FTF 2016
TECHNOLOGY FORUM

QorIQ LS2088A PCIE EP SR-IOV PROGRAMMING FLOW

FTF-DES-N1852

RICHARD NIE
PRINCIPAL ENGINEER
FTF-DES-N1852
MAY 18, 2016

AGENDA

- I/O Virtualization and SR-IOV Background Introduction
- Introduction to Alternative Routing Interpretation (ARI)
- SR-IOV Extended Capability and Discovery
- Reset Mechanism and Function Level Reset (FLR)
- SR-IOV Configuration
- SR-IOV EP Features in QorIQ LS2088A Processor
- QorIQ LS2088A Processor PEX3 SR-IOV EP's iATU and BAR Local Software Programming Procedure
  - Inbound iATU Programming Procedure for PEX3 SR-IOV EP
  - Inbound PF BAR Configuration and Programming Procedure
  - Inbound VF BAR Configuration and Programming Procedure

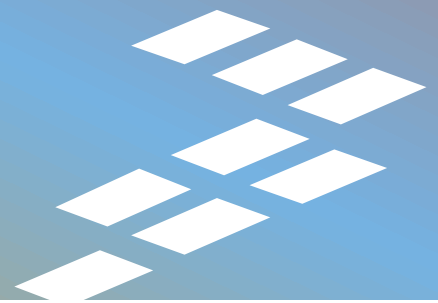
Session Introduction

- The SR-IOV provides a cost-effective way to allow different software images achieve independent data movement targeting virtual functions while sharing IO resources on a single PCIe EP device.
- This session introduces the SR-IOV related features and programming model of QorIQ LS2088A Processor’s PCI Express Gen3 Endpoint (EP).

Objectives

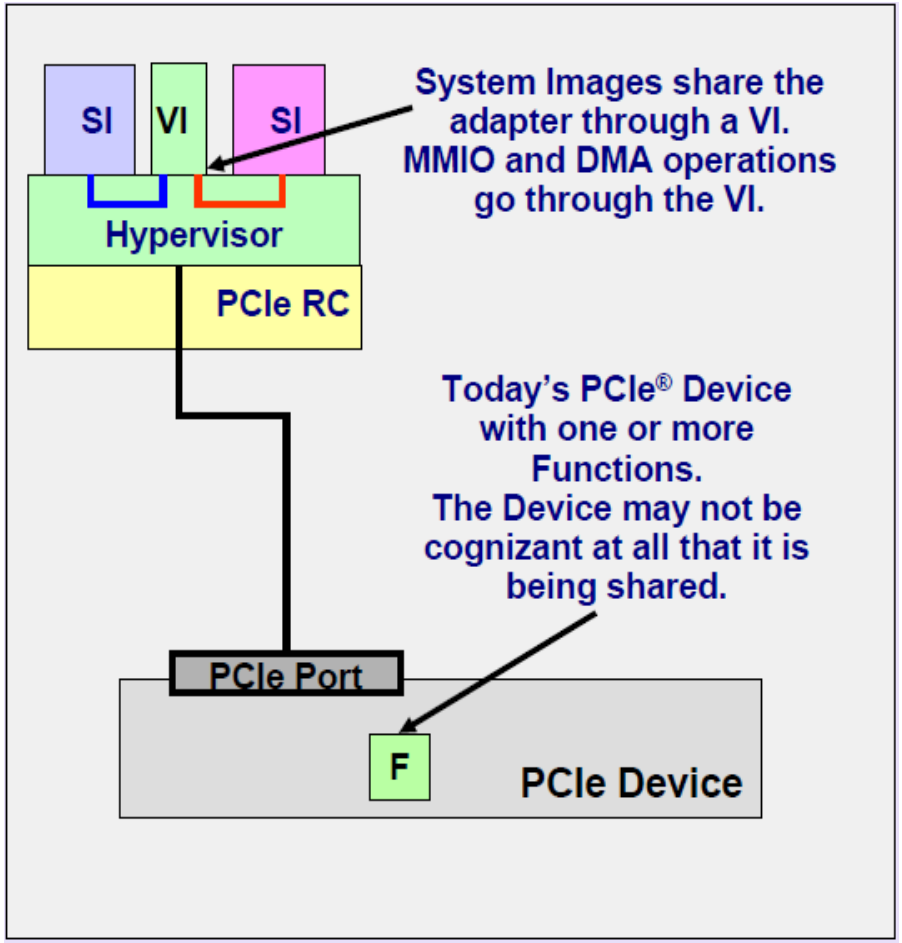
- After completing this training, you will be able to:
  - Understand what I/O Virtualization is and the SR-IOV architecture
  - Understand the terms, definitions, and role of all key SR-IOV components
  - Understand how the Alternative Routing Interpretation (ARI) and Routing ID (RID) are used to solve the SR-IOV VF configuration and routing issue when more than 8 functions are introduced in an SR-IOV EP.
  - Learn how to discover and configure the SR-IOV capability structure.
  - Learn QorIQ LS2088A Processor's SR-IOV feature implementation.

I/O Virtualization and SR-IOV Background Introduction

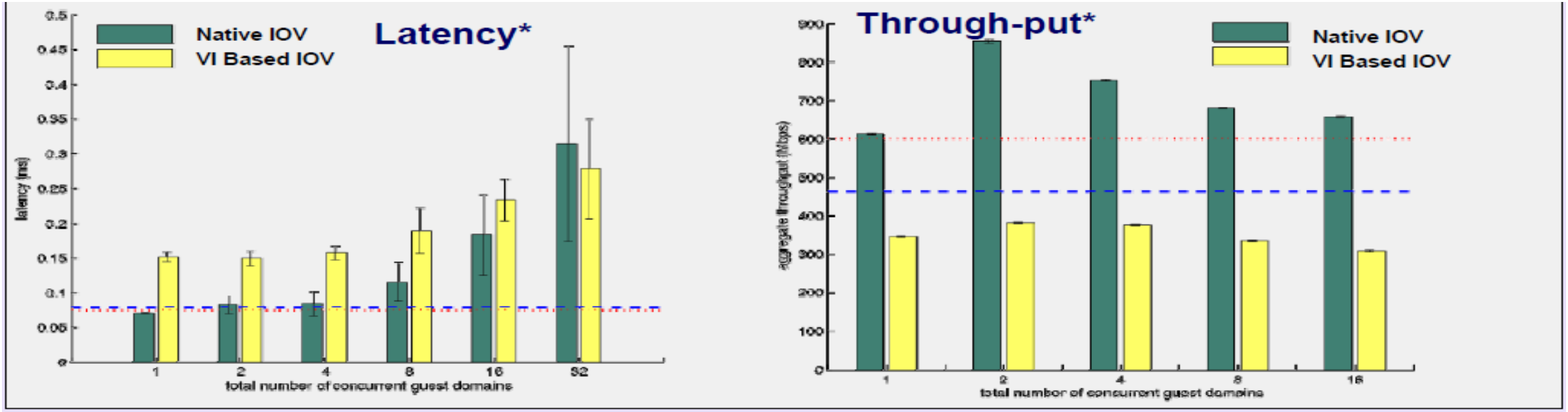


Traditional IOV (I/O Virtualization)

- Traditional approach w/o SR-IOV
  - 1 or more System Images (SI) share the PCI device via VI
  - Virtualization Intermediaries (VI) are used to safely share I/Os
    - VI takes sole ownership of the underlying hardware
      - VI abstracts the hardware to present to each SI with its own virtual system
  - Pros:
    - No new HW (like Virtualization Enablers) needed in either RC or PCIe Device
  - Cons:
    - Performance overhead due to VI's interception to each I/O or DMA transaction



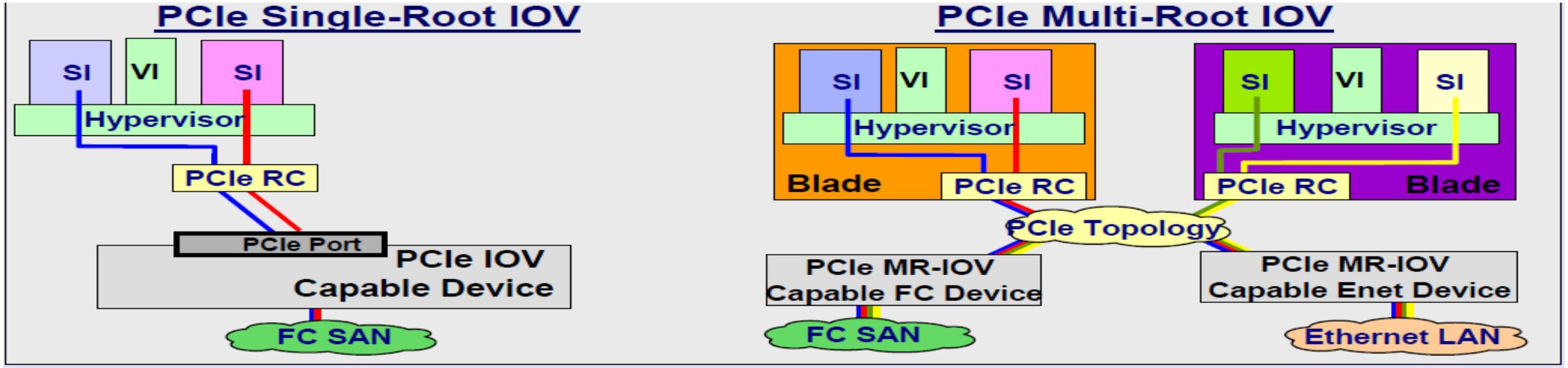
Performance of Today's IOV



❖ Source: Self-Virtualization I/O: High Performance, Scalable I/O Virtualization in Multi-core Systems; R. Himanshu, I. Ganev, K. Schwan – Georgia Tech and J. Xenidis - IBM

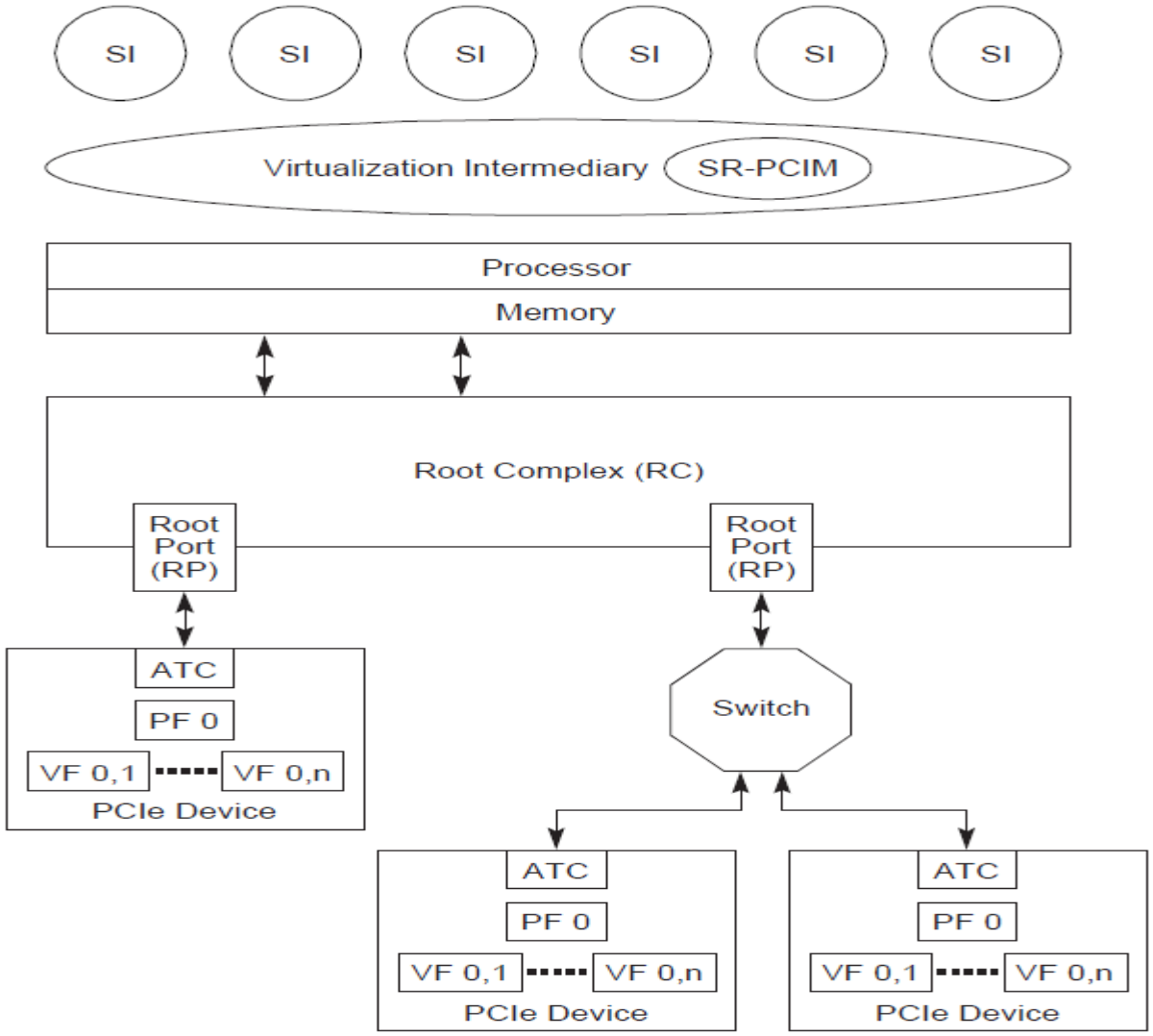
- VI based IOV adds path latency on every I/O transaction
- Native IOV significantly improves performance

PCI-SIG's IOV Overview



- PCI-SIG standardized mechanisms that enable PCIe devices to be directly shared, with no run-time overheads:
  - Single-Root IOV (SR-IOV): Direct sharing between SIs on a single system
  - Multi-Root IOV (MR-IOV): Direct sharing between SIs on multiple systems
- PCI-SIG IOV Specification only covers device's "north-side"

