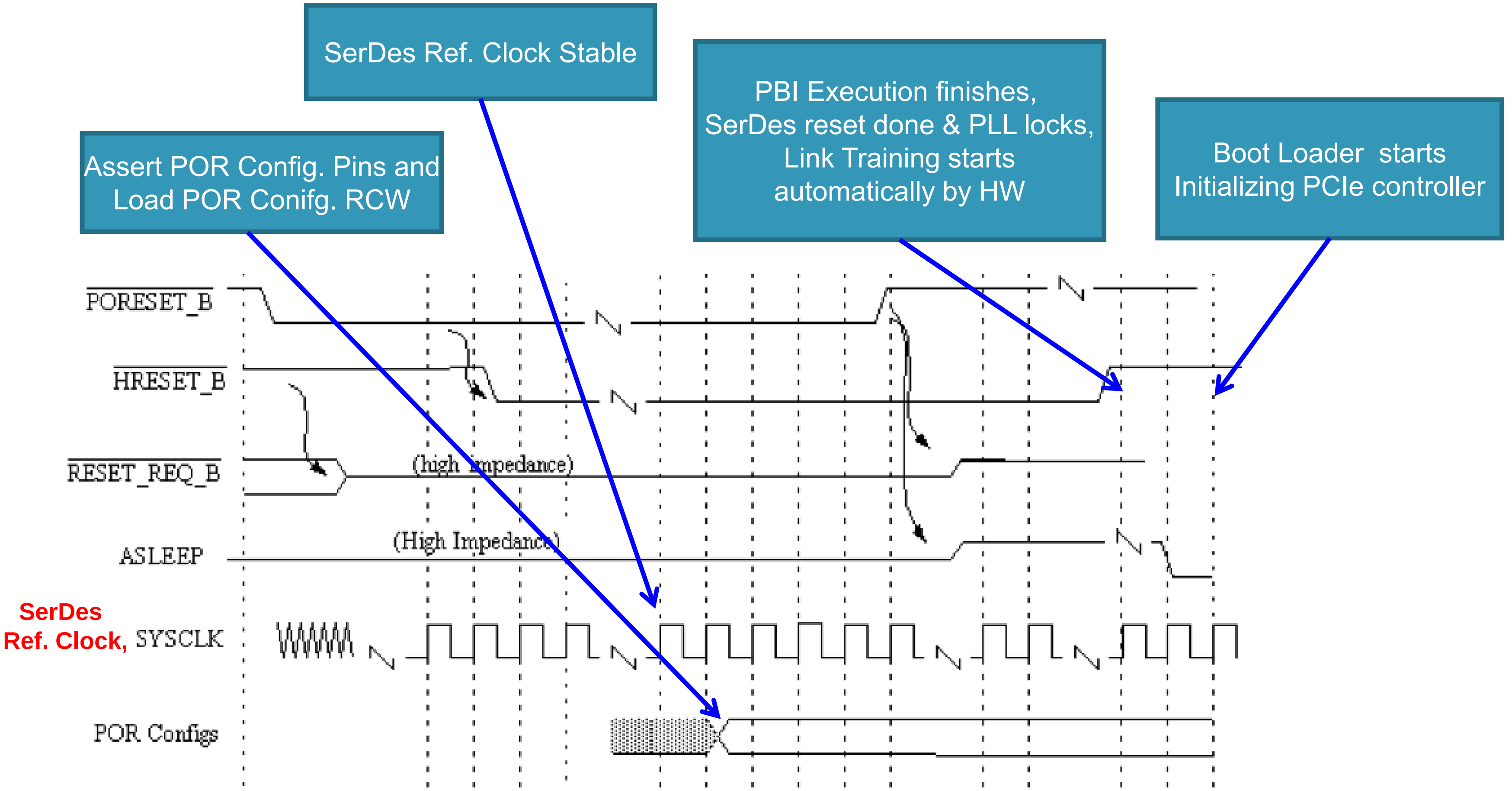


Figure 4-18. Power-On Reset Sequence

Note: Only PCI Express related content is shown here.

PCIe Controller's Configuration Space *defined by base spec*

- The PCI Compatible Configuration Header :
  - The first 64 Bytes (0x00 – 0x3F) in the PCI/PCI Express Configure Space – PCI Legacy Software compatible
  - Which Header Type to use?
    - RC: use Type 1 Header
    - EP: use Type 0 Header
- The PCI Compatible Device-Specific Configuration Space:
  - Power Management Capability Structure – for both RC & EP
  - The PCI Express specific registers sit here – for both RC & EP, some registers and/or bits have different definition or only applicable in either RC or EP.
  - MSI and MSI-X Message Capability Structure – for EP Only!
- The PCI Express Extended Configuration Space:
  - PCI Express Advanced Error Reporting Capability Structure
  - PCI Express Extended ARI Capability Structure – for ARI Forwarding Capable RC Only!
  - PCI Express SR-IOV Capability Structure – for SR-IOV EP Only!

