NetSpeed Orion

Technical Reference Manual

Version: ORION-18.04

Revision: 0.0

NetSpeed Orion Technical Reference Manual

About This Document

This document is written for system designers, system integrators, and programmers who are designing or programming a System-on-Chip (SoC) using NetSpeed System's Orion NoC IP.

Audience

This document is intended for users of NocStudio:

- NoC Architects
- NoC Designers
- SoC Architects

Prerequisite

Before proceeding, you should generally understand:

- Basics of Network on Chip technology
- AMBA interconnect standards

Related Documents

The following documents can be used as a reference to this document.

- NetSpeed NocStudio Orion User Manual
- NetSpeed Orion IP Integration Spec

Customer Support

For technical support about this product, please contact support@netspeedsystems.com

For general information about NetSpeed products refer to: www.netspeedsystems.com

Revision History

Revision	Date	Updates
0.0	Jun 16, 2018	Initial Release

Contents

Figures

Tables

1 Introduction

1.1 NetSpeed Orion Overview

NetSpeed Orion is a physically aware, high-performance Network-on-chip (NoC) IP that is used for rapidly designing and analyzing highly efficient and scalable interconnects for a wide variety of SoCs. To quickly produce efficient, high-performance NoC IPs, Orion uses a requirements-driven design approach. Orion uses number of state-of-the-art algorithms to provide robust end-to-end QoS and application-level deadlock avoidance. The solution can scale from low- to medium-end SoCs with 10s of IP blocks to high-end SoCs with 100s of IP blocks and provides bandwidth scaling, optimal latency and clock frequencies of up to 3 GHz. Orion is built upon following the following fundamental design principles.

1.1.1 Requirements driven approach

Orion is configured and optimized using NocStudio - a NoC architecture exploration platform and interconnect synthesizer. NocStudio enables architects to design, configure and simulate NetSpeed's NoC IP as well as evaluate multiple SoC architectures. The interconnect can be designed and customized based on system requirements such as bandwidth, latency, traffic profiles, as well as fine-grained requirements such as total and per-flow system bandwidth and chip layout.

1.1.2 Physically aware latency optimized design

Orion design is physically aware of the layout of the on-chip system components producing an interconnect topology that is customized for the SoC layout. Being physically aware ensures that wiring congestions does not occur late in the design cycle and appropriate number of buffers and pipeline stages are present at various fabric channels to enable smooth backend design. Latency sensitive traffic can use dedicated connections to reduce arbitration and congestions, and 16 levels of QoS are supported for fine-grained bandwidth allocation and prioritization. Based on the system traffic specification and SoC physical layout, NoC topology, fabric components, and their placements are automatically computed using machine-learning and graph theory algorithms to optimize the design for area and power.

1.2 Configurability Options

NetSpeed Orion provides user configurability and flexibility across multiple design dimensions. In addition to providing flexibility of number of ports, interface widths, layout portioning, power and voltage partitioning, etc., significant configurability is also provided to the architect in defining the NoC topology as well as other system level characteristics. Orion supports the

following industry standard protocols - AXI4, AXI3, AXI-lite, ACE-lite, AHB and APB – along with many proprietary protocols.

2 Functional Description: System Interconnect

This section describes key system interconnect, NoC architecture concepts and features. Below figure illustrates the basic building blocks of the NetSpeed System Interconnect architecture.

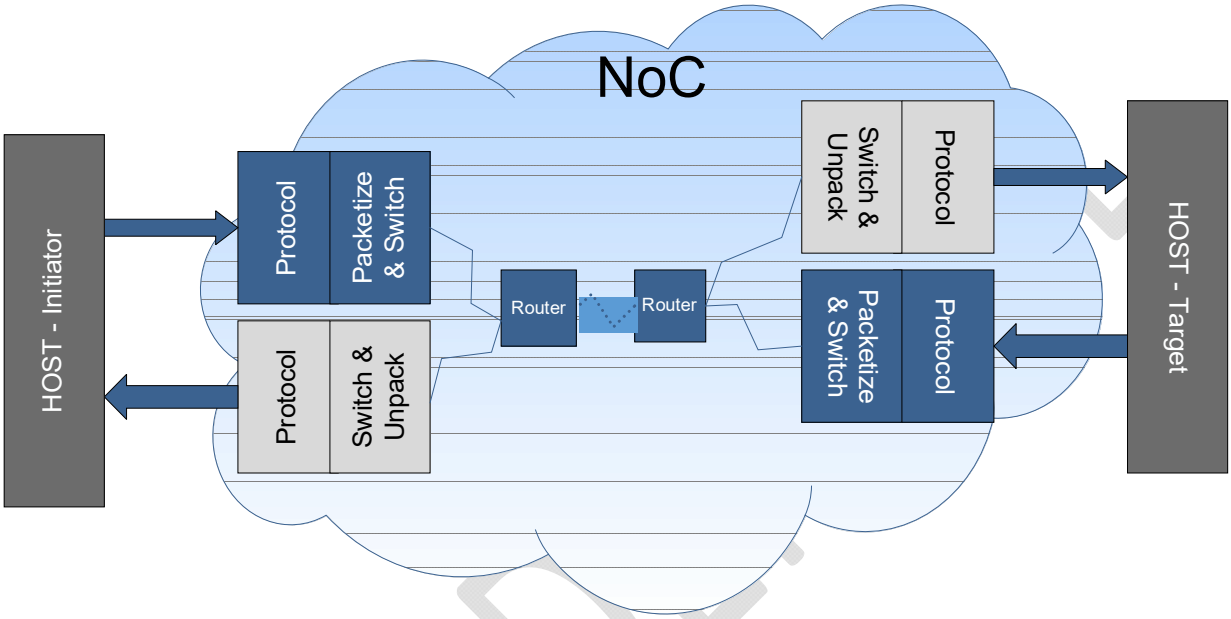


Figure 1. Bridge and Router Functions in the NoC

A bridge can connect a master of slave block to the NoC and perform the required operations to support the master and slave communication as per the protocol standards. A router can have four directional links, referred to as *north* (N), *south* (S), *east* (E), and *west* (W). It also can have up to four additional links to connect to up to four *host* (H, I, J, K). All eight links are identical and can be attached to bridges or to other routers.

2.1 NoC Topology

The NetSpeed NoC is a heterogeneous mesh-based interconnect that uses router elements organized in a heterogeneous 2D-mesh topology interconnected using point-to-point links. Below figure illustrates a full 2D-mesh interconnect, with a router at each mesh cross point.

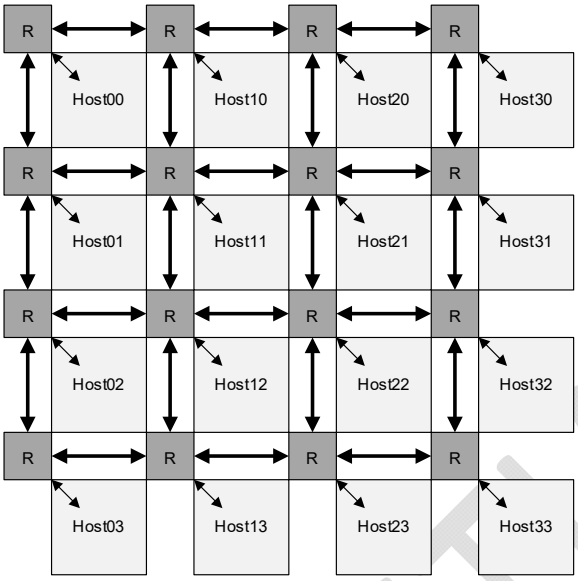


Figure 2. A 4x4 Homogeneous Grid or Mesh Interconnect

The grid has a specific number of routers on the X and Y axes. The number is determined by the size of the network, with 4x4 being the number used in the figure. A router is identified on the grid using its XY coordinate. Each router has four directional ports (N, S, E, and W). The router can transmit and receive messages on each port over the interconnect wires, forming a point-to-point link between the router and the one adjacent to the port. Each router has four additional ports (H, I, J, K) that act as standard host-port connections through which the router connects to the host port of a host block. Host blocks receive and/or transmit messages from/to the network through the host ports. Host blocks and host ports connected to router injection ports are also shown in the figure. The directional ports, if not connected to an adjacent router, can be connected to a host port.

Heterogeneous mesh interconnects can also be designed using NocStudio. An example of such a design is shown in the figure. In this example, the hosts are heterogeneous in size and shape and are interconnected using a customized mesh topology. The customized mesh can be viewed as a sparsely-populated full mesh created by selectively removing one or more routers and/or one or more links from a full mesh. Using the host block locations (XY coordinates), sizes, and shapes, NocStudio automatically instantiates the routers and links to provide the needed connectivity.

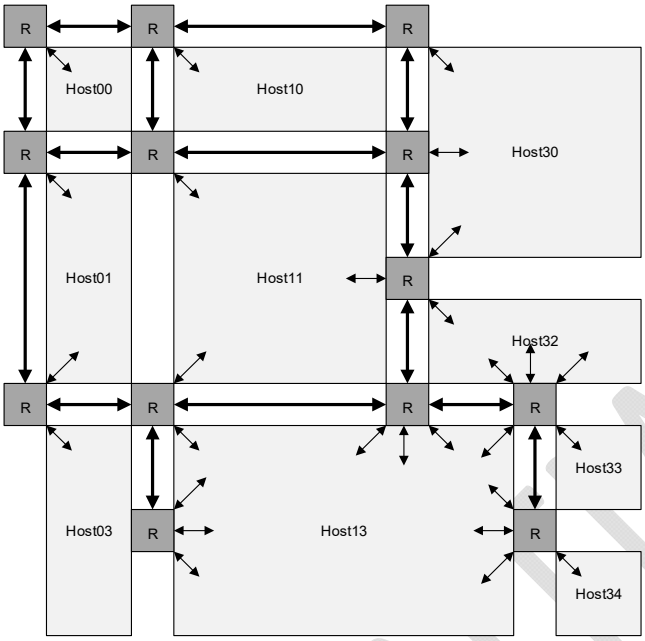


Figure 3 A 4x4 heterogeneous grid or mesh interconnect

In a heterogeneous mesh, a host block can connect to one or more ports of one or more adjacent routers to transmit and receive messages. A host can only connect to multiple ports if it spans multiple grid cells. In that case, it can connect to the available router ports in the cells it spans. Referring to the above figure, Host13 is 3x2 (X by Y) in size, spanning six grid cells. The figure shows it connected to 11 ports on 5 routers, as follows:

- The H & K and one directional port (east) on the left router.
- The H on the top-left router
- The H & I port and one directional port(south) on the top middle router
- The I port of the top right router
- The I & J port and one directional port(west) on the right router

Notice that the six grid cells have two cross points inside the 3x2 host. Because links cannot go over a host, routers that might exist at the cross points cannot connect to other routers. Therefore, no routers are instantiated at the cross points. Because the cross-point routers are missing, the directional ports of adjacent routers are available for host-port connection. Each host port or bridge within the NoC has a unique identifier (ID) called the *bridge ID* that is used for routing messages.

2.1.1 Multiple Physical and Virtual Networks

The NoC can contain up to eight (in 7-series version of the IP) physical mesh networks, or layers, for increased bandwidth and traffic isolation. Each physical layer operates in parallel, independent of the others. Up to four virtual networks can exist on each physical network, wherein different messages can be transmitted over different virtual networks. To implement virtual networks, every physical link in the physical mesh network has up to four virtual channels (VC). Virtual channels provide logical links over the physical channels connecting two routers ports. Each VC has an independently-allocated and flow-controlled flit buffer in the router nodes. In any given clock cycle, only one VC can transmit data on its physical channel.

The virtual network carrying a message is determined by NocStudio during the traffic mapping time (invoked with the **map/map_opt** commands) based on the priority and QoS requirements of traffic flows, and on the deadlock-avoidance requirements. Thereafter, all bridges are programmed so that messages are injected on the correct virtual and physical networks based on their QoS and destination information.

2.1.2 Routing Choices in a Heterogeneous Topology

In a mesh NoC, shortest-path dimension-ordered routing is commonly used. In this routing, packets are routed along routers in one mesh dimension until they reach a router whose first-dimension coordinate matches the coordinate of the destination router. Then, the packet is turned in the second dimension toward the destination and continues until it reaches that destination. For example, in a 2D mesh, packets can first travel along the X dimension, and then along the Y dimension, creating an XY route. Routing can also be done via an YX route, i.e., travelling the Y dimension first. It is also possible for packets to take a staircase route with more than one turn. A staircase route has the same number of hops as an XY or YX route but requires more elaborate route information to be carried with the packets.

NoC allows routes to have up to sixteen turns; it is expected that, in most topologies, four turns will be sufficient to provide the needed connectivity and hence is the default in NocStudio. User can change this default to up to sixteen turns based on which, route information is encoded. NocStudio determines routes between sources and destinations and stores them in the transmitter bridges. By default, NocStudio gives preference to XY route, then to YX route; if neither of these routes is available NocStudio will use the shortest route available while honoring the set number of turns. Default routes may be modified with “set_route” command.

2.2 ROUTERS

Below figure shows the schematic symbol of a router component in the NocStudio RTL library. The NetSpeed NoC uses 8-port routers. Four directional ports connect to the four adjacent neighboring routers and 4 additional ports connects to host ports using protocol specific bridges.

Each router port may be bidirectional and uses flits to exchange message packets over the NoC. To generate RTL, NocStudio instantiates routers and link components based on the topological structure of the architected NoC. Routers employ several micro-architectural optimizations to minimize the effects of HOL blocking and provide high switching throughput. Internal paths, buffering, routing logic, QoS logic, arbitration and channel allocation logic, etc. are tuned and optimized for high-frequency operation and low latency. Support for various NocStudio features is closely integrated into the router. The Router component is also highly configurable to allow NocStudio to optimize each router in the network for best area, power and performance. Each router utilizes one clock cycle for internal processing logic. External link traversal can be optionally allocated an additional cycle or combined with the internal processing cycle based on operating frequency and latency requirements. Optional pipelining can also be deployed on longer links. A user interacts with NocStudio to individually optimize various sections of the generated NoC for physical design requirements.

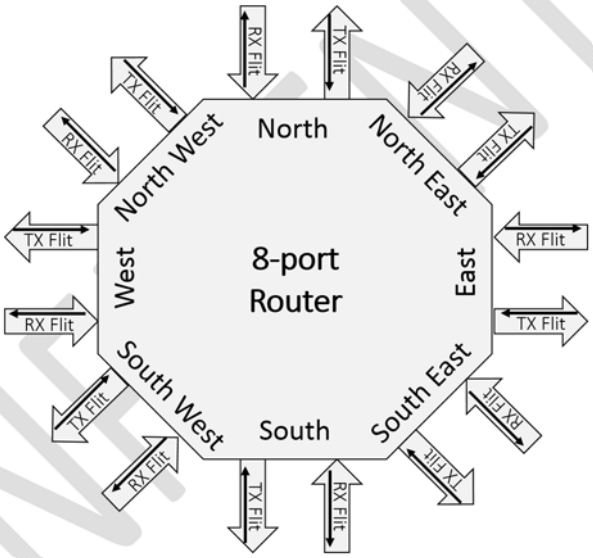


Figure 4. Schematic Symbol of a NocStudio RTL-Library Router Component

2.3 INFORMATION TRANSPORT IN THE NOC

A bridge maps transaction messages issued from host port interfaces to NoC layers and VCs. The bridge also does the route computation, enforces QoS (fixed priority and weights), and packetizes data for transport over the NoC. Packetized data is sent by routers to the receiving bridge, which unpacks the data and translates it back to the host protocol. A request and response paths are shown in the figure.

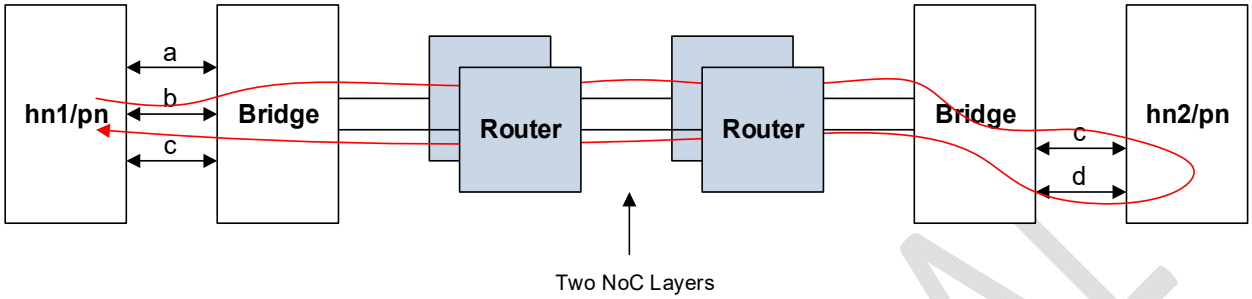


Figure 5. Information Transport in the NoC

2.3.1 Packetization

To transmit data between host ports and router injection/ejection ports, host transmit-data signals must be packetized and converted into a NoC-specific packet format. The bridge component converts the host output signals into NoC packets for transmission, and it converts packets received from the NoC routers back to the host input signals for delivery. With bridges between the host port and the router injection/ejection ports, a variety of hosts can be supported and connected using the same NoC.

All communication between router ports in the NoC is done using messages in the form of standard packets. NocStudio determines the route between source-host and destination-host port pairs based on several factors, such as the load on various links along the paths, message QoS, topology, etc. Every packet injected into the NoC by a bridge carries information about its destination host port and the route it should take. As a packet travels within the NoC, the routers use this information to send the packet to the next router until it reaches the destination router. The destination router delivers the packet to the bridge connected to the destination-host port.

Packets are composed of one or more flits. A single flit is transmitted or received at once (in one cycle) between various routers ports. The first packet flit is marked as start-of-packet (SOP) and the last flit is marked as end-of-packet (EOP).

2.3.2 Arbitration

When an SOP is received at a router port, it participates in arbitration to allocate the output VC (this corresponds to the buffer allocated for the VC in the next downstream router). When the SOP wins arbitration, the output VC is allocated to the packet, and all subsequent packet flits use the same VC. The VC cannot be used by another packet until the EOP flit arrives, freeing the VC for use by another packet. When an output VC is allocated to a packet at an input VC, it must arbitrate for the output port. This is because multiple VCs can exist on the output port allocated

to multiple packets at the input. This second arbitration occurs for every router output port. All packets at router input ports that have a VC allocated on the output port participate. The winning packet sends its flit in the same cycle. Thus, multiple VCs at a port can be interleaved. However, multiple packets are never interleaved within a single VC.

2.3.3 NoC Channel Width and Heterogeneity

Flits are carried in NoC router channels, which are restricted to certain widths. Each flit carries overhead, and the number of flit payload bits varies with the flit size. Unless a flit is an EOP flit, the number of payload bits is restricted to a power-of-two multiple of **cell_size**. That value is fixed in NocStudio for the entire NoC project (the NoC project **cell_size** can be modified with the **mesh_prop** command). EOP flits can contain any integer multiple of the **cell_size** payload bits. The value of **cell_size** must be selected based on the host-port signal properties to maximize the packing efficiency of the bridges. The figure illustrates a packet that is organized as two large flits or four small flits.

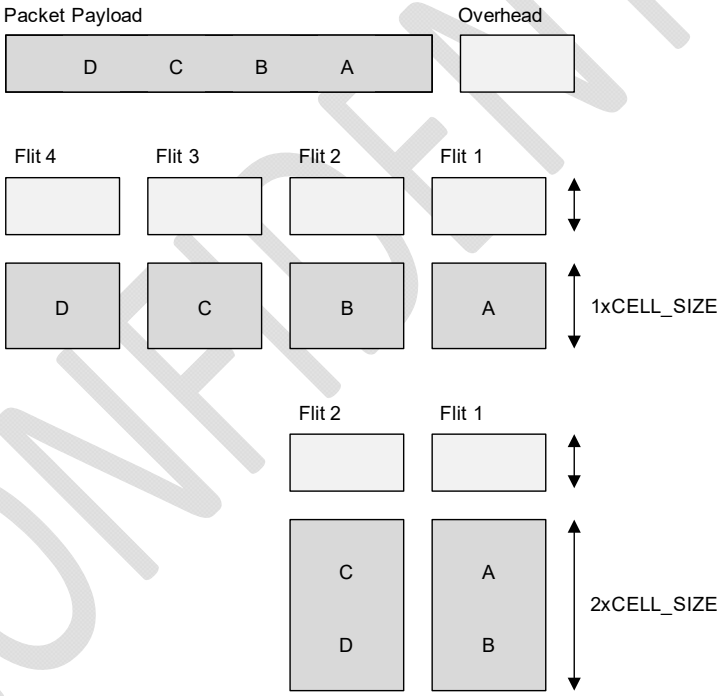


Figure 6. NoC Packet Organized in Two Different Flit Sizes

As packets traverse NoC channels of different widths, multiple flits can be combined into a flit with a larger payload, or a large flit can be sub-divided into multiple flits with smaller payloads. When flits are combined or divided, overhead bits are unchanged except for bits that identify a flit as EOP or SOP. For example, when flits in a two-flit packet (first flit is SOP and second is EOP) are combined, the resulting flit is marked as both SOP and EOP. Because a non-EOP flit is restricted to a power-of-two multiple of **cell_size**, only a power-of-two multiple of flits can be

combined to form a large flit (unless an EOP is present). Similarly, a large non-EOP flit can only be sub-divided into a power-of-two number of small flits.

The NoC channel and buffer widths are sized to the message overhead bits and a power-of-two multiple of the **cell_size**. Flits adapt to the channel widths as they travel from one channel to another—either the flit payload is sub-divided into multiple flits, or multiple flit payloads are merged into a larger flit. When a bridge injects flits at the router injection port, the bridge must adjust the transmitted flit payload size to match the injection-port channel width. When flits are ejected from a router port to the bridge, the flit size is adjusted to match the router ejection-channel width. To achieve high clock rates, the NoC restricts the global flit-size conversion ratio between 1:16. Locally, within a bridge the conversion ratio is 1:16, and within a router it is 1:4. During traffic mapping and channel sizing, NocStudio ensures that channels are sized to meet this restriction.

Even if the NoC channels are sized at power-of-two multiples of **cell_size**, non-power-of-two flits can be combined if an EOP flits is in the set of flits being combined. The resulting EOP flit can contain a non-power-of-two multiple of flits. When an EOP flit moves from a wider channel to narrower channel it gets divided into the correct number of smaller flits. Figure illustrates merging and dividing flits under various channel-width scenarios.

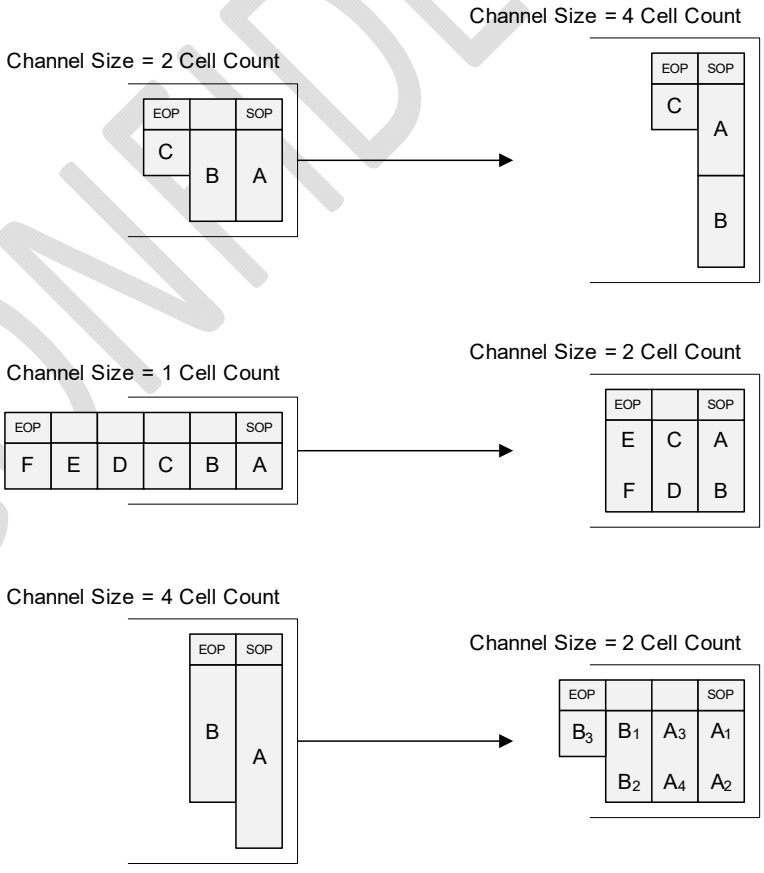


Figure 7. Merging and Dividing Flits with Various Channel Widths

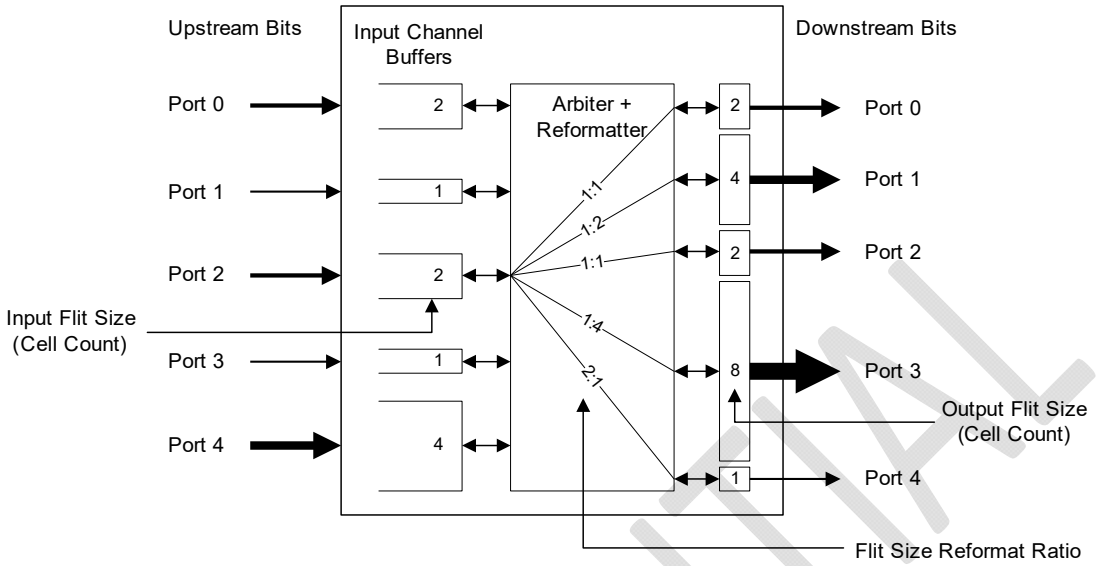


Figure 8. Combinations of Flit Merging and Division at Router Ports

2.3.4 Determining NoC Channel Widths

NocStudio sizes NoC channels based on host-port data widths, the number of packet flits that traverse a channel, and the channel bandwidth requirement from all traffic flows sharing the channel.

The widths of router port channels connected to bridge interfaces are determined using bridge interface data. The number of bits an interface transmits or receives per cycle is rounded to an integer multiple of **cell_size**, and the channel is sized at this value. Single-beat messages (transmitted over the interface in a single cycle) are translated into a single flit packet. Multi-beat messages are translated into a multi-flit packets. Channels carrying multi-flit packets are rounded up to the nearest power-of-two multiple of **cell_size**. This is because two or four packet flits can be merged into a larger flit, or a large flit can be divided into multiple flits. Thus, all router port channels carrying single-flit packets are integer multiples of **cell_size** wide, and router port channels carrying multi-flit packets are power-of-two multiples of **cell_size** wide.

A similar convention is used for router directional channels, with the exception that a variety of messages can traverse the channels between various interfaces. Thus, all transaction messages and the corresponding packets are examined. If all packets are single-flit packets, the width can be an integer multiple of **cell_size**. If any packet is multi-beat, the width is rounded to nearest power-of-two multiple of **cell_size**. The channel widths can be increased during NoC optimizations using the bandwidth requirements and flit size distributions.

For more information on channel optimizations, refer to the **tune_links** and **analyze_links** commands. Refer to the VC Mapper section for a step-by-step detailed channel-width assignment

process. A global setting—**max_channel_width**—restricts the NoC channel widths, and it can be viewed or modified with the **mesh_prop** command.

2.4 CLOCKING

2.4.1 Clock Domains and Clock Crossing

Figure shows a NocStudio *clock-domain view*. The panel on the left shows a primary NoC layer and the panel on the right shows the Regbus layer. At bottom right is a legend showing this design's four clock domains. Host and bridge clock domains can be specified in NocStudio. Designers can also select an area of the chip in NocStudio and assign a clock domain to that area. The hosts in dark blue and the bottom left portion of the chip have been assigned the *other* clock domain. In the bottom right is an *apbbr* clock domain at 200 Mhz.

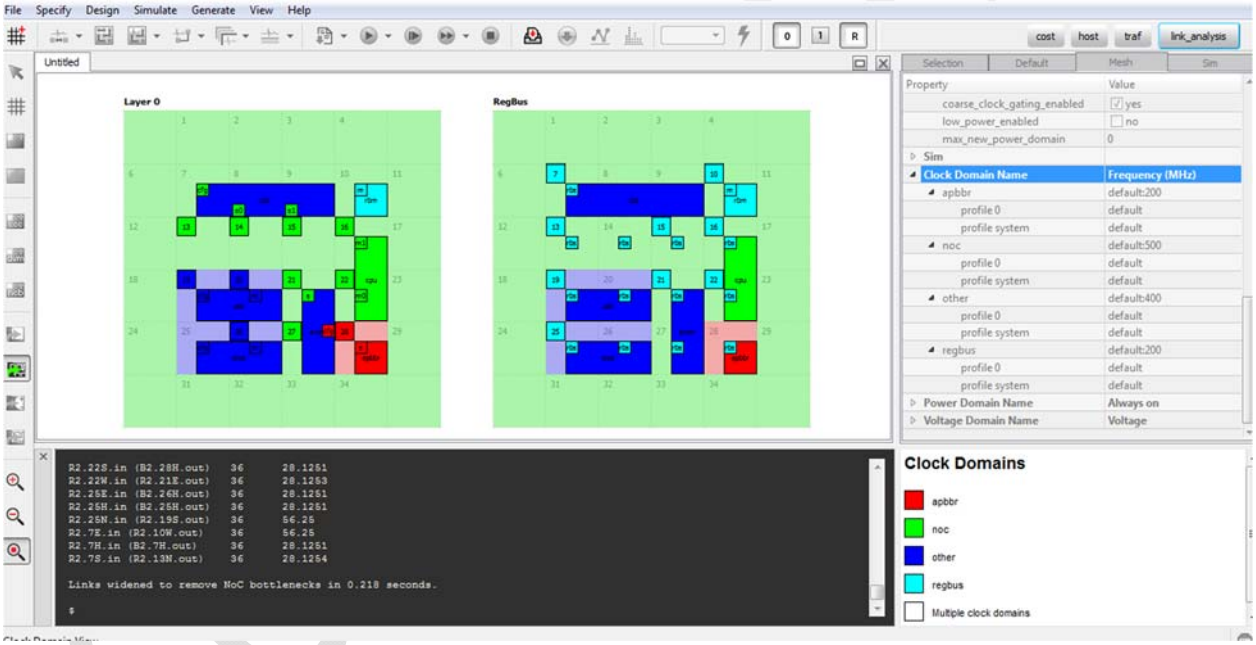


Figure 9. Multiple Clock Domains and Clock Crossings

The Regbus clock domain is fixed and asynchronous with the primary-layer clock domains. Therefore, all Regbus elements on the right panel are shown in the Regbus clock domain.

On the left panel displaying the primary layer, the routers in the *other* and *apbbr* clock domains use same clock. NocStudio assigns the specified clock domain to those routers and computes the link bandwidths and utilization with the specified domain frequency. The NocStudio performance simulator also uses domain frequency to dynamically model the transport and queuing in different NoC clock domains.

Bridge and host clock boundaries must be specified by the user and can be synchronous or asynchronous. NoC fabric clock boundaries are determined by NocStudio using the constraints in the clock-domain area specification supplied by the user. When crossing a clock-domain boundary between routers #20 and #21 in below figure, an asynchronous boundary is inserted at the router inputs.

Clock crossing between hosts and the NoC happen within a bridge. A list of clock crossing exists is generated by NocStudio in the NoC Reference Manual. There are different kinds of clock crossings. The "async" clock crossing refers to an asynchronous clock, where the frequency and phase of the clocks have no necessary relationship.

The "ratio_slow" and "ratio_fast" are phase-aligned synchronous clock crossers with an N:1 or 1:N ratio. "ratio_slow" refers to a clock crosser where the host is running slower than the NoC. The "ratio_fast" refers to a clock crosser where the host is running faster than the NoC. The synchronous clocks crossers require a frequency as well as a phase relationship. To achieve phase alignment, it is expected that the source of the ratio clocks will be the same. This structure works regardless of whether the source clock is faster, slower, or equal to the destination clock. The only requirement is that transfers happen only on a known shared rising edge. An external enable signal must be provided that indicates the cycle before the shared rising edge. This signal will be used to allow a transfer from the source clock domain to the destination clock domain.

Regbus clocks the ring at the primary-layer frequency used at the node, and the clock boundary between the ring and the Regbus frequency is implemented at the RingMaster. All primary layer NoC elements at a node use the same clock frequency, although they might have a clock boundary at their interfaces.

2.4.2 In-Link Domain crossing (ILDC)

Any router-router or router-bridge link of the NoC can be configured to enable asynchronous clock domain crossing. This is an alternative to performing asynchronous clock domain crossing within router or bridge input VC buffers. ILDC partitions the NoC link into two independent domains with separate design hierarchies. Position of this structure on the link can be specified through NocStudio. The figure below shows that the structure is virtual channel aware and comprises of WR and RD partitions, which can be included into different module groups according to domains.

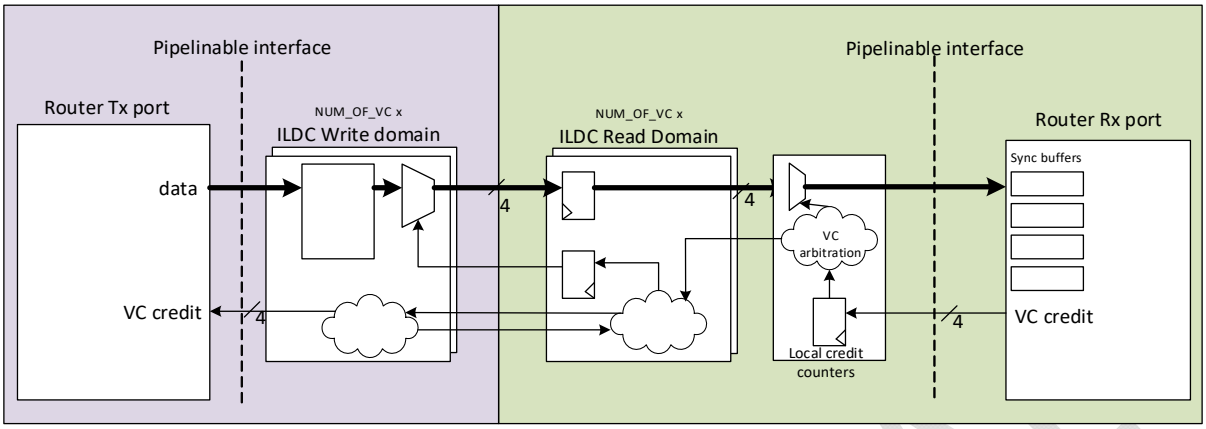


Figure 10 In-Link Domain crossing structure

2.4.2.1 *Automated domain crossers*

In earlier releases of NocStudio, routers containing one or more async ports were not skipped during mapping. It is now possible to skip the routers which have async ports, and instead have Domain Crossers be present across the async link. This is done through the *prop_default keep_routers_for_async*, which is set to "yes" by default, meaning that async routers are not skipped. When this prop is turned "no", the async routers are skipped and instead, there are Domain Crossers added on the async links, under appropriate conditions.

For an asynchronous link, the Domain Crossers are added automatically in four cases:

i. When the keep_routers_for_async is set to "no" and the length of the link is more than 1000 um.
ii. If the port is present on a router of an ecc or parity enabled layer
iii. If the link is across two different RTL groups
iv. If the link crosses voltage domains.

2.4.3 Clock Gating

NocStudio and the generated RTL supports clock gating of NoC elements such as bridges, routers, and pipeline flops. There are two types of clock gating supported:

1. **Coarse-grained clock gating:** This is done at the NoC-element level based on its activity.
2. **Fine-grained clock gating:** This is done at the flop level within a NoC element.

Each NoC element has a clock input pin that connects to the element's clock-distribution root. The synthesis tool does fine-grain clock gating by exploiting logic-level opportunities to insert local clock gating at the flop level. Coarse-grain clock gating is enabled using the NocStudio property:

```
mesh_prop coarse_clock_gating_enabled yes
```

By default, this property is set to **no**.

Below figure shows NetSpeed NoC clock gating.

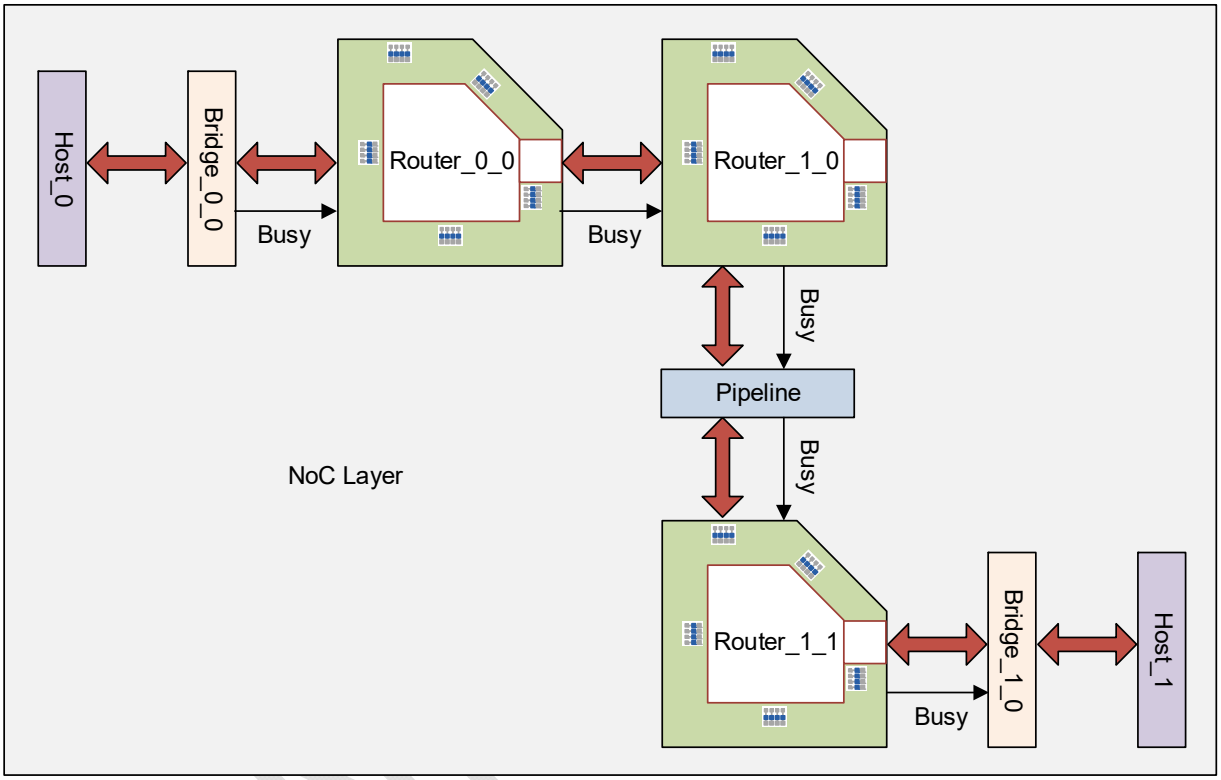


Figure 11. Clock Gating as Implemented in NetSpeed NoC

2.4.3.1 Coarse-Grained Clock Gating

Applying coarse-grained clock gating to NoC elements saves power when those elements are idle for a period of time. This differs from fine-grained clock gating (usually done during logic synthesis) which controls clock gating on a cycle-by-cycle basis. Coarse-grained clock gating shuts off entire clock tree branches within a NoC element, saving power in ungated flops and in the clock network itself. Coarse-grained clock gating can be applied to the following elements:

- Routers
- AXI Bridges
- Pipeline stages inserted by NocStudio

- Regbus routers

Coarse-grained clock gating shuts off NoC elements under the following conditions:

- There are no transactions buffered or being processed internally, and all credits have been returned from the neighboring blocks.
- There are no transactions buffered and none inbound to the NoC element from a neighboring block.
- The inactivity described above has persisted for the programmed number of cycles.

A clock-gated NoC element awakes from its clock-gated state when a flit bound for the element is detected at a neighboring element. Because NoC elements can have registered or unregistered outputs, their internal pipelines differ and therefore have a different wake latency. An extra latency cycle can be incurred each time a NoC element wakes.