Generic Platform Configuration w/ SR-IOV and IOV Enablers



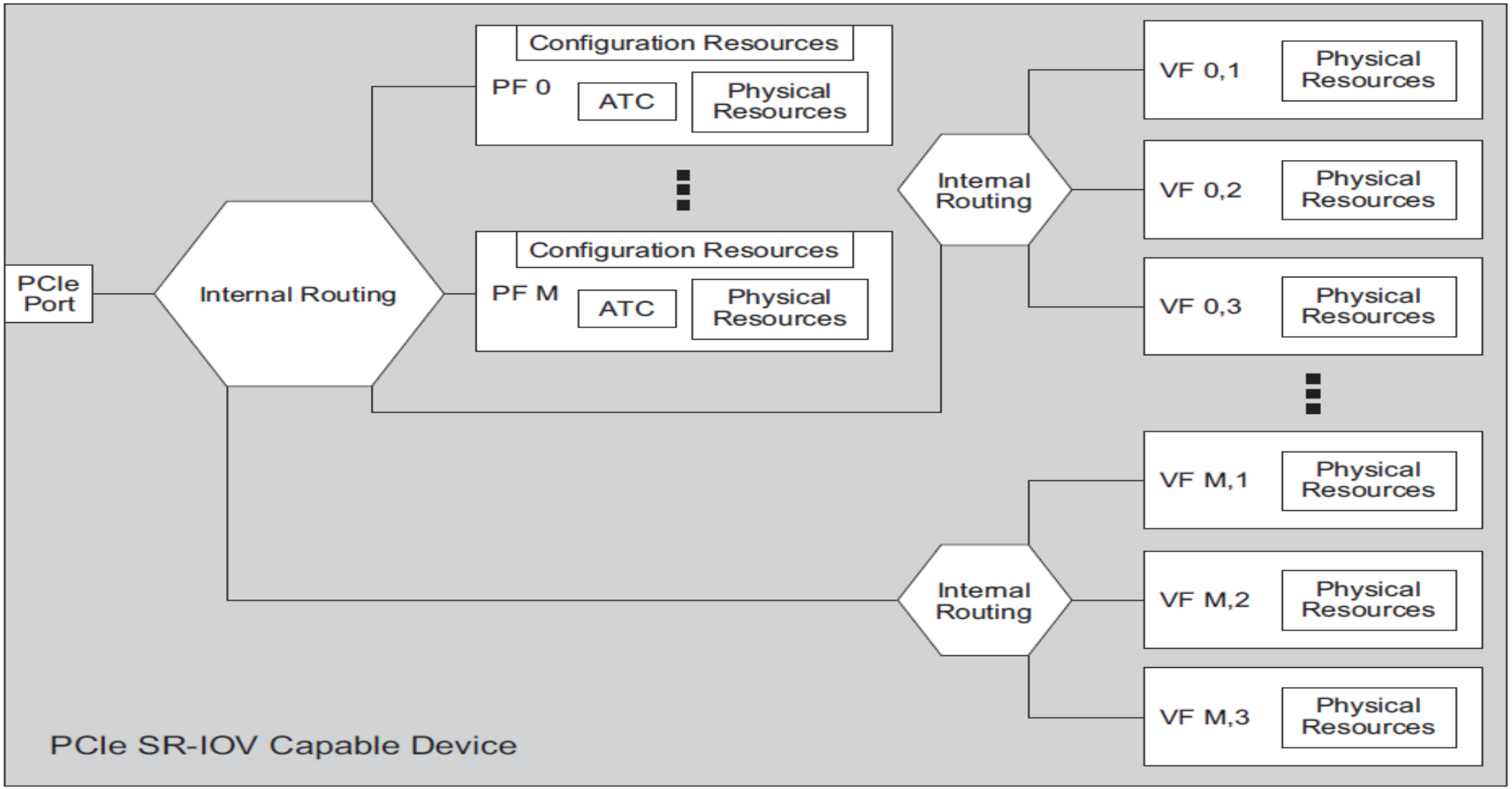
What is I/O Virtualization?

- From EP Card point of view:
  - One physical device looks like multiple devices
    - Multiple views of the same physical device
      - Example: multiple views of SAS controller or NIC
  - Virtual devices appear completely independent
    - Each virtual device is assigned its own PCIe addresses
    - Each virtual device's BARs can be separated by the system page size
    - Each virtual device appears as a separate PCIe function in configuration space
    - May have different settings for various configuration registers
    - Need to keep cross-"device" traffic isolated
    - EP may export a mix of "base" functions and IOV-enabled functions in SR-IOV

What is I/O Virtualization? (continued)

- From a system point of view:
  - System Image (SI) is a real or virtual system of CPU(s), Memory, O/S, I/O, etc.
    - SI is just another name of an OS running on top of a Virtualization Manager (e.g., an instance of Linux running in a host system)
    - Multiple SIs may run on one or more sets of hardware
      - Example 1: VMWare running Win32 & Linux on a single CPU
      - Example 2: Blade server running multiple-OS each on a single blade
  - Each System Image (SI) sees its own PCIe hierarchy
    - Even if NO end devices are actually shared
    - Only its portion of shared end devices

Example SR-IOV-Capable Device with Multi-PF



I/O Virtualization Terms and Definitions – (1 of 4 Slides)

- IOV – I/O Virtualization:
  - A hardware method of exporting multiple views of the same device

- SR-IOV – Single Root I/O Virtualization
  - PCI-SIG defined methods for exporting multiple views of the same device in a traditional PCIe base fabric
  - No PCIe protocol changes. Works with existing PCIe fabrics
  - May be ignored by SR-IOV unaware software/firmware
    - In this case, it works just like base PCIe

I/O Virtualization Terms and Definitions – (2 of 4 Slides)

- MR-IOV – Multi-Root I/O Virtualization
  - Just for reference only. NXP currently does not support MR-IOV
  - PCI-SIG defined methods for exporting and managing multiple views of the same device in a PCIe base fabric that can be connected to more than one root port
  - From each root port's perspective, the fabric appears to be a traditional PCIe base fabric
  - No change to Root Complex (RC). No change to non-MR EPs
  - Changes to MR aware switches and EPs to support multiple *Virtual Hierarchies*
  - PCIe protocol changes (TLP Header) and new automatic training sequences for MR aware switches and EPs

I/O Virtualization Terms and Definitions – (3 of 4 Slides)

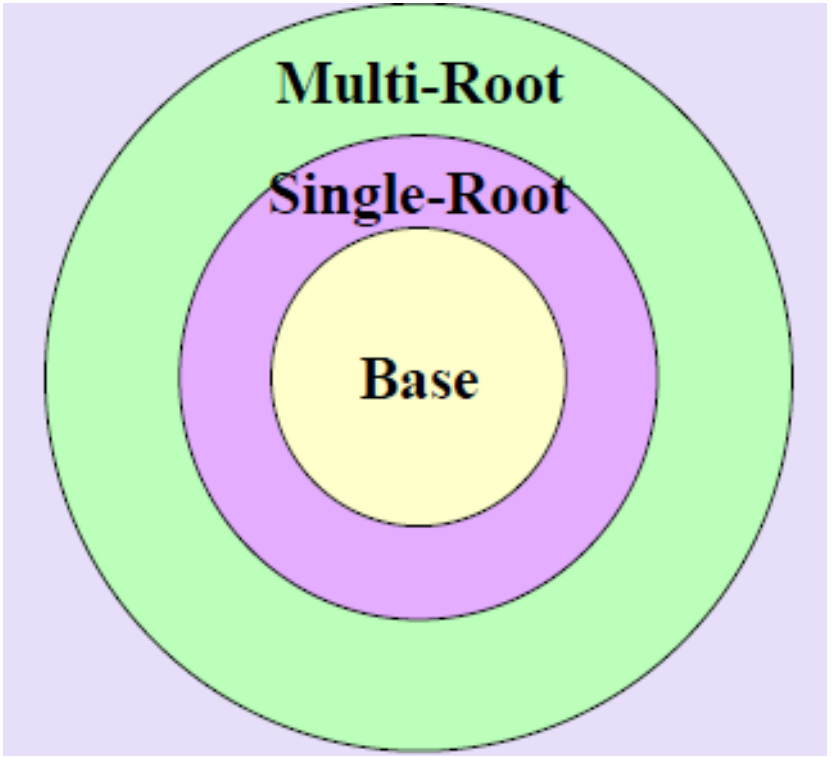
- Function
  - PCIe base-defined function as with base PCI and PCIe
- Virtual Function (VF)
  - A virtual view of a physical device
    - Looks like a Function: RID, configuration space, memory space, etc.
    - Some common configuration space fields are shared among VFs, which are controlled via a single PF
- Physical Function (PF)
  - A function that includes the SR-IOV capability
    - An anchor for creating *Virtual Functions* and reporting errors and events that cannot be attribute to a single *virtual function*
- Base Function (BF)
  - A function that includes MR-IOV capability
    - An anchor for creating/managing *Virtual Hierarchies* and PFs within VHs and reporting errors and events that cannot be attributed to a single VH

I/O Virtualization Terms and Definitions – (4 of 4 Slides)

- System Image (SI)
  - The OS running in a domain (e.g., Linux in a host system)
- Virtual Intermediary (VI)
  - Hypervisor that provides physical resource protection between SIs running on a system
- Single Root PCI Manager (SR-PCIM)
  - The SR-IOV specification's *name* for the configuration, management and protection software/firmware for SR-IOV implementations
- Multi Root PCI Manager (MR-PCIM)
  - The MR-IOV specification's *name* for the configuration and management software/firmware for MR-IOV implementations

I/O Virtualization Implementation and Interoperability

- Attachment of existing PCIe base components
  - RC, Switches, EPs, and Bridges
- A solution to use a combination of existing base and IOV-aware components and maximize interoperability across them:
  - SR-IOV capabilities are a superset of the PCIe base specification
  - MR-IOV capabilities build on the SR-IOV capabilities or base capabilities
  - IOV-capable components are backwards compatible with existing software
  - SR-IOV ***does not change*** any base component like physical, data link and transaction layer
    - Optional changes are implemented with the newly defined capability structures



Role of Key SR-IOV Components [Impact to EP] – (1 of 2 Slides)

- Physical Function (PF)
  - A PCIe Function that includes the SR-IOV capability
    - The SR-IOV Capability structure is used by SR-PCIM to configure and manage PF and its VFs
- Virtual Function (VF)
  - A virtual view of a physical device with its unique Routing ID (RID)
  - A SR-IOV device needs:
    - *n* sets of configuration space to support *n* VFs
    - *n* sets of registers to support *n* VFs
    - *n* sets of internal logic to support *n* VFs (maybe)
  - VF with SR-PCIM → VFs are created/managed by SR-PCIM
    - Once created, it can be probed and accessed through the RC using normal access methods
    - The VF's *configuration resources* are restricted to be accessed by SR-PCIM (a trusted software), NOT by SI
  - VF with SI → used by SIs to access *data movement resources* on the SR-IOV EP
    - However a VF can be serially shared by different SI
      - E.g. a VF can be assigned to one SI and then reset and assigned to another SI
  - VF with PF → Each VF is associated with a single PF for SR-IOV
    - All VFs associated with a PF must be the same device type (e.g. same network device) as PF

Role of Key SR-IOV Components [Impact to EP] – (2 of 2 Slides)

- Single Root PCI Manager (SR-PCIM)
  - A software component running at PCIe RC responsible for configuration, management and protection of an IOV-enabled fabric and devices
  - Creates, manages and assigns VFs to SIs
  - To create VFs, use the SR-IOV extended capability in PFs:
    - Write to PF's NumberVFs register with Number of VFs desired in this PF
    - Write to PF's VF BAR registers
      - Memory BARs are not duplicated in the VF configure space
      - One set of BARs for all VF's memory space
    - Read from the Supported Page Size register, *choose ONE* desired value and write it to the System Page Size register
  - Handles events that cannot be associated with a single VF/SI

Configuration Space Requirements for SR-IOV

- SR-PCIM must be able to discover PFs and configure them
  - SR-IOV Extended Capability
- Each VF must have a unique Routing ID
  - Unique configuration space address to discover the VF instance
  - Some bit fields contain the read-only copies of the associated PF
  - Unique Routing ID (RID) used in interrupts, messages, R/W requests, etc.
- Compatibility with the PCIe Base
  - Retain header layout for type 0 and 1 headers
  - No need to implement all bits
  - Same configuration space requirement for read/write, routing rules, etc.
  - Minimize bits that must be implemented per VF
    - Alias bits where possible
    - Implement bits where required

Other SR-IOV Considerations – (1 of 2 Slides)

- Same PCIe base protocol

- Fits into existing PCIe hierarchies today
  - Single PCIe address spaces – partitioned and allocated above the RC
    - Using Routing ID (RID) in packets to track transactions back to the appropriate SI

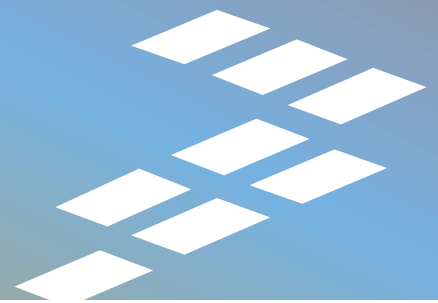
- After reset, only the PFs exist
  - PFs may be used exactly like any function today

- SR-PCIM may configure the PF's SR-IOV Capability and create VFs, assign them to SIs

Other SR-IOV Considerations – (2 of 2 Slides)