PCIe Controller Type 0 Configuration Header – for EP

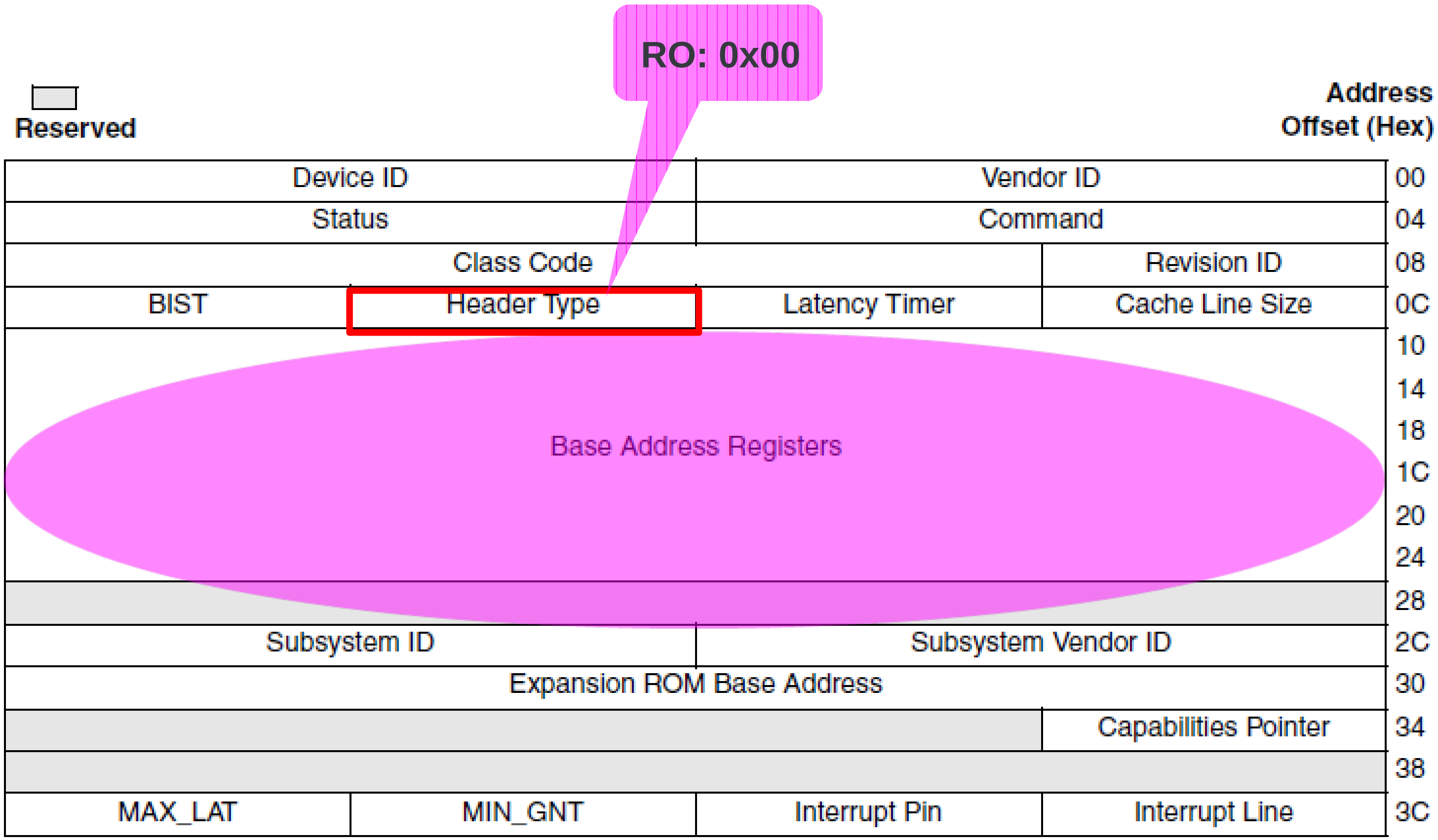


Figure 20-76. PCI Express PCI-Compatible Configuration Header—Type 0

PCIe Controller Type 1 Configuration Header – for RC

RO: 0x01

Reserved

				Address Offset (Hex)
Device ID		Vendor ID		00
Status		Command		04
Class Code			Revision ID	08
BIST	Header Type	Latency Timer	Cache Line Size	0C
Base Address Register 0				10
				14
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	18
Secondary Status		I/O Limit	I/O Base	1C
Memory Limit		Memory Base		20
Prefetchable Memory Limit		Prefetchable Memory Base		24
Prefetchable Base Upper 32 Bits				28
Prefetchable Limit Upper 32 Bits				2C
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		30
			Capabilities Pointer	34
Expansion ROM Base Address				38
Bridge Control		Interrupt Pin	Interrupt Line	3C

Figure 20-90. PCI Express PCI-Compatible Configuration Header—Type 1

PCIe Controller PCIe Device-Specific Configuration Header

				Offset
PCI-Compatible Configuration Header (See Section 20.3.9, "PCI Compatible Configuration Headers," for more information.)				00–3F
Power Mgmt Capabilities		Next Pointer (0x50)	Power Mgmt Capability ID	40
Data		Power Management Status & Control		44
				48–4C
MSI Message Control		Next Pointer (70)	MSI Message Capability ID	50
MSI Message Address				54
MSI Upper Message Address				58
		MSI Message Data		5C
				60–6C
PCI Express Capabilities		Next Pointer (0xB0—EP mode) (NULL—RC mode)	PCI Express Capability ID	70
Device Capabilities				74
Device Status		Device Control		78
Link Capabilities				7C
Link Status		Link Control		80
Slot Capabilities				84
Slot Status		Slot Control		88
		Root Control (RC mode only)		8C
Root Status				90
Device Capabilities 2				94
Device Status 2		Device Control 2		98
Link Capabilities 2				9C
Link Status 2		Link Control 2		A0
				A4–AC
MSI-X Message Control		Next Pointer (NULL)	MSI-X Message Capability ID	B0
Table Offset				B4
PBA Offset				B8
				BC–FF

Applicable to EP Only (MSI registers, 50–5C)

Applicable to RC Only (Slot and Root registers, 84–90)

Applicable to SR-IOV EP Only (MSI-X registers, B0–B8)

PCIe Controller Extended Configuration Space

Reserved (shaded)

		Address Offset (Hex)
PCI Compatible Configuration Header (See Section 20.3.9, "PCI Compatible Configuration Headers," for more information.)		000–03F
PCI-Compatible Device-Specific Configuration Space (See Section 20.3.10, "PCI Compatible Device-Specific Configuration Space," for more information.)		040–0FF
Next Capability Offset (RC=NULL;EP=0x140)/ Capability Version	Advanced Error Reporting Capability ID	100
Uncorrectable Error Status		104
Uncorrectable Error Mask		108
Uncorrectable Error Severity		10C
Correctable Error Status		110
Correctable Error Mask		114
Advanced Error Capabilities and Control		118
Header Log		11C, 120, 124, 128
Root Error Command		12C
Root Error Status		130
Error Source ID	Correctable Error Source ID	134
(Reserved)		138
(Reserved)		13C
Next Capability Offset (0x158)/Capability Version	ARI Extended Capability	148
ARI Control	ARI Capabilites	14C
(Reserved)		150
(Reserved)		154
Next Capability Offset (NULL)/Capability Version	SR-IOV Extended Capability ID	178
SR-IOV Capabilities		17C
SR-IOV Status	SR-IOV Control	180
TotalVFs (RO)	InitialVFs (RO)	184
(Reserved) / Function Dependency Link	NumVFs (RW)	188
VFStride (RO)	FirstVF Offset (RO)	18C
VF Device ID	(Reserved)	190
Supported Page Size (RO)		194
System Page Size (RW)		198
VF BAR0 (RW)		19C
VF BAR1 (RW)		1A0
VF BAR2 (RW)		1A4
VF BAR3 (RW)		1A8
VF BAR4 (RW)		1AC
VF BAR5 (RW)		1B0

Applicable to RC Only (Root Error Command, Root Error Status, Error Source ID / Correctable Error Source ID)

Applicable to ARI-Forwarding-Capable RC Only (148–14C)

Applicable to SR-IOV EP Only (178–1B0)

PCIe Controller *Internal* Configuration Space

- Gen3 Control Register and DBI-Read-Only Write Enable Register

Byte Offset	Register Name
0x890	Gen3 Control Register
0x8BC	DBI Read-Only Write Enable Register

- iATU (internal Address Translation Unit) registers
  - Accessed by using the Indirect Addressing scheme

Table 20-239. iATU Register Map

Byte Offset	Description
0x900	iATU Index Register
0x904	iATU Region Control 1 Register
0x908	iATU Region Control 2 Register
0x90C	iATU Region Lower Base Address Register
0x910	iATU Region Upper Base Address Register
0x914	iATU Region Limit Address Register
0x918	iATU Region Lower Target Address Register
0x91C	iATU Region Upper Target Address Register
0x920	iATU Region Control 3 Register

Each register actually has two copies:
- One outbound,
- One inbound

Required POR Configuration Before Link Training

- POR configuration to be done before de-asserting HRESET# - by loading RCW during HRESET# assertion period
  1) Host/Agent Mode – Root Complex (RC) / Endpoint (EP) selection
  2) Where to boot from? – boot ROM location
  3) Normal boot or boot hold-off?
  4) SerDes related RCW configuration:
     - Link Width and Lane Assignment – SRDS_PRTCL_Sn
     - Power down unused PLL – SRDS_PLL_PD_PLLn
     - Select valid SerDes Ref. Clock frequency – SRDS_PLL_REF_CLK_SEL_Sn
     - Set desired link speed of the PCI Express controller – SRDS_DIV_PEX_Sn
  5) Minimum CCB frequency selection
  6) PBI code execution if any
  7) *Device can now proceed to de-assert HRESET#*
     - *SerDes will be released from reset and SerDes PLL locks.*
     - *This kicks off the PCI Express link training automatically*

