NetSpeed Crux

Technical Reference Manual

Version: CRUX-16.09

September 27, 2016

NetSpeed Crux 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 Crux 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

Related Documents

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

- NetSpeed NocStudio Crux User Manual
- NetSpeed Crux 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

Contents

About This Document..2

Audience..2

Prerequisite..2

Related Documents..2

Customer Support..2

1 Introduction..9

1.1 NetSpeed Crux Overview..9

1.2 Configurability Options..9

2 Functional Description: System Interconnect..10

2.1 NoC Topology..10

2.2 Routers..13

2.3 Bridging from Host to NoC..14

2.4 Information Transport in the NoC..17

2.5 Clocking..21

3 Deadlock Avoidance..26

3.1 Quick Primer on Deadlocks..26

3.2 Constructing Deadlock-Free Interconnects..28

4 Quality of Service Support..30

4.1 Traffic Classes..31

4.2 Strict priority based allocation..33

4.3 Weighted bandwidth allocation..36

4.4 Rate Limiting Hosts..39

4.5 Dynamic Priority Support for Isochronous Traffic..44

5 NoC Serviceability: Regbus Layer..46

5.1 The Register Bus..46

5.2 NoC Registers..51

5.3 Error Responses To Register Accesses ... 52

5.4 User Register Bus Access ... 53

5.5 Register Bus Master Interface ... 54

5.6 Expected Usage of Register Bus Master ... 57

5.7 Ring Slave to Host Interface ... 57

5.8 Atomic Operations ... 58

5.9 Restrictions ... 61

6 Programmers Model ... 63

6.1 Router Registers ... 63

6.2 Bridge registers ... 70

6.3 Regbus Bridge Registers ... 81

7 Appendix A: NetSpeed Streaming Interface Protocol ... 82

7.1 NSIP Transmitter ... 82

7.2 NSIP Receiver ... 85

7.3 Credit Based Flow Control at TX and RX Interfaces ... 87

7.4 Width Ratios and Conversion Summary ... 88

7.5 Ordering Requirements ... 90

7.6 NetSpeed Streaming Bridge Overview ... 90

7.7 Adding Streaming Bridge and Traffic in NocStudio ... 92

7.8 Streaming Bridge Specification ... 93

7.9 QoS ... 94

Figures

Figure 1. Bridge and Router Functions in the NoC 10
Figure 2. A 4x4 Homogeneous Grid or Mesh Interconnect 11
Figure 3 A 4x4 heterogeneous grid or mesh interconnect 12
Figure 4. Schematic Symbol of a NocStudio RTL-Library Router Component 14
Figure 5: NetSpeed Streaming Bridge functions 15
Figure 6. Information Transport in the NoC 17
Figure 7. NoC Packet Organized in Two Different Flit Sizes 18
Figure 8. Merging and Dividing Flits with Various Channel Widths 20
Figure 9. Combinations of Flit Merging and Division at Router Ports 20
Figure 10. Multiple Clock Domains and Clock Crossings 22
Figure 11 In-Link Domain crossing structure 23
Figure 12. Clock Gating as Implemented in NetSpeed NoC 24
Figure 13. Simple Deadlock Graph with Two Resources 27
Figure 14. Simple Interconnect with Deadlock 27
Figure 15 - System Level Dependency Graph 29
Figure 16: System with 2 masters and one slave 34
Figure 17: Class/Priority mapping 34
Figure 18: Snapshot of Performance Simulation with same class, same priority 35
Figure 19:Snapshot of performance simulation with 2 classes, different priority 35
Figure 20:Snapshot of performance simulation with 2 classes, same priority 36
Figure 21: 3 master, 1 slave system 37
Figure 22: Performance results with ratios 3:2:1 38
Figure 23 - Token Count increases at specified rate. Packet transmit decrements count. 41
Figure 24 - Token Bucket 42
Figure 25. Regbus Master Bridge 47
Figure 26. Regbus Layer Communication 48
Figure 27. Regbus Tunnel Connects Primary NoC Layer to Regbus Layer 49
Figure 28: Regbus Address Map 51
Figure 29: Register bus master bridge 54
Figure 30: Waveform showing read requests and responses at the register bus master interface 57
Figure 31: Waveform showing write requests and responses at the register bus master interface 57
Figure 32 : Waveform showing ring slave read requests and responses (4B and 8B) 61
Figure 33 : Waveform showing ring slave write requests and responses (4B and 8B) 61

Figure 34 Router Input VC Status (RIVCS) register. 63
Figure 35 Router Output VC Status (ROVCS) register. 64
Figure 36 Router Event Interrupt Status (RE) register. 65
Figure 37 Router Event Interrupt Mask (REM) register. 66
Figure 38 Router event counter control (RECC) register. 67
Figure 39 Router Event Counter (REC) register. 68
Figure 40 Router ID (RID) register. 68
Figure 41 Router Parity Error (RPERR) register. 69
Figure 42 Router Mask Parity Error (RPERR) register. 70
Figure 43 QoS Profile Data (P) register. 70
Figure 44 Bridge Transmit Event Counter Control (T) register. 71
Figure 45 Bridge Receive Event Counter Control (R) register. 72
Figure 46 Bridge Transmit FIFO Status (BTS) register. 73
Figure 47 Bridge Receive FIFO Status (BRS) register. 74
Figure 48 Bridge Rx Upsizer Status (BRUS) register. 74
Figure 49 Bridge Tx Upsizer Status (BTUS_0) register. 75
Figure 50 Bridge Tx Upsizer Status (BTUS_1) register. 75
Figure 51 Receive Interrupt Status (RXE) register. 75
Figure 52 Receive Interrupt Mask (RXEM) register. 76
Figure 53 Transmit Interrupt Status (TXE) register. 76
Figure 54 Transmit Interrupt Mask (TXEM) register. 77
Figure 55 Tx Bridge ID (TXID) register. 78
Figure 56. Rx Bridge ID (RXID) register. 78
Figure 57 Streaming TX Rate Limiter (BTRL) register. 78
Figure 58 Credit parity error on each layer (BTPERR) register. 79
Figure 59 Interrupt Mask Credit parity error on each layer (BTPERRM) register. 79
Figure 60 Receive layer ECC/Parity errors (BRPERR) registers. 80
Figure 61 Receive layer ECC/Parity errors Masks (BRPERRM) registers. 81
Figure 62 System Clock Gate Control Register in RegBus Slave Bridge (RegBus Ring Master) for Slaves on that node (RBSLVCG). 81
Figure 63: Timing of Single beat transactions at TX Bridge 84
Figure 64: Timing of Multi beat transactions at TX Bridge 85
Figure 65: Timing of Single beat transactions at RX Bridge 86
Figure 66: Timing of Multi beat transactions at RX Bridge 87
Figure 67: Credit based flow control at TX and RX interfaces of NSIP agents. 87
Figure 68: Timing of Multi beat transaction with credit flow control at TX Bridge 88
Figure 69: Timing of Multi beat transaction with credit flow control at RX Bridge 88

Figure 70 Four TX interfaces (left) and four RX interface (right) that may and may not communicate with each other based on interface widths and single beat prop 89
Figure 71 Illustration of width conversion of data beats of a message in NSIP protocol from a narrow TX interface to a wide RX interface 90
Figure 72 NetSpeed streaming bridge Interfaces to host port and to NoC.......... 91
Figure 73 Illustration of a NoC with streaming host ports and traffic between them 92

Tables

Table 1 Host and NetSpeed Streaming Bridge Interface 16
Table 2 - Rate limiter Configuration Register 43
Table 3: Register attribute table 51
Table 4: Register bit attribute table 52
Table 5: Response table for NoC Register Accesses 52
Table 6: Response table for User Register Bus Accesses 53
Table 7: Register Bus Master Interface signals 54
Table 8: Register slave to host interface 58
Table 9 NoC Streaming Bridge TX signals from Streaming Host port to NoC 83
Table 10 NoC Streaming Bridge RX signals from NoC to Streaming Host port 86

1 Introduction

1.1 NetSpeed Crux Overview

NetSpeed Crux is a scalable, 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, Crux uses a requirements-driven design approach. Crux 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. Crux is built upon following the following fundamental design principles.

1.1.1 Requirements driven approach

Crux 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

Crux 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 Crux 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.

2 Functional Description: System Interconnect

This section describes key system interconnect, NoC architecture concepts and features.

Figure 1 illustrates the basic building blocks of the NetSpeed System Interconnect architecture.

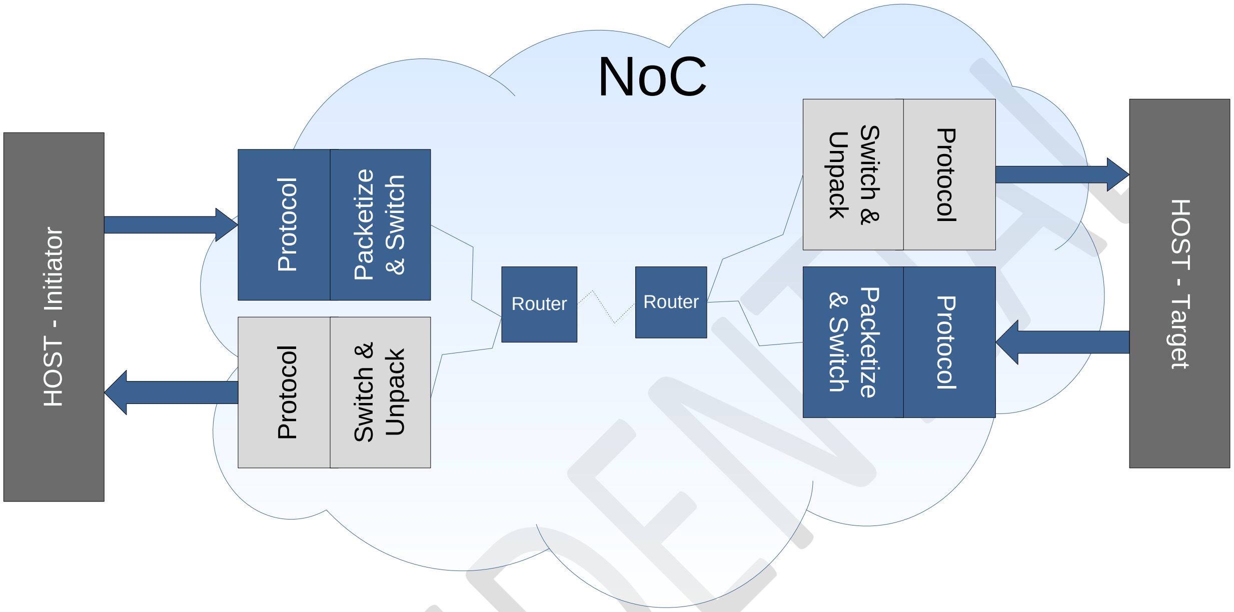


Figure 1. Bridge and Router Functions in the NoC

A bridge can connect a master or slave block to the NoC and perform the required operations to support the master and slave communication as per the NSIP protocol standards. Bridges packetizes the host blocks transactions into NetSpeed packet format during injection into NoC and de-packetizes them during ejection. 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. Figure 2 illustrates a full 2D-mesh interconnect, with a router at each mesh cross point.

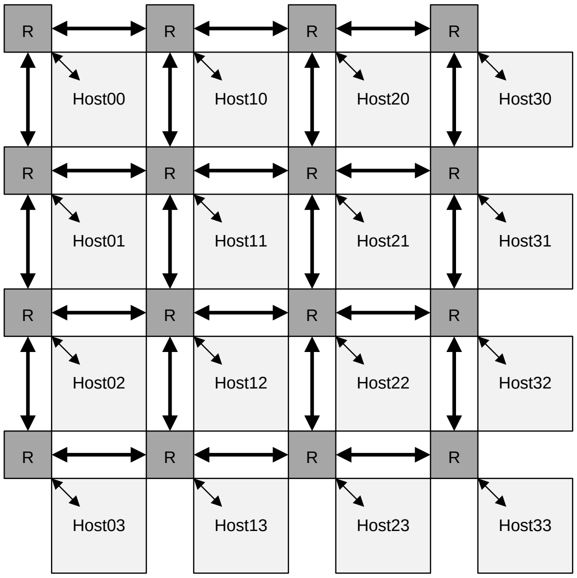


Figure 2. A 4x4 Homogeneous Grid or Mesh Interconnect

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 Figure 2. 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 Figure 2. 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 Figure 3 **A 4x4 heterogeneous grid or mesh interconnect**. 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.

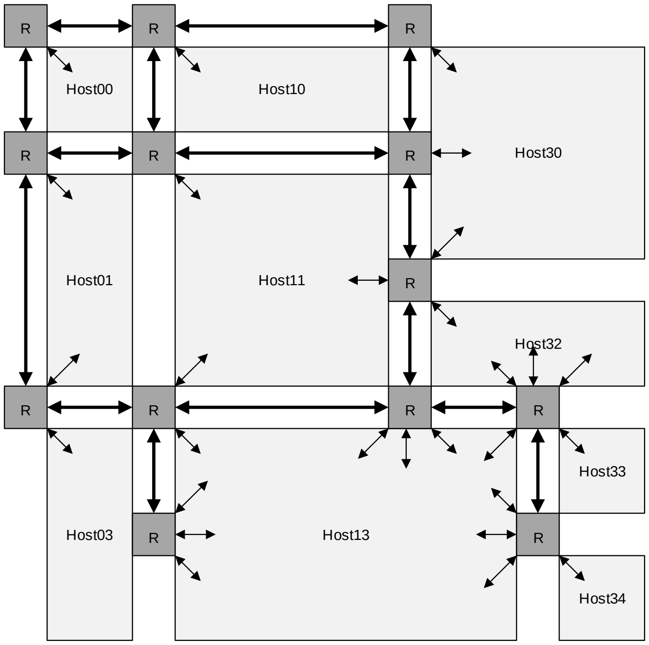


Figure 3 A 4x4 heterogeneous grid or mesh interconnect

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 Figure 3 **A 4x4 heterogeneous grid or mesh interconnect**, Host13 is 3x2 (X by Y) in size, spanning six grid cells. The figure 8 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

Figure 4 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.