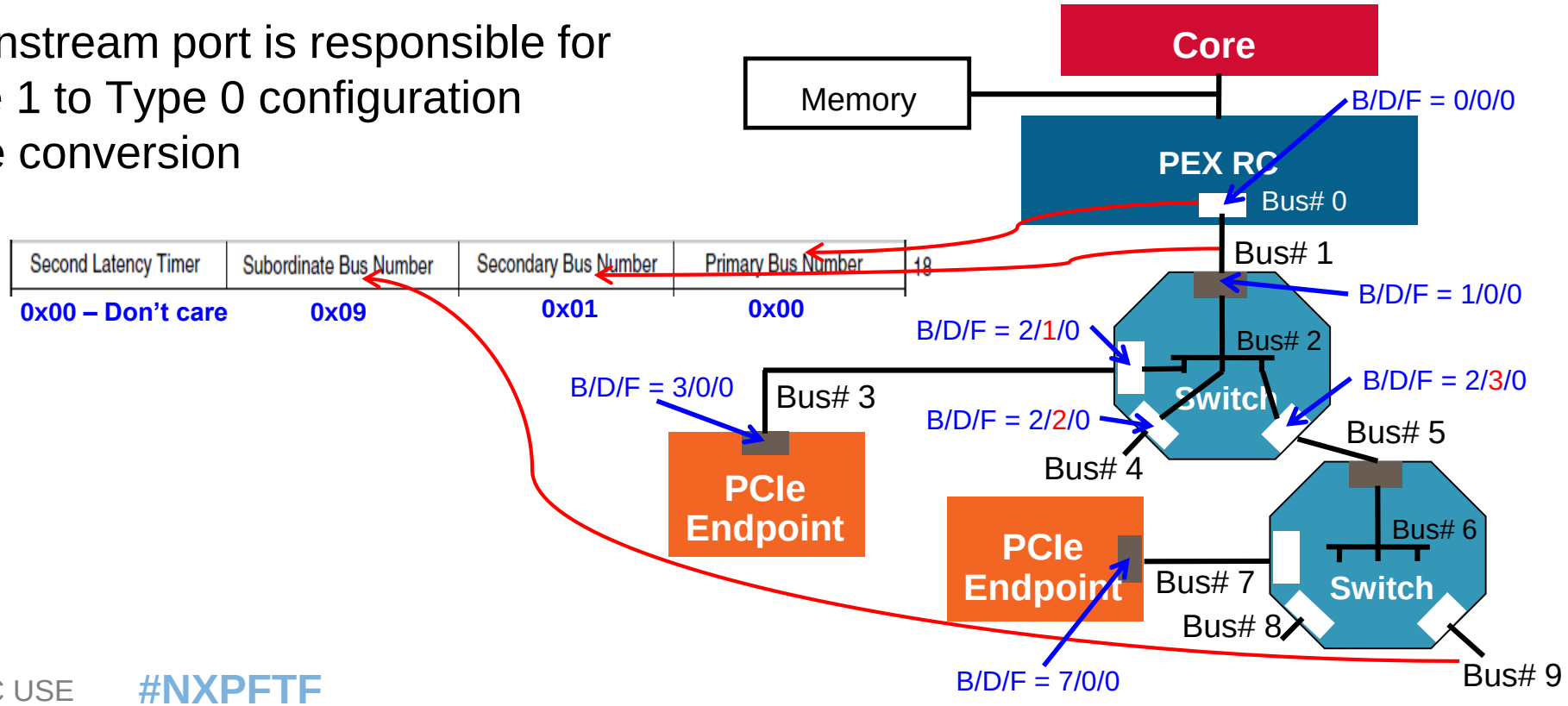
- Supports thousands of virtual views
  - Alternative Routing Interpretation (ARI) support allows up to 256 Functions using a single bus number
  - IOV-aware PFs must be on the first bus number
  - SR-IOV EPs may consume multiple bus numbers
    - Permits VF expansion for situation, where > 256 functions desired
    - Capture the RID only on Type 0 configure write transactions targeting the first bus number
    - No change to the downstream port above the SR-IOV EP
      - Configure transactions targeting function numbers > 255 are received as “Type 1” configuration transactions and are consumed by the SR-IOV EP

Introduction to Alternative Routing Interpretation (ARI)



Bus# / Device# / Function# (B/D/F) Defined by Base Spec

- This example is based on traditional B/D/F routing without SR-IOV
  - The RC's Type 1 Header's Bus Number Register at offset 0x18 is configured as 0x0009_0100
  - Notice that EP's Device# is always 0 in all B/D/F assignment, same for the switch's upstream port
  - Downstream port is responsible for Type 1 to Type 0 configuration cycle conversion



Re-define Routing ID Based on ARI

Table 2-7: Header Field Locations for non-ARI ID Routing

Field	Header Location
Bus Number[7:0]	Bits 7:0 of Byte 8
Device Number[4:0]	Bits 7:3 of Byte 9
Function Number[2:0]	Bits 2:0 of Byte 9

Table 2-8: Header Field Locations for ARI ID Routing

Field	Header Location
Bus Number[7:0]	Bits 7:0 of Byte 8
Function Number[7:0]	Bits 7:0 of Byte 9

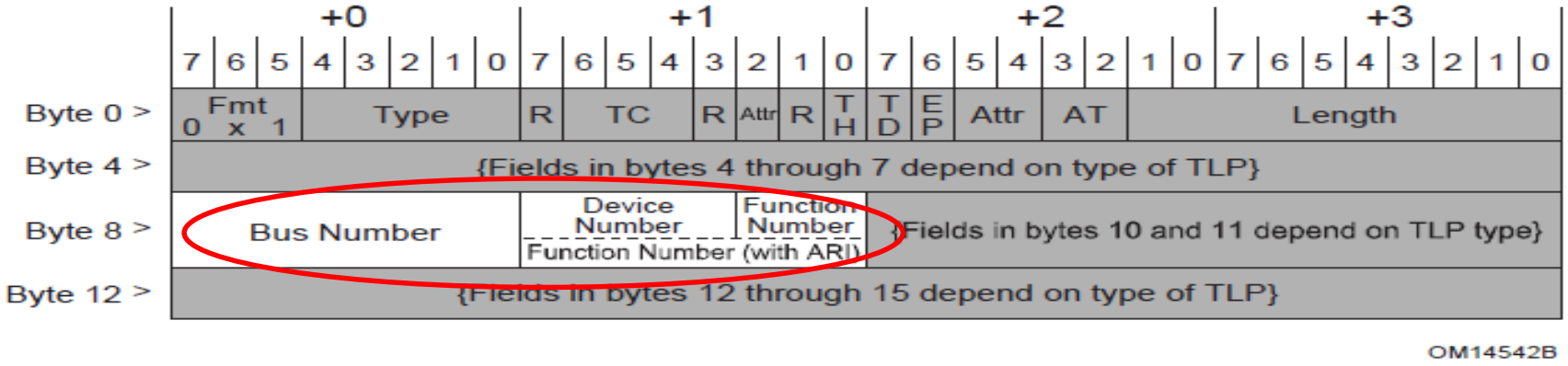


Figure 2-9: ID Routing with 4 DW Header

TLP Header's Routing ID Implication

- Without ARI (traditional Routing ID, Requester ID, Completer ID)
  - Routing ID = {Bus Number, Device Number, Function Number}
  - Bus Number [7:0] (8 bits) ➔ up to 256 Bus Numbers per link
  - Device Number [4:0] (5 bits) ➔ up to 32 Device Numbers per bus
    - However non-zero device number is usually used by switches
  - Function Number [2:0] (3 bits) ➔ up to 8 Function Numbers per device

- With ARI
  - Routing ID = {Bus Number, Function Number}
  - Bus Number [7:0] (8 bits) ➔ up to 256 Bus Numbers
  - No more device number field
  - Function Number [7:0] (8 bits) ➔ up to 256 Function Numbers
  - New problem:
    - None-zero value showing up in the original Device Number bit field. Such configure transaction is blocked by downstream port when converting Type 1 to Type 0 → ARI Forwarding comes to rescue!

How Does ARI work? – (1 of 2 Slides)

- At RC side:
  - Device Capabilities 2 Register [5] = ARI Forwarding Supported (RO)
    - When set, indicates that this RC supports ARI Forwarding (TLPs)
  - Device Control 2 Register [5] = ARI Forwarding Enable (R/W)
    - When set by ARI-aware software, allows RC to issue configure type 0 cycles with device number != 0
  - Hints:
    - It only matters for the downstream port and its immediately below device
    - No need to enable ARI Forwarding if the immediate downstream device is not ARI-capable

Offset 0x94 — Access: Read only in RC, PV-Read only in EP

	31	5	4	3	2	1	0
R	—	ARIFS	CPL_TO_DS	CPL_TO_RS			
W							
Reset	0000_0000_0000_0000_0000_0000_000		1	0	1	1	1
PF	Implemented per PF as defined in base specification						
VF	From PF						

Figure 20-135. PCI Express Device Capabilities 2 Register *(* For LS2088A)*

Offset 0x98 — Access: Read/Write in RC, PV-Read/Write in EP

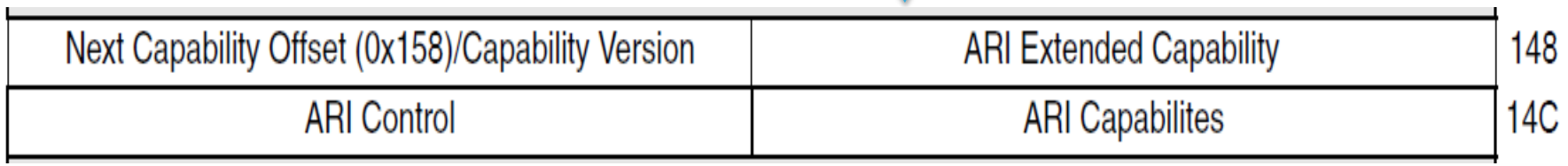
	15	6	5	4	3	0
R	—		ARIFE	CPL_TOD	CPL_TO_VAL	
W						
Reset	All zeros					
PF	Implemented per PF as defined in base specification					
VF	Copy from PF					

Figure 20-136. PCI Express Device Control 2 Register *(* For LS2088A)*

How Does ARI work? – (2 of 2 Slides)

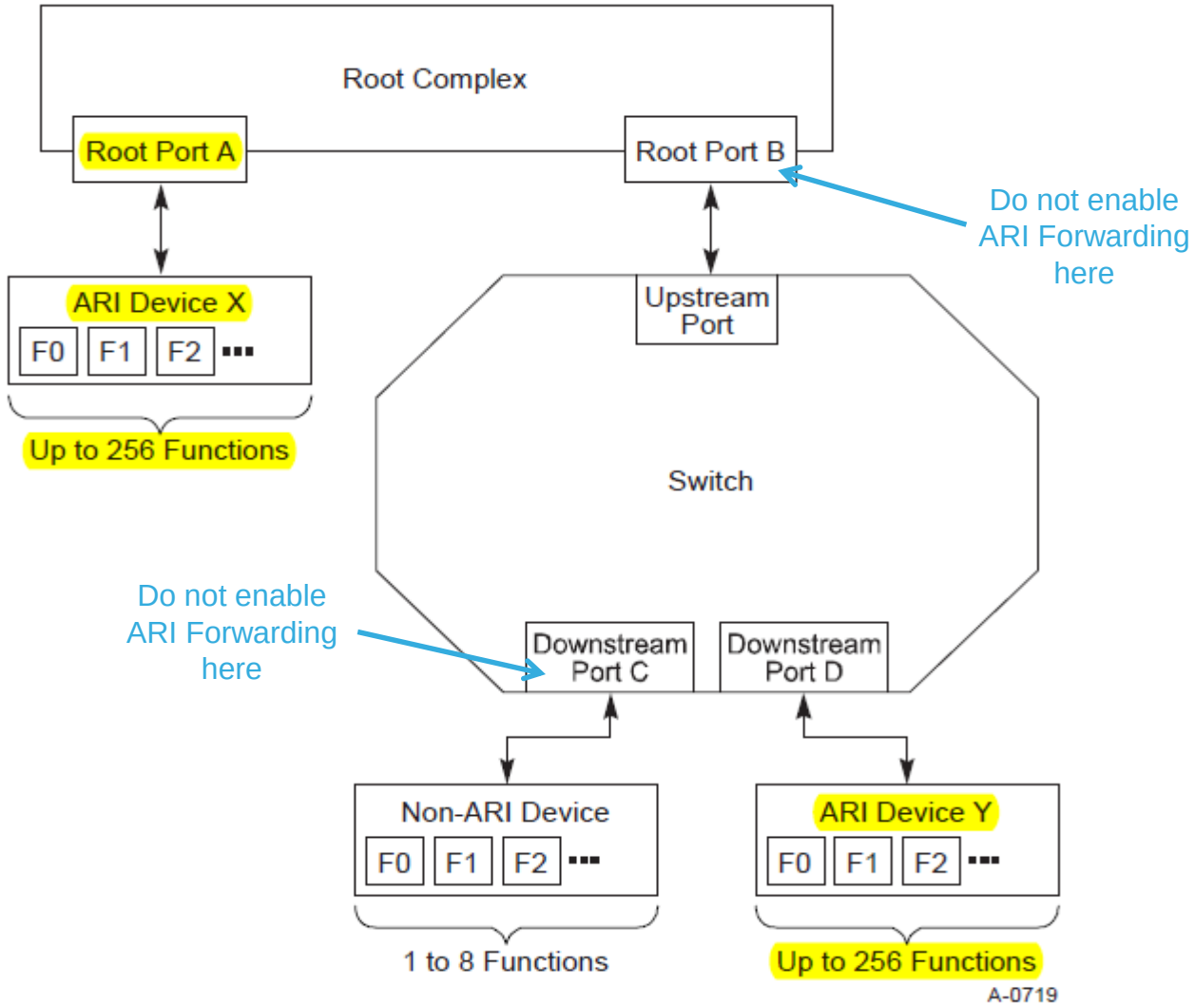
- At EP side:
  - ARI Extended Capability structure (normally applicable to EP only)

- ARI-aware software needs to:
  - Discover this capability to detect if the immediately below EP is ARI-capable
    - For QorIQ LS2088A, since it only supports SR-IOV, there is nothing to set in EP's ARI capability structure
  - If yes, enable ARI Forwarding at the immediately above downstream port
  - Discover and configure the Extended Functions (all PFs & VFs) of the ARI-device

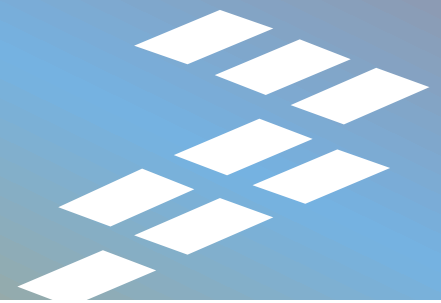


(For LS2088A)*

Example System Topology with ARI Devices



SR-IOV Extended Capability and Discovery



SR-IOV Extended Capability in QorIQ LS2088A Processor

31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
Next Capability Offset NULL = 0x000		Capability Version 0x1	PCI Express Extended Capability ID 0x0010	00h	178h
0x0000	SR IOV Capabilities (RO)		0x0002	04h	17Ch
0x00 SR IOV Status 0x00			0x00 SR IOV Control 0xnn	08h	180h
0x00 TotalVFs (RO) 0x40			0x00 InitialVFs (RO) 0x40	0Ch	184h
RsvdP 0x00	Function Dependency Link (RO) PF0: 0x00 PF1: 0x01		NumVFs (RW) 0x00nn	10h	188h
PF0: 0x0100 VF Stride (RO) PF1: 0x0100			PF0: 0x0004 First VF Offset (RO) PF1: 0x0100	14h	18Ch
VF Device ID			0x00 RsvdP 0x00	18h	190h
0x0000	Supported Page Sizes (RO)		0x0553	1Ch	194h
System Page Size (RW)				20h	198h
32-bit BAR	VF BAR0 (RW)			24h	19Ch
32-bit BAR	VF BAR1 (RW)			28h	1A0h
64-bit BAR	VF BAR2 (RW)			2Ch	1A4h
	VF BAR3 (RW)			30h	1A8h
64-bit BAR	VF BAR4 (RW)			34h	1ACh
	VF BAR5 (RW)			38h	1B0h