PCIe Controller Initialization After Link Training – RC Mode

- **Internal configuration space registers** – PEXn Block Offset + Register Offset
  1) DBI Read-Only Write Enable Register (0x8BC)
  2) iATU Registers (0x900 – 0x 91C)
- **Standard configuration space registers** – PEXn Block Offset + Register Offset
  1) PCI Compatible Configuration Header (Type 1): 0x00 – 0x3C
     - Minimum to configure:
       - Command Register,
       - BAR0 Register (if allowing EPs to access RC's internal CCSR space)
       - Bus Number Registers,
       - Mem_Base/Limit Register,
       - Prefetchable Mem_Base/Limit Register,
       - Bridge Control Register

PCIe Controller Initialization After Link Training – RC Mode

- **Standard configuration space registers** – PEXn Block Offset + Register Offset (continue)
  2) PCI Compatible Device-Specific Configure Space: 0x40 – 0xFF
     - Power Management Capability Structure: 0x40 – 0x47
       - Used by OS/PM Driver, normally no need to configure
     - PCI Express Capability Structure: 0x70 – 0xA3
       - Minimum to configure:
         - Device Control Register
         - Link Control Register, Root Control Register
  3) PCI Express Advanced Error Report. Capability Structure: 0x100 – 0x137
     - Minimum to configure:
       - Uncorrectable Error Mask Register, Correctable Error Mask Register
       - Root Error Command Register
  4) PCI Express ARI Extended Capability Structure: 0x148 – 0x14F
       - Minimum to configure (for ARI Forwarding-Capable RC):
         - ARI Control Register

Caveat before initiating any transaction

- For RC
  - Always check & confirm that the link is up before issuing any outbound configuration cycle to downstream devices
  - Keep Command Register [Bus Master Enable, Memory Space] bits clear until bus scan is finished, all devices are fully configured and ready for memory transaction
- For EP
  - Boot loader needs to finish the EP initialization as fast as possible – there is only 100 msec allowed between the de-assertion of Slot Reset and EP's readiness to accept configuration cycles from the remote host
  - Always check & confirm that the link is up and Command Register [Bus Master Enable, Memory Space] bits are set by the remote RC before initiating any memory transactions

iATU (internal Address Translation Unit) Register Map

- Using an ***Indirect Addressing scheme*** to access the registers except the Index Register
  - Each register address actually has two registers implemented:
    - One Outbound and one Inbound
  - Always write to the Index Register first ***before touching ANY OTHER iATU registers***
  - The Index Register only has two bit fields defined:
    - Bit [31], REGION_DIR → 0b for Outbound; 1b for Inbound
    - Bit [2:0], REGION_INDEX → valid setting = 000b to 101b to cover all 6 regions (windows)

Table 20-239. iATU Register Map

Byte Offset	Description	
0x900	iATU Index Register	→ Controls which particular region's registers to be accessed right after
0x904	iATU Region Control 1 Register	→ Controls some programmable bit fields of a **TLP header DW0**
0x908	iATU Region Control 2 Register	→ Select the **MATCH_MODE** to use and **REGION_EN** bit
0x90C	iATU Region Lower Base Address Register	→ Controls the start & end address of a region prior to
0x910	iATU Region Upper Base Address Register	translation.
0x914	iATU Region Limit Address Register	→ The Limit is the bits [31:12] of the end address of base
0x918	iATU Region Lower Target Address Register	→ Controls the start & end address of a region after the
0x91C	iATU Region Upper Target Address Register	translation.
0x920	iATU Region Control 3 Register	→ Used by SR-IOV EP

iATU Programming Example #1 – RC, Outbound CFG

- What do we need to generate an outbound 3DW Configuration TLP?
  - DW0: → Need to program the Type only!
    - **Fmt** = **00b** for CfgRd0 and CfgRd1; **10b** for CfgWr0 and CfgWr1 → derived from ARM core's instruction
    - Program the "**Type**" bit field in the IATU_REGION_CTRL_1_OFF_OUTBOUND_0 register: **00100** vs. **00101**
  - DW1: → No need to program! Controller hardware fills in the following:
    - **Requester ID** = RC's Bus# : Device# : Function# = **0x00**
    - **Last DW Byte Enable** and **First DW Byte Enable**: determined by ARM core's instruction
  - DW2: → Completer's Bus#, Dev#, Function# and Register# to be accessed!
    - **Completer's B:D:F** is derived from the iATU region's translated Target Address bits [31:16]
    - **Completer's Ext. Register Number** and **Register Number** are derived from the iATU region's translated Target Address bits [11:2]

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td><td></td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>Byte 0</td><td>R</td><td colspan="2">Fmt</td><td colspan="5">Type</td><td>R</td><td colspan="3">TC</td><td colspan="4">R</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td colspan="2">R</td><td colspan="10">Length</td><td>DW0</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="4">Last DW BE</td><td colspan="4">1st DW BE</td><td>DW1</td></tr>
<tr><td>Byte 8</td><td colspan="8">Bus Number</td><td colspan="5">Device Number</td><td colspan="3">Function Number</td><td colspan="4">R</td><td colspan="4">Ext Reg. Number</td><td colspan="6">Register Number</td><td colspan="2">R</td><td>DW2</td></tr>
</table>

**3 DW Configure TLP Header Shown*

iATU Programming Example #1 – RC, Outbound CFG (cont.)

- The mapping of the translated Target Address [31:16] to the completer's Bus# : Device# : Function#

XALI0/1/2 or AHB/AXI Slave Interface Bits	CFG TLP Header Field
31:24	Bus Number
23:19	Device Number
18:16	Function Number
11:8	Extended Register Number
7:2	Register Number

iATU Programming Example #1 – RC, Outbound CFG (cont.)

- iATU register programming procedure – using PEX4 as example:

1) **Set up the Index Register**
   - ❑ Write 0x0000_0001 to Index Register at offset 0x900
     - ❖ Note 1: This sets REGION_DIR = 0 and REGION_INDEX = 001 → use Region# 1 as outbound
     - ❖ Note 2: From now on until the next write to the Index Register, all the iATU registers touched between 0x904 and 0x920) are the "_OUTBOUND_" copies of the registers.

2) **Set up the Region Base and Limit Address Registers (watch: max 40 bits)**
   - ❑ Write 0x0000_0000 to Region# 1's Lower Base Address Register at offset 0x90C
   - ❑ Write 0x0000_0016 to Region# 1's Upper Base Address Register at offset 0x910
   - ❑ Write 0xFFFF_FFFF to Region# 1's Limit Address Register at offset 0x914
     - ❖ Note: This configures the Region# 1 as an 4-Gbyte outbound region with the following base address:

       0x0000_0016_0000_0000 – 0x0000_0016_FFFF_FFFF (valid internal system address for PEX4)

3) **Set up the Region Target Address Registers**
   - ❑ Write 0x0000_0000 to Region# 1's Lower Target Address Register at offset 0x918
   - ❑ Write 0x0000_0000 to Region# 1's Upper Target Address Register at offset 0x91C
     - ❖ Note: This configures the 4-Gbyte outbound Region# 1's target address as

       0x0000_0000_0000_0000 – 0x0000_0000_FFFF_FFFF (bottom 4 Gbytes)

iATU Programming Example #1 – RC, Outbound CFG (cont.)

