PCIe® and AMBA Ordering Case Study

Gary Dick
Staff Design Engineer
Cadence Design Systems

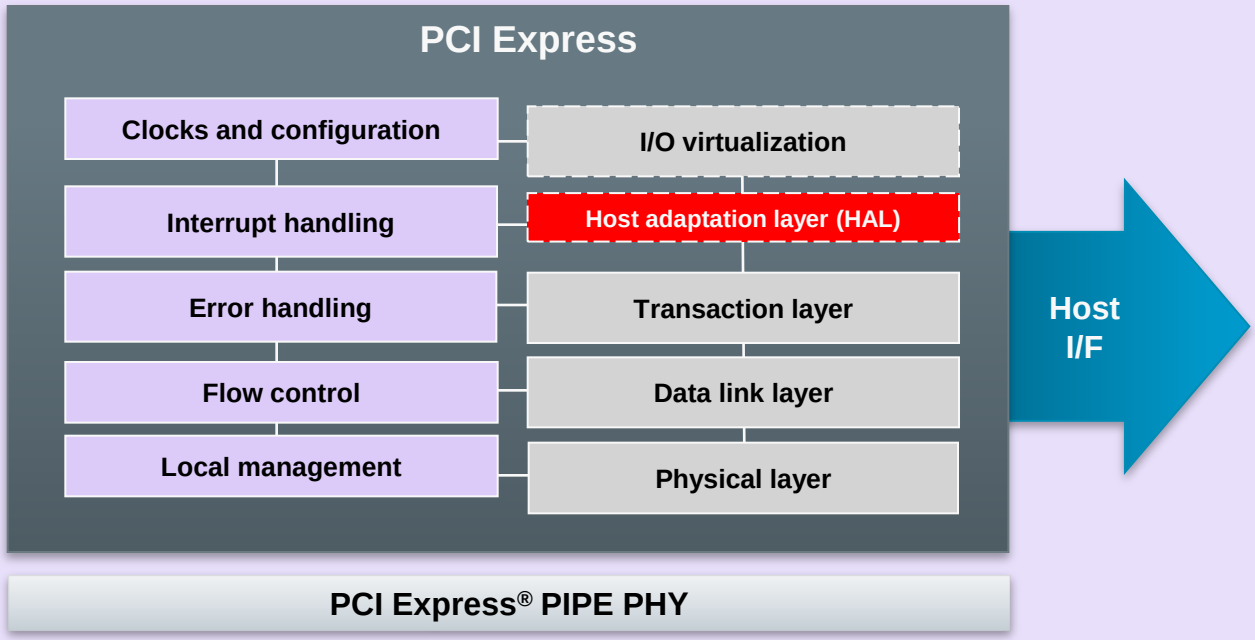
Agenda

- Background – why AMBA?*
- Architecture Considerations for AMBA-PCIe®*
- Ordering Rules
- Ordering Examples
- Ordering Walkthrough
- Ordering Solutions
- Summary

* Presented at ARM TechCon October 2012 by Ashwin Matta, Cadence

Background: Why AMBA?

- What are the typical interfaces to a PCIe controller if not AXI?
- Example: Host Adaption Layer (HAL) Interface



Background: HAL

- Provides simple PCI-like interface to clients
- Implements PCIe transaction ordering rules with TL layer
- Shields user from complexity of PCIe protocol
- Configurable datapath width:
  32, 64, 128, 256 bits
- High performance due to streaming behavior
- However, majority of SoC customers want a standard interconnect

Background: HAL v AXI

- Comparison of HAL v AXI Interface

Item	HAL	AXI
DataTransmit	• Streaming interface on TX side. • Assumes all data ready to be sent. • No store and forward.	• Store and Forward support
Clocking	• Application CLK = CORE_CLK • E.g. 500MHz for PCIe 3.0	• Asynchronous Clocking • Buffer handles Clock Boundary
Target Customers	• Devices without a local processor • Timing critical streaming data • Non-AMBA, synchronous systems	• SoC developers. • Existing AMBA based devices

Architectural Considerations

- **Key Point**
  - ✓ We are bridging between two inherently different protocols

PCIe Bus Interface	AXI3 Bus Interface
Packet-based: • Header + payload up to 4kB • Multiple address spaces that are dynamically assigned during enumeration • Different TLP types and other necessary attributes (eg. TC, Func #)	Burst-based: • Command + up to 16 data beats • Single fixed address with single burst type
Fixed frequency (125/250/500 MHz for PCIe 1/2/3)	Variable frequency depending on system implementation
PCIe spec has own ordering rules (posted/non-posted, RO, ID-based ordering)	Ordering rules for AXI bus coherency

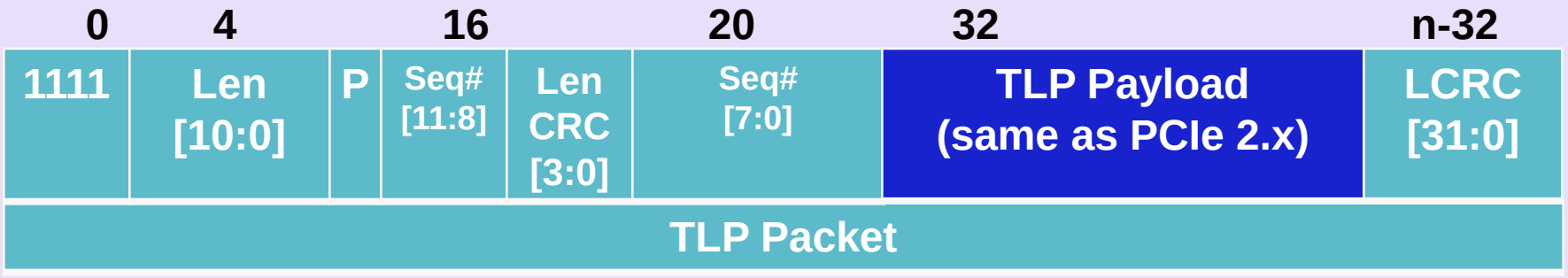
Architectural Considerations

The following section briefly reviews some of the main Architectural Considerations of AXI-PCIe, we then deep dive on Ordering Rules

1. Information Exchange Mechanism
2. Mapping of Information
3. Quantum of Data Transfer
4. Ordering Rules across two protocols

