PCIe Interoperability and Optimization Proposal for Ncore 3.2.1

Table of Contents

Table of Tables

Assumptions

PCIe supports multiple interleaved transactions, sharing a common transport medium. The different transaction types are listed in table 1 (Transaction Type column). The PCIe root complex in an SoC would convert these transactions into ACE-Lite protocol when interfacing to an Ncore 3 IOAIU.
Depending on the address space selected and the semantics of the transaction, Ncore 3 would then propagate these to one of 3 types of functional interfaces:

- coherent memory --> DCE
- non-coherent memory --> DMI
- non-coherent memory and device transactions --> DII
- The PCIe IP ACE-Lite bus master will only use ReadNoSnoop, WriteNoSnoop, ReadOnce and WriteUnique transactions

Table 1: Expected PCIe Transaction to AXI mapping

Item	Transaction Type	Mode	Target	ACE-Lite Transaction - AxCACHE[3:0]
1	Memory Read	Non-Posted	DCE[1]	normal, cacheable, bufferable
2			DMI	normal[2]
3			DII	normal, non-cacheable[3]
4	Memory Write	Posted	DCE	normal, cacheable, bufferable
5			DMI	normal, non-cacheable, bufferable
6			DII	normal, non-cacheable, bufferable/non-bufferable[4]
7	Memory Read Lock[5]	Non-Posted	DCE/DMI	normal, non-bufferable
8			DII	normal, non-cacheable, non-bufferable
9	IO-Read	Non-Posted	DII	device, non-bufferable
10	IO-Write	Non-Posted	DII	device, non-bufferable
11	Configuration Read (Type 0/1)	Non-Posted	DII	device, non-bufferable
12	Configuration Write (Type 0/1)	Non-Posted	DII	device, non-bufferable
13	Message	Posted	DII	device, non-cacheable, non-bufferable
14	AtomicFetchAdd[6]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable
15	AtomicSwap[6]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable
16	AtomicCAS[6]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable

ARM protocol specifies certain behavior - which, for certain sequences requires some attention to avoid deadlocks, especially if traffic arriving from one PCIe device will be forwarded to a different PCIe device (bridge functionality).

Response:

- What AXI/ACE-Lite transaction encoding/attribute setting are MSI transactions using
  a. MSI Transactions are "Memory" type transactions - an MSI Write would be a "Memory Write"
  b. The bufferability attribute is generated (in default condition) based on the NS bit on the PCI link packets.

     If NS[7] = 1, ACELite = Non-Bufferable

     if NS = 0, ACELite = Bufferable
- What encoding/attribute setting distinguishes posted from non-posted memory writes

  Posted PCIE link Writes are always generated with AXI-ID = 0

  Non-Posted PCIE link Writes are always generated with AXI-ID != 0

[1] All cacheable memory accesses are bufferable
[2] Technically any normal memory access is supported
[3] Cacheable transactions are not expected
[4] Both transaction types are legal - SNPS IP uses non-bufferable transactions
[5] AXI4 does no longer support locked transactions
[6] Our ACE-Lite implementation supports atomics but atomics must be cacheable and will not terminate properly if the addressed region is not backed up by SMC or if the region is not addressing scratch pad memory
[7] NS = non-snoop property of PCIe TLP

ACE-Lite Transactions generated by Synopsys PCIe Controller with AMBA Bridge

The controller supports the AXI4 protocol with additional support for the ACE-Lite protocol. To enable these features, you must set AMBA_INTERFACE =3 and CC_ACELITE_ENABLE =1.

ACE-Lite Features and Limitations:

- The AXI bridge slave does not implement ACE; it ignores the value of the DOMAIN, BAR and SNOOP fields; barrier transactions are not supported
- The AXI bridge master interface generates requests that are dependent upon the NS and TH bits
- The master interface always sets AxBAR and AxSNOOP to '0' so that requests are restricted to ReadNoSnoop, WriteNoSnoop, ReadOnce, and WriteUnique
- You can override the value of the DOMAIN field, except for message requests where it is always 11
- The master does not support barriers
- Cache coherency is not optional, and the master always uses the NS bit, unless you override the DOMAIN signals

Table 2: ACE Lite Transactions - SNPS PCIe IP

Cache Coherency	**Request Type**	**Memory Type[8] 0: Peripheral 1: Main Memory**	**NS[9]**	**TH**	**AxSNOOP**	**AxBAR**	**AxDomain**	**Shareability Domain**	**Memory Type**
Non-coherent	Message	X	X	X	4b0000	2b00	2b11	System	Device, Non-bufferable
Non-coherent	Read or Write	0	X	X	4b0000	2b00	2b11	System	Device, Non-bufferable
Non-coherent	Read or Write	1	1	X	4b0000	2b00	2b11	System	Normal, Non-cacheable Non-bufferable
Coherent	Read	1	0	0	4b0000	2b00	2b10	Outer Shareable	Writeback, No-allocate
Coherent	Write	1	0	0	4b0000	2b00	2b10	Outer Shareable	Writeback, No-allocate
Coherent	Read	1	0	1	4b0000	2b00	2b10	Outer Shareable	Writeback, Read-allocate
Coherent	Write	1	0	1	4b0000	2b00	2b10	Outer Shareable	Writeback, Write-allocate