NoC bridges have a common counter that generates heart beat pulses at a programmable interval. These pulses are synchronized to different clock gating domains in the bridge. Within a bridge CG domain, four consecutive heatbeat pulses is used as the interval over which inactivity will initiate coarse clock gating of the domain. Similarly, NoC routers have a hysteresis counter value that determines the number of inactive cycles needed for a router to clock-gate itself. These intervals should be set high enough such that a few cycles of expected inactivity do not send the element into a clock-gated state, increasing the average path latency by forcing most packets to wake the element. Default value of hysteresis/hear-beat counters is 100 cycles; this default value can be configured through NocStudio properties. Further, the count values are programmable registers in the NoC elements and can be updated through regbus writes. Coarse-grained clock gating is managed by hardware, but in each NoC element, a programmable override register is provided to disable this feature.

3 Functional Description: Non-Coherent Components

This section describes the non-coherent components of the NoC, that is used for rapidly designing and analyzing highly efficient and scalable non-coherent interconnects for a wide variety of SoCs.

3.1 Bridging from Host to NoC

A bridge can connect a master of slave block to the NoC and perform the required operations to support the master and slave communication as per the AMBA protocol standards. Master bridges exist for ACE, ACE-lite, AXI4, AXI3, AXI-Lite, and AHB-Lite protocols. Those protocols also have slave bridges for connecting the NoC to slave devices. In addition, an APB bridge exists for attaching APB slave devices. Bridges packetizes the host blocks transactions into NetSpeed packet format during injection into NoC and de-packetizes them during ejection.

A host can have multiple AMBA ports for transmitting and receiving data to/from the NoC. A bridge component converts the host-port messaging protocol into a packetized protocol for the NoC. Bridges are automatically instantiated by NocStudio based on the specified host-port protocol. There is one bridge per host port, and the bridge connects the host port to routers at the mesh grid points. Multiple routers can exist at a mesh grid point, one router for each NoC layer. In that case the bridge connects the host port to each router.

Bridge parameters and properties are assigned by NocStudio based on the high-level specification of traffic and hosts. Some bridge properties are made visible to the user. Those properties can be modified with **bridge_prop** commands in NocStudio. Refer to this command for the list of user-modifiable bridge parameters. Bridges are designed and optimized for low-latency and high-frequency operation. Address lookup, route-information encoding, QoS, protocol-related conversions and processing, etc., are all tuned and configured with NocStudio based on optimizations or the user specification.

AXI4, AXI3, AXI-Lite, AHB-Lite, and APB (v2, v3, v4) bridges are all supported by NocStudio.

3.1.1 AXI4 Master Bridge

Figure 12 shows a block diagram of the NetSpeed AXI4 master bridge. The master bridge contains a layer for AXI4 protocol processing, and a switch layer that communicates with routers in up to sixteen NoC layers.

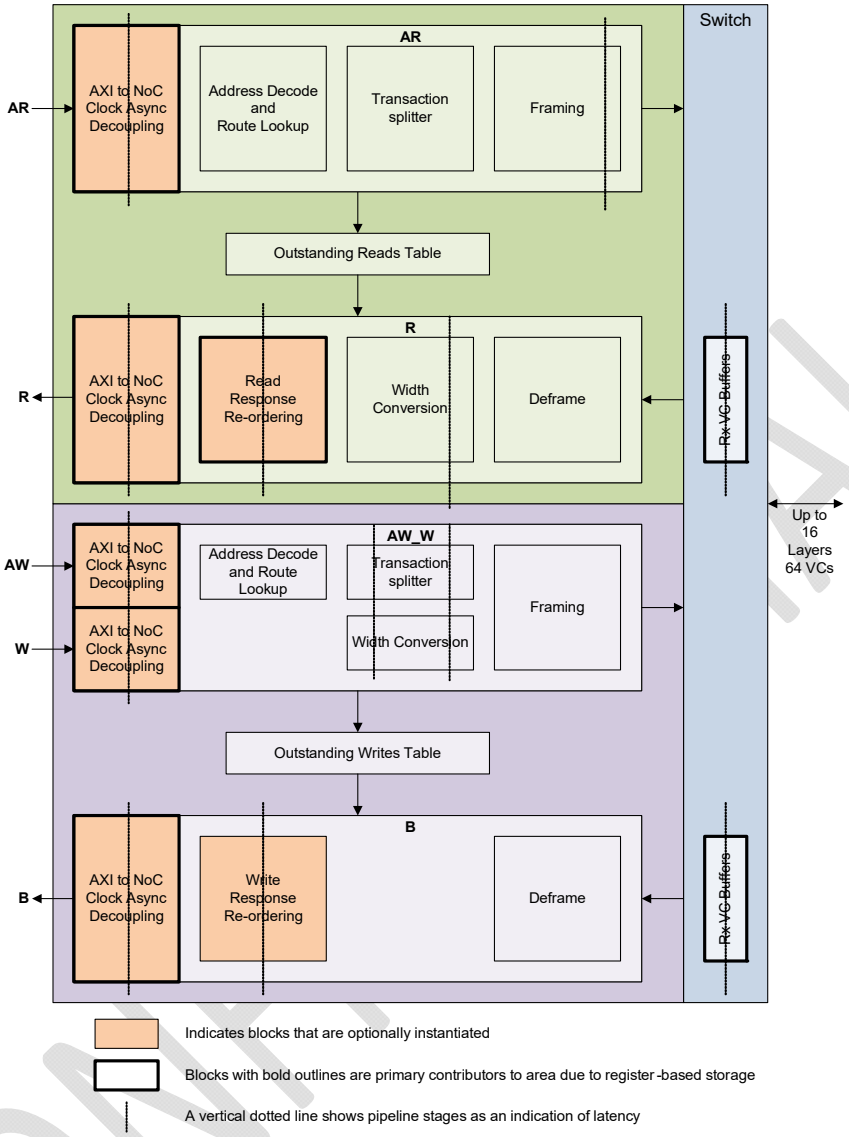


Figure 12. AXI4 Master Bridge

The main features of the master bridge are listed in the following sections.

3.1.1.1 AXI4 Channels and NoC

- The master bridge provides an AXI4 slave interface to a host AXI4 master port. Five standard AXI4 channel—read command (AR), write command (AW), write data (W), read response (R) and write response (W)—are supported.
- Up to sixteen physical NoC layers with four virtual channels each are available to transport the AXI4 channels. These are allocated by NocStudio based on QoS, bandwidth, deadlock, and other requirements.
- The AR, AW, and W channels are packetized and transmitted on NoC layer virtual channels and packets are received from NoC layer virtual channels for R and B channels.

- A write command from an AW channel and the corresponding write-data burst from a W channel are combined into a single packet. AW could be sent as the side band of the write packet for higher write bandwidth or could be sent as a header in-band with the write packet for lower area.

3.1.1.2 Decoding and Routing

- Command addresses on AR and AW channels are used to decode the destination slave device. AXI QoS is used to determine the associated NoC QoS through on a configured table. NoC QoS and optional addressing hashing is used to determine the destination, physical route, virtual channel, and NoC layer for transmitting the transaction.
- Slave devices are identified by address ranges specified in the form of a base address and mask, OR lo-hi formats. The number of address ranges accessible by a master bridge are set by NocStudio from user specifications. A base address and mask corresponding to these ranges are held in master bridge programmable registers. Multiple address ranges can be specified for a single slave and these can have different access privileges.
- Address ranges can be programmed as disabled, read-only, or write-only. During address decode, the transaction ARPROT/AWPROT is compared with the access privilege programmed for an address range. A failed access check results in a decode error response for the transaction.
- Each address range can also be associated with hash functions which are used in the destination/route lookup process
- Address ranges also have a defined priority, allowing multiple matches in the table to be resolved to the higher priority match
- Address look up can also be configured to yield a relocated address which should be sent to the selected slave.

3.1.1.3 Flow control

- On AXI4 channels, ready/valid flow control specified by the standard is implemented. On the router side, credit-based flow control is performed on the virtual channels.

3.1.1.4 AXI4 Specific Features

- The master bridge supports a configurable number of outstanding transactions on the read and write channels.
- Logic for ordering R and B responses processed out-of-order by the network for higher performance can be optionally instantiated on the master bridges. Responses to the master are returned in the order that the requests were received.
- INCR transactions are split at specific address boundaries based on certain rules described in section 1.9
- FIXED transactions are split into multiple single beat INCRs

- WRAP transactions different from 64-byte, 32-byte, or 16B cache-line size are considered a fatal error, and handling by the NoC is not guaranteed. An interrupt is raised to indicate this fatal error.
- Split transaction responses are transparently coalesced to match the original master command. Optionally, read response segments of a split transaction can be interleaved with transactions with different AID
- R and W data channels are processed based on the commands supporting width conversion when communicating with AXI slaves having different AXI data widths.

3.1.1.5 Errors and Stalls

- An unknown-address or access-privilege violation on the AR or AW channels causes a decode error that stalls the command channels until the DECERR response can be issued on the R or B channel, respectively.
- Read/write transactions to a slave device cause a decode error if the corresponding read/write traffic was not specified through NocStudio.
- Changing the QoS level while commands are outstanding on that AID can momentarily stall the channel if the change reorders the command to a slave over the network.
- If response reordering logic is not included, then a temporary stall occurs if a command sequence can cause network reordering of the responses.

3.1.2 AXI4 Slave Bridge

Figure 13 shows a block diagram of the NetSpeed AXI4 slave bridge. The slave bridge contains a layer for AXI4 protocol processing, and a switch layer that communicates with routers in up to sixteen NoC layers.

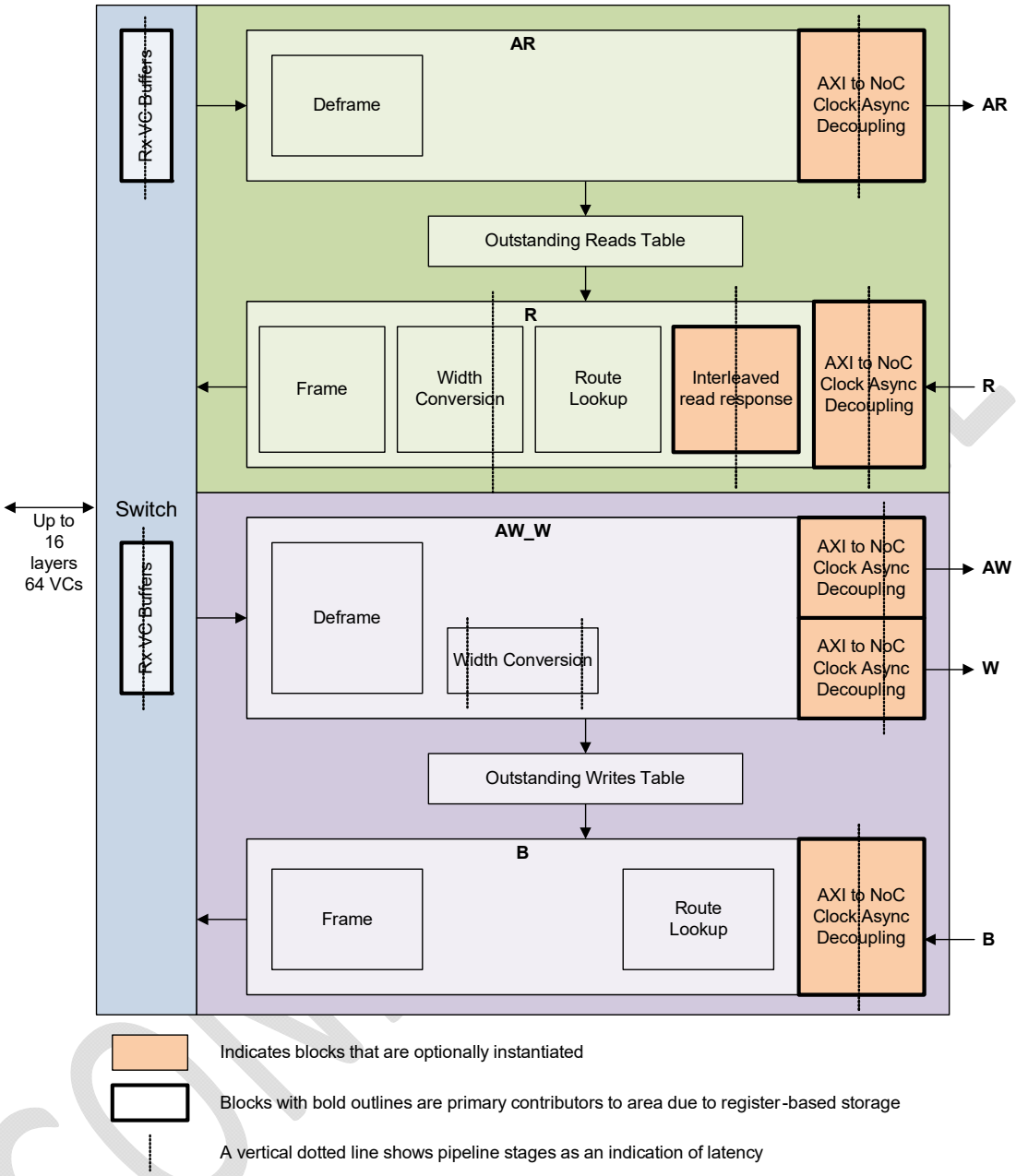


Figure 13. AXI4 Slave Bridge

The main features of the slave bridge are listed in the following sections.

3.1.2.1 AXI4 Channels and NoC

- The slave bridge provides an AXI4 interface to a host AXI4 slave port. Five standard AXI4 channel—read command (AR), write command (AW), write data (W), read response (R) and write response (W)—are supported.

- Up to sixteen physical NoC layers with four virtual channels each are available to transport the AXI4 channels. These are allocated by NocStudio based on QoS, bandwidth, deadlock, and other requirements.
- The R and B channels are packetized and transmitted on NoC layer virtual channels and packets are received from NoC layer virtual channels for AR, AW, and W channels.
- A write command from an AW channel and the corresponding data burst on the W channel are de-packetized from a single packet.
- Optionally AR and AW interfaces can be configured to have host virtual channels with credit-based flow control. NoC virtual channels are mapped to host virtual channels using configured tables

3.1.2.2 Decoding, Routing, and Flow Control

- The ID and QoS from the original AR and AW commands are retained in the slave bridge to route the corresponding R and B responses back to the master.
- On AXI4 channels, ready/valid flow control specified by the standard is implemented. On the router side, credit-based flow control is performed on the virtual channels. Optionally AR/AW channels can have virtual channels with credit-based flow control.

3.1.2.3 AXI4 Specific Features

- The slave bridge supports a configurable number of outstanding read and write commands that can be issued to the attached slave device.
- Width conversion is supported for R and W data channels, enabling communication with AXI masters that have different AXI data widths.
- Slave bridges can optionally instantiate a block for processing interleaved read responses if the attached slave device requires it.

3.1.3 AXI3 Master and Slave Bridge

AXI3 master and slave bridges are also supported in AMBA NoCs, as a variant of the AXI4 bridges. Write-interleaving as specified in the AXI3 standard is not supported. Locked transfers in AXI3 are not supported.

3.1.4 AXI4-Lite Slave Bridge

The AXI to AXI4-Lite bridge translates incoming AXI transactions into AXI4-Lite transactions. This module is used along with the AXI4 slave bridge to connect the NoC with an AXI4-Lite slave with the following features:

- Configurable 32-bit or 64-bit AXI4 master interface on the ingress side.
- Configurable 32-bit or 64-bit AXI4-Lite on the egress side.

Figure 14 shows a block diagram of the AXI to AXI4-Lite Bridge. On the ingress side, the bridge is compliant with the AXI4 interface specification. On the egress side it is compliant with the AXI4-Lite interface specification. To simplify address definition and decode of downstream bridges, this bridge can be configured to send the AxREGION bits. Note that AxREGION bits are not part of AXI4-Lite interface. The bridge can also be configured to pass ARSIZE to allow narrow read access on the slave. Narrow write accesses can be done using WSTRB and does not require AWSIZE.

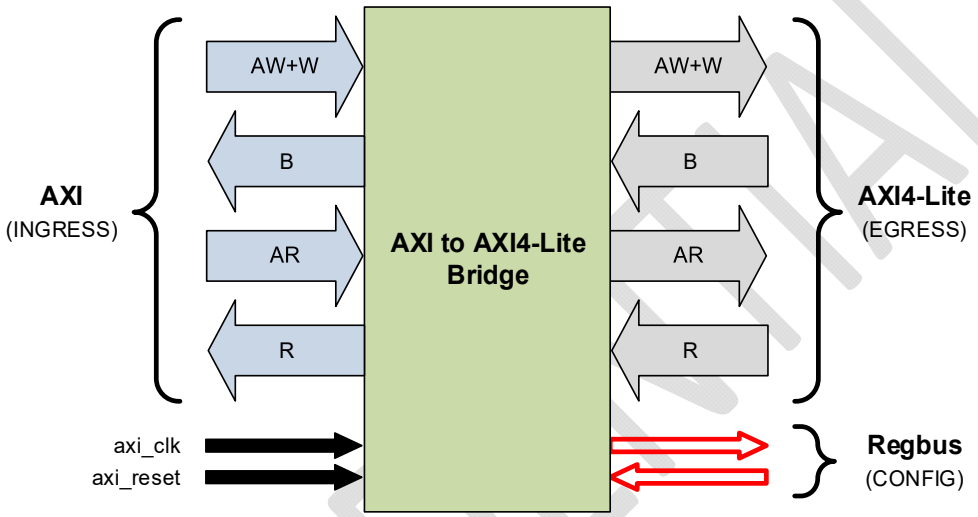


Figure 14. AXI to AXI4-Lite Bridge

3.1.5 APB Bridge

Figure 15 shows a block diagram of the AXI to APB bridge. The AXI to APB bridge provides the capability to attach legacy APB slaves to AXI4 masters. This layer is used with the AXI4 slave bridge to form a NoC to APB bridge. The bridge translates incoming AXI4-Lite transactions into APB transactions. It has the following features:

- One 32-bit AXI4 ingress port.
- Up to 16 APB slaves on the egress port.
- Each APB slave port is configurable to support APB2/APB3/APB4 peripherals.
- 32-bit wide APB interface selection per client.
- AxREGION based PSEL generation.
- Independently configurable buffer sizes for request and response channels.
- Define secure slaves, which are protected by the bridge from non-secure transactions.

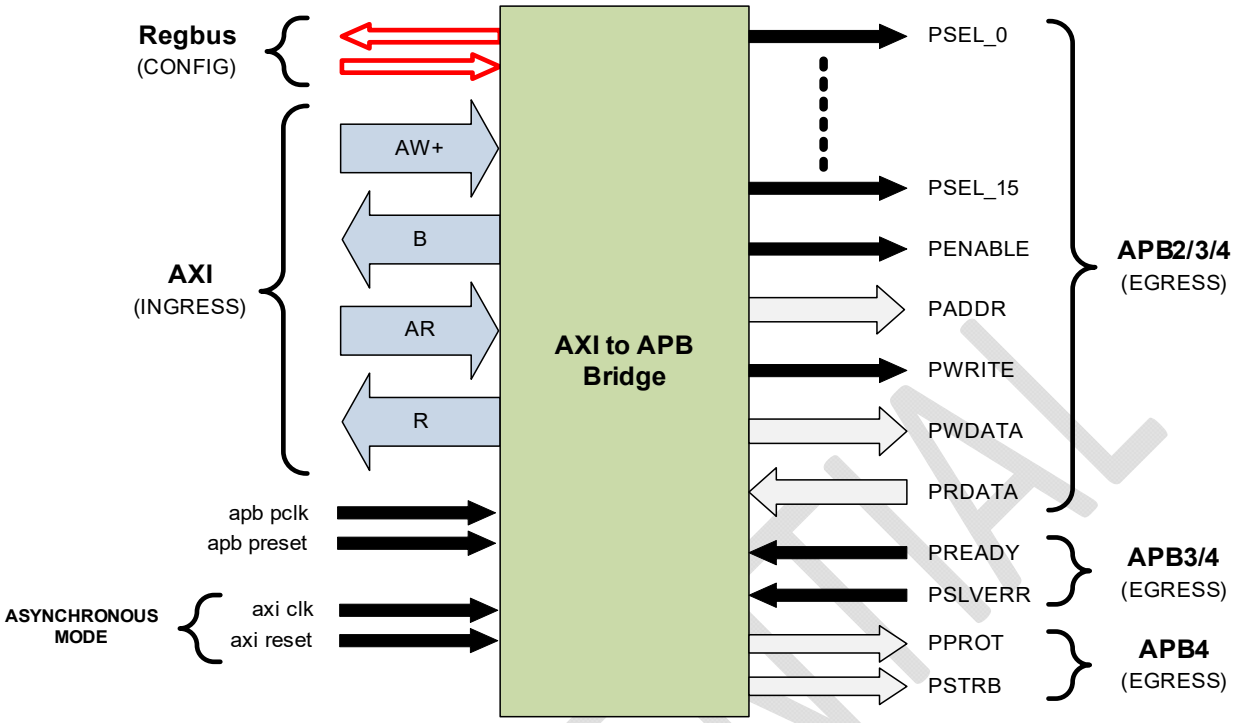


Figure 15. AXI to APB Bridge

On the egress side, the bridge is compliant with the AMBA APB2/3/4 specification. On the ingress side it is compliant with the AXI4-Lite interface specification.

3.1.6 AHB-Lite Master Bridge

NetSpeed Gemini supports the AHB-Lite standard. An AHB-Lite master connects to the NoC using the AHB-Lite to AXI bridge in series with an AXI4 master bridge. An AHB-Lite subsystem consists of a master, slaves, decoder, and multiplexer/arbiter. The AHB-Lite master bridge includes the logic for a slave, decoder, and multiplexer/arbiter, and exposes a *mirrored master interface*. This enables an efficient point-to-point connection with an AHB-Lite master device.

Figure 16 shows a block diagram of the AHB-Lite to AXI master bridge. The following summarizes the features:

- AHB-Lite mirrored master interface.
- Data widths of 32-bit, 64-bit, and 128-bit. Address widths of 32-bit and 64-bit.
- Writes are bufferable or non-bufferable depending on HPROT[2].
- HPROT[2] can be overridden to force non-bufferable write behavior.
- Single read outstanding.
- Configurable number of outstanding bufferable writes.

- 1KB address boundaries per AHB transaction, and the NoC splits transactions into 64-byte chunks.

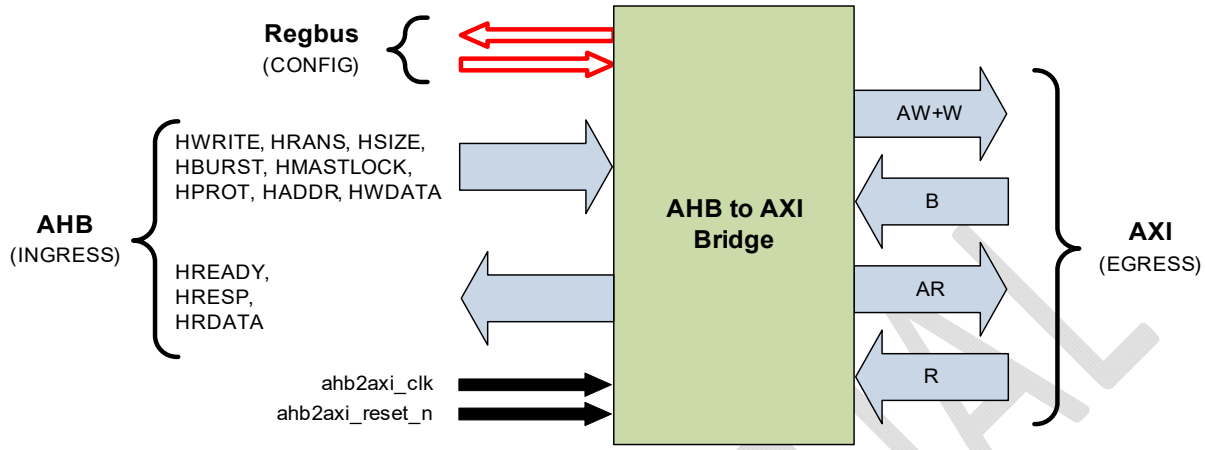


Figure 16. AHB-Lite to AXI Master Bridge

3.1.7 AHB-Lite Slave Bridge

Figure 17 shows a block diagram of the AXI to AHB-Lite bridge. The following summarizes the features:

- AHB-Lite mirrored slave interface.
- One to sixteen AHB-Lite slaves can be connected to a single AHB-Lite slave bridge.
- Data widths of 32-bit, 64-bit, and 128-bit. Address widths of 32-bit and 64-bit.
- Slaves of different data widths can be connected to the same converter.
- The AHB-Lite slave bridge handles transaction conversion from AXI4 and AXI3 masters on the NoC. Transfers must be address-aligned to the AHB-Lite interface HSIZE requirements and cannot have checker board (0101) write strobes.
- The NoC implements AxREGION-based decode when more than one AHB-Lite slave device is on an AHB-Lite slave bridge. REGION IDs are provided to NocStudio at the time of NoC specification, and the AHB-Lite slave bridge generates the required HSELs.

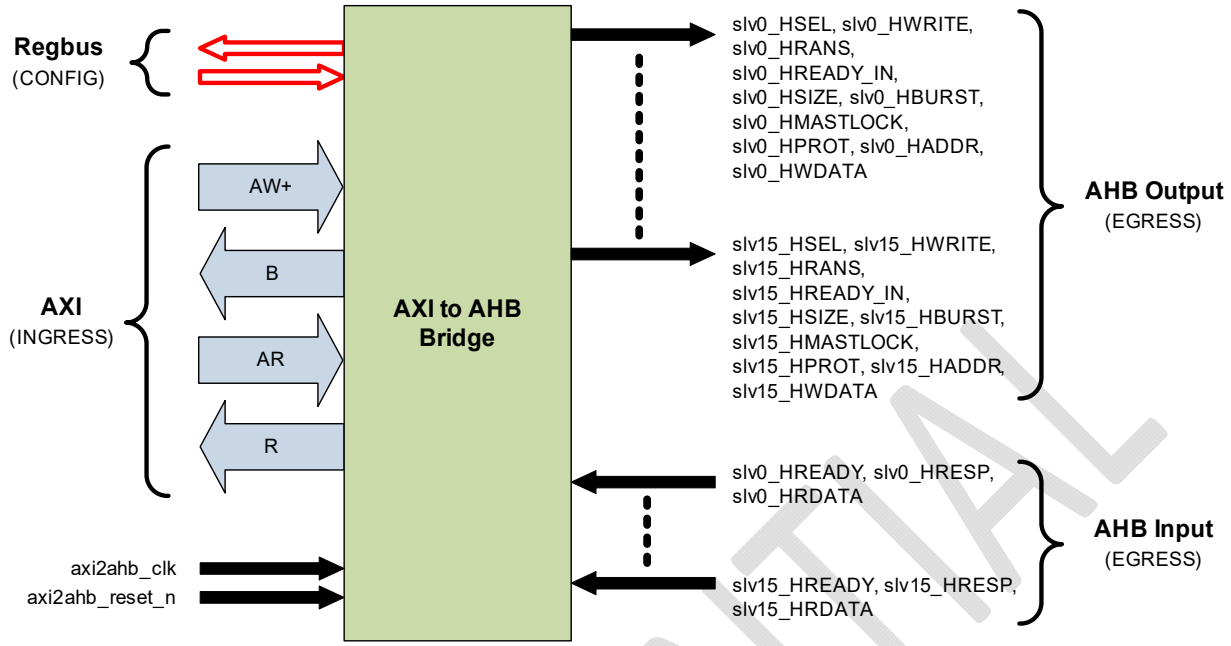


Figure 17. AXI to AHB-Lite Bridge

3.1.8 Shared interface bridge

This block acts as a master port aggregator allowing several master ports to be aggregated into a single master port which then connects to the NoC through a common master bridge. This allows multiple master ports to share logic for packetization, clock conversion, ordering, switching etc. This also allows a host with multiple master ports to connect to the NoC through a master bridge at a single grid point instead of spreading out over multiple grid points with a master bridge per port. Each port of the SIB can be of a different data width and can be of AXI4 or AXI3 type.

To the master bridge, a SIB appears like a normal AXI4 master port. It is important that when identifying candidates for grouping, user understands the bandwidth requirement for each master so that combined traffic doesn't exceed the bandwidth of the master bridge.

Transaction from narrow ports are sent as AXI narrows on the aggregated port. Write data is not interleaved, each transaction must be finished before the next port can get access. So idle cycle from a low bandwidth master port can affect bus utilization. SIB is ideal for grouping low bandwidth masters' ports of similar data size.

SIB can support aggregation of up to 16 master ports. SIB also implements a feature where two ports can be specified as mirrors of each other. SIB checks that the two master interfaces match every cycle, any mismatch is reported as an error.

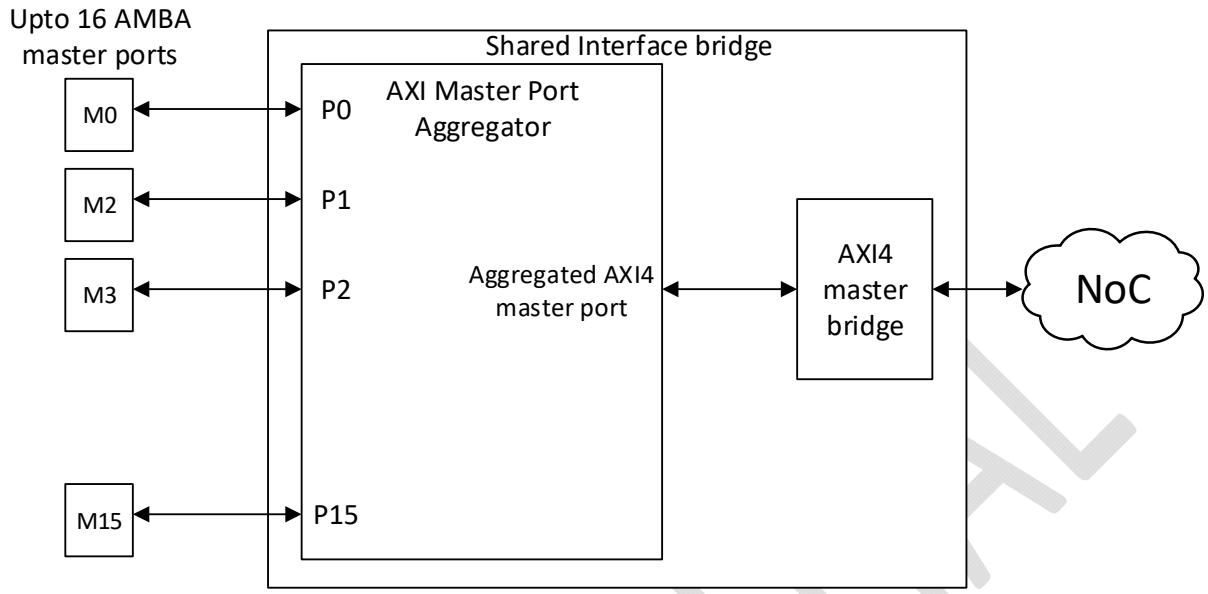


Figure 18 Shared Interface Bridge

3.1.9 Reorder Bridge

The reorder bridge is an agent type supported within the NetSpeed Orion AMBA and Gemini NoCs. It acts as a convergence point for traffic going from masters to slaves. The purpose of this agent is to allow the sharing of resources in the NoC by multiple agents. This is an optional component that can be used to reduce total area cost within the system.

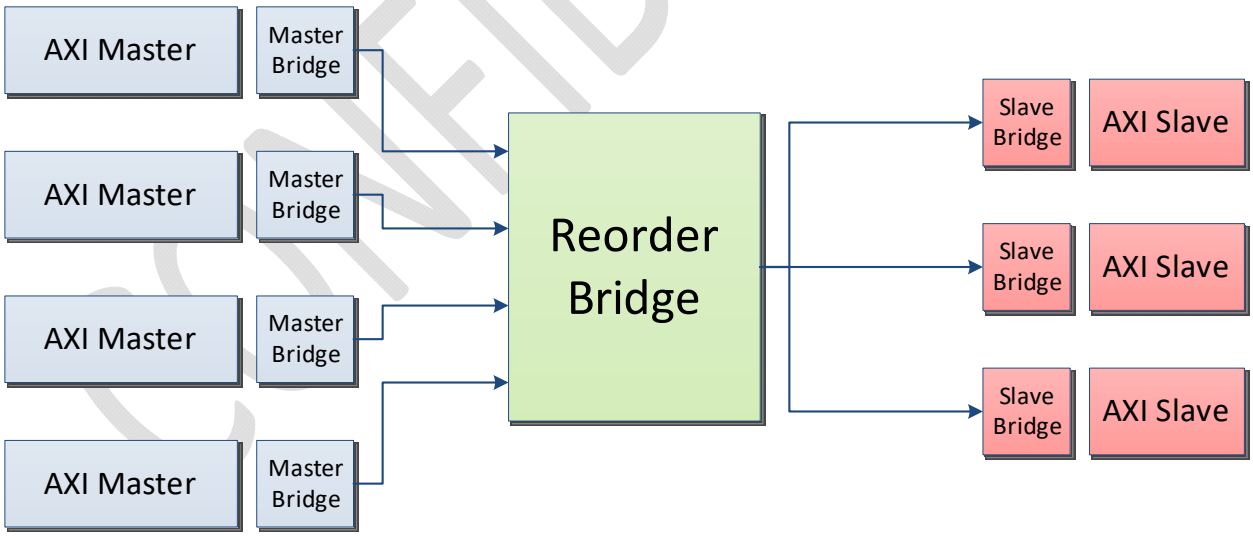


Figure 19: Reorder Bridge NoC Component

Each master port can be configured to communicate to a Reorder Bridge. The Reorder Bridges can be placed anywhere on the chip just like any other agent, although it is best located at intermediate points between the masters and slaves.

The reorder bridge can reduce hardware by allowing resources to be located at the reorder bridge instead of in the master bridges themselves. One primary example of this is the reorder buffers in the system. By locating them at this convergence point, they can be shared by all of the masters, allowing a better dynamic utilization of resources, and ultimately requiring few total resources.

Multiple Reorder Bridges could be used in the system. Since all requests from the masters are diverted to the reorder bridge, it can become a bandwidth bottleneck in the system. Multiple Reorder Bridges would increase the potential bandwidth.

Also, since request will first have to be diverted to a Reorder Bridge, the latency of those requests will increase by some amount. If the Reorder Bridge is not located between the masters and the slaves they are talking to, it will incur even higher latency costs. In a large system, multiple Reorder Bridges could be used to reduce these latency costs.

3.1.9.1 Reorder bridge benefits

3.1.9.1.1 Reducing Reorder Buffer Entries

Reorder buffers may be needed in a system where agents reuse AxIDs in their requests. If traffic is sent to different slaves, but with the same AxID, the only way to ensure the response is returned in order is to preallocate storage in a reorder buffer, so if responses are returned out of order, the new responses can wait for older responses.

The Reorder Bridge allows for the reorder buffers to be located in a single place, shareable by all of the masters connected to it. Without this, each agent would have to size their reorder buffers based on their worstcase traffic expectations. But it is unlikely that all agents can hit their worstcase traffic loads simultaneously. So, by creating a common pool of resources that are dynamically allocated based on actual use can reduced the total storage needed.

On top of the dynamic utilization benefit, having a single structure for the reorder buffer allows a better physical design of the reorder buffers, potentially reducing cost by even more. A register file may be more likely to be used in a configuration with many more entries.

3.1.9.1.2 Reducing AID table storage

The AID table has three major functions:

- tracking of requests by AxID to determine if serialization is necessary when requests are sent to different targets.
- hold any width-conversion information so that read responses can be properly modified for the interface.
- to track outstanding requests and have timeout monitors for each to determine if a request is stuck.

The AxID table of a master bridge is often a significant amount of storage, so reducing the area by eliminating the AxID table is very useful. The reorder buffer enables this by pooling these capabilities at a single location where they can be shared.

3.1.9.1.3 Reducing Address Map Storage

Another advantage to the Reorder Bridge is that the address maps at the requesting agents may be simplified, since all requests can be sent to the Reorder Bridge and decode errors can be handled there.

3.1.9.2 Using the reorder bridge

3.1.9.2.1 Instantiating the Reorder bridge

NocStudio should allow the instantiation of the reorder bridge, and masters have an additional property set to indicate that they should send all of their read/write traffic to that reorder bridge. The reorder buffer of the Reorder Bridge is user determined for sizing.

3.1.9.2.2 Address Map and Security Limitations

Any address map or security details that would normally be assigned to a master must be assigned to the Reorder Bridge instead. Since the Reorder bridge will not distinguish between the masters, the same security controls must apply to all agents connected to the bridge.

3.1.9.3 Limitations and implications of using the reorder bridge

3.1.9.3.1 Unwanted Connectivity Security Hole

If a master sends requests to the Reorder Bridge, the requests could be sent on to any slave the Reorder Bridge is connected to. This creates a potential security hole because it would allow a master to connect to a slave when traffic wasn't explicitly set up in NocStudio to allow it. If there are masters with less connectivity than other masters talking to the same Reorder Bridge, a security hole is created. Local address maps would fix this by preventing requests to the restricted slaves.

3.1.9.3.2 Security Filtering Differences

NetSpeed security filtering is handled at the master bridge. If no address table is present in the master bridge that goes to a ROB, it will inherent security filtering at the ROB. If different masters have different security requirements, this would cause a security violation. Also, these security features are usually programmable at the master bridge. Removing an address table at the master bridge will prevent this kind of control.

3.1.9.3.3 Power Management (Fence/Drain)

Without an AID table, power management is a little different for a master. Typically, the AID table would track each request, and the power domains that each request is using. Since the Reorder Bridge now provides the AID table functionality, the master effectively only sees one target, which is the Reorder Bridge. Instead of needing to track each request separately to determine the power requirements, the master bridge can just keep a count of outstanding transactions. If any requests are outstanding, the power domains to the Reorder Bridge remain active.

3.1.9.3.4 Timeout Handling

When using a reorder bridge, since the master bridge only keeps a count of requests, it can do a very coarse-grain timeout. The Reorder Bridge that holds the AID table therefore handles more detailed/fine-grained timeouts. There is a small difference since the request does not start being tracked until it arrived at the Reorder Bridge, but that will typically be only a few cycles. This mechanism allows the timeout to detect any broken slaves, for instance.

3.2 Address Maps and Configurability Options

3.2.1 Address Maps

Address map is used to define the connectivity between masters and slaves. Address ranges can be specified at slave bridges. Each range has a specified name, a target slave port, and the address range itself. A slave device can have multiple address ranges assigned to it. Address ranges can be non-continuous.

When traffic is specified between a master and a slave, the connectivity between them are automatically build. It is possible to use address range to selectively build connection between a master and specific address range in a slave.

Each address range can be disabled or enabled through register access if register access is enabled in the configuration. The value of address range can be programmed if programmability is enabled for the address range.

There are two ways to specify address ranges; low/high and base/mask. Specifying with low/high uses lower-bound address and upper-bound address for the range. Base/mask pair provide more flexibility to express address interleaving. For example, it's possible to specify 1GB address space with interleaving at 64B boundary.

3.2.2 Address Relocation

In Many cases, the capability to override the incoming address value is useful. The address relocation is used to achieve this operation.

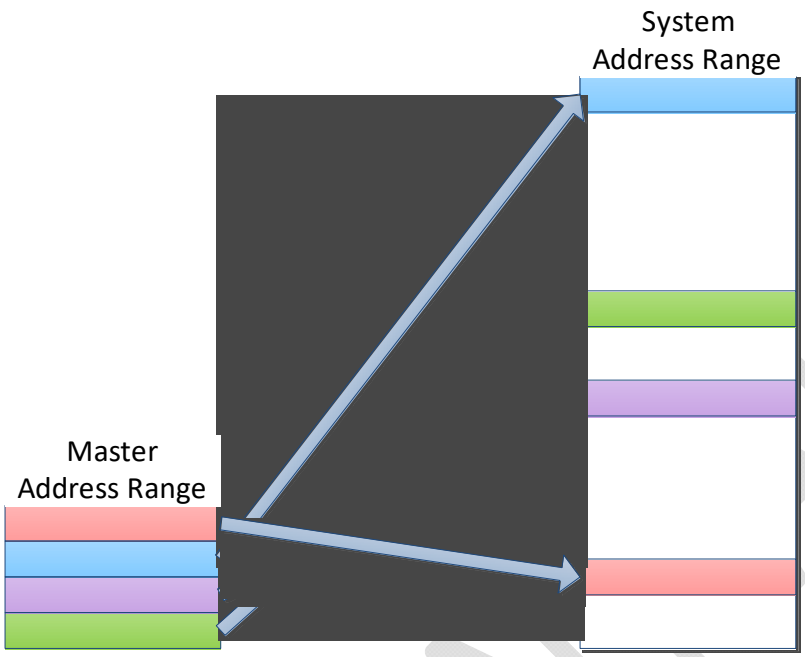


Figure 20: Master with small address space can access regions in a much larger address space

In the diagram above, the master range is shown on the left with a much smaller range than the system address range. However, each sub-range can be individually programmed to map to a location within the system address range.

This operation can be performed by assigning address ranges to the master bridge with a relocation value. The relocation address overwrites the original address and the pack is sent with the system address. When master address width is smaller, additional bits are added on top of the original address.