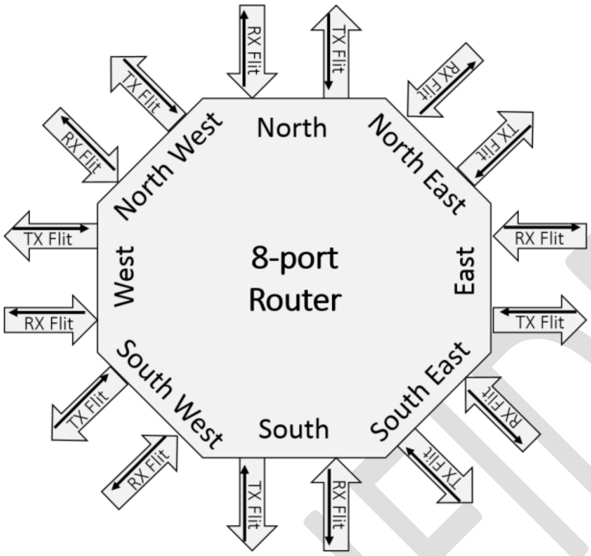


Figure 4. Schematic Symbol of a NocStudio RTL-Library Router Component

2.3 Bridging from Host to NoC

A host may have multiple Streaming ports through which it can transmit and receive data to/from the NoC. A bridge component converts a host port's messaging protocol into a packetized protocol for NoC. Bridges are automatically instantiated by NocStudio based on the protocol specified for host port in NocStudio. There is one bridge per host port; the bridge connects the host port to routers at the grid points. Multiple routers may exist at a grid point of the mesh, one router for each NoC layer, in which case the bridge will connect 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; these properties can be directly modified with "host prop" and "bridge_prop" commands in the

NocStudio. Please refer to these commands 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 through NocStudio based on optimizations or user specification.

2.3.1 NetSpeed Streaming Bridge

The Streaming Bridge maps the transaction messages issued from host port interfaces to NoC layers and VCs. The bridge also performs the route computation, enforces QoS (fixed priority and weights), and performs the flow control and arbitration.

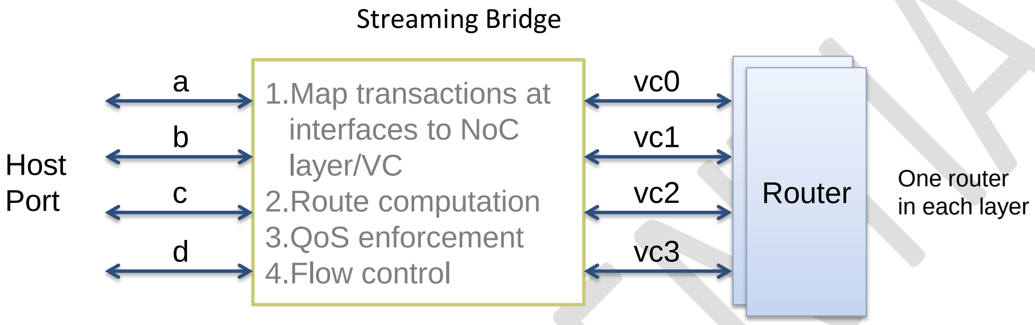


Figure 5: NetSpeed Streaming Bridge functions

The NetSpeed streaming protocol bridge provides a relatively simple interface to the host ports and performs simple packetization of host port data into NoC packets.

Streaming hosts using the streaming bridges to communicate with each other can use up to four interfaces allowed by the bridge. The interfaces are named a, b, c and d. Whether an interface is bidirectional or not, depends on the traffic specification, i.e. if a traffic transaction message only leaves (arrives at) an interface, then the interface becomes an output-only (input-only), otherwise it remains bidirectional. The width of each direction of an interface is configurable in terms of power-of-two multiples of cell_size, and can be viewed and modified using the **bridge_prop** command.

A streaming host's interface may communicate with any other streaming host's interface. Each injected transaction message must have a destination bridge id (globally unique identifier) and interface id. Note that two interfaces with different width can communicate with each other. For streaming host ports, each transaction message hop must be explicitly specified using **add_traffic**. The dependencies between different transaction hop messages are inferred from **add_traffic** and they can also be specified explicitly with the **add_dep** command. An interface may transmit or receive traffic of multiple priorities and weights, properties indicated in the transaction using the 4-bit QoS signal.

The signals at the host port side of a NetSpeed streaming bridge are listed below; the direction is with respect to the host port.

Table 1 Host and NetSpeed Streaming Bridge Interface

Signal Name	Dir		Description
Ingress interface to NoC from Host			
destination bridge id	OUT	8-bit	Destination bridge id (its globally unique identifier) for this message
interface id	OUT	2-bit	Destination host port's interface id for this message
data beat valid	OUT	1-bit	data valid
start of transaction message	OUT	1-bit	start of message
end of transaction message	OUT	1-bit	end of message
quality of service attribute of the message	OUT	4-bit	QoS of the message
data beat	OUT	EGRESS_WIDTH_*-bit	Data
credit increment return	IN	1-bit	credit increment feedback
Egress interface from NoC to Host			
data beat valid	IN	1-bit	data valid
start of transaction message	IN	1-bit	start of message
end of transaction message	IN	1-bit	end of message
data beat	IN	INGRESS_WIDTH_*-bit	Data
credit increment return	OUT	1-bit	credit increment feedback

Here * can be replaced with a, b, c or d for the four host port interfaces

Each transaction injected at an interface forms a single NoC packet.

2.4 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 path is shown in Figure 6.

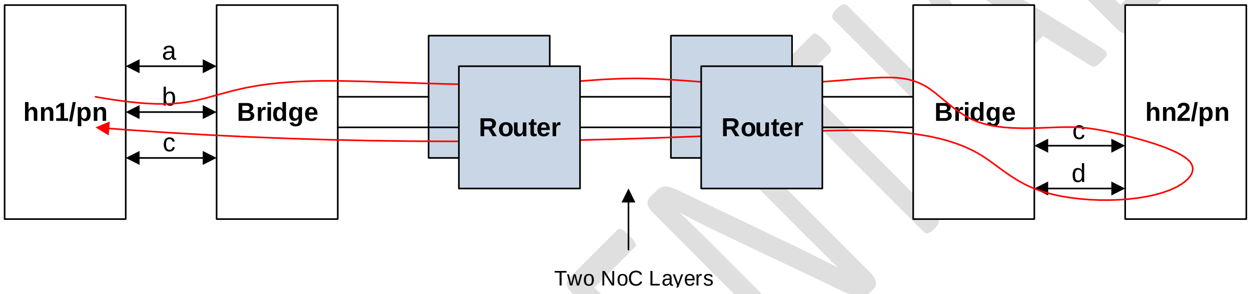


Figure 6. Information Transport in the NoC

2.4.1 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).

2.4.2 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.

2.4.3 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 the interfaces cell-size. The cell size value is fixed in NocStudio for the entire NoC project (the NoC project **cell_size** can be modified with the **mesh_prop** command. Multiple cell-sizes can be supported by setting this property to -1. Interfaces communicating to each other must share a cell-size, and each physical network can only support one cell-size). EOP flits can contain any integer multiple of the cell size payload bits. The value of the cell size must be selected based on the host-port signal properties to maximize the packing efficiency of the bridges. Figure 7 illustrates a packet that is organized as two large flits or four small flits.

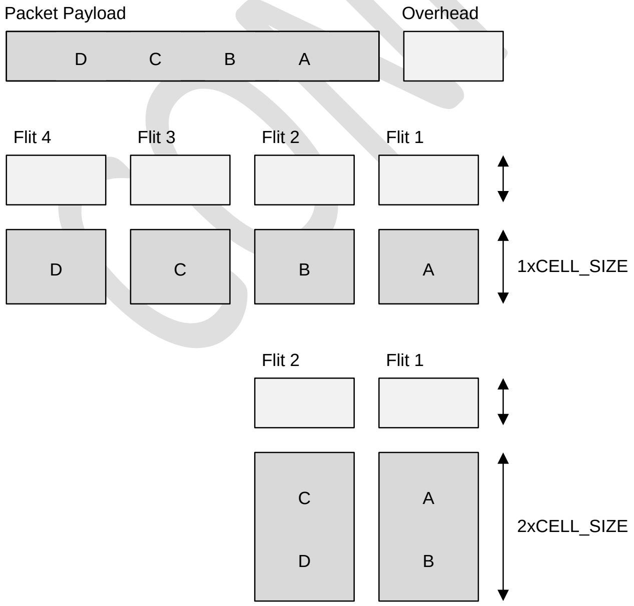


Figure 7. NoC Packet Organized in Two Different Flit Sizes

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 the 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 for a single cell size. 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 8 illustrates merging and dividing flits under various channel-width scenarios.

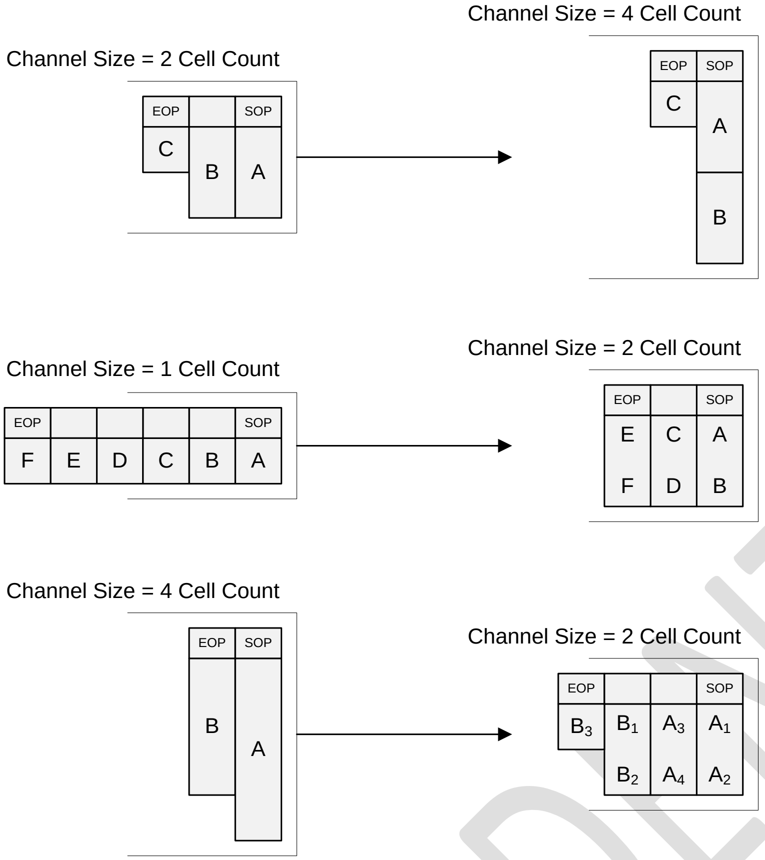

Channel Size = 2 Cell Count
EOP
SOP
C
B
A
Channel Size = 4 Cell Count
EOP
SOP
C
A
B
Channel Size = 1 Cell Count
EOP
SOP
F
E
D
C
B
A
Channel Size = 2 Cell Count
EOP
SOP
E
C
A
F
D
B
Channel Size = 4 Cell Count
EOP
SOP
B
A
Channel Size = 2 Cell Count
EOP
SOP
B_3
B_1
A_3
A_1
B_2
A_4
A_2

Figure 8. Merging and Dividing Flits with Various Channel Widths

Figure 9 shows how various router paths might have to merge and divide different flit sizes.

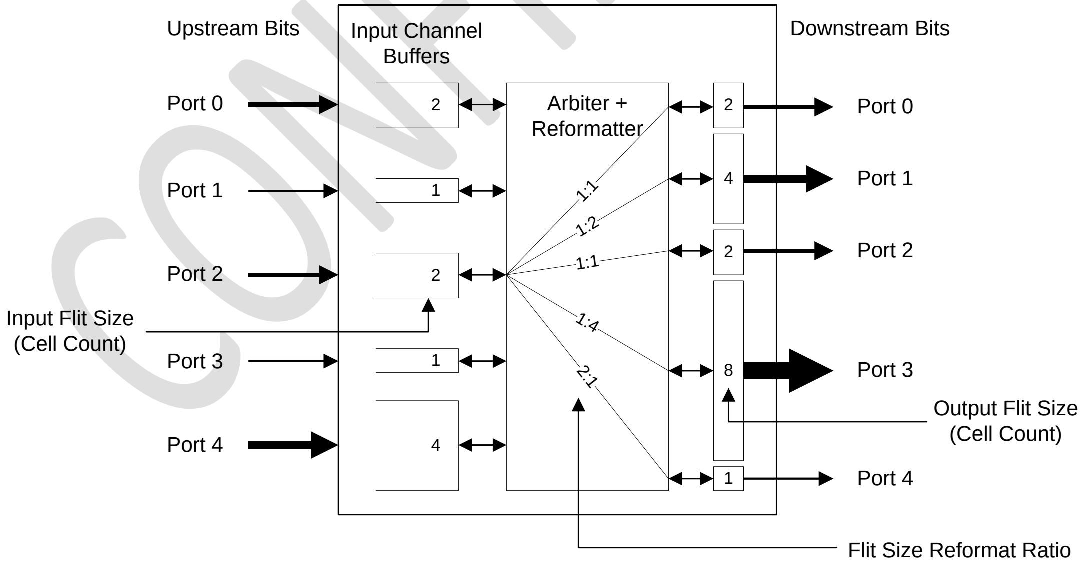


Figure 9. Combinations of Flit Merging and Division at Router Ports

2.4.4 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 a power-of-2 multiple of the 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 the 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 the cell size wide, and router port channels carrying multi-flit packets are power-of-two multiples of the cell size wide.