Information Exchange: PCIe

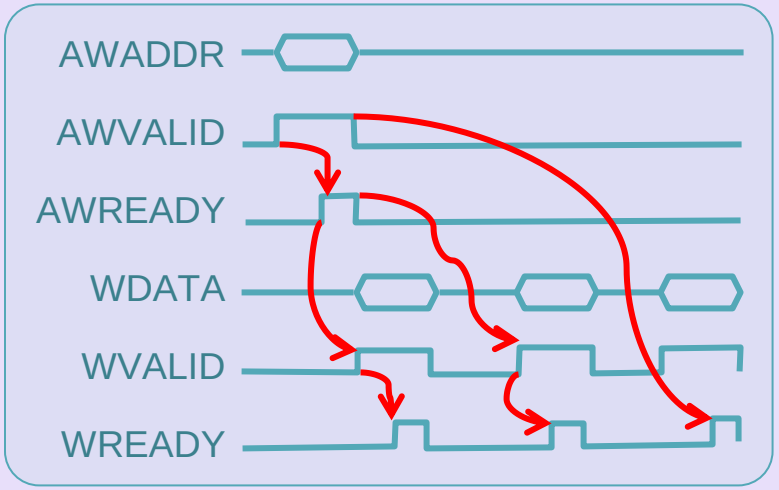
- PCIe is Packet Based, Layered Architecture



n is in multiples of 4 symbols=1DW

- Encapsulation of Header and Data
- Header in turn encapsulates all attributes of request
- Packets not always routed by address, e.g. Messages
- Each PCIe Layer adds error control fields to the Packet e.g. LCRC, ECRC
- Flow Control handled on a per-packet basis using flow control credits

Information Exchange: AXI



- AXI is burst based
- Pipelined protocol
- Split Request Response
- Request information presented on multiple parallel buses
- Flow Control handled by Valid and Ready Handshaking on a burst basis
  - ✓ Each request and data beat needs acknowledgment

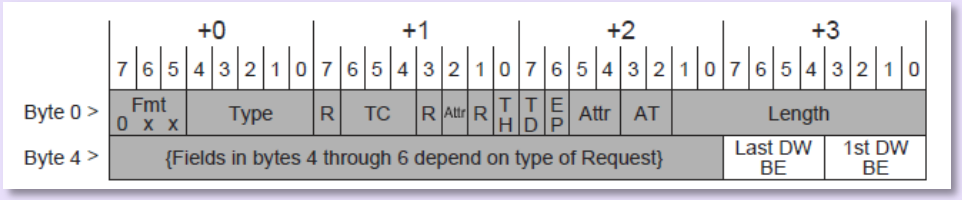
AXI-PCIe Requirements 1/4

- Storage of burst transactions from AXI interface before passing to cut-through HAL
- Encapsulation of separate AXI control signals into single PCIe Header (Tx)
- Disassembly of single PCIe Header into separate AXI control signals (Rx)
- Splitting of received PCIe requests as required to match AXI size limits

Mapping of Information

Example: Byte Enable Alignment in Tx direction

- PCIe has byte enable header fields
  - ✓ FBE[3:0]: First Byte Enable
  - ✓ LBE[3:0]: Last Byte Enable
  - ✓ DWORDS of data in between are considered to have a Byte Enable of 4'b1111



- AXI provides a write strobe for every 8 bits of Data on the DATA_WIDTH bus
- AXI could provide a 128bit stripe with only one byte enabled, bits[127:120]
- AXI-PCIe Device is required to form the PCIe Header & Data to correctly convey the first valid byte of data to the PCIe Link partner
  - ✓ AXI byte address translated to PCIe DW address and FBE pair
  - ✓ Data shifted to map AXI data to PCIe data
  - ✓ Write strobes converted to appropriate FBE/LBE combination

AXI-PCIe Requirements 2/4

- Data and Address translation
- Byte Enable conversion to / from LBE and FBE
- Different widths between AXI and PCIe need to be taken into consideration when designing conversion logic
  - ✓ PCIe datapath width driven by # of lanes, core clock frequency and transfer rates
  - ✓ AXI datapath width driven by client bus fabric configuration

Quantum of Data

Quantum of Data

- Maximum payload sizes are different
  - ✓ AXI 3 Maximum is 16 Beats x Data_Width
  - ✓ PCIe maximum is 4K Bytes, range is 128Bytes to 4KBytes for max_payload
- PCIe further complicated, for example
  1. Endpoint Device Supports 4K max payload
     - Advertised in Device **Capabilities** Register
  2. Switch above EP in PCIe hierarchy supports 256 Bytes
  3. Root Complex sets our Endpoint's max_payload to a value of 256 Bytes
     - Set in Device **Control** Register

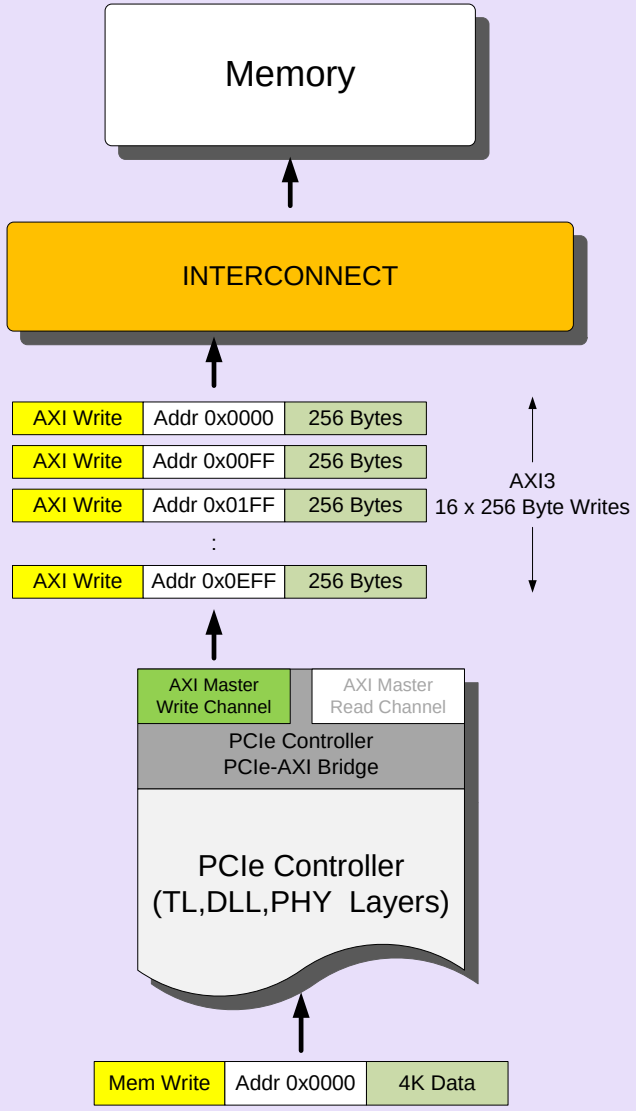
Quantum Example

Split Example – Inbound Write Request

- PCIe device supports 4K Byte max_payload
- AXI3 Supports Burst=16, Data_Width=32bits
- Incoming request from RC is 4K Mem Write