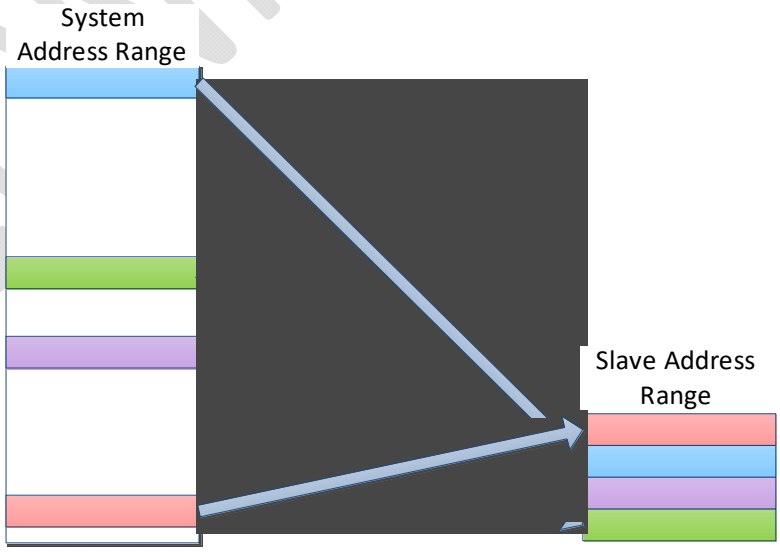


Figure 21: Multiple system address ranges can be compressed to appear as a contiguous slave region

Similar situation can occur when system address is mapped to smaller slave address as above. The relocation address can override the address in this case as well.

3.2.3 Hash Function

When a memory range is shared between two similar devices, such as two channels of DRAM, the function used to select between the two ranges **Figure 3: Multiple system address ranges can be compressed to appear as a contiguous slave region** is important for effective bandwidth allocation. Using a single address bit, for instance, may not achieve a good distribution of requests over time. A hash function can be used to provide a more randomized distribution.

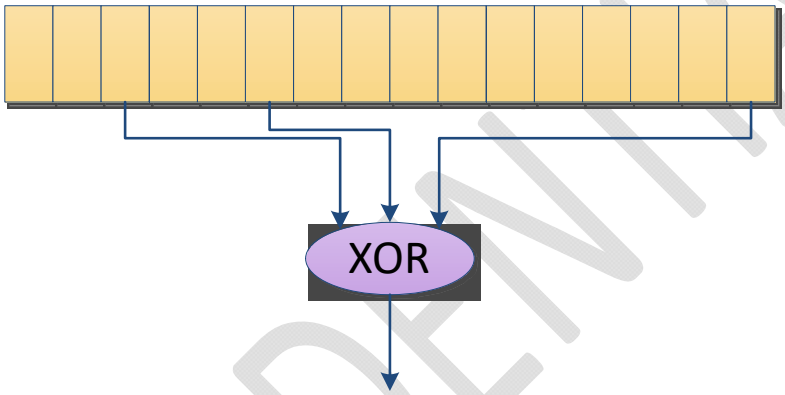


Figure 22: Simple XOR hash function

A common hashing function is an XOR of some of the address bits. By using a combination of upper order bits and lower order bits, distribution can be more randomized.

One requirement for hashing is that the address space is split evenly between the two targets. This allows each target to predictably handle ½ of the total address space. XOR automatically provides this even distribution. The hashed address ranges must be easily compressed to ½ size address ranges. The compression can be done by dropping one bit of each hash function from the total address space.

The hashing description works fine for a pair of targets where the address space is split evenly between them. More targets can easily be handled as long as they are power-of-2 in size. The easiest way to handle the larger set of targets is to use the same hashing function mechanism but using different bits for each hash function. So, a 4 target hash would have two hash outputs, each driven by non-overlapping set of hash bits.

Once hash function and a group of slaves with the hash function are defined, address range can be specified for the slave group. Hash bits can be configured to be programmable.

3.3 AID HANDLING AND TRANSACTION ORDERING

Master and slave agents in the NoC can specify different AID widths on their interface. The NoC ensures ordering of requests with the same AID from a master to a slave and ordering of all responses having the same AID back to the master.

Master bridge will enforce serialization to avoid potential AID ordering hazard. To reduce serialization, a reorder buffer can be enabled on a master bridge. The reorder buffer holds responses which have arrived out of order from the NoC and issues them back to the master in the correct request order.

In the absence of a reorder buffer, a new transaction having the same AID as an outstanding transaction is halted on the interface if it is destined to a different slave or is using a different NoC QoS compared to the outstanding request. When reorder buffer is present a request is serialized only if it uses a different NoC QoS compared to an outstanding request with the same AID.

At the slave bridge, AxID to the slave is derived from ID of the master sending the transaction and the original AxID of the transaction. Master's ID is represented in the number of bits required to binary encode the masters in the NoC. For e.g. in a NoC with 8 master agents MASTER_ID is represented by 3-bits. Transaction's original AxID is carried in a container whose size is equal to the widest AxID in the NoC, by padding with zeros on the MSB. At the slave bridge a configuration is available to pick the slave's AxID width number of LSbits from the concatenation {MASTER_ID, System AxID} or {System AxID, MASTER_ID}.

Another mode is available at the slave bridge, where every transaction outstanding to the attached slave has a unique AxID. This allows the slave to have no AID ordering logic. Two transactions with the same original AxID will get assigned unique IDs and so the slave could reorder them. To correct the response order, either the master bridges need reorder buffer or they need to guarantee than only transactions with unique AxIDs are outstanding in the NoC.

3.4 SPLITTING OF AMBA TRANSACTIONS

Master bridges split AMBA read and write transactions at specific address boundaries. Cases under which transaction splitting is triggered are enumerated below.

a. Coherent transactions from ACE and ACEL masters are split at 64B boundary
b. Transactions sent to slaves providing interleaved read responses are split at 64B boundary
c. Transactions from a master bridge configured to have a read reorder buffer are split at 64B boundary

d. Transactions from a master bridge with traffic to a slave supporting virtualized command interface or traffic to 'Reorder-Bridge' are split at 64B boundary
e. Non-coherent transactions are split at granularity which can be configured at each master bridge. Default value of this is 1024B
f. If the programmed address ranges on a master bridge have lower granularity than the configured split size, then the split size is overridden to match the smallest address range granularity

When transactions are split at a master bridge, responses returning for the split segments have to be merged. When re-order buffer is enabled, it is also used to coalesce the split response segments. Alternatively, the bridge can be configured to return read response segments from different AIDs in an interleaved manner.

Transaction splitting can lead to serialization, in the absence of re-order buffer or if the master doesn't support interleaved read responses. In this case a transaction that needs splitting will wait for all prior outstanding responses to return. Also, at the end of splitting, a new transaction will not be accepted till all outstanding split segments responses are returned.

3.5 WIDTH CONVERSION

Agents with different AXI_DATA_WIDTH can intercommunicate over the NoC. Bridges and routers perform data and command transformations required for correct and efficient communication between agents of different data widths.

Master and slave bridges issue transaction packets (commands, data, responses) into the NoC without knowledge of the receiving end's data width. Beats of narrow data transfers are packed and unused bytes in transactions with unaligned addresses are also removed prior to sending packets on the NoC. Write command can either be sent on NoC sideband for improved throughput or can be framed at the head of a write data packet for lower area.

For transfers from narrow agents to wider agents, routers perform upsizing of NoC data flits to deliver NoC packets at the wider width of the destination. Similarly, for transfers from wider agents to narrow agents, routers perform downsizing of NoC data flits on the path to destination.

AXI command AxLEN and AxSIZE are appropriately modified at the slave bridge. For example, for transfers from a wider master to a narrow slave, AxSIZE is reduced to the interface width of the slave and AxLEN is correspondingly increased. Similarly, for transfers from a narrow master to a wider slave, AxSIZE can be increased and AxLEN reduced.

AxCACHE[1] marks transactions as modifiable or non-modifiable. Modifiable transactions provide greater flexibility in NetSpeed NoC to transport and modify transactions passing

through the system for greater performance. Non-modifiable transactions are honored. However, some transactions marked as non-modifiable will still be subjected to modification, for example if width conversion operation requires that for functional correctness.

3.6 VIRTUAL SLAVE INTERFACE

It is common to want a slave device to have multiple interfaces so that different kinds of traffic can be sent without interference. For instance, a memory controller may want to have an interface for normal traffic, as well as one for traffic with real-time requirements, such as display or audio. Even the normal traffic may be sub-divided into latency sensitive flows such as CPU traffic, and other traffic that is much less sensitive to latency. To avoid head-of-line blocking issues, these various traffic classes need to have separate interfaces to the memory controller or another slave. If they share an interface, then one traffic class can block the others. Multiple physical interfaces can get very expensive because of the number of pins required, as well as the number of physical wires that have to be transported to the slave. Virtual interfaces can reduce both problems.

AXI protocol does not support virtual interfaces. It uses a READY/VALID handshake. When a request is made on the interface, it must stay there until accepted by the other side. If a low priority request is blocked, higher priority requests will get stuck behind them. To support different traffic classes, the slave would have to utilize multiple physical interfaces, with each interface requiring its own bridging logic.

This feature augments an AXI slave port interface to allow virtual channel awareness across it (creating virtual interfaces), without actually changing any of the existing signals, either in number or in meaning.

To create multiple virtual interface on the same physical interface, a new flow control mechanism is added on top of the existing AXI protocol signals. The flow control addition uses a credit-based flow control mechanism.

The following diagram shows the additional interface signals that allow a virtual interface for an AR path.

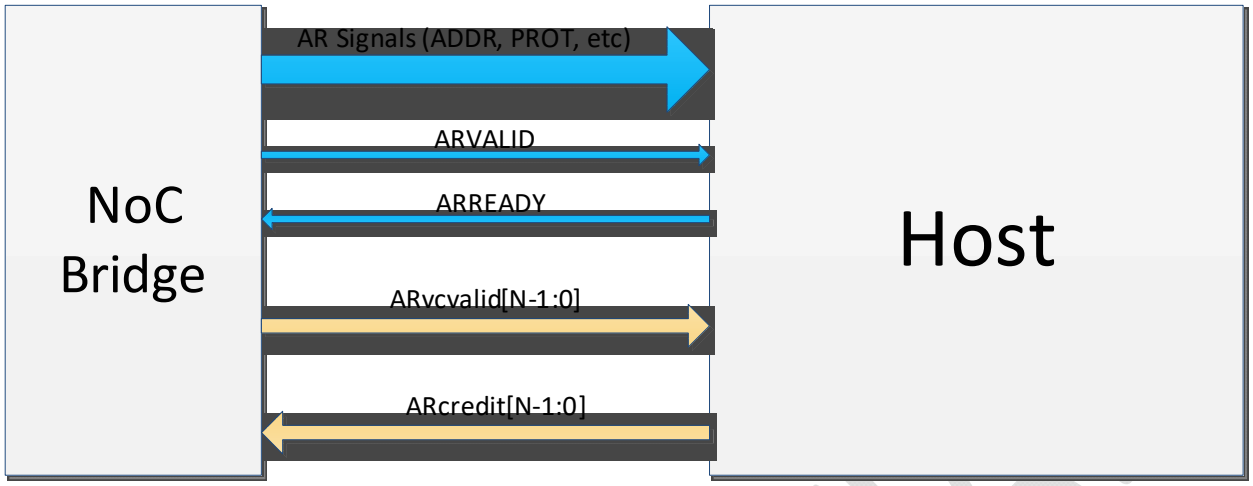


Figure 23: AR interface additions

The top three signal sets, in blue, are the standard AXI interface signals. These don't change. There is an ARVALID and ARREADY for flow control, as well the other AR signals including ARADDR and ARID.

The bottom two signal sets are new and are used as an add-on to the interface to allow virtual interfaces. These signals are ARvcvalid and ARcredit. Each of these signals has a width equal to the number of virtual interfaces desired (N). So, if 3 interfaces are needed, each of these signals will be 3 bits wide.

The credit information is a method by which the Host can send information to the NoC bridge to indicate that it has a dedicated resource available for that virtual channel. By communicating with these credits, the bridge is able to know ahead of time whether the Host will be able to accept a request of that type. If no credit is available, the bridge will not send a request to the Host for that traffic type, since once it does, it can create head-of-line blocking.

When the bridge detects that a credit is available for a traffic type, it can choose to send that traffic type to the Host. It will do so by setting the normal ARVALID signal along with other signals. In the same cycle, it will indicate to the Host that which virtual channel the request is for using the ARvcvalid signal.

The host must still assert ARREADY for the request to complete. If the ARREADY signal is deasserted, the bridge will continue to attempt to send that AR request until it successfully sees ARREADY with ARVALID. Since dedicated storage is expected, and that storage has been communicated via a credit signal, it is possible that the host will never deassert ARREADY, since any request being sent on the interface could drain.

Whenever ARVALID is asserted, one and only one vc_valid signal is asserted Credits can be returned on any cycle after the request has been accepted and is only dependent on the Host freeing up a resource. Multiple credits can be sent on a single cycle (up to one per VC).

Virtual channels on AW,W channels are shown below

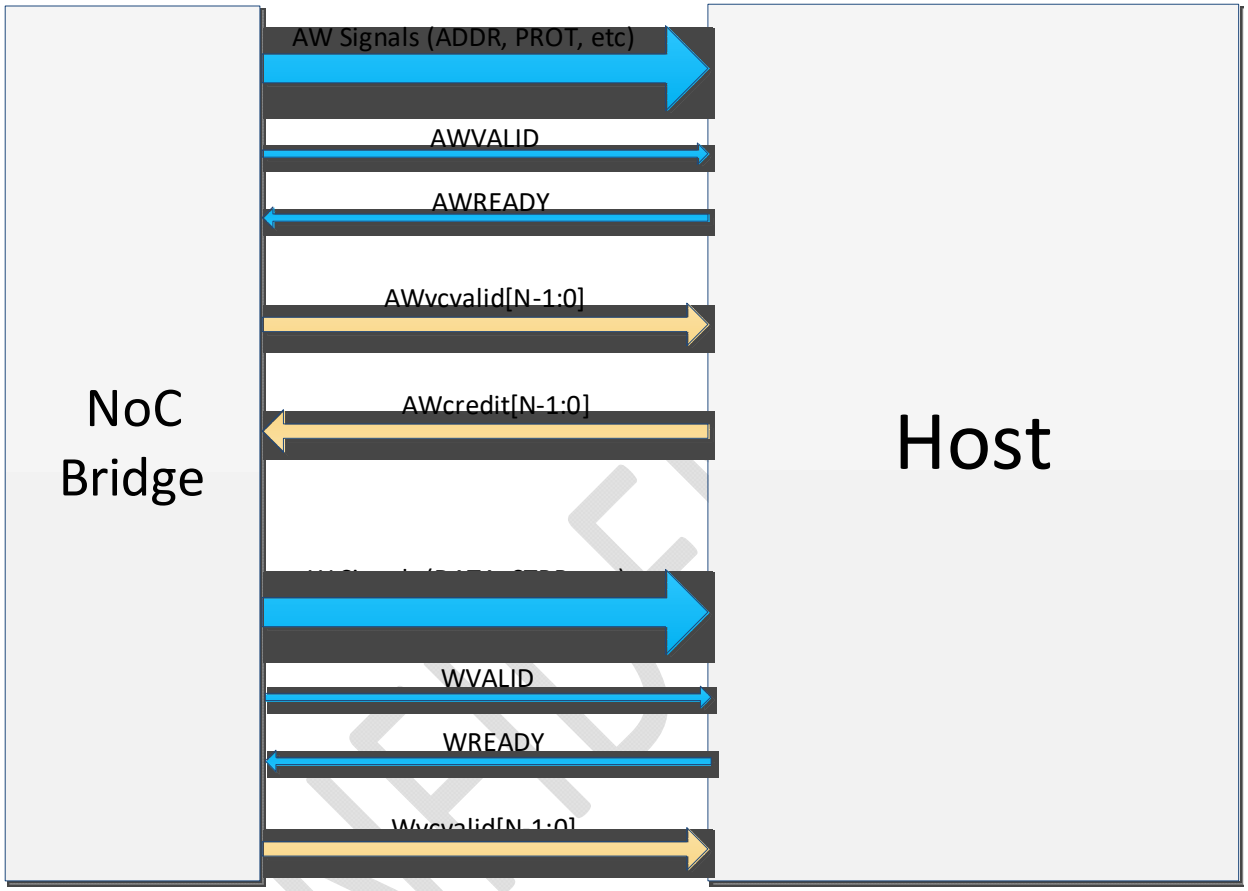


Figure 24 AW-W interface additions

AW and W are flow controlled together, hence a data pre-allocation mechanism is needed on the host for the virtual interfaces. A single credit from the slave would indicate that it can accept an AW command, as well as a burst of W data up to 64B. A credit must be available before either the AW or W packets are sent on the interface.

There are up to 64 virtual channels from the NoC on a slave bridge, these can be mapped to the configured number of virtual interfaces on AW and AR channels. A NoC VC can send to only one interface VC, but an interface VC can receive from multiple NoC VCs.

Transaction tables on the slave bridge for the AR and AW channels can also be partitioned among the virtual host interfaces with dedicated reservation for each virtual interface as well as a common shared pool of table entries.

3.7 INTERRUPT

The NoC produces one or more interrupt signals. The interrupts can be triggered on various error conditions, including ECC error, decode errors, or illegal commands.

3.8 RESTRICTIONS

- AXI error handling has the following limitation
  - For reads responses for narrow transfers, if RRESP for the same response has mixture of different error types per beat, end-to-end checker may flag a false RRESP mismatch. This is a pathological case and not expected to be seen in normal usage.

4 Low Power Support

4.1 NetSpeed Power Management Architecture

The power management architecture is comprised of 3 layers: the PMU – SoC power control circuitry provided by the user, the NetSpeed Power Supervisor (NSPS), and NoC itself.

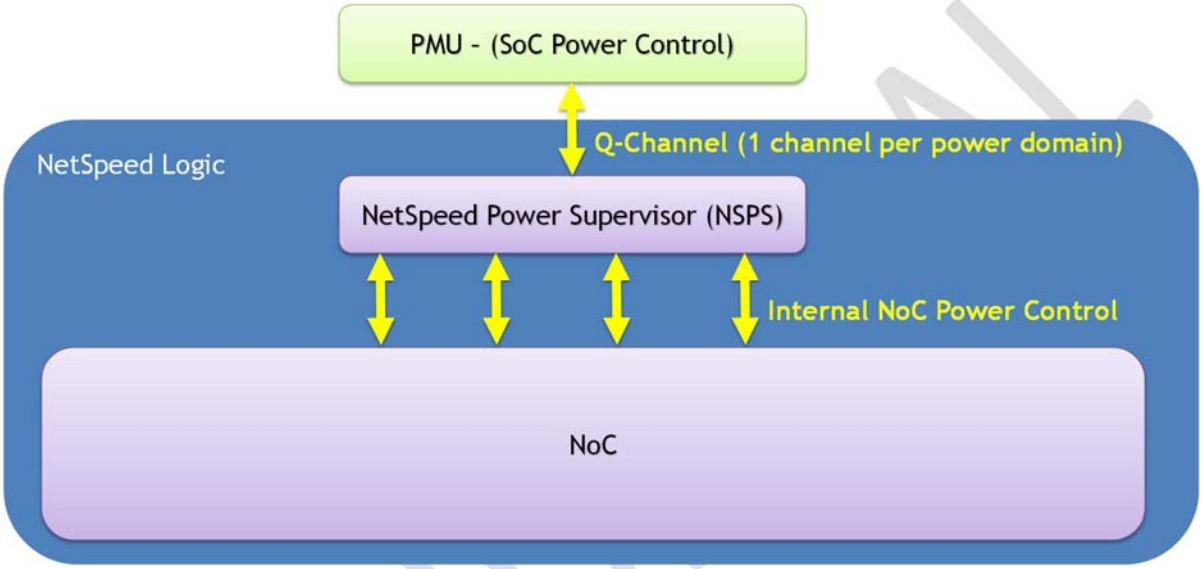


Figure 25 NetSpeed Power Management Architecture

4.1.1 PMU (SoC Power Management Unit)

SoC power management logic, the PMU, is responsible for coordinating power state changes amongst all the elements of the device, including the NoC, but also including the hosts that interact with the NoC and logic beyond. The PMU is provided by the user and generates all power gating controls (isolation, power gate enable, etc.) – NetSpeed logic does not generate these. The PMU is generally expected to exist in an "always on" power domain.

The PMU coordinates power state changes with the NoC via an AMBA Q-channel interface as defined in the ARM Low Power Interface Specification.

4.1.2 NetSpeed Power Supervisor

The NetSpeed Power Supervisor (NSPS) abstracts away NoC microarchitectural details to present a simple well-defined interface to the PMU for communication of power intent. For each power domain in the NoC, it maintains a Power Domain Finite State Machine (PD FSM) that provides high-level sequencing of the operations required for power removal and power restoration. This FSM drives the Q-channel interface (QREQn/QACCEPTn/QDENY/QACTIVE) to the PMU in conjunction with driving signals to elements in the NoC, needed to coordinate

power sequencing activity. It is mapped into a power domain that is always on with respect to the rest of the power domains in the NoC, which could be the same power domain as the PMU, possibly co-located with it.

The NS Power Supervisor also has aggregation logic to combine acknowledgment signals and wake request signals returned from NoC elements. This logic may be distributed in the design to minimize wiring impact.

4.1.3 NoC Element Power Management Logic:

Logic in each of the NoC elements implements required power management functionality, including supporting coordination with the NSPS (e.g., fencing and draining, idle status and sleep ack, etc.). This logic is located within each NoC element (bridges and routers).

4.2 Power Domains and Relationships to Clock and Voltage Domains

Power domains are the logical construct through which power control is communicated between the PMU and the NoC. They nominally describe the boundaries of regions that share common status regarding clock gating (at the power domain level) and power gating. The sections below describe the interactions between power domains, clock domains and voltage domains.

4.2.1 Clock Domains

Clock domains may span power domain boundaries, as long as all the power domains have a common voltage domain. Clock domains may not span voltage domain boundaries. However, where clock gating is intended at any level above an individual NoC element (e.g., routers and bridges), one or more power domains must be defined that in aggregate match the boundary of the gated clock domain. For each clock domain, a distinct clock input pin is provided at the NoC interface for each power domain that it spans. A clock gate and corresponding enable may be inserted in front of these pins to create gated clock domains. The SoC PMU would request a power state change for the affected power domain(s) from the NSPS prior to changing the gated status of this clock domain.

In cases where coarse grained clock gating is intended above the NoC element level but at finer granularity than would be power gated (i.e., a power gating domain spans multiple clock gating domains), power domains must be defined for each gated clock domain, and when power gating the aggregating domain, the SoC PMU must request power state change for all the gated clock domains that are part of the aggregate power gating domain. The converse also applies.

4.2.2 Multiple Voltage Domains

Orion LP supports the definition of multiple voltage domains. Each power domain is assigned to a single voltage domain, with the implication that a power domain may not span multiple voltage domains. However, a voltage domain may be divided into multiple power domains which have independent power state controls. In other words, power domains may not span voltage domains, but a voltage domain may contain multiple power domains. Where signals in the NoC cross voltage domain boundaries, NocStudio will infer appropriate level-shifting structures in the CPF collateral it generates.

Note as stated above, clock domains may not span voltage domains.

4.3 Low Power Signaling Interface Between PMU and NSPS

Orion LP implements the Q-channel low-power signaling protocol according to the ARM AMBA Low Power Interface Specification. One Q-channel interface is provided for each power domain defined for the NoC. Four signals are involved: *QREQn_<PD>* (driven by PMU), *QACCEPTn_<PD>*, *QDENY_<PD>* and *QACTIVE_<PD>* (these latter 3 driven by NSPS). This interface allows the PMU to issue requests to the NSPS to safely remove and restore power to each power domain. This interface also allows the NSPS to issue requests to wake power domains. The PMU is ultimately responsible for waking domains, either in response to NSPS requests or other criteria.

Following is a brief description of how this interface is used to power down and subsequently power up a domain:

4.3.1 Power Down Sequence

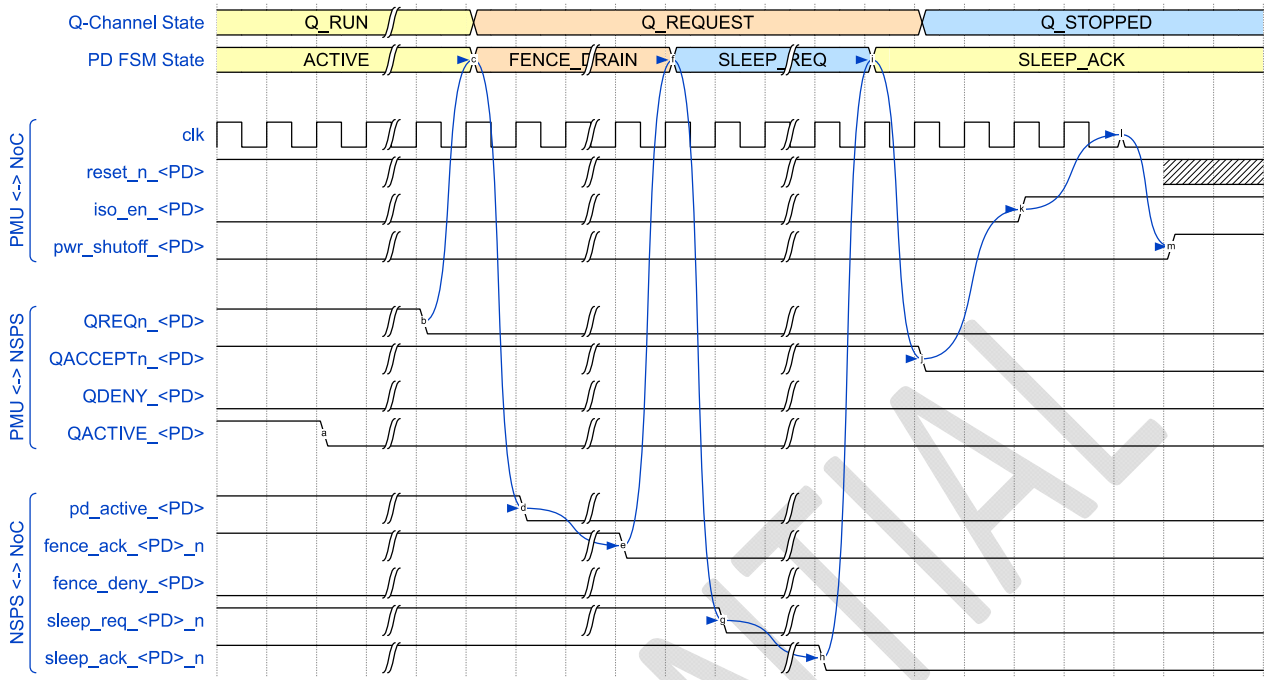


Figure 26 Power Down Waveform Sequence

1. Normal Operation: *QREQn_<PD>*, *QACCEPTn_<PD>* == 1, and *QDENY_<PD>* == 0. *QACTIVE_<PD>* may be in either state.
2. Power Down Request
    a. PMU drives *QREQn_<PD>* low
    b. NSPS decides whether it can accept power down request or not:
        i. Will accept: drives *QACCEPTn_<PD>* low
        ii. Will not accept: drives *QDENY_<PD>* high

Note: PMU must hold *QREQn_<PD>* low until NSPS responds by asserting either *QACCEPTn_<PD>* or *QDENY_<PD>*, and it must return to normal operation by raising *QREQn_<PD>* high and waiting for *QACCEPTn_<PD>* to go high or *QDENY_<PD>* to go low before initiating a new power down request (for this domain).

3. Powered Down: *QREQn_<PD>*, *QACCEPTn_<PD>*, *QDENY_<PD>* and *QACTIVE_<PD>* all == 0.
    a. PMU removes power
        i. Asserts *iso_en_<PD>*
        ii. Disables clocks
        iii. Asserts *pwr_shutoff_<PD>*

4.3.2 Power Up Sequence

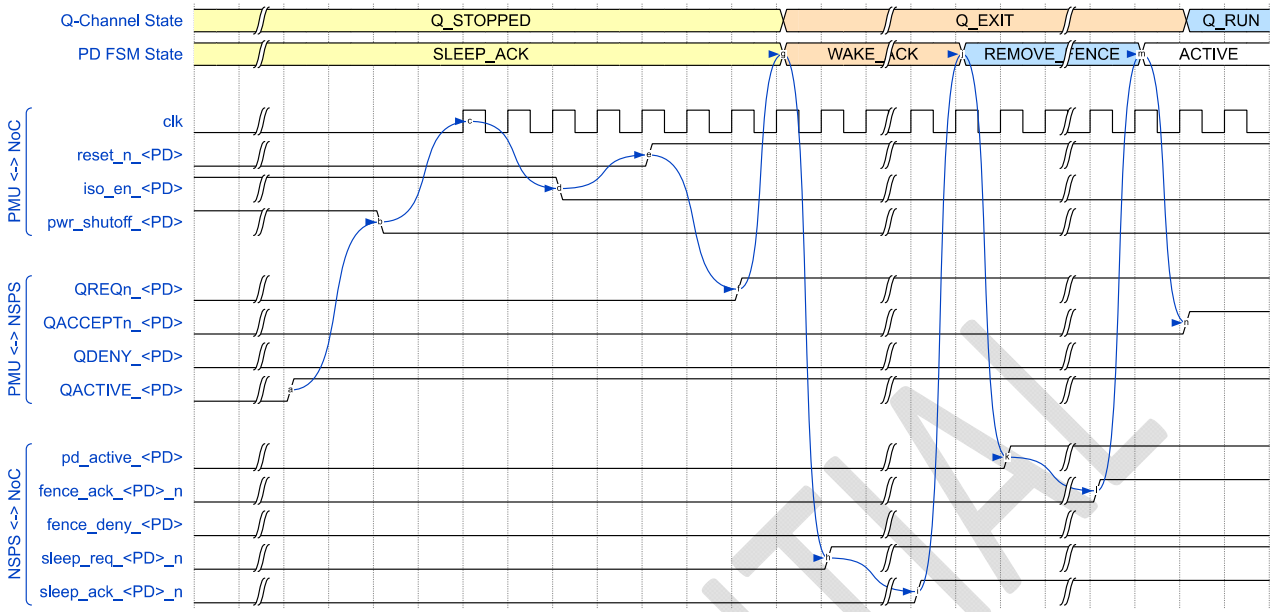


Figure 27 Power Up Sequence Waveforms

1. Wake Request initiated by NSPS: drives *QACTIVE_<PD>* high.
2. Power Up
    a. PMU restores power
        i. De-asserts *pwr_shutoff_<PD>*
        ii. Enables clocks and asserts *reset_n_<PD>*
        iii. De-asserts *iso_en_<PD>*
        iv. De-asserts *reset_n_<PD>*
    b. PMU drives *QREQn_<PD>* high.
    c. NSPS acks by driving *QACCEPTn_<PD>* high when logic is safely restored for normal operation.
3. Normal Operation: *QREQn_<PD>*, *QACCEPTn_<PD>* == 1, and *QDENY_<PD>* == 0. *QACTIVE_<PD>* may be in either state.

4.4 Fencing and Draining

Fencing and draining is a feature of NetSpeed power management that prevents any loss or corruption of transactions due to power state changes. In response to requests from the PMU to shut down one or more power domains (signaled by driving the associated QREQn signals low), the NSPS communicates with all master bridges to ensure 2 things:

1. No new transactions that require the power domain that is going down are allowed to enter the NoC – this is "fencing."

2. Any transactions in progress at the time the QREQn power down request is received are monitored for completion prior to acknowledging the request – this is "draining."

To support this feature, all master bridges constantly monitor the requested power status of all relevant power domains via signals driven by the NSPS. When a bridge observes a request to shut down, it initiates fencing and draining for that power domain. Fencing begins immediately, and an acknowledgement signal to the NSPS is asserted only when all outstanding transactions dependent upon that power domain have completed.

The address look-up tables in the master bridges include information about which power domains are required to be in the active state for a given transaction request to be completed successfully. This information, combined with the power status information, is used to make a dynamic decision whether to fence or forward each newly arriving transaction on the host interface.

4.4.1 Fencing Behavior

When a transaction is fenced, it will be handled in one of two ways depending on how the NoC is configured:

1. DECERR response: completed immediately by the master bridge with a DECERR status.
2. Auto-Wake Request: transaction will be held in local storage in the master bridge while it signals NSPS to request any blocking power domains to be woken by asserting QACTIVE.
    a. Note: the PMU should take action to wake an auto-wake enabled power domain whenever QACTIVE is received for a domain that is powered down. Otherwise the transaction will remain blocked, which in turn blocks all further transactions on that channel (even those that do not have the same power domain dependencies).

The type of response is controlled for each master bridge via the bridge property *axi4m_autowake_enable* in NocStudio. When regbus is enabled, this setting appears in a register that may be dynamically configured.

In addition to the master bridge setting, NocStudio maintains a property for each power domain which indicates whether or not it is wakeable. This is set via an optional argument to the *add_power_domain* command, or it may also be updated via the *set_power_domain_auto_wakeup* command.

Auto-wake requests will be signaled by asserting QACTIVE_<PD> for each power domain that is blocking a transaction only when both the following conditions exist:

1. Bridge property *axi4m_autowake_enable* is set true.

2. All power domains that are currently blocking the transaction are auto-wake enabled.

Correspondingly, if any power domain blocking the transaction is not wakeable, the transaction response is forced to become a DECERR, and QACTIVE_<PD> is not asserted to wake any power domains.

4.4.2 Fencing and QDENY

When a power down request (*QREQn_<PD>* 1->0) is received for an auto-wake capable power domain, if there are any pending transactions in the NoC that depend on that domain, the NSPS will reject the power down request by asserting *QDENY_<PD>* instead of initiating fencing and draining. The PMU must return Q-channel to the Q_RUN state by raising *QREQn_<PD>*, and it must wait for all activity that depends on the target power domain to complete before a power down request will be accepted (though it may retry repeatedly while waiting). This behavior holds regardless of the fencing response type configuration at the master bridges (*axi4m_autowake_enable*).

Note that *QACTIVE_<PD>* is driven high whenever there is pending activity dependent upon a power domain. This may be used by the PMU during the Q_RUN state to determine whether or not a power down request is likely to be accepted.

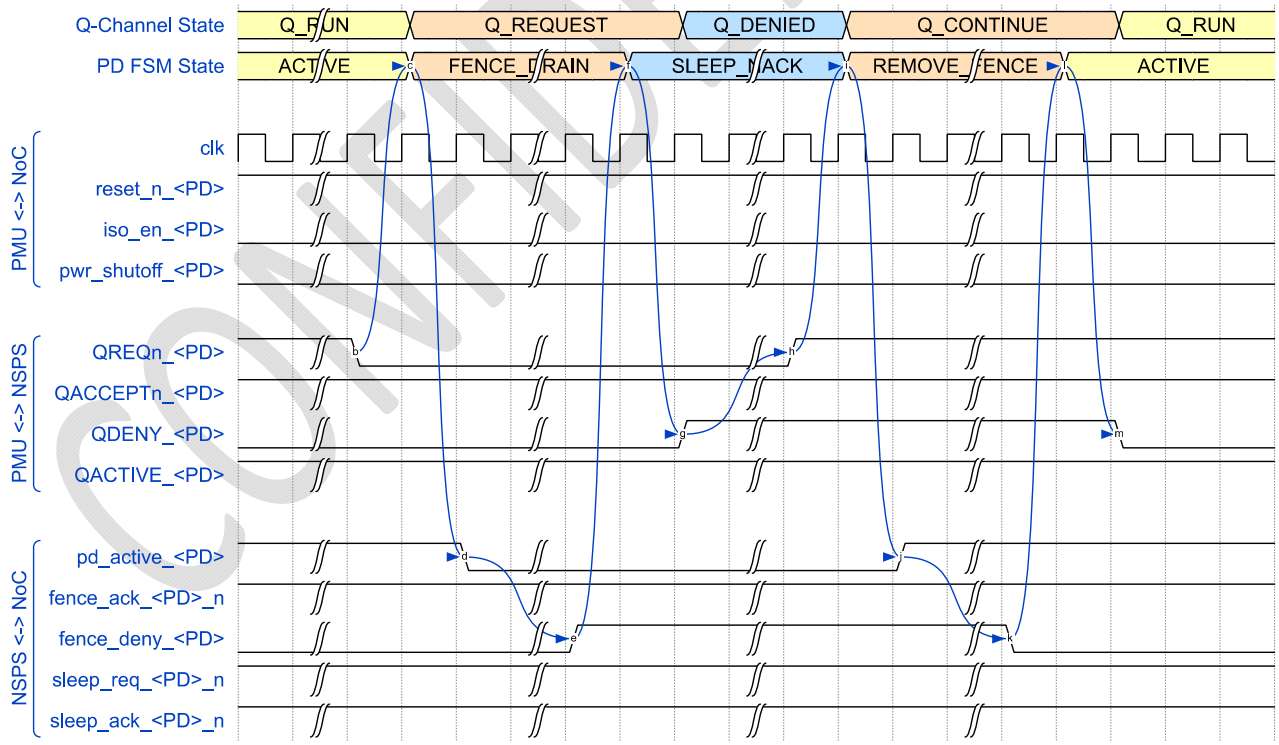


Figure 28 QDENY Waveform Sequence

4.5 Low Power Signals

4.5.1 NetSpeed Power Supervisor Interface to PMU

The interface between the NetSpeed Power Supervisor and the PMU follows the AMBA Low Power Interface Specification, specifically the Q-Channel defined there. It is defined to be asynchronous and capturing logic on both sides must properly synchronize the signals to a local clock.

One set of the signals is implemented per power domain. The interface will not change through NoC design iterations as long as the power domains are not changed.

Signal Name	Source	Destination	Purpose
QREQn_<PD>	PMU	NSPS	Inform NS logic of intent to remove/restore power from/to power domain <PD>. When transitioning 1->0, requests power-down, when transitioning 0->1, requests wake-up.
QACCEPTn_<PD>	NSPS	PMU	Response to *QREQn_<PD>*, acknowledges request for power down or power up.
QDENY_<PD>	NSPS	PMU	Asserted instead of *QACCEPTn_<PD>* to reject a power down request.
QACTIVE_<PD>	NSPS	PMU	When powered down (Q_STOPPED state - *QREQn_<PD>* and *QACCEPTn_<PD>* are both low, NSPS in SLEEP_ACK state), *QACTIVE_<PD>* is asserted to signal a wake-up request. In other states, it serves to indicate the idle status of NoC logic within the power domain.

Table 1 NSPS to PMU Interface

4.5.2 PMU to Power Domain Interface

The PMU (or related logic in the SoC) drives the following signals related to power management, one set of these signals per power domain.

Signal Name	Source	Destination	Purpose

reset_n_<PD>	PMU	NoC elements	Per power domain reset. This is asserted to establish clean state following the first power-on event and subsequent power cycles for the power domain. Point to multipoint, subject to clock/voltage domain crossing treatment.
iso_en_<PD>	PMU	isolation clamps	Defined in ns_soc_ip.cpf, this controls isolation clamps that are inferred where outputs of the power domain connect to inputs that exist in other power domains. Clamping function is enabled when this signal is driven high.
pwr_shutoff_<PD>	PMU	power gates	Defined in ns_soc_ip.cpf, this controls power gates for the power domain. Power is removed when this signal is asserted high.

Table 2 PMU to Power Domain Interface

4.6 BRIDGE LOGIC IN HOST POWER DOMAIN

There are a couple of situations where a portion of the bridge logic must exist in the host's power domain (as specified by the *power_domain_host* property of the bridge).

4.6.1 AHBLM Compound Bridge

For the AHB Lite master bridge, protocol conversion logic translates the host interface protocol to AXI4 which is driven to an AXI4 master bridge. In this case, due to protocol restrictions described further in section 4.7, the conversion logic must live in the host's power domain. When the bridge's *power_domain* property is set to a different power domain than the *power_domain_host* property, the boundary between these domains appears between the protocol converter and the AXI bridge.

4.6.2 Voltage Domain Boundaries between Host and Bridge

Where a voltage domain boundary appears, NocStudio inserts a set of asynchronous clock crossing structures built from specially designed FIFOs. These FIFOs separate the read mux and associated logic into one voltage domain and the storage array and associated write strobe logic into the other voltage domain. Level shifters are properly inferred to allow safe signal crossing between the two voltage domains.

When a voltage boundary is created between host and bridge, one side of the voltage domain crossing structures must live in the host power domain. In the case of compound bridges, the voltage domain crossing structures are inserted between the protocol conversion logic and the AXI4 bridge. In all other cases, the voltage domain crossing structures are inserted between the host and bridge at the host interface to the bridge.

4.7 AHB Lite Master Bridge (AHBLM)

The AHB protocol, due to its pipelined nature, requires the host interface of the AHBLM bridge to hold HREADY high whenever the interface is idle. This means the bridge must always accept any newly initiated transactions, so there is no protocol compliant way for AHBLM to hold off the host interface at a clean transaction boundary. Because of this limitation, the following restriction applies:

The host driving the AHBLM interface must guarantee that no new transactions are initiated any time the host power domain (*power_domain_host)* is not in the Q_RUN state.

4.8 Regbus Master and Tunnel

The regbus master bridge must be configured so that its power domain (*power_domain*) and its host's power domain (*power_domain_host*) must be the same. When regbus tunnel is deployed, the rbm/s and rbm/m instances must be configured such that their *power_domain* and *power_domain_host* properties are all the same.