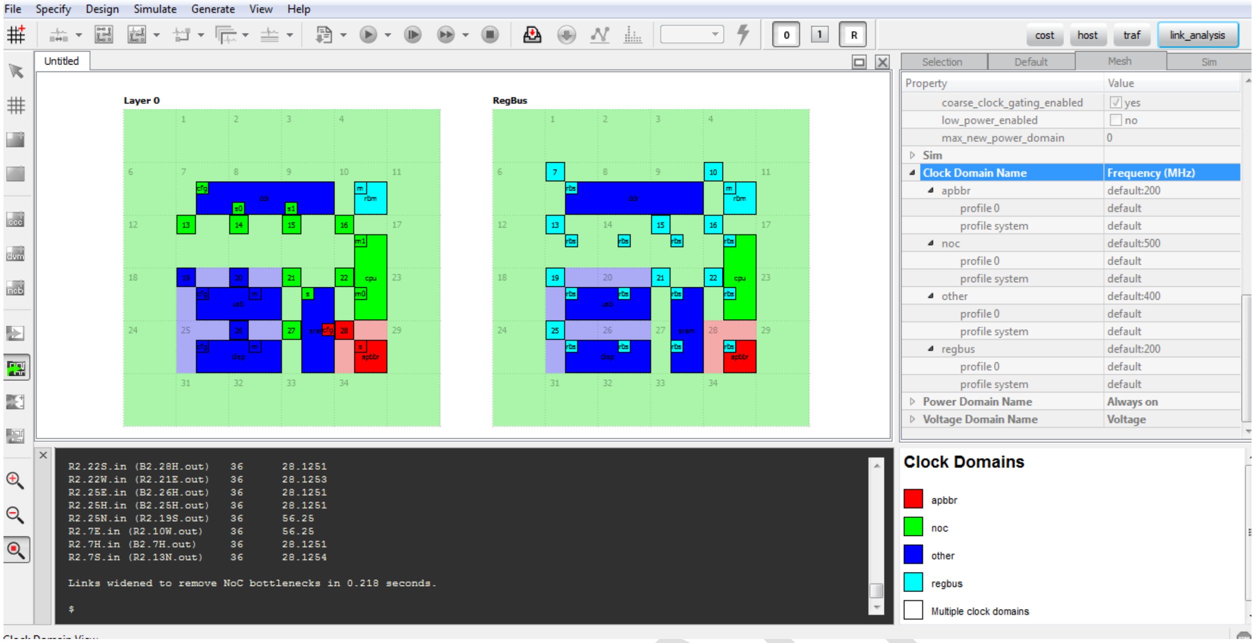
A similar convention is used for router directional channels, with the exception that a variety of messages can traverse the channels between various interfaces. Thus, all transaction messages and the corresponding packets are examined. The channel widths can be increased during NoC optimizations using 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.

2.5 CLOCKING

2.5.1 Clock Domains and Clock Crossing

Below 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. In below figure, 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.



Property	Value
coarse_clock_gating_enabled	☑ yes
low_power_enabled	□ no
max_new_power_domain	0
Sim	
Clock Domain Name	Frequency (MHz)
apbbr	default:200
profile 0	default
profile system	default
noc	default:500
profile 0	default
profile system	default
other	default:400
profile 0	default
profile system	default
regbus	default:200
profile 0	default
profile system	default
Power Domain Name	Always on
Voltage Domain Name	Voltage

```
R2.22S.in (B2.28H.out)   36      28.1251
R2.22W.in (R2.21E.out)   36      28.1253
R2.25E.in (B2.26H.out)   36      28.1251
R2.25H.in (B2.25H.out)   36      28.1251
R2.25N.in (R2.19S.out)   36      56.25
R2.7E.in (R2.10W.out)    36      56.25
R2.7H.in (B2.7H.out)     36      28.1251
R2.7S.in (R2.13N.out)    36      28.1254

Links widened to remove NoC bottlenecks in 0.218 seconds.

$
```

Clock Domains

- 🟥 apbbr
- 🟩 noc
- 🟦 other
- 🟦 regbus
- □ Multiple clock domains

Figure 10. 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 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.

2.5.2 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.

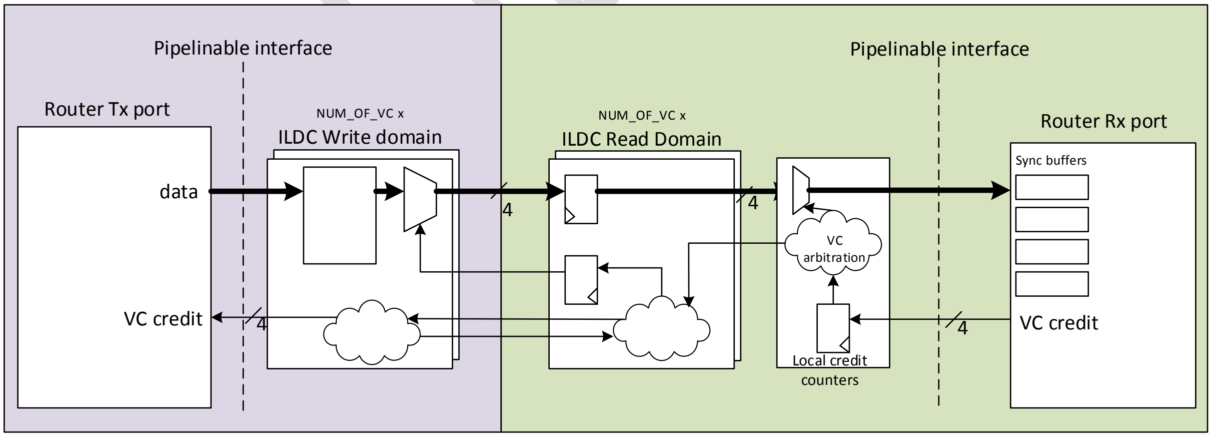


Figure 11 In-Link Domain crossing structure

2.5.3 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.

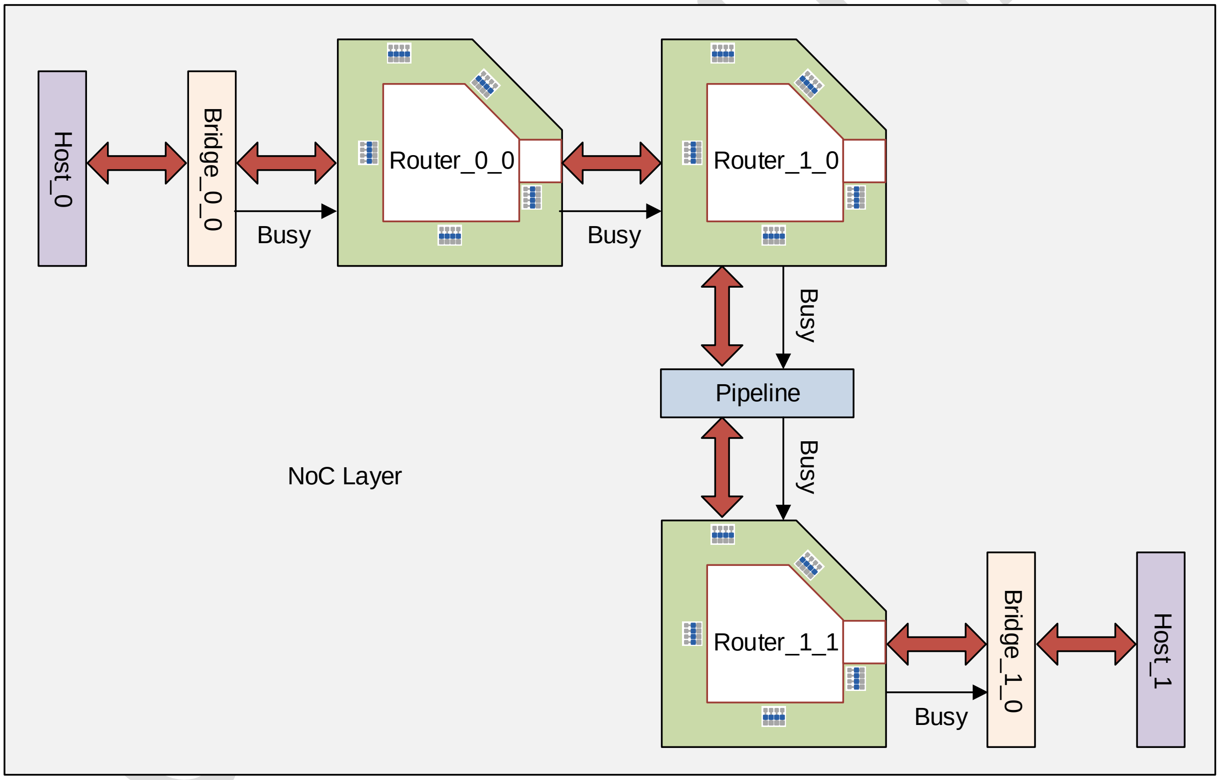


Figure 12. Clock Gating as Implemented in NetSpeed NoC

2.5.3.1 *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
- 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 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.

3.1 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 13 shows the resources and dependencies for this example.

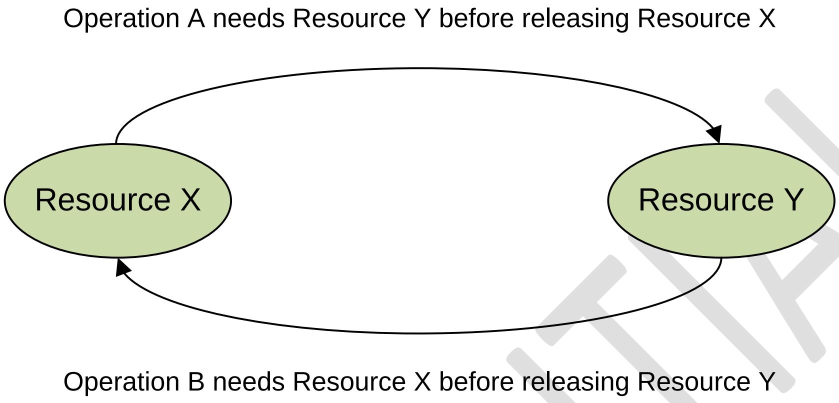

Operation A needs Resource Y before releasing Resource X
Resource X
Resource Y
Operation B needs Resource X before releasing Resource Y

Figure 13. Simple Deadlock Graph with Two Resources

Figure 13 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 14 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.

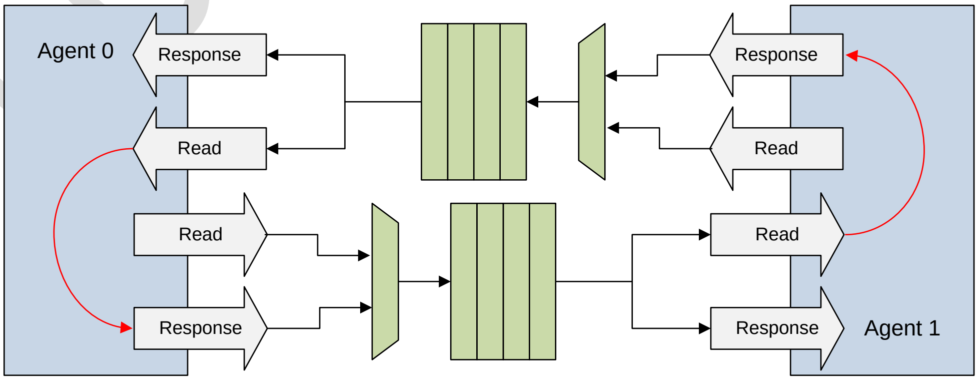


Figure 14. Simple Interconnect with Deadlock

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.

3.2 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 14 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 14, 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 15, NocStudio also generates a comprehensive system dependency graph as well so that architects can crosscheck and ensure that there are no deadlocks.

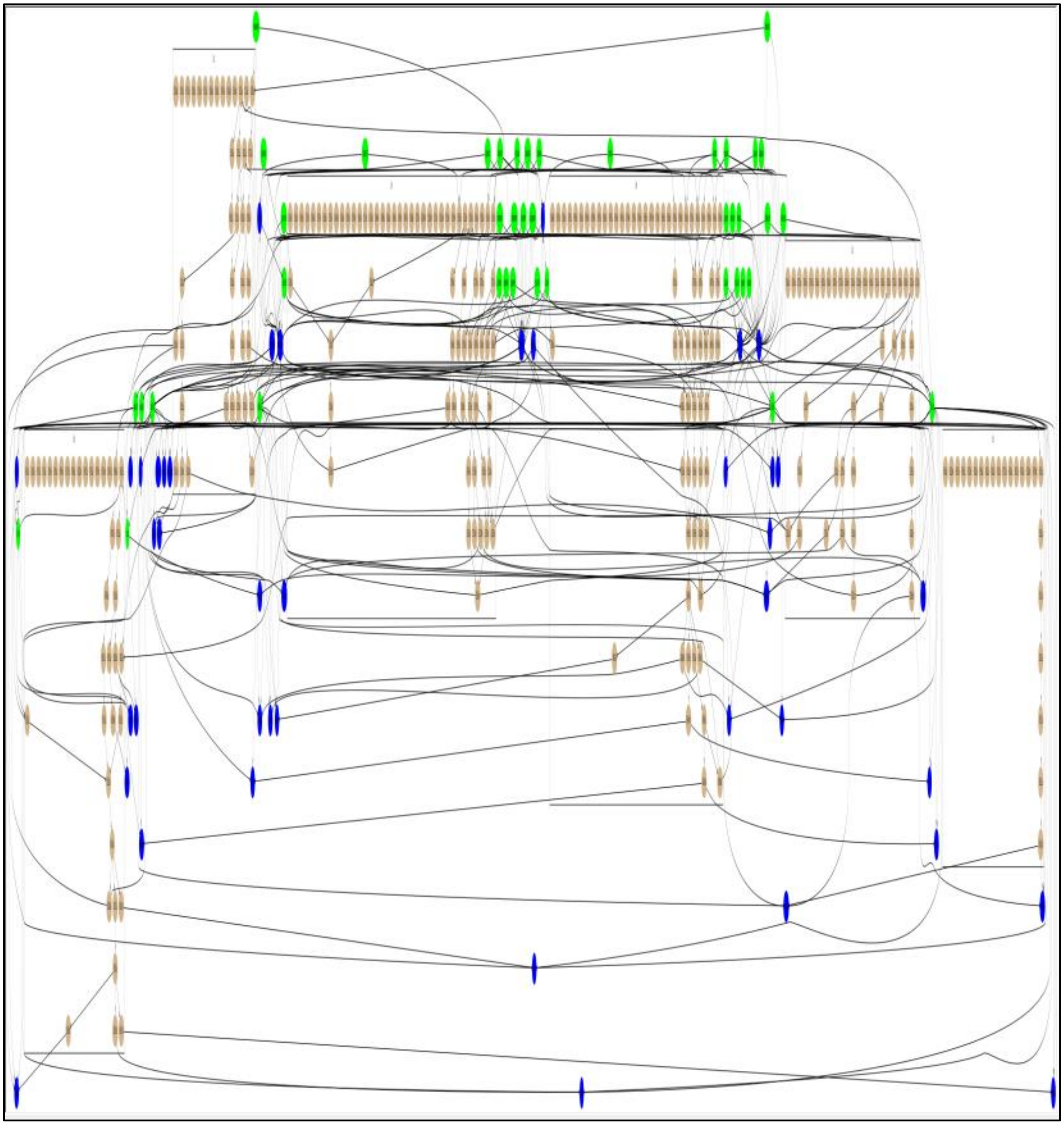


Figure 15 - System Level Dependency Graph

4 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 requirement 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

4.1 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.