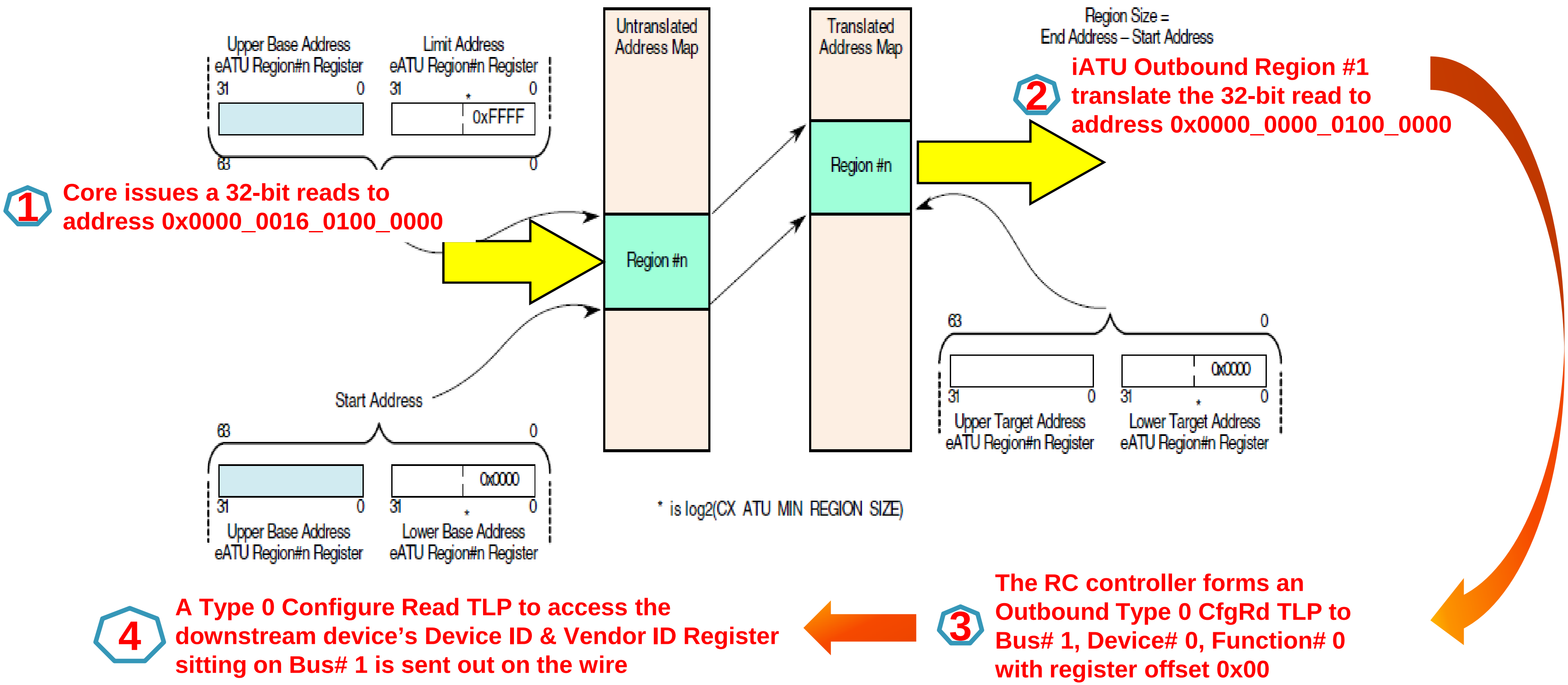
- iATU register programming procedure – using PEX4 as example (continue):

4) **Set up the Region Control 1 Register**
   - ❑ Write 0x0000_0004 to Region# 1's Control 1 Register at offset 0x904
     - ❖ Note: This sets the TYPE = 00100b → Outbound Region# 1 will be used to generate Type 0 CFG TLP

5) **Enable the Region# 1 by writing to Region Control 2 Register**
   - ❑ Write 0x8000_0000 to Region# 1's Control 2 Register at offset 0x908

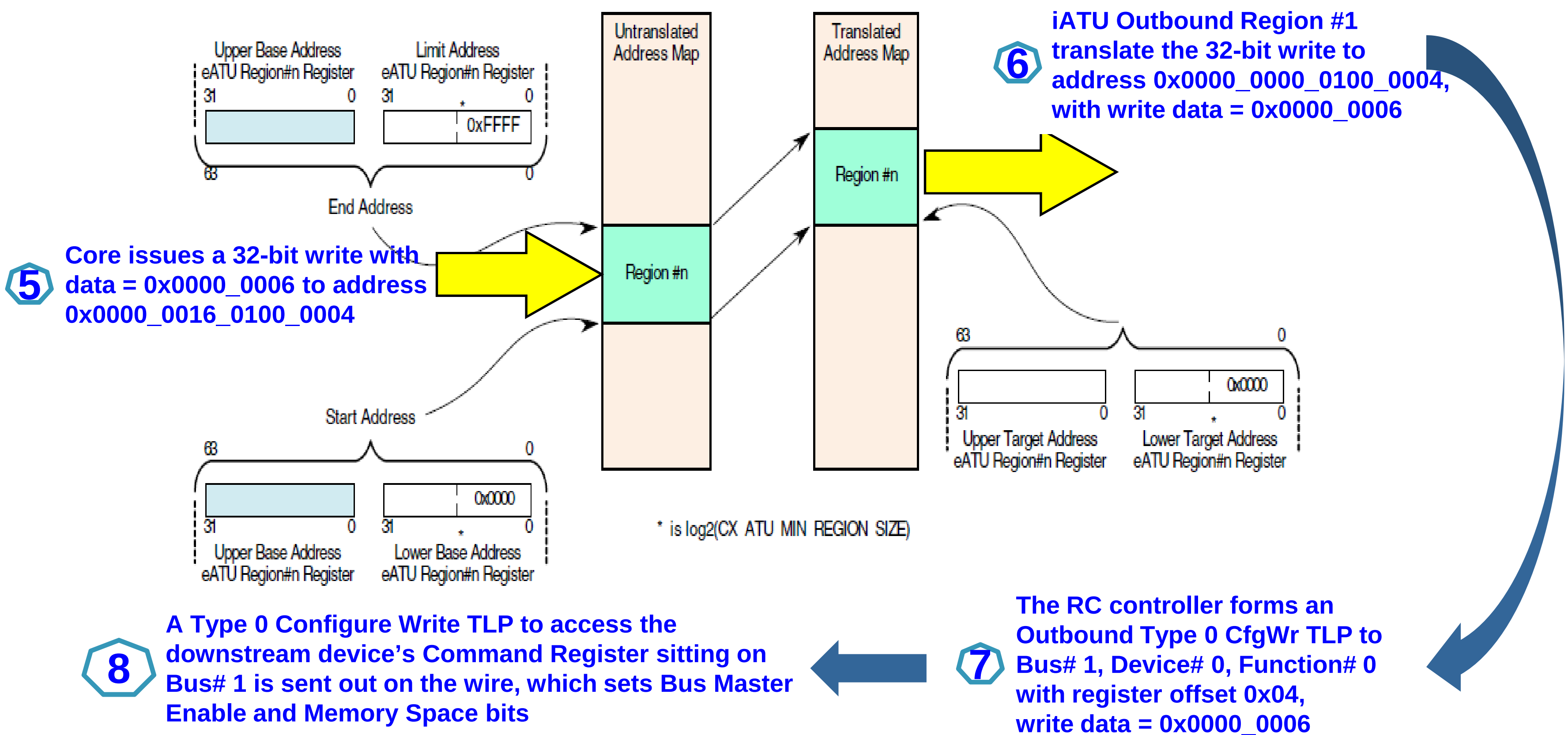
iATU Programming Example #1 – RC, Outbound CFG (cont.)