NocStudio automatically enforces these restrictions by updating all properties that must remain the same to take the new value whenever an update is made to any of the properties.

4.9 Shared Interface Bridge

A shared interface bridge instance inherits its *power_domain* and *power_domain_host* settings from the *power_domain* setting of all the master bridges that feed into it, and that setting must be the same across all these bridges. NocStudio verifies this when mapping is executed, and it will flag an error if the *power_domain* settings are not consistent.

4.10 Always On Power Domains

Always on power domains are those for which the *power_domain_always_on* property is set to "yes." The fundamental expectation is that logic in such power domains is powered before or coincident with other power domains in the NoC (i.e., at initial start-up or boot time), and that it remains powered continuously during operation of the device. They are defined in CPF power intent files so that signal transitions between them and other power domains can be recognized

and properly handled. However, these power domains do not have power gating (*pwr_shutoff_<PD>*) or isolation (*iso_en_<PD>*) logic or signaling inferred in the CPF files as such logic is unnecessary. Q-channel interfaces and the associated NSPS power control logic is not present for always on power domains.

4.10.1 Clock Gated Only Power Domains

Clock gated only power domains are always on power domains that also have the *power_domain_force_q_channel* property set to "yes." These domains are treated like other always on power domains in CPF files, so they do not have power shutoff or isolation circuitry, but a Q-channel interface and associated NSPS power control logic is provided which allows the PMU to switch these domains between active (Q_RUN) and quiet (Q_STOPPED) states. Fencing and draining is implemented for these power domains, allowing clocks to be safely gated when they are in the Q_STOPPED state.

4.11 RESTRICTIONS

- User regbus bridges are not currently fully supported in low power mode
- Multi-voltage: AHBLM master bridge, APB & AHB slave bridges do not implement multi-voltage support.
- Support has not been implemented for:
    - DVFS.
    - UPF power intent format
        - Impacts LP simulation and synthesis with Synopsys tools.

5 Deadlock Avoidance

Applications running on modern day heterogeneous SoCs can generate complex inter-communication messages between the various IP blocks. Such complex, concurrent, multi-hop communication between various cores can result in deadlock situations on the interconnect. Deadlock occurs in a network when messages are unable to make progress to their destination because they are waiting on one another to free up resources, usually buffers and channels. Deadlocks due to blocked buffers can quickly spread over the entire network, paralyzing further operation of the system. Deadlocks can occur both at the network level as well as the protocol level.

NOTE: If NocStudio detects a protocol deadlock that cannot be solved, it notifies the user of the deadlock and the primary cause so architects can fix it by changing their protocol. Finally, NocStudio generates a comprehensive system dependency graph as well so that architects can crosscheck and ensure that there are no deadlocks.

NetSpeed Gemini IP is constructed to be deadlock free. NetSpeed Gemini uses graph theory-based approach and formal techniques to ensure that there are no cycles in the entire message dependency chain of the system. Since there are no cycles, there cannot be a deadlock. To achieve this, NocStudio captures the inter-dependencies between various messages and interfaces in the system using a simple specification system. NocStudio then augments it with additional dependency information interpreted from the protocol definition and inferred from system traffic information. The combined dependency specification is used to ensure full deadlock avoidance – both at the network-level and the protocol-level.

5.1 Quick Primer on Deadlocks

A deadlock is a forward-progress issue. A deadlock occurs when two or more operations cannot complete because they are waiting for completion of one of the other operations. Because each operation is waiting for one of the others, none can make progress. The system is deadlocked.

Deadlocks occur when operations need multiple resources to complete, and the different operations acquire those resources in a contradictory order. For example, if operation A and operation B need both resource X and resource Y to complete, they can deadlock if operation A acquires resource X while operation B acquires resource Y. Each operation has half the resources needed to complete, but cannot acquire the remaining resource until the other operation releases it. However, the other operation won't release the needed resource it until it has acquired both resources.

This deadlock situation can be represented in graphical form showing the resources X and Y and connecting them through dependencies created by the operations. If resource X was acquired by operation A, it cannot be released again until operation A acquires resource Y. There is a dependency between X and Y because of operation A. Similarly, operation B creates a dependency between Y and X. Figure 29 shows the resources and dependencies for this example.

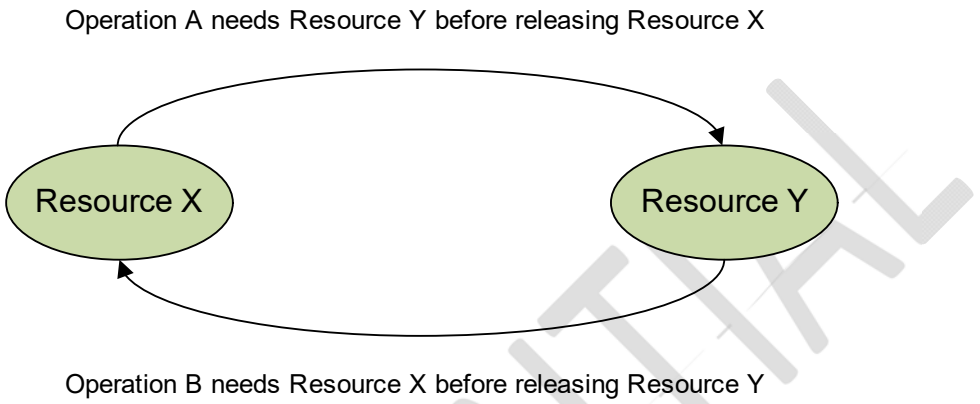


Figure 29. Simple Deadlock Graph with Two Resources

Figure 29 shows a deadlock is possible because of the dependencies' circular nature. Each first resource can require the second resource be acquired before the first can be released. But the second resource can be acquired by another operation that requires the first be released. The circular nature of these dependencies results in a deadlock.

The interconnect resources are the various buffers or FIFO entries. Packets move from one resource to another in the network, requiring the buffer ahead of it to be available before it can move forward and free up the prior buffer. Deadlocks can occur if the dependencies create a loop.

Figure 30 shows a simple system where two hosts (agents) can issue read requests to the other and receive responses. In this system, the interconnect uses a common buffer pool for all traffic moving in the same direction. Reads or read responses moving from Agent 1 to Agent 0 share the same buffers.

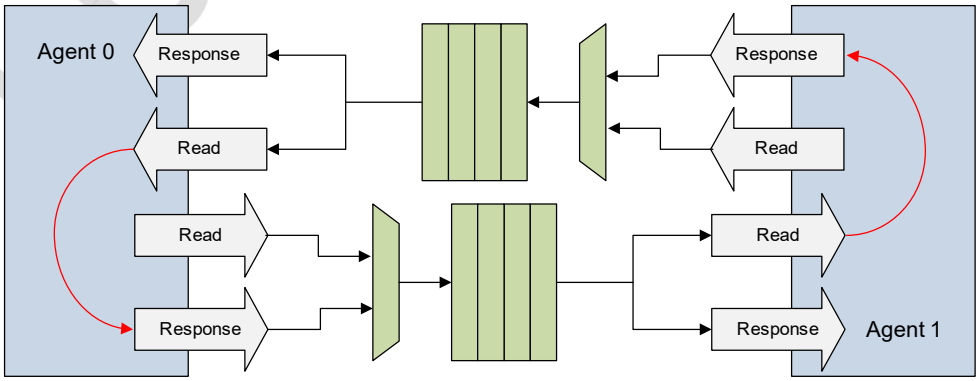


Figure 30. Simple Interconnect with Deadlock

Deadlock can occur where the red arrows indicate a dependency. Here, the dependency is that read requests can complete only when they can issue a read response in the other direction. A deadlock occurs if buffers in both directions are full of read requests and there is no way to send read responses.

5.2 CONSTRUCTING DEADLOCK-FREE INTERCONNECTS

NetSpeed Gemini achieves full deadlock detection and resolution by partitioning complex protocol transactions into the simpler sub-flows from one endpoint to the next. The subflows are heuristically mapped to virtual channels in a way that the number of global virtual networks remains small. Mapping sub-flows independently decouples the virtual channels used for various regions of a single flow and increases the availability of virtual channels by decreasing the scope of a virtual channel mapping. This strategy is effective even if the total number of virtual channels used globally is fairly small. The deadlock in Figure 30 can be avoided by having separate resources for the read and read-response packets. In that case, read responses can drain, allowing reads to make progress. Adding a virtual channel to the network creates an alternative read-response path through the network.

The order in which sub-flows are processed and mapped to virtual networks is of paramount importance too. Machine learning algorithms are used to automatically learn the correct processing order and converge to an optimal solution quickly. In addition to network level deadlocks, protocol level deadlocks may exist. Protocol deadlocks arise when there is cyclic dependency in the way packets are generated and consumed by the endpoints of the NoC. To detect protocol deadlocks, properties of all system components in terms of how they produce and consume network packets and these packets are inter-related to each other are required. To address this problem, NetSpeed Gemini uses a simple yet flexible and powerful formal language to capture the deadlock relevant properties of various system components and uses this information to identify and isolate protocol deadlocks. Subsequently this information is also used to construct the network level deadlock-free NoC.

For NocStudio to avoid system deadlocks, it must have accurate information about the dependencies between the traffic flows. If some dependencies are not described or are described incorrectly, the resulting NoC will be incompatible with the hosts connected to it. This will likely cause a system deadlock. In Figure 30, NocStudio must be alerted to the dependency between the host read traffic and the read-response traffic sent in the other direction. NocStudio analyzes the traffic flows and resource dependencies and creates additional virtual channels as required to

avoid deadlocks. As shown in Figure 31, NocStudio also generates a comprehensive system dependency graph as well so that architects can crosscheck and ensure that there are no deadlocks.

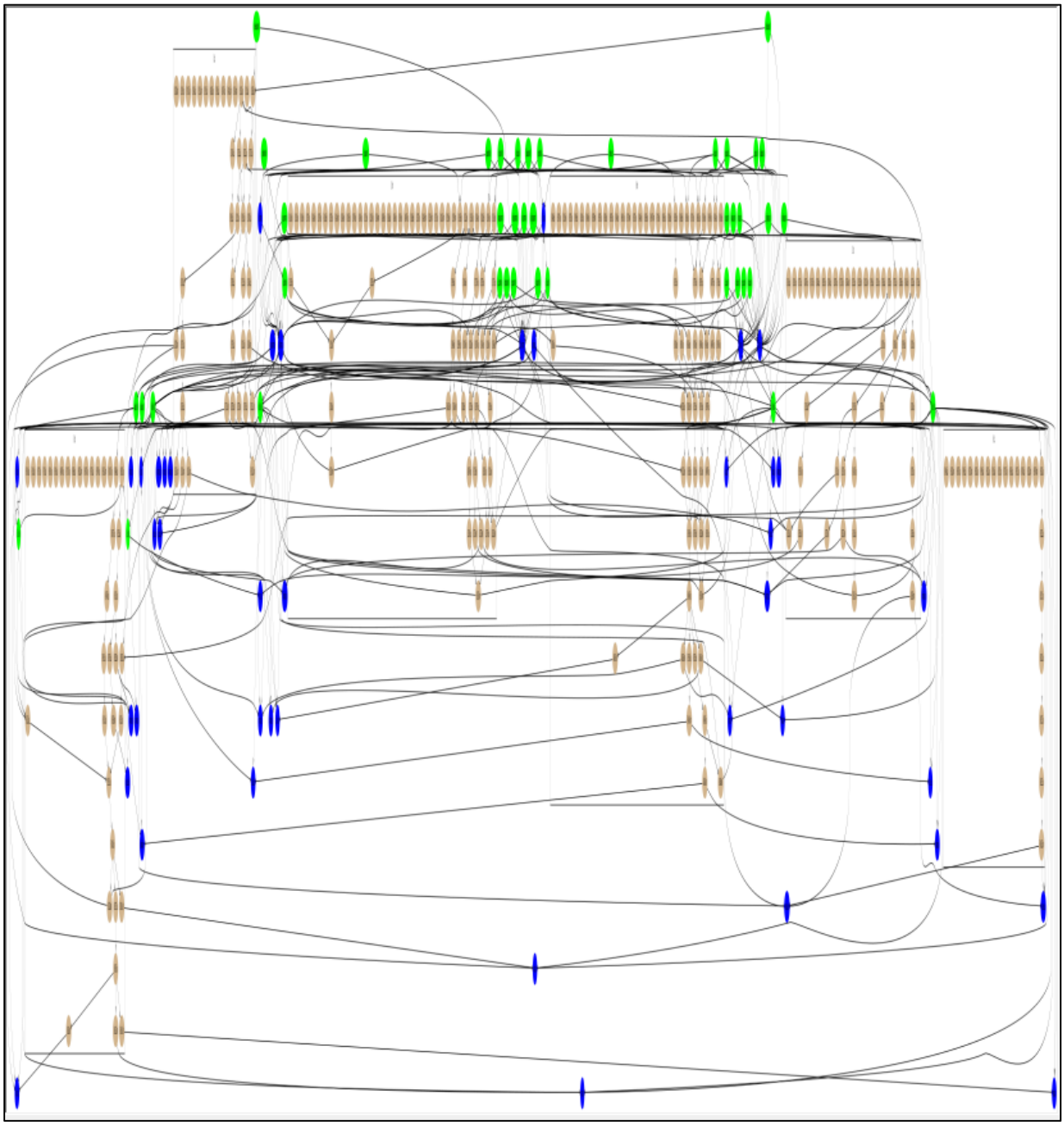


Figure 31 - System Level Dependency Graph

5.3 PROTECTION AGAINST SLAVE DEPENDENCY BEHAVIOR

NetSpeed has support for specific slaves that exhibit a dependency between their read and write responses. Specifically, some slaves send write responses in the middle of a multibeat

read data response, with the expectation that write response cannot be blocked, and pauses sending the read data until then. This introduces dependency between the channels, and NetSpeed has added support to automatically size the write responses FIFOs to avoid this. This is done by adding a configurable property to the slave bridge, which the user can enabled based on their slave behavior.

6 Security

6.1 Connectivity-Based Firewalls

When NocStudio generates a NoC, it creates routing paths between hosts using the traffic descriptions listed in the NocStudio configuration. The NoC is built minimizing connectivity, so if traffic wasn't listed between two hosts, there won't be a connection between those hosts. Even if they reside in a common network, the hosts are not logically connected and traffic cannot be sent between them. This can be used to prevent selected masters from communicating with selected slaves. Any request from a master that targets an unconnected slave causes a decode error in the bridge connected to the master. The request will not be sent to the slave.

Because connectivity is created during NoC construction, this acts as a static and permanent firewall. Each master can be individually configured for connectivity.

This security option can be controlled using the **add_traffic** command in NocStudio.

6.2 Address Range Connectivity

During NoC construction, a slave address range can be divided into multiple parts. Each address range can have different security control. A master can be set up to communicate to a subset of a slave's address range. For example, a device can have two address ranges, one for secure traffic and one for non-secure traffic. Non-secure hosts can be configured to access the non-secure address range and prevented from accessing the secure address range. This capability provides finer control over connectivity. Connectivity can be controlled on a per-address range basis, rather than a per-slave basis.

In NocStudio, master address-range assignment is done using **add_range_to_master**. If this is not used, address-range assignment occurs automatically based on traffic from the master to the slave.

6.3 Propagation of TrustZone Bit

The standard security feature in AXI-based interconnects is filtering based on packet protection bits. AXI networks support a 3-bit protection field as part of the read-request and write-request packets. The ARPROT[2:0] and AWPROT[2:0] fields can be used to propagate security information from the master to the destination. If the slave supports TrustZone filtering, or if TrustZone controller IP is instantiated before the slave, filtering can be handled outside of the NoC.

In coherent systems, the AxPROT[1] bit is used as an additional address bit for coherent requests and is always propagated through CCC or LLC where two other bits, AxPROT[0] and AxPROT[2], are not. This ensures that coherency IP treats non-secure and secure accesses to an address as if they were different addresses, preventing an access to one address from accessing the data in the other. AxPROT[2] and AxPROT[0] are transferred as value zero.

These mechanisms are supported in NetSpeed IP.

6.4 Selective Traffic Filtering

Instead of blocking all requests from a master to a slave, it can be useful to selectively filter traffic from the master to the slave. An example is a CPU that can send both secure and non-secure traffic. The hardware must allow request filtering on a per-address range using the security bit, AxPROT[1].

NetSpeed allows selective request filtering. Four bits of packet information are available for filtering. AxPROT[2:0] bits can be used for filtering and specifying read versus write. For each of these, the NoC can be configured to only allow traffic to be sent when one or more of these bits matches a predefined value. Each filter bit has an enable bit indicating the filter bit is included in the security lookup, and a polarity bit indicating which logical value is required to pass the security check. For example, AxPROT[1] can be enabled and the polarity set to zero so that only secure accesses (AxPROT[1]=1b1) are allowed. All of the security bits can be used concurrently if desired.

Selective traffic filtering can be controlled per-address range and per-master. The per-address range control allows a slave to have multiple address ranges with different security options. A slave that has a secure and non-secure address range can allow hosts to access either range. Only secure traffic can access the secure range. The non-secure range can be configured to permit access by both types of traffic or only non-secure traffic.

Each master connected to the NoC has individually-controlled security options. Different masters can have different security requirements for the same address range. A secure slave can permit all traffic from some secure hosts and only selected traffic from other hosts.

A request filtered through this mechanism causes a decode error. This is because the checks are done as part of the address-map lookup. If the traffic does not match the security requirements, the address lookup fails and functions as if the address range is not mapped to a target. The request stops in the bridge connected to the master port and is not transferred to the target slave.

Security options can be added during NoC construction using NocStudio. The **add_range** and **add_range_to_master** options can be used. The **add_range** option can create a default security

option for all masters sending traffic to that address range. The **add_range_to_master** option provides a per-master security control.

6.5 Programmable Address Map

NetSpeed IP supports programmable address ranges, including per-address range security-control features. This enables multiple additional security options.

The programmable address registers reside in bridges connected to masters. Because each bridge has its own registers, they can be individually controlled.

6.5.1 Disabling Address Ranges

Each programmable address range has a control bit that can disable an address range. This can be used to dynamically isolate a slave address range from a master, preventing requests between them. This functions similarly to fabric-connectivity filtering, but can be enable or disabled as needed.

6.5.2 Changing the Address Map

Because the address map is programmable, address ranges used by security features can be changed. A slave with secure and non-secure regions can be modified to change the region sizes or locations. This can be combined with address range enable/disable to change the number of security ranges.

NOTE: Address range registers are associated with a specific target slave. Although the ranges can change, the range target is unchanged. An unused range register cannot target a different slave.

NOTE: Programmable address registers are specified during construction. Any additional registers must be included at that time. Those registers can be initially disabled, if desired.

6.5.3 Modifying Traffic Filtering

The selective traffic-filtering controls are part of the programmable address registers. Thus, security filtering can be dynamically enabled or disabled. For example, during construction a slave address range can be specified as accessible by all hosts, and software can later modify the security requirements.

6.6 Interface Overrides

Because AxPROT bits are passed through the network to the destination, the system designer might need to override the bits coming from the interface. This can be true if an IP component

cannot be trusted to set the protection bits correctly. For example, if AxPROT is set to indicate TrustZone secure, the host would have access to secure data.

An interface can optionally override the AxPROT[2:0] bits to ensure the system designer has full controllability. This provides control over the AxPROT bits that are transmitted within the network. If the override is set to non-secure, all requests from that host are tagged as non-secure. The override can be specified during NoC construction.

6.7 User Bits

NetSpeed supports the propagation of user bits (AxUSER) within the network. This can be used to pass additional information with read and write packets, such as security options not supported natively in NetSpeed IP.

As an example, user bits with additional security information can be sent through the NoC to the target destination. The target destination can then reject requests based on the user-bit content.

7 Quality of Service Support

Quality of Service (QoS) is a wide-reaching concept that refers to all techniques used within a network to provide control of the arbitration choices in order to satisfy various performance goals. QoS can be used to provide minimum bandwidth guarantees, or maximum latency guarantees. It can be used to divide memory bandwidth according to the important of the agents requesting it. It can be used to prioritize some traffic over others, and to change this prioritization dynamically or through configuration.

QoS ultimately controls the arbitration decisions of the network, coordinating them in order to achieve system level performance goals. This is particularly challenging in a distributed network where many arbitration decisions happen, but a coordinated effect is desired.

QoS is a system level function. The SoC architect must determine the QoS goals of the system and ensure that the system architecture satisfies those goals. This means making globally coordinated choices based on an SoC's requirements.

Individual agents cannot be relied upon to make globally coordinated decisions. For example, the AMBA QOS bits that get transferred with a request are generated by the agent, which doesn't comprehend the system requirements or relative agent priority. In a congested system, each agent may increase the QOS value to indicate they want more bandwidth or lower latency, but this quickly devolves to a situation where every agent demands to be highest priority. The most aggressive agents would demand highest priority first, even if this adversely affects the overall system.

To globally coordinate the system, the architect must specify the relative priority of the traffic flows, the expected allocate of memory bandwidth, and other key system level QoS requirements. The network must then enforce those requirements using the QoS intent and the dynamic conditions within the interconnect to make coordinate decisions.

NetSpeed IP provides a toolbox of global QoS features that provide end-to-end control of the arbitration decisions.

- Traffic classes for traffic isolation
- Strict priority-based allocation
- Weighted bandwidth allocation
- Dynamic priority support
- Rate-limiter functions

7.1 Traffic Classes

One of the primary features of NetSpeed QoS is the concept of traffic classes. A traffic class is a category of traffic flows that have similar qualities in the system. Different traffic classes have different requirements in the system.

Traffic classes provide traffic isolation in a system. Each traffic class is guaranteed to use separate virtual or physical resources. This prevents head-of-line blocking in the system. If one traffic class is stalled, it won't prevent other traffic classes from continuing.

One common use for traffic classes is to place traffic to very slow devices into a separate virtual channel. If a device takes microseconds to return a response, it could potentially stall requests to lower latency devices, such as memory. By isolating the traffic to use different resources, the slow devices won't stall the faster device, boosting overall performance in the system.

Traffic classes may also be used by different master agents, to allow one set of traffic to bypass the other in the network. This is described below in the Strict Priority section.

In NetSpeed IP, a traffic class can be defined through the traffic specification. Each transaction specified by command add_traffic/ add_traffic_b contains a 4-bit Class value (0-15). A master can send some traffic in one traffic class, and other traffic in another. Similarly, a slave may receive traffic from two or more traffic classes.

Since traffic classes use separate physical resources, the use of traffic classes should be carefully controlled to keep total NoC area optimized.

7.1.1 Specifying Traffic Class in NocStudio

In NocStudio, a traffic class can be assigned using the add_traffic or add_traffic_b command. The following shows an example where a traffic flow is created using class 2.

```
add_traffic class 2 rates 1 1 m2/m ar s/s
```

Each traffic command can specify a traffic class. The add_traffic_b command syntax would look like this:

```
add_traffic_b class 3 m0/mprt.ar bw 1 s0/sprt.ar
```

In the add_traffic_b command, this is actually defining a two-hop traffic flow. It sends a read request from the m0 master port to the s0 slave port, and it implies a read response in the other direction. The class is specified before the traffic hops are, which means it applies to both hops.

If no traffic class is specified, class 0 will be used by default.

7.1.2 Different Traffic Class for Request and Response

In many cases, it may make sense to have requests and responses have different traffic classes. Traffic flows to memory, for instance, may want to have different traffic classes for the different requests. This is particularly true when different priorities are used for the traffic classes. You may want one set of requests to be able to bypass another.

The response path may not need separate traffic classes. If a low priority response is stuck leaving the memory controller, it can block high priority responses as well. It will often make sense to have these responses share a single traffic class.

This can also significantly reduce area within a NoC. Traffic classes require separate virtual or physical channels, which requires more storage and possibly more links. While this may be useful for traffic going to memory, the expense of the response path may be unnecessary. And since data paths are often wider than request paths, this can be a significant area impact.

NocStudio allows each hop of a traffic flow to indicate a separate traffic class. In the following example, a request uses class 1 while the response uses class 2.

```
add_traffic_b  m0/mprt.ar bw 1 class 1 s0/sprt.ar/sprt.r class 2 m0/mprt.r
```

Note that the class is specified within the details of each hop. If a class isn't specified, it will use the default the whole command. That default is specified by indicating the class outside of the hop information.

```
add_traffic_b class 3 m0/mprt.ar bw 1 class 1 s0/sprt.ar
```

In the example above, class 3 is specified as the default class for the transaction. Since only the first hop is specified in the transaction, and it has class 1 specified, the request will use class 1 while the response will use class 3. If no default class is specified for the command, the default is class 0.

7.1.3 Mapping AMBA QoS Values

In some rare cases, an architect may generate two classes of traffic between a master and a slave for the same kind of traffic. A master may send reads as either class 1 or class 2, as an example. When this is needed, the decision of which class to use for a traffic flow will be controlled by the AMBA QoS bits. A mapping function can be created to map the AMBA QoS bits to traffic class.

The mapping between AMBA QoS and NoC traffic class can be specified by add_traffic/ add_traffic_b commands.

The first command shown below maps the AMBA QoS value of 0, 1, 2, 3 to NoC Class 4. If no other QoS values are specified for this master and slave interface pair, any other AMBA QoS value also maps to a NoC QoS value of 4. The second command maps a different set of AMBA QoS

values to a NoC Class 3. This gives the master the ability to send traffic through two different classes to the same destination. However, to guarantee ordering at the point of QoS transition, traffic is serialized between the two different class streams. This is not intended for fine-grain selection of traffic class.

```
add_traffic  amba_qos {0 1 2 3} class 4 rates 0.1 0.2 profile 1 m1/m0.ar <-1 -1 1>
        s1/s0.ar
add_traffic  amba_qos {4 5 6 7} class 3 rates 0.1 0.2 profile 1 m1/m0.ar <-1 -1 1>
        s1/s0.ar
```

If no amba_qos mapping is specified, all AMBA QoS values use the default NoC class value for that interface channel.

7.2 STRICT PRIORITY BASED ALLOCATION

Since QoS deals with different kinds of traffic, one of the strongest traffic controls is the prioritization of the traffic. Each traffic class can be assigned a global system priority (with 4 priority levels).

The prioritization happens between traffic classes to avoid the head-of-line blocking problem that occurs when they share resources. The problem is that when a high priority request arrives behind a lower priority request, it would get stalled behind the lower priority request. Even if the network raised the priority of the requests ahead of it, it would still require draining all low priority requests ahead of it, significantly impacting the prioritization goal.

Strict priority is implemented in the NoC using priority-based arbitration at each router and bridge so that higher priority packets always win arbitration. If two virtual channels are arbitrating for the same physical link, the high priority request will win, allowing high priority traffic to move quickly through the network.

One use of strict priority is to allow requests from different masters to the same slave to have different priorities. Requests with real-time requirement can use a high priority traffic class to move around lower priority traffic flows. Systems may include multiple priority levels for traffic going to memory.

Strict priority is a powerful feature but needs to be used cautiously. A higher priority channel can starve a lower priority channel.

7.2.1 How to Define the Priority

There is a one-to-one mapping between NoC Class id and its priority. By default, the 2 LSB of Class id is used as its priority. If a different mapping is desired, users may use the command *class_pri_map* to do this.

7.2.2 Examples

Here are some examples on strict priority-based allocation.

All examples are based on the same design with 2 masters and 1 slave.

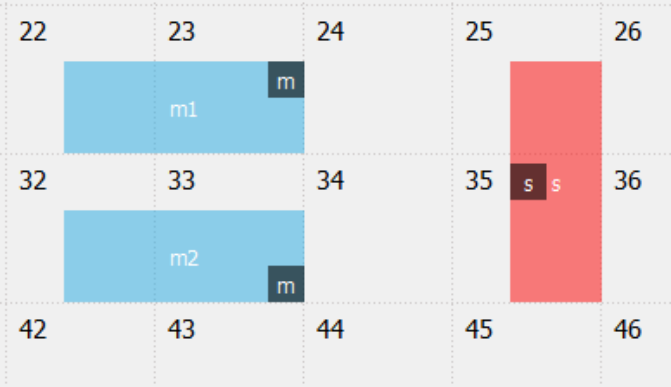


Figure 32: System with 2 masters and one slave

The relationship between priorities and various Class id can be reported through command "class_pri_map".

```
$ class_pri_map
Class  0, priority (0,-)
Class  1, priority (1,-)
Class  2, priority (2,-)
Class  3, priority (3,-)
Class  4, priority (0,-)
Class  5, priority (1,-)
Class  6, priority (2,-)
Class  7, priority (3,-)
Class  8, priority (0,-)
Class  9, priority (1,-)
Class  10, priority (2,-)
Class  11, priority (3,-)
Class  12, priority (0,-)
Class  13, priority (1,-)
Class  14, priority (2,-)
Class  15, priority (3,-)
```

Figure 33: Class/Priority mapping

Example #1

Let us define traffics from both masters to have the same Class (thus, same priority). This can be done as follows:

```
add_traffic class 0 rates 1 1 m2/m ar s/s
add_traffic class 0 rates 1 1 m1/m ar s/s
```

During arbitration, they will each take turns to pass. This can be validated through PerfSim (Performance-Simulator) within NocStudio that loading (Load%) of m1/m.ar.out and m2/m.ar.out are both 50%.

Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%	Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%
m1/m.ack.out	0	0	1000	-	-	-	-	m2/m.r.in	5000	64	1000	50.00%	4	100.00%	50.00%
m1/m.ar.out	5000	0	1000	50.00%	-	100.00%	50.00%	m2/m.wu.out	0	64	1000	-	-	-	-
m1/m.aww.out	0	64	1000	-	-	-	-	s/s.ar.in	10000	0	1000	100.00%	-	200.00%	50.00%
m1/m.b.in	0	0	1000	-	-	-	-	s/s.aww.in	0	64	1000	-	-	-	-
m1/m.r.in	5000	64	1000	50.00%	4	100.00%	50.00%	s/s.b.out	0	0	1000	-	-	-	-
m1/m.wu.out	0	64	1000	-	-	-	-	s/s.r.out	10000	64	1000	100.00%	8	200.00%	50.00%
m2/m.ack.out	0	0	1000	-	-	-	-								
m2/m.ar.out	5000	0	1000	50.00%	-	100.00%	50.00%								
m2/m.aww.out	0	64	1000	-	-	-	-								
m2/m.b.in	0	0	1000	-	-	-	-								

Figure 34: Snapshot of Performance Simulation with same class, same priority

Example #2

Let us define traffics from m2 to have a higher priority than that of m1 by assigning different Class to the traffics. With different Class, traffics will be traveled on different virtual channels. This can be done as follows:

```
add_traffic class 1 rates 1 1 m2/m ar s/s
add_traffic class 0 rates 1 1 m1/m ar s/s
```

During arbitration, traffics from m2 will have higher priority than traffics from m1. This can be validated through PerfSim within NocStudio that loading (Load%) of m2/m.ar.out is 100%; whereas, loading of m1/m.ar.out is 0.

Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%	Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%
m1/m.ack.out	0	0	1000	-	-	-	-	m2/m.r.in	10000	64	1000	100.00%	8	100.00%	100.00%
m1/m.ar.out	0	0	1000	-	-	-	-	m2/m.wu.out	0	64	1000	-	-	-	-
m1/m.aww.out	0	64	1000	-	-	-	-	s/s.ar.in	10000	0	1000	100.00%	-	200.00%	50.00%
m1/m.b.in	0	0	1000	-	-	-	-	s/s.aww.in	0	64	1000	-	-	-	-
m1/m.r.in	0	64	1000	-	-	-	-	s/s.b.out	0	0	1000	-	-	-	-
m1/m.wu.out	0	64	1000	-	-	-	-	s/s.r.out	10000	64	1000	100.00%	8	200.00%	50.00%
m2/m.ack.out	0	0	1000	-	-	-	-								
m2/m.ar.out	10000	0	1000	100.00%	-	100.00%	100.00%								
m2/m.aww.out	0	64	1000	-	-	-	-								
m2/m.b.in	0	0	1000	-	-	-	-								

Figure 35:Snapshot of performance simulation with 2 classes, different priority

Example #3

Let us define traffics from both masters to have different Class, but with same priority. With different Class, traffics will be traveled on different virtual channels. This can be done as follows:

```
add_traffic class 0 rates 1 1 m2/m ar s/s
add_traffic class 4 rates 1 1 m1/m ar s/s
```

During arbitration, they will each take turns to pass. This can be validated through PerfSim (Performance-Simulator) within NocStudio that loading (Load%) of m1/m.ar.out and m2/m.ar.out are both 50%.

Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%	Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%
m1/m.ack.out	0	0	1000	-	-	-	-	m2/m.r.in	5000	64	1000	50.00%	4	100.00%	50.00%
m1/m.ar.out	5000	0	1000	50.00%	-	100.00%	50.00%	m2/m.wu.out	0	64	1000	-	-	-	-
m1/m.aww.out	0	64	1000	-	-	-	-	s/s.ar.in	10000	0	1000	100.00%	-	200.00%	50.00%
m1/m.b.in	0	0	1000	-	-	-	-	s/s.aww.in	0	64	1000	-	-	-	-
m1/m.r.in	5000	64	1000	50.00%	4	100.00%	50.00%	s/s.b.out	0	0	1000	-	-	-	-
m1/m.wu.out	0	64	1000	-	-	-	-	s/s.r.out	10000	64	1000	100.00%	8	200.00%	50.00%
m2/m.ack.out	0	0	1000	-	-	-	-								
m2/m.ar.out	5000	0	1000	50.00%	-	100.00%	50.00%								
m2/m.aww.out	0	64	1000	-	-	-	-								
m2/m.b.in	0	0	1000	-	-	-	-								

Figure 36:Snapshot of performance simulation with 2 classes, same priority

7.3 WEIGHTED BANDWIDTH ALLOCATION

When arbitration occurs between packets of the same priority level, a weighted bandwidth allocation is available. In weighted allocation policy, the resource bandwidth is divided among all contending flows based on a pre-specified set of weights.

The NetSpeed traffic weighting mechanism has two important qualities. It is work-conserving, and it is ratio-conserving. This combination is extremely challenging in a distributed network.

The work-conserving property indicates that whenever there is available bandwidth to be utilized, the network is able to utilize it. Some QoS mechanisms divide bandwidth statically, which mean if an agent doesn't consume its share of the bandwidth, the bandwidth is wasted. The NetSpeed solution allows other agents to utilize any bandwidth unused by an agent.

Work-conserving mechanisms are often implemented as a weighted round-robin arbitration. However, these are not ratio-conserving. A ratio-conserving mechanism guarantees that when some agents do not utilize all of their bandwidth, that bandwidth is distributed to the other agents based on the initially defined ratios. This requires that per-router arbitration decisions can modify the weights dynamically based on which agents are making requests.

This provides a total bandwidth allocation control that has programmable ratios, and can gracefully handle changes in dynamic conditions to enforce those defined ratios even when other agents are idle

NetSpeed IP uses dynamic weight adjustment algorithms that are fully distributed and provides full end-to-end weighted fairness. The algorithm is lightweight and comes at almost no additional logic area or timing complexity and provides end-to-end fairness under almost all scenarios.

7.3.1 How to Define the Weight

Weights of different traffic classes can be defined through bridge properties "qos_*_weight_value".

7.3.2 Examples

Here are some examples on weighted bandwidth allocation.

All examples are based on the same design with 3 masters and 1 slave.

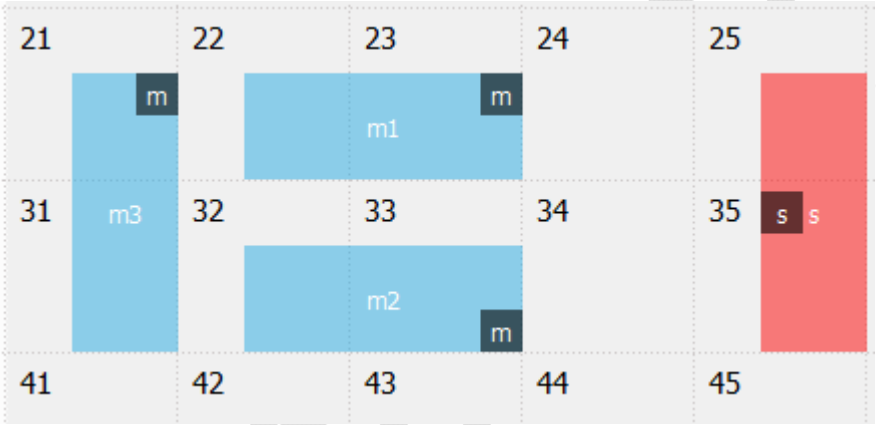


Figure 37: 3 master, 1 slave system

All traffics from m1, m2, and m3 to slave s are of QoS 0.

```
add_traffic qos 0 rates 1 1 m1/m ar s/s
add_traffic qos 0 rates 1 1 m2/m ar s/s
add_traffic qos 0 rates 1 1 m3/m ar s/s
```

The weight for QoS 0 at bridge m1/m is defined as 10, m2/m as 20, and m3/m as 30.

```
bridge_prop m1/m qos_0_weight_value 10
bridge_prop m2/m qos_0_weight_value 20
bridge_prop m3/m qos_0_weight_value 30
```

Example #1

In a case where traffics from all 3 masters are contending for resources, the ratio of resulting bandwidth for each master bridge follows that of the weight:

BW at m1/m: BW at m2/m: BW at m3/m= 10: 20: 30= 1: 2: 3

This can be validated through PerfSim within NocStudio that the ratio of the loading (Load%) of m1/m.ar.out: m2/m.ar.out: m3/m.ar.out = 16.66%: 33.32%: 50.02%= 1: 2: 3.

Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%	Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%
m1/m.ack.out	0	0	1000	-	-	-	-	m2/m.wu.out	0	64	1000	-	-	-	-
m1/m.ar.out	1666	0	1000	16.66%	-	100.00%	16.66%	m3/m.ack.out	0	0	1000	-	-	-	-
m1/m.aww.out	0	64	1000	-	-	-	-	m3/m.ar.out	5002	0	1000	50.02%	-	100.00%	50.02%
m1/m.b.in	0	0	1000	-	-	-	-	m3/m.aww.out	0	64	1000	-	-	-	-
m1/m.r.in	1664	64	1000	16.64%	1.3312	100.00%	16.64%	m3/m.b.in	0	0	1000	-	-	-	-
m1/m.wu.out	0	64	1000	-	-	-	-	m3/m.r.in	5004	64	1000	50.04%	4.0032	100.00%	50.04%
m2/m.ack.out	0	0	1000	-	-	-	-	m3/m.wu.out	0	64	1000	-	-	-	-
m2/m.ar.out	3332	0	1000	33.32%	-	100.00%	33.32%	s/s.ar.in	10000	0	1000	100.00%	-	300.00%	33.33%
m2/m.aww.out	0	64	1000	-	-	-	-	s/s.aww.in	0	64	1000	-	-	-	-
m2/m.b.in	0	0	1000	-	-	-	-	s/s.b.out	0	0	1000	-	-	-	-
m2/m.r.in	3333	64	1000	33.33%	2.6664	100.00%	33.33%	s/s.r.out	10000	64	1000	100.00%	8	300.00%	33.33%

This ratio can also be observed from the graph below.