4.1.1 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.

4.1.2 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 request 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 classs 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, an 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.

4.1.3 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.

4.2 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.

4.2.1 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.

4.2.2 Examples

Here are some examples on strict priority based allocation.

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

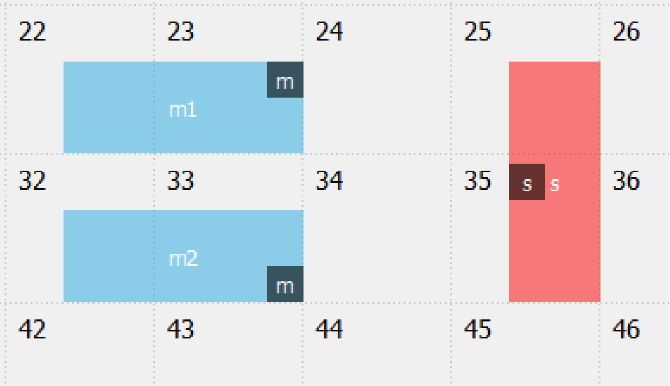
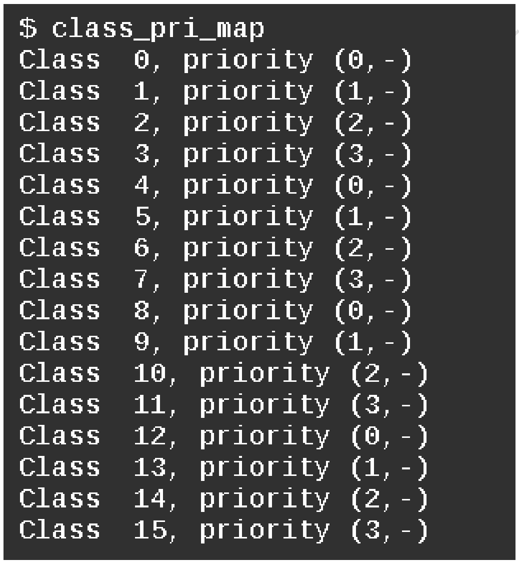


Figure 16: System with 2 masters and one slave

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 17: Class/Priority mapping

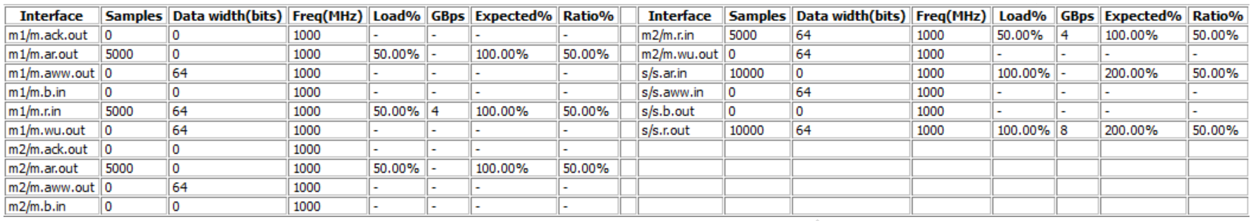
Example #1

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 18: Snapshot of Performance Simulation with same class, same priority

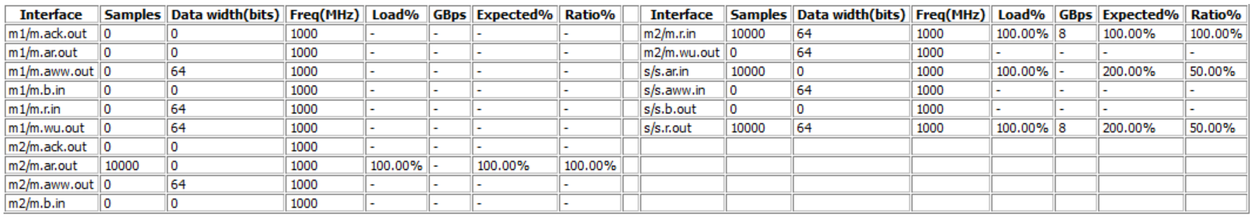
Example #2

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 19:Snapshot of performance simulation with 2 classes, different priority

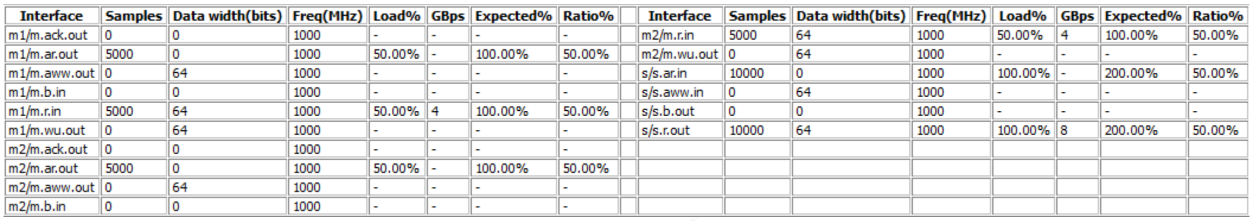
Example #3

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 20:Snapshot of performance simulation with 2 classes, same priority

4.3 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.

4.3.1 How to Define the Weight

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

4.3.2 Examples

Here are some examples on weighted bandwidth allocation.

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

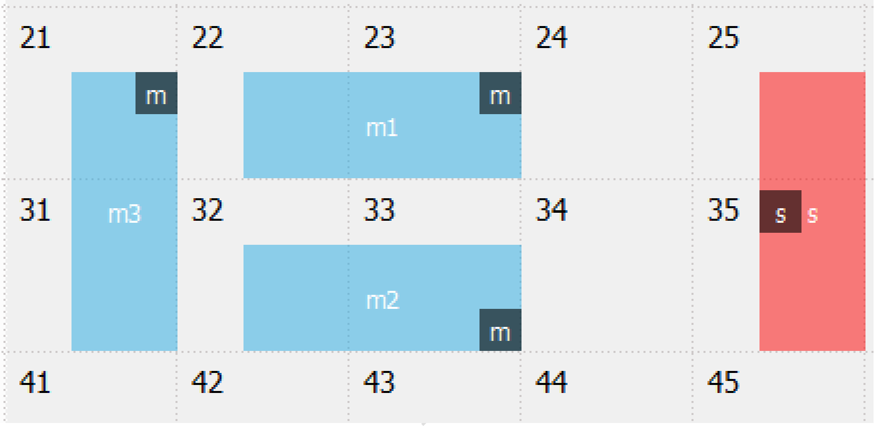
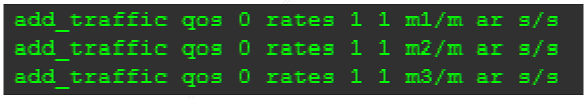


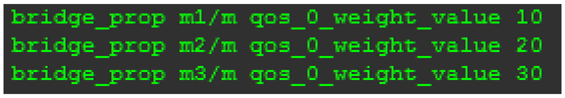
Figure 21: 3 master, 1 slave system

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.



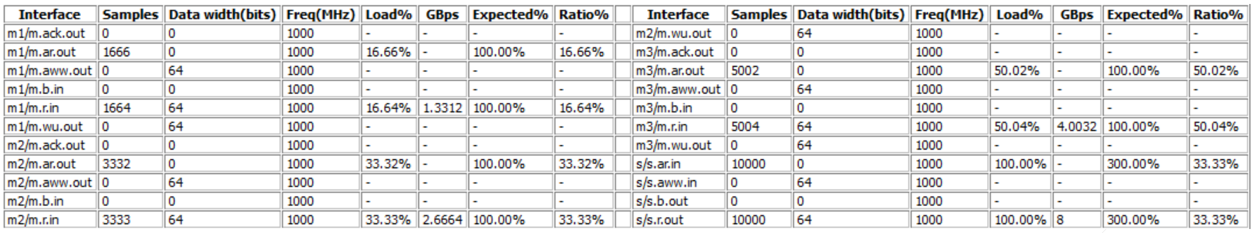
```
bridge_prop m1/m qos_0_weight_value 10
bridge_prop m2/m qos_0_weight_value 20
bridge_prop m3/m qos_0_weight_value 30
```

<u>Example #1</u>

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.

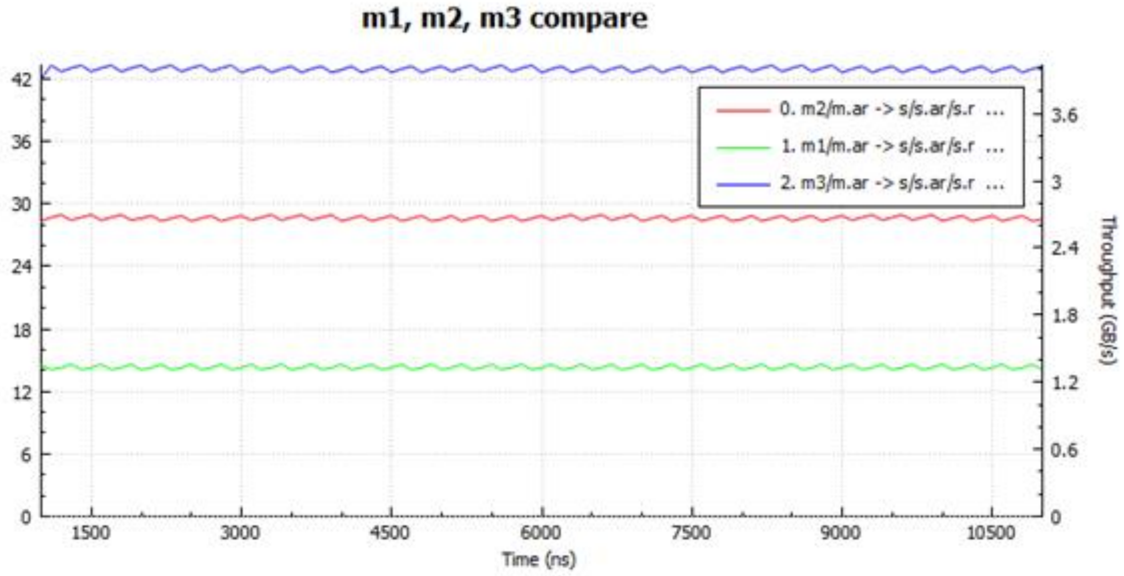


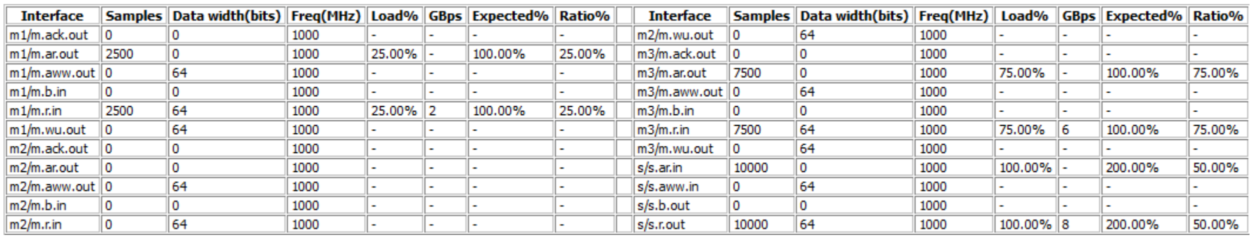
Figure 22: Performance results with ratios 3:2:1

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:

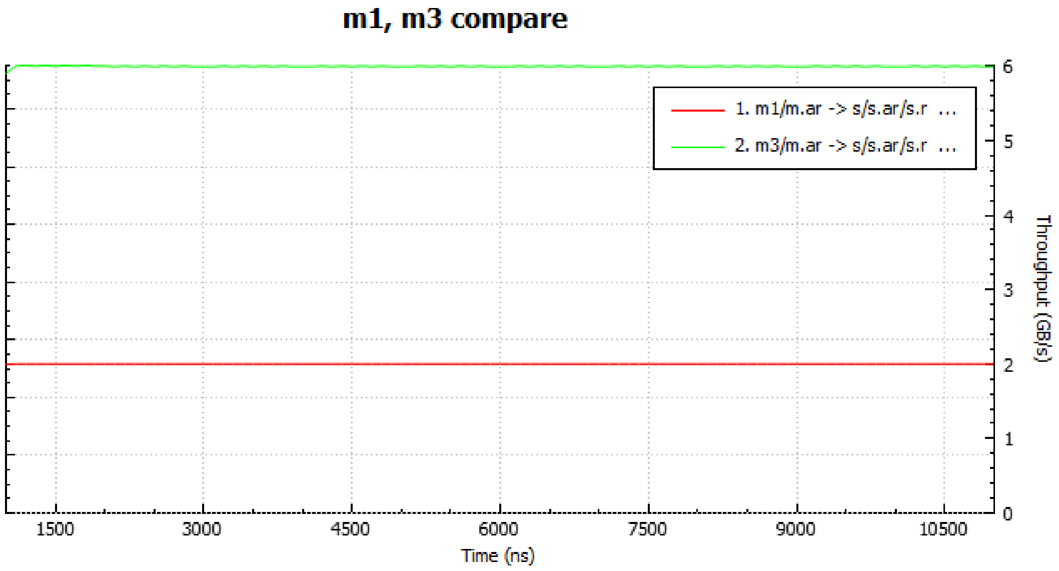
BW at m1/m: BW at m3/m= 10: 30= 1: 3.