[8] Set by the CFG_MEMTYPE_VALUE field of the COHERENCY_CONTROL_1_OFF register

[9] NS = non-snoop property of PCIe TLP

This table has been copied from SNPS PCIe IP Databook, Table 7-82 in section 7.4.2

Item	PCIe Transaction Type	Mode	Target	ACE-Lite Transaction - AxCACHE[3:0]
1	Memory Read	Non-Posted	DCE[10]	normal, cacheable, bufferable
2			DMI	normal
3			DII	normal, non-cacheable
4	Memory Write	Posted	DCE	normal, cacheable, bufferable
5			DMI	normal, non-cacheable, bufferable
6			DII	normal, non-cacheable, bufferable/non-bufferable
7	Memory Read Locked[11]	Non-Posted	DCE/DMI	normal, non-bufferable
8			DII	normal, non-cacheable, non-bufferable
9	IO-Read	Non-Posted	DII	device, non-bufferable
10	IO-Write	Non-Posted	DII	device, non-bufferable
11	Configuration Read (Type 0/1)	Non-Posted	DII	device, non-bufferable
12	Configuration Write (Type 0/1)	Non-Posted	DII	device, non-bufferable
13	Message	Posted	DII	device, non-cacheable, non-bufferable
14	AtomicFetchAdd[12]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable
15	AtomicSwap[6]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable
16	AtomicCAS[6]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable

[10] All cacheable memory accesses are bufferable

[11] AXI4 does no longer support locked transactions

[12] Our ACE-Lite implementation supports atomics but atomics must be cacheable and will not terminate properly if the addressed region is not backed up by SMC or if the region is not addressing scratch pad memory

The rules for PCIe transactions (see PCIe specification, Section 2.4 Transaction Ordering)

Table 3 defines the ordering requirements for PCIe Transactions. the columns represent a first issued transaction and the rows represent a subsequently issued transaction. The table entry indicates the ordering relationship between the two transactions. The table entries are defined as follows:

Yes The second transaction (row) must be allowed to pass the first (column) to avoid deadlock. (When blocking occurs, the second transaction is required to pass the first transaction. Fairness must be comprehended to prevent starvation.)

Y/N There are no requirements. The second transaction may optionally pass the first transaction or be blocked by it.

No The second transaction must not be allowed to pass the first transaction. This is required to support the producer/ consumer strong ordering model.

Table 3: PCIe 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 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

Posted Request is a Memory Write Request or a Message Request.
Non-Posted Request (with Data) Configuration Write Request, an I/O Write Request, or an AtomicOp Request.
Non-Posted Request Read Request or an NPR with Data

Row A, B, C Special cases:
a) applies to all transactions unless condition b) applies
b) Requests are allowed to pass when
- RO (relaxed ordering attribute) is set
- IDO (ID ordering) applies and the Requester IDs are different
- Both requests have been issued using different PASID

Row D 2a) Completion must not pass a Posted Request
2b) Exception:
- An I/O or Configuration Write Completion is permitted to pass a Posted Request
- Completions with IDO are permitted to pass a Posted Request if they belong to independent flows (C.ID <> R.ID)

5a) Completions with different Transaction IDs are permitted to pass each other
5b) Completions with the same Transaction ID are not allowed to pass each other

The rules for AMBA transactions (see ARM IHI 0022G, Section A6.3 Transactions and Ordering)

A transaction is a read or a write to one or more address locations. The locations are determined by **AxADDR** and any relevant qualifiers such as the Non-secure bit in **AxPROT**.

- Ordering guarantees are given only between accesses to the same Memory location or Peripheral region when the same AxID is used.
- A transaction to a Peripheral region must be entirely contained within that region.
- A transaction that spans multiple Memory locations has multiple ordering guarantees.

Transactions can be either of type Device or Normal:

Device: Device transactions can be used to access Peripheral regions or Memory locations. In Ncore 3.0 architecture, these transactions will be forwarded to DII.

Normal: Normal transactions are used to access Memory locations and are not expected to be used to access Peripheral regions.

A Normal access to a Peripheral region must complete in a protocol-compliant manner, but the result is IMPLEMENTATION DEFINED.

A write transaction can be either Non-Bufferable or Bufferable. It is possible to send an early response to Bufferable writes.

Guarantees before a completion response is received

In AMBA AXI/ACE-Lite protocol the transaction ID identifies a "flow" of associated transactions, originating at a requester agent and terminating at a target agent. Since the protocol does not identify individual transactions, ordering rules for transport, execution and responses are required to maintain consistency and to associate the request with a response.

All of the following guarantees apply to transactions from the same master, using the same ID:

- A Device write DW1 is guaranteed to arrive at the destination before Device write DW2, where DW2 is issued after DW1 and to the same Peripheral region.
- A Device read DR1 is guaranteed to arrive at the destination before Device read DR2, where DR2 is issued after DR1 and to the same Peripheral region.
- A write W1 is guaranteed to be observed by a write W2, where W2 is issued after W1 and to the same Memory location.
- A write W1 that has been observed by a read R2 is guaranteed to be observed by a read R3, where R3 is issued after R2 and to the same Memory location.

