

PCIe and AXI domain traffic ordering - A Novel Approach

Sathish Kumar Sivakumar, Venkatesh Veluvolu, Suraj Augustine, Shibin Bose Kuvera, Gaurav Agarwal
 META, Building 1.04 - Embassy Golf Links Business Park, Bangalore, India

Abstract-This paper presents four design approaches to maintain PCIe ordering rules in the AXI domain, specifically ensuring Strongly Ordered (SO) writes complete after all preceding Relaxed Order (RO) writes. While initial approaches face performance and dependency issues, later designs leverage unique RO IDs. Approach IV, an enhanced counter-based method, offers the most effective and efficient solution for achieving full PCIe Gen6 throughput with minimal overhead.

I. INTRODUCTION

PCIe has many strict ordering rules defined in the PCIe specification as far as posted and Non-posted transactions are concerned in order to honor the producer consumer ordering model and also to avoid deadlock avoidance in a system. Basically, the consumer should not read the data from the target unless the producer has finished writing to the target location. These ordering rules are explained as in the below Table I. [1]

These ordering rules are adhered in the PCIe domain and the moment these transactions enter the AXI domain, due to the nature of AXI having all five separate channels, it is impossible to guarantee PCIe ordering in the AXI domain in a straightforward manner. As an addition to the above ordering rule table, one of the system requirements that we had to address in our product is to schedule a SO (Strongly Ordered write) only after all the previous RO (Relaxed order writes) have been completed (BRESP for all the previous RO writes have returned back from their actual targets). The SO indication need not be necessarily the PCIe TLP RO bit and this can be decided based on a variety of other attributes like packet address, steering tag etc and hence the ordering needs to be implemented in the AXI domain. Fig. 1 snapshot gives the requirement for the same.

Table I					
PCIe Ordering Rule Table from Specification					
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

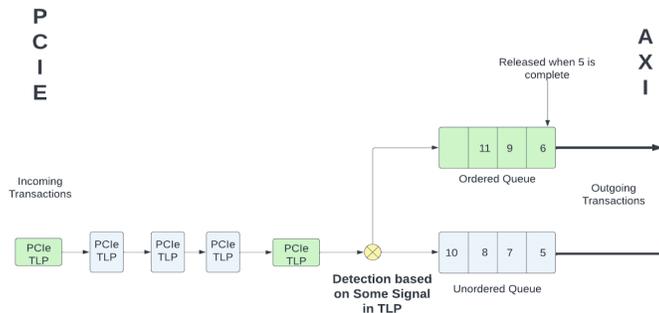


Fig. 1. Requirement

As shown in Fig. 1, the TLPs in the ordered queue are supposed to be scheduled only after the previous TLPs in the unordered queue have been scheduled and completed. That is TLP 6 can be scheduled on the AXI domain only when the responses for all the previous 5 TLPs have come back in the AXI domain and PCIe TLP 9 can be scheduled only after the responses for all the previous TLPs in unordered queue and the TLPs in the ordered queue. That is TLP 9 can be scheduled on AXI only after the responses for all the previous TLPs till 8 have come back. In the event of lack of the ensuing ordering schemes in hardware, the only way to enforce this ordering is to issue a read from PCIe domain which will ensure that all the previous writes are completed prior to this and then issue a SO write. Such a read transaction will be highly detrimental to the overall scheme of things due to the involved latencies of these read transactions.

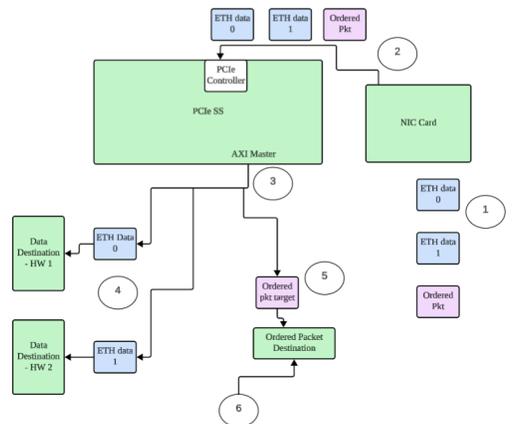


Fig. 2. Packet flow events

Above Fig. 2 explains the sequence of events,

1. Data Packets followed by Strongly Ordered Packet
2. Enter the SoC via PCIe adhering to the PCIe ordering Rules
3. PCIe Transmits the writes on the AXI master
4. Data packet targets
5. Ordered packet targets
6. CPU or the consumer hardware reading the ordered packet contents via intr.