(Value in register reflects implementation in LS2088A)*

SR-IOV Capability Structure Discovery

- SR-PCIM software probes the configure space of each PF
  - Note that, a device may have a mix of PFs and VFs
    - Each Function (PF or VF) consumes one Routing ID
    - If the device is ARI-capable (as detected from its ARI capability structure), it may consume more than 8 functions
  - Each PF has a SR-IOV Extended Capability structure
    - Capability ID register value of 0x0010 indicates it's a SR-IOV capability structure
    - SR-IOV capabilities register value of QorIQ LS2088A = 0x0000_0002
      - Only Bit #1 is set, indicating that ARI Capable Hierarchy is preserved across certain power state transitions

31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
Next Capability Offset NULL = 0x000		Capability Version 0x1	PCI Express Extended Capability ID 0x0010	00h	178h
0x0000	SR IOV Capabilities (RO)		0x0002	04h	17Ch

(Value in register reflects implementation in LS2088A)*

SR-IOV VF Discovery

- SR-PCIM software probes the configure space of each PF
  - InitialVFs indicates number of VFs initially associated with the PF
  - TotalVFs indicates the maximum number of VFs that could be associated with the PF
  - Since QorIQ LS2088A Processor supports SR-IOV only:
    - TotalVFs = InitialVFs = 0x40, reflects total 64 VFs supported per PF

31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
Next Capability Offset NULL = 0x000		Capability Version 0x1	PCI Express Extended Capability ID 0x0010	00h	178h
0x0000 SR IOV Capabilities (RO)			0x0002	04h	17Ch
0x00 SR IOV Status	0x00		0x00 SR IOV Control 0xnn	08h	180h
0x00 TotalVFs (RO)	0x40		0x00 InitialVFs (RO) 0x40	0Ch	184h
RsvdP 0x00	Function Dependency Link (RO) PF0: 0x00 PF1: 0x01		NumVFs (RW) 0x00nn	10h	188h

(Value in register reflects implementation in LS2088A)*

SR-IOV Supported Page Sizes Discovery

- SR-PCIM software probes the configure space of each PF
  - Supported Page Size indicates the page sizes supported by all VFs of PF
    - Each bit, when set, indicates that the PF (actually for all its VFs) supports a system page size of 2^{n+12} starting from 4 KB (e.g. if bit 0 is set, the PF supports 4 KB page)
    - The register content of 0x0000_0553 indicates that the PFs in QorIQ LS2088A support all the required system page size, such as 4 KB, 8 KB, 64 KB, 256 KB, 1 MB and 4 MB
    - The value of this register will be used during SR-IOV configuration phase to align VF BAR apertures on system page boundaries
      - SR-PCICM software will choose one among all supported system page sizes to program the System Page Size register during SR-IOV configuration phase

31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
0x0000	Supported Page Sizes (RO)		0x0553	1Ch	194h
System Page Size (RW)				20h	198h

(Value in register reflects implementation in LS2088A)*

SR-IOV VF BAR Discovery

- SR-PCIM software probes the configure space of each PF
  - Each PF has its own independent set of PF BARs in its standard configuration space (Type 0 Header)
  - Each PF also has another set of VF BARs (decoders) in the configuration space, as part of the SR-IOV extended capability structure
    - All VFs within the same PF share this BAR set
    - All VFs have a single MSE bit to control the memory space for all these VFs
    - Write all “1”s and then read back to determine the size (aperture) of a VF BAR
    - For QorIQ LS2088A Processor, each VF of a PF has two 32-bit and two 64-bit VF BARs
    - All VF BARs of the PF must be aligned to the System Page Size programmed
      - Amount of address space decoded by each BAR shall be an integral multiple of the System Page Size

31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
32-bit BAR		VF BAR0 (RW)		24h	19Ch
32-bit BAR		VF BAR1 (RW)		28h	1A0h
64-bit BAR		VF BAR2 (RW)		2Ch	1A4h
		VF BAR3 (RW)		30h	1A8h
64-bit BAR		VF BAR4 (RW)		34h	1ACh
		VF BAR5 (RW)		38h	1B0h

(Value in register reflects implementation in LS2088A)*

SR-IOV First VF Offset and VF Stride Discovery

- SR-PCIM software probes the configure space of each PF
  - First VF Offset defines the Routing ID offset (from the PF's RID) of the first VF
  - VF Stride defines the Routing ID offset from one VF to the next one for all VFs associated with the PF → the RID Offset to subsequent VFs
  - The Routing ID of VF N can be determined by:
    - (PF Routing ID + First VF Offset) Module 2^{16}
  - After setting NumVFs and VF's ARI Capable Hierarchy bit, SR-PCIM can read the First VF Offset and VF Stride to determine how many bus numbers to be consumed by all VFs of the PF

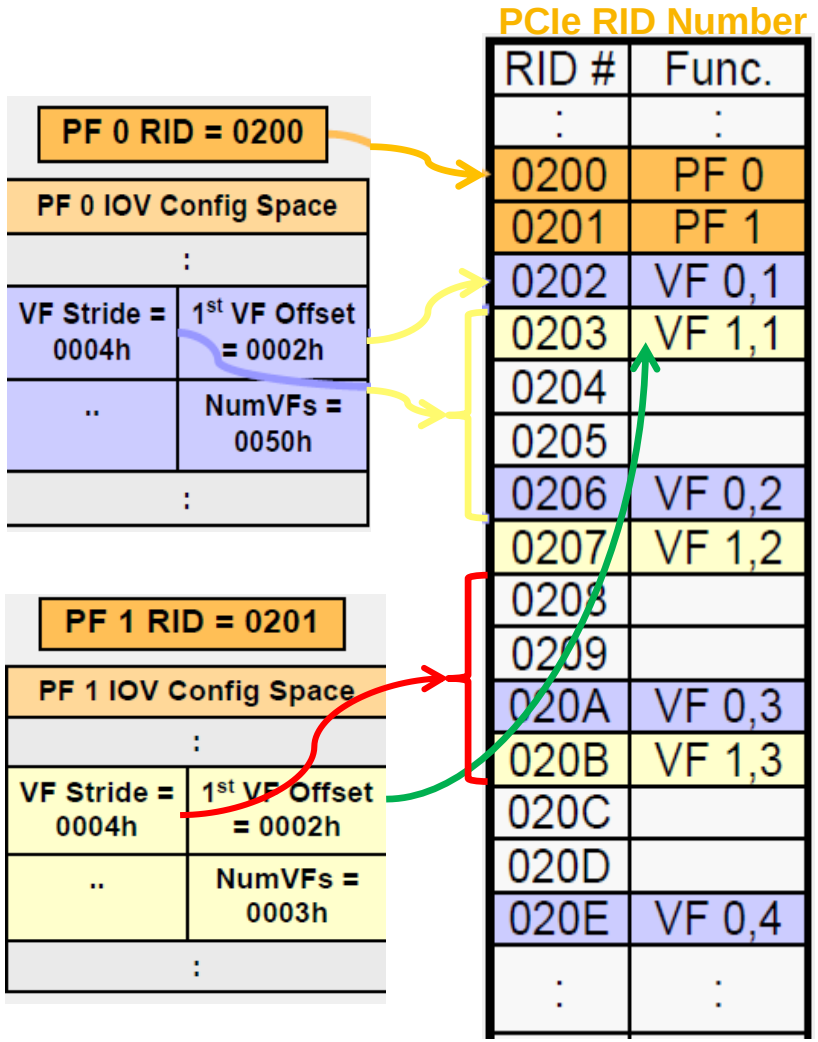
31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
RsvdP 0x00	Function Dependency Link (RO) PF0: 0x00 PF1: 0x01		NumVFs (RW) 0x00nn	10h	188h
PF0: 0x0100 VF Stride (RO) PF1: 0x0100			PF0: 0x0004 First VF Offset (RO) PF1: 0x0100	14h	18Ch

(Value in register reflects implementation in LS2088A)*

PF and VF RID Semantics

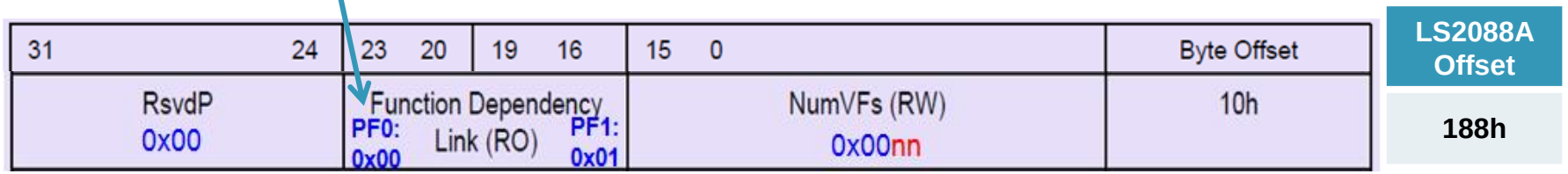
- PF and VF RIDs must not overlap given valid NumVFs setting across all PFs of a device
- PFs must be on the first Bus# captured from any Type 0 Configure Write request
  - VFs may reside on different bus number(s) than the associated PF
- For QorIQ LS2088A, if Bus# = 01
  - PF0 RID = 0100
  - PF1 RID = 0101
  - PF0's (only list some VFs here)
    - VF1 (VF 0, 1) RID = 0104
    - VF2 (VF 0, 2) RID = 0204
    - VF3 (VF 0, 3) RID = 0304
  - PF1's (only list 1 VF here)
    - VF1 (VF 1, 1) RID = 0201
    - VF1 (VF 1, 2) RID = 0301

PF and VF RID Assignment Example (not for LS2088A)



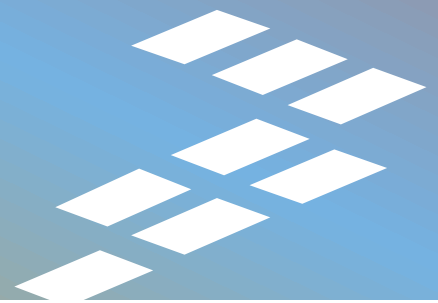
SR-IOV Function Dependency Discovery

- SR-PCIM software probes the configure space of each PF
  - Function Dependency is an optional field used to describe vendor specific dependencies between sets of functions
    - If a PF is independent from other PFs of a Device, this field shall contain its own Function Number
  - For QorIQ LS2088A Processor, both PF0 and PF1 are independent



(Value in register reflects implementation in LS2088A)*

Reset Mechanism and Function Level Reset (FLR)



Reset Mechanisms

- Three reset mechanisms are supported:
  - Conventional Reset
    - Resets all PF and VF state
  - Function Level Reset (FLR) that targets a PF
    - Resets a PF and its associated VFs
  - Function Level Reset (FLR) that targets a VF
    - Resets a single VF

Conventional Reset

- PCIe Conventional Reset includes Fundamental Reset and In-band Hot Reset
  - Note that link down event to EP is treated same as Hot Reset
- A PCIe Conventional Reset to a SR-IOV Device shall cause all functions (including PFs, VFs and VF context) to be reset to their original, power-on state
- If a PF has its VFs enabled and a Conventional Reset is issued to the device, the device must reset all PF and VF state:
  - The PF must disable its SR-IOV capabilities and reverts back to being a PCI function
  - Configurable SR-IOV capabilities are reset to default values
  - The PF's SR-IOV Control register [VF Enable] is cleared
    - VFs no longer exist after a Conventional Reset
  - MSE and BME are both off
  - All BAR values used by the PF and VFs are indeterminate
  - All interrupts are disabled

Introduction Function Level Reset (FLR)

- FLR is a mechanism defined by the PCIe Base Specification for resetting a specific EP function
  - It allows software to quiesce and reset EP hardware with function-level granularity
  - Only the targeted function is affected by the FLR operation
  - The PCIe Link State must not be affected by an FLR
  - For further detail of FLR, refer to §6.6.2 in PCIe Base Specification Rev 2.1

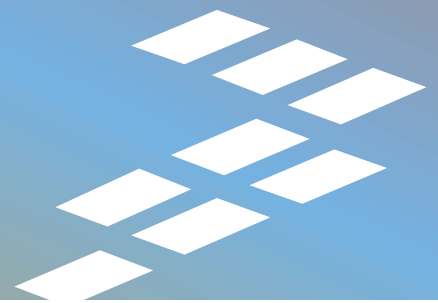
- How to issue an FLR?
  - PCIe Base Specification adds two bits for a PCIe EP
    - EP's Device Capabilities Register [28] = FLRC
      - When set, indicates this EP supports FLR capability
    - EP's Device Control Register [15] = IFLR (Initiate an FLR)
  - Software can write a 1b to the EP's Device Control Register [15] = IFLR bit of the targeted Function to initiate an FLR

FLRs to PFs and VFs

- An FLR targeting a PF:
  - Reset the PF's SR-IOV Extended Capability structure
  - The VFs no longer exist, since VF Enable bit is cleared

- An FLR targeting a VF:
  - Only affect VF's state
  - Not affect the VF's existence. It still consumes a RID
  - Not affect the address mapping assigned
    - VF's BAR registers and VF MSE bit are unaffected by FLR to VF

SR-IOV Configuration



Configuring the Number of VFs Desired

- Before this step, it's assumed that the host SR-PCIM software already:
  - Discovered and enabled the ARI
    - For SR-IOV EP, SR-PCIM software sets the EP's SR-IOV Control [4] = "ARI Capable Hierarchy"
  - Read TotalVFs and InitialVFs
- Set the NumVFs desired, that defines the number of VFs to be enabled
  - NumVFs must be ≤ TotalVFs (same as InitialVFs for QorIQ LS2088A Processor)
  - After configuring NumVFs, SR-PCIM may read the First VF Offset and VF Stride
    - This allows system to determine how many Bus Numbers to be consumed by the PF's VFs, although those Bus Numbers are not actually used before VF Enable is set
  - When VFs are enabled later, the PCIe SR-IOV Device associates NumVFs worth of VFs with the PF

31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
Next Capability Offset NULL = 0x000		Capability Version 0x1	PCI Express Extended Capability ID 0x0010	00h	178h
0x0000 SR IOV Capabilities (RO)			0x0002	04h	17Ch
0x00 SR IOV Status	0x00		0x00 SR IOV Control 0xnn	08h	180h
0x00 TotalVFs (RO)	0x40		0x00 InitialVFs (RO) 0x40	0Ch	184h
RsvdP 0x00	Function Dependency Link (RO) PF0: 0x00 PF1: 0x01		NumVFs (RW) 0x00nn	10h	188h