The guarantees imply that there are ordering guarantees between Device and Normal accesses to the same Memory location.

Guarantees from a completion response

A completion response guarantees all of the following:

- A completion response to a read request guarantees that it is observable to a subsequent read or write request from any master.
- A completion response to a write request guarantees that it is observable to a subsequent read or write request from any master. This observability is a requirement of a system that is multi-copy atomic.

Systems that contain Arm Architecture-compliant processors must be multi-copy atomic. That is, the Multi_Copy_Atomicity property must be ***True***.

The response to a Bufferable write request can be sent from an intermediate point. It does not guarantee that the write has completed at the endpoint, but it is observable to future transactions.

Response ordering guarantees

Transaction responses have all the following ordering guarantees:

- A read R1 is guaranteed to receive a response before the response to a read R2, where R2 is issued from the same master after R1 with the same ID.
- A write W1 is guaranteed to receive a response before the response to a write W2, where W2 is issued from the same master after W1 with the same ID.

What rules do we follow in IOAIU?

Current uArch specification

- Maintain separate AxID linked lists for read and writes, responses for same AxID will be in order within a read or write channel
- Generate Ordering bits based on AxCache[3:1] values for transactions with same AxID
    - 0: WriteNonCohFull - transaction with no older dependent transaction with same AxID (separate list for reads and writes)
    - 2: if (AxCache[3:1] > 0) (Normal) and there is an older transaction with same AxID (separate list for reads and writes)
    - 3: if (AxCache[3:1]==0) (Device) and there is an older transaction with same AxID (separate list for reads and writes)
- Add a CSR bit to run in producer consumer mode
    - In this mode IO AIU will block same AxID transactions until the STR response is sent for the dependent transaction (separate list for reads and writes)
    - When mode is not enabled then blocking is not required
- Same cache line address blocking until STR response is sent. This is done across reads and writes to avoid hazards (WAW, WAR, RAW)
- Writes will have at-least one or more reserved OTT entries and always make forward progress form IO AIU perspective.

The impact of the uArchitectural rules applied to PCIe transaction sequences

- The current implementation of Ncore does not support the concept of posted requests - order is enforced between all writes with the same AxID
    - 'non-posted' writes will not pass 'posted' writes (C2a) - **required**
    - 'non-posted' writes will not pass 'non-posted' writes (C4) - **less optimal, but legal**[13]
    - 'posted' writes will not pass other 'posted' writes (A2a) with the same ID or address - **required**
    - 'posted' writes will not be able to pass 'non-posted' writes (A4a) with the same ID - in principle this violates the requirement but:
        a. If posted writes use a different ID than NPRs, this problem goes away[14]
        b. Ncore guarantees forward progress for writes vs. reads - posted writes will make forward progress[15]
        c. Is having no posted writes a problem with performance? - **Yes**
    - relaxed ordering is implied by not using the same ID (A2b, A4b, C2a, C4) - **allowed**
- Read requests will pass all write requests, except if a hazard exists
    - read requests must not pass 'posted' write requests with the same ID and the same address[16] (B2a) - **implemented** (we respect WAR, RAW, WAW order, independent of AxID)
    - read requests may pass 'posted' write requests with a different ID (B2b) - **allowed**

Notes:

1. This are AXI rules - we allow to relax them for memory targets by address region (GPRAR registers)
2. Strict request ordering based on the AxID - but performance suffers
3. Yes, a single open entry does not look like much but, writes will always have priority over reads, so when OTT is full, a write will get in for every returning transaction (pipelining)
4. This is not explicitly stated, but inferred from memory consistency and R-W ordering requirements - reads will always return the data written by a write that has been issued earlier.

---

[13] This are AXI rules - we allow to relax them for memory targets by address region (GPRAR registers)

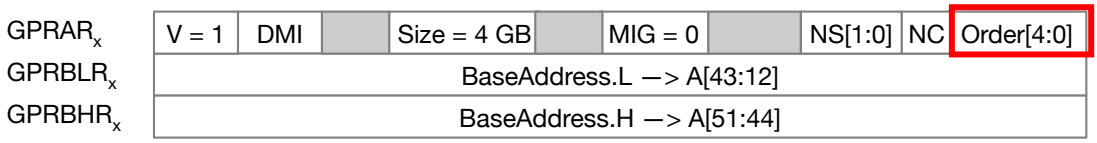
[14] Strict request ordering based on the AxID - but performance suffers

[15] Yes, a single open entry does not look like much but, writes will always have priority over reads, so when OTT is full, a write will get in for every returning transaction (pipelining)

[16] This is not explicitly stated, but inferred from memory consistency and R-W ordering requirements - reads will always return the data written by a write that has been issued earlier.