DESIGN APPROACHES:

This paper discusses 4 design approaches to achieve the above ordering requirement between PCIe and AXI domains and the pros and cons of these approaches. All of the approaches involve using the Arteris NoC and the Gen6 Synopsys pcie controller.

Design Approach - I:

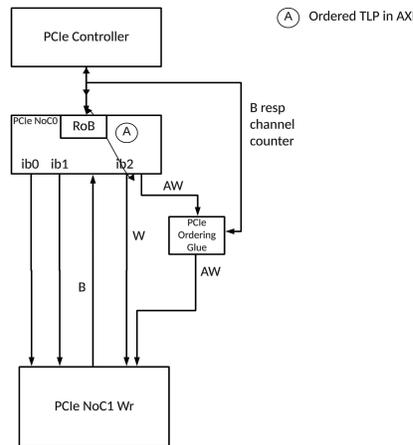


Fig. 3. Design Approach - I - Baseline Diagram

In this approach, we make use of the layered Arteris NoC architecture in the Subsystem and the functionality of RoB (Read reorder buffer) in the path in order to achieve the needed ordering. Synopsys PCIe controller outputs all AXI writes with the same ID 0 and these transactions hit the RoB in the NoC before proceeding further into the SoC. This RoB is present for other system reasons like transaction splitting etc and we could leverage the same for this ordering requirement as well. The RoB ensures that its output towards the PCIe controller is always ordered as the incoming transactions are all with the same ID and hence we can make use of the simple counter to track the number of RO (Relaxed Ordering) transactions that the HW needs to wait before scheduling the needed SO (Strongly Ordered) transaction. So, the design keeps track of the number of ROs that it has to wait before scheduling the SO and at the time of SO arrival, loads this value into the FSM along with a separate path in the Arteris NoC that is dedicated for the SO traffic. The SO transactions sit in a fifo in the logic and the logic will also have a BRESP counter to track the number of in-order of bresps from the output of RoB and once this value equals the stored value in the FSM, the head of entry in the SO FIFO can be scheduled. This scheme is simple to implement but has some performance implications depending on the SO frequency and the application RTT (Round trip time) of SO transactions.

The achievable BW in this scheme is dependent on two factors as below,

1. SO write transaction frequency
2. SO RTT.

The max achievable BW can be characterized using the below equation,

$$\text{Formulae} = ((\text{RO_PER_SO}) * 512) / (\text{RTT} + \text{RO_PER_SO})$$

This is happening because of the fact that the responses to all the ROs before a SO are getting accumulated in the RoB and even though we will be scheduling the subsequent RO transactions but since the responses are locked up in the RoB waiting for the SO response to come, there will be a fall in BW. This is again attributable to the fact that the PCIe controller sends all write transactions with the same ID 0.

With this limitation, the max possible BW that can be achieved is given in the below table assuming MPS and MRRS as 512B and total outstanding of 512,

Table II		
Bandwidth achieved for given RTT and RO per SO		
SO RTT (in ns)	RO per SO	Theoretical BW (GBps)
1500	256	74.64
1000	256	104.36
500	256	114
1500	128	40.26
1000	128	58.1
500	128	104.36
400	128	114

Design Approach - II & III & IV:

Both approaches II, III and IV need unique IDs to be generated for all ROs from the PCIe controller. In order to enable unique IDs for the posted AXI transactions, below parameters need to be enabled in the PCIe controller.

- CC_IB_P_MNG_ID_EN
- CC_IB_P_MNG_ID_RO_REMAP_EN

Once we have these parameters enabled, then the PCIe controller will generate an unique ID for all the RO Posted transactions as per the below Fig. 4.