Bridge must split the write request

- Bridge creates multiple AXI write transactions
- AWID remains the same for each part of the write
- Addresses calculated by the bridge for each Write Request



AXI-PCIe Requirements 3/4

- Bridge needs to obey the PCIe max_payload value in PCIe Configuration space “Device Control Register”
- Bridge needs to obey the PCIe max_read_request_size value and not ask for more than this for Read operations
- Bridge must handle splitting TLPs into multiple requests on the AXI Master interface
- Suitable buffering must exist to handle the max payload
  - ✓ Care should be paid to actual max_payload
- Bridge changes with AXI4 support of up to 256 beats per transfer
  - ✓ Bridge must handle splitting of large outbound AXI4 transfer requests into multiple TLPs

Architectural Considerations

The following section briefly reviews some of the main Architectural Considerations of AXI-PCIe, we then deep dive on Ordering Rules

1. Information Exchange Mechanism
2. Mapping of Information
3. Quantum of Data Transfer
4. **Ordering Rules across two protocols**

Ordering Rules: AXI

Ordering Rules Summary: AXI

- Independent Read and Write Channels
- Strict ordering for same AxID per channel
- No ordering requirement between channels
- No ordering requirement for different AxID
- Quite straight forward

AXI Ordering
“The AXI protocol enables out –of – order transaction completion and the issuing of multiple outstanding addresses.”
“The ID signals support out-of-order transactions by enabling each port to act as multiple ordered ports”
“Transactions from different masters have no ordering restrictions. They can complete in any order.”
Transactions from the same master, but with different ID values, have no ordering restrictions. They can complete in any order.
Others...

Ordering Rules: PCIe

Ordering Rules Summary: PCIe

- Ordering rules are on 3 packet types:
  - ✓ Posted (E.g. Memory Write)
  - ✓ Non-Posted (E.g. Memory Read)
  - ✓ Completion (E.g. Read data)

Table 2-34: Ordering Rules Summary

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	NPR with Data (Col 4)	
Posted Request (Row A)		a) No b) Y/N	Yes	Yes	a) Y/N b) Yes
Non-Posted Request	Read Request (Row B)	a) No b) Y/N	Y/N	Y/N	Y/N
	NPR with Data (Row C)	a) No b) Y/N	Y/N	Y/N	Y/N
Completion (Row D)		a) No b) Y/N	Yes	Yes	a) Y/N b) No

- Ordering rules apply between packets of the same type and between packets of differing types
- Relaxed ordering and ID-based ordering in attribute fields can also affect ordering
- Traffic Class and Virtual Channel mapping affect ordering

Ordering Considerations

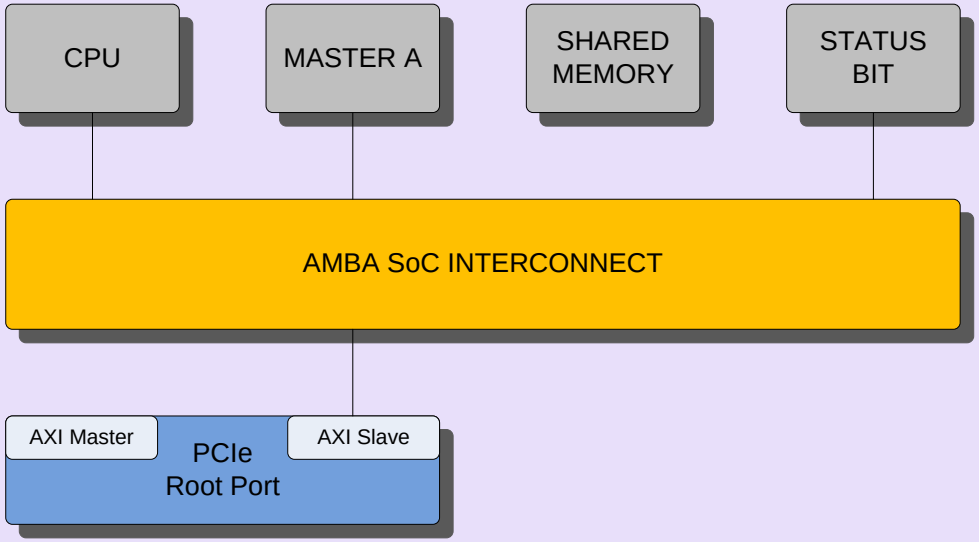
The following sections highlight some of the key Ordering Considerations of AXI-PCIe.

1. Inbound: Write after Write (WAW)
2. Inbound: Read after Write (RAW)
3. Outbound: Basic Read Ordering
4. Outbound: Read after Read (RAR)
5. Outbound: Read after Write (RAW)
6. Outbound: Mem Write after Config Write (WACR)