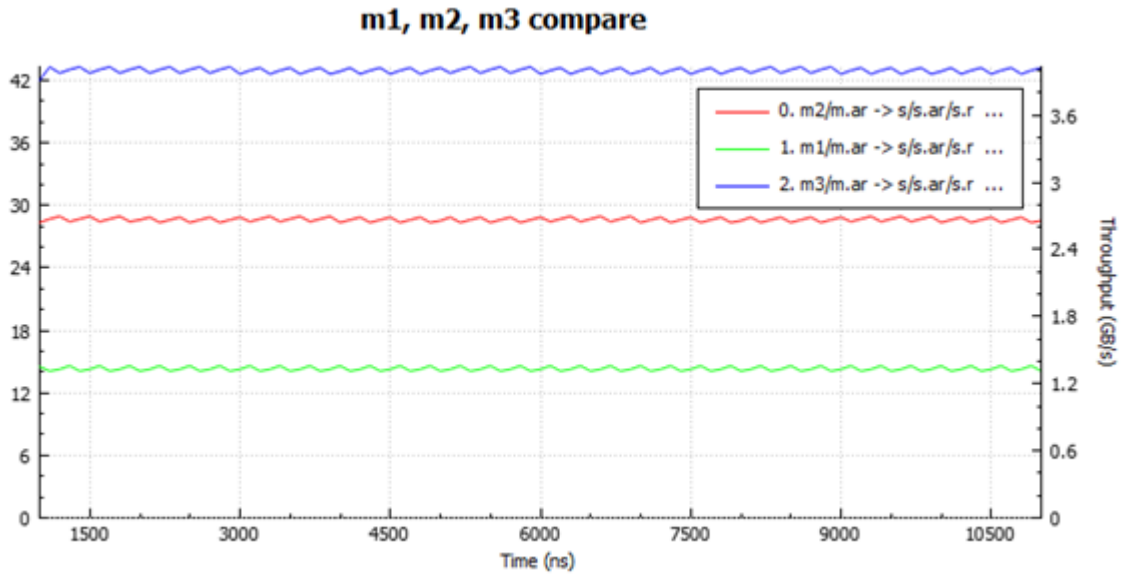


Figure 38: Performance results with ratios 3:2:1

Example #2

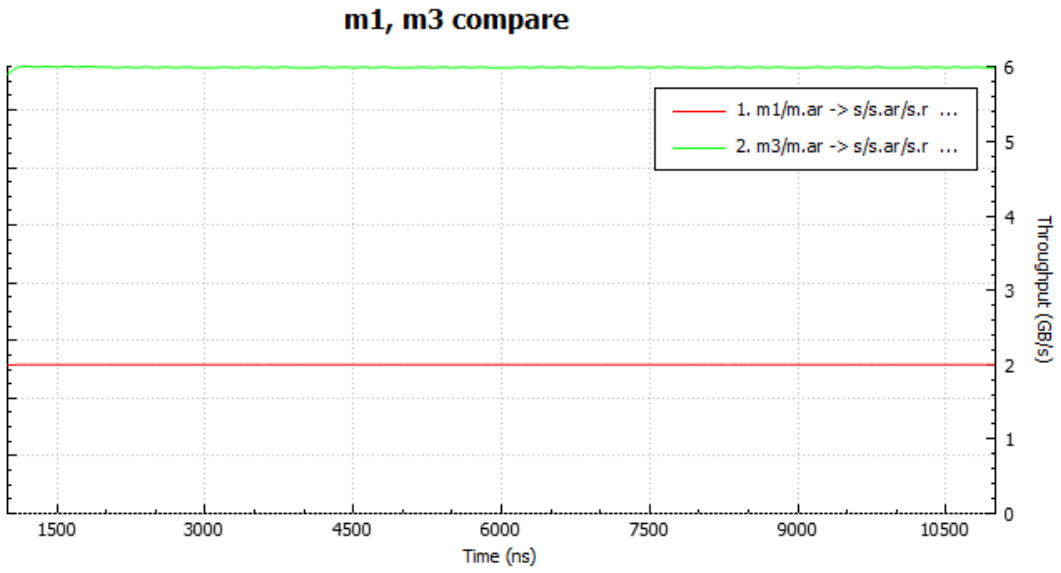
In a case where only 2 traffics are contending for resources, the ratio of resulting bandwidth for each of the 2 master bridges follows their defined weights:

BW at m1/m: BW at m3/m= 10: 30= 1: 3.

This can be validated through PerfSim within NocStudio that the ratio of the loading (Load%) of m1/m.ar.out: m3/m.ar.out= 25%: 75%= 1: 3.

Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%	Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%
m1/m.ack.out	0	0	1000	-	-	-	-	m2/m.wu.out	0	64	1000	-	-	-	-
m1/m.ar.out	2500	0	1000	25.00%	-	100.00%	25.00%	m3/m.ack.out	0	0	1000	-	-	-	-
m1/m.aww.out	0	64	1000	-	-	-	-	m3/m.ar.out	7500	0	1000	75.00%	-	100.00%	75.00%
m1/m.b.in	0	0	1000	-	-	-	-	m3/m.aww.out	0	64	1000	-	-	-	-
m1/m.r.in	2500	64	1000	25.00%	2	100.00%	25.00%	m3/m.b.in	0	0	1000	-	-	-	-
m1/m.wu.out	0	64	1000	-	-	-	-	m3/m.r.in	7500	64	1000	75.00%	6	100.00%	75.00%
m2/m.ack.out	0	0	1000	-	-	-	-	m3/m.wu.out	0	64	1000	-	-	-	-
m2/m.ar.out	0	0	1000	-	-	-	-	s/s.ar.in	10000	0	1000	100.00%	-	200.00%	50.00%
m2/m.aww.out	0	64	1000	-	-	-	-	s/s.aww.in	0	64	1000	-	-	-	-
m2/m.b.in	0	0	1000	-	-	-	-	s/s.b.out	0	0	1000	-	-	-	-
m2/m.r.in	0	64	1000	-	-	-	-	s/s.r.out	10000	64	1000	100.00%	8	200.00%	50.00%

This ratio can also be observed from the graph below.



7.3.3 Usage – Weighted bandwidth

Please note the actual bandwidth allocation depends on many factors like burst length, data width, AXI ID, max outstanding and transaction injection.

- Arbitration occurs only when multiple transactions contend for the resource at the same arbitration point (ex: router). If transactions arrive at the different time, or if they go to different outputs of the same router, there is no arbitration.
- When there is not enough traffic to cause enough arbitration, the bandwidth allocation result can be different from QoS weight settings, but requestors get requested bandwidth in that case.
- Arbitration is applied at a packet granularity and not at flit granularity.
- If same QoS weight is applied to 2 bridges and one of them has 2 times longer burst length, arbitration may end up with the same number of packets for those 2 bridges, but one has 2 times bigger bandwidth.

- When a bridge cannot issue a new command because of max outstanding or serialization, it is possible that the bridge ends up with smaller bandwidth compared to QoS weight settings.

7.4 RATE LIMITING HOSTS

While the above QoS mechanisms are sophisticated and effective, NetSpeed also provides a simple rate-limiter function. This can be used in conjunction with the other mechanisms to provide another axis of control. For instance, if a traffic class has a higher priority than another, it can cause starvation. A rate-limited can prevent that agent from consuming all of the network bandwidth by enforcing a maximum bandwidth limit.

NetSpeed NoC supports programmable rate limiters at all transmitting NoC interfaces. The rate limiters limit the rate at which traffic may be injected into the NoC at various interfaces to a programmed rate. This simple congestion control is effective in any system but is not work-conserving. This means that if additional bandwidth is available, the agent will be unable to use it. This can leave bandwidth unutilized within the system.

NOTE: Users can judiciously enable and choose the rate limit values at various interfaces based on the SoC traffic and QoS requirements and can further reprogram them dynamically in presence of the changing requirements.

7.4.1 Why and When Are Rate-Limiters Needed

Rate-limiters can be used in a variety of situations to either replace the other QoS mechanism, or to supplement them.

Example #1

When the shared resource's (e.g. memory's) bandwidth can be statically partitioned between all contenders, then each contender's interface may be rate limited to its fair share thereby providing fair bandwidth sharing without the possibility of congesting the NoC.

Example #2

In another scenario, rate limiters may be placed on a certain subset of agent that may have low-priority, highly bursty traffic to ensure that the low priority traffic burst does not temporarily congest the NoC enough to affect the high priority traffic. In such cases the high priority contenders may not be rate limited or their rates can be programmed at a high value.

NOTE: Rate limiters are programmable on every interface (i.e. they can be enabled/disabled or the rates can be modified with a register write access), which further increases their effectiveness. Furthermore, they are

recommended when work-conserving weighted QoS and end-to-end strict-priority QoS may be unnecessary and expensive in terms of logic area.

7.4.2 How to Define Rate Limiters Using NocStudio

In NocStudio, there are three interface properties to set the rate limits:

- **peak_rate_limit**: specifies the maximum rate of beats at a bridge interface
- **avg_rate_design_limit**: specifies the average rate of beats at a bridge interface
- **rate_limit_bucket_size**: specifies the maximum bucket size of an interface rate limiter

Rate values specify the data beat rate for interfaces with data such as streaming, amba r, amba aww interfaces, and message rate for single beat interfaces such as amba ar, and amba b. Rates can be between 0 and 1 (inclusive) indicating the fraction of cycles that the interface is active. Rate registers in hardware are 12-bit therefore the rate granularity is $1/(2^{12})$.

Peak rate limits are used in two ways within NocStudio. Along with the avg_rate_design_limit, the peak_rate_limit is used during NoC construction to determine bandwidth requirements at peak or average loads. Peak rate limit is also used in NocStudio performance simulation – during simulation both rx and tx interfaces are rate limited to this value. Peak rate limits less than 1 are also programmed as the default rate limit value of the rate limit registers in hardware.

NOTE: NocStudio performance simulator supports rate limiting of both rx and tx interfaces for performance evaluation purposes while the NoC hardware only supports rate limiters at the tx interfaces.

7.4.3 Implementation of Rate Limiters

The rate-limiter is a flow-control mechanism that prevents a packet from being sent into the network unless enough time has passed since the last packet. It is implemented by selecting a transmission rate and adding a token at the determined rate. A packet can only transmit if a token is available, and so can only transmit at the rate that the tokens are added.

A token bucket is implemented that accumulates these tokens over time. By allowing an interface to accumulate tokens over a period of time, it allows rates to be limited over a larger window, while still allowing a small amount of bursty traffic.

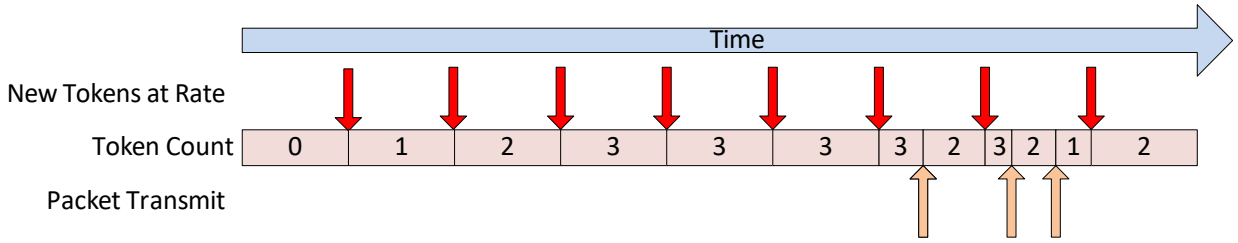


Figure 39 - Token Count increases at specified rate. Packet transmit decrements count.

In the diagram above, the token count is increasing over time at a specified rate. The token count will saturate when it hits its maximum, which in this example is 3. When a packet is sent on this interface, the toke count is decremented. This ensure that the packet transmission rate does not exceed the rate limit except within a small window defined by the token bucket size.

7.4.4 Specifying A Rate

The rate-limiter requires a rate to be specified. In NocStudio, this value is set with the property peak_rate_limit and can be set to a value from 0 up to and including 1. A rate limiter with a rate of 1 will have no effect.

In hardware, the rate is specified as a 12-bit number. The value of the number follows this equation, where N is the programmed value.

rate = $N/(2^{12})$

The hardware implements the rate calculation as a 12-bit adder where the overflow bit is used as the token arrival bit. This defines the granularity for the rate limiting function. For example, to specify a rate of 1 token every 5 cycles (or 20%), N should be specified as 819(decimal) or 0x333 (hex). When added together 5 times, the value will nearly reach approximately 4096 (=2^{12}), so one packet can be sent every 5 cycles.

7.4.5 Token Bucket Sizing

The token bucket size allows for an agent to accumulate tokens over a window of time. This allows the agent to issue a burst of requests over a smaller window, while still being limited in

the long run. The larger the bucket, the larger the short-term burst can be.

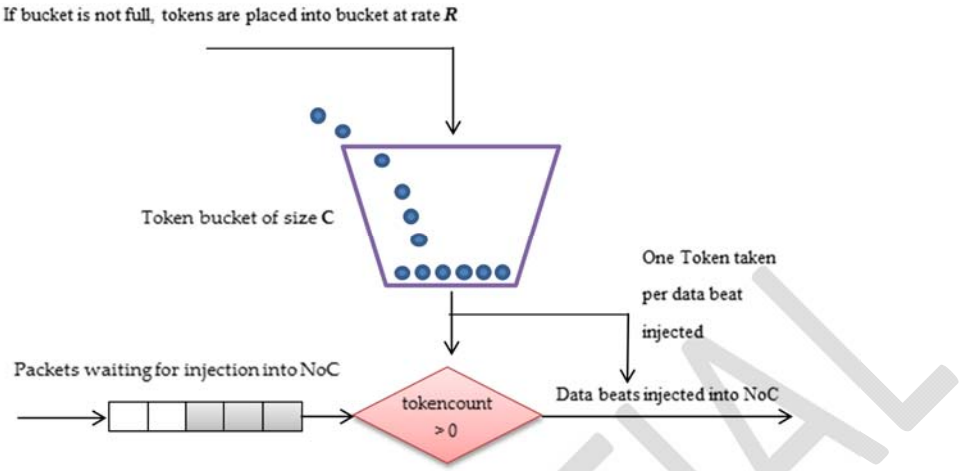


Figure 40 - Token Bucket

As shown above, tokens are added at the designated rate. The size of the token bucket is limited, so it will stop accumulating tokens while it is full. If the interface wants to transmit, the token count must be greater than zero. If it is, the packet can be sent and a token can be removed from the bucket.

NOTE: The size of the bucket can be reduced or increased to provide additional control, depending on the design requirements. More or less burstiness is possible. The token bucket size can be programmed to be of any size between 1 and 15. In NocStudio, this value is programmed using the interface property rate_limit_bucket_size.

7.4.6 Token Usage

For command transfers, a single token is used to transmit the command. For data transfers, each data beat utilizes a token.

7.4.7 Rate Limit Register

The rate limit is applied only when the enable bit of the rate limiter configuration register is set to 1. The rate limiter configuration register (per tx interface) has the following fields.

Bits	Function
11:0	Rate Limit Value, for traffic issue to the NoC from the host interface
15:12	Reserved
19:16	Bucket Size. This indicates the maximum number of token that may be accumulated at an interface when rate limiters are enabled

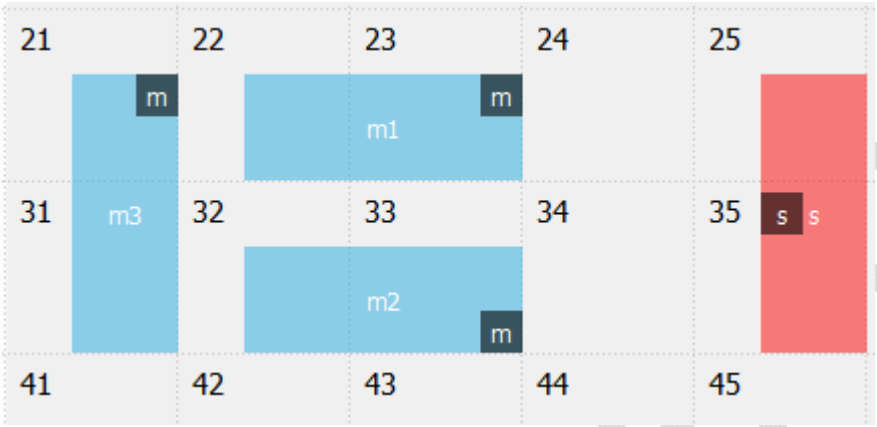
20	Rate Limit logic enable. Rate limiter logic is used for arbitration only when the enable bit of the rate limiter configuration register is set to 1

Table 3 - Rate limiter Configuration Register

7.4.8 Examples

Here are some examples on rate limiting hosts.

All examples are based on the same design with 3 masters and 1 slave.



All traffics from m1, m2, and m3 to slave s are of QoS 0.

In addition to defining traffics from all masters to slave, let us also define the rate_limit for m3. These are done as follows:

```
add_traffic qos 0 rates 1 1 m1/m aww s/s
add_traffic qos 0 rates 1 1 m2/m aww s/s
add_traffic qos 0 rates 1 1 m3/m aww s/s

ifce_prop m3/m.aww.out peak_rate_limit 0.02
ifce_prop m3/m.aww.out avg_rate_design_limit 0.01
```

After mapping, performance simulation can be run in either average or peak mode. In average mode, the result is as follows:

Transfer rates for each interface:
Load is percent load at the interface in flits per cycle; GBps is data bandwidth for data interfaces;
Expected is percent load expected based on analyze traffic; Ratio is ratio between actual and expected load.

Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%		Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%
m1/m.ar.out	0	0	1000	-	-	-	-		m3/m.b.in	100	0	1000	1.00%	-	100.00%	1.00%
m1/m.aww.out	4800	64	1000	48.00%	3.84	400.00%	12.00%		m3/m.r.in	0	64	1000	-	-	-	-
m1/m.b.in	1200	0	1000	12.00%	-	100.00%	12.00%		s/s.ar.in	0	0	1000	-	-	-	-
m1/m.r.in	0	64	1000	-	-	-	-		s/s.aww.in	10000	64	1000	100.00%	8	1200.00%	8.33%
m2/m.ar.out	0	0	1000	-	-	-	-		s/s.b.out	2500	0	1000	25.00%	-	300.00%	8.33%
m2/m.aww.out	4800	64	1000	48.00%	3.84	400.00%	12.00%		s/s.r.out	0	64	1000	-	-	-	-
m2/m.b.in	1200	0	1000	12.00%	-	100.00%	12.00%									
m2/m.r.in	0	64	1000	-	-	-	-									
m3/m.ar.out	0	0	1000	-	-	-	-									
m3/m.aww.out	400	64	1000	4.00%	0.32	400.00%	1.00%									

It can be observed that the loading (Load%) for m3/m.aww.out is at 4.00%; whereas, m1/m.aww.out and m2/m.aww.out are at 48.00%. Please note that, by default, each message is of 4 flits. Taking into consideration that we have set the avg_rate_design_limit to be 0.01, the real loading on the interface would be 0.01* 4= 0.04= 4.00%. The remaining 96% is co-shared by m1 and m2.

If to run the simulation in peak mode, the result is as follows:

Transfer rates for each interface:
Load is percent load at the interface in flits per cycle; GBps is data bandwidth for data interfaces;
Expected is percent load expected based on analyze traffic; Ratio is ratio between actual and expected load.

Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%		Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%
m1/m.ar.out	0	0	1000	-	-	-	-		m3/m.b.in	200	0	1000	2.00%	-	100.00%	2.00%
m1/m.aww.out	4600	64	1000	46.00%	3.68	400.00%	11.50%		m3/m.r.in	0	64	1000	-	-	-	-
m1/m.b.in	1150	0	1000	11.50%	-	100.00%	11.50%		s/s.ar.in	0	0	1000	-	-	-	-
m1/m.r.in	0	64	1000	-	-	-	-		s/s.aww.in	10000	64	1000	100.00%	8	1200.00%	8.33%
m2/m.ar.out	0	0	1000	-	-	-	-		s/s.b.out	2500	0	1000	25.00%	-	300.00%	8.33%
m2/m.aww.out	4600	64	1000	46.00%	3.68	400.00%	11.50%		s/s.r.out	0	64	1000	-	-	-	-
m2/m.b.in	1150	0	1000	11.50%	-	100.00%	11.50%									
m2/m.r.in	0	64	1000	-	-	-	-									
m3/m.ar.out	0	0	1000	-	-	-	-									
m3/m.aww.out	800	64	1000	8.00%	0.64	400.00%	2.00%									

It can be observed that the loading (Load%) for m3/m.aww.out is at 8.00%; whereas m1/m.aww.out and m2/m.aw.out are at 46%. The same reasoning applies that 8.00%= 0.02* 4= 0.08.

7.5 Dynamic Priority Support for Isochronous Traffic

7.5.1 Isochronous Traffic

A common requirement in mobile SoCs is to have some support for isochronous traffic. Isochronous traffic has real-time requirements. Common examples are audio and display traffic. A chip's display engine must be able to fetch the frame information within a bounded amount of time so it can display it to the monitor. If it cannot make the real-time requirement, the image display will be corrupted, and some display engines may deadlock.

While the display traffic has very strict upper bounds, they don't benefit at all from data returning early. And since the upper bounds are often quite large (10s of microseconds), it makes no sense to treat these as high priorities as that can reduce the performance of the rest of the system. Instead, it is common to treat this traffic as having dynamically controlled priority. This traffic will have low priority at first. The isochronous traffic will complete opportunistically when there is available bandwidth, letting more latency sensitive traffic go first. However, when the upper bound approaches, the isochronous traffic must be able to increase priority. The increased priority does not just affect new transactions. It must increase the priority of all prior isochronous transactions.

7.5.2 Two Priority Level Specification

The dynamic priority support for isochronous traffic is available in NetSpeed IP. Dynamic priority allows traffic classes to be specified with two priority levels. A side-band input will be created for each traffic class with alternative priority levels. The input will change the behavior of all bridges and routers to use the alternative priority. This will affect the behavior of all outstanding and new requests in that traffic class.

Each traffic class can be given two priority levels, although they will default to having only one. Using the *class_pri_map* command in NocStudio, the original priority for that class can be modified. This command also allows setting up an alternative priority level.

While isochronous traffic is one obvious use for this dynamic priority control, this feature can be used widely. An agent with high bandwidth traffic creation may be set as lower priority so it doesn't starve more latency sensitive traffic, but if it is being starved itself, could raise the priority to a higher level.

8 NoC Serviceability: Regbus Layer

8.1 THE REGISTER BUS

NetSpeed bridges and routers support registers for address ranges, QoS weights, error logging, event counting, and interrupt generation and masking. These registers can be used to debug or configure the network and must be accessed by a privileged host, using an access layer that remains active even when the data layers are stalled. NocStudio provides the option of adding a *Regbus* layer that meets these requirements, accessed using a single Regbus master bridge.

8.1.1 Regbus Master Bridge

The privileged master unit that manages the network must interact with the Regbus layer through the Regbus master bridge. A block diagram of the bridge is shown in Figure 41. The Regbus master bridge is a specialized version of an AXI bridge with the following restrictions:

- The AXI interface assumes a 32-bit master.
- AxLEN is restricted to 0 or 1 to allow either 32-bit or 64-bit register access.
- The NoC bridge address and router elements are determined and allocated by NocStudio. These are not user modifiable.
- The register-bus master bridge can be configured to have up to 16 outstanding read requests and 16 outstanding write requests.

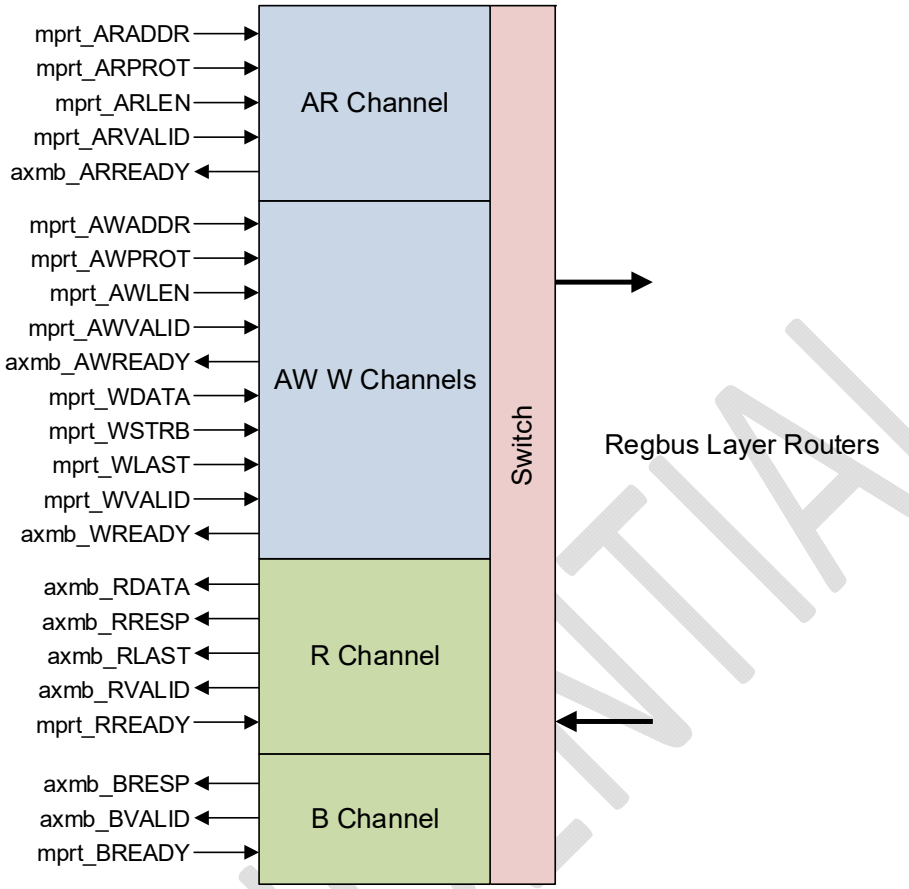


Figure 41. Regbus Master Bridge

8.1.2 The Regbus Layer

As shown in Figure 42, the Regbus layer is physically separate from other NoC layers. It is implemented using NetSpeed routers and uses the same topology as the other layers. At each grid point or node in the multilayer NoC, a *RingMaster* unit is connected to a Regbus layer router. All configurable registers in every bridge or router at that node are accessible through ring interconnects from the RingMaster. By default, NocStudio attempts to minimize the Regbus cost by sizing data widths to 32-bit or lower. NocStudio allows minimal user intervention during build of this network.

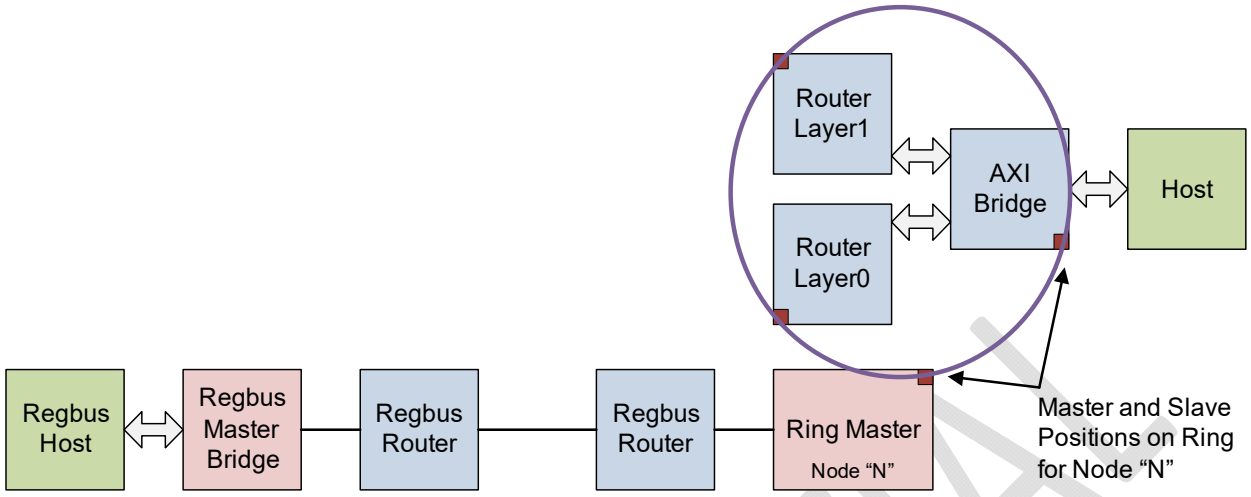


Figure 42. Regbus Layer Communication

By default, the Regbus master bridge, Regbus routers, and RingMaster all run at a common frequency. However, different clock domains can be defined on the regbus layer similar to normal NoC layers. The Ringmaster runs synchronously with the connected regbus router but can serve one or more rings each in its own clock domain. Each ring is in the same clock domain as the normal layer bridge or router elements it is serving.

NocStudio assigns all NoC elements to a contiguous address space and programs the addresses into the Regbus master-bridge address tables. The addresses are not user modifiable.

8.1.3 Connecting to a Regbus Master over the Primary NoC

Occasionally, a host on the Primary NoC layer (such as a CPU) might need to configure another NoC host, access its internal registers to monitor status, or collect information for performance and debug. The CPU might not have an additional port to connect to the Regbus master bridge. To handle this, the NoC architecture provides a NetSpeed *tunnel block* that acts as a slave on the primary NoC layers and as a master to the Regbus master bridge.

Figure 43 shows how a CPU that is a primary NoC layer host connects to the Regbus master bridge. This provides connectivity to the Regbus layer, and access to NoC internal registers and host registers through configuration ports on the Regbus ring.

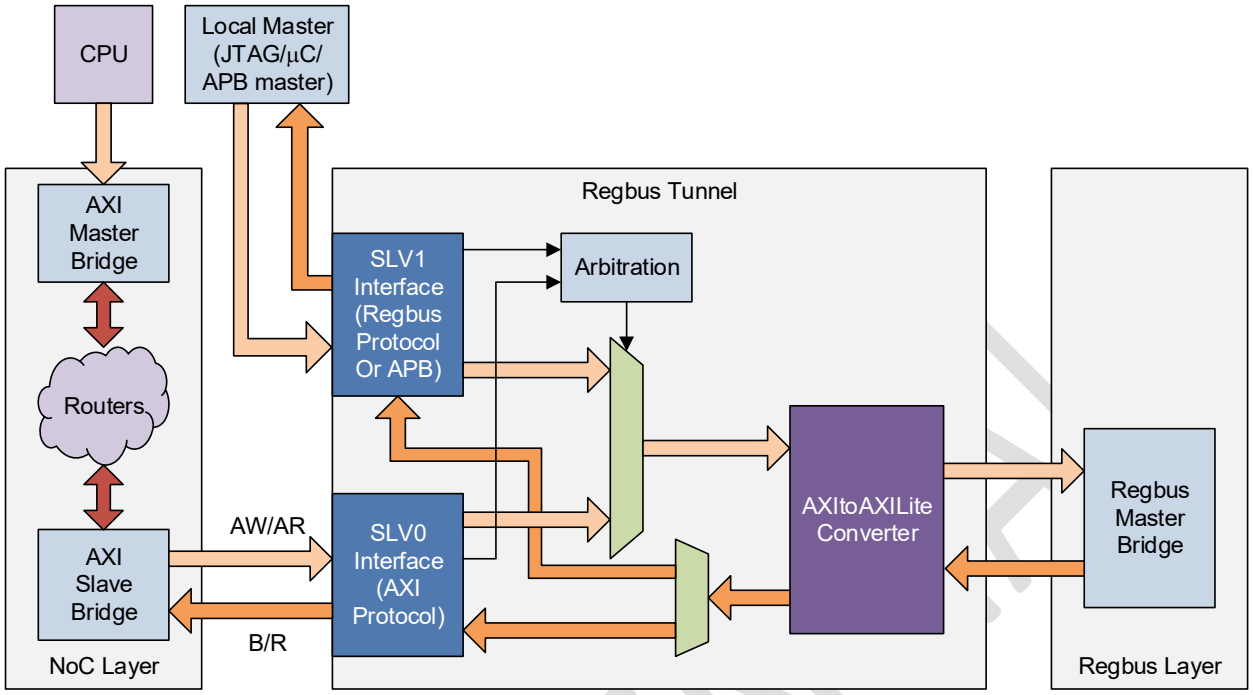


Figure 43. Regbus Tunnel Connects Primary NoC Layer to Regbus Layer

Traffic flows must be set up between one or more privileged masters on the NoC and the tunnel slave bridge. The tunnel address range can be a contiguous space covering all host and NoC configuration-register spaces and can have secure access attributes defined through NocStudio. This allows only privileged code running on the CPU to access this secure space. Host configuration-register space is defined on host configuration bridges and is mapped to the Regbus master bridge by NocStudio.

An additional port is provided on the tunnel unit for other masters, such as a JTAG or boot controller. This port can be configured as a 32-bit AXI-Lite port or an APB port. Arbitration between the ports is done within the tunnel.

8.1.4 Configuring the regbus

The NetSpeed NoC Register Bus provides access to the registers of the NoC elements. In addition to NetSpeed's own registers, we provide the feature of providing register bus access to a user's host registers. This access is made via the Register Bus Master (or through a host via the Tunnel). The Register Bus Master packetizes the access onto the register bus layer, to the specified host. There are four interfaces available to connect the host's registers: APB, AHB lite, AXI4 lite and a NetSpeed Native Register interface.

8.1.4.1 Usage with tunnel

When accessing the register bus via the Tunnel, the tunnel range comes into play. Example:

```
add_range rbm/s rbm_s_tunnel_range 0x1_0000_0000:0xfff_ffff_0000_0000 programmable 0
```

The above command defines the system address space for registers which is accessible through tunnel. This encompasses both the user register space and the NetSpeed NoC register space.

NoC address space can be allocated in two configurations. In compacted mode, the amount of address space taken up by NoC registers is lesser. Non-compacted mode consumes more address space but allows simpler decoding in the regbus master bridge.

Address space for user registers are assigned using add_range command. For example, following command assigns a range to a user register port h1/reg1

```
add_range h1/reg1 h1_reg_1  0x0000_5000-0x0000_50FF  0
```

Mesh property noc_register_base can be used to define the base address of NoC registers within regbus address map. By default, NocStudio assigns the address above the last user host register range to internal NoC registers. It is up to the user to size the tunnel range, and adjust the noc_register_base so that the tunnel range covers the entire user register space plus the NoC register space.

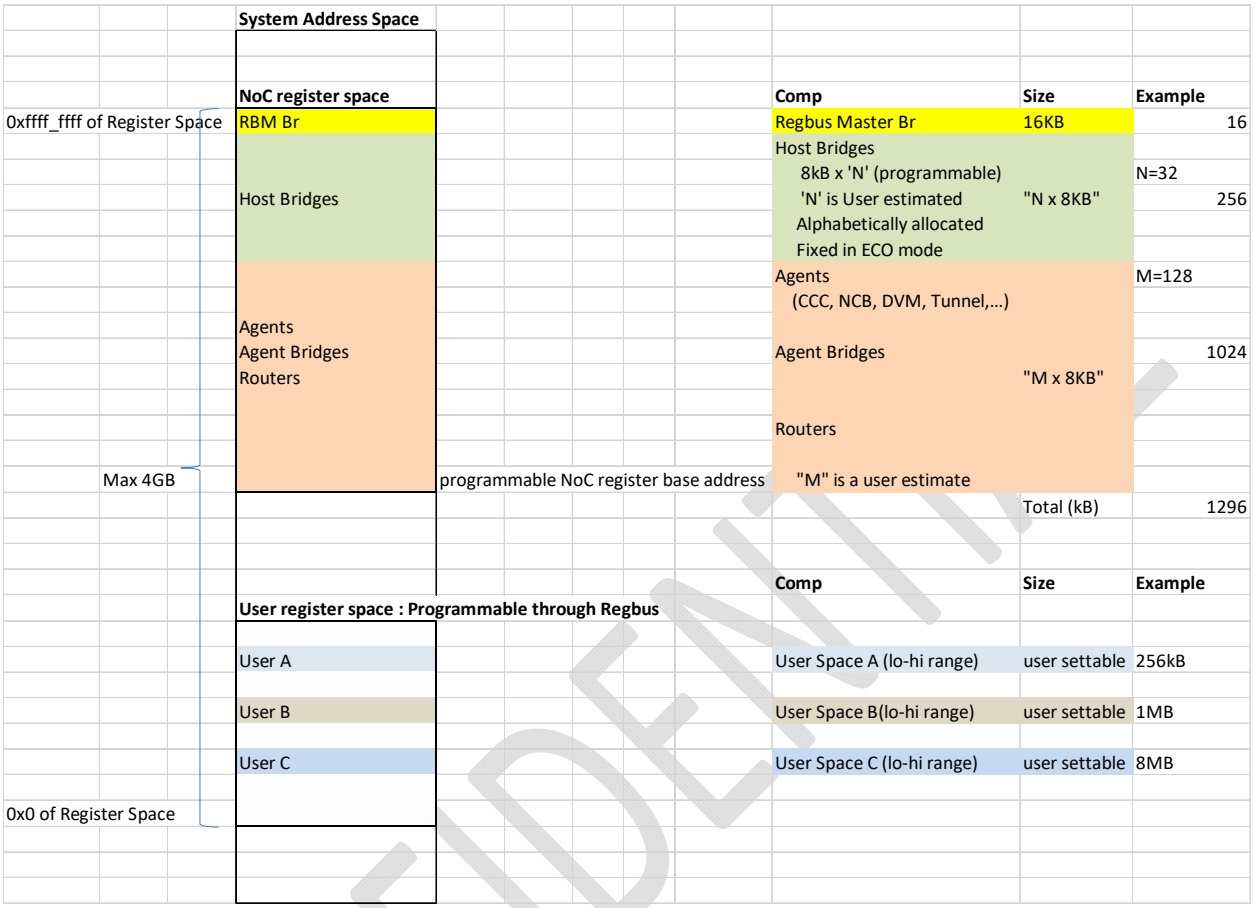


Figure 44: Regbus Address Map

8.2 NoC Registers

NoC registers are automatically created by NocStudio and placed in a fixed register bus address map. This address map is unrelated to any address map within the main NoC design.

For details of the registers and register address map, refer to noc_reference_manual.html and noc_registers.csv (which only appears if register bus is enabled) generated by NocStudio in the project directory.

Registers can be 32-bit wide, or 64-bit wide. Register sizes are indicated by the width of their reset values inside noc_registers.csv (or noc_reference_manual.html). Within noc_registers.csv, the following register attribute nomenclature is followed.

Table 4: Register attribute table

Register attribute	Description

rw	Read-Write register. All bits in this register are writable (except for u, A, B)
r	Read-only register. All bits in this register are read-only and cannot be written to. These are usually status registers
wzc	Write-zero-to-clear register. This register contains fields that must be written with zeroes to clear. These are usually error registers

Each individual bit inside a register has fine-grained bit attributes. Reset values of the registers are concatenations of each of these bit attributes in bit order.

Table 5: Register bit attribute table

Register bit attribute	Description
u	Unused. These bits have no associated flops and return 0 when read
r	Reserved. These bits are reserved for future expansion and have associated flops. Flop reset value is 0
A	Unwritable 0. These bits are part of a bigger field, but do not have associated flops to save area
B	Unwritable 1. These bits are part of a bigger field, but do have associated flops to save area
0	Reset value of 0. These bits have an associated flop
1	Reset value of 1. These bits have an associated flop

8.3 Error Responses To Register Accesses

NetSpeed NoC registers can be 32-bit wide or 64-bit wide. All NoC registers are aligned to 64-bit addresses. Each NoC register also has a secure/non-secure attribute. The register bus master allows 32-bit as well as 64-bit accesses to the register space. Some accesses may return errors due to decode failures. Below is a list of combinations and their expected error responses.

Table 6: Response table for NoC Register Accesses

Type of Access	Response
32-bit access to defined 32-bit register	Okay
64-bit access to defined 64-bit register	Okay

64-bit access to defined 32-bit register	Okay
32-bit access to defined 64-bit register	Okay. Each half of the 64-bit register can be accessed using 32-bit access
32-bit access to non-existing register address	Decode Error
64-bit access to non-existing register address 64-bit access to an address which is aligned to 32-bits Read access to secure register with AxPROT[1] = 1 Write access to secure register with AxPROT[1] = 1 Read/Write access to non-secure register with any AxPROT[1]	Decode Error Decode Error No read performed. 0 data and decode error response is returned No write performed. Decode error response is returned Okay

8.4 User Register Bus Access

The NocStudio User Manual contains the description on how to add access for a user's registers via the NetSpeed Register Bus. Please check your release version to see if this is supported for your release.

There are four protocols via which this can be done: AHB-lite, AXI4-lite, APB and a NetSpeed Native Register Protocol. Data width may be 32-bits or 64-bits wide. Narrow accesses are not supported on any of these interfaces. Responses to narrow accesses are returned as decode errors.

Table 7: Response table for User Register Bus Accesses

Type of Access	Response
32-bit access to 32-bit interface	Okay
64-bit access to 64-bit interface	Okay
64-bit access to 32-bit interface	Decode Error
32-bit access to 64-bit interface	Decode Error

8.5 Register Bus Master Interface

The register master is the entry port into the register layer. This privileged master unit that manages the register bus network must interact with this layer through the Regbus master bridge. The Regbus master bridge is a specialized version of an AXI bridge.