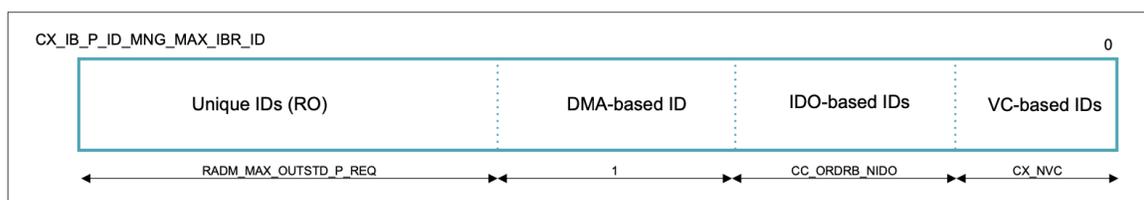


Fig. 4. Unique ID scheme for PCIe Controller

We have a single VC design and all HDMA transactions get generated with an ID of “1” always. And for META configurations, CC_ORDRB_NIDO parameter is 0 and RADM_MAX_OUTSTD_P_REQ parameter has a value of 512 and hence the unique ids that get generated for posted RO transactions will be from 2 to 513. SO packets will get an ID of 0.

Design Approach - II:

Once the unique IDs are available for the posted writes, we can make use of the below scheme to maintain ordering and also sustain bandwidth.

Lets assume we support up to 64 SO at any point of time out of the total outstanding supported of 512. So, we maintain a counter on a per SO basis and hence there will be 64 SO counters each having 9 bits wide to support total OT of 512. These counters count the number of RO transactions till the SO transaction arrives. Also, as the RO

transaction gets driven downstream, we extract the AWID and send it as part of the user bits and also have them looped back from the Arteris NoC on buser bits. We allot a unique ID for all the ROs before the SO (This unique ID will be $(ACT_SO_ID = SO_ID_CNT + 2)$ - this SO_ID_CNT will range from 0-63 when supporting SO OT of 64) and when the SO arrives, the current count of the Number of ROs to be waited before the SO could be scheduled will get written to the SO fifo in the ordering glue logic along with the AW transaction attributes.

Since the ROs before the SO are sent with the same ID, the RoB in the PCIe NoC0 will ensure that the responses to those ROs are in order. At the output of the NoC0 towards the PCIe controller, there will be a glue logic that will be calculating the Number of BRESPs received on the ACT_SO_ID basis. In the ordering glue, the FSM will check if the count read from the FIFO head is equal to the corresponding $BRESP_COUNTER[ACT_SO_ID]$. If it equals, then it means all the RO prior to the SO in the head of FIFO have completed and all of them have received their BRESPs and then this SO can be scheduled. Once the SO has been scheduled, one can clear the $BRESP_COUNTER[ACT_SO_ID]$ back to 0.

The one potential drawback with this scheme is that since the ROs before a SO are all scheduled with the same ID, then we introduce a dependency between these ROs in the RoB in the NoC as their responses have to be vacated in order back to the PCIe controller. This added dependency could hold back write responses to PCIe controller causing holdups in packets due to PCIe ordering requirements or limit the issue of new downstream packets from the controller as it is limited by outstanding limit.

Design Approach - III:

With approach III our goal was to

- Avoid any unnecessary dependencies in packets as explained above.
- Move from a very targeted use case to scale to a generic solution with no limit to SO frequency, worst case all the outstanding could be SO.
- A design which could scale based on the outstanding requirements per chip.

As we remove the limit on the SO packet frequency, we design for the worst case where all inbound packets could be SO, hence we consider SO count of 512 for as design with 512 outstanding. On a per SO basis, the design will maintain a 512 bit vector corresponding to the RO Unique IDs that the controller can generate. As and when a RO transaction comes from the PCIe controller with a unique AW_ID, the design ordering glue will take the RO_ID as index and go to the particular SO ID based vector (at the start this SO-ID will be 9'd0 out of the possible 512 SO transactions) and make the corresponding entry as "1". The SO_ID [range 0-511] is also appended as MSB to the unique RO IDs and then sent downstream towards the Arteris NoC.

Now, when the SO transaction is issued from the controller, the AW transaction attributes of the SO write get written to the FIFO in the ordering glue along with the current SO_ID [Range 0-511].