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%
m1/m.ack.out	0	0	1000	-	-	-	-
m1/m.ar.out	2500	0	1000	25.00%	-	100.00%	25.00%
m1/m.aww.out	0	64	1000	-	-	-	-
m1/m.b.in	0	0	1000	-	-	-	-
m1/m.r.in	2500	64	1000	25.00%	2	100.00%	25.00%
m1/m.wu.out	0	64	1000	-	-	-	-
m2/m.ack.out	0	0	1000	-	-	-	-
m2/m.ar.out	0	0	1000	-	-	-	-
m2/m.aww.out	0	64	1000	-	-	-	-
m2/m.b.in	0	0	1000	-	-	-	-
m2/m.r.in	0	64	1000	-	-	-	-
m2/m.wu.out	0	64	1000	-	-	-	-
m3/m.ack.out	0	0	1000	-	-	-	-
m3/m.ar.out	7500	0	1000	75.00%	-	100.00%	75.00%
m3/m.aww.out	0	64	1000	-	-	-	-
m3/m.b.in	0	0	1000	-	-	-	-
m3/m.r.in	7500	64	1000	75.00%	6	100.00%	75.00%
m3/m.wu.out	0	64	1000	-	-	-	-
s/s.ar.in	10000	0	1000	100.00%	-	200.00%	50.00%
s/s.aww.in	0	64	1000	-	-	-	-
s/s.b.out	0	0	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.



4.4 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.

4.4.1 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.

4.4.2 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 8-bit therefore the rate granularity is 1/(2^8).

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.

4.4.3 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.

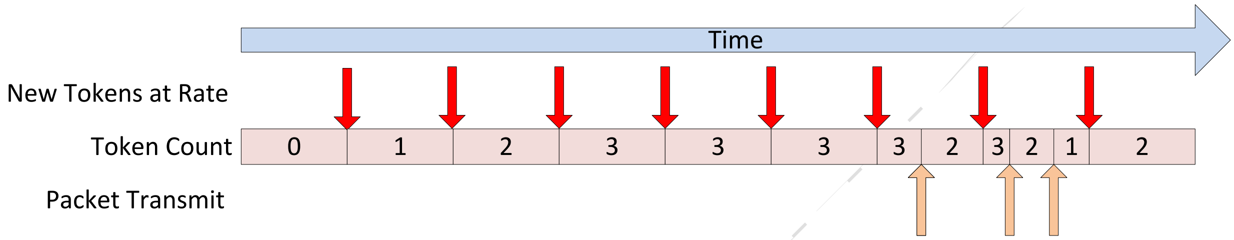


Figure 23 - Token Count increases at specified rate. Packet transmit decrements count.

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.

4.4.4 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 an 8-bit number. The value of the number follows this equation, where N is the programmed value.

rate = $N/(2^8)$

The hardware implements the rate calculation as an 8-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 51(decimal) or 0x33 (hex). When added together 5 times, the value will nearly reach approximately 256 (=2^8), so one packet can be sent every 5 cycles.

4.4.5 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.

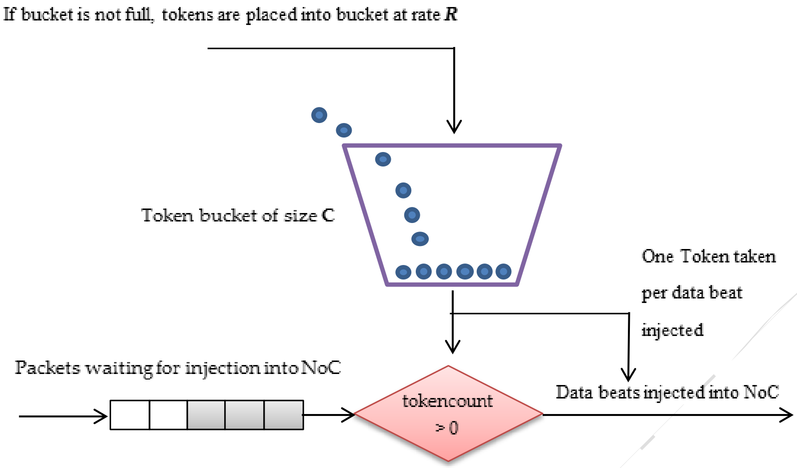

If bucket is not full, tokens are placed into bucket at rate *R*
Token bucket of size C
One Token taken per data beat injected
Packets waiting for injection into NoC
tokencount > 0
Data beats injected into NoC

Figure 24 - 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.

4.4.6 Token Usage

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

4.4.7 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
7:0	Rate Limit Value, for traffic issue to the NoC from the host interface
15:8	Reserved

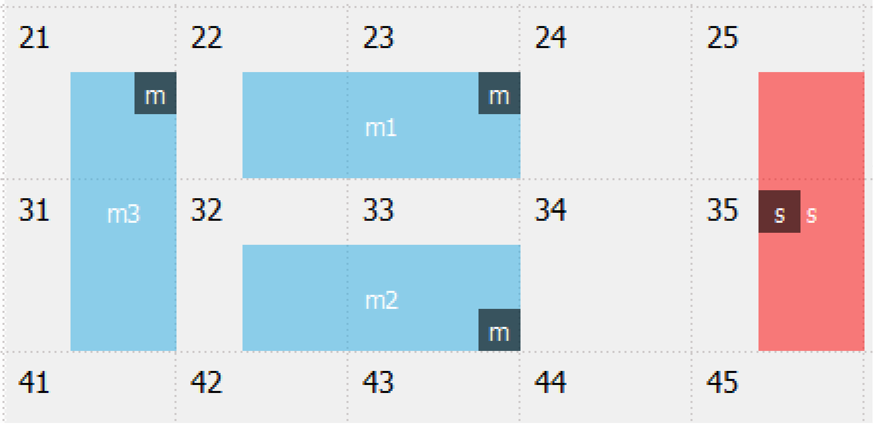
19:16	Bucket Size. This indicates the maximum number of token that may be accumulated at an interface when rate limiters are enabled
20	Rate Limit logic enable. Rate limiter logic is used for arbitration only when the enable bit of the rate limiter configuration register is set to 1

Table 2 - Rate limiter Configuration Register

4.4.8 Examples

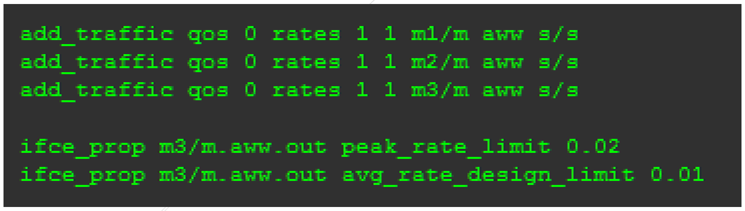
Here are some examples on rate limiting hosts.

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.

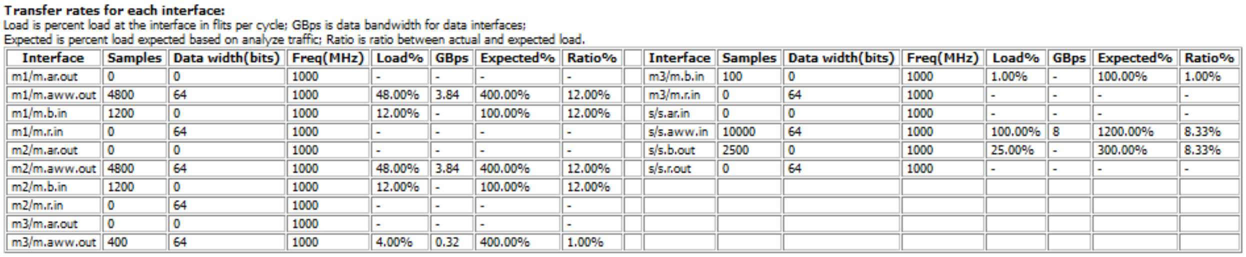
In addition to defining traffics from all masters to slave, let us also define the rate_limit for m3. These are done as follows:



```
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:

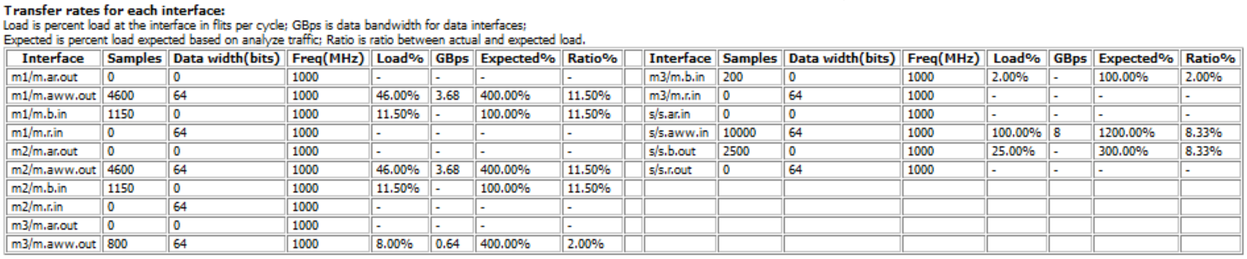


Transfer rates for each interface:
Load is percent load at the interface in flits per cycle; GBps is data bandwidth for data interfaces;
Expected is percent load expected based on analyze traffic; Ratio is ratio between actual and expected load.

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.

4.5 Dynamic Priority Support for Isochronous Traffic

4.5.1 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 priority 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.

4.5.2 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.

5 NoC Serviceability: Regbus Layer

5.1 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.

5.1.1 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 25. 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.

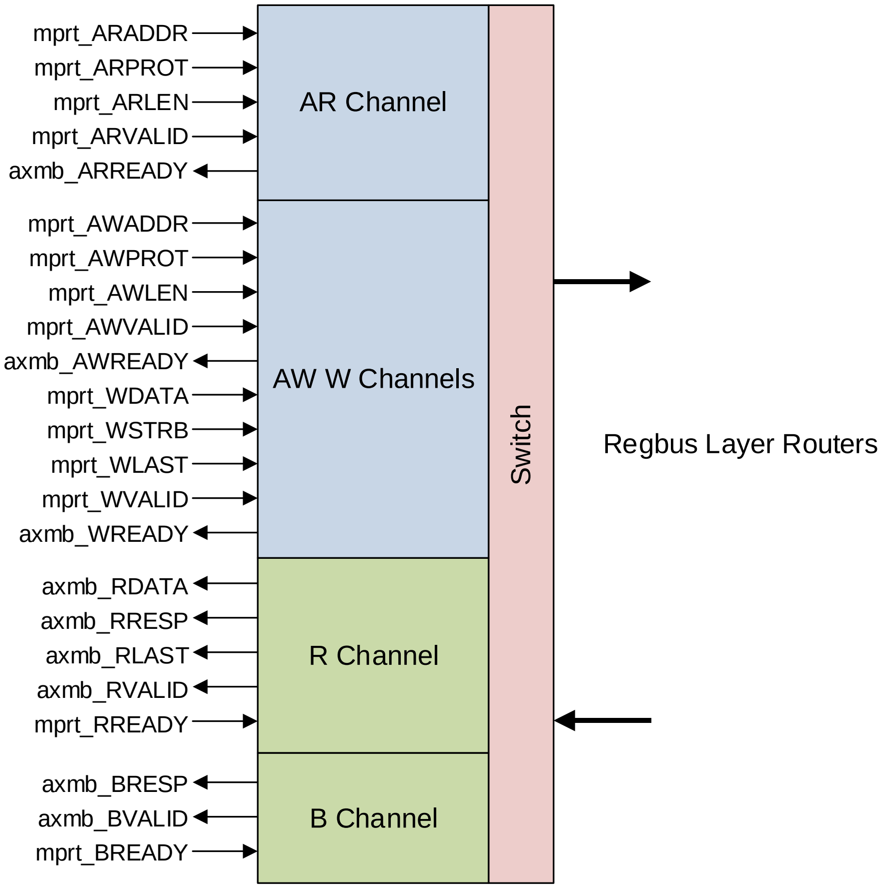


Figure 25. Regbus Master Bridge

5.1.2 The Regbus Layer

As shown in Figure 26, 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.

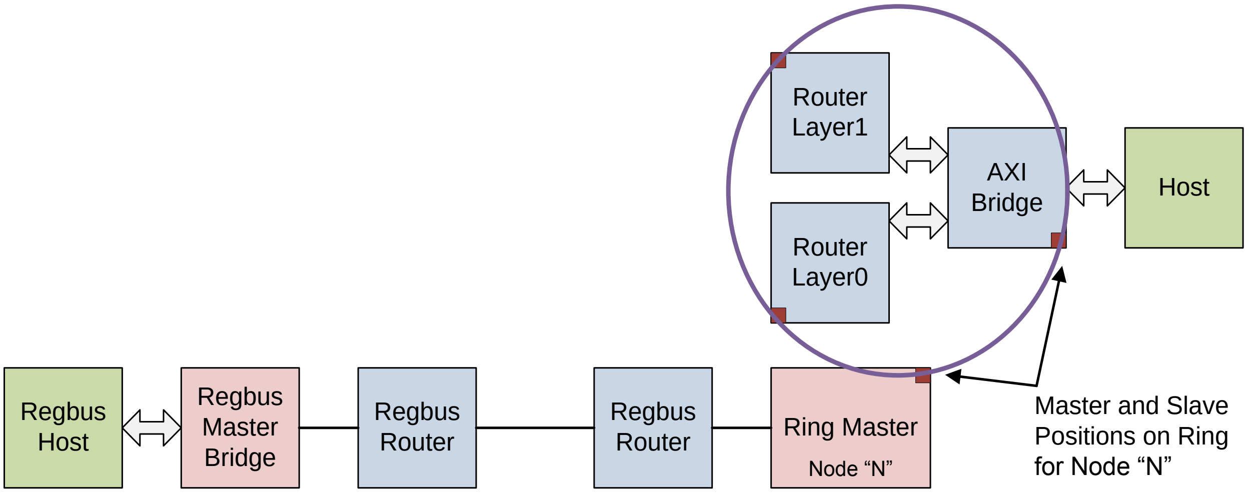


Figure 26. Regbus Layer Communication

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.

5.1.3 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 27 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.

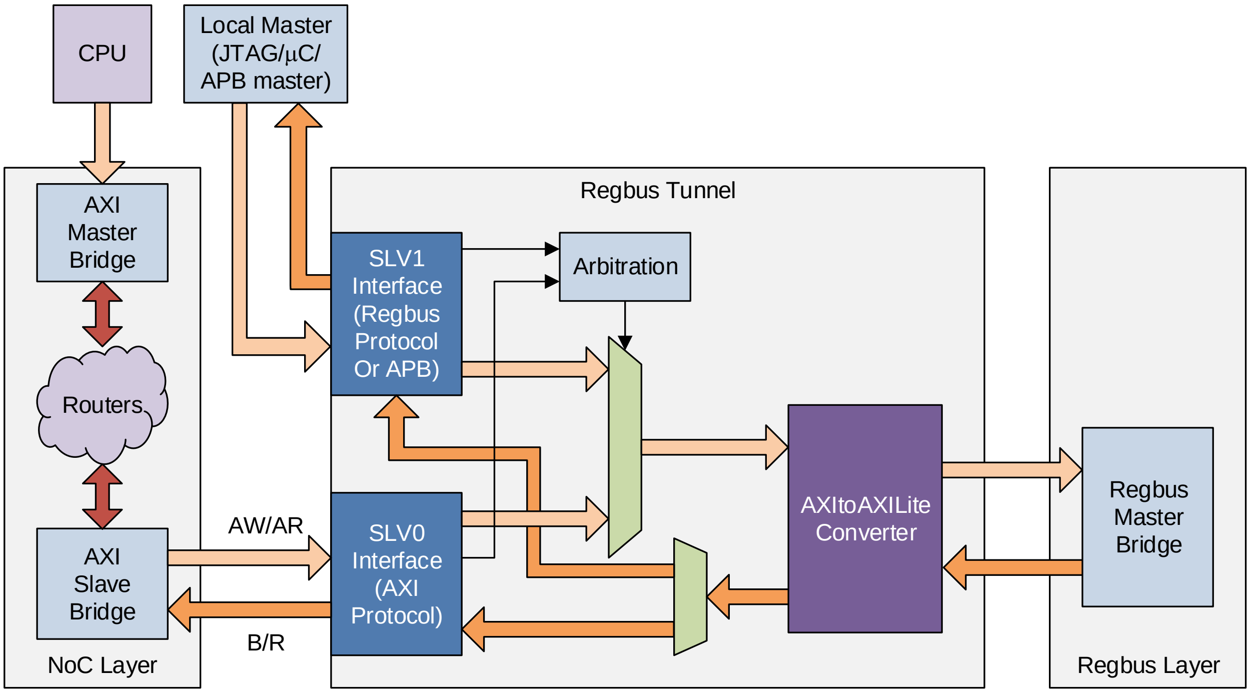


Figure 27. Regbus Tunnel Connects Primary NoC Layer to Regbus Layer

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.

5.1.4 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.

5.1.4.1 *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
```