Thanks and acknowledgement to ARM for collaborating on these ordering scenarios

Ordering Example 1: WAW

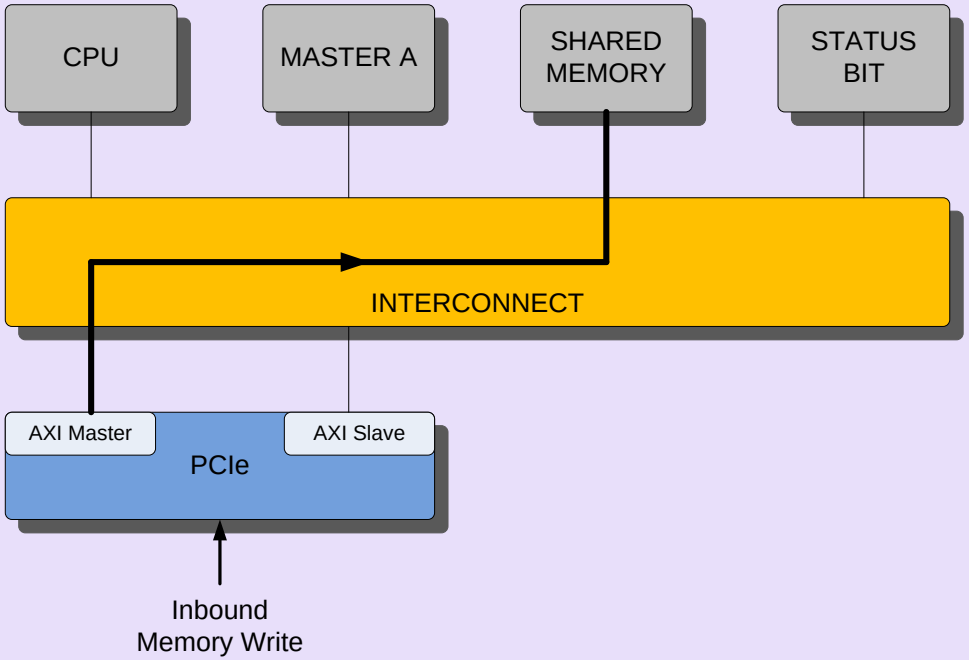
- WAW: Write after Write
- Inbound Scenario:
  - ✓ Endpoint device is sending a number of Memory Writes to Host Memory
  - ✓ Memory Writes are all related to each other
  - ✓ When all Memory Writes are complete the Endpoint sets a status bit to indicate “Writes Complete”
  - ✓ Status write could be MSI – a Memory Write
  - ✓ MASTER A is snooping the Status Bits, waiting for “Writes Complete”
  - ✓ When Compete MASTER A will read data from Memory



- Q.) How we do ensure MASTER A gets all the data it requires?
- ...let us firstly describe what could go wrong!

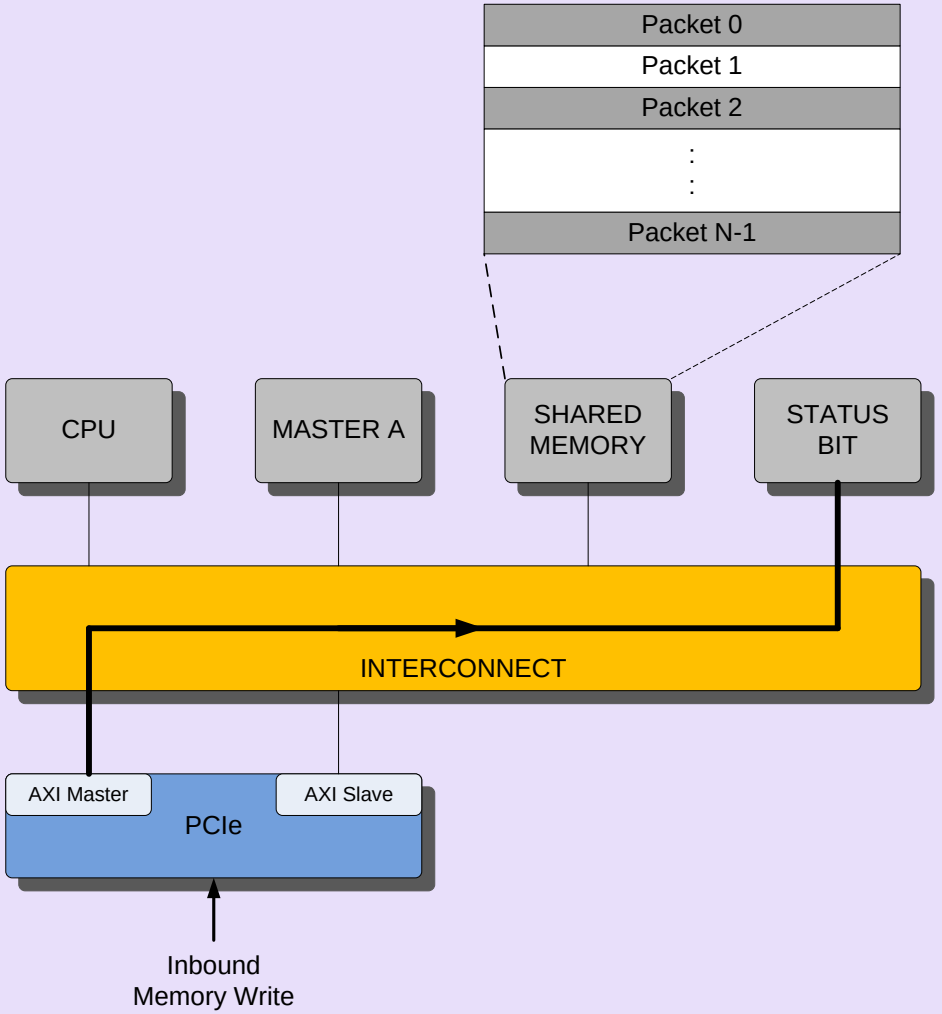
Ex. 1 Walkthrough: Step 1/3

- Memory Writes
  - ✓ Memory Writes received from Link Partner
  - ✓ PCIe-to-AXI Bridge in Controller converts to AXI Master Write
  - ✓ Interconnect Routes Memory Write to Shared Memory
  - ✓ Repeated for all Memory Writes
- Notes:
  - ✓ PCIe bridge handles max_payload protocol differences