Performance improvement discussion

- Prefer to define ordering by address region (CMN does it that way) instead of target FUnit, each region is defined by a GPRAR



- New order control field in each GPRAR$_x$
- The control field (bits 0.. 2) within this register controls the rules when a transaction may be **issued** to the internal target (DCE/DMI or DII)[17]
- The **Policy** field determines the behavior at the target agent - DCE and DMI ignore the policy setting - and the response behavior
- Device transactions (AxCACHE[3:1] == 3b000) will enforce strict request ordering (Policy == 2b11) for all requests with the same AxID - these shall only be used for targets behind a DII.
- Memory request addressing DII targets may use stricter rules (Request order)

Table 4: CSR Configuration

Order	Function	Description
0	Reserved	Always read as zero
1	ReadID (ARID)	No ordering by AxID (free listing) for reads[18]
2	WriteID (AWID)	No ordering by AxID (free listing) for writes[15]
4:3	Policy 11 Endpoint Order 10 Relaxed Order 00 Reserved 01 Write Order	Policy determines the setting of our internal OR[1:0] field which defines the behavior of the agent at the bottom of Ncore Endpoint[19] order: ▪ All transactions to the same peripheral region are issued and completed in order ▪ Override the ordering mode of the incoming transaction to *Endpoint Order* Relaxed order[20]: ▪ Responses within a flow (same AxID) are ***ordered*** if ReadID/WriteID bits are zero, otherwise they are ***unordered*** ▪ ***Ordered*** transactions will be issued as soon as the previous transaction in the ordered sequence has been ordered at the target (StrReq has been returned) and command credits are available ▪ ***Unordered*** transactions will be issued as soon as command credits are available at the target ▪ Write: BRESP may be returned (earliest) as soon as DtwRsp has been received, following AXI ordering rules (all older responses have been sent) Write order: ▪ This mode has been added in version 3.2.1 to allow writes to follow PCI rules (A2a, C2a) at the DII Reserved: ▪ Shall be treated as Endpoint order (the same behavior as Policy = 11)

[17] Read/Write ordering rules may be different for each target region - Setting ReadID/WriteID field overrides AxID ordering!
[18] Setting ReadID and WriteID (freelisting) will force the internal OR[1:0] field to 2b00 (unordered) for memory targets
[19] Read: Issue --> DtrReq, Response --> DtrReq --- Write: Issue --> DtwRsp, Response --> DtwRsp. Setting ReadID or WriteID bits is not recommended for this policy and may result in undefined behavior
[20] The equivalent of ***Strict Response*** ordering can be achieved by using ***Relaxed Order*** and setting the ReadID and WriteID bits

Recommended Settings

Table 5: Recommended Settings

AxCACHE[3:0]	Target	Policy [1:0]	WriteID (AWID)	ReadID (ARID)	Description
4b000x	DII - Device	2bxx	0	0	Device property of transaction overrides policy setting
4b0001	DII - PCIe Device	2b01	0	0	Write ordering at DII keeps writes ordered, but allows posted writes to pass reads
4bxx1x	DII - Memory	2b11	0	0	Transaction to memory mapped device functions (GIC) - this setting guarantees completion of all outstanding writes to any target using the same AWID
4bxx1x	DII - Memory	2b10	0	0	Memory transaction (SRAM, ROM, Flash) - AxID ordered requests, Hazard check at PoS[21] or in AIU
4bxx1x	DII - Memory	2b10	0	1	Memory devices with write ordering requirements (some Flash), Hazard check in AIU
4bxx1x	single DCE/DMI	2b10	1	1	Issue of transactions free listed, Response follows Relaxed Order policy, Hazard check in AIU Best bandwidth
4bxx1x	interleaved DCE/DMI	2b10	1	1	Issue of transactions free listed, Response follows Relaxed Order policy, Hazard check in AIU Best bandwidth
4bxx1x	DCE/DMI	2b10	0	0	AXI ordering enforced - severe bandwidth penalty

Concerns about complete freewheeling

Sequential Consistency[22] cannot be guaranteed for interleaved memory transactions - a sequence of memory writes by a writer will alternately/sequentially target different memory agents (e.g. interleaving 4 DCEs at 64 bytes will result in transactions for sequential cache lines to be sent to different DCEs and/or DMI). This may be no issue for most applications.

0x0000 --> DCE_{00}, 0x0040 --> DCE_{01}, 0x0080 --> DCE_{02}, 0x00c0 --> DCE_{03}, 0x0100 --> DCE_{00} ... each DCE implements the Point-of-Coherence for part of the interleaved address space. If an observer agent's read requests do not arrive at the DCEs in the same order (due to different transport latency) the write-read order at each PoC may differ and writes may be observed in a different order than they were issued.

	DCE_{00}	DCE_{01}	DCE_{02}	DCE_{03}
t_0	W_0	W_1	W_2	W_3
t_1	R_0	R_1	W_6	R_3
t_2	R_4	R_5	R_2	R_7
t_3	W_4	W_5	R_6	W_7
t_4	R_8	R_9	W_9	W_{10}

Race condition between W_6 and R_2 has W_6 arrive earlier than R_2 - agent DCE_{02} sees a different request order and the data returned in R_3 will be different (returning the previous content of memory location 2) than expected under the assumption of a sequential write. This is a problem of having multiple PoC processing a sequential data flow originating at a single source.