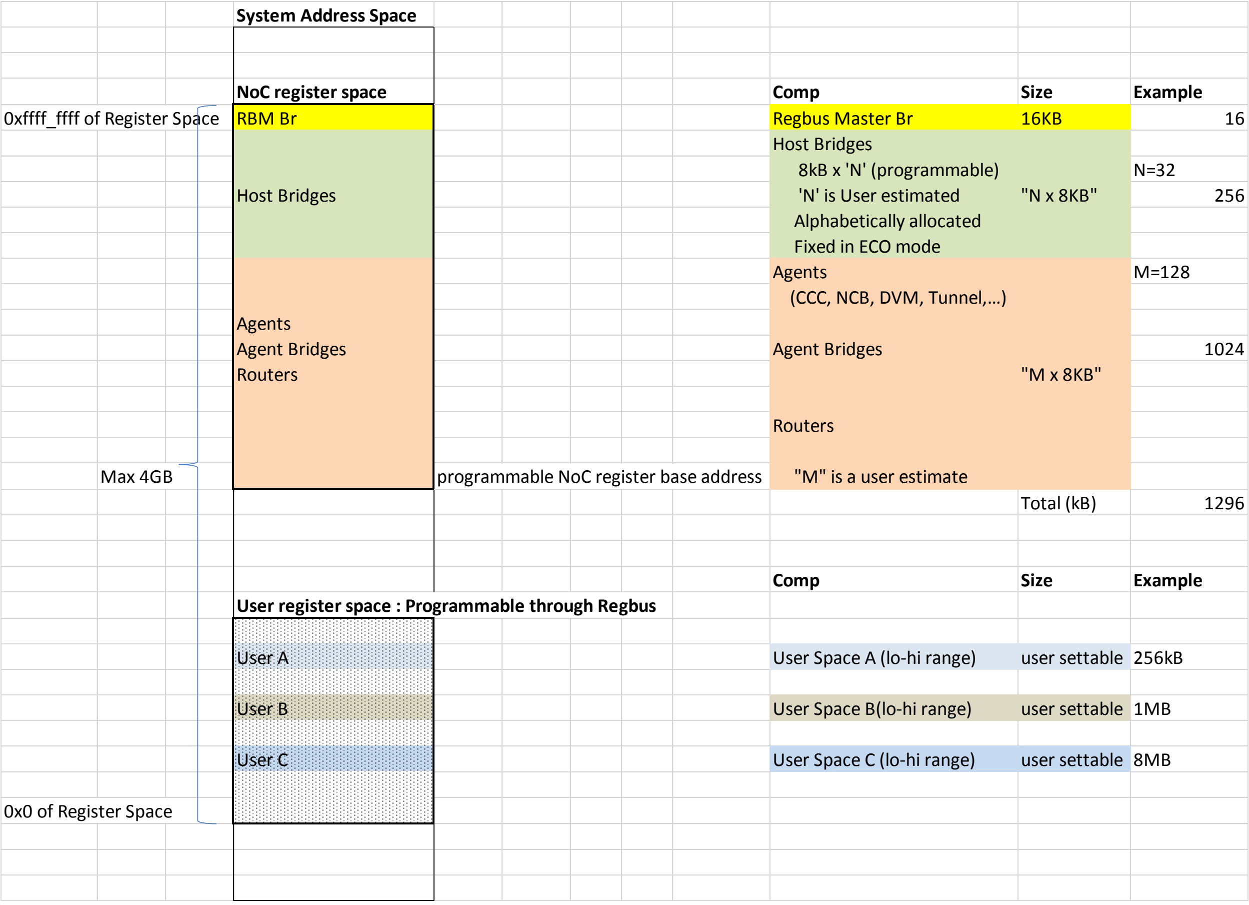
The above command defines the system address space for registers which is accessible through tunnel. This encompasses both the user register space and the NetSpeed NoC register space.

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.



Comp	Size	Example
Regbus Master Br	16KB	16
Host Bridges		
8kB x 'N' (programmable)		N=32
'N' is User estimated	"N x 8KB"	256
Alphabetically allocated		
Fixed in ECO mode		
Agents		M=128
(CCC, NCB, DVM, Tunnel,…)		
Agent Bridges		1024
	"M x 8KB"	
Routers		
"M" is a user estimate		
	Total (kB)	1296

Comp	Size	Example
User Space A (lo-hi range)	user settable	256kB
User Space B(lo-hi range)	user settable	1MB
User Space C (lo-hi range)	user settable	8MB

Figure 28: Regbus Address Map

5.2 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 3: 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 4: 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

5.3 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 5: 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	Decode Error
64-bit access to an address which is aligned to 32-bits	Decode Error
Read access to secure register with AxPROT[1] = 1	No read performed. 0 data and decode error response is returned
Write access to secure register with AxPROT[1] = 1	No write performed. Decode error response is returned
Read/Write access to non-secure register with any AxPROT[1]	Okay

5.4 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 6: 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

5.5 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

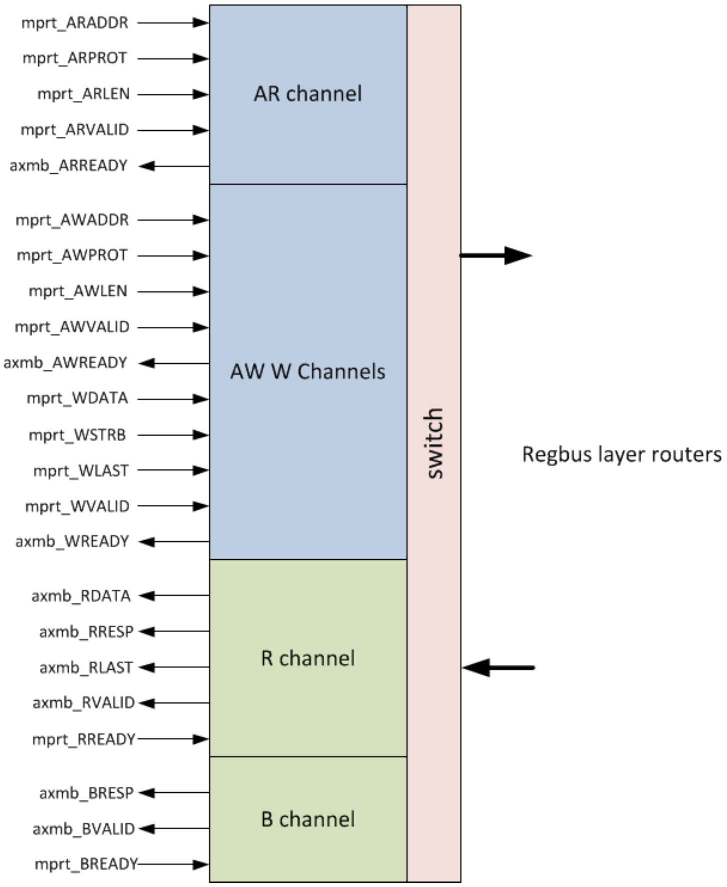


Figure 29: Register bus master bridge

The list of input signals is specified below:

Table 7: 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

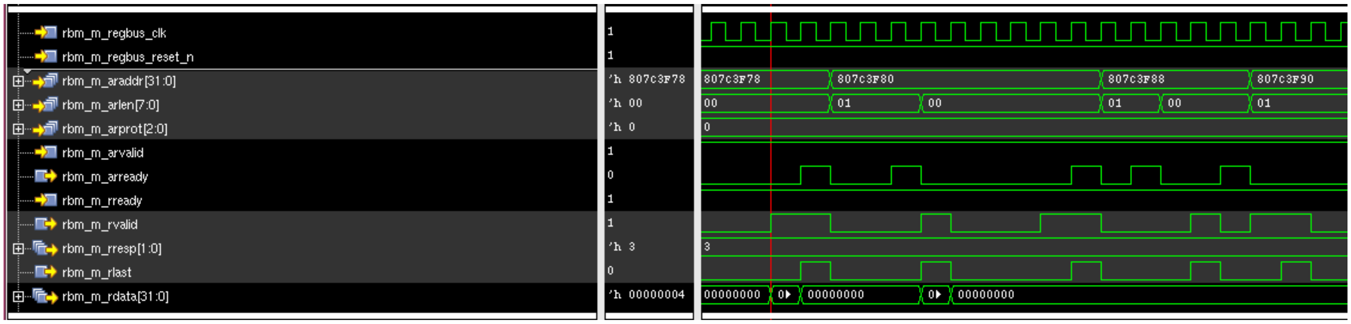


Figure 30: Waveform showing read requests and responses at the register bus master interface

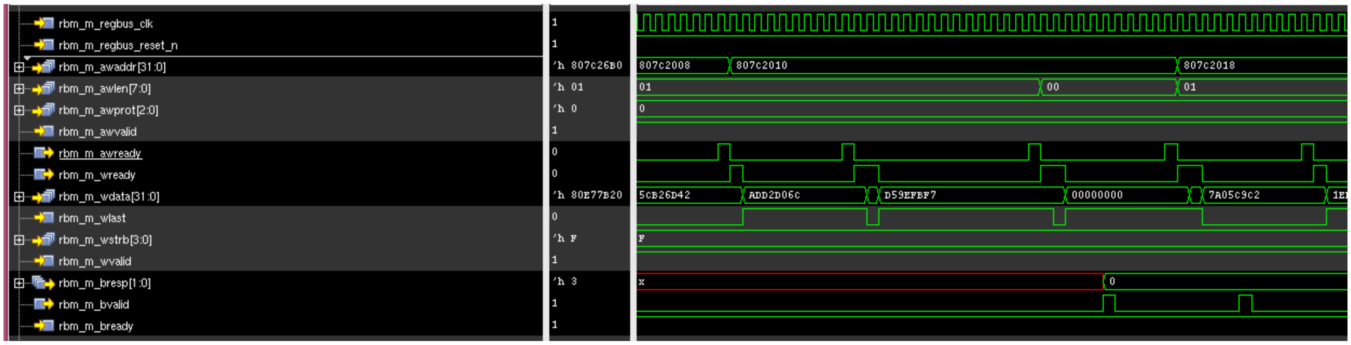


Figure 31: Waveform showing write requests and responses at the register bus master interface

5.6 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.

5.7 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.

5.8 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 8: 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

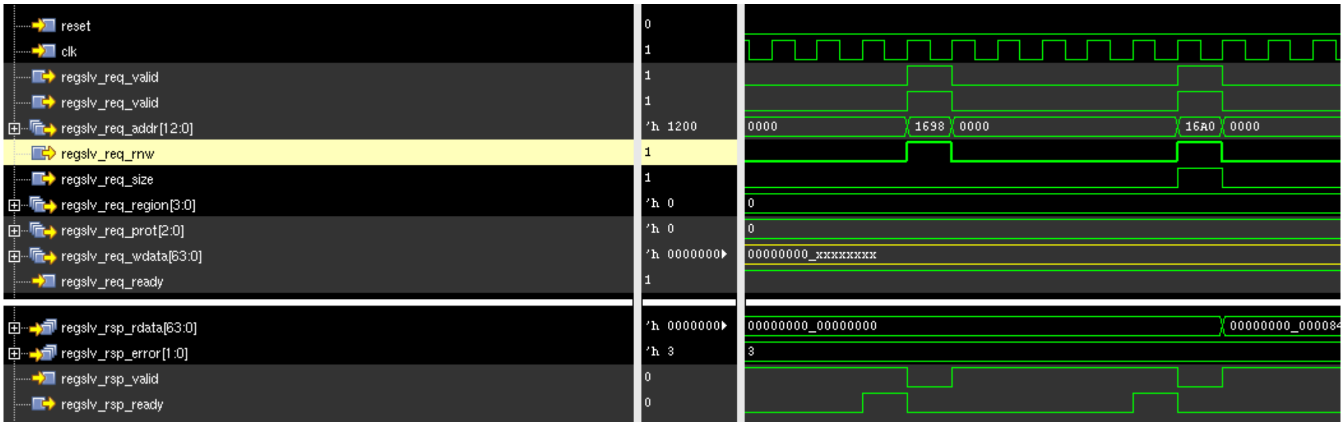


Figure 32 : Waveform showing ring slave read requests and responses (4B and 8B)

Figure 32 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.