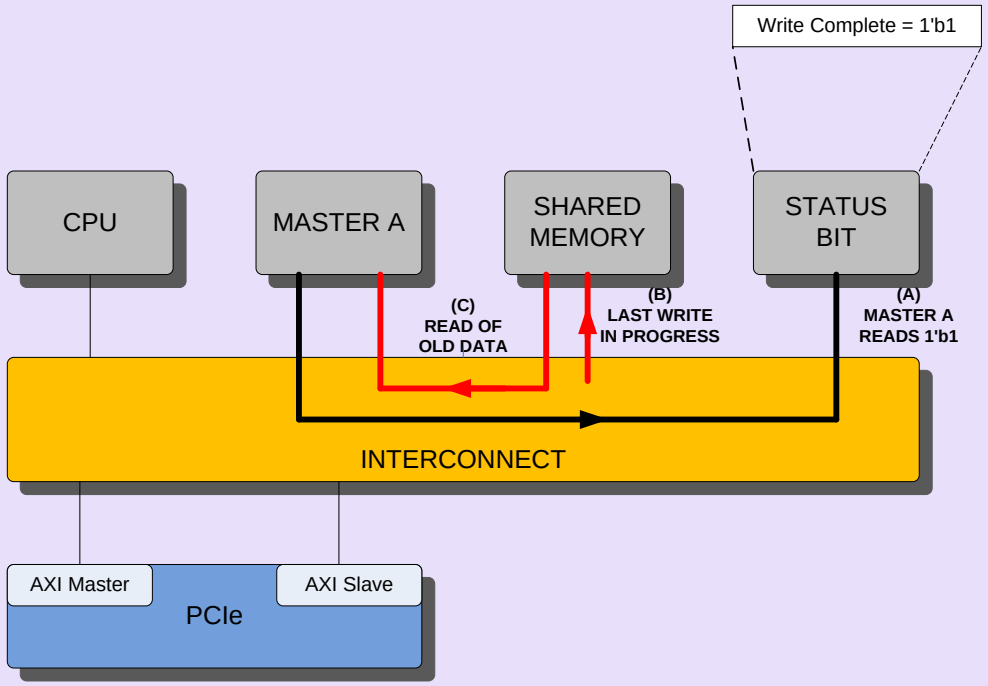
Ex. 1 Walkthrough: Step 2/3

- Status Bit Write
  - ✓ Send the final Memory Write
  - ✓ Now set the “Write Complete” bit in Host using Memory Write
  - ✓ Could be MSI, a Memory Write to an agreed Host Address
  - ✓ Interconnect has passed the BRESP responses back to PCIe Controller in the correct order



Ex. 1 Walkthrough: Step 3/3

- MasterA “snoops” Status Bit (A)
  - ✓ Polls the status bit
  - ✓ Once it reads a “1” is starts to read the data from Host Memory
- Issue:
  - ✓ Q.) What if the last Memory Write is still taking place (B)
  - ✓ **A.) MASTER A could read old data from Memory (C)**

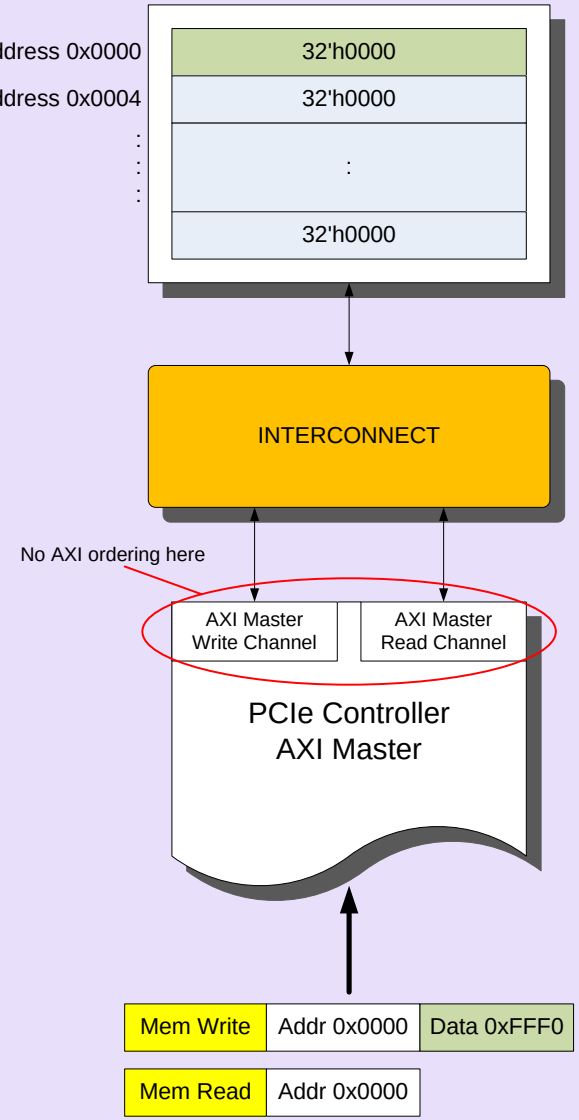


WAW Solutions for PCIe IP

- Serialize all Memory Writes on AXI Master Interface
  - ✓ Send Memory Write and wait for BRESP before sending the next Memory Write
  - ✓ Pros: Ensures Write after Write Observability by all
  - ✓ Cons: Bandwidth Inefficient
- Understand AMBA Address Map
  - ✓ Only Serialize Writes when they are not to the same address region, e.g. 4k boundary
  - ✓ Use the same AWID for the same region
  - ✓ Writes within the same region (Slave) are not serialized, when we switch to a new region, controller waits for BRESP for all Writes

Ordering Example 2: RAW

- RAW: Read after Write
- Inbound Scenario:
  - ✓ EndpointA device sends a Memory Write to Root PortA
  - ✓ EndpointA device sends a Memory Read to Root PortA
  - ✓ AXI would traditionally handle these requests on two independent channels (Write Channel and Read Channel) – which have no ordering rules between
  - ✓ PCIe Producer/Consumer models dictates that the Non-Posted request must not overtake the Posted request
    - AXI rules do not natively ensure this
- Q.) How do we ensure that the PCIe ordering rules are followed?



Ex. 2 Walkthrough: RAW

- PCIe Controller IP must enforce ordering rules across the AXI channels
- Write Request is driven on AXI Write Channel
- Write Data is read out from buffer and passed to AXI Write Channel
- AXI bridge waits for BRESP in response to AXI Write before it sends the Read Request on AXI Read Channel
- Only when the BRESP is received does the PCIe Controller start the Read Request on the AXI Master Read Channel