(Value in register reflects implementation in LS2088A)*

Configuring the System Page Size and VF's BARs

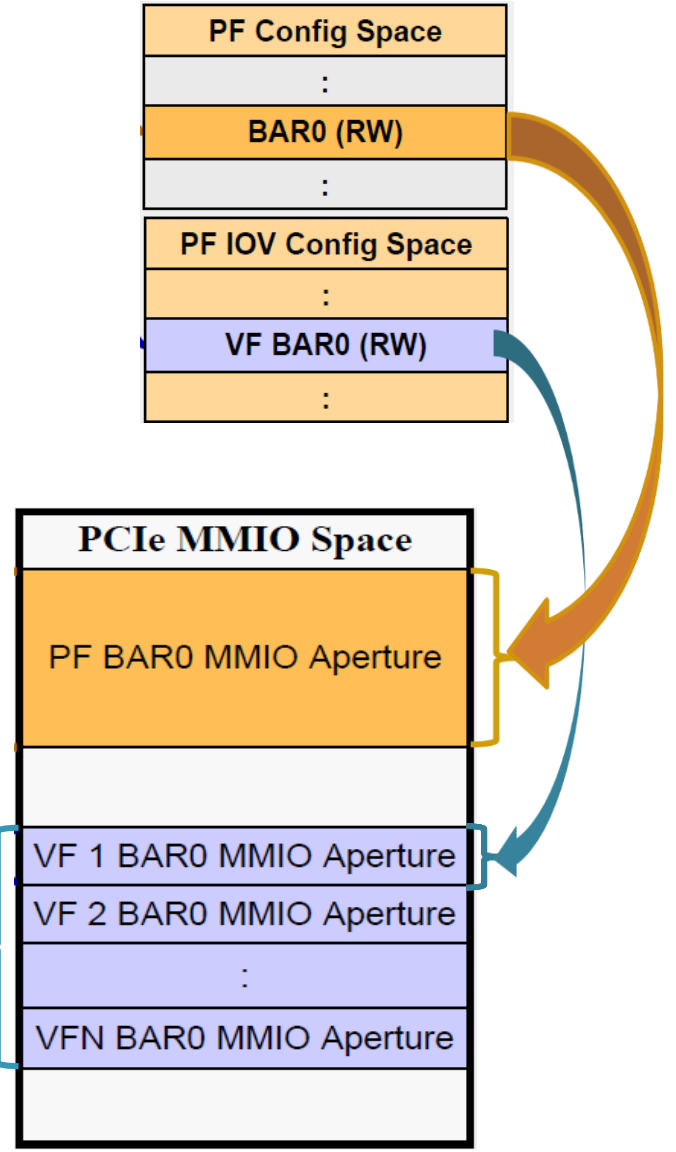
- System Page Size defines the page size the system will use
  - Value of System Page Size must be set to one of the Supported Page Size
  - System Page Size is used by the PF to align the MMIO aperture defined by each BAR to a system page boundary
  - The System Page Size must be configured/written before enabling the VF (setting SR-IOV Control register [0] = "VF Enable" bit)
  - Based on the discovered memory aperture (size) required for each VF BAR and the System Page Size, software can write the address into each VF BAR, that becomes the starting address for that BAR on the first VF of its PF

31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
0x0000	Supported Page Sizes (RO)		0x0553	1Ch	194h
System Page Size (RW)				20h	198h
32-bit BAR	VF BAR0 (RW)			24h	19Ch
32-bit BAR	VF BAR1 (RW)			28h	1A0h
64-bit BAR	VF BAR2 (RW)			2Ch	1A4h
	VF BAR3 (RW)			30h	1A8h
64-bit BAR	VF BAR4 (RW)			34h	1ACh
	VF BAR5 (RW)			38h	1B0h

(Value in register reflects implementation in LS2088A)*

PF and VF BAR Semantics

- The configured address of (each) VF BARn
  - Is used by Device to set the starting address for VF BARn on the first VF
- For each VF BAR, the memory space associated with the 2nd and higher VF
  - Is derived from the starting address of the first VF and the memory space aperture
  - BARn apertures for all VFs of a PF are in a consecutive memory region
    - Logically concatenates all VFs of a PF into one contiguous chunk of PCIe memory space for each implemented VF BARn
- The VF BAR's MMIO space is not enabled until both VF Enable and VF MSE are set



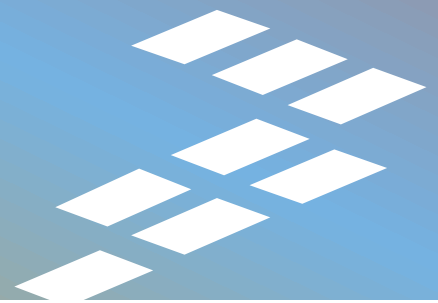
Enabling VFs

- Set the PF's SR-IOV Control register [3] = "VF MSE" bit first
  - VF MSE (Memory Space Enable) is shared among all VFs within a PF
- Set the VF's Command register [2] = "Bus Master Enable" bit
  - **Yes, each VF has this bit by itself. It's located at the VF's Type 0 Header**
- ***At last***, a PF's VFs can be enabled by setting "VF Enable" bit
  - SR-IOV Control register [0] = "VF Enable"
  - Clearing "VF Enable" effectively destroys all VFs
  - Setting "VF Enable" effectively creates all VFs
  - System software must wait at least 100 ms after changing "VF Enable" bit from a 0 to a 1, before issuing VFs to be enabled

31 24	23 20	19 16	15 0	Byte Offset	LS2088A Offset
Next Capability Offset NULL = 0x000		Capability Version 0x1	PCI Express Extended Capability ID 0x0010	00h	178h
0x0000 SR IOV Capabilities (RO) 0x0002				04h	17Ch
0x00 SR IOV Status 0x00			0x00 SR IOV Control 0xnn	08h	180h
0x00 TotalVFs (RO) 0x40			0x00 InitialVFs (RO) 0x40	0Ch	184h
RsvdP 0x00	Function Dependency Link (RO) PF0: 0x00 PF1: 0x01		NumVFs (RW) 0x00nn	10h	188h

(Value in register reflects implementation in LS2088A)*

SR-IOV EP Features in QorIQ LS2088A Processor



QorIQ LS2088A Processor SR-IOV EP Feature Highlight (*Preliminary*)

- One PCI Express Controller (PEX3) supports SR-IOV in EP mode
  - Supports SR-IOV specification Rev 1.1 with 2 PFs and 64 VFs per PF
  - Supports Function Level Reset (FLR)
  - Supports Alternative Routing Interpretation (ARI)
  - Supports MSI-X for PF/VF with 8 MSI-X vector per PF or VF
  - Each PF has its dedicated 8-Kbyte CCSR memory-mapped register space:
    - PF0:
      - 0x360_0000 – 0x360_0FFF: For PF0's PF registers (iATU registers are shared by both PF0 and PF1)
      - 0x360_1010 – 0x360_1024: For PF0's BAR Mask registers (BAR0_MASK to BAR5_MASK)
      - 0x368_0000 – 0x368_0FFF: For shared PF0 & PF1 PEX Lookup Table (PEXLUT) registers
      - 0x36C_0000 – 0x36C_0FFF: For PF0's PF Control registers
    - PF1:
      - 0x362_0000 – 0x362_0FFF: For PF1's PF registers
      - 0x362_1010 – 0x362_1024: For PF1's BAR Mask registers (BAR0_MASK to BAR5_MASK)
      - 0x36x_0000 – 0x36x_0FFF: For PF1's PF Control registers (offset is still TBD, not used for iATU and PF or VF BAR setup)

PCIe Controller Type 0 Configuration Header – for EP

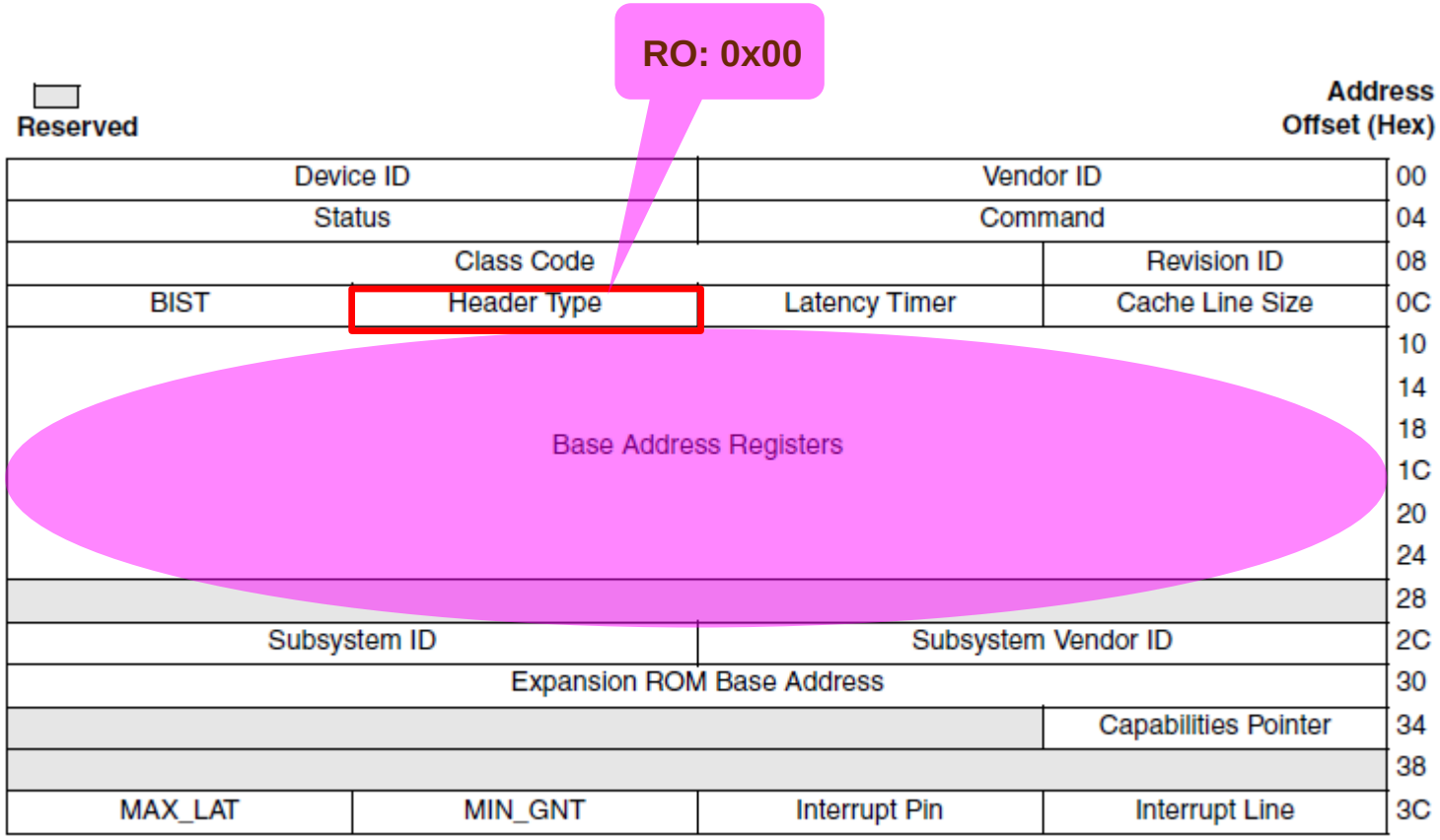
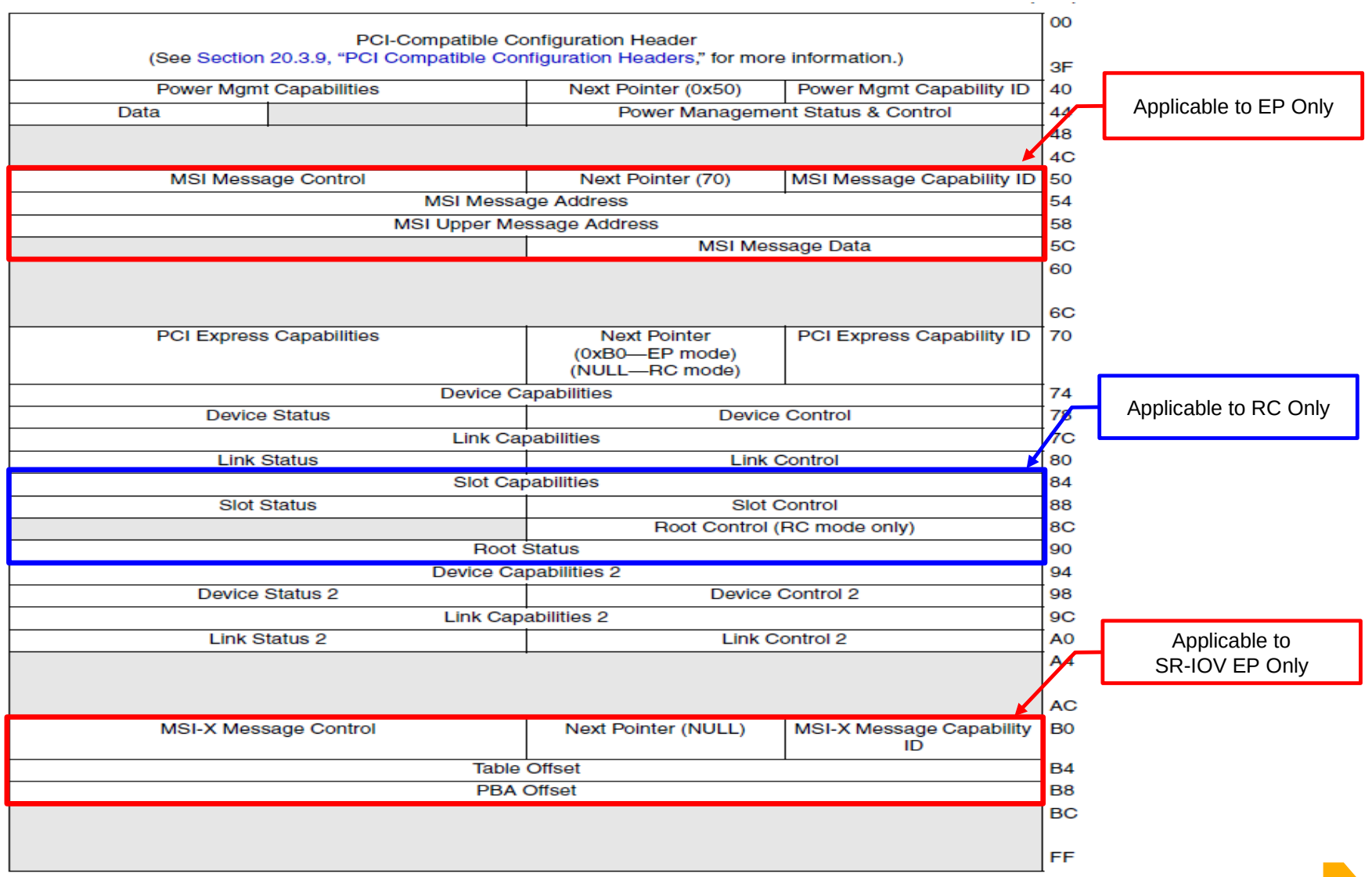
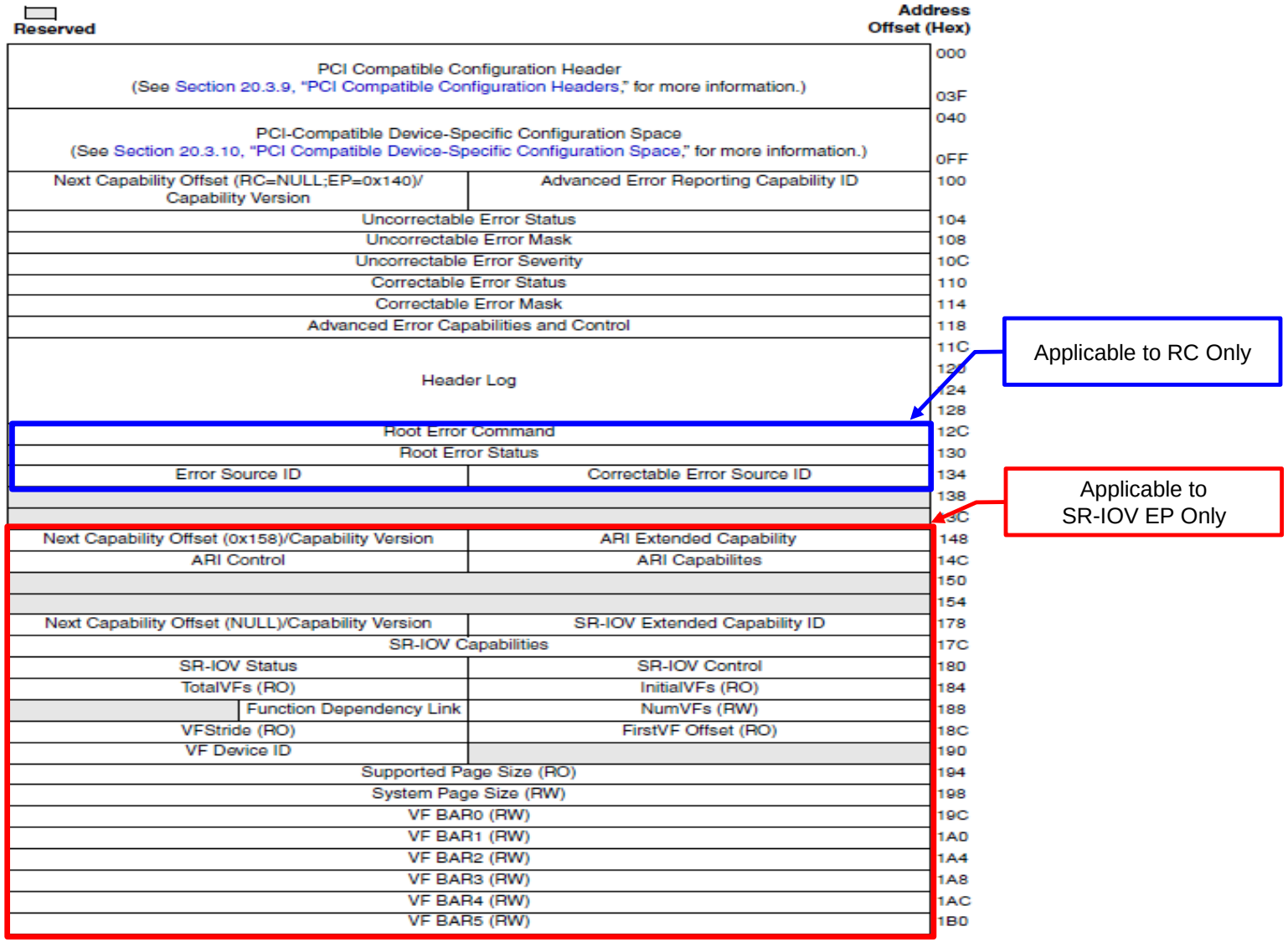