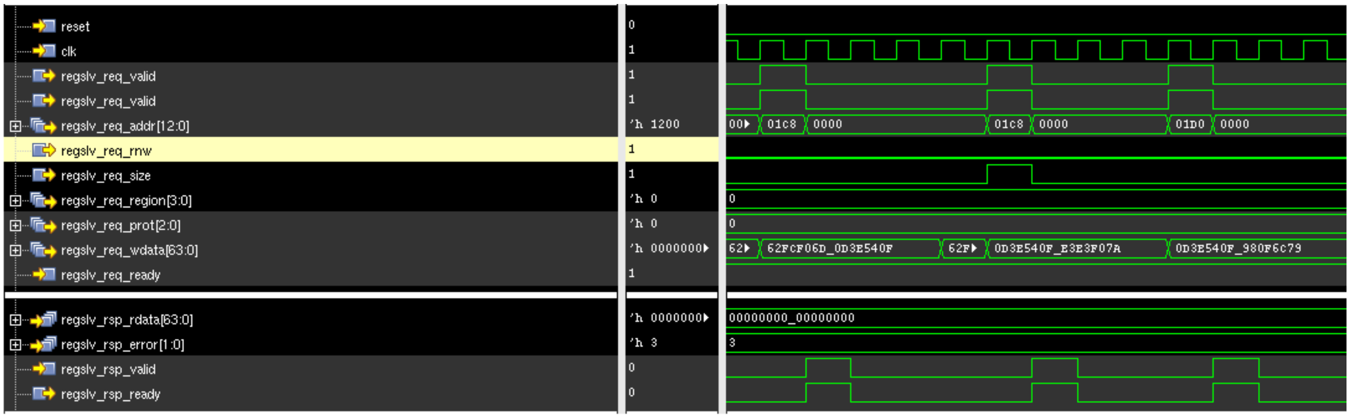


Figure 33 : Waveform showing ring slave write requests and responses (4B and 8B)

Figure 33 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.

5.9 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.

6 Programmers Model

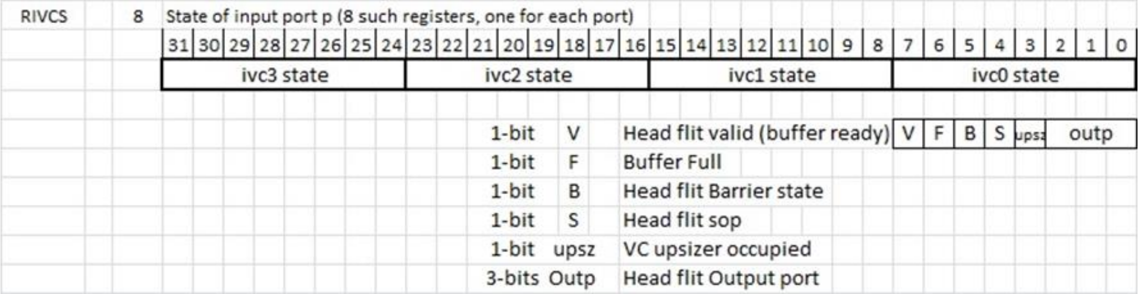
NetSpeed NoC delivers a rich set of registers used for NoC control, debug and performance monitoring of the NoC. This section describes all available registers to SoC designers. The complete list of registers implemented for a specific design can be found in the noc_reference_manual.html under the <project>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

Registers are divided into the following categories:

- Router registers
- Bridge registers
- Regbus registers

6.1 Router Registers

6.1.1 Router Input VC Status (RIVCS)



RIVCS 8 State of input port p (8 such registers, one for each port)

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
ivc3 state								ivc2 state								ivc1 state								ivc0 state							

7	6	5	4	3	2	1	0
V	F	B	S	upsz	outp		

1-bit	V	Head flit valid (buffer ready)
1-bit	F	Buffer Full
1-bit	B	Head flit Barrier state
1-bit	S	Head flit sop
1-bit	upsz	VC upsizer occupied
3-bits	Outp	Head flit Output port

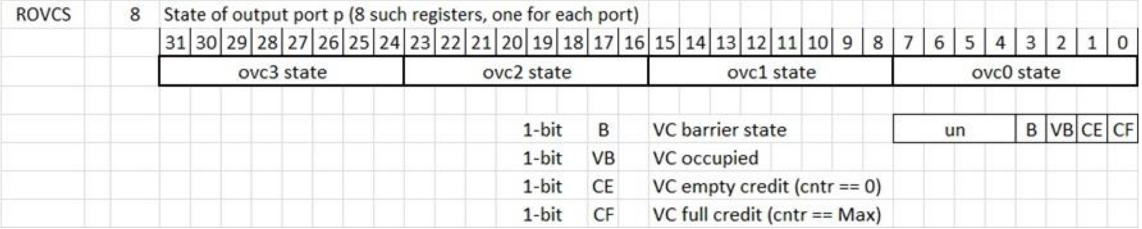
Figure 34 Router Input VC Status (RIVCS) register.

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.

- Head flit valid:
  - 1'b0: Indicates that the VC buffer is empty
  - 1'b1: Indicates that the VC buffer has a flit ready;
- Buffer full:
  - 1'b1: Indicates that the VC buffer is full
  - 1'b0: Indicates that the VC buffer has space to receive flits from the link
- Head flit barrier state:

  - 1'b1: Indicates that the head flit of the VC is of the "QoS Barrier" type
  - 1'b0: Indicates that the head flit of the VC is of the "QoS Normal" type
- Head flit sop:
  - 1'b1: Indicates that the head flit is a start of packet. This also indicates that this input VC 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 has already acquired the VC on the output port
- VC upsizer occupied:
  - 1'b1: Indicates that the flit accumulator on this VC 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
- Head flit output port:
  - Value indicates the router output port to which the packet at the head of the VC 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

6.1.2 Router Output VC Status (ROVCS)



ROVCS 8 State of output port p (8 such registers, one for each port)

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
ovc3 state								ovc2 state								ovc1 state								ovc0 state							

			un	B	VB	CE	CF
1-bit	B	VC barrier state					
1-bit	VB	VC occupied					
1-bit	CE	VC empty credit (cntr == 0)					
1-bit	CF	VC full credit (cntr == Max)					

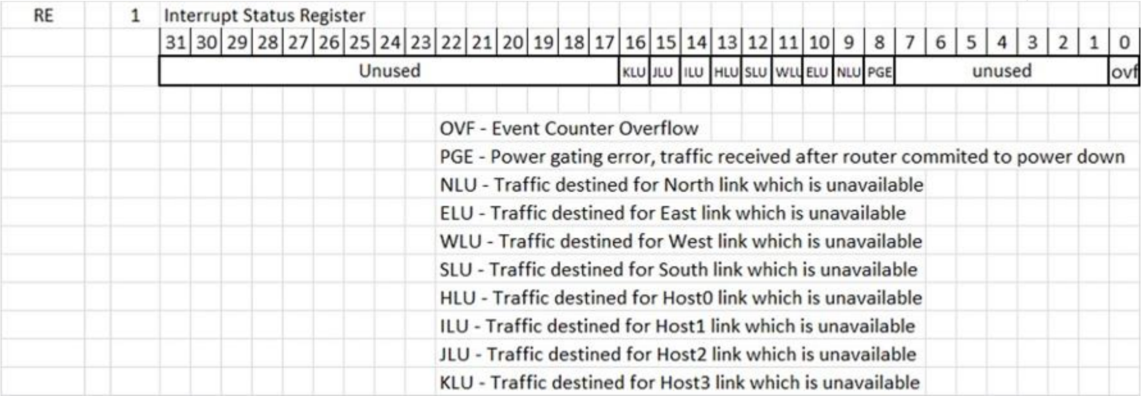
Figure 35 Router Output VC Status (ROVCS) register.

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).

- VC barrier state:
  - 1'b1: Indicates that the output VC is currently in "QoS Barrier" state and will only allow flits of "barrier" type from the corresponding VCs on the input ports
  - 1'b0: Indicates that the output VC is currently in "QoS Normal" state and only allow flits of "normal" type from the corresponding VCs on the input ports;
- VC occupied:
  - 1'b1: Indicates that this output VC is currently locked to the corresponding VC on one of the input ports.

  - 1'b0: Indicates that this output VC is free and can be acquired by the corresponding VC on one of the input port for the transmission of a packet
- VC empty credit:
  - 1'b1: Indicates that this output VC has no credit for transmission of flits to the downstream link
  - 1'b0: Indicates that credit is available for transmission to downstream link
- VC full credit:
  - 1'b1: Indicates that the credit level with this VC is at the maximum provisioned value

6.1.3 Router Event Interrupt Status (RE)



RE 1 Interrupt Status Register

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
Unused															KLU	JLU	ILU	HLU	SLU	WLU	ELU	NLU	PGE	unused							ovf

OVF - Event Counter Overflow
PGE - Power gating error, traffic received after router commited to power down
NLU - Traffic destined for North link which is unavailable
ELU - Traffic destined for East link which is unavailable
WLU - Traffic destined for West link which is unavailable
SLU - Traffic destined for South link which is unavailable
HLU - Traffic destined for Host0 link which is unavailable
ILU - Traffic destined for Host1 link which is unavailable
JLU - Traffic destined for Host2 link which is unavailable
KLU - Traffic destined for Host3 link which is unavailable

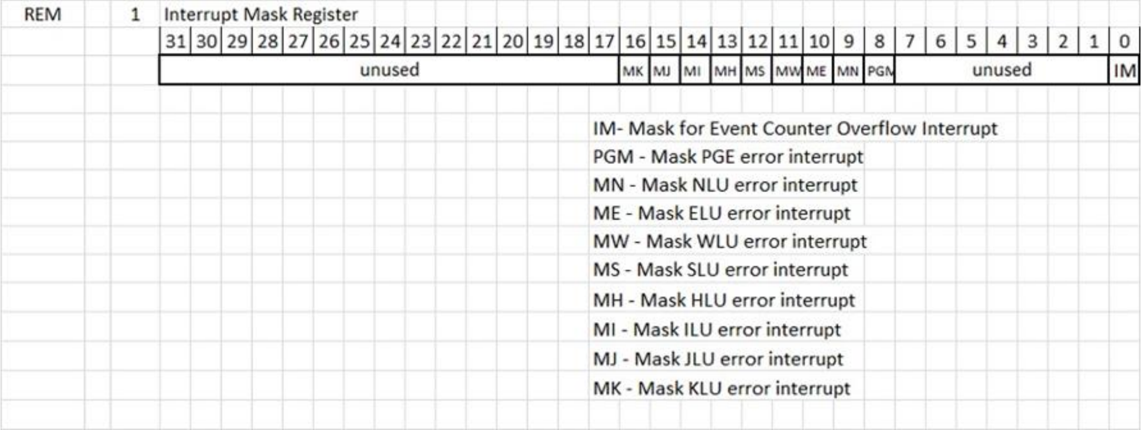
Figure 36 Router Event Interrupt Status (RE) register.

This register tracks the interrupt or error events that can occur in the router. 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.

- OVF:
  - 1'b1 on this status bit indicates that the router event counter has overflowed (32'hFFFFFFFF -> 32'dh0). This is a sticky status bit which must be written with 1'b0 to be cleared.
- PGE:
  - Power gating error, traffic received after router committed to power down.
- NLU:
  - Traffic destined for North link which is unavailable.
- ELU:
  - Traffic destined for East link which is unavailable.
- WLU:
  - Traffic destined for West link which is unavailable.

- SLU:
  - Traffic destined for South link which is unavailable.
- HLU:
  - Traffic destined for Host link which is unavailable.
- ILU:
  - Traffic destined for Host1 link which is unavailable.
- JLU:
  - Traffic destined for Host2 link which is unavailable.
- KLU:
  - Traffic destined for Host3 link which is unavailable.

6.1.4 Router Event Interrupt Mask (REM)



REM 1 Interrupt Mask Register

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
unused															MK	MJ	MI	MH	MS	MW	ME	MN	PGM	unused							IM

IM- Mask for Event Counter Overflow Interrupt
PGM - Mask PGE error interrupt
MN - Mask NLU error interrupt
ME - Mask ELU error interrupt
MW - Mask WLU error interrupt
MS - Mask SLU error interrupt
MH - Mask HLU error interrupt
MI - Mask ILU error interrupt
MJ - Mask JLU error interrupt
MK - Mask KLU error interrupt

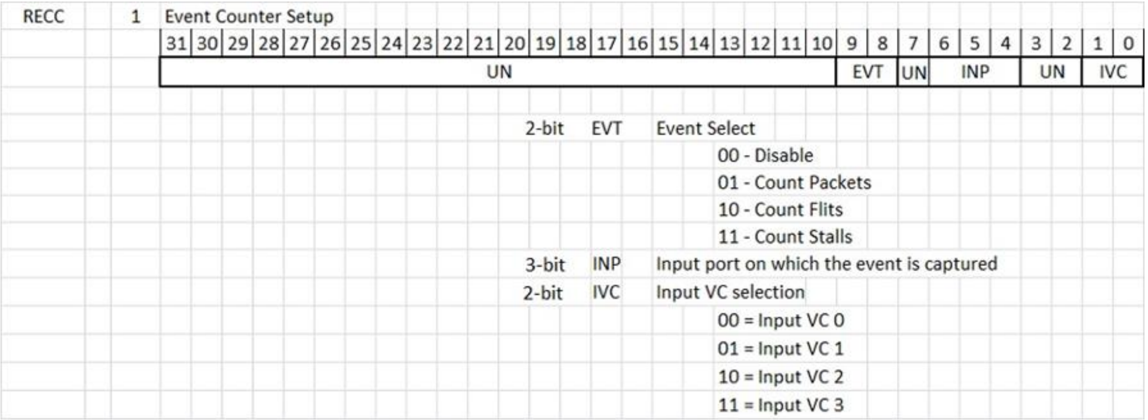
Figure 37 Router Event Interrupt Mask (REM) register.