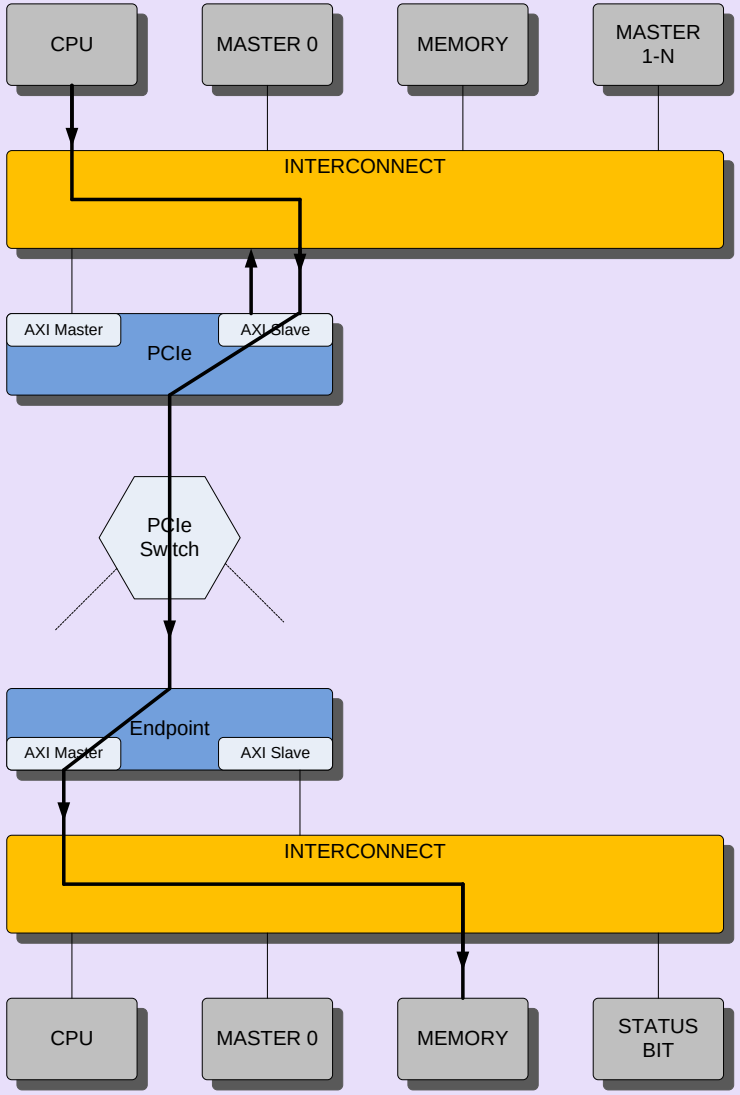
RAW Solutions for PCIe IP

- PCIe Controller IP waits for BRESP for all Writes
  - ✓ Before it sends a Memory Read Request on the AXI Read Channel
  - ✓ Required for Read after Write observability
- Service Received Posted Requests ahead of Received Non Posted Requests
  - ✓ PCIe Controller will store received Posted and Non-Posted Requests in different SRAM partitions
  - ✓ Non-Posted requests will only be passed to AXI Read Channel when there are no complete Posted Requests in the buffer

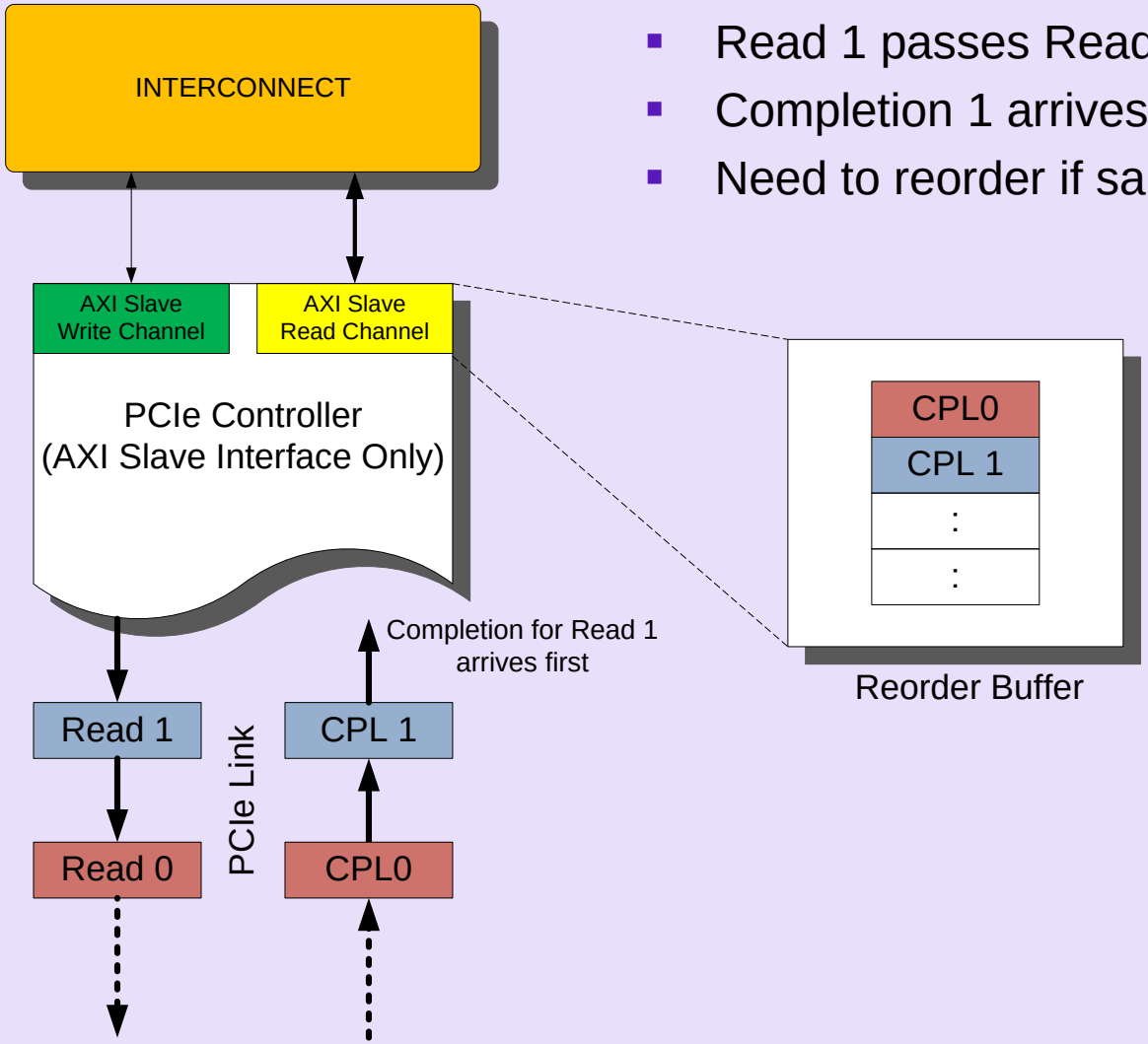
Ordering Example 3: Basic Read Data Ordering

OUTBOUND:

- AXI Master performs two Read Requests using the same ARID=0
- AXI ordering rules state that the read data must return in the same order as requested by the same ID
- PCIe will create two separate Non-Posted requests, one for each AXI Read
  - ✓ PCIe will assign a TAG identifier to each request, e.g. TAG0, TAG1
- PCIe ordering rules state that a NP can pass an NP
  - ✓ **Read Data 1 could arrive on PCIe before Read Data 0**
- Bridge cannot simply pass back the data as it arrives!