- PEX4 RC Outbound CfgRd0 TLP Generation Example
  - Type 0 Configure Read TLP to the Device ID & Vendor ID Register of the downstream device sitting on Bus# 1



iATU Programming Example #1 – RC, Outbound CFG (cont.)

- PEX4 RC Outbound CfgWr0 TLP Generation Example
  - Type 0 Configure Write TLP to set the Command Register [Bus Master Enable, Memory Space] bits of the downstream device sitting on Bus# 1



iATU Programming Example #2 – RC, Outbound MRd

- What do we need to generate an outbound 4DW Memory Read TLP?
  - DW0: → Need to program the Type only!
    - **Fmt** = **00b** for 32-bit MRd; **01b** for 64-bit MRd → **Rd** is derived from the transaction type at AXI interface
    - Program the "**Type**" bit field in the IATU_REGION_CTRL_1_OFF_OUTBOUND_0 register: **00000b for Mem.**
  - DW1: → No need to program! Controller hardware fills in the following:
    - **Requester ID** = RC's Bus# : Device# : Function# = **0x00**
    - **Last DW Byte Enable** and **First DW Byte Enable**: determined by internal bus master's transaction address
  - DW2: → Completer's upper 32-bit address!
    - **Completer's upper 32-bit address** is derived from the iATU region's translated Target Address bits [63:32]
  - DW3: → Completer's lower 32-bit address!
    - **Completer's lower 32-bit address** is derived from the iATU region's translated Target Address bits [31:0]

	+0 (bits 7 6 5 4 3 2 1 0)	+1 (bits 7 6 5 4 3 2 1 0)	+2 (bits 7 6 5 4 3 2 1 0)	+3 (bits 7 6 5 4 3 2 1 0)	
Byte 0	R (7), Fmt (6:5), Type (4:0)	R (7), TC (6:4), R (3:0)	TD (7), EP (6), Attr (5:4), R (3:2), Length (1:0)	Length (7:0)	DW0
Byte 4	Requester ID		Taq	Last DW BE (7:4), 1st DW BE (3:0)	DW1
Byte 8	Address [63:32]				DW2
Byte 12	Address [31:2]			R (1:0)	DW3

**4 DW Configure TLP Header Shown*

iATU Programming Example #2 – RC, Outbound MRd (cont.)

- iATU register programming procedure – using PEX4 as example:

1) **Set up the Index Register**
   - ❑ Write 0x0000_0002 to Index Register at offset 0x900
     - ❖ Note 1: This sets REGION_DIR = 0 and REGION_INDEX = 010 → use Region# 2 as outbound
     - ❖ Note 2: From now on until the next write to the Index Register, all the iATU registers touched between 0x904 and 0x920) are the "_OUTBOUND_" copies of the registers.

2) **Set up the Region Base and Limit Address Registers (watch: max 40 bits)**
   - ❑ Write 0x0000_0000 to Region# 2's Lower Base Address Register at offset 0x90C
   - ❑ Write 0x0000_0017 to Region# 2's Upper Base Address Register at offset 0x910
   - ❑ Write 0x0FFF_FFFF to Region# 2's Limit Address Register at offset 0x914
     - ❖ Note: This configures the Region# 2 as an 256-Mbyte outbound region with the following base address:

       0x0000_0017_0000_0000 – 0x0000_0017_0FFF_FFFF (valid internal system address for PEX4)

3) **Set up the Region Target Address Registers**
   - ❑ Write 0x0000_0000 to Region# 2's Lower Target Address Register at offset 0x918
   - ❑ Write 0x0000_0001 to Region# 2's Upper Target Address Register at offset 0x91C
     - ❖ Note: This configures the 256-Mbyte outbound Region# 2's target address as

       0x0000_0001_0000_0000 – 0x0000_0001_0FFF_FFFF (256 Mbyte right above the bottom 4 Gbyte)

iATU Programming Example #2 – RC, Outbound MRd (cont.)

- iATU register programming procedure – using PEX4 as example (continue):

  4) **Set up the Region Control 1 Register**

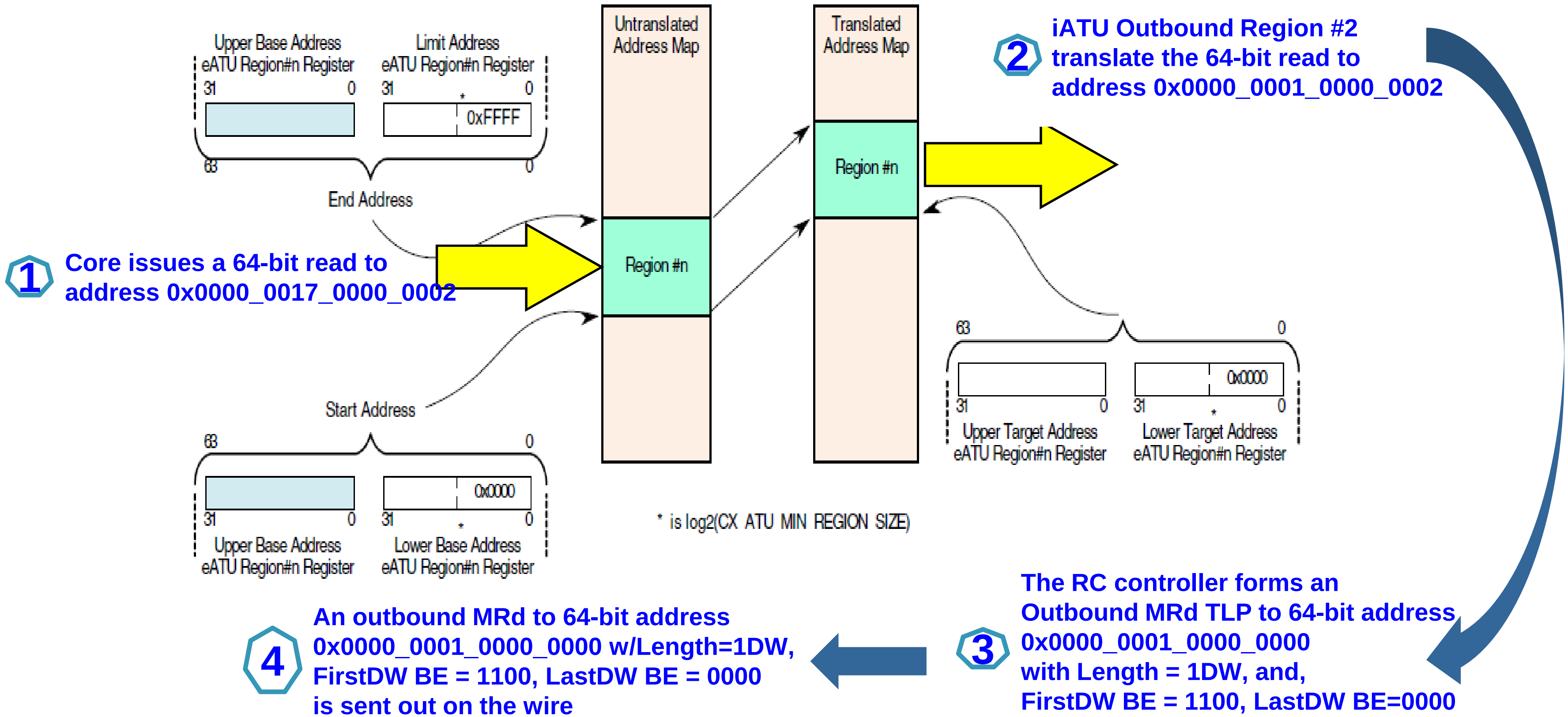
     - ❑ Write 0x0000_0000 to Region# 2's Control 1 Register at offset 0x904
       - ❖ Note: This sets the TYPE = 00000b → Outbound Region# 2 will be used to generate Memory Rd or Wr TLP

  5) **Enable the Region# 2 by writing to Region Control 2 Register**

     - ❑ Write 0x8000_0000 to Region# 2's Control 2 Register at offset 0x908

iATU Programming Example #2 – RC, Outbound MRd (cont.)