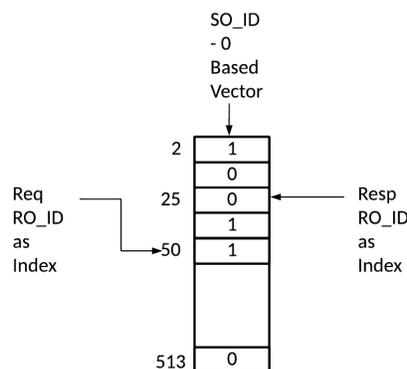


Fig. 5. Per SO_ID vector representation

Now when the RO Write responses come back from the actual target, the ordering logic will compare each of the responses based on the BID value (SO_ID as MSB and RO_ID as LSB) and go to the 512 bit vector corresponding to the SO_ID in the response and using the RO_ID as index, it will make that entry as 0 indicating that the Response has come back. Meanwhile the FSM in the ordering glue will have the head of the SO fifo entry available and will keep checking the SO_ID based vector if all the entries in its 512 bit vector has indeed become “0”. Once it becomes 0, it means that all the prior RO writes responses have come back and we are good to schedule this SO.

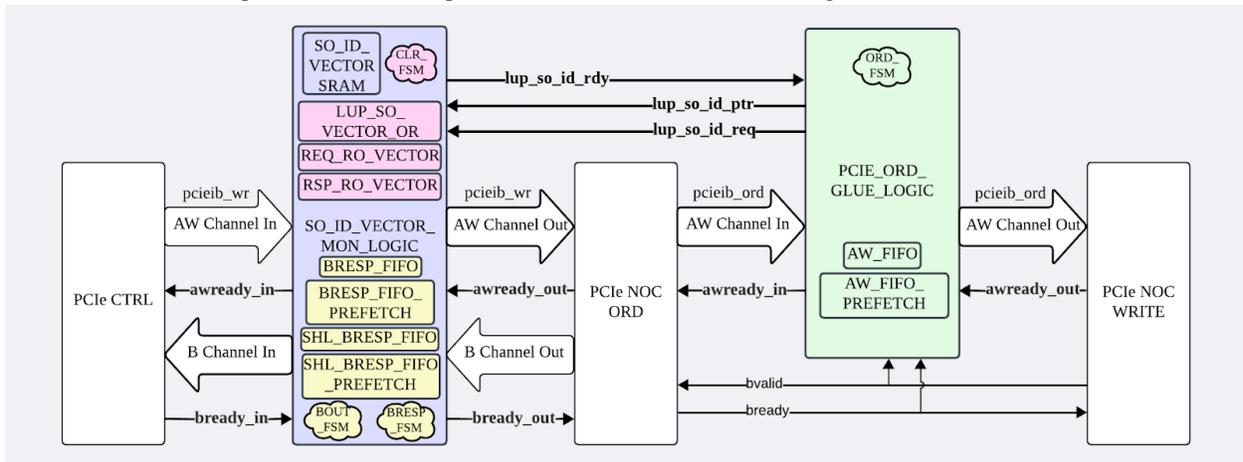


Fig. 6. Scheme III Block Diagram

Lets see this scheme with an example, For the traffic in Fig. 6 above, when the first SO write comes, let's assume that ROs with unique ID starting from ID 2 to 50 have come and hence the `SO_ID[0][511:0]` vector will have all 1's from bit positions [50:2]. Now all the RO responses can come out of order and as and when they come back, we take the `SO_ID` from the MSB of the incoming BID and go to the corresponding vector and with `RO_ID` as index, update that entry as 0.

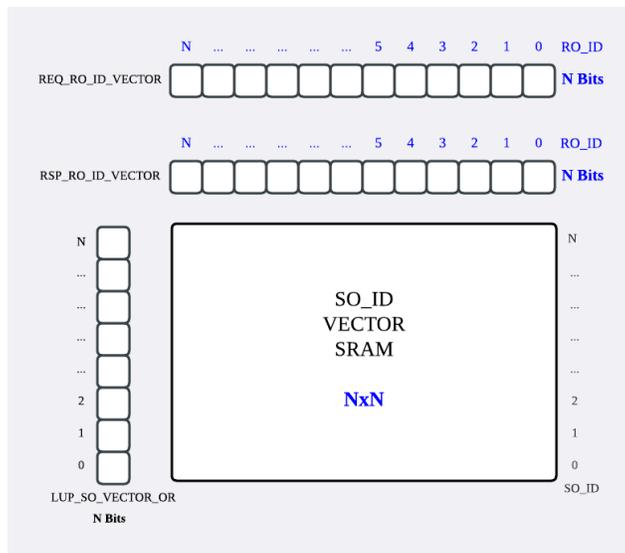


Fig. 7. Vector resource in approach III



The SO_ID vector monitoring glue and the PCIe ordering glue will together monitor the head of fifo entry in the ordering glue to see if the corresponding vector has all its 512 bits as 0 and once 0, the PCIe ordering glue will go ahead and schedule the SO write downstream.