Figure 20-76. PCI Express PCI-Compatible Configuration Header—Type 0

PCIe Controller PCIe Device-Specific Configuration Header



PCIe Controller Extended Configuration Space



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 (MISC_CONTROL_1_OFF)

- iATU (internal Address Translation Unit) registers
  - Total of 24 inbound memory regions for the entire PEX3 SR-IOV EP controller
  - Total of 256 outbound memory regions for the entire PEX3 SR-IOV EP controller
  - Programmable location and size (from 4 KB to 4 GB) for each region

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

iATU Address Translation Diagram

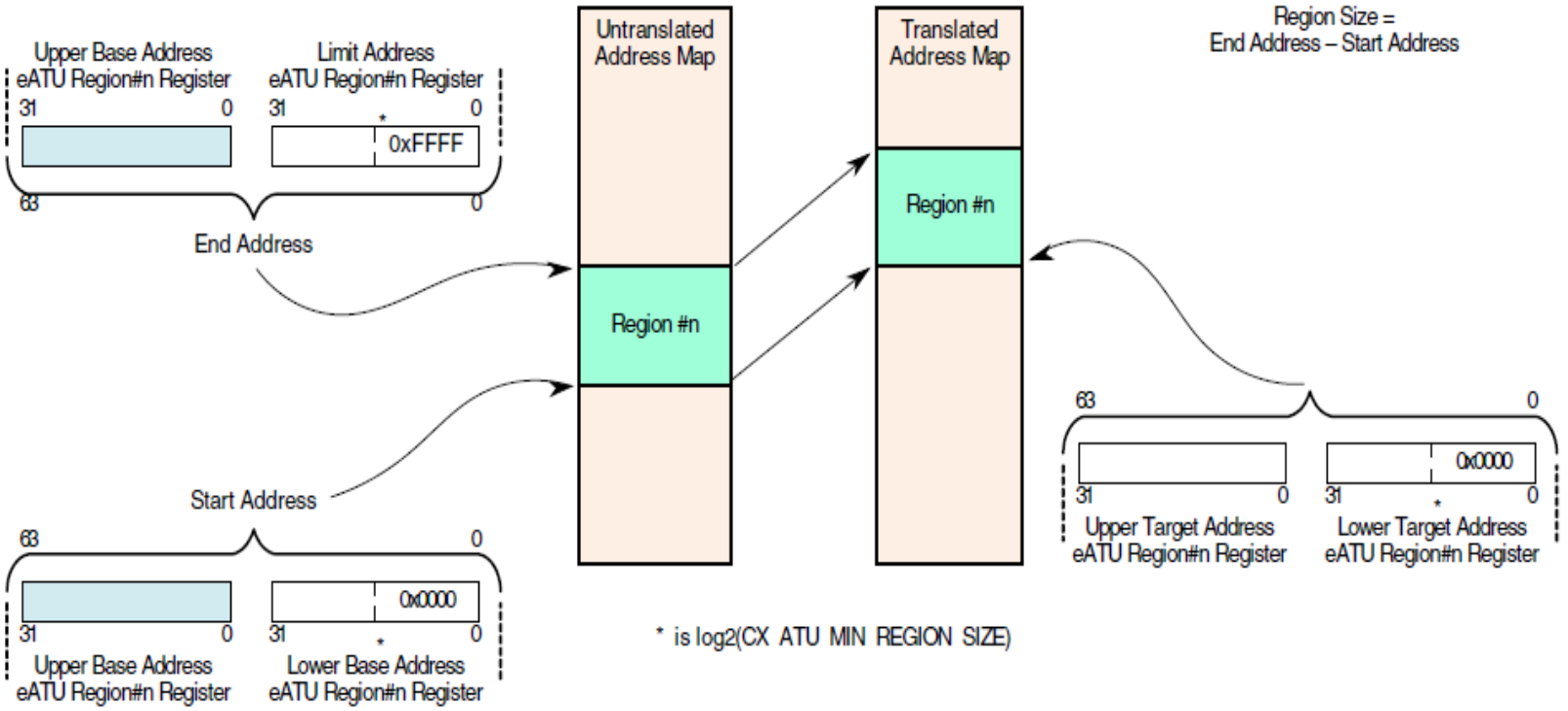
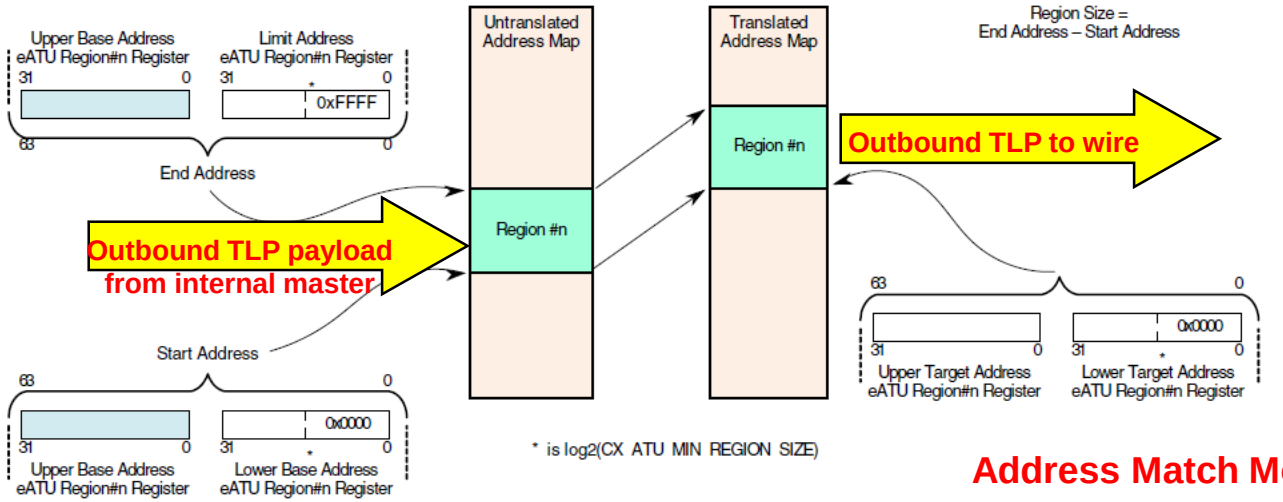
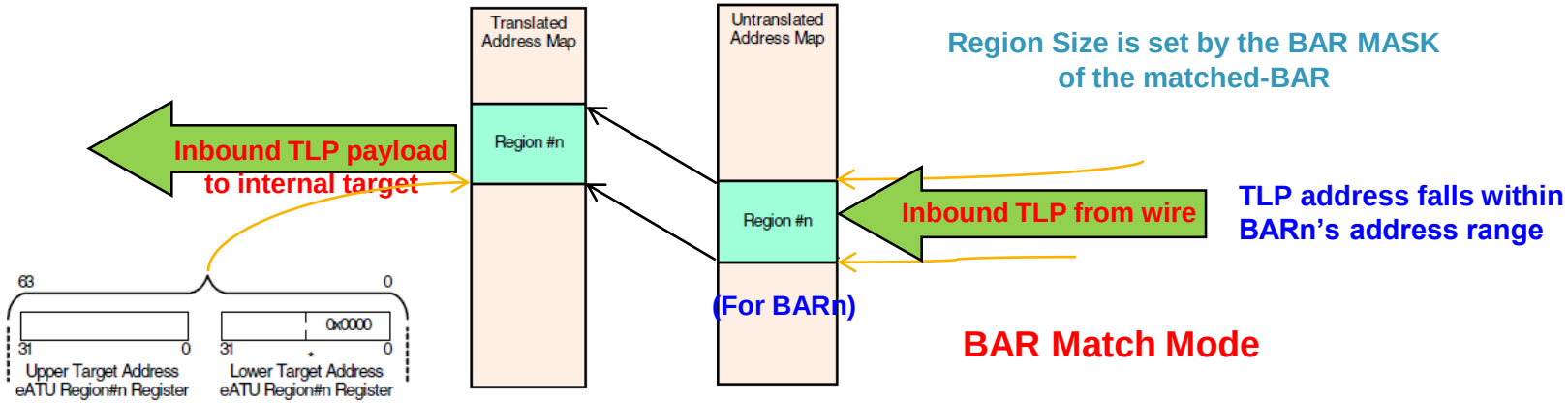


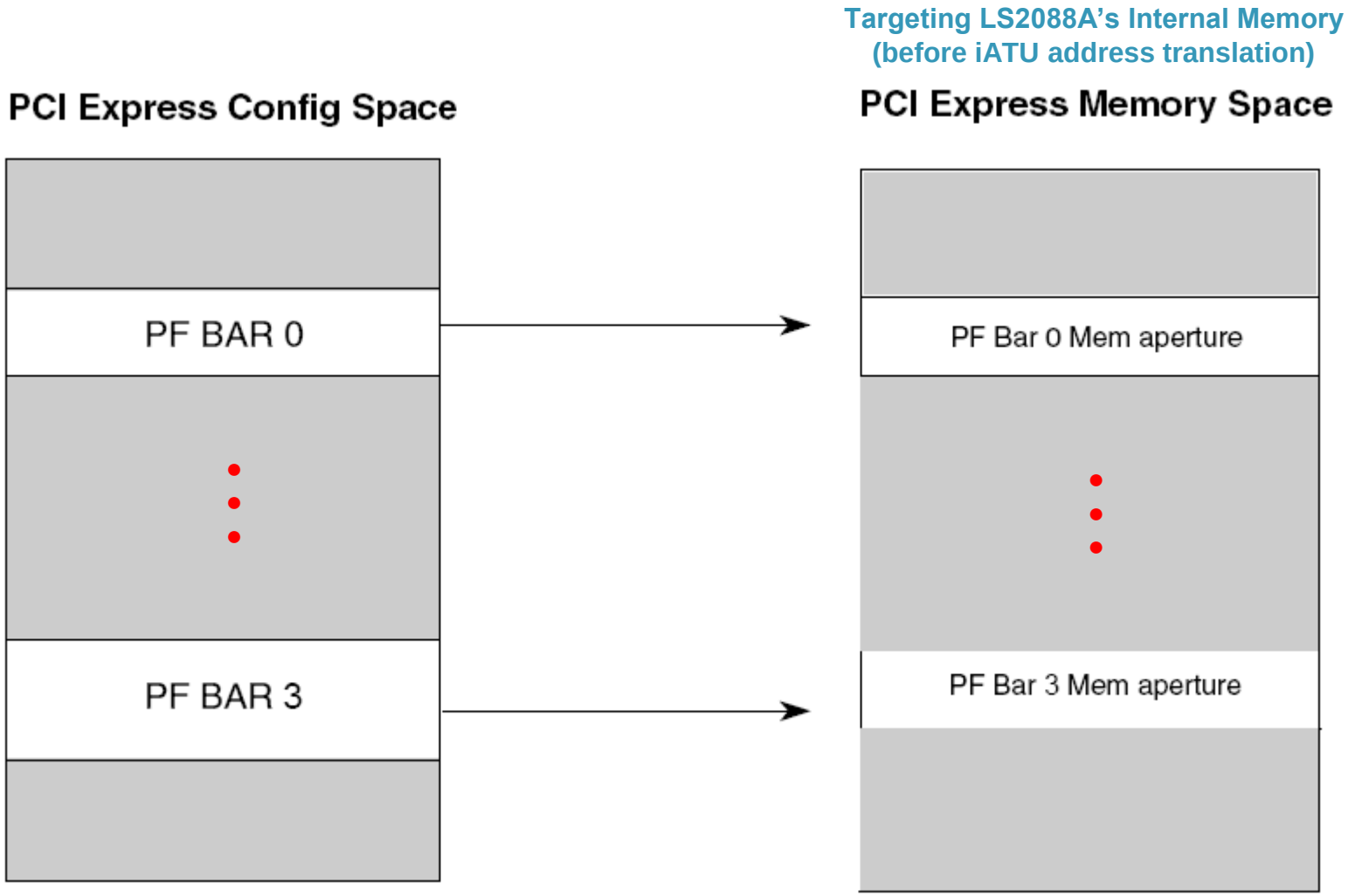
Figure 20-238. iATU Address Region Mapping: Outbound and Inbound (Address Match Mode), 64-bit Address