- PEX4 RC Outbound 64-bit MRd TLP Generation Example
  - Outbound MRd TLP to read 2 bytes at PCIe 64-bit address 0x0000_0001_0000_0002



iATU Programming Example #3 – RC, Inbound MRd

- How do we handle an inbound 4DW Memory Read TLP received from the wire?
  - DW0: → Need to program the Type only for the inbound region
    - **Fmt** = **00b** for 32-bit MRd; **01b** for 64-bit MRd → **Rd** is derived from the inbound MRd TLP
    - Program the "**Type**" bit field in the IATU_REGION_CTRL_1_OFF_INBOUND_0 register: **00000b for Mem.**
  - DW1: → No need to program! Controller hardware fills in the following:
    - **Requester ID** = RC's Bus# : Device# : Function# = **This is now the downstream device's B#:D#:F#**
    - **Last DW Byte Enable** and **First DW Byte Enable**: first and last DW byte offset address
  - DW2: → Completer's (LS2 RC's) upper 32-bit address!
    - **Completer's upper 32-bit address** will be compared/matched with iATU region's Base Address bits [63:32]
  - DW3: → Completer's (LS2 RC's) lower 32-bit address!
    - **Completer's lower 32-bit address** will be compared/matched with iATU region's Base Address bits [31:0]

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td><td></td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>Byte 0</td><td>R</td><td colspan="2">Fmt</td><td colspan="5">Type</td><td>R</td><td colspan="3">TC</td><td colspan="4">R</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td colspan="2">R</td><td colspan="10">Length</td><td>DW0</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="4">Last DW BE</td><td colspan="4">1st DW BE</td><td>DW1</td></tr>
<tr><td>Byte 8</td><td colspan="32">Address [63:32]</td><td>DW2</td></tr>
<tr><td>Byte 12</td><td colspan="30">Address [31:2]</td><td colspan="2">R</td><td>DW3</td></tr>
</table>

**4 DW Configure TLP Header Shown*

iATU Programming Example #3 – RC, Inbound MRd (cont.)

- iATU register programming procedure – using PEX4 as example:

1) **Set up the Index Register**
   - ❑ Write 0x8000_0001 to Index Register at offset 0x900
     - ❖ Note 1: This sets REGION_DIR = 1 and REGION_INDEX = 001 → use Region# 1 as inbound
     - ❖ Note 2: From now on until the next write to the Index Register, all the iATU registers touched between 0x904 and 0x920) are the "_INBOUND_" copies of the registers.

2) **Set up the Region Base and Limit Address Registers (external side now)**
   - ❑ Write 0x0000_0000 to Region# 1's Lower Base Address Register at offset 0x90C
   - ❑ Write 0xA000_0000 to Region# 1's Upper Base Address Register at offset 0x910
   - ❑ Write 0x0FFF_FFFF to Region# 1's Limit Address Register at offset 0x914
     - ❖ Note: This configures the Region# 1 as an 256-Mbyte inbound region with the following base address at PCIe side: 0xA000_0000_0000_0000 – 0xA000_0000_0FFF_FFFF

3) **Set up the Region Target Address Registers (internal side now)**
   - ❑ Write 0x8000_0000 to Region# 1's Lower Target Address Register at offset 0x918
   - ❑ Write 0x0000_0000 to Region# 1's Upper Target Address Register at offset 0x91C
     - ❖ Note: This configures the 256-Mbyte inbound Region# 1's target address at DDR as:

       0x0000_0000_8000_0000 – 0x0000_0000_8FFF_FFFF (256 Mbyte above 0x8000_0000 in DDR)

iATU Programming Example #3 – RC, Inbound MRd (cont.)

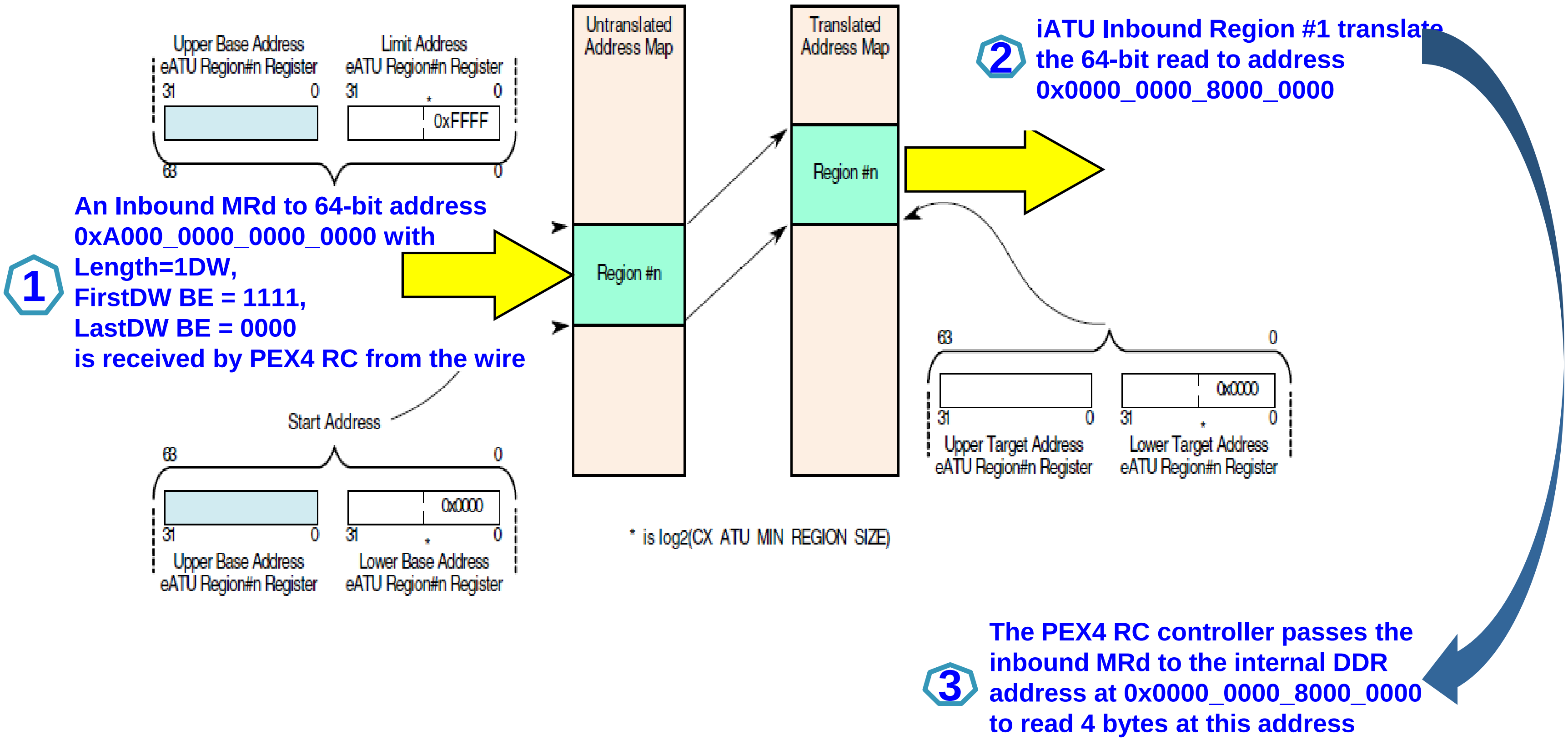
- iATU register programming procedure – using PEX4 as example (continue):

  4) **Set up the Region Control 1 Register**
     - ❑ Write 0x0000_0000 to Region# 1's Control 1 Register at offset 0x904
       - ❖ Note: This sets the TYPE = 00000b → Inbound Region# 1 will be used to generate Memory Rd or Wr TLP

  5) **Enable the Region# 1 by writing to Region Control 2 Register**
     - ❑ Write 0x8000_0000 to Region# 1's Control 2 Register at offset 0x908

iATU Programming Example #3 – RC, Inbound MRd (cont.)