Streaming write ordering

Writes arriving at the IOAIU from an external agent (e.g. PCIe complex) follow AXI protocol. Write transactions are tagged with the AWID, all transactions carrying the same ID are part of the same flow and for memory targets shall be considered sequentially ordered within that flow.

In most use cases, the agent observing a sequential stream of writes within a flow needs to observe these writes exactly in the same order they were created. This is called Ordered Write Observation and required by sequential memory consistency.

[21] Khaleel says, that hazard check may fail under rare circumstances - need to understand the conditions and fix if necessary

[22] This is a standard problem when traffic is split to interleaved memory channels - this is a trade-off with performance impact. Applications using Producer-Consumer models need to be aware of this behavior and shall apply appropriate caution

Ncore implements Ordered Write Observation by making writes visible to observers when the ordering point receives the StrRsp message; this is the idication, that the write has been completed at the POS. In CCMP StrRsp concludes the write operation.

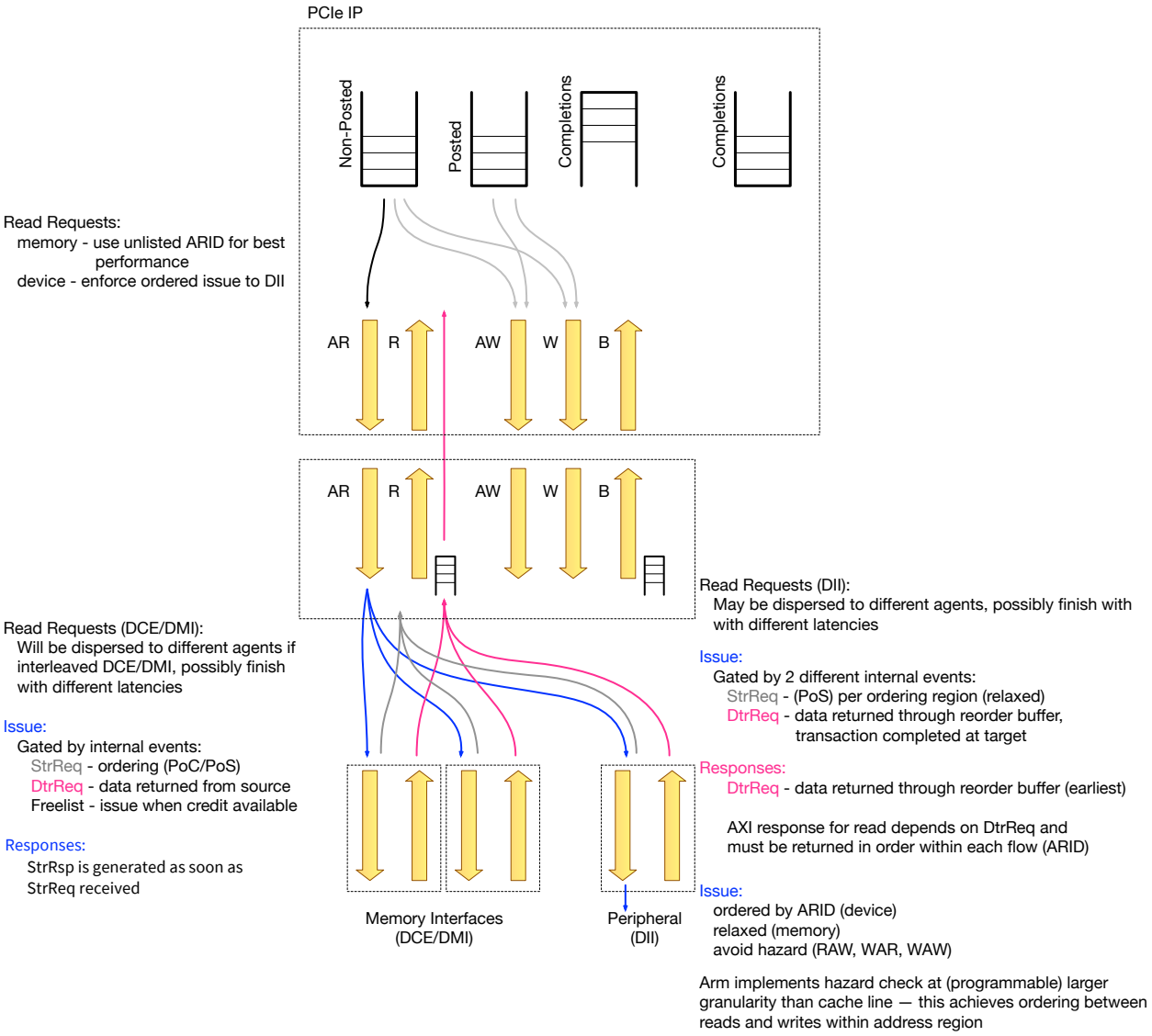
Transaction Progress through the system

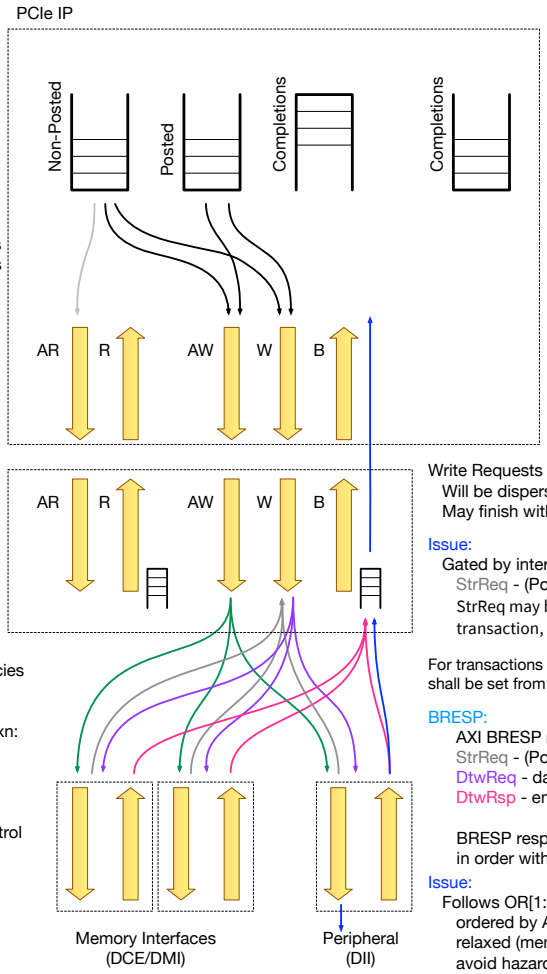
Acceptance of transactions

In AXI or ACE-Lite protocol read and write transactions may be arriving concurrently. The protocol does not infer any ordering between read and write channels in that case.

NCAIU

AXI or ACE-Lite transactions arriving at the NCAIU carry AxCACHE[3:0] attributes - NCAIU shall treat any transaction with AxCACHE[3:1] == 3b000 as device ordered and enforce request order - these transactions are usually sent towards physical devices and will require strong ordering.





Write Requests:
memory - use AWID = 0 for posted writes
use AWID != for all other writes
device - enforce ordered issue to DII

Write Requests (DCE/DMI):
Will be dispersed to different agents
May become ordered with different latencies

Issue:
Gated by ordering events of dependent txn:
StrReq - ordering (PoC/PoS)
DtwReq - data sent to target
Freelist - issue when credit available

OR[1:0] in transaction may be set from control register configuration but will be ignored at destination.

Write Requests (DII):
Will be dispersed to different agents
May finish with different latencies

Issue:
Gated by internal events, depends on CSR setting:
StrReq - (PoS) per ordering region (relaxed)
StrReq may be used to send the next pipelined transaction, ordering will occurr at the target DII

For transactions with AWCACHE Properties of OR[1:0] in transaction shall be set from control register configuration

BRESP:
AXI BRESP response for write may be sent early
StrReq - (PoS) per ordering region (relaxed)
DtwReq - data sent from reorder buffer
DtwRsp - endpoint has accepted data

BRESP response follows AXI rules and must be returned in order within each flow (AWID)