iATU Address Translation Example – for EP

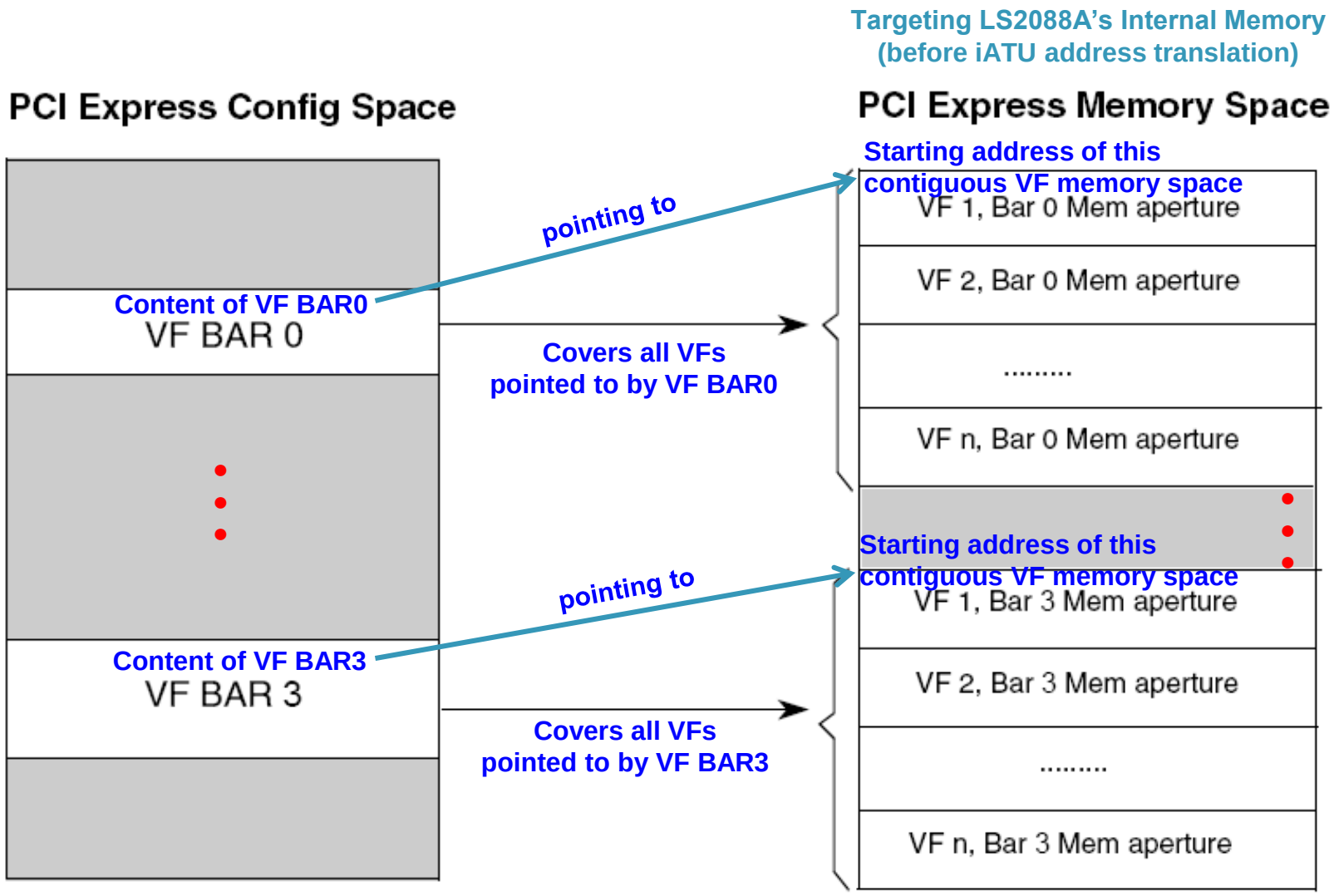




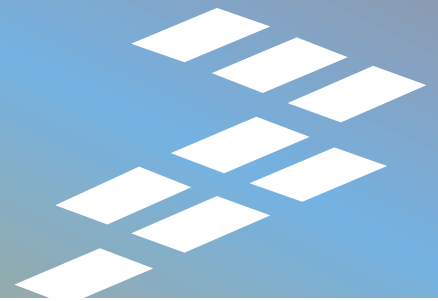
Inbound Access via PF BARs for QorIQ LS2088A PEX3 SR-IOV EP



Inbound Access via VF BARs for QorIQ LS2088A PEX3 SR-IOV EP



QorIQ LS2088A Processor PEX3 SR-IOV EP's iATU and BAR Local Software Programming Procedure



Tip for PEX3 SR-IOV EP's Self-Initialization

- 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
  - Before setting each PF's PEX_PFn_CONFIG [CFG_READY] bit, the boot loader needs to
    - Self-configure each PF and VF's BAR characteristics and size via iATU registers so that the remote RC can read the correct information required for SR-IOV EP's application

- 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

- One set of iATU registers shared for both PF0 and PF1 starting at address 0x360_0900
- 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 [7:0], REGION_INDEX → valid setting 0x00 to 0x17 (or *0xFF*) to cover all 24 (or *256*) regions for inbound (or *outbound*)

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 address of a region prior to translation.
0x910	iATU Region Upper Base Address Register
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 translation.
0x91C	iATU Region Upper Target Address Register
0x920	iATU Region Control 3 Register → Only used for SR-IOV EP VF's outbound RID generation

Special Note for PEX3 SR-IOV EP's BAR0 and BAR1

- BAR0 setup at EP side:
  - According to PCIe base specification, BAR0 at offset 0x10 is an 32-bit non-prefetchable BAR
  - Software should ensure that both the PREFETCH and TYPE bits are configured accordingly in the BAR0 registers
- BAR1 setup at EP side:
  - As indicated in the BIR bit field of the MSI-X PBA Offset/BAR register located at offset 0xB8, the BAR1 at offset 0x14 is reserved for EP's MSI-X usage.
  - BAR1's internal mapping is fixed and not controlled by iATU. Internally, it is hardwired to QorIQ LS2088A Processor's SRAM instead of system memory with size of 64 Mbytes.
  - EP's local software does not need to configure the iATU, BAR1 and BAR1_MASK registers for BAR1. The default setting should work as long as the BAR1 is not disabled and then re-enabled.
    - Otherwise, the BAR1 and BAR1_MASK registers should be re-configured with the following characteristics: 32-bit, non-prefetchable and 64 MByte as size. These characteristics may lose when the BAR1 is disabled.
  - Only the remote host (RC) can access EP's BAR1 content. The EP's local software can not do so.
  - The MSI-X needs to be enabled by setting Bit [15, Enable] of the MSI-X Message Control Register located at offset 0xB2

Other Note for the Programming Procedure

- Always use “Read-Modify-Write” to update registers with the exception below:
  - To update all the address registers, use write instead of “Read-Modify-Write”
  - All the mask shadow registers (for example, BARn_MASK and etc.) are actually Write-Only. Therefore, use write instead of “Read-Modify-Write”.

- The goal of the programming procedure is to cover all the configuration required for one inbound iATU region and its related BAR of QorIQ LS2088A PEX3 SR-IOV EP
  - How to program all the required iATU registers for an inbound iATU region for ***EITHER*** a PF BARn ***OR*** a VFBARn shared by VFs
    - ***A PF and its VFs can NOT share the same inbound iATU region!!***
  - How to program all the required BARn and BARn_MASK registers for both PF and VFs

- EP’s local software can follow the same procedure to configure other inbound iATUs and BARs. The recommended programming order is:
  - Configure an inbound iATU region required for ***each*** PF BARn first followed by the associated PF BARn & BARn_MASK setup. Repeat the same procedure for each PF BARn.
  - Configure another inbound iATU required for ***ALL*** VFs of the each PF followed by the associated VFBARn & VFBARn_MASK setup. Repeat the same procedure for each VFBARn.

- EP’s local software should finish all the required configuration for SR-IOV EP’s outbound iATU regions (not covered in this presentation) as well before setting the CFG_READY bit

Inbound iATU Programming Procedure for PEX3 SR-IOV EP

- iATU programming procedure below to cover ***EITHER*** a PF BARn ***OR*** a VFBARn shared by VFs
- The PF iATU and VFs iATU programming procedure are almost the same *except Step# 5*

1) **Select a region to use via the Index Register**

   - Write to Index Register (IATU_VIEWPORT_OFF) at offset 0x900
     - Bit [31, REGION_DIR] = 1 (inbound)
     - Bit [2:0, REGION_INDEX] = desired Region# N (from 0x00 to 0x17, total of 24 inbound regions/windows allowed)
     - Note: The above two bit fields “act as” the “virtual chip select” for a set of iATU registers. From now on until the next write to the Index Register, all the iATU registers touched between 0x904 and 0x920 are part of the “_INBOUND_” “Region# N” group registers.
     - Example: If REGION_DIR = 1 and REGION_INDEX = 0x01 → We are now accessing Inbound Region# 1 group iATU registers until the next write to this Index Register again.

Inbound iATU Programming Procedure for PEX3 SR-IOV EP (cont.)

2) **Set up the Region's internal translated address via Target Address Registers**

- Write to Region# N's Lower Target Address Register (IATU_LWR_TARGET_ADDR_OFF_INBOUND_0) at offset 0x918
  - Bit [31:12, LWR_TARGET_RW] = [31:12] portion of the desired lower 32-bit address
- Write to Region# N's Upper Target Address Register (IATU_UPPER_TARGET_ADDR_OFF_INBOUND_0) at offset 0x91C
  - Bit [31:0, UPPER_TARGET_RW] = [63:32] portion of the desired upper 32-bit address
  - Note 1: only the lower 8 bits are valid due to the 40-bit internal address
  - Note 2: The target address to be programmed must be aligned to the size of either the PF BAR or VF aperture, depending on whether the iATU region is used for PF or VFs.
  - Example: If Lower Target Address Register = 0x8000_0000 and Upper Target Address Register = 0x0, this configures the inbound Region# 1's target address at DDR as: 0x0000_0000_8000_0000, assuming the PF BAR size (or VF Aperture size) is 256 MB, depending on whether the iATU region is used for PF or VFs.

Inbound iATU Programming Procedure for PEX3 SR-IOV EP (cont.)

3) **No need to set up the Region's external PCIe address in Base & Limit Address Registers**

- 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.

4) **Set up the PF Number and BAR Type via Region Control 1 Register**

- Write to Region# N's Region Control 1 Register (IATU_REGION_CTRL_1_OFF_INBOUND_0) at offset 0x904
  - Bit [24:20, CTRL_1_FUNC_NUM] = desired PF Function Number, PF# (either 0 or 1)
  - Bit [4:0, TYPE] = 00000b for memory BAR

Inbound iATU Programming Procedure for PEX3 SR-IOV EP (cont.)

5) Set up the Region's matching characteristics via Region Control 2 Register

- Write to Region# N's Region Control 2 Register (IATU_REGION_CTRL_2_OFF_INBOUND_0) at offset 0x908
  - Bit [30, MATCH_MODE] = 1 (BAR Match Mode)
  - Bit [26, VFBAR_MATCH_MODE_EN] = ***0b for PF iATU setup*** or ***1b for VFs iATU setup***

    For VFs iATU, setting this bit to 1b allows inbound memory transaction match a VFBARn to be further matched with a single inbound iATU region shared by all VFs within a PF.
  - Bit [20, VF_MATCH_EN] = 0 (This bit must be cleared for both PF and VFs iATU setup)
  - Bit [19, FUNC_NUM_MATCH_EN] = 1 (Function Number Match Enable, ***1b setting required for both PF and VFs iATU setup***)
  - Bit [10:8, BAR_NUM] = desired PF or VF BAR Number for the current inbound iATU region to be programmed

6) Enable this inbound iATU region via Region Control 2 Register

- Write to Region# N's Region Control 2 Register (IATU_REGION_CTRL_2_OFF_INBOUND_0) at offset 0x908
  - Bit [31, REGION_EN] = 1 (Region Enable)

Inbound PF BAR Configuration for PEX3 SR-IOV EP

- This covers the additional configuration required for a PF BAR associated with the current inbound iATU region under configuration
- There are two BAR registers for each PF BAR
  1) The **BARn** register at PF's normal Type 0 configure space offset (For example, 0x18 for BAR2_REG) with the following bits to be configurable
     - Bit [3, PREFETCH], Read-Only normally. Writable with RO_WR_EN bit turned on
     - Bit [2:1, TYPE], Read-Only normally. Writable with RO_WR_EN bit turned on
  2) The **BARn_MASK** shadow register located at its equivalent BARn register's offset + 0x1000 (For example, 0x1018 for BAR2_MASK register) with the following bits to be configurable
     - Bit [31:1, MASK], Write-Only BARn Mask.
     - Bit [0, BARn_EN], Write-Only BARn Enable.

Inbound PF BAR Programming Procedure for PEX3 SR-IOV EP

1) **Enable a PF BAR to prepare for further configuration via BARn_MASK Register**

- Write to DBI Read-Only Write Enable Register (MISC_CONTROL_1_OFF) at offset 0x8BC
  - Bit [0, RO_WR_EN] = 1
  - Note: This is required for updating Read-Only and mask registers.