To remove any requirement for SO frequency among RO transactions, the number of SO_ID vectors is also kept at MAX_OUTSTANDING. The microarchitecture and design of the logic needed to account for the following. The number of the vector array needed for an outstanding of 512 is 512X512. To provide bit level control for the update of these bits would make it difficult to converge on timing. Also, setting of the vector array is done on downstream AXI transactions and clearing is done on AXI responses for these transactions. To make the design area optimal, we use a 512x512 memory with 2 prefetched arrays maintained as flop array. One flop array is for updating the vector when request comes downstream from the controller. The other vector is maintained to clear bit when the responses are received back. We need to maintain 2 different vectors as on the request side we will be working on one SO_ID vector while the responses could come for any previous vectors. Whenever a SO packet comes on the request side or response comes for a SO_ID vector not currently loaded, the current vector is written back to the memory and the new vector initialized or read out of the memory. Additional checks ensure that if the response is for a vector already loaded in the request vector, the vector in the request vector is loaded directly to the response vector. No delay is introduced in the request channel. On the response channel, if a response warrants a load from the memory it could introduce a one cycle delay for memory store and load operation. We add FIFO on the response channel to prevent backpressure on this path.

When a SO write request is held in the AW FIFO, the AW packet is held until all RO transactions before it completes as indicated by B Channel response. An ORd 512 bit vector is maintained with the reduction OR and is readily available for the AW output FSM to check before issuing the AW packet. This avoids the need to compute the status after fetching from the memory.

With this approach, we wont see the performance drop seen for higher SO frequency with higher RTT and even the dependency among ROs that we saw in Design Approach - II has been avoided in this approach.

Design Approach - IV:

The approach IV is an enhanced version of approach III. While increasing the total outstanding support, the width of the vector array needs to be increased along with depth to support all the outstanding transactions as SO. This will challenge the timing and also leads to an increase in memory area. To address these challenges, we have used a counter based approach here.

Instead of “N” bit request vector and response vector in approach III, a “Log2(N)” bit request counter and response counter has been used in approach IV. Here the N is 512 if the total outstanding transactions are 512 and the counter width becomes 9. The N bit vector array in the memory also replaced with Log2(N) bit counter for each SO. Since the number of bits in the request and response vector become reduced from N to Log2(N) in the counter based approach. The same way the memory requirement also reduced from NxN to Log2(N)xN.

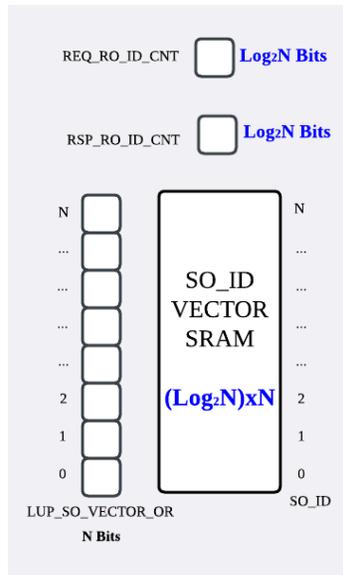


Fig. 8. Counter resource in approach IV

In approach III the request transaction will SET the bit in the vector array based on the AWID and the response will CLEAR the bit in the vector array based on the BID, while in approach IV the request transaction will increment the counter by 1 and the response will decrement the counter by 1. The counter load operations from Memory and/or from request counters are exactly same as the approach III.

CONCLUSION:

On the whole, design approach 4 is the best possible approach in order to solve the problem at hand which will provide us the full throughput of PCIe Gen6 without much overhead.

REFERENCES

- [1] PCI-SIG, "PCI Express base specification revision 5.0, version 1.0," Beaverton, OR, USA, May 28, 2019.
- [2] ARM, AMBA AXI protocol specifications, Available at, <http://www.arm.com>, 2003.
- [3] Arteris Flex NoC documentation. <https://www.arteris.com/wp-content/uploads/2024/10/arteris-flexnoc-reorder-buffer-ds.pdf>