Issue:
Follows OR[1:0] ordering rule in transaction
ordered by AWID (device)
relaxed (memory)
avoid hazard (RAW, WAR, WAW)

Arm implements hazard check at (programmable) larger granularity than cache line — this achieves ordering between reads and writes within address region

Further ideas:

- Support early write responses if target is DCE or DMI and the write is a buffered write to memory. Sending an early response implies that the receiving agent takes ownership of the transaction, including the data and promises to follow the ordering rules of AXI/ACE-Lite
    - write transactions are accepted - AWADDR
    - WReady is returned - interface is ready to accept data
    - WData is received
    - BReady is returned as soon as the last data beat has arrived
    - Now the responsibility for transaction ordering is on the AIU
    - No other posted write transaction may pass a transaction that has been responded to
    - No read request with the same transaction ID may pass an earlier transaction that has been responded to
    - Posted write transactions may be issued as soon as no other posted write transaction with the same ID is ahead of it (all older have at least received StrReq - they have been ordered)
    - The entry in OTT, and the buffer space holding the data, can be released as soon as DtwReq has been issued - the next agent on the way to memory has now ownership of the transaction

- Use the "ordering" bits in our protocol to propagate the ordering rule for each transaction
- Treat *WriteUnique* and *WriteNoSnp* differently
- remove hazard checks at the top of the pipeline - if transactions make it to the destination, the hazard would be managed, yet the performance loss from stalling can be avoided as long as ordering between transactions on the way to PoC/PoS can be maintained
- add programmable granularity to hazard check in DII. This will support ordering between reads and writes within a local address region - avoid side effects
- Distinguish between posted/bufferable writes and non-bufferable writes - extend the ID by one bit 'B' to mark a bufferable write --> ID = {B, AXI-ID[n:0]} and use the following rule to allow passing:

Table 6: PCIe order enforcement

Type$_{New}$	**B$_{New}$**	**Type$_{OTT}$**	**B$_{OTT}$**	**ID$_{New}$ == ID$_{OTT}$**	**Comment**	**Action**
R	0	W	X	No	No match, different flow	pass
W	X	R	0	Don't care	Hazard Check	pass
R	X	R	0	Don't care	Hazard Check	pass
W	0	W	0	Yes	NPR - NPR, Hazard Check WAW	pass
W	0	W	1	Yes	NPR - Posted	no pass
W	1	W	0	Yes	Posted - NPR	pass
W	1	W	1	Yes	Posted - Posted	no pass

Implementing additional optimizations supported by PCIe, e.g. the relaxed ordering requirements introduced in PCIe 3 (IDO), would require visibility of PCIe Requester ID and PASID (stream ID, address space identification) as part of the arriving ACE-Lite transaction. It is assumed that this would be handled within the PCIe root complex.