- Interface on the AXI side assumes a 32b master.
- AxLEN restricted to 0,1 to allow either 32b or 64b register access
- Address of NoC bridge and router elements are decided and allocated by NocStudio. These are not user modifiable.
- The register bus master bridge can be configured to have as many as 16 outstanding requests on reads and 16 on writes

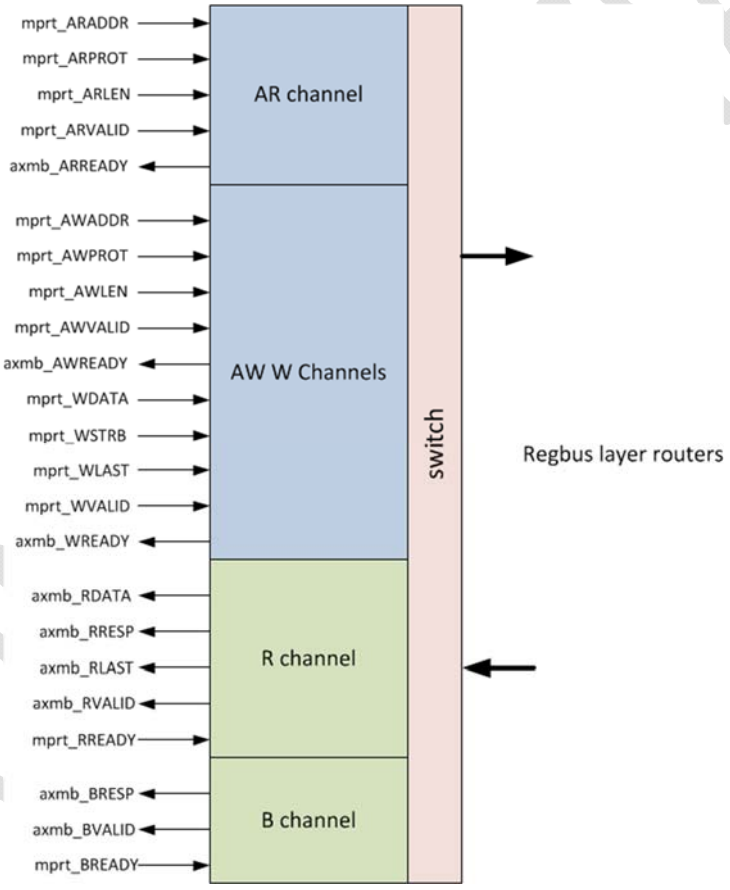


Figure 45: Register bus master bridge

The list of input signals is specified below:

Table 8: Register Bus Master Interface signals

Signals	Width (number of bits)	Usage	Description
Inputs			
rbm_m_regbus_clk	1	Mandatory	Register bus clock (may or may not be the same as the chosen noc clock)
rbm_m_regbus_reset_n	1	Mandatory	Active low reset
rbm_m_araddr	32	Mandatory	32-bit register read address (Bit 31 set to 0 for non-NetSpeed registers)
rbm_m_arprot	3	Mandatory	Read protection bits
rbm_m_arvalid	1	Mandatory	Read valid signal
rbm_m_arlen	1	Mandatory	Read length. 0 indicates 32B read. 1 indicates 64B read
rbm_m_rready	1	Mandatory	Read response ready signal indicating acceptance of read response
rbm_m_awaddr	32	Mandatory	32-bit register write address (Bit 31 set to 0 for non-NetSpeed registers)
rbm_m_awprot	3	Mandatory	Write protection bits
rbm_m_awvalid	1	Mandatory	Write valid signal
rbm_m_awlen	1	Mandatory	Write length. 0 indicates 32B read. 1 indicates 64B read
rbm_m_wdata	32	Mandatory	32-bit Write data
rbm_m_wstrb	4	Mandatory	Write strobe or byte enables
rbm_m_wvalid	1	Mandatory	Write data valid signal
rbm_m_wlast	1	Mandatory	Indicates the last beat of data. Set on the first beat if 32B, set on second bit if 64B

rbm_m_bready	1	Mandatory	Write response ready signal indicating acceptance of write response
Outputs			
rbm_m_arready	1	Mandatory	Read ready signal indicating acceptance of read request
rbm_m_rdata	32	Mandatory	32-bit response data
rbm_m_rresp	2	Mandatory	2-bit read response. 2'b00-okay, 2'b11-decode error, 2'b10-slave error
rbm_m_rvalid	1	Mandatory	Read response valid signal
rbm_m_rlast	1	Mandatory	Indicates the last beat of data. Set on the first beat if 32B, set on second bit if 64B
rbm_m_awready	1	Mandatory	Write command ready signal indicating acceptance of write request
rbm_m_wready	1	Mandatory	Write data ready signal indicating acceptance of write data
rbm_m_bresp	2	Mandatory	2-bit read response. 2'b00-okay, 2'b11-decode error, 2'b10-slave error
rbm_m_bvalid	1	Mandatory	Write response valid signal

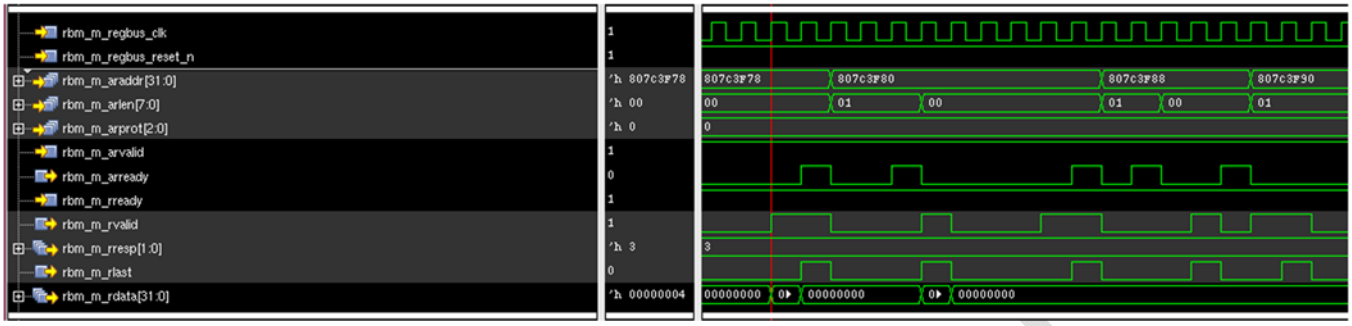


Figure 46: Waveform showing read requests and responses at the register bus master interface

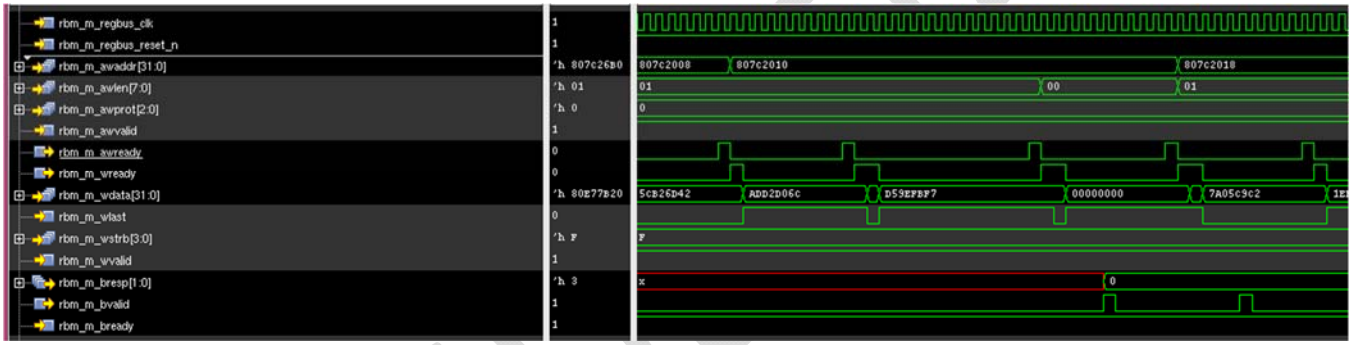


Figure 47: Waveform showing write requests and responses at the register bus master interface

8.6 Expected Usage of Register Bus Master

The NetSpeed Bridges and Routers support registers for QoS weights, error logging, event counting, and interrupt generation and masking. As these registers can be used to debug the state of the network, they must be accessed by a privileged host, and by an access layer that remains alive even if the data layers are stalled. Host registers connected to the regbus layer are also extended the advantage of debug through the regbus layer if the data layers are stalled.

The privileged host, or the 'Register Bus Master', can be part of a larger agent that handles configuration, power, reset and debug. It may also have a port on the data layers of the NoC through which it is controlled by CPUs so that the CPUs can access the regbus layer indirectly.

8.7 Ring Slave to Host Interface

On the ring slave to host interface, a combined read/write bus is used. The interface is very similar to an AXI-lite interface. It follows the same flow control ready/valid protocol. This interface runs on the chosen NoC clock. It also has an active high reset.

Rules:

- If more than one request is permitted to be outstanding to the host, the host must return the responses to the ring slave in order. Read responses must be returned in order with respect to each other. Similarly, write responses must be returned in order with respect to each other. Read response ordering with respect to write responses (or vice versa) is not expected. Read and write responses may come back out of order with respect to each other, as long as they are ordered within their respective channels.
- The address requested on the bus is the lowest address being requested. For example, a 32-bit or 4B write request to an address 0x40 indicates that the write is meant for byte offsets 0x43, 0x42, 0x41, 0x40.
- Flow control by means of a ready signal is present on this interface. The valid signal, if asserted, must remain asserted until it receives a ready. All fields on the interface must also remain unchanged until the ready has been received. There are two sets of valid/ready signals: req_valid/req_ready, rsp_valid/rsp_ready.
- A ring slave can be allowed to have multiple outstanding requests to the host indicated by the programmable parameter P_REGBUS_RSLV_NUM_OUTSTANDING.

8.8 Atomic Operations

On the ring slave to host interface, each request and response is transferred in a single cycle. Whether a write is a 32-bit write or a 64-bit write, all bits of write data are presented on the interface at the same time. The same is true for read response data. The single cycle transfer makes all transactions on this interface inherently atomic.

Table 9: Register slave to host interface

Signals	Width (number of bits)	Usage	Description
Inputs			
clk	1	Mandatory	Same as chosen noc clock
reset	1	Mandatory	Active high reset
regslv_rsp_valid	1	Mandatory	When 1, indicates a valid response from the host
regslv_rsp_rnw	1	Mandatory	When 1, indicates a read response. When 0, indicates a write response
regslv_rsp_rdata	32 or 64 (parameter)	Mandatory	The data is transferred in the same cycle as

			regslv_rsp_valid. If size=0, the least significant 32 bits are the ones returned to the regbus master
regslv_rsp_err	2	Mandatory	2-bit. Indicates slave error when slave exists, but no register at the location specified. The slave is free to return a decode error instead of a slave error if it so chooses. (AMBA spec: 2'b10=Slave error (slave exists, but no register at the location specified). 2'b11=Decode error (no slave exists). Decode error will be returned by the ring master when it receives a request back from the ring that wasn't accepted by any slave)
regslv_req_ready	1	Mandatory	When asserted at the same time as regslv_req_valid, indicates the acceptance of that request
Outputs			
regslv_req_valid	1	Mandatory	When 1, indicates a valid request from ring slave to the host
regslv_req_addr	31 or less (parameter)	Mandatory	Register read or write address
regslv_req_rnw	1	Mandatory	Read not Write. When regslv_req_valid=1,

			regslv_req_rnw=1, a read is being requested. When regslv_req_valid=1, regslv_req_rnw=0, a write is being requested
regslv_req_size	1	Mandatory	0 indicates a 32-bit request. 1 indicates a 64-bit request
regslv_req_region	4	Optional	Passes along the address map sub-slave information for devices behind this device
regslv_req_prot	3	Optional	Passes along the 3-bit ARPROT/AWPROT field presented to the register bus master for this transaction
regslv_req_wdata	32 or 64 (parameter)	Mandatory	The data is transferred in the same cycle as regslv_req_valid. P_REGBUS_RSLV_DATA_WIDTH can be 32-bit or 64-bit. If P_REGBUS_RSLV_DATA_WIDTH=64 and size=0, it indicates the least significant 32 bits should be accessed, that is, bits 31:0
regslv_req_wstrb	4 or 8	Optional	Indicates the write strobes or byte enables for write data
regslv_rsp_ready	1	Mandatory	When asserted at the same time as regslv_rsp_valid, indicates the acceptance of that request

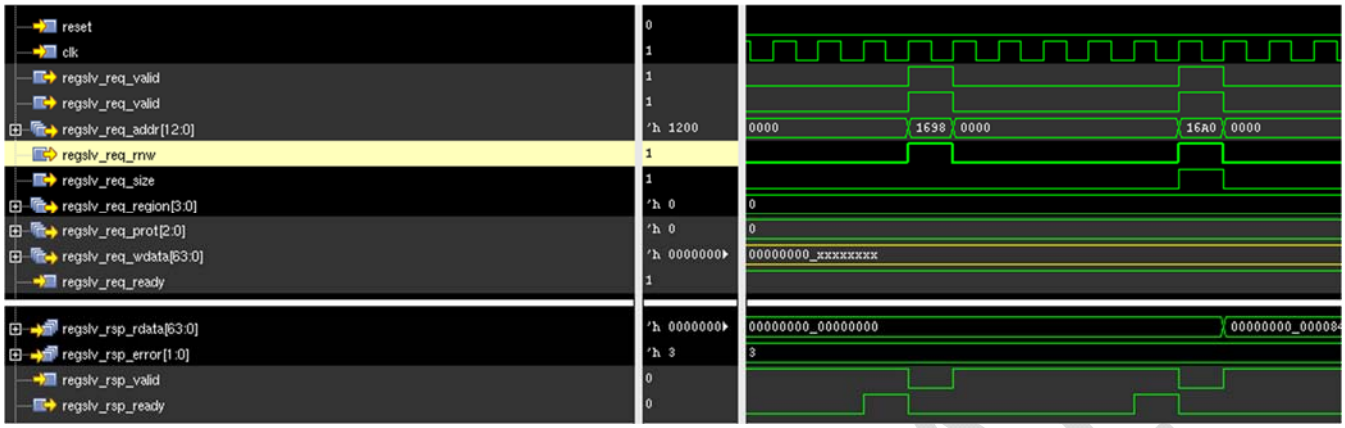


Figure 48 : Waveform showing ring slave read requests and responses (4B and 8B)

Figure 48 shows examples of 4B and 8B read requests and their responses. In this example, read responses show decode errors. Write data (regslv_req_wdata) is don't-care because these are read requests.

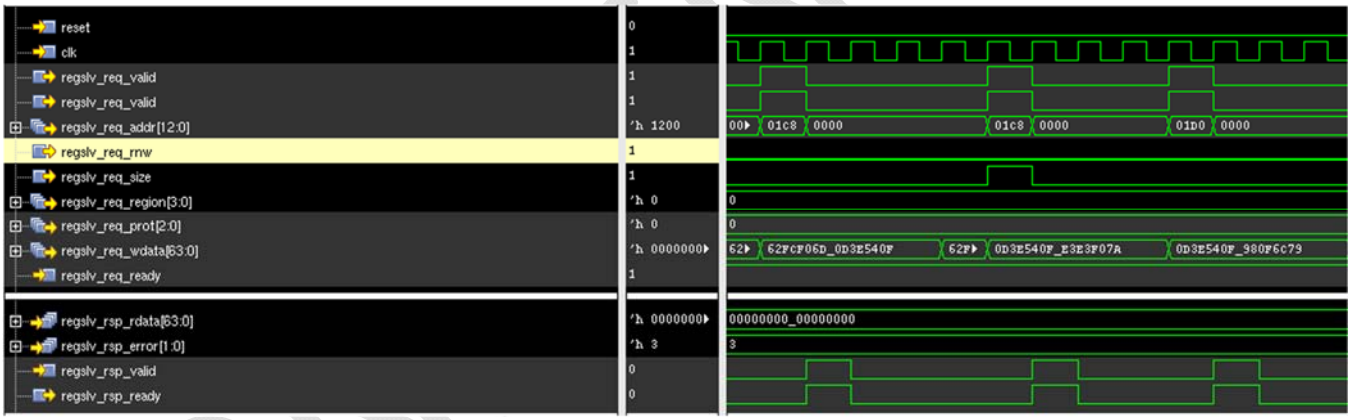


Figure 49 : Waveform showing ring slave write requests and responses (4B and 8B)

Figure 49 shows examples of 4B and 8B write requests and their responses. In this example, the write responses are decode errors. Read data (regslv_req_rdata) is don't-care because these are write requests.

8.9 RESTRICTIONS

- Regbus master bridge implements a customized AXI protocol with 32b data width, AxLEN of 0, 1 to support 32b, 64b register accesses over a 32b down-sizeable NoC layer interface.
- User reg bus, have the following restrictions:
    - Transaction size needs to equal the interface size:
        - If you have a 32-bit rb native interface, 32-bit user reg bus accesses are supported

    - If you have a 64-bit rb native interface, 64-bit user reg bus accesses are supported. 32-bit accesses are not supported.
- Host errors returned from the rb native interface are not supported.
- Async interface at rb native is not supported.
- Write strobes (byte enables) are not supported. This is similar to what we support for the NOC internal registers.

9 Safety and Reliability

For safe and reliable operation of a device, error free and fault-tolerant operation of the interconnection networks used in the device is crucial. Random faults can occur in the storage elements and wiring resources used by a system wide interconnect. Such errors must be detected and corrected when possible and all uncorrected errors must be notified to system software for intervention.

9.1 TRANSPORT ERROR DETECTION AND CORRECTION

Data is exchanged between agents through the NoC using a packet protocol. Different levels of transport error resilience can be configured for the NoC transport infrastructure. Packets transported over the NoC can be broadly viewed as comprising of three fields.

1. Data field: This is usually some power of two multiples of an integer number of bits. Interfaces to agents and data part of NoC links belong to this category. This part can undergo upsizing and downsizing while being transported across the NoC.
2. Sideband field: An example of this is AW command carried on sideband of the AWW channel. This field does not undergo resizing through the network.
3. Packet control fields: A packet also has signals for routing, delineation, credit return etc.

9.2 END TO END TRANSPORT ERROR CHECKING

Any flow through the NoC can be configured to provide error checking using ECC or parity. ECC uses hamming code with additional parity bit to provide SECDED code. This code can correct single bit errors and detect double bit errors in a block of data. Parity only allows detection of odd number of bit errors.

9.2.1 Data protection: Per flit ECC

On a transmit bridge for every layer with ECC protection enabled, ECC is calculated over each data flit and sent along with the flit. At the receiving end, ECC is used to detect and correct any errors in the data flit received from NoC layer before delivering to the receive host interface.

- Granularity of data width over which ECC is computed is derived and configured by NocStudio globally on each NoC layer. Smallest possible granularity is the CELL_SIZE configured on that layer. However, if the narrowest interface communicating on that layer or narrowest NoC link on that layer is *N**CELL_SIZE, then this must be the granularity over which ECC is computed. Note that narrower granularity increases area overhead for ECC but provides higher detection and correction coverage. Configured granularity is a power-of-2 multiple of cell size on a layer.

An example is regbus layer, where each interface is typically 36-bits (4-cells), but NoC links can be as narrow as 9-bits (1-cell) if downsizing is performed. In this case, ECC granularity would be 9-bits.

- User can specify a maximum granularity over which ECC is to be computed. Consider a NoC where all host interfaces are 512-bits with no downsizing in the NoC. In this case, the default ECC calculation granularity will be 512-bits. However, the user may choose to specify a smaller granularity of 64-bits for ECC computation to allow better timing performance.

  In summary, global granularity selected by NocStudio for ECC computation will be the smaller value between narrowest link/interface width and user specified maximum granularity. Every cycle, Multiple ECC/Parity code words are computed in parallel, one for each '*granularity'* wide segment of data/sideband of the flit. Computed ECC is transported similar to data flits and will undergo upsizing and downsizing with its associated data flit. ECC generation at the transmitting end and detection and correction at the receiving end will add a cycle each to overall path latency.

9.2.2 Data error detection: Per flit parity

As an alternative to ECC, user may configure parity to be transported with the data flits for detecting odd number of bit errors. Granularity of data width over which parity is calculated and transported is as specified for ECC. Parity based protection does not add latency to the path.

9.2.3 Data protection: Transport of user provided ECC

Another alternative allows the use to generate ECC on the data and provide it on the interface using USER bits. In this case, NoC merely transports the ECC bits from transmitting to receiving end. Note that this option is only applicable to DATA flits which do not undergo any modification in the NoC. Command fields can be modified by the NoC and hence user provided ECC will lose its integrity.

User provided ECC should be provided per byte of data through the '*Per byte user bits'* interface. This is transported in the data cells and can hence undergo upsizing/downsizing in the NoC.

9.2.4 Sideband protection: ECC or Parity

Similar to data, information carried in packet sideband will be protected end-to-end using ECC or parity. Sideband associated with an interface has the same width over the entire network, this field does not undergo upsizing/downsizing in the NoC. Sideband width is increased to the next multiple of ECC computation granularity using msb 0 padding. At the transmitting end,

ECC is calculated on sideband segments at the selected granularity and at the receiving end, error detection and correction are performed.

9.3 HOP TO HOP ERROR CHECKING

If data or user side band is protected by ECC, then error check operations on these fields are only performed at the NoC endpoints. However, if parity is applied to data and sideband, then parity error detection on these fields occurs at every hop of the network. Similarly, other fields of packet are covered by parity error detection at every hop of the network.

9.3.1 Protection of packet control fields

These fields associated with every packet flit can undergo modifications as the packet is routed over the NoC. At the transmitter, parity is calculated over these fields and sent along with the flit. At every downstream hop, parity field is used to detect any error and may be recomputed for the next hop. A dedicated parity bit is used to protect each of these signal groups.

Packet delineation fields

Name	Width	
flit_valid	4	Flit valid
flit_sop	1	Start of packet
flit_eop	1	End of packet
flit_bv	log2(DATA_WIDTH)	This signal is present only on the router links. This indicates the number of cells valid in the EOP flit of a packet.

Packet routing information

Name	Width	
flit_route_info	P_ROUTE_INFO_WIDTH	Routing information
req_outp	3	Next hop output port

Link flow-control credits

Name	Width	

credit_inc	4	Credit return

9.4 INTERFACE PARITY

Interface parity protection is provided to AXI/ACE protocol interfaces.

9.4.1 Configuration

Channel	Parity signals	Signals protected	*Configuration Parameter*	
			intfparity_addr_per_byte	***intfparity_per_byte***
AR	ARADDRPAR	ARADDR	0: single parity bit 1(default): one parity bit per byte	NA
	ARPAR	All command bits except address. Refer to AR/AW concatenation section	NA	0(default): single parity bit 1: one parity bit per byte
AW	AWADDRPAR	AWADDR	0: single parity bit 1(default): one parity bit per byte	NA
	AWPAR	All command bits except address. Refer to AR/AW concatenation section	NA	0(default): single parity bit 1: one parity bit per byte
W	WPAR	{WDATA[8*i+:8], WSTRB[i]}	NA	NA
R	RPAR	RDATA[8*i+:8]	NA	NA
	RRESPPAR	{RRESP, RID, RLAST}	NA	0(d) : single parity bit

				1 : one parity bit per byte
B	BRESPPAR	{BRESP, BID}	NA	0(d) : single parity bit 1 : one parity bit per byte
AC	ACADDRPAR	ACADDR	0: single parity bit 1(d): one parity bit per byte	NA
	ACPAR	{ACSNOOP, ACPROT}	NA	0(d) : single parity bit 1 : one parity bit per byte
CD	CDPAR	CDATA[8*i+:8]	NA	NA

9.4.2 Functional description

- Master on an interface, provides parity calculated over all fields of AR and AW commands
- Slave provides parity for R and B
- Data parity is provided per byte
- Parity is valid for every beat of information on these interfaces. Parity is checked off at the receiving end of the same interface before any transformation is performed.
- AR, AW, W, R, B, CR, CD, AC and ACK are channels on ACE interfaces. Depending on the type of an ACE port, some of the channels may be absent. The exact set of signals on any of these channels also depends on the port type.
    - Parity protection can be using a single bit for the entire interface or one bit per 8-bits of interface signals.
    - Data interfaces R, W, and CD always compute a parity bit per byte of data.
    - AR, AW channels: two optional parity signals added on the interface:
        - Address parity: Parity over AxADDR
        - Command party: Parity over all other Ax fields
    - CR, BRESP, RRESP, AC: Single bit or byte granular parity
    - WDATA, CDDATA, RDATA: parity protection per byte of data

    - W interface has a parity bit per byte of WDATA and corresponding WSTRB.
- Parity is checked off as data bytes are received on the source interface, but it is also sent through unchanged to the remote end to repeat the byte parity check on the remote receiving end.
- This also offers coverage of the ASYNC FIFO, skid stage and ratio sync buffer of the valin/valout block

9.4.3 AR/AW Control signal concatenation

Below table depicts the order of concatenation of the AR/AW control signals, starting with MSB at the top, to the LSB at the bottom. MSB padded with 0 to nearest multiple of 8.

AR Control signals

ACE	ACEL	AXI4	AXI3
ARID, P_MST_AID_R_WIDTH	ARID, P_MST_AID_R_WIDTH	ARID, P_MST_AID_R_WIDTH	ARID, P_MST_AID_R_WIDTH
ARLEN, 8	ARLEN, 8	ARLEN, 8	{4'd0, ARLEN}
ARSIZE, 3	ARSIZE, 3	ARSIZE, 3	ARSIZE, 3
ARBURST, 2	ARBURST, 2	ARBURST, 2	ARBURST, 2
ARCACHE, 4	ARCACHE, 4	ARCACHE, 4	ARCACHE, 4
ARPROT, 3	ARPROT, 3	ARPROT, 3	ARPROT, 3
ARQOS, 4	ARQOS, 4	ARQOS, 4	4'd0
ARLOCK, 1	ARLOCK, 1	ARLOCK, 1	ARLOCK, 1
ARREGION, 4	ARREGION, 4	ARREGION, 4	ARREGION, 4
ARSNOOP, 4	ARSNOOP, 4	4'd0	4'd0
ARDOMAIN, 2	ARDOMAIN, 2	2'd0	2'd0
ARBAR, 2	ARBAR, 2	2'd0	2'd0

AW Control signals

ACE	ACEL	AXI4	AXI3

AWID, P_MST_AID_W_WIDTH	AWID, P_MST_AID_W_WIDTH	AWID, P_MST_AID_W_WIDTH	AWID, P_MST_AID_W_WIDTH
AWLEN, 8	AWLEN, 8	AWLEN, 8	{4'd0, AWLEN}
AWSIZE, 3	AWSIZE, 3	AWSIZE, 3	AWSIZE, 3
AWBURST, 2	AWBURST, 2	AWBURST, 2	AWBURST, 2
AWCACHE, 4	AWCACHE, 4	AWCACHE, 4	AWCACHE, 4
AWPROT, 3	AWPROT, 3	AWPROT, 3	AWPROT, 3
AWQOS, 4	AWQOS, 4	AWQOS, 4	4'd0
AWLOCK, 1	AWLOCK, 1	AWLOCK, 1	AWLOCK[0]
AWREGION, 4	AWREGION, 4	AWREGION, 4	AWREGION, 4
AWSNOOP, 3	AWSNOOP, 3	3'd0	3'd0
AWDOMAIN, 2	AWDOMAIN, 2	2'd0	2'd0
AWBAR, 2	AWBAR, 2	2'd0	2'd0
AWUNIQUE, 1	AWUNIQUE, 1	1'd0	1'd0

9.4.4 Limitations

- Specifications
  - Supports all master bridges except AHBLM, AXI4L
  - Supports all slave bridges except AHB, AXILITE, APB
  - No ARM R5/R7 compatibility support
- Implementation
  - AxUSER bits are not covered
  - Not supported for virtual AXI interface
  - Not supported on interface with VDC
  - No support for NetSpeed Agents (CCC, IOCB, LLC, DVM)
  - No DAU support
  - No Tunnel support
  - Limited Interface support (No WC, CR, CDC, RACK, WACK, valid/ready)
- NocStudio
  - No External Sanity Support for IF Parity (parity pins are not connected in the External Sanity TB, won't elaborate)

  - Using default_prop on parity props can enable parity on unsupported bridges.
- Backend
  - Incomplete SDC verification of IF Parity Pins

9.5 CONFIGURING ERROR CHECKING

Error checking on a flow is specified in NocStudio using *add_traffic* commands.

add_traffic [ecc/parity] …

This maps the flow on a NoC layer with ECC or parity-based error checking. Every bridge and router on that layer is configured to perform the selected error checking operations.

mesh_prop errorcheck_granularity

This property allows the user to specify an upper limit for granularity over which ECC is to be computed on data and sideband fields. NocStudio chooses the smaller value between this upper limit and the narrowest virtual channel data width on the layer.

Parity results in one extra bit for every '*granularity*' wide data/sideband segment. ECC requires 5 to 10 extra code bits for every segment.

Granularity (bits)	Code size per segment (bits)
1-11	5
12-26	6
27-57	7
58-120	8
121-247	9
248-502	10

502 bits is the maximum supported value of granularity.

For coherent NoCs, error checking can be enabled through *add_traffic* commands or as part of *setup_coherency* command.

setup_coherency –[ecc/parity]

9.6 ERROR REPORTING

On layers with error checking enabled, every router has status bits logging any parity error detected on data segment, user sideband segments, packet delineation fields, routing information and credit information on router links and within router virtual channel buffers. These status bits are sticky and raise an interrupt by default. These interrupts can be disabled by programming the interrupt masking registers for the status bits. Similarly, bridges log errors status with maskable interrupts for layers where error checking is enabled. Details of these are available in the register manual.

9.7 CONFIGURATION AND STATUS REGISTER PROTECTION

Configuration and status registers accessible over *REGBUS* can be provided error detection capability.

mesh_prop register_parity_enabled [yes/no]

Above NocStudio mesh property is used to enable this feature in a NoC. When enabled, all registers which are updated by *REGBUS* writes or internal events have parity bit for error detection. Each byte of a register has dedicated parity bit, and parity mismatch in any byte results in *register-parity-error* interrupt status bit to be set. By default, this generates an interrupt but can be masked by writing to appropriate interrupt mask register of the NoC element hosting the register.

Parity of a register content is calculated and recorded when a valid update such as reset, register write or internal status update occurs. Parity of the register is continuously checked against the recorded value and any mismatch is recorded.

10 Programmers Model

NetSpeed NoC delivers a rich set of registers used for NoC control, debug and performance monitoring of the NoC. This section describes all available registers to SoC designers. The complete list of registers implemented for a specific design can be found in the noc_reference_manual.html under the <project> folder when the RTL is generated by NocStudio.

Registers are divided into the following categories:

- Router registers
- Bridge registers
- AMBA registers
  - Axi Master registers
  - Axi Slave registers
- Regbus Master/Slave Bridge registers
- Protocol Converter registers
  - AHB2AXI
  - AXI2AHB
  - APB

10.1 Router Registers

10.1.1 RID – Router ID

This register holds layer and position information for the router. It is a read-only register. It can be used for debugging software access to the NoC elements by confirming that a read has successfully targeted the correct NoC element.

Attribute: R

Bit field description:

- ONE[24] - One
- ZERO[23:21] - Zeroes
- POS[20:5] - 16-bit position ID of this router in the NoC
- LAYER[4:0] - 5-bit identifier of the NoC layer on which this router is located

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

u	ONE	ZERO	POS	LAYER

Table 10 ID register.

10.1.2 RPERR – Router Parity Error

There is one register for each router port capturing parity error events occurring on the port. Parity errors are monitored on router physical link and also on data read from VC buffers of the router. Error status bits are sticky. First detected error while the status bit is in cleared state sets the bit. The bit needs to be explicitly cleared using zero write, before another error can be logged for that status bit. Following fields of information transported over the NoC are monitored for error at router ports.

1. Data Parity: Parity is checked over multiple segments of data in each flit. Parity error in any segment will be recorded in the data parity status bit. Note that parity is checked on data only if parity mode error check is enabled on the router's layer. In ECC mode, data parity is not monitored on each router.
2. User sideband parity: Similar to data field above.
3. Packet control parity: Parity over start of packet, end of packet, byte valid and data valid fields of a flit.
4. Routing information parity: Parity over routing information carried in every flit.
5. Credit parity: Parity monitored over credits returned downstream port.

Attribute: WZC

Bit field description:

- RI_3[31] - 1'b1: Parity Error in VC 3 Buffer Routing Information
- PK_3[30] - 1'b1: Parity Error in VC 3 Buffer Packet Delineation Controls
- SB_3[29] - 1'b1: Parity Error in VC 3 Buffer User Sideband
- D_3[28] - 1'b1: Parity Error in VC 3 Buffer Data
- RI_2[27] - 1'b1: Parity Error in VC 2 Buffer Routing Information
- PK_2[26] - 1'b1: Parity Error in VC 2 Buffer Packet Delineation Controls
- SB_2[25] - 1'b1: Parity Error in VC 2 Buffer User Sideband
- D_2[24] - 1'b1: Parity Error in VC 2 Buffer Data
- RI_1[23] - 1'b1: Parity Error in VC 1 Buffer Routing Information
- PK_1[22] - 1'b1: Parity Error in VC 1 Buffer Packet Delineation Controls
- SB_1[21] - 1'b1: Parity Error in VC 1 Buffer User Sideband
- D_1[20] - 1'b1: Parity Error in VC 1 Buffer Data
- RI_0[19] - 1'b1: Parity Error in VC 0 Buffer Routing Information
- PK_0[18] - 1'b1: Parity Error in VC 0 Buffer Packet Delineation Controls

- SB_0[17] - 1'b1: Parity Error in VC 0 Buffer User Sideband
- D_0[16] - 1'b1: Parity Error in VC 0 Buffer Data
- CR[4] - 1'b1: Parity Error in Link Credit From Downstream Router
- RI[3] - 1'b1: Parity Error in Link Routing Information
- PK[2] - 1'b1: Parity Error in Link Packet Delineation Controls
- SB[1] - 1'b1: Parity Error in Link User Sideband
- D[0] - 1'b1: Parity Error in Link Data

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RI_3	PK_3	SB_3	D_3	RI_2	PK_2	SB_2	D_2	RI_1	PK_1	SB_1	D_1	RI_0	PK_0	SB_0	D_0	u											CR	RI	PK	SB	D

Table 11 RPERR register.

10.1.3 RPERRM – Router Parity Error Mask

One mask register bit for each parity status bit in RPERR. When mask bit is set, corresponding parity error does not cause an interrupt. Default state is reset for all mask bits, allowing interrupt on any parity error event

Attribute: RW

Bit field description:

- RI_3[31] - Mask Parity Error in VC 3 Buffer Routing Information.
- PK_3[30] - Mask Parity Error in VC 3 Buffer Packet Delineation Controls.
- SB_3[29] - Mask Parity Error in VC 3 Buffer User Sideband.
- D_3[28] - Mask Parity Error in VC 3 Buffer Data.
- RI_2[27] - Mask Parity Error in VC 2 Buffer Routing Information.
- PK_2[26] - Mask Parity Error in VC 2 Buffer Packet Delineation Controls.
- SB_2[25] - Mask Parity Error in VC 2 Buffer User Sideband.
- D_2[24] - Mask Parity Error in VC 2 Buffer Data.
- RI_1[23] - Mask Parity Error in VC 1 Buffer Routing Information.
- PK_1[22] - Mask Parity Error in VC 1 Buffer Packet Delineation Controls.
- SB_1[21] - Mask Parity Error in VC 1 Buffer User Sideband.
- D_1[20] - Mask Parity Error in VC 1 Buffer Data.
- RI_0[19] - Mask Parity Error in VC 0 Buffer Routing Information.
- PK_0[18] - Mask Parity Error in VC 0 Buffer Packet Delineation Controls.
- SB_0[17] - Mask Parity Error in VC 0 Buffer User Sideband.
- D_0[16] - Mask Parity Error in VC 0 Buffer Data.

- CR[4] - Mask Parity Error in Link Credit From Downstream Router.
- RI[3] - Mask Parity Error in Link Routing Information.
- PK[2] - Mask Parity Error in Link Packet Delineation Controls.
- SB[1] - Mask Parity Error in Link User Sideband.
- D[0] - Mask Parity Error in Link Data.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RI_3	PK_3	SB_3	D_3	RI_2	PK_2	SB_2	D_2	RI_1	PK_1	SB_1	D_1	RI_0	PK_0	SB_0	D_0	u											CR	RI	PK	SB	D

Table 12 RPERRM register.

10.1.4 RE – Router Event

This register tracks the interrupt or error events that can occur in the router. The only interrupt event is the event counter overflow. This register is readable, and can be cleared by performing a write with the write data bits set to 0 for the bits that should be cleared.

Attribute: WZC

Bit field description:

- KLU[16] - 1'b1: Traffic destined for K link which is unavailable
- JLU[15] - 1'b1: Traffic destined for J link which is unavailable
- ILU[14] - 1'b1: Traffic destined for I link which is unavailable
- HLU[13] - 1'b1: Traffic destined for H link which is unavailable
- SLU[12] - 1'b1: Traffic destined for South link which is unavailable
- WLU[11] - 1'b1: Traffic destined for West link which is unavailable
- ELU[10] - 1'b1: Traffic destined for East link which is unavailable
- NLU[9] - 1'b1: Traffic destined for North link which is unavailable
- PGE[8] - 1'b1: Power gating error, traffic received after router committed to power down
- OVFO[2]
  - 1'b1: In this status bit indicates that the router output event counter has overflowed (32'hFFFFFFFF -> 32'dh0), this is a sticky status bit
  - 1'b0: To clear
- CSR_PARERR[1] - 1'b1: Parity error in config/status registers
- OVFI[0]
  - 1'b1: In this status bit indicates that the router input event counter has

overflowed (32'hFFFFFFFF -> 32'dh0), this is a sticky status bit
- 1'b0: To clear

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U															KLU	JLU	ILU	HLU	SLU	WLU	ELU	NLU	PGE	u					OVFO	CSR_PARERR	OVFI

Table 13 RE register.

10.1.5 REC – Router Event Counter

This register holds the event counter. The value can be read to determine the current count value. The value can be written to initialize the counter. When events trigger a count, the counter will increment. When the counter increments at its highest value, it will roll over to zero and the overflow will mark the Router Event Interrupt Status register, which could trigger an interrupt.

Attribute: RW

Bit field description:

- EVENT_CNTR[31:0]
  - 32'bit event incrementing counter.
    Rollover from 32'hFFFFF → 32'd0 sets the rollover status bit RE

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVENT_CNTR																															

Table 14 REC register.

10.1.6 RECC – Router Event Counter Control

This register is used to select which hardware events will increment the event counter.

Attribute: RW

Bit field description:

- EVT[9:8]
  - 11: Generates count event when VC has valid data, but is stalled

- 10: Generates count event on every flit received for the selected input port and selected input VCs, this can be used to count total flits received on a router input port
- 01: Generates count event on every EOP received for the selected input port and selected input VCs, this can be used to count packets received on a router input port
- 00: Disable

- INP[6:4] - Input port on which the event is captured
- IVC[1:0]
  - 11: Input VC 3
  - 10: Input VC 2
  - 01: Input VC 1
  - 00: Input VC 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																						EVT	u	INP				u		IVC	

Table 15 RECC register.

10.1.7 REM – Router Event Mask

This register is used to select whether the interrupt events in the Router Event Interrupt Status register should send an interrupt when asserted. If the corresponding bit is set to 1, an interrupt will not be sent. This register can be read and written to.

Attribute: RW

Bit field description:

- MK[16] - 1'b1: Mask KLU error interrupt
- MJ[15] - 1'b1: Mask JLU error interrupt
- MI[14] - 1'b1: Mask ILU error interrupt
- MH[13] - 1'b1: Mask HLU error interrupt
- MS[12] - 1'b1: Mask SLU error interrupt
- MW[11] - 1'b1: Mask WLU error interrupt
- ME[10] - 1'b1: Mask ELU error interrupt
- MN[9] - 1'b1: Mask NLU error interrupt
- PGM[8] - 1'b1: Mask PGE error interrupt
- OVFOM[2]
  - 1'b1: Masks or disables an interrupt from being generated by the output event

count overflow status bit (RE)
- 1'b0: Enables an interrupt to be generated when event counter status bit is set

- CSR_PARERRM[1] - 1'b1: Mask CSR parity error interrupt
- OVFIM[0]
  - 1'b1: Masks or disables an interrupt from being generated by the input event count overflow status bit (RE)
  - 1'b0: Enables an interrupt to be generated when event counter status bit is set

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u															MK	MJ	MI	MH	MS	MW	ME	MN	PGM	u					OVFOM	CSR_PARERRM	OVFIM

Table 16 REM register.

10.1.8 RIVCS – Router Input VC Status

This register indicates the current status of a single input port of a router. Each register tracks the status of up to 4 virtual channels for the input port. There are 8 RIVCS per router, one for each router's input port.

Attribute: R

Bit field description:

- V_3[31] - 1'b1: Head flit valid (buffer ready) in VC 3
- F_3[30] - 1'b1: Buffer full in VC 3
- B_3[29]
  - 1'b1: Indicates that the head flit of the VC 3 is of the 'QoS Barrier' type
  - 1'b0: Indicates that the head flit of the VC 3 is of the 'QoS Normal' type
- S_3[28]
  - 1'b1: Indicates that the head flit is a start of packet. This also indicates that this input VC 3 has not yet acquired its corresponding output VC
  - 1'b0: Indicates that the head flit is not a start of packet. Also indicates that this input VC 3 has already acquired the VC on the output port
- UP_3[27]
  - 1'b1: Indicates that the flit accumulator on this VC 3 for upsizing to an output port is currently holding a flit
  - 1'b0: Indicates that either the upsizing accumulator is empty or there is no upsizing from the VC 3

- OUTP_3[26:24]
  - Value indicates the router output port to which the packet at the head of the VC3 is destined to: 3'd0:N, 3'd1:E, 3'd2:W, 3'd3:S, 3'd4:H, 3’d5:I, 3’d6:J, 3’d7:K
- V_2[23] - 1'b1: Head flit valid (buffer ready) in VC 2
- F_2[22] - 1'b1: Buffer full in VC 2
- B_2[21]
  - 1'b1: Indicates that the head flit of the VC 2 is of the 'QoS Barrier' type
  - 1'b0: Indicates that the head flit of the VC 2 is of the 'QoS Normal' type
- S_2[20]
  - 1'b1: Indicates that the head flit is a start of packet. This also indicates that this input VC 2 has not yet acquired its corresponding output VC
  - 1'b0: Indicates that the head flit is not a start of packet. Also indicates that this input VC 2 has already acquired the VC on the output port
- UP_2[19]
  - 1'b1: Indicates that the flit accumulator on this VC 2 for upsizing to an output port is currently holding a flit
  - 1'b0: Indicates that either the upsizing accumulator is empty or there is no upsizing from the VC 2
- OUTP_2[18:16]
  - Value indicates the router output port to which the packet at the head of the VC2 is destined to: 3'd0:N, 3'd1:E, 3'd2:W, 3'd3:S, 3'd4:H, 3’d5:I, 3’d6:J, 3’d7:K
- V_1[15] - 1'b1: Head flit valid (buffer ready) in VC 1
- F_1[14] - 1'b1: Buffer full in VC 1
- B_1[13]
  - 1'b1: Indicates that the head flit of the VC 1 is of the 'QoS Barrier' type
  - 1'b0: Indicates that the head flit of the VC 1 is of the 'QoS Normal' type
- S_1[12]
  - 1'b1: Indicates that the head flit is a start of packet. This also indicates that this input VC 1 has not yet acquired its corresponding output VC
  - 1'b0: Indicates that the head flit is not a start of packet. Also indicates that this input VC 1 has already acquired the VC on the output port
- UP_1[11]
  - 1'b1: Indicates that the flit accumulator on this VC 1 for upsizing to an output port is currently holding a flit
  - 1'b0: Indicates that either the upsizing accumulator is empty or there is no upsizing from the VC 1

- OUTP_1[10:8]
  - Value indicates the router output port to which the packet at the head of the VC1 is destined to: 3'd0:N, 3'd1:E, 3'd2:W, 3'd3:S, 3'd4:H, 3’d5:I, 3’d6:J, 3’d7:K
- V_0[7] - 1'b1: Head flit valid (buffer ready) in VC 0
- F_0[6] - 1'b1: Buffer full in VC 0
- B_0[5]
  - 1'b1: Indicates that the head flit of the VC 0 is of the 'QoS Barrier' type
  - 1'b0: Indicates that the head flit of the VC 0 is of the 'QoS Normal' type
- S_0[4]
  - 1'b1: Indicates that the head flit is a start of packet. This also indicates that this input VC 0 has not yet acquired its corresponding output VC
  - 1'b0: Indicates that the head flit is not a start of packet. Also indicates that this input VC 0 has already acquired the VC on the output port
- UP_0[3]
  - 1'b1: Indicates that the flit accumulator on this VC 0 for upsizing to an output port is currently holding a flit
  - 1'b0: Indicates that either the upsizing accumulator is empty or there is no upsizing from the VC 0
- OUTP_0[2:0]
  - Value indicates the router output port to which the packet at the head of the VC0 is destined to: 3'd0:N, 3'd1:E, 3'd2:W, 3'd3:S, 3'd4:H, 3’d5:I, 3’d6:J, 3’d7:K

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
V_3	F_3	B_3	S_3	UP_3	OUTP_3			V_2	F_2	B_2	S_2	UP_2	OUTP_2			V_1	F_1	B_1	S_1	UP_1	OUTP_1			V_0	F_0	B_0	S_0	UP_0	OUTP_0		

Table 17 RIVCS register.