Ex 3: Walkthrough



- Read 1 passes Read 0
- Completion 1 arrives before Completion 0
- Need to reorder if same ARID

Read Reorder Solutions for PCIe IP

- NP may pass an NP
  - ✓ May happen at any point in the PCIe hierarchy
  - ✓ i.e. In PCIe devices other than your own

Row Pass Column?		Posted Request	Non-Posted Request		Completion
			Read Request	NPR with Data	
Posted Request		a.) No b.) Y/N	Yes	Yes	a.) Y/N b.) Yes
Non-Posted Request	Read Request	a.) No b.) Y/N	Y/N	Y/N	Y/N
	NPR with Data	a.) No b.) Y/N	Y/N	Y/N	Y/N
Completion		a.) No b.) Y/N	Yes	Yes	a.) Y/N b.) No

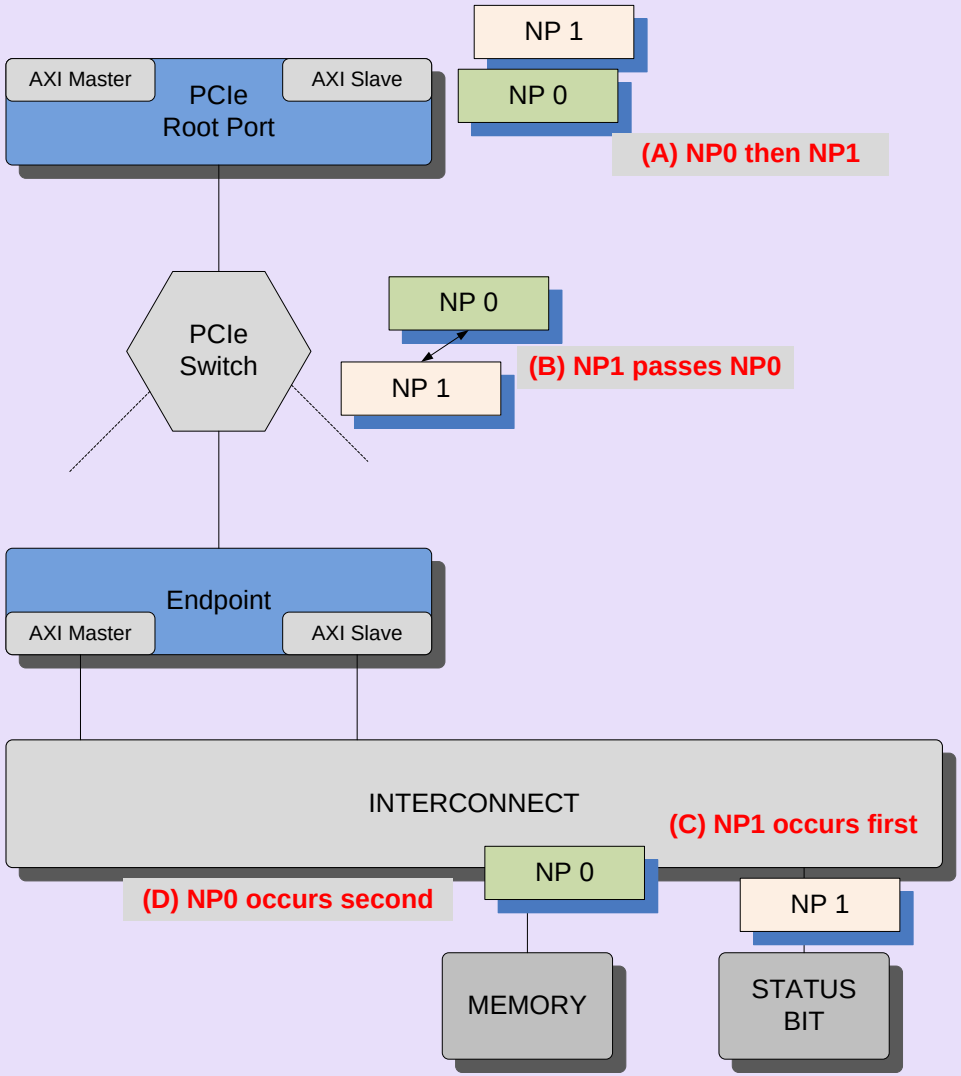
- The PCIe-AXI bridge needs to be aware of this possible reordering, however unlikely
- We need to consider worst case scenarios
  - ✓ Lets say our Controller can have 32 Max Outstanding NP requests: NP0 to NP31
  - ✓ In theory the Read data (Completion) for NP31 could return first
  - ✓ PCIe-AXI Bridge must handle this worst case scenario and wait for NP0 to return and pass this back to the AXI Read Channel first
  - ✓ Re-Order logic required for Completions

Ordering Example 4: RAR

- **Outbound Read after Read**
- Previous Read Reordering Example does not tell the whole story for AXI
  - ✓ Yes, PCIe-AXI Bridge reorders Read Data Completions
  - ✓ Yes, PCIe-AXI Bridge provides support for worst case buffering
  - ✓ However...how can we guarantee that the Read Data is Valid?
- NP0 and NP1 are sent by our PCIe device
  - ✓ NP1 overtakes NP0 at some point
  - ✓ NP1 is executed at the remote end before NP0 is executed
  - ✓ PCIe-AXI bridge will reorder Completion, but does not take into account that the actual Read Operations were executed in the wrong order

Ex.4 Walkthrough

- A more realistic example would be a PCIe device performing a number of Memory/FIFO Reads
- The Last Read is followed by a Read of a Status Register
- Here we show two Reads for simplicity
  - ✓ NP0 = Read from Memory
  - ✓ NP1 = Read from Status bit
- PCIe Ordering rules allow NP1 to pass NP0
- At the AXI interface of the Root Port, we Read Data for the Status Bit Read (NP1) returns 1'b1 suggested we still have data to read