Proposed Changes to uArchitecture of NCAIU

This chapter describes the changes required to the NCAIU architecture to support the new functionality described in the previous sections of this document.

GPRAR - support

The GPRARs for each NCAIU need to add 5 bits to control the transaction behavior for each address aperture. The configuration bits 4:0 are currently not used across all AIU architectures and their function in CAIU will not change.[23]

Table 7: GPRAR Format

CAIU GPRAR$_x$: General Purpose Region Attribute Register [Offset: 0x0400]				
Bit	**Name**	**Description**	**Access**	**Reset**
0	Rsvd	Reserved	RZWI	0x0
1	ReadID	No ordering by ARID (free listing)	RW	0x0
2	WriteID	No ordering by AWID (free listing)	RW	0x0
4:3	Policy	Response policy: 11: Endpoint/Request Order - do not set ReadID/WriteID! All memory transactions are issued and completed in order (issue from DtrReq/DtwRsp), BRESP from DtwRsp 10: Relaxed Order - next dependent transaction is issued as soon as ordering at the target has been achieved (StrReq), BRESP from DtwReq 01: Write ordering at DII - keeps writes ordered, but allows posted writes to pass reads 00: Reserved, shall be treated as Endpoint/Request Order	RW	0x0
8:5	Rsvd	Reserved	RZWI	0x0
13:9	HUI	Identifier of the home unit, depending on type **Peripheral:** Index of target DII in enumerated list **Memory:** Memory Interleave Group **HierGateway:** Gateway Interleave Group **Chip-to-Chip:** Gateway Interleave Group	RW	0x0
19:14	Rsvd	Reserved	RZWI	0x0
24:20	Size	This field represents the size of the region in a binary number with a range of 0 .. 31. The size of the region is defined by RegionSize = IGSize * $2^{(size+12)}$ Bytes	RW	0x0
28:25	Rsvd	Reserved	RZWI	0x0
30:29	HUT	This field indicates the Home Unit Type: 00: System Memory 01: Hierarchical Gateway Interface (Ncore 4.x) 10: Peripheral 11: Chip-to-Chip interface Note, the pattern has been chosen to retain compatibility with the current, single bit selector (bit 30) 0: System Memory 1: DII	RW	0x0
31	Valid	This bit indicates if the region is valid 0: Invalid mapping 1: Valid region mapping	RW	0x0

[23] CHI and ACE CAIU will not support these settings and propagate the protocol inherited attributes - register settings will be one either ignored or required to use a specified pattern (otherwise expect undefined behavior)

New Rules for the transaction processing

Transaction management

- Maintain separate linked lists for read and writes by AxID
  - transactions may be issued depending on ordering requirements using the AxID
  - AxID ordering when issueing requests towards the target agent may be disabled for an address region by GPRAR[1][24] for read accesses and GPRAR[2][25] for write accesses
    - Setting GPRAR[x] == 1 disables checking
    - The default value (after reset) will be 0 (checking enabled)
  - Responses for same AxID must be in order within a read or write channel - responses for individual flows (all transactions with the same AxID) from downstream agents may arrive out-of-order and must be reordered to meet AXI specification
- Same cache line address blocking until the StrRsp has been sent. This is done across reads and writes to avoid hazards (WAW, WAR, RAW)
  - This check shall be controlled by GPRAR[0] the "hazard" bit
  - Checking shall be disabled when hazard == 1
  - The default value (after reset) will be 0 (checking enabled)

Transaction Issue

- Each transaction will be started by sending a CmdReq. The command includes a 2-bit wide "ordering" field OR[1:0]. This field determines request processing at the target agent (DCE, DMI, DII). Valid settings for OR[1:0] are

Table 8:Ordering fields

OR[1:0]	Function	Description
2b11	Endpoint ordered	Strongly ordered with respect to the same endpoint device
2b10	Strongly ordered[26]	Strongly ordered with respect to a previous access from the same source to the same location. Writes to the same address issued by the same agent (FUnitID and AxID) must meet requirements of sequential consistency
2b01	Write order	Write ordering occurs at DII - this mode is used in conjunction with disabled ID checking. Transactions will be issued at the AIU without checking for the completion of older writes from the same AxID
2b00	Not ordered	No ordering required for this access

- Target agents with memory semantics (DCE/DMI) will ignore the OR[1:0] setting
- Read/Write requests shall be issued when they are ready-to-issue, i.e. they meet certain conditions. The conditions to issue a transaction depend on the type of transaction (Read/Write), GPRAR [4:0], the AxCACHE[3:1] value and an "*Event*" trigger that removes the transaction from the dependency chain (eg. StrRsp). In AXI4, all transactions with AxCACHE[3:1] == 3b000 are considered "device" transactions and must be endpoint ordered