10.1.9 ROEC – Router Output Event Counter

This register holds the output event counter. The value can be read to determine the current count value. The value can be written to initialize the counter. When events trigger a count, the counter will increment. When the counter increments at its highest value, it will roll over to zero and the overflow will mark the Router **OUTPUT** Event Interrupt Status register, which could trigger an interrupt.

Attribute: RW

Bit field description:

- EVENT_CNTR[31:0] - 32'bit event incrementing counter. Rollover from 32'hFFFFF -> 32'd0 sets the rollover status bit RE

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVENT_CNTR																															

Table 18 ROEC register.

10.1.10 ROECC – Router Output Event Counter Control

This register is used to select which hardware events will increment the event counter.

Attribute: RW

Bit field description:

- EVT[10:8] -
  - 100: Port stalled. Input flits are available for the port, but no output VC has credit
  - 011: Generates count event when flits are available to be sent to output VC, but the VC has no credit
  - 010: Generates count event on every flit sent on the selected output port and selected output VCs, this can be used to count total flits sent on a router output port
  - 001: Generates count event on every EOP sent on the selected output port and selected output VCs, this can be used to count packets sent on a router output port
  - 000: Disable
- OP[6:4] - Output port on which the event is captured
- OVC[3:0] - Bit map to select output VCs to monitor events on

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																					EVT			u		OP			OVC		

Table 19 ROECC register.

10.1.11 ROVCS – Router Output VC Status

This register indicates the current status of one of the output ports of a router. Each register tracks the status of up to 4 virtual channel for the output port. There are 8

ROVCS per router, one for each router's output port (only 5 are active registers, while the other 3 are reserved).

Attribute: R

Bit field description:

- rsv[27] - Reserved
- VB_3[26]
  - 1'b1: Indicates that this output VC 3 is currently locked to the corresponding VC on one of the input ports.
  - 1'b0: Indicates that this output VC 3 is free and can be acquired by the corresponding VC on one of the input port for the transmission of a packet.
- CE_3[25]
  - 1'b1: Indicates that this output VC 3 has no credit for transmission of flits to the downstream link.
  - 1'b0: Indicates that credit is available for transmission to downstream link.
- CF_3[24]
  - 1'b1: Indicates that the credit level with this VC 3 is at the maximum provisioned value.
- rsv[19] - Reserved
- VB_2[18]
  - 1'b1: Indicates that this output VC 2 is currently locked to the corresponding VC on one of the input ports.
  - 1'b0: Indicates that this output VC 2 is free and can be acquired by the corresponding VC on one of the input port for the transmission of a packet.
- CE_2[17]
  - 1'b1: Indicates that this output VC 2 has no credit for transmission of flits to the downstream link.
  - 1'b0: Indicates that credit is available for transmission to downstream link.
- CF_2[16]
  - 1'b1: Indicates that the credit level with this VC 2 is at the maximum provisioned value.
- rsv[11] - Reserved
- VB_1[10]
  - 1'b1: Indicates that this output VC 1 is currently locked to the corresponding VC on one of the input ports.

- 1'b0: Indicates that this output VC 1 is free and can be acquired by the corresponding VC on one of the input port for the transmission of a packet.

- CE_1[9]
  - 1'b1: Indicates that this output VC 1 has no credit for transmission of flits to the downstream link.
  - 1'b0: Indicates that credit is available for transmission to downstream link.
- CF_1[8] -
  1'b1: Indicates that the credit level with this VC 1 is at the maximum provisioned value.
- rsv[3] - Reserved
- VB_0[2]
  - 1'b1: Indicates that this output VC 0 is currently locked to the corresponding VC on one of the input ports.
  - 1'b0: Indicates that this output VC 0 is free and can be acquired by the corresponding VC on one of the input port for the transmission of a packet.
- CE_0[1]
  - 1'b1: Indicates that this output VC 0 has no credit for transmission of flits to the downstream link.
  - 1'b0: Indicates that credit is available for transmission to downstream link.
- CF_0[0]
  - 1'b1: Indicates that the credit level with this VC 0 is at the maximum provisioned value.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u				rsv	VB_3	CE_3	CF_3	u				rsv	VB_2	CE_2	CF_2	u				rsv	VB_1	CE_1	CF_1	u				rsv	VB_0	CE_0	CF_0

Table 20 ROVCS register.

10.2 STREAMING BRIDGE REGISTERS

10.2.1 BRPERR0 – Bridge Receive Parity Error 0

Receive bridge parity error status register monitoring parity errors on enabled layers from 0 to 7 (BRPERR0), and from 8 to 15 (BRPERR1). Parity/ECC error are monitored and captured for physical link to the bridge on each NoC layer. Following fields are monitored.

1. Data ECC/Parity: Parity/ECC is checked over multiple segments of data in each flit. An error in any segment will be recorded in the data ECC/parity error status bit. In ECC mode, single bit errors are corrected and the event is recorded.
2. User sideband ECC/parity: Similar to data field above.
3. Packet control parity: Parity over start of packet, end of packet, byte valid and data valid fields of a flit.

This register makes use of the logical layer mapping (and not the physical layer mapping). For the physical to logical table, please refer to the Physical to Logical Layer Mapping section in the help.

Attribute: WZC

Bit field description:

- PK0[4] - Parity error in packet delineation controls in layer 0
- SBC0[3] - Correctable single bit user sideband error (only ECC) in layer 0
- SB0[2] - User sideband ECC/parity error in layer 0
- DC0[1] - Correctable single bit data error (only ECC) in layer 0
- D0[0] - Data ECC/parity error in layer 0

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u								u								u								u							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u								u								u								u			PK0	SBC0	SB0	DC0	D0

Table 21 BRPERR0 register.

10.2.2 BRPERR1 – Bridge Receive Parity Error 1

Receive bridge parity error status register monitoring parity errors on enabled layers from 0 to 7 (BRPERR0), and from 8 to 15 (BRPERR1). Parity/ECC error are monitored and captured for physical link to the bridge on each NoC layer. Following fields are monitored.

1. Data ECC/Parity: Parity/ECC is checked over multiple segments of data in each flit. An error in any segment will be recorded in the data ECC/parity error status bit. In ECC mode, single bit errors are corrected and the event is recorded.
2. User sideband ECC/parity: Similar to data field above.
3. Packet control parity: Parity over start of packet, end of packet, byte valid and data valid fields of a flit.

This register makes use of the logical layer mapping (and not the physical layer mapping). For the physical to logical table, please refer to the Physical to Logical Layer Mapping section in the help.

Attribute: WZC

Bit field description:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u								u								u								u							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u								u								u								u							

Table 22 BRPERR1 register.

10.2.3 BRPERRM0 – Bridge Receive Parity Error Mask 0

Mask registers for receive bridge parity error interrupts from register BRPERR0 and BRPERR1. One mask register bit for each parity status bit in BRPERR. When mask bit is set, corresponding parity error does not cause an interrupt. Default state is reset for all mask bits, allowing interrupt on any parity error event.

This register makes use of the logical layer mapping (and not the physical layer mapping). For the physical to logical table, please refer to the Physical to Logical Layer Mapping section in the help.

Attribute: RW

Bit field description:

- PK0[4] - Mask Parity error in packet delineation controls in layer 0
- SBC0[3] - Mask Correctable single bit user sideband error (only ECC) in layer 0
- SB0[2] - Mask User sideband ECC/parity error in layer 0
- DC0[1] - Mask Correctable single bit data error (only ECC) in layer 0
- D0[0] - Mask Data ECC/parity error in layer 0

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u								u								u								u							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u								u								u								u			PK0	SBC0	SB0	DC0	D0

Table 23 BRPERRM0 register.

10.2.4 BRPERRM1 – Bridge Receive Parity Error Mask 1

Mask registers for receive bridge parity error interrupts from register BRPERR0 and BRPERR1. One mask register bit for each parity status bit in BRPERR. When mask bit is set, corresponding parity error does not cause an interrupt. Default state is reset for all mask bits, allowing interrupt on any parity error event.

This register makes use of the logical layer mapping (and not the physical layer mapping). For the physical to logical table, please refer to the Physical to Logical Layer Mapping section in the help.

Attribute: RW

Bit field description:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u								u								u								u							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u								u								u								u							

Table 24BRPERRM1 register.

10.2.5 BRS – Bridge Receive fifo Status

These registers track the status of the bridge's receive FIFOs from the NoC. Since there is up to 16 layers of the NoC, there are 16 registers. Each register tracks the status of one virtual channel, with up to 4 virtual channels per layer.

Attribute: R

Bit field description:

- F_3[29] - Buffer full for VC 3 Layer 0
- B_3[28] - Head flit barrier state for VC 3 Layer 0
- S_3[27] - Head flit sop for VC 3 Layer 0
- V_3[26] - Head flit (buffer ready) for VC 3 Layer 0
- OUTI_3[25:24] - Head flit output interface for VC 3 Layer 0
- F_2[21] - Buffer full for VC 2 Layer 0
- B_2[20] - Head flit barrier state for VC 2 Layer 0

- S_2[19] - Head flit sop for VC 2 Layer 0
- V_2[18] - Head flit (buffer ready) for VC 2 Layer 0
- OUTI_2[17:16] - Head flit output interface for VC 2 Layer 0
- F_1[13] - Buffer full for VC 1 Layer 0
- B_1[12] - Head flit barrier state for VC 1 Layer 0
- S_1[11] - Head flit sop for VC 1 Layer 0
- V_1[10] - Head flit (buffer ready) for VC 1 Layer 0
- OUTI_1[9:8] - Head flit output interface for VC 1 Layer 0
- F_0[5] - Buffer full for VC 0 Layer 0
- B_0[4] - Head flit barrier state for VC 0 Layer 0
- S_0[3] - Head flit sop for VC 0 Layer 0
- V_0[2] - Head flit (buffer ready) for VC 0 Layer 0
- OUTI_0[1:0] - Head flit output interface for VC 0 Layer 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u		F_3	B_3	S_3	V_3	OUTI_3		u		F_2	B_2	S_2	V_2	OUTI_2		u		F_1	B_1	S_1	V_1	OUTI_1		u		F_0	B_0	S_0	V_0	OUTI_0	

Table 25 BRS register.

10.2.6 BRUS – Bridge Receive Upsizer Status

This register tracks the status of the bridge receiver upsizer/downsize structure. It can be used with the other status registers to check for packets that are still occupying the bridge. Each of the host's receiving interfaces, up to 4, can have upsizing/downsizing logic, and this register tracks the status of all 4 interfaces.

Attribute: R

Bit field description:

- V_D[3] - Interface D upsizer/downsizer valid
- V_C[2] - Interface C upsizer/downsizer valid
- V_B[1] - Interface B upsizer/downsizer valid
- V_A[0] - Interface A upsizer/downsizer valid

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																												V_D	V_C	V_B	V_A

Table 26 BRUS register.

10.2.7 BTPERR – Bridge Transmit Parity Error

Transmit bridge parity error status register. One register bit per layer, to monitor error in credit return signals from the downstream port. Error status bits are sticky. First detected error while the status bit is in cleared state sets the bit. The bit needs to be explicitly cleared using zero write, before another error can be logged for that status bit.

Attribute: WZC

Bit field description:

- L15[15] - 1'b1: Credit parity error on layer 15
- L14[14] - 1'b1: Credit parity error on layer 14
- L13[13] - 1'b1: Credit parity error on layer 13
- L12[12] - 1'b1: Credit parity error on layer 12
- L11[11] - 1'b1: Credit parity error on layer 11
- L10[10] - 1'b1: Credit parity error on layer 10
- L9[9] - 1'b1: Credit parity error on layer 9
- L8[8] - 1'b1: Credit parity error on layer 8
- L7[7] - 1'b1: Credit parity error on layer 7
- L6[6] - 1'b1: Credit parity error on layer 6
- L5[5] - 1'b1: Credit parity error on layer 5
- L4[4] - 1'b1: Credit parity error on layer 4
- L3[3] - 1'b1: Credit parity error on layer 3
- L2[2] - 1'b1: Credit parity error on layer 2
- L1[1] - 1'b1: Credit parity error on layer 1
- L0[0] - 1'b1: Credit parity error on layer 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																L15	L14	L13	L12	L11	L10	L9	L8	L7	L6	L5	L4	L3	L2	L1	L0

Table 27 BTPERR register.

10.2.8 BTPERRM – Bridge Transmit Parity Error Mask

Mask register for transmit bridge parity error interrupts. One mask register bit for each parity status bit in BTPERR. When mask bit is set, corresponding parity error does not cause an interrupt. Default state is reset for all mask bits, allowing interrupt on any parity error event.

Attribute: RW

Bit field description:

- L15[15] - 1'b1: Interrupt Mask Credit parity error on layer 15
- L14[14] - 1'b1: Interrupt Mask Credit parity error on layer 14
- L13[13] - 1'b1: Interrupt Mask Credit parity error on layer 13
- L12[12] - 1'b1: Interrupt Mask Credit parity error on layer 12
- L11[11] - 1'b1: Interrupt Mask Credit parity error on layer 11
- L10[10] - 1'b1: Interrupt Mask Credit parity error on layer 10
- L9[9] - 1'b1: Interrupt Mask Credit parity error on layer 9
- L8[8] - 1'b1: Interrupt Mask Credit parity error on layer 8
- L7[7] - 1'b1: Interrupt Mask Credit parity error on layer 7
- L6[6] - 1'b1: Interrupt Mask Credit parity error on layer 6
- L5[5] - 1'b1: Interrupt Mask Credit parity error on layer 5
- L4[4] - 1'b1: Interrupt Mask Credit parity error on layer 4
- L3[3] - 1'b1: Interrupt Mask Credit parity error on layer 3
- L2[2] - 1'b1: Interrupt Mask Credit parity error on layer 2
- L1[1] - 1'b1: Interrupt Mask Credit parity error on layer 1
- L0[0] - 1'b1: Interrupt Mask Credit parity error on layer 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																L15	L14	L13	L12	L11	L10	L9	L8	L7	L6	L5	L4	L3	L2	L1	L0

Table 28 BTPERRM register.

10.2.9 BTRL – Bridge Transmit Rate Limiter

This is a register per host interface of Tx Bridge for QoS, used to control the rate of Traffic injection from host to the NoC.

Attribute: RW

Bit field description:

- EN[20]
  - 1'b1: Rate limit logic enable; rate limiter logic is used for arbitration only.
  - 1'b0: Rate limit logic disable

- CNT[19:16]
  - Max Count Value for Token. Anytime the token count is greater than zero, the host gets qualified to inject message into NoC.
- WT[11:0] - Starting Weight, for traffic issue to the NoC from the host interface.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u											EN	CNT				u				WT											

Table 29 BTRL register.

10.2.10 BTUS_0 – Bridge Transmit Upsizer Status 0

These two registers (BTUS_0 and BTUS_1) track the status of the bridge transmitter upsizer/downsize structure. They can be used with the other status registers to check for packets that are still occupying the bridge. Each NoC layer, up to 16, can have upsizing/downsizing logic, and these 2 registers track the status of all 16 layers (BTUS_0 from 0 to 7 and BTUS_1 from 8 to 15).

Attribute: R

Bit field description:

- L7_D[31] - Interface upsizer status for interface D, Layer 7
- L7_C[30] - Interface upsizer status for interface C, Layer 7
- L7_B[29] - Interface upsizer status for interface B, Layer 7
- L7_A[28] - Interface upsizer status for interface A, Layer 7
- L6_D[27] - Interface upsizer status for interface D, Layer 6
- L6_C[26] - Interface upsizer status for interface C, Layer 6
- L6_B[25] - Interface upsizer status for interface B, Layer 6
- L6_A[24] - Interface upsizer status for interface A, Layer 6
- L5_D[23] - Interface upsizer status for interface D, Layer 5
- L5_C[22] - Interface upsizer status for interface C, Layer 5
- L5_B[21] - Interface upsizer status for interface B, Layer 5
- L5_A[20] - Interface upsizer status for interface A, Layer 5
- L4_D[19] - Interface upsizer status for interface D, Layer 4
- L4_C[18] - Interface upsizer status for interface C, Layer 4
- L4_B[17] - Interface upsizer status for interface B, Layer 4
- L4_A[16] - Interface upsizer status for interface A, Layer 4
- L3_D[15] - Interface upsizer status for interface D, Layer 3
- L3_C[14] - Interface upsizer status for interface C, Layer 3
- L3_B[13] - Interface upsizer status for interface B, Layer 3

- L3_A[12] - Interface upsizer status for interface A, Layer 3
- L2_D[11] - Interface upsizer status for interface D, Layer 2
- L2_C[10] - Interface upsizer status for interface C, Layer 2
- L2_B[9] - Interface upsizer status for interface B, Layer 2
- L2_A[8] - Interface upsizer status for interface A, Layer 2
- L1_D[7] - Interface upsizer status for interface D, Layer 1
- L1_C[6] - Interface upsizer status for interface C, Layer 1
- L1_B[5] - Interface upsizer status for interface B, Layer 1
- L1_A[4] - Interface upsizer status for interface A, Layer 1
- L0_D[3] - Interface upsizer status for interface D, Layer 0
- L0_C[2] - Interface upsizer status for interface C, Layer 0
- L0_B[1] - Interface upsizer status for interface B, Layer 0
- L0_A[0] - Interface upsizer status for interface A, Layer 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L7_D	L7_C	L7_B	L7_A	L6_D	L6_C	L6_B	L6_A	L5_D	L5_C	L5_B	L5_A	L4_D	L4_C	L4_B	L4_A	L3_D	L3_C	L3_B	L3_A	L2_D	L2_C	L2_B	L2_A	L1_D	L1_C	L1_B	L1_A	L0_D	L0_C	L0_B	L0_A

Table 30 BTUS_0 register.

10.2.11 BTUS_1 - Bridge Transmit Upsizer Status 1

These two registers (BTUS_0 and BTUS_1) track the status of the bridge transmitter upsizer/downsize structure. They can be used with the other status registers to check for packets that are still occupying the bridge. Each NoC layer, up to 16, can have upsizing/downsizing logic, and these 2 registers track the status of all 16 layers (BTUS_0 from 0 to 7 and BTUS_1 from 8 to 15).

Attribute: R

Bit field description:

- L15_D[31] - Interface upsizer status for interface D, Layer 15
- L15_C[30] - Interface upsizer status for interface C, Layer 15
- L15_B[29] - Interface upsizer status for interface B, Layer 15
- L15_A[28] - Interface upsizer status for interface A, Layer 15
- L14_D[27] - Interface upsizer status for interface D, Layer 14
- L14_C[26] - Interface upsizer status for interface C, Layer 14
- L14_B[25] - Interface upsizer status for interface B, Layer 14
- L14_A[24] - Interface upsizer status for interface A, Layer 14
- L13_D[23] - Interface upsizer status for interface D, Layer 13

- L13_C[22] - Interface upsizer status for interface C, Layer 13
- L13_B[21] - Interface upsizer status for interface B, Layer 13
- L13_A[20] - Interface upsizer status for interface A, Layer 13
- L12_D[19] - Interface upsizer status for interface D, Layer 12
- L12_C[18] - Interface upsizer status for interface C, Layer 12
- L12_B[17] - Interface upsizer status for interface B, Layer 12
- L12_A[16] - Interface upsizer status for interface A, Layer 12
- L11_D[15] - Interface upsizer status for interface D, Layer 11
- L11_C[14] - Interface upsizer status for interface C, Layer 11
- L11_B[13] - Interface upsizer status for interface B, Layer 11
- L11_A[12] - Interface upsizer status for interface A, Layer 11
- L10_D[11] - Interface upsizer status for interface D, Layer 10
- L10_C[10] - Interface upsizer status for interface C, Layer 10
- L10_B[9] - Interface upsizer status for interface B, Layer 10
- L10_A[8] - Interface upsizer status for interface A, Layer 10
- L9_D[7] - Interface upsizer status for interface D, Layer 9
- L9_C[6] - Interface upsizer status for interface C, Layer 9
- L9_B[5] - Interface upsizer status for interface B, Layer 9
- L9_A[4] - Interface upsizer status for interface A, Layer 9
- L8_D[3] - Interface upsizer status for interface D, Layer 8
- L8_C[2] - Interface upsizer status for interface C, Layer 8
- L8_B[1] - Interface upsizer status for interface B, Layer 8
- L8_A[0] - Interface upsizer status for interface A, Layer 8

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L15_D	L15_C	L15_B	L15_A	L14_D	L14_C	L14_B	L14_A	L13_D	L13_C	L13_B	L13_A	L12_D	L12_C	L12_B	L12_A	L11_D	L11_C	L11_B	L11_A	L10_D	L10_C	L10_B	L10_A	L9_D	L9_C	L9_B	L9_A	L8_D	L8_C	L8_B	L8_A

Table 31 BTUS_1 register.

10.2.12 P – qos Profile

This register describes the weight value of each QoS supported at the bridge. Each byte of this register must be greater than or equal to 3. Each transmitting bridge supports up to 16 QoS profiles. Each QoS is composed of pri and weight, however only the weight is programmable, therefore is part of the registers.

QoS data is composed of four registers, P0, P1, P2 and P3, each of which contains the weight of four profiles. Depending upon how many QoS profiles are enabled, the appropriate bits in the following registers are available.

Attribute: RW

Bit field description:

- WT_QOS_3[31:24] - Weight of QoS profile 3
- WT_QOS_2[23:16] - Weight of QoS profile 2
- WT_QOS_1[15:8] - Weight of QoS profile 1
- WT_QOS_0[7:0] - Weight of QoS profile 0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WT_QOS_3								WT_QOS_2								WT_QOS_1								WT_QOS_0							

Table 85. P register.

10.2.13 RXE – Bridge Receive interrupt Event

This register tracks the interrupt events in the receive portion of the streaming bridge. It resets to 0, but as these conditions occur, the corresponding bits are set to 1. This register can be read and can also be cleared by sending a write with bits set to 0 for the bits that should be cleared.

There are four events that can signal an interrupt. If the host sends more credits than the streaming bridge can take, it will signal an interrupt to indicate a protocol violation has occurred. Each interface has its own status bit. These interrupts cannot be masked.

Attribute: WZC

Bit field description:

- EVC1_OFLW[6]
  - Event counter1 overflow. This event can be masked so that no interrupt is sent on an overflow condition.
- PARITY_ERR[5] - Register parity error interrupt
- EVC_OFLW[4]
  - Event counter overflow. This event can be masked so that no interrupt is sent on an overflow condition.
- CRC_OFLW_D[3] - Credit counter overflow for interface D

- CRC_OFLW_C[2] - Credit counter overflow for interface C
- CRC_OFLW_B[1] - Credit counter overflow for interface B
- CRC_OFLW_A[0] - Credit counter overflow for interface A

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																									EVC1_OFLW	PARITY_ERR	EVC_OFLW	CRC_OFLW_D	CRC_OFLW_C	CRC_OFLW_B	CRC_OFLW_A

Table 32 RXE register.

10.2.14 RXEM – Bridge Receive Event Mask

This register is used to decide which of the error/interrupt events specified in the Transmit Interrupt Status register should trigger an interrupt. Since only the events in bit 4 can be masked, only bit 4 is used in this register.

Attribute: RW

Bit field description:

- EVC1_OFLW_MASK[6]
  - 1'b1: When is set to 1, the corresponding interrupt event will not send an interrupt to the system.
  - 1'b0: The corresponding interrupt event will send an interrupt to the system.
- PARITY_ERR_MASK[5] - Interrupt mask for register parity error.
- EVC_OFLW_MASK[4]
  - 1'b1: When is set to 1, the corresponding interrupt event will not send an interrupt to the system.
  - 1'b0: The corresponding interrupt event will send an interrupt to the system.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																									EVC1_OFLW_MASK	PARITY_ERR_MASK	EVC_OFLW_MASK	u			

Table 33 RXEM register.

10.2.15 RXID – Receive bridge ID

This register holds a unique 8-bit identifier for the receiving bridge. It is a read-only register. It can be used for debugging software access to the NoC elements by confirming that a read has successfully targeted the correct NoC element.

Attribute: R

Bit field description:

- ZEROES[15:8] - Forced to zero
- ID[7:0]
  - A unique 8-bit identifier assigned to the bridge to uniquely identify it on the NoC. It is equal to the corresponding TXID 8-bit identifier on the Tx side of the bridge.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																ZEROES								ID							

Table 34 RXID register.

10.2.16 TXE – Transmit Event

This register tracks error or interrupt conditions. It resets to 0, but as these conditions occur, the corresponding bits are set to 1. This register can be read and can also be cleared by sending a write with bits set to 0 for the bits that should be cleared. This register works in combination with the Transmit Interrupt Mask register to determine when an interrupt is transmitted.

Attribute: WZC

Bit field description:

- PARITY_ERR[8] - Register parity error interrupt.
- FIFO_OVERFLOW_D[7]
  - Host interface FIFO D overflow. Indicates that one of the per-interface FIFOs at the transmitting bridge to NoC has overflowed. This event will always trigger an interrupt and cannot be masked
- FIFO_OVERFLOW_C[6]
  - Host interface FIFO C overflow. Indicates that one of the per-interface FIFOs at

the transmitting bridge to NoC has overflowed. This event will always trigger an interrupt and cannot be masked

- FIFO_OVERFLOW_B[5]
  - Host interface FIFO B overflow. Indicates that one of the per-interface FIFOs at the transmitting bridge to NoC has overflowed. This event will always trigger an interrupt and cannot be masked
- FIFO_OVERFLOW_A[4]
  - Host interface FIFO A overflow. Indicates that one of the per-interface FIFOs at the transmitting bridge to NoC has overflowed. This event will always trigger an interrupt and cannot be masked
- EVENT_CNTR_OVERFLOW[3]
  - 1'b1: Sets if the event counter overflows, this event can be masked so that no interrupt is sent on an overflow condition
- TRANS_ILLEGAL_DEST_QOS[2]
  - 1'b1: Sets if a transaction is received from bridge for which there is no entry present in the vcmap, i.e. the destination and/or QoS is not supported, this is a decode error. This event can be masked to not send an interrupt, but the packet will be dropped in the bridge.
- SOP_AFTER_SOP[1]
  - 1'b1: Sets if a SOP is received after SOP, this event will always trigger an interrupt and cannot be masked.
- TRANS_WITHOUT_SOP[0]
  - 1'b1: Sets if a transaction is initiated w/o SOP, this event will always trigger an interrupt and cannot be masked.

31:9	8	7	6	5	4	3	2	1	0
u	PARITY_ERR	FIFO_OVERFLOW_D	FIFO_OVERFLOW_C	FIFO_OVERFLOW_B	FIFO_OVERFLOW_A	EVENT_CNTR_OVERFLOW	TRANS_ILLEGAL_DEST_QOS	SOP_AFTER_SOP	TRANS_WITHOUT_SOP

Table 35 TXE register.

10.2.17 TXEM – Transmit Event Mask

This register is used to decide which of the error/interrupt events specified in the Transmit Interrupt Status register should trigger an interrupt. Since only the events in bit 2 and 3 can be masked, only bit 2 and 3 are used in this register. When one of the bits in this register is set to 1, the corresponding interrupt event will not send an interrupt to the system.

Attribute: RW

Bit field description:

- PARITY_ERR_MASK[8] - Interrupt mask for register parity error
- EVENT_CNTR_OVERFLOW[3] - Interrupt mask for event counter overflow
- TRANS_ILLEGAL_DEST_QOS[2]- Interrupt mask for illegal destination QoS

31:9	8	7:4	3	2	1:0
u	PARITY_ERR_MASK	u	EVENT_CNTR_OVERFLOW	TRANS_ILLEGAL_DEST_QOS	u

Table 36 TXEM register.

10.2.18 TXID – Transmit bridge ID

This register holds a unique 8-bit identifier for the transmitting bridge. It is a read-only register. It can be used for debugging software access to the NoC elements by confirming that a read has successfully targeted the correct NoC element.

Attribute: R

Bit field description:

- ZEROES[15:8] - Forced to zero
- ID[7:0] - A unique 8-bit identifier assigned to the bridge to uniquely identify it on the NoC. It is equal to the corresponding RXID 8-bit identifier on the Rx side of the bridge.

31:16	15:8	7:0
u	ZEROES	ID

Table 37 TXID register.

10.3 AMBA REGISTERS

Master Bridge

10.3.1 AM_ADBASE

These registers specify the base addresses and masks of different slave ranges accessible from this master. One base, mask, and reloc register set per address range assigned to

the master. These registers can be individually designated as read-only or read-write based on NocStudio property assigned to address ranges.

Even if the register is read-only, the range can be disabled using the appropriate bits described below which are always programmable. A slave address range is specified using the above base address and mask pair. An address on the AR or AW channel has a match against a range if it satisfies the equation

AxADDRS & AM_ADMASK[i] == AM_ADBASE[i]

Note that programmed 'base' must already factor the 'mask'. The base should not have a 1'b1 bit where the corresponding mask bit is 1'b0. What this means is that the programmed base should already have performed a bit-wise AND operation with the 'mask'. An address which doesn't match any range results in a decode error response. Note that programming of these registers must ensure that an address matches only against one range. Match against multiple ranges is a fatal error and will raise an interrupt.

Address ranges are specified at 64B cache line boundary. Lower six bits if AM_ADBASE and AM_ADMASK are used for specifying access permissions on an address range.

AM_ADBASE[5:0]

5	4	3	2	1	0
X	DI	R/Wn	I	NS	P

AM_ADMASK[5:0]

5	4	3	2	1	0
X	X	Valid	I	NS	P

Bits [2:0] act as value and mask for checking against AxPROT of an incoming command. A command is allowed access to a range if

AxPROT & AM_ADMASK[2:0] == AM_ADBASE[2:0] & AM_ADMASK[2:0]

If the above check fails, then the command is denied access to the range and decode error response is returned. The encoding is specified below.

AM_ADBASE[4] Disable	AM_ADMASK[3] RD/WRn valid	AM_ADBASE[3] RD/WRn	Interpretation
1	X	X	range disabled
0	1	1	Read only
0	1	0	Write only
0	0	X	read/write

Attribute: RW

Bit field description:

- BASE_ADDRESS[63:6] - Base address
- LLC[5] - LLC disable
- DI[4] - 1'b1: Address range disabled
- R_Wn[3]
  - 1'b1: Read enabled to range
  - 1'b0: Write enabled to range
- I[2] - 1'b1: Instruction
- NS[1] - 1'b1: Non-secure
- P[0] - 1'b1: Privileged

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
BASE_ADDRESS																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BASE_ADDRESS																										LLC	DI	R_Wn	I	NS	P

Table 38 AM_ADBASE register.

10.3.2 AM_ADMASK

See AM_ADBASE.

Attribute: RW

Bit field description:

- MASK[63:6] - Mask
- RSV[5:4] - Reserved
- VAL[3] - 1'b1: R_Wn field is valid

- I[2] - 1'b1: Instruction field is valid
- NS[1] - 1'b1: Non-secure field is valid
- P[0] - 1'b1: Privileged field is valid

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
MASK																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MASK																										RSV		VAL	I	NS	P

Table 39 AM_ADMASK register.

10.3.3 AM_ADRELOCSLV

Register used to relocate a master address to slave address.

Attribute: R

Bit field description:

- **RELOC**[31:6] - Reloc

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RELOC																										u					

Table 40 AM_ADRELOCSLV register.

10.3.4 AM_ADRELOCSYS

Register used to relocate a master address to system address.

Attribute: R

Bit field description:

- **RELOC**[31:6] - Reloc

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RELOC																										u					

Table 41 AM_ADRELOCSYS register.

10.3.5 AM_AROVRD

AR override.

Attribute: RW

Bit field description:

- arqos_enb[23:20]
  - 1'b1 indicates bit positions where ARQOS value is overridden
  - 1'b0 indicates bit positions where ARQOS is unchanged.
- arqos_val[19:16] - Value to override incoming ARQOS
- arprot_enb[14:12]
  - 1'b1 indicates bit positions where ARPROT value is overridden
  - 1'b0 indicates bit positions where ARPROT is unchanged.
- arprot_val[10:8] - Value to override incoming ARPROT
- arcache_enb[7:4]
  - 1'b1 indicates bit positions where ARCACHE value is overridden.
  - 1'b0 indicates bit positions where ARCACHE is unchanged.
- arcache_val[3:0] - Value to override incoming ARCACHE

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u								arqos_enb			arqos_val	u				arprot_enb			u		arprot_val			arcache_enb				arcache_val			

Table 42 AM_AROVRD register.

10.3.6 AM_AWOVRD

AW override.

Attribute: RW

Bit field description:

- awqos_enb[23:20]
  - 1'b1 indicates bit positions where AWQOS value is overridden
  - 1'b0 indicates bit positions where AWQOS is unchanged.
- awqos_val[19:16] - Value to override incoming AWQOS
- awprot_enb[14:12]
  - 1'b1 indicates bit positions where AWPROT value is overridden
  - 1'b0 indicates bit positions where AWPROT is unchanged.
- awprot_val[10:8] - Value to override incoming AWPROT
- awcache_enb[7:4]
  - 1'b1 indicates bit positions where AWCACHE value is overridden
  - 1'b0 indicates bit positions where AWCACHE is unchanged.
- awcache_val[3:0] - Value to override incoming ARCACHE

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u								awqos_enb			awqos_val				u	awprot_enb				u	awprot_val			awcache_enb				awcache_val			

Table 43 AM_AWOVRD register.

10.3.7 AM_BRIDGE_ID

Unique identifier assigned to the master bridge.

Attribute: R

Bit field description:

- ZEROES[15:8] - Forced to zero
- ID[7:0] - Unique bridge ID

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																ZEROES								ID							

Table 44 AM_BRIDGE_ID register.

10.3.8 AM_CADDR

This register is part of statistics gathering on the AR and AW command channels. This is the address value which is checked against AR, AW command channels in conjunction with the mask below to filter commands for statistics gathering.

Attribute: RW

Bit field description:

- CAPTURE_ADDR[63:0] - Capture address

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
CAPTURE_ADDR																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAPTURE_ADDR																															

Table 45 AM_CADDR register.

10.3.9 AM_CADDRMSK

If command address on the AR, AW channel logically ANDed with this mask is equal to the value specified in AM_CADDR, then an address match has occurred. Note that only lowest significant bits equal to the master's address width are used in the comparison.

Attribute: RW

Bit field description:

- CAPTURE_ADDR_MASK[63:0] - Capture address mask

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
CAPTURE_ADDR_MASK																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CAPTURE_ADDR_MASK																															

Table 46 AM_CADDRMSK register.

10.3.10 AM_CCMD0

Values of command fields/pins that are compared against AR, AW, R, W channel interface signals to filter commands/events for statistics gathering. Two selections can be made for statistics gathering, counting filtered commands or measuring latency of filtered commands.

Attribute: RW

Bit field description:

- TYP[32]
  - 1'b1: Count captured command
  - 1'b0: Count response latency of captured command
- INTFID[30:28]
  - 000: AR (can count captured event or response latency)
  - 001: AW (can count captured event or response latency)
- VAL[25] - 1'b1: Valid
- RDY[24] - 1'b1: Ready
- LOC[23] - 1'b1: Lock
- PROT[22:20] - Prot
- QOS[19:16] - QoS
- CACHE[15:12] - Cache
- BAR[9:8] - Bar
- DOMAIN[5:4] - Domain
- SNOOP[3:0] - Snoop

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															TYP
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u	INTFID		u	VAL	RDY	LOC	PROT		QOS		CACHE		u	BAR		u	DOMAIN		SNOOP												

Table 47 AM_CCMD0 register.

10.3.11 AM_CCMD1

Values of command fields/pins that are compared against AR, AW, R, W channel interface signals to filter commands/events for statistics gathering. Two selections can be made for statistics gathering, counting filtered commands or measuring latency of filtered commands.

Attribute: RW

Bit field description:

- TYP[32]
  - 1'b1: Count captured command
  - 1'b0: Count response latency of captured command
- INTFID[30:28]
  - 000: AR (can count captured event or response latency)
  - 001: AW (can count captured event or response latency)
- VAL[25] - 1'b1: Valid
- RDY[24] - 1'b1: Ready
- LOC[23] - 1'b1: Lock
- PROT[22:20] - Prot
- QOS[19:16] - QoS
- CACHE[15:12] - Cache
- BAR[9:8] - Bar
- DOMAIN[5:4] - Domain
- SNOOP[3:0] - Snoop

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															TYP
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u	INTFID		u	VAL	RDY	LOC	PROT		QOS		CACHE		u	BAR		u	DOMAIN		SNOOP												

Table 48 AM_CCMD1 register.

10.3.12 AM_CCMDMSK0

If Command fields on AR, AW channel logically ANDed with this mask are equal to the corresponding command field values in AM_CCMD0 then a command match has occurred. Address and command value match occurring together constitute events for the statistics counters.

Attribute: RW

Bit field description:

- VAL[25] - 1'b1: Valid
- RDY[24] - 1'b1: Ready
- LOC[23] - 1'b1: Lock

- PROT[22:20] - Prot
- QOS[19:16] - QoS
- CACHE[15:12] - Cache
- BAR[9:8] - Bar
- DOMAIN[5:4] - Domain
- SNOOP[3:0] - Snoop

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u						VAL	RDY	LOC	PROT			QOS			CACHE			u		BAR		u		DOMAIN		SNOOP				

Table 87. AM_CCMDMSK0 register.

10.3.13 AM_CCMDMSK1

If Command fields on AR, AW channel logically ANDed with this mask are equal to the corresponding command field values in AM_CCMD0 then a command match has occurred. Address and command value match occurring together constitute events for the statistics counters.

Attribute: RW

Bit field description:

- VAL[25] - 1'b1: Valid
- RDY[24] - 1'b1: Ready
- LOC[23] - 1'b1: Lock
- PROT[22:20] - Prot
- QOS[19:16] - QoS
- CACHE[15:12] - Cache
- BAR[9:8] - Bar
- DOMAIN[5:4] - Domain
- SNOOP[3:0] - Snoop

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u						VAL	RDY	LOC	PROT			QOS			CACHE			u		BAR		u		DOMAIN		SNOOP				

Table 49 AM_CCMDMSK1 register.

10.3.14 AM_CFG

Configures the master bridge's support for autowake of power domains.

When set, master bridge halts a request and issues wakeup requests for power domains that need to powered up to complete the transaction. The power domains should support auto wake. When reset, master bridge issues DECERR for any transaction which has dependent power domains in sleep state.

Attribute: RW

Bit field description:

- AW[0] - 1'b1: Autowake enabled
  1'b0: Autowake disabled

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																														AW	

Table 50 AM_CFG register.

10.3.15 AM_CNTR0

32-bit counter which is used to count the captured statistics events. This counter can hold the count of commands filtered on the AR, AW channels. When measuring command latency, this counter holds the denominator or sum of number of cycles between command and response for multiple commands over which latency is measured.

Attribute: RW

Bit field description:

- CNTR[31:0] - Counter

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNTR																															

Table 51 AM_CNTR0 register.

10.3.16 AM_CNTR1

32-bit counter which is used to count the captured statistics events. This counter can hold the count of commands filtered on the AR, AW channels. When measuring command latency, this counter holds the denominator or sum of number of cycles between command and response for multiple commands over which latency is measured.