- Inbound 64-bit MRd TLP Handling by PEX4 RC Example
  - Inbound MRd TLP to read 4 bytes at internal DDR address 0x0000_0000_8000_0000



iATU Programming Example #4 – Non-SR-IOV EP, Inbound MRd

- How do we handle an inbound 4DW Memory Read TLP received from the wire?
  - DW0: → Need to program the Type for the inbound region
    - **Fmt** = **00b** for 32-bit MRd; **01b** for 64-bit MRd → **Rd** is derived from the inbound MRd TLP
    - Program the "**Type**" bit field in the IATU_REGION_CTRL_1_OFF_INBOUND_0 register: **00000b for Mem.**
  - DW1: → No need to program! Controller hardware fills in the following:
    - **Requester ID** = RC's Bus# : Device# : Function# = **This is now the upstream remote RC's B#:D#:F#**
    - **Last DW Byte Enable** and **First DW Byte Enable**: first and last DW byte offset address
  - DW2: → Completer's (LS2 EP's) upper 32-bit address!
    - **Completer's upper 32-bit address** will be compared/matched with iATU region's Upper BAR register
  - DW3: → Completer's (LS2 EP's) lower 32-bit address!
    - **Completer's lower 32-bit address** will be compared/matched with iATU region's BAR register

<table>
<tr><td></td><td colspan="8">+0</td><td colspan="8">+1</td><td colspan="8">+2</td><td colspan="8">+3</td><td></td></tr>
<tr><td></td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>Byte 0</td><td>R</td><td colspan="2">Fmt</td><td colspan="5">Type</td><td>R</td><td colspan="3">TC</td><td colspan="4">R</td><td>TD</td><td>EP</td><td colspan="2">Attr</td><td colspan="2">R</td><td colspan="10">Length</td><td>DW0</td></tr>
<tr><td>Byte 4</td><td colspan="16">Requester ID</td><td colspan="8">Tag</td><td colspan="4">Last DW BE</td><td colspan="4">1st DW BE</td><td>DW1</td></tr>
<tr><td>Byte 8</td><td colspan="32">Address [63:32]</td><td>DW2</td></tr>
<tr><td>Byte 12</td><td colspan="30">Address [31:2]</td><td colspan="2">R</td><td>DW3</td></tr>
</table>

**4 DW Configure TLP Header Shown*

iATU Programming Example #4 – Non-SR-IOV EP, Inbound MRd (cont.)

- iATU register programming procedure – using PEX1 as example:

1) **Set up the Index Register**
   - ❑ Write 0x8000_0001 to Index Register at offset 0x900
     - ❖ Note 1: This sets REGION_DIR = 1 and REGION_INDEX = 001 → use Region# 1 as inbound
     - ❖ Note 2: From now on until the next write to the Index Register, all the iATU registers touched between 0x904 and 0x920) are the "_INBOUND_" copies of the registers.

2) **Set up the Region Target Address Registers (internal side now)**
   - ❑ Write 0x8000_0000 to Region# 1's Lower Target Address Register at offset 0x918
   - ❑ Write 0x0000_0000 to Region# 1's Upper Target Address Register at offset 0x91C
     - ❖ Note: This configures the 256-Mbyte inbound Region# 1's target address at DDR as:

       0x0000_0000_8000_0000 – 0x0000_0000_8FFF_FFFF (256 Mbyte above 0x8000_0000 in DDR)

3) **No need to set up Region Base and Limit Address Registers (external side)**
     - ❖ Note: Don't have to configure them since we will be using BAR Match Mode instead of Address Match Mode for EP's inbound memory access

iATU Programming Example #4 – Non-SR-IOV EP, Inbound MRd (cont.)

- iATU register programming procedure – using PEX1 as example (continue):

4) **Set up the Region Control 1 Register**
   - ❑ Write 0x0000_0000 to Region# 1's Control 1 Register at offset 0x904
     - ❖ Note: This sets the TYPE = 00000b → Inbound Region# 1 will be used to generate Memory Rd or Wr TLP

5) **Set up and Enable the Region# 1 by writing to Region Control 2 Register**
   - ❑ Write 0xC000_0200 to Region# 1's Control 2 Register at offset 0x908
     - ❖ Note: This configures the MATCH_MODE = 1b, BAR_NUM = 010b → Inbound Region# 1 is enabled and will be using BAR Match Mode (BAR2 at PEX1 EP's Type 0 Header offset 0x18) for inbound MRd and MWr access

iATU Programming Example #4 – Non-SR-IOV EP, Inbound MRd (cont.)

- iATU register programming procedure – using PEX1 as example (continue):
  - *Note that the Step# 6 and 7 below are only required when the BAR is disabled and re-enabled again. By default, all BARs are enabled. The Prefetchable and Memory Space bits of the BAR2 and BAR4 are also set by default.*

6) **Set up the PEX1 EP's BAR2_MASK_REG**
   - ❑ Write 0x0000_0001 to PEX1 EP's BAR2_MASK_REG at offset 0x1018
     - ❖ Note 1: Uses the shadow mask register for BAR2 to enable the BAR2. Just enable it first. Don't set the mask yet.
     - ❖ Note 2: The BAR2_MASK_REG shadow register defines 2 bit fields: Bit [0] = BAR enable; Bit [31:1] = BAR Mask, Write-Only, writing 1 to a bit means it's part of the size.

7) **Set up the PEX1 EP's BAR2 register's Prefetchable and 64-bit Mem. Bits**
   - ❑ Write 0x0000_0001 to PEX1 EP's configure offset 0x8BC.
     - ❖ Note: This turns on the DBI_RO_WR_EN bit, in order to write to BAR2 that is Read-Only in normal mode.
   - ❑ Write 0x0000_000C to PEX1 EP's BAR2 Register at Type 0 Header offset 0x18
     - ❖ Note: This configures BAR2 as a pre-fetchable 64-bit memory BAR.
   - ❑ Write 0x0000_0000 to PEX1 EP's configure offset 0x8BC
     - ❖ Note: This turns off the DBI_RO_WR_EN bit to return BAR2 back as Read-Only register.

iATU Programming Example #4 – Non-SR-IOV EP, Inbound MRd (cont.)