Table 9: Device Transactions (AxCACHE[3:0] = 4b000x)

Type	Event	GPRAR[4:0]	OR[1:0]	Description
Read	DtrReq/StrReq	5bxxxx0	2b11	Endpoint ordered, transactions are issued in order[27]
Write	StrReq	5bxxxx0	2b11	

[24] Ignore ARID when set
[25] Ignore AWID when set
[26] Current documentation: ***Request Ordered***
[27] Initiator will not send another request until the current transaction has received RRESP/BRESP - therefore StrReq may be used to issue the next transaction to simplify the implementation

Table 10: Memory Read Transactions (AxCACHE[3:0] = 4bxx1x)

Event[28]	GPRAR[4:0]	OR[1:0]	Description
StrReq/DtrReq[29]	5b10x00	2b10	Issue respects ARID as dependency, hazard check enabled
StrReq/DtrReq[29]	5b11xx0	2b11	Endpoint ordered, transactions are issued in order
StrReq/DtrReq[29,30]	5b10x10	2b00	Issue: Ignores ARID as dependency, hazard detected, this transaction does not generate a dependency for subsequent transactions

Table 11: Memory Write Transactions (AxCACHE[3:0] = 4bxx1x)

Event[28]	GPRAR[4:0]	OR[1:0]	Description
StrReq	5b100x0	2b10	Issue respects AWID as dependency, hazard check enabled
StrReq	5b11xx0	2b11	Endpoint ordered, transactions are issued in order - a DII agent will return DtwRsp only after receiving BRSP
StrReq[30]	5b101x0	2b00	Issue: Ignore AWID as dependency, no hazard detected, this transaction does not generate a dependency for subsequent transactions
StrReq[30]	5b10101	2b01	Issue: Ignores AWID as dependency, hazard detected, this transaction does not generate a dependency for subsequent write transactions at the IOAIU but, if sent to a DII target, it will issue writes in sequential order

Ordered Write Implementation

Ordered write transactions guarantee that sequential write requests arriving on one agent would become visible to any (different) observing agent in the same order they are issued. On the CHI interface this property will be enabled by hazard checking for each write - the barrier for a write must be kept until all previous writes

Response Generation to Initiator Agent

- All responses within an AXI read or a write flow (same ARID or AWID), must be generated in the order the transactions arrive at the NCAIU. This property must be guaranteed, independent of their completion order at downstream agents.
- The earliest time a transaction may be responded to is a function of the policy programmed into GPRAR [4:3], the arrival of messages (StrReq, DtrReq, DtwRsp) from downstream agents and the type of transaction (read/write).
- Responses may be generated when the event message arrives and no older transaction with the same AxID is outstanding (issued and waiting to complete, or completed and waiting to generate a response)

Table 12: Device Transactions (AxCACHE[3:0] = 4b000x)

Type	Event	Response	Description
Read	DtrReq	RRESP	Endpoint ordered, responses may be generated when the event message arrives and no older transaction with the same AxID is outstanding (issued or completed and waiting to generate a response)
Write	DtwRsp	BRESP	

[28] Event refers to message that removes the transaction as a dependency
[29] Whatever event arrives first
[30] Ignoring AxID, transaction can progress when the transaction causing the hazard has been ordered

Table 13: Memory Transactions (AxCACHE[3:0] = 4bxx1x)

Type	Event	GPRAR[4:3]	Response	Description
Read	DtrRsp	3b10	RRESP	Transactions are issued and completed in order at the downstream agent, data will be returned and saved in a reorder buffer
Read	DtrRsp	3b11	RRESP	
Write	DtwReq	3b10	BRESP	Ordered response, may be sent as soon as the responsibility for the transaction, including data, has been transferred to the completer
Write	DtwRsp	3b11	BRESP	Endpoint ordering, response will only be sent after the transaction has completed at the endpont

AXI spec "Ordered write observation" says it "can support the Producer/Consumer ordering model with improved performance". AXI4 spec does not mention PCIE spec, but we know that the PCIE spec uses a Producer/Consumer ordering model. For instance, in a system like this:

> *CPU <-> PCIE Controller <-> PCIE AXI Bridge <-> AXI with Device and DDR slaves (Device IP is connected to AXI slave data port0 and APB register port, DDR memory module is connected to AXI slave data port)*

The CPU performs the follwing two operations,

- CPU writes data to DDR
- CPU writes Device APB register to start Device activity

Because PCIE memory writes (both prefetchable and non-prefetchable) are posted, i.e. without responses, PCIE AXI Bridge will perform the above two operations successively with the same ID but without waiting for BRESP. Before the data reaches DDR, Device may have seen the APB register write and starts reading the data, so the data may be old and invalid.

If "**Ordered write observation**" is supported, there will be no such a problem, because it requires the interface "**if two write transactions, with the same ID, are observed by all other agents in the system in the same order that the transactions are issued**", i.e. if APB register write is observed by Device, it will guarantee data write to DDR has been observable to Device.