Attribute: RW

Bit field description:

- CNTR[31:0] - Counter

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNTR																															

Table 52 AM_CNTR1 register.

10.3.17 AM_ERA

This is the address on AR channel for which a decode error was detected. This corresponds to the status register bit e0 in AM_ERR.

Attribute: R

Bit field description:

- READ_DECERR_ADDRS[63:0] - Read decerr address

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
READ_DECERR_ADDRS																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
READ_DECERR_ADDRS																															

Table 53 AM_ERA register.

10.3.18 AM_ERR

These error status bits record the first error event and have to be cleared by writing a 1'b0 before new errors are recorded.

Attribute: WZC

Bit field description:

- E46[46] - 1'b1: CDDATA Parity Err
- E45[45] - 1'b1: WDATA Parity Err
- E44[44] - 1'b1: AWADDR Parity Err
- E43[43] - 1'b1: AW Parity Err
- E42[42] - 1'b1: ARADDR Parity Err
- E41[41] - 1'b1: AR Parity Err
- E40[40] - 1'b1: Indicates that portcheck detected error (SIB mode only)
- E35[35] - 1'b1: Parity error in configuration/status registers
- E34[34] - 1'b1: Traffic sent to a noc layer which is power gate
- E33[33] - 1'b1: Capture counter1 overflow
- E32[32] - 1'b1: Capture counter0 overflow
- E24[24] - 1'b1: Unexpected narrow write detected
- E23[23] - 1'b1: Write WRAP not equal to supported cacheline size
- E22[22] - 1'b1: Write response timeout
- E21[21] - 1'b1: Write address multi-hit
- E20[20] - 1'b1: Write exclusive split
- E19[19] - 1'b1: Non-modifiable WRAP
- E18[18] - 1'b1: Write slave error
- E17[17] - 1'b1: Write address decode error from slave
- E16[16] - 1'b1: Local write address decode error
- E8[8] - 1'b1: Unexpected narrow read detected
- E7[7] - 1'b1: Read WRAP not equal to supported cacheline size: A WRAP command of unsupported cache line size was detected
- E6[6] - 1'b1: Read response timeout: Read response timeout occurred. With timeout enabled, a response wasn't received within the expected interval
- E5[5] - 1'b1: Read address multi-hit: An AR command matched against multiple entries in the address table
- E4[4] - 1'b1: Read exclusive split: An AR command of FIXED burst type was detected
- E3[3] - 1'b1: Non-modifiable WRAP: A WRAP command marked as non-modifiable (ARCACHE[0]=0) was detected

- E2[2] - 1'b1: Read slave error: A slave error response was received from a slave device
- E1[1] - 1'b1: Read address decode error from slave: A decode error response was received from a slave device
- E0[0] - 1'b1: Local read address decode error: ARADDR did not find a match in the master bridges address table and a decode error was issued

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																	E46	E45	E44	E43	E42	E41	E40	u				E35	E34	E33	E32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u							E24	E23	E22	E21	E20	E19	E18	E17	E16	u							E8	E7	E6	E5	E4	E3	E2	E1	E0

Table 54 AM_ERR register.

10.3.19 AM_EWA

This is the address on AW channel for which a decode error was detected. This corresponds to the status register bit e16 in AM_ERR.

Attribute: R

Bit field description:

- WRITE_DECERR_ADDRS[63:0] - Write decerr address

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
WRITE_DECERR_ADDRS																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WRITE_DECERR_ADDRS																															

Table 55 AM_EWA register.

10.3.20 AM_HASH_FUNC

These registers are used for programmable hash functions. They are the size of the SYSTEM address width, minus the 6-bit offset bits. Any reprogramming of these values can require the slave to understand the new hashing function in order to successfully compress the address space.

Attribute: RW

Bit field description:

- HASH[31:6] - Hash bits
- F[0]
  - 1'b1: Force hash bit. When hash function register is programmed to 0, this will cause the corresponding hash bit to be forced to 1'b1. A non-zero hash function will cause inversion of the corresponding hash bit.
  - 1'b0: Default. When hash function register is programmed to 0, the corresponding hash bit will be 1'b0

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HASH																										u					F

Table 56 AM_HASH_FUNC register.

10.3.21 AM_INTM

Interrupt mask register. Individual bit positions match the error bit positions in AM_ERR. When an INTM bit is set, occurrence of the corresponding error event will not cause an interrupt to be raised. When 1'b0, error event will cause interrupt to be raised.

Attribute: RW

Bit field description:

- M46[46] - 1'b1: CDDATA Parity Intr Mask
- M45[45] - 1'b1: WDATA Parity Intr Mask
- M44[44] - 1'b1: AWADDR Parity Intr Mask
- M43[43] - 1'b1: AW Parity Intr Mask
- M42[42] - 1'b1: ARADDR Parity Intr Mask
- M41[41] - 1'b1: AR Parity Intr Mask
- M40[40] - 1'b1: Mask interrupt for SIB portcheck error (SIB mode only)
- M35[35] - 1'b1: Mask interrupt on csr parity errors
- M34[34] - 1'b1: Mask interrupt on traffic to PG layer
- M33[33] - 1'b1: Counter 1 overflow interrupt mask
- M32[32] - 1'b1: Counter 0 overflow interrupt mask
- M24[24] - 1'b1: Mask interrupt for write channel
- M23[23] - 1'b1: Mask interrupt for write channel
- M22[22] - 1'b1: Mask interrupt for write channel
- M21[21] - 1'b1: Mask interrupt for write channel

- M20[20] - 1'b1: Mask interrupt for write channel
- M19[19] - 1'b1: Mask interrupt for write channel
- M18[18] - 1'b1: Mask interrupt for write channel
- M17[17] - 1'b1: Mask interrupt for write channel
- M16[16] - 1'b1: Mask interrupt for write channel
- M8[8] - 1'b1: Mask interrupt for read channel
- M7[7] - 1'b1: Mask interrupt for read channel
- M6[6] - 1'b1: Mask interrupt for read channel
- M5[5] - 1'b1: Mask interrupt for read channel
- M4[4] - 1'b1: Mask interrupt for read channel
- M3[3] - 1'b1: Mask interrupt for read channel
- M2[2] - 1'b1: Mask interrupt for read channel
- M1[1] - 1'b1: Mask interrupt for read channel
- M0[0] - 1'b1: Mask interrupt for read channel

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																	M46	M45	M44	M43	M42	M41	M40	u				M35	M34	M33	M32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u							M24	M23	M22	M21	M20	M19	M18	M17	M16	u							M8	M7	M6	M5	M4	M3	M2	M1	M0

Table 57 AM_INTM register.

10.3.22 AM_LATNUM0

This register is programmed with the number of commands over which latency is to be measured. When this register counts down to 0, latency measurement is complete and average latency can be computed using:

Average command latency = Value in AM_CNTR0/Value which was programmed in AM_LATNUM0

There are two sets of counters available for gathering statistics. AM_CCMD1, AM_CCMDMSK1, AM_CNTR1, AM_LATNUM1 constitute the second bank of counters and are similar to the above set.

Attribute: RW

Bit field description:

- CNTR[31:0] - Counter

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNTR																															

Table 58 AM_LATNUM0 register.

10.3.23 AM_LATNUM1

This register is programmed with the number of commands over which latency is to be measured. When this register counts down to 0, latency measurement is complete and average latency can be computed using:

Average command latency = Value in AM_CNTR0/Value which was programmed in AM_LATNUM0

There are two sets of counters available for gathering statistics. AM_CCMD1, AM_CCMDMSK1, AM_CNTR1, AM_LATNUM1 constitute the second bank of counters and are similar to the above set.

Attribute: RW

Bit field description:

- CNTR[31:0] - Counter

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNTR																															

Table 59 AM_LATNUM1 register.

10.3.24 AM_OSSLV

This register is used to check if there are any outstanding read/write commands to a slave specified by field slvid. NocStudio provides a table of slvids corresponding to the slave ports accessible from a master bridge. Outstanding status is reflected in AM_STS.

Attribute: RW

Bit field description:

- SLVID[15:0] - A slave ID associated with the current master for command outstanding status

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																SLVID															

Table 60 AM_OSSLV register.

10.3.25 AM_STS

When reordering is disabled on the master bridge, hazard stall occurs if the master tries to access a new slave device while response from a different slave is outstanding on the same AID.

This is because the responses can arrive out of order and the bridge is not equipped to correct the order. Without re-order buffers, hazard stalls also occur if a new large command needs to be split while there are older commands outstanding, or a large command just finished sending all its split segments but all responses have not returned yet.

When reordering is enabled, stall due to hazard occurs if a new command arrives, whose NoC QoS is different from the NoC QoS of commands outstanding on that AID.

Attribute: R

Bit field description:

- AWO[7]
  - 1'b1: Write commands are outstanding to the slave specified in OSSLV register
- ARO[6]
  - 1'b1: Read commands are outstanding to the slave specified in OSSLV register
- AWS[5] - 1'b1: AW channel is stalled on hazard
- ARS[4] - 1'b1: AR channel is stalled on hazard
- WOE[3] - 1'b1: Write Outstanding queue Empty
- ROE[2] - 1'b1: Read Outstanding queue Empty
- WOF[1]
  - 1'b1: Maximum supported number of write commands are outstanding waiting

for response and no more requests can be accepted
- 1'b0: Master bridge can accept more write requests

- ROF[0]
  - 1'b1: Maximum supported number of read commands are outstanding waiting for response and no more requests can be accepted
  - 1'b0: Master bridge can accept more read requests

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																								AWO	ARO	AWS	ARS	WOE	ROE	WOF	ROF

Table 61 AM_STS register.

10.3.26 AM_TOCFG

This register is used to configure response timeouts.

AM_TOCFG[8] (En) needs to be set for timeout tracking to be enabled. When this bit is 1'b0, no timestamps are recorded to generate timeout interrupts. A 64-bit free running counter is used to time the response interval.

AM_TOCFG[5:0] (TI) specifies the lower bit index into this counter, from where 2-bits are picked up and recorded as the arrival time stamp of every incoming AR and AW command. If response for a command does not return before the current time stamp rolls to arrival time stamp minus 1, the response is assumed to have timed-out and an interrupt is raised along with the slave ID to which the timed-out request was sent.

When changing the TI field, first write to the register with the En field cleared, then write a second time with the TI field to its new value, then a 3rd write to restore the En field to Enabled. During this update while the En field is cleared, existing timers will be cancelled, and new timer starts will be inhibited.

Attribute: RW

Bit field description:

- EN[8]
  - 1'b1: Enabled timeout tracking, a 64-bit free running counter is used to time the response interval.
  - 1'b0: No timestamps are recorded to generate timeout interrupts

- TI[5:0] - Timer index, index of a 64-bit counter from where timestamp is picked

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																							EN	u		TI					

Table 62 AM_TOCFG register.

10.3.27 AM_TOSLVID

AR slvid and AW slvid fields indicate slave IDs to which a read, write response timeout was detected.

Note that slvid encoding is not same as the bridge ID of the slave. NocStudio provides a table mapping the slvids to the actual slave ports accessible from the master bridge.

Attribute: R

Bit field description:

- AW_SLVID[31:16] - Slave ID of timed out AW request
- AR_SLVID[15:0] - Slave ID of timed out AR request

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AW_SLVID																AR_SLVID															

Table 63 AM_TOSLVID register.

10.3.28 AS_AROVRD

AR Overrides.

Attribute: RW

Bit field description:

- arqos_enb[23:20]
  - 1'b1 indicates bit positions where ARQOS value is overridden
  - 1'b0 indicates bit positions where ARQOS is unchanged.
- arqos_val[19:16] - Value to override incoming ARQOS
- arprot_enb[14:12]
  - 1'b1 indicates bit positions where ARPROT value is overridden
  - 1'b0 indicates bit positions where ARPROT is unchanged.
- arprot_val[10:8] - Value to override incoming ARPROT
- arcache_enb[7:4]
  - 1'b1 indicates bit positions where ARCACHE value is overridden
  - 1'b0 indicates bit positions where ARCACHE is unchanged.
- arcache_val[3:0] - Value to override incoming ARCACHE

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u								arqos_enb				arqos_val				u	arprot_enb			u	arprot_val				arcache_enb				arcache_val		

Table 64 AS_AROVRD register.

10.3.29 AS_AWOVRD

AW Overrides.

Attribute: RW

Bit field description:

- awqos_enb[23:20]
  - 1'b1 indicates bit positions where AWQOS value is overridden.
  - 1'b0 indicates bit positions where AWQOS is unchanged.
- awqos_val[19:16] - Value to override incoming AWQOS
- awprot_enb[14:12]
  - 1'b1 indicates bit positions where AWPROT value is overridden
  - 1'b0 indicates bit positions where AWPROT is unchanged.
- awprot_val[10:8] - Value to override incoming AWPROT
- awcache_enb[7:4]
  - 1'b1 indicates bit positions where AWCACHE value is overridden.
  - 1'b0 indicates bit positions where AWCACHE is unchanged.
- awcache_val[3:0] - Value to override incoming AWCACHE

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u								awqos_enb		awqos_val			u			awprot_enb				u		awprot_val			awcache_enb			awcache_val			

Table 65 AS_AWOVRD register.

10.3.30 AS_BRIDGE_ID

Unique identifier assigned to the slave bridge.

Attribute: R

Bit field description:

- ZEROES[15:8] - Forced to zero
- ID[7:0] - Unique bridge ID

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																ZEROES								ID							

Table 66 AS_BRIDGE_ID register.

10.3.31 AS_CCMD

Not applicable for current release.

Attribute: RW

Bit field description:

- UNK[63:0] - UNK

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNK																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UNK																															

Table 67 AS_CCMD register.

10.3.32 AS_CCMDMSK

Not applicable for current release.

Attribute: RW

Bit field description:

- UNK[63:0] - UNK

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
UNK																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
UNK																															

Table 68 AS_CCMDMSK register.

10.3.33 AS_CNTR

Not applicable for current release.

Attribute: R

Bit field description:

- CNTR[31:0] - Counter

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNTR																															

Table 69 AS_CNTR register.

10.3.34 AS_ERR

These error status bits record the first error event and have to be cleared by writing a 1'b0 before new errors are recorded.

Attribute: WZC

Bit field description:

- E38[38] - 1'b1: ACADDR Parity error
- E37[37] - 1'b1: AC Parity error
- E36[36] - 1'b1: BRESP Parity error
- E35[35] - 1'b1: RRESP Parity error
- E34[34] - 1'b1: RDATA Parity error
- E33[33] - 1'b1: Parity error in config/status registers
- E32[32] - 1'b1: Traffic sent to a noc layer which is power gated
- E19[19] - 1'b1: Write command modified: A write command which was marked as non-modifiable was modified by the slave bridge
- E18[18] - 1'b1: Unknown write response destination: BID from write response produces a destination which is not present in the routing table
- E17[17] - 1'b1: Write slave error response: Slave error response received from slave device for write command
- E16[16] - 1'b1: Write decode error response: Decode error response received from slave device for write command
- E4[4] - 1'b1: Read command modified: A read command which was marked as non-modifiable was modified by the slave bridge
- E3[3] - 1'b1: Interleaved read response: Interleaved read response. This can occur if interleaved read response is received from a slave device for which a de-interleaver was not specified
- E2[2] - 1'b1: Unknown read response destination: RID from read response produces a destination which is not present in the routing table
- E1[1] - 1'b1: Read slave error response: Slave error response received from slave device for read command
- E0[0] - 1'b1: Read decode error response: Decode error response received from slave device for read command

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																									E38	E37	E36	E35	E34	E33	E32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u												E19	E18	E17	E16	u											E4	E3	E2	E1	E0

Table 70 AS_ERR register.

10.3.35 AS_INTM

Interrupt mask register. Individual bit position matches the error bit positions in AS_ERR. When an INTM bit is set, occurrence of the corresponding error event will not cause an interrupt to be raised. When 1'b0, error event will cause interrupt to be raised.

Attribute: RW

Bit field description:

- M38[38] - ACADDR parity interrupt Mask
- M37[37] - AC parity interrupt Mask
- M36[36] - BRESP parity interrupt Mask
- M35[35] - RRESP parity interrupt Mask
- M34[34] - RDATA parity interrupt Mask
- M33[33] - Mask interrupt on csr parity errors
- M32[32] - Mask interrupt on traffic to PG layer
- M19[19] - Mask interrupts for write channel
- M18[18] - Mask interrupts for write channel
- M17[17] - Mask interrupts for write channel
- M16[16] - Mask interrupts for write channel
- M4[4] - Mask interrupts for read channel
- M3[3] - Mask interrupts for read channel
- M2[2] - Mask interrupts for read channel
- M1[1] - Mask interrupts for read channel
- M0[0] - Mask interrupts for read channel

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																									M38	M37	M36	M35	M34	M33	M32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u												M19	M18	M17	M16	u									M4	M3	M2	M1	M0		

Table 71 AS_INTM register.

10.3.36 AS_STS

Slave bridge status bits.

Attribute: R

Bit field description:

- ROE[3] - 1'b1: There are no read commands outstanding to the attached slave device
- WOE[2] - 1'b1: There are no write commands outstanding to the attached slave device
- ROF[1] - 1'b1: Maximum number of supported read commands are outstanding to the attached slave device awaiting response, no more read

commands will be issued to slave till responses are received.
- 1'b0: Slave bridge can accept more read commands from the NoC

- WOF[0] - 1'b1: Maximum number of supported write commands are outstanding to the attached slave device awaiting response, no more write commands will be issued to slave till responses are received.
- 1'b0: Slave device can expect more write commands from NoC

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																												ROE	WOE	ROF	WOF

Table 72 AS_STS register.

10.4 REGBUS REGISTERS

10.4.1 AM_NOCVER_ID

Version identifier for the NoC. This read-only register is available only on the regbus master. This register is not available on pother master bridges and access will result in decode error response.

Attribute: R

Bit field description:

- **NOC_VERSION_ID**[31:0] - NoC version ID

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOC_VERSION_ID																															

Table 73 AM_NOCVER_ID register.

10.5 PROTOCOL CONVERTER REGISTERS

AHB Bridge

10.5.1 AHBM_CTL

Control Register.

Attribute: RW

Bit field description:

- WNP[0] - 1'b1: Force non-posted writes.
  - 1'b0: Not forced to non-posted writes.

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																														WNP	

Table 74 AHBM_CTL register.

10.5.2 AHBM_IM

Interrupt Mask Register.

Attribute: RW

Bit field description:

- EWRP[2] - 1'b1: Mask interrupt due to Illegal WRAP.
- WER[1] - 1'b1: Mask interrupt due to Write error.
- RER[0] - 1'b1: Mask interrupt due to Read error.

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																													EWRP	WER	RER

Table 75 AHBM_IM register.

10.5.3 AHBM_STS

Status Register.

Attribute: WZC

Bit field description:

- EWRP[2] - 1'b1: Illegal WRAP detected.

- WER[1] - 1'b1: Write error detected.
- RER[0] - 1'b1: Read error detected.

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																													EWRP	WER	RER

Table 76 AHBM_STS register.

10.5.4 AHBS_CTL

Control Register.

Attribute: RW

Bit field description:

- REG[16] - 1'b1: Issue error response on unaligned read address.
  - 1'b0: Issue OK response even if unaligned address was aligned and issued.
- ZEROES[15:0] - Zeroes

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u															REG	ZEROES															

Table 77 AHBS_CTL register.

10.5.5 AHBS_IM

Interrupt Mask Register

Attribute: RW

Bit field description:

- REM[17] - 1'b1: No interrupt raised on RE.
  - 1'b0: Interrupt raised on RE.
- WEM[16] - 1'b1: No interrupt raised on WE.
  - 1'b0: Interrupt raised on WE.
- ZEROES[15:0] - Zeroes

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u														REM	WEM	ZEROES															

Table 78 AHBS_IM register.

10.5.6 AHBS_STS

Status Register.

Attribute: WZC

Bit field description:

- RE[17] - 1'b1: Unaligned read address was received.
- WE[16] - 1'b1: Unaligned write address (wrt asize)., or partial write strobes (wrt asize) was received.
- ZEROES[15:0] - Zeroes

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u														RE	WE	ZEROES															

Table 79 AHBS_STS register.

APB Bridge

10.5.7 APBSLV_BRIDGE_ID

Bridge ID register.

Attribute: R

Bit field description:

- **ZEROES**[15:8] - Zeroes
- **ID**[7:0] - Unique bridge id.

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																ZEROES								ID							

Table 80 APBSLV_BRIDGE_ID register.

10.5.8 APBSLV_BRIDGE_VERSION

Bridge version register.

Attribute: R

Bit field description:

- **VER**[3:0] - Bridge version.

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																												VER			

Table 81 APBSLV_BRIDGE_VERSION register.

10.5.9 APBSLV_SLVS_SLEEP_STATUS

Slave sleep status register.

Attribute: RW

Bit field description:

- **STS_15**[15] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_14**[14] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_13**[13] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_12**[12] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_11**[11] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_10**[10] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active

- **STS_9**[9] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_8**[8] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_7**[7] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_6**[6] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_5**[5] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_4**[4] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_3**[3] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_2**[2] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_1**[1] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active
- **STS_0**[0] - 1'b1: Slave is in sleep mode
  - 1'b0: Slave is active

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
u																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
u																STS_15	STS_14	STS_13	STS_12	STS_11	STS_10	STS_9	STS_8	STS_7	STS_6	STS_5	STS_4	STS_3	STS_2	STS_1	STS_0

Table 82 APBSLV_SLVS_SLEEP_STATUS register.

11 SystemC Model Support

In order to integrate C++ based models into custom environments it is necessary to define interfaces and APIs to enable integration and interacting with 3rd party models, debuggability and analysis of results. NetSpeed provides C++ based models for such integration purposes based on the SystemC TLM2 standard. This chapter describes the interfaces and how such models can be efficiently used by customers.

There is a variety of usages for C++ based models. The current mode specifically addresses the need for fast functional modeling in order to fit into platform models for software and firmware verification and development. Such models of typically referred to as LT (loosely timed) of PV (programmers view).

Other models (for other purposes) will be provided in future releases.

11.1 Prerequisites

There are some minimum prerequisites that any model deployment requires. First of all, a C++ compiler with support for the C++11 standard. Most modern compilers provide such support. For instance, gcc introduced the required support with version 4.4. Some of the older gcc version require the switch -std=c++0x. Newer versions enable C++11 by default.

Since the NetSpeed models are based on SystemC and the TLM2 standard, the user has to provide a SystemC version and the TLM2 library (since SystemC-2.3.1 included in the SystemC installation) which is compile compatible with the Accelera standard.

The host interfaces use AMBA based interfaces. In order to model those interfaces correctly the AMBA extensions for TLM2 sockets are used in the NetSpeed SystemC model.

Reference implementations for those prerequisites can be provided upon request.

11.1.1 Reference test environment

Since a large variety of configurations exists in the market (version of OS, machine type, compiler version, SystemC version, etc.) NetSpeed limits the number of combinations that are being tested before any release (see table below). Many other configurations will most likely work, since the models are provided as source code.

Table 83 Internal test environments

Model type	OS	compiler	Other libraries
LT	Linux RHEL5	gcc-4.8.0	SystemC-2.3.1

LT	Windows10	VC-12	SystemC-2.3.1

11.2 Model Generation

The SystemC LT model is generated as part of the output generation step (gen_ip). There is a property, which has to be enabled in the configuration file in order to enable the generation of SystemC code:

- prop_default sysc_enable yes

In addition, there is a license check. The license feature NocStudio_GEN_SYSC is guarding the generation of SystemC code. After the model generation step there will be a subdirectory systemc/PV_model under the directory indicated by the design name. This is the same directory which is used for any other output of the gen_ip step (RTL, IP-XACT, register file, etc.).

The top-level class which contains the interconnect code will be in a file called pv_fabric.h. This file also requires some other generated files which must be kept consistent if those files are moved for integration purposes.

11.3 Integration

NetSpeed is providing the model for the interconnect, as well as a test environment, which can be used to check the model fidelity. This test environment can be used as example (how the integration into a custom C++ environment could look like) or as a starting point. The initiator model in this test environment is reading stimulus from trace files, which are generated based on the traffic information specified in the configuration file. The diagram below shows the general structure of the generated test environment.

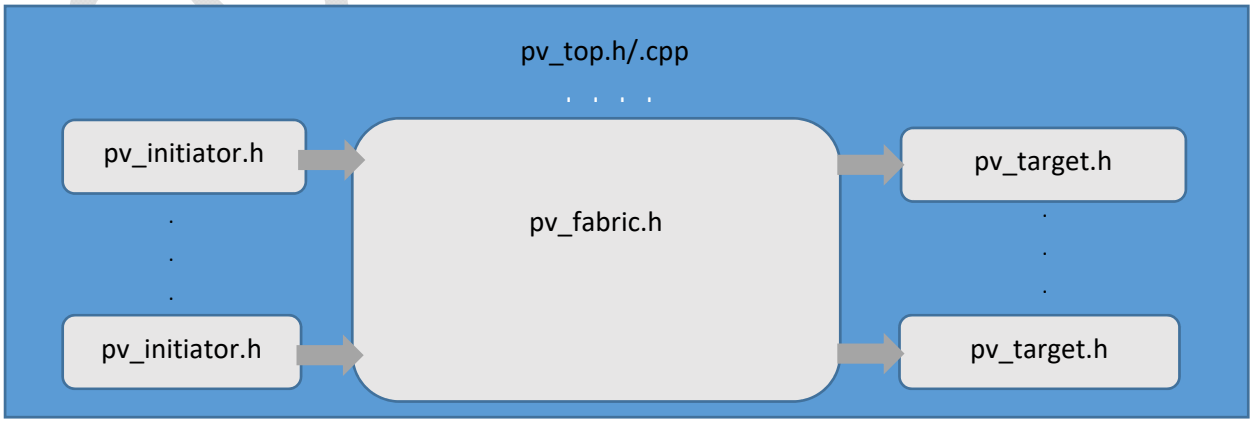


Figure 51 Basic structure of LT model

11.4 Performance considerations

It is very important to provide models, which can sustain high performance under the required workload for integration into software environments. The NetSpeed LT model provides an abstract, functional model, which allows for very high performance. The model supports typical performance optimizations like temporal decoupling and direct memory access (DMI).

11.5 Register accesses

The LT model is providing access to all registers in the interconnect. Some of those registers will not have a functional impact due to the nature of the LT model. Nevertheless, SW or FW will be able to access any register locations which maintains software compatibility between different versions of the NetSpeed models.

12 Appendix A: AMBA Protocol Support

This appendix describes support in NetSpeed Gemini for **AMBA AXI and ACE Protocol Specification Rev E.**

12.1 ACE / AXI4 Feature Adoption

The table below provides a high-level summary of ACE / AXI4 features supported by NetSpeed AXI NoC.

ACE/AXI4 Feature	NetSpeed AMBA NoC Support
READY/VALID Handshake	Full forward and reverse direction flow control of AMBA protocol-defined READY/VALID handshake.
Transfer Length	Non-coherent transactions support · 1 to 256 beats for incrementing bursts and · 2 to 16 beats for wrap bursts. · If width conversion is needed, these requirements must hold even on the width converted requests. Coherent ACE agents must use 64B cacheline sized and aligned transactions
Data Width	Agents can have data widths of 32, 64, 128, 256 and 512 bits. R channel and W channel must have equal data size.
Transfer Size	Narrow transactions are fully supported
Burst type	WRAP requests must be 16B, 32B or 64B for non-coherent transaction. Coherent WRAP transactions must be sized to 64B cacheline. FIXED transactions are supported by splitting them into multiple single beat INCRs
Long bursts	Long transactions may be split into multiple transactions at a configurable boundary. Non-coherent transactions will be split at 1024B boundaries by default. Coherent transactions from ACE and ACE-lite agents are split at 64B boundary.
Read/Write only	The IP supports only read/write mode. Read-only or write-only interfaces are not currently supported.
Exclusive Access	NetSpeed AMBA NoC can pass exclusive access transactions across a system. AXI4 does not support locked transfers Note that an exclusive access burst of 128B must not be performed on master bridges might split at 64B boundary. Exclusive access sent to an AXI3 slave must not violate the maximum transfer size supported by the interface.

Cache bits	NetSpeed AMBA NoC passes cache bits across a system. Cache Bit [1] can mark transaction as modifiable or non-modifiable. Modifiable transactions provide greater flexibility in the NetSpeed AXI NoC to transport and modify transactions passing through the system for greater performance. Non-modifiable transactions are honored, however some transaction marked as non-modifiable will still be subjected to modification, for example if width conversion operation requires that for functional correctness.
Protection bits	NetSpeed AMBA NoC passes protection bits across a system Access control to address ranges can be configured to use the transaction's protection bits
Quality of Service (QoS) Bits	NetSpeed AMBA NoC passes QoS bits across a system. QoS bits are also used for priority and weight assignments within the NoC.
REGION Bits	NetSpeed NoC generates region bits as part of address lookup and transports it. These regions bits are specified as part of user configuration of the address ranges. External REGION bits from a master is dropped.
User Bits	NetSpeed AXI NoC passes user bits across a system. The facility to transport user bits around a system allows special purpose custom systems to be built that require additional transaction-based sideband signaling. An example use of USER bits would be for transferring parity or debug information.
Read Interleaving	If interleaved read responses are expected from a slave, then a de-interleaving block should be added on the corresponding slave bridge. Transactions to slave supporting read interleaving will be split at 64B boundaries. Masters supporting interleaved read responses can enable the NoC to interleave responses of split segments of a read transaction.
Agents	Fully coherent and IO coherent agents are supported
Barriers	Both memory and synchronization barriers are supported
DVM	NetSpeed NoC supports DVM transactions for maintenance of a virtual memory system. Both DVMv7 and DVMv8 versions are supported.
Snoop Filter	Optional external snoop filtering is supported
Reset	NetSpeed AMBA NoC generally resets all VALID outputs within 16 cycles of reset, and has a reset pulse width requirement of 16 cycles or greater. Holding AXI ARESETn for 16 cycles of the slowest AXI clock is generally a sufficient reset pulse width for NetSpeed AXI NoC.

12.2 AMBA Signal Adoption

The tables below provide a high-level summary of AMBA signals supported by NetSpeed AXI NoC.

12.2.1 Global AXI Signals

Signal	ACE
ACLK	AXI port clock
ARESETn	Active low reset

12.2.2 Write Address Channel Signals

Signal	ACE
AWID	Fully supported. System AID width is equal to widest AID among master interface ports. On slave interface ports, AID width can be equal, greater or less than system AID width. Masters need only output the set of ID bits that it varies (if any) to indicate re-orderable transaction threads. Masters do not need to output the constant portion that comprises the Master ID, as this is appended by the NetSpeed NoC.
AWADDR	Fully supported. On Master and slave ports, address width can be greater or less than system address width. Range of supported address widths is 14 to 60-bits.
AWLEN	Non-Coherent transfers support bursts: • Up to 256 beats for incrementing (INCR). • Up to 16 beats for WRAP.
AWSIZE	Transfer width 8 to 512 bits supported.
AWBURST	INCR andWRAP fully supported. FIXED transactions are split into multiple single beat INCRs
AWLOCK	Exclusive access supported
AWCACHE	NetSpeed AXI NoC will pass Cache bits across a system. Signal bits can be selectively overridden on master or slave port bridge
AWPROT	NetSpeed AXI NoC passes Protection bits across a system. Signal bits can be selectively overridden on master or slave port bridge. Can be used for access control.
AWQOS	NetSpeed AXI NoC passes QoS bit across a system. QoS bits are also used for priority and weight assignments for flows
AWREGION	Supported. This input is unused on the master port interface and is configured to be generated as part of address lookup
AWUSER	User bits per AW transaction is transported across NoC to the destination. If an AW transaction is split into multiple transactions, then user bits of the original request is repeated for each of the resultant transaction.

AWSNOOP	All coherent, IO coherent and non-coherent write transactions are supported. WriteUnique transactions from ACE master agents cannot cross a 64B boundary
AWDOMAIN	All shareability domains supported
AWBAR	Write barriers fully supported
AWUNIQUE	Optionally supported for agents requiring WriteEvict
AWVALID	Fully supported.
AWREADY	Fully supported.

12.2.3 Write Data Channel Signals

Signal	ACE
WDATA	32, 64, 128, 256, 512 bit widths supported.
WSTRB	Fully supported.
WLAST	Fully supported.
WUSER	Number of user bits per byte of the interface can be configured. This is transported across the NoC
WVALID	Fully supported.
WREADY	Fully supported

12.2.4 Write Response Channel Signals

Signal	ACE
BID	Fully supported. See AWID for more information.
BRESP	Fully supported. For an AW requests which had been split into multiple AW requests, any change in write response between individual split segments results in SLVERR write response being sent to master.
BUSER	User bits are defined per B response transaction. For an AW request which had been split into multiple AW requests, user bits associated with write response of first split AW segment is delivered as the BUSER bits of the entire command. BUSER of subsequent segments are ignored.
BVALID	Fully supported.
BREADY	Fully supported.

12.2.5 Read Address Channel Signals

Signal	ACE
ARID	Fully supported. System AID width is equal to widest AID among master interface ports. On slave interface ports, AID width can be equal, greater or less than system AID width. Masters need only output the set of ID bits that it varies (if any) to indicate re-orderable transaction threads. Masters do not need to output the constant portion

	that comprises the Master ID, as this is appended by the NetSpeed NoC.
ARADDR	Fully supported. On Master and slave ports, address width can be greater or lesser than system address width. Range of supported address widths is 14 to 60-bits.
ARLEN	Non-Coherent transfers support bursts: • Up to 256 beats for incrementing (INCR). • Up to 16 beats for WRAP.
ARSIZE	Transfer width 8 to 512 bits supported.
ARBURST	INCR and WRAP fully supported. FIXED transactions are split into multiple single beat INCRs
ARLOCK	Exclusive access supported
ARCACHE	NetSpeed AXI NoC will pass Cache bits across a system. Signal bits can be selectively overridden on master or slave port bridge
ARPROT	NetSpeed AXI NoC passes Protection bits across a system. Signal bits can be selectively overridden on master or slave port bridge. Can be used for access control.
ARQOS	NetSpeed AXI NoC passes QoS bit across a system. QoS bits are also used for priority and weight assignments for flows
ARREGION	Supported. This input is unused on the master port interface and is configured to be generated as part of address lookup
ARUSER	User bits per AR transaction is transported across NoC to the destination. If an AR transaction is split into multiple transactions, then user bits of the original request is repeated for each of the resultant transaction.
ARVALID	Fully supported.
ARREADY	Fully supported.
ARSNOOP	All coherent, IO coherent and non-coherent read transactions are supported. ReadOnce transactions from ACE master agents cannot cross a 64B boundary
ARDOMAIN	All shareability domains supported
ARBAR	Read barriers fully supported

12.2.6 Read Data Channel Signals

Signal	ACE
RID	Fully supported. See ARID for more information.
RDATA	Data widths of 32, 64, 128, 256 and 512 bits supported
RRESP	Fully supported. If a change in read response is detected between beats of a read response, subsequent beats are marked with SLVERR
RLAST	Fully supported.
RUSER	RUSER bits have two parts, one part defined per byte of the interface and second defined for the entire R response transaction For an AR request which had been split into multiple AR requests, user bits associated with read response of first split AR segment is delivered as the per

	transaction part of RUSER bits for the entire command. Per transaction RUSER bits of subsequent segments are ignored.
RVALID	Fully supported.
RREADY	Fully supported.

12.2.7 Snoop Address Channel Signals

Signal	ACE
ACADDR	Supported on coherent ACE agents and DVM master agents
ACSNOOP	Supported on coherent ACE agents and DVM master agents
ACPROT	Supported on coherent ACE agents and DVM master agents
ACVALID	Supported on coherent ACE agents and DVM master agents
ACREADY	Supported on coherent ACE agents and DVM master agents

12.2.8 Snoop Response Channel Signals

Signal	ACE
CRRESP	Supported on coherent ACE agents and DVM master agents
CRVALID	Supported on coherent ACE agents and DVM master agents
CRREADY	Supported on coherent ACE agents and DVM master agents

12.2.9 Snoop Data Channel Signals

Snoop data (CD) channel is optional on an ACE agent. This channel is also not supported on DVM only master agents. If present, CDDATA width can be configured independent of the agent's AXI data width

Signal	ACE
CDDATA	Optionally supported on coherent ACE agents
CDLAST	Optionally supported on coherent ACE agents
CRVALID	Optionally supported on coherent ACE agents
CRREADY	Optionally supported on coherent ACE agents

12.2.10 Acknowledge Channel Signals

Signal	ACE
RACK	Read acknowledge supported from coherent ACE agents
WACK	Write acknowledge supported from coherent ACE agents

12.3 AXI4-Lite Feature Adoption

AXI4 Lite Feature	NetSpeed AXI NoC Support
Ports	AXI4-Lite master and slave ports can connect to NetSpeed AXI NoC. Intercommunication between AXI4 agents and AXI4-Lite agents is supported, conversions are performed by the NoC
Conversion	INCR transactions of length > 1 from AXI masters to AXI4-Lite slaves, are

	split into multiple AXI4-Lite transactions of 32-bit or 64-bit size. Responses from the slave are converted back to the format expected by the AXI master. WRAP requests sent to an AXI4-Lite slave will result in a SLVERR response
Transfer Length	AXI4-Lite is restricted to single beat transactions
Data Width	32-bit and 64-bit AXI4-Lite interfaces are supported
Address Width	Support range of address width is 14-bit to 60-bits
Transfer size	AXI4-Lite does not support narrow transfers and all transactions are of full data width
AID	No ID bits are present in AXI4-Lite. Multiple outstanding transactions are supported; however, all transactions must be ordered.

12.4 AXI3 Feature Adoption

AXI3 Feature	NetSpeed AXI NoC Support
Transfer Length	Burst length is restricted to the range 1 to 16 beats
Splitting	If a transaction from an AXI4 master would exceed ALEN on an AXI3 slave port, then the transaction is split into multiple transactions at 16 beat boundaries.
WID	This signal is available on the W data channel interface. However, write interleaving is not supported
AxLOCK[1:0]	AXI3 locked transactions are not supported and are converted to Normal transactions

12.5 AHB-Lite Feature Adoption

AHB-Lite Feature	NetSpeed AXI NoC Support
Version	AHB-Lite master and AHB-Lite slave devices can connect to NetSpeed AXI NoC.
Data Width	32, 64, 128 bits
Address Width	Range of 14 – 60 bits
HPROT	HPROT[0] maps to AxPROT[2] after logic inversion (due to polarity change) HPROT[1] maps to AxPROT[1] HPROT[2] maps to AxCACHE[0] HPROT[3] maps to AxCACHE[3], AxCACHE[2] and AxCACHE[1] after logic duplication
HSIZE	8, 16, 32, 64, 128-bit transfer size
HMASTLOCK	Locked transfers are not supported. HMASTLOCK will be ignored.
HBURST	All standard specified modes supported. INCR (Incrementing burst of

	unspecified length) is split at 64B boundary.
HSELx	Up to 16 AHB-Lite slaves can connect to an AHB-Lite slave bridge of the NoC. Slaves on a given slave bridge can have different address and data width but common endian format
Response delay	On AHB-Lite master, bufferable transactions will be provided early response and non-buffer transactions will receive response from the end point.
Endian format	Each AHB-Lite interface on NetSpeed NoC can be configured to handle little endian or big endian format
Write strobes	AHB-Lite interface does not support write strobes. However, write transactions from AXI masters to AHB-Lite slaves can use partial write strobes. Empty write strobes result in no AHB access and OK response.
Address alignment	AXI masters can perform write accesses to AHB slaves with any address alignment. Read transactions from AXI masters with unaligned addresses are changed to aligned addresses. Read is performed to the aligned address and response can optionally be marked with SLVERR based on a programmable register bit.

12.6 APB Feature Adoption

APB Feature	NetSpeed AXI NoC Support
Version	APB 2/3/4 slaves can be connected to NetSpeed AXI NoC
Conversion	INCR transactions from AXI masters are broken down into multiple single beat APB transactions at APB slave bridge. Responses from the slaves are converted back to the format expected by the AXI master. WRAP requests sent to an APB slave will result in a SLVERR response
Data Width	32-bit APB slave devices are supported
Address Width	Address is fixed at 32-bit
PSELx	Up to 16 APB slave devices can be connected to single APB slave bridge. Each APB slave is identified by a REGION associated with its address range
PREADY	Supported on APB 3, 4 slaves to allow slave to extend an APB transfer
PPROT	Supported on APB 4 slaves
PSLVERR	Supports slave error from APB 3, 4 slave devices and remaps appropriately to response sent back to master
Address alignment	All read transaction address from AXI masters to APB slaves must be aligned to 32-bits Narrow read or write transactions sent to APB slave are modified to be full prior to conversion to APB, i.e. any AxSIZE from masters is changed to 2 Write transaction address to APB 2, 3 slaves must be aligned to 32-bit (unaligned access results in SLVERR response).

	APB 4 slaves can be sent unaligned write transaction addresses.
PSTRB	Supported on APB 4 slaves to allow partial byte updates during write transfers. Write transactions from AXI masters to APB 2, 3 slaves must not use partial write strobe. Partial write strobes result in SLVERR response? Empty write strobes result in no access and OK response.

2870 Zanker Road,

Suite 210,

San Jose, CA 95134

(408) 617-5209

http://www.netspeedsystems.com