- iATU register programming procedure – using PEX1 as example (continue):

8) **Set up the BAR Mask (Size) in PEX1 EP's BAR2_MASK_REG**

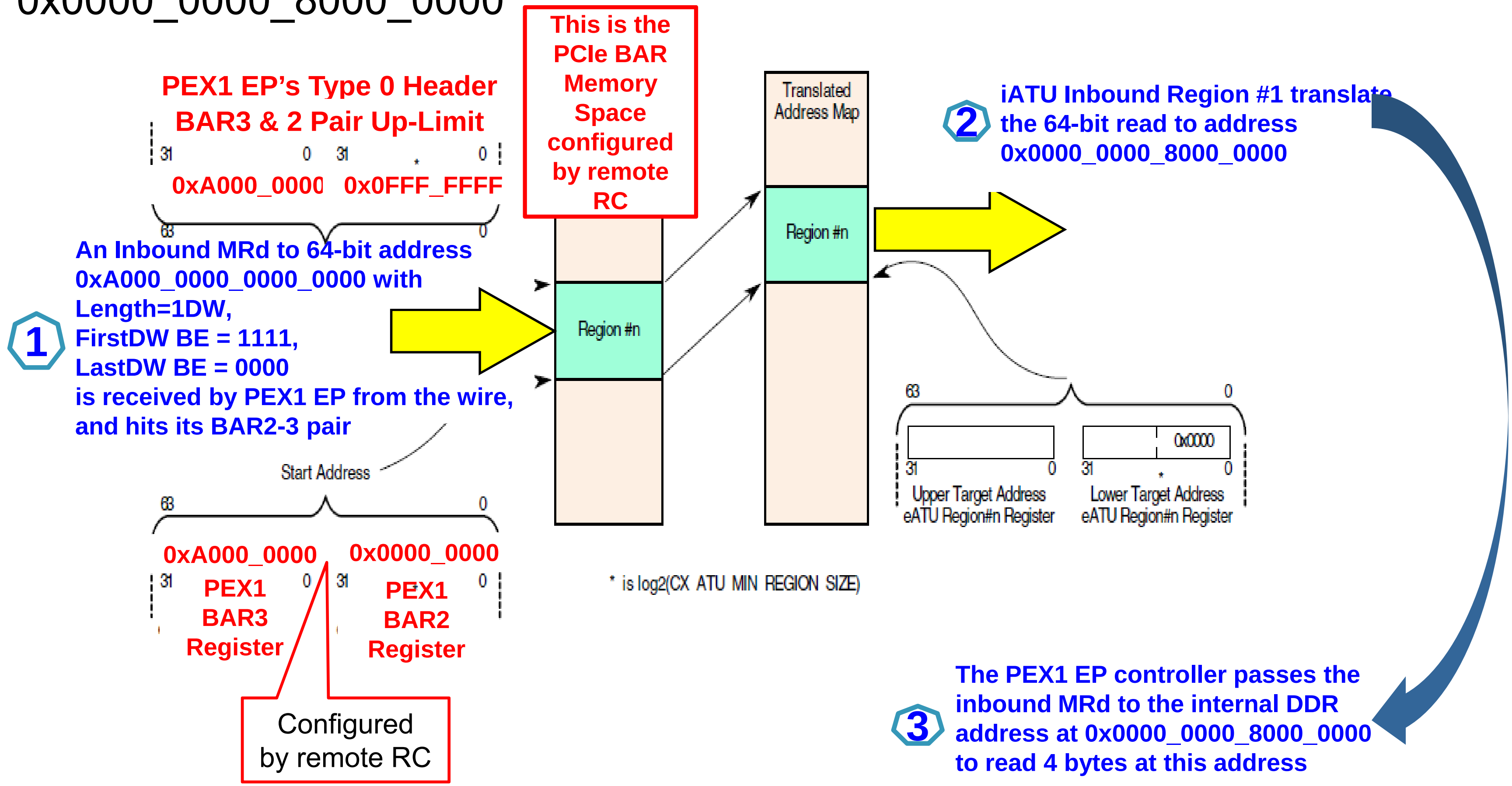
   - ❑ Write 0x0FFF_FFFF to PEX1 EP's BAR2_MASK_REG at offset 0x1018
     - ❖ Note 1: Keep enable bit turned on at the shadow mask register for BAR2.
     - ❖ Note 2: The BAR2_MASK_REG shadow register's BAR Mask Bits are at [31:1]. The above setting reflects a size mask of 0x0FFF_FFFF, which is 256 MB. The bits [31:28] = 0x0, which will be writeable by the remote RC to configure a BAR2's address later.

9) **Set up the BAR Mask (Size) in PEX1 EP's BAR3_MASK_REG**

   - ❑ Write 0x0000_0000 to PEX1 EP's BAR3_MASK_REG at offset 0x101C
     - ❖ Note: This makes all the bits of BAR3 to be writable by the remote RC later to configure the BAR3 at PEX1 EP's Type 0 Header offset 0x1C. BAR3 becomes the upper 32-bit BAR address of this 64-bit BAR2 & BAR3 pair.

iATU Programming Example #4 – Non-SR-IOV EP, Inbound MRd (cont.)

- Inbound 64-bit MRd TLP Handling by PEX1 EP Example
  - Inbound MRd TLP to read 4 bytes at internal DDR address 0x0000_0000_8000_0000



Caveat for iATU Configuration

- Always start with programming the Index Register first and remember any further access to the rest of iATU registers is now tied to the copy of the "configured region direction and number"

- Understand the correct meaning of the Base and Target Registers
  - Base refers to the iATU side before address translation
  - Target refers to the iATU side after the address translation

- It's better to program the internal side registers (could be either Base or Target, depending on outbound or inbound).

- Region Control 2 Register [MATCH_MODE] usage guideline for inbound memory transactions
  - For RC mode, set MATCH_MODE = Address Match Mode
  - For EP mode, set MATCH_MODE = BAR Match Mode, then follow the Example #4 to configure the BARn_MASK_REG (shadow) and BARn registers.

Summary

Summary

- PCI Express is a complicated high-speed serial data transmission protocol with 3 layers defined: Transaction Layer, Data Link Layer, and Physical Layer.
- A good understanding of the PCI and PCIe specification will speed up the development and debug process.
- Plan ahead with QCS to ensure the correct RCW configuration is used for SerDes Lanes & related PLLs, and PCI Express required reference clock frequency and maximum speed desired.
- The LS2085A integrated PCI Express controllers features a new internal programming model that needs to be followed.

Useful References

- Books:
  - PCI System Architecture, Fourth Edition, Tom Shanley, Don Anderson, MindShare, Inc., 2002
  - PCI Express System Architecture, Ravi Budruk, Don Anderson, Tom Shanley, MindShare, Inc., 2006
- Freescale AppNotes:
  - AN4311, SerDes Reference Clock Interfacing and HSSI Measurements Recommendations
- PCI-SIG Specifications:
  - PCI Local Bus Specification, Revision 2.3, March 29, 2002
  - PCI Bus Power Management Interface Specification, Revision 1.2, March 3, 2004
  - PCI Express Base Specification, Revision 1.0a, April 15, 2003
  - PCI Express Base Specification, Revision 1.1, March 28, 2005
  - PCI Express Base Specification, Revision 2.0, December 20, 2006
  - PCI Express Base Specification, Revision 2.1, March 04, 2009
  - PCI Express Base Specification, Revision 3.0, November 10, 2010
  - PCI Express Card Electromechanical Specification, Revision 1.1, March 28, 2005
  - PCI Express Card Electromechanical Specification, Revision 2.0, April 11, 2007
  - PCI Express Card Electromechanical Specification, Revision 3.0, July 21, 2013
- Tools
  - QCVS tool
  - http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=PE_QORIQ_SUITE&fsrch=1&sr=1&pageNum=1

www.Freescale.com