RAR Solutions for PCIe IP

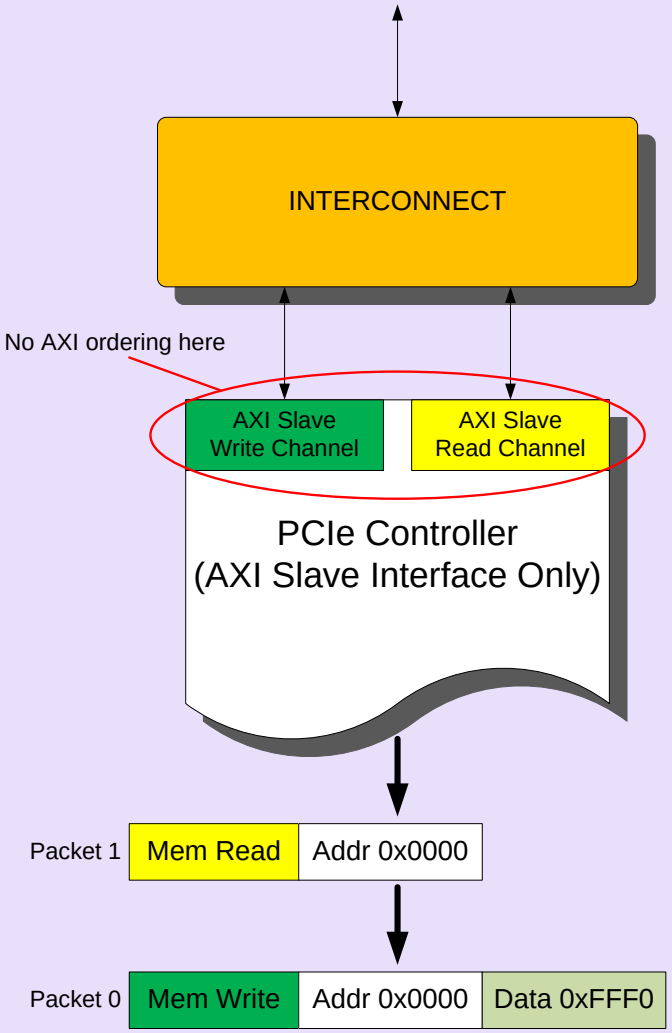
- PCIe Controller IP allows Reads to be ‘Serialized’
  - ✓ Option 1: Maximum of one NP in-flight at a time
    - Only allow one outstanding NP request at a time (Tag 0)
    - Pros: Ensures Read after Read observability
    - Cons: Bandwidth inefficient (Extended TAG allows 256 in-flight NP and customers use this)
  - ✓ Option 2: Per packet ‘Serialize’ Flag
    - Read request can have a flag set to ensure it is serialized, e.g. “For this NP request make sure that all previous NP requests have been completed”
      - Can use TAG lookup to determine flags in flight
      - Need to also hold off subsequent NPs
    - Pros: Ensures Read after Read observability
    - Cons: Bandwidth inefficient, but not as bad as Option 1

Ordering Example 5: RAW

- **Outbound Read after Write**
- AXI does not support ordering between AXI Read and AXI Write transactions
- For Read after Write we want to ensure that the Write transaction arrives at the Endpoint before the Read
  - ✓ PCIe Ordering rules do not allow an Non Posted Request to pass a Posted Request
- To ensure Read after Write the PCIe Controller IP must coordinate responses back to AXI versus data passed to the PCIe Link

Ex 5: Walkthrough RAW

- AXI Master controls when it makes a request on the PCIe Controller AXI Slave interface
- To ensure Read after Write
  - Write request made on controller's AXI Slave Write Channel
  - Write data passed to controller's AXI Write Channel
  - ...wait for BRESP!
  - Read Request made on AXI Read Channel
- BRESP only returned when we are "past the point of serialization"



RAW Solutions for PCIe IP

- PCIe Controller IP coordinates BRESP
  - ✓ BRESP returned by controller only when we can guarantee that the Posted Write will be sent over the controller's PCIe Link
    - i.e. it is in a Pipeline through the Protocol Layers and can't be overtaken before it reaches the Serial Link
    - Once on the PCIe Link, PCIe ordering rules will ensure that subsequent Non-Posted requests do not overtake it
  - ✓ CPU waits for BRESP before making Read request on the Read Channel

- If we didn't wait for BRESP?
  - ✓ Until then, Write data is being stored in an Asynchronous Buffer
  - ✓ A Read request made on the AXI Read Channel during this time could get sent ahead of Write as Write is not complete

Ordering Example 6: WACW

- **Outbound Write after Config Write**
- A blocked Config Write (NP) should not block a subsequent Posted Write
- For AXI, both a Config Write and a Posted Write use the same AXI Slave Write Channel
- Outbound Config Write could be blocked by
  - ✓ Lack of available TAGS (E.g. we have 32 Memory Reads in flight)
  - ✓ Lack of NP Credits from Link Partner
- The Posted Write should pass the blocked Config Write

Ex 6: Walkthrough part 1/2

- Root Port performs 32 Non Posted Config Read Requests – on the AXI Slave Read Channel
- Root Port performs 1 Non Posted Config Write Request – on the AXI Slave Write Channel
  - ✓ Using AWID=0
- Config Write gets blocked due to lack of NP credits
- Next Request is a Posted Write using AWID=0
  - ✓ It is stuck being the blocked Configuration Write
  - ✓ PCIe ordering rules state that Posted must pass Non-Posted

Ex 6: Walkthrough part 2/2

- Q.) How can we let the Posted pass?
- A.) Move Configuration Write to a “set-aside buffer” and let Posted Write pass?
  - ✓ We need to free up the AXI Slave Write Channel
  - ✓ We still have to send back the BRESP and BID in the correct order, in this case BID=0 for both requests
  - ✓ Once Posted Write is sent we can store the BRESP
  - ✓ When NP credits free up we can then send the Configuration Write
  - ✓ We can now send back two BRESP for BID=0 (AWID=0)

WACW Solutions for PCIe IP

- Implement a set-aside buffer to temporarily store blocked NP Write requests
  - ✓ Need to store BID for block NP Writes
  - ✓ Compare AWID for NP Write to Posted Write AWID
    - This dictates BID and BRESP
- Provide NP set-aside count status registers to the Client logic in order for Client to decide whether to send a further NP Config Write request
- Decisions?
  - ✓ How deep is the set-aside buffer?
  - ✓ How many Posted Writes can pass the set aside buffer?
    - How many BRESP responses do we need to store?

AXI-PCIe Requirements 4/4 Ordering Summary

- PCIe Controller IP providers have to think ‘outside-the-PCIe-box’
  - ✓ Knowledge of the AXI signalling alone is not sufficient
  - ✓ Knowledge of what sits behind the AXI interface is required
  - ✓ More specifications to read...
- IP must be flexible in order to maximize bandwidth efficiency whilst meeting AXI and PCIe ordering rules
  - ✓ Allow IP users to control when they implement Serialization

Thank you for attending the PCI-SIG Developers Conference

Israel 2013

For more information please go to www.pcisig.com