- Write to BARn_MASK Shadow Register at BARn's offset + 0x1000
  - Bit [0, BARn_EN] = 1 (Enable the BARn for this PF#, for example BAR2 for PF# 1)
  - Note: The PF BARn mast be enabled first before updating its associated mask.
  - Example: PF# 1's BAR2 is located at PF# 1's Type 0 configure space offset 0x18 (0x362_0018). The associated BAR2_MASK is located at same configure space's offset 0x1018 (0x362_1018).

Inbound PF BAR Programming Procedure for PEX3 SR-IOV EP (cont.)

- The Step# 2 below is only required when any 64-bit PF BAR (BAR2 or BAR4) is disabled and re-enabled. By default, all PF BARs are enabled. The prefetchable and memory space bits of the BAR2 and BAR4 are also set by default.

2) Re-touch the PREFETCH and TYPE bit fields for 64-bit Prefetchable PF BAR

- With RO_WR_EN bit turned on, write to the current PF's BARn Register at this PF's Type 0 configure space BARn offset
  - Bit [3, PREFETCH] = 1 (Prefetchable)
  - Bit [2:1, TYPE] = 10b (64-bit BAR)
  - Note 1: Not applicable for any PF's 32-bit BAR (BAR0 or BAR1)
  - Note 2: PF0's BAR2 locates at 0x360_0018, BAR4 at 0x360_0020

    PF1's BAR2 locates at 0x362_0018, BAR4 at 0x362_0020

Inbound PF BAR Programming Procedure for PEX3 SR-IOV EP (cont.)

3) **Set up the BARn's Mask (Size) for the current PF BAR via BARn_MASK Register**

- Write to BARn_MASK Shadow Register at BARn's offset + 0x1000
  - Bit [31:1, MASK] = desired lower 32-bit of the current PF#'s BAR mask
  - Bit [0, BARn_EN] = 1 (Enable the BARn for this PF#, for example BAR2 for PF# 1)
  - Note: When writing the mask, need to preserve the BARn_EN bit.
  - Example: PF# 1's BAR2's associated BAR2_MASK shadow register is located at PF#1's Type 0 configure space's offset 0x1018 (0x362_1018)

4) **For 64-bit BAR pair, set up the BARn+1's Mask (Size) via the BARn+1_MASK Register**

- Write to BARn+1_MASK Shadow Register at BARn+1's offset + 0x1000
  - Bit [31:0, MASK] = desired upper 32-bit portion of the current PF#'s 64-bit BAR mask
  - Note: Applicable for any PF's BAR3_MASK or BAR5_MASK shadow register
  - Example: PF# 1's BAR3's associated BAR3_MASK register is located at PF#1's Type 0 configure space's offset 0x1020 (0x362_1020).

Inbound VF BAR Configuration for PEX3 SR-IOV EP

- This covers the additional configuration required for a **VF** BAR associated with the current inbound iATU region under configuration
- There are two BAR registers for each **VF** BAR
  1) The **VFBARn** register at PF's normal configure space offset (For example, 0x1A4 for VFBAR2) with the following bits to be configurable
     - Bit [3, PREFETCH], Read-Only normally. Writable with RO_WR_EN bit turned on
     - Bit [2:1, TYPE], Read-Only normally. Writable with RO_WR_EN bit turned on
  2) The **VFBARn_MASK** shadow register located at its equivalent VFBARn register's offset + 0x1000 (For example, 0x11A4 for VFBAR2_MASK register) with the following bits to be configurable
     - Bit [31:1, MASK], Write-Only VFBARn Mask
     - Bit [0, BARn_EN], Write-Only BARn Enable

Inbound VF BAR Programming Procedure for PEX3 SR-IOV EP

1) **Enable a VF BAR to prepare for further configuration via VFBARn_MASK Register**

- Write to DBI Read-Only Write Enable Register (MISC_CONTROL_1_OFF) at offset 0x8BC
  - Bit [0, RO_WR_EN] = 1
  - Note: This is required for updating Read-Only and mask registers.

- Write to VFBARn_MASK Shadow Register at VFBARn's offset + 0x1000
  - Bit [0, BARn_EN] = 1 (Enable the VFBARn for this PF#, for example VFBAR2 for PF# 1)
  - Note: The VFBARn mast be enabled first before updating its associated mask.
  - Example: PF# 1's VFBAR2 is located at PF# 1's configure space offset 0x01A4 (0x362_01A4). The associated VFBAR2_MASK is located at same configure space's offset 0x11A4 (0x362_11A4).

Inbound VF BAR Programming Procedure for PEX3 SR-IOV EP (cont.)

2) Set up the VFBARn's Mask (Size) for the current VF BAR via VFBARn_MASK Register

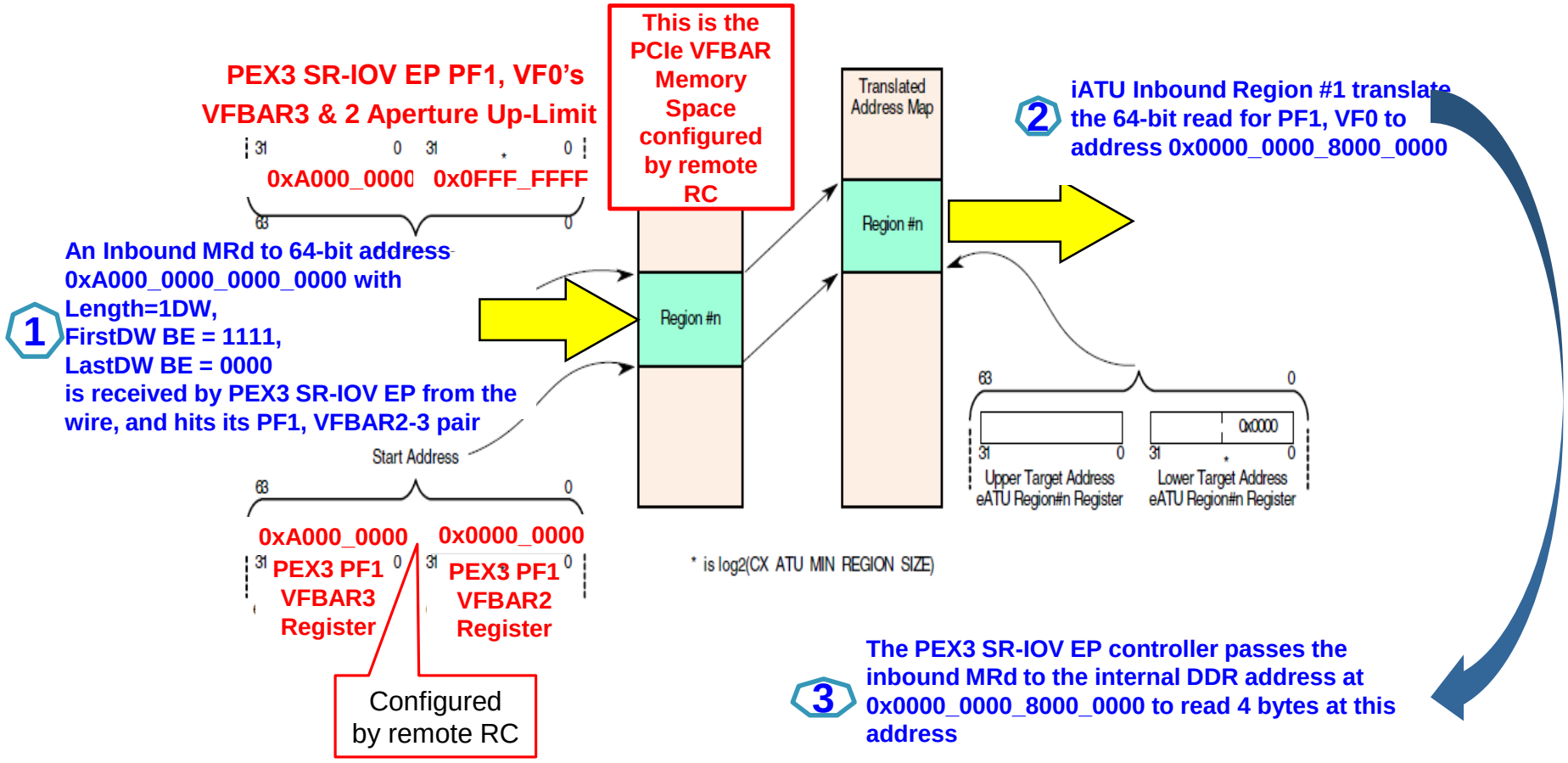
- Write to VFBARn_MASK Shadow Register at VFBARn's offset + 0x1000
  - Bit [31:1, MASK] = desired lower 32-bit of the current PF's VFBARn mask (aperture size)
  - Bit [0, BARn_EN] = 1 (Enable the VFBARn for this PF#, for example VFBAR2 for PF# 1)
  - Note 1: When writing the mask, need to preserve the VFBARn_EN bit.
  - Note 2: The VFBARn is shared by all the VFs within a PF. Therefore the VFBARn mask size configured becomes the memory aperture size for all VFs within a PF, which must be an integral multiple of the System Page Size.
  - Example: PF# 1's VFBAR2's associated VFBAR2_MASK shadow register is located at PF#1's configure space's offset 0x11A4 (0x362_11A4)

3) For 64-bit BAR pair, set up VFBARn+1's Mask (Size) via the VFBARn+1_MASK Register

- Write to VFBARn+1_MASK Shadow Register at VFBARn+1's offset + 0x1000
  - Bit [31:0, MASK] = desired upper 32-bit portion of the current PF#'s 64-bit VFBAR mask (aperture size)
  - Note: Applicable for any PF's VFBAR3_MASK or VFBAR5_MASK shadow register
  - Example: PF# 1's VFBAR3's associated VFBAR3_MASK register is located at PF#1's configure space's offset 0x11A8 (0x362_11A8).

Inbound MRd transaction going through SR-IOV EP VF BAR and iATU Example

- Inbound 64-bit MRd TLP Handling by QorIQ LS2088A PEX3 SR-IOV EP Example
  - Inbound MRd TLP to read 4 bytes at internal DDR address 0x0000_0000_8000_0000



Summary

- SR-IOV allows for different software images to share IO resources that exist on an EP device by introducing Virtual Functions (VFs) among a Physical Function (PF)
- With SR-IOV supported in EP, different devices or different software tasks can share IO resources such as Gigabit Ethernet controllers with minimum code modification
- The PCIe Controller 3 of QorIQ LS2088A Processor in SR-IOV EP mode supports 2 PFs and 64 VFs per PF (total of 128 VFs)
- An ARI and SR-IOV-aware software like SR-PCIM is essential to discover and configure both ARI and SR-IOV capabilities in RC and EP.
- The firmware running at QorIQ LS2088A Processor SR-IOV EP side should initialize the required PF and VF memory resources properly before setting the CFG_READY bit to accept configuration cycles from remote PCIe RC/host

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
- **PCI-SIG Developers Conference 2008 presentation:**
  - Single Root IOV, by Renato Recio (IBM)
- **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 2.1, March 04, 2009
  - PCI Express Base Specification, Revision 3.1, October 08, 2014
  - PCI Express Card Electromechanical Specification, Revision 3.0, July 21, 2013
  - Single Root I/O Virtualization and Sharing Specification, Rev 1.1

Software Products and Services

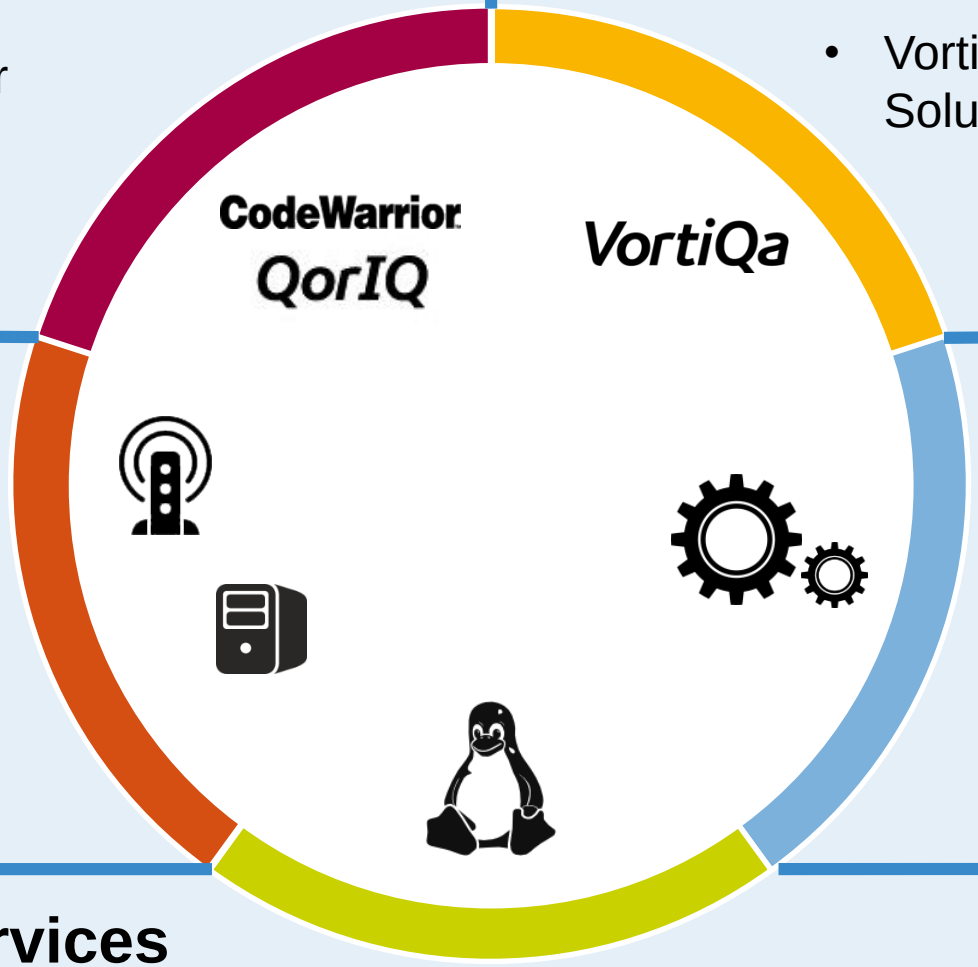
Visit us in the Tech Lab – #247

Development Tools

- CodeWarrior

Runtime Products

- VortiQa Software Solutions



Solutions Reference

- IOT Gateway
- OpenWRT+

Integration Services

- Security Consulting
- Hardened Linux

Linux® Services

- Commercial Support
- Performance Tuning

Accelerate **Customer Time-to-Market**

Deliver **Commercial Software, Support, Services and Solutions**

Simplify **Software Engagement with NXP**

Create **Success!**

Find us online at www.nxp.com/networking-services

SECURE CONNECTIONS
FOR A SMARTER WORLD

ATTRIBUTION STATEMENT