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.

- IM:
  - 1'b1: Masks or disables an interrupt from being generated by the event count overflow status bit (RE)
  - 1'b0: Enables an interrupt to be generated when event counter status bit is set
- PGM:
  - Mask PGE error interrupt.
- MN:
  - Mask NLU error interrupt.
- ME:
  - Mask ELU error interrupt.

- MW:
    - Mask WLU error interrupt.
- MS:
    - Mask SLU error interrupt.
- MH:
    - Mask HLU error interrupt.
- MI:
    - Mask ILU error interrupt.
- MJ:
    - Mask JLU error interrupt.
- MK:
    - Mask KLU error interrupt.

6.1.5 Router Event Counter Control (RECC)



RECC	1	Event Counter Setup

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
UN																						EVT		UN	INP			UN		IVC	

2-bit	EVT	Event Select	
			00 - Disable
			01 - Count Packets
			10 - Count Flits
			11 - Count Stalls
3-bit	INP	Input port on which the event is captured	
2-bit	IVC	Input VC selection	
			00 = Input VC 0
			01 = Input VC 1
			10 = Input VC 2
			11 = Input VC 3

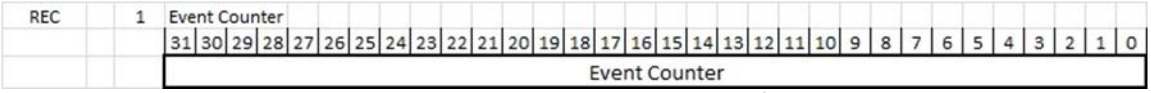
Figure 38 Router event counter control (RECC) register.

This register is used to select which hardware events will increment the event counter. This is a read-write register.

- EVT:
    - Selects an event for the counter
    - 2'b00: Counter disabled (default)
    - 2'b01: 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
    - 2'b10: 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

-
   - 2'b11: Generates count event when VC buffer is full for any selected VC on the selected input port
- INP:
   - Specified the input router port on which the events are captured. Only one input port can be used for counting events at one time.
   - 3'd0: N, 3'd1: E, 3'd2: W, 3'd3: S, 3'd4: H, 3'd5: I, 3'd6: J, 3'd7: K
- IVC:
   - Selects the VC for event capture
   - 2'b00: Input VC0
   - 2'b01: Input VC1
   - 2'b10: Input VC2
   - 2'b11: Input VC3

6.1.6 Router Event Counter (REC)



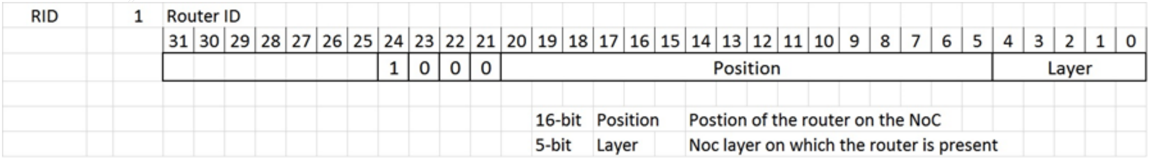
REC	1	Event Counter																															
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Event Counter																															

Figure 39 Router Event Counter (REC) register.

This register holds the event counter. It is a read-write register. 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.

- Event Counter:
   - 32'bit event incrementing counter. Rollover from 32'hFFFFFFFF -> 32'd0 sets the rollover status bit RE.

6.1.7 Router ID (RID)



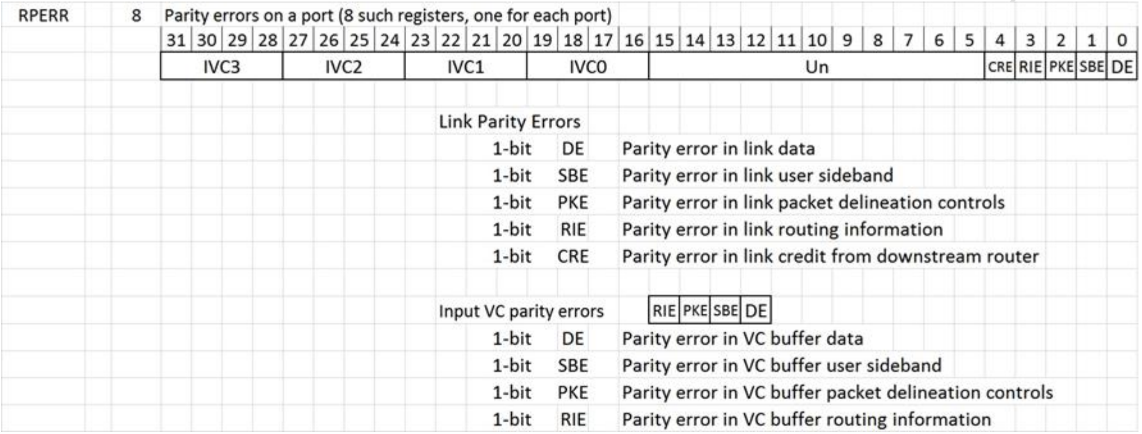
RID	1	Router ID																															
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									1	0	0	0	Position																Layer				
														16-bit		Position		Postion of the router on the NoC															
														5-bit		Layer		Noc layer on which the router is present															

Figure 40 Router ID (RID) register.

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.

- Layer:
  - 5-bit identifier of the NoC layer on which this router is located.
- Position:
  - 16-bit position ID of this router in the NoC.

6.1.8 Router Parity Error (RPERR)



RPERR 8 Parity errors on a port (8 such registers, one for each port)

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
IVC3				IVC2				IVC1				IVC0				Un											CRE	RIE	PKE	SBE	DE

Link Parity Errors

1-bit	DE	Parity error in link data
1-bit	SBE	Parity error in link user sideband
1-bit	PKE	Parity error in link packet delineation controls
1-bit	RIE	Parity error in link routing information
1-bit	CRE	Parity error in link credit from downstream router

Input VC parity errors

RIE	PKE	SBE	DE

1-bit	DE	Parity error in VC buffer data
1-bit	SBE	Parity error in VC buffer user sideband
1-bit	PKE	Parity error in VC buffer packet delineation controls
1-bit	RIE	Parity error in VC buffer routing information

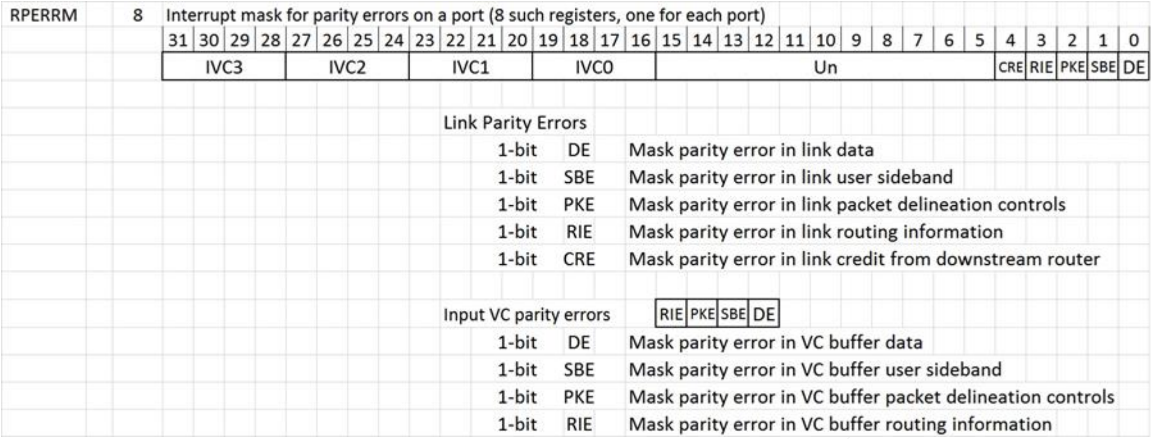
Figure 41 Router Parity Error (RPERR) register.

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.

This is a wzc register.

6.1.9 Router Mask Parity Error (RPERRM)



RPERRM		8	Interrupt mask for parity errors on a port (8 such registers, one for each port)																														
			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
			IVC3				IVC2				IVC1				IVC0				Un											CRE	RIE	PKE	SBE	DE

Link Parity Errors

1-bit	DE	Mask parity error in link data
1-bit	SBE	Mask parity error in link user sideband
1-bit	PKE	Mask parity error in link packet delineation controls
1-bit	RIE	Mask parity error in link routing information
1-bit	CRE	Mask parity error in link credit from downstream router

Input VC parity errors

RIE	PKE	SBE	DE

1-bit	DE	Mask parity error in VC buffer data
1-bit	SBE	Mask parity error in VC buffer user sideband
1-bit	PKE	Mask parity error in VC buffer packet delineation controls
1-bit	RIE	Mask parity error in VC buffer routing information

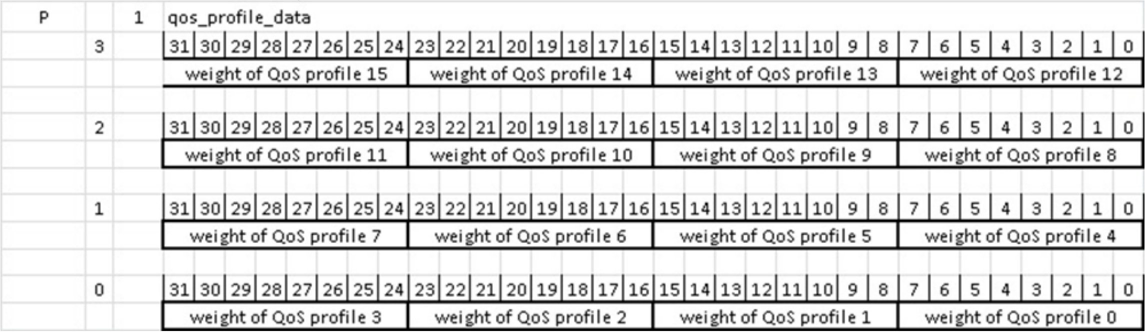
Figure 42 Router Mask Parity Error (RPERR) register.

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

This is a read-write register.

6.2 BRIDGE REGISTERS

6.2.1 QoS Profile Data (P)



P		1	qos_profile_data																															
	3		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
			weight of QoS profile 15								weight of QoS profile 14								weight of QoS profile 13								weight of QoS profile 12							
	2		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
			weight of QoS profile 11								weight of QoS profile 10								weight of QoS profile 9								weight of QoS profile 8							
	1		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
			weight of QoS profile 7								weight of QoS profile 6								weight of QoS profile 5								weight of QoS profile 4							
	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
			weight of QoS profile 3								weight of QoS profile 2								weight of QoS profile 1								weight of QoS profile 0							

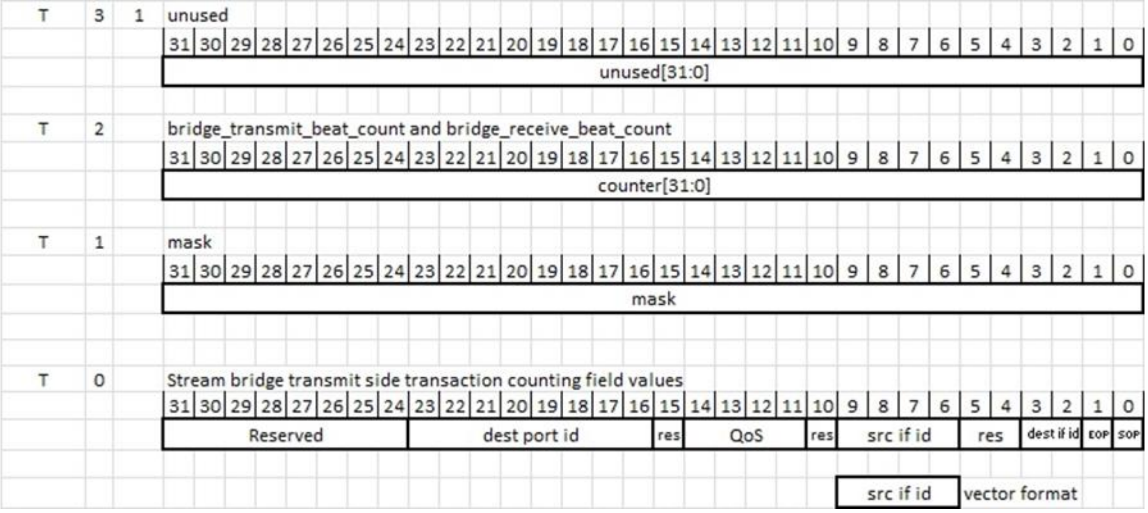
Figure 43 QoS Profile Data (P) register.

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 priority 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.

Reset value: 0x03030303

6.2.2 Bridge Transmit Event Counter Control (T)



T 3 1 unused

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
unused[31:0]																															

T 2 bridge_transmit_beat_count and bridge_receive_beat_count

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
counter[31:0]																															

T 1 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																															

T 0 Stream bridge transmit side transaction counting field values

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
Reserved								dest port id								res	QoS				res	src if id				res		dest if id		EOP	SOP
																						src if id				vector format					

Figure 44 Bridge Transmit Event Counter Control (T) register.

The event counter control registers can be used to count performance or debug events in the transmit portion of the streaming bridge. This is the portion of the bridge that accepts packets from the host and sends it to the NoC. There are 4 register in the event counter control register set.

Register 3 is currently unused.

Register 2 is the event counter itself. It is 32 bits wide. Whenever a selected event occurs in hardware, the event counter will increment. This register defaults to zero. It can be read at any time, and can be written to any value. When the counter reaches its peak value, it will roll over to zero and continue counting. The rollover condition can be set up to trigger an interrupt.

Register 1 is a mask register. It defaults to zero, meaning no events are counted by default. To enable counting, the user can write 1s to the appropriate bits.

The mask is used in combination with Register 0 to select the event(s) to count. Register 0 can be programmed with comparison fields. Register 1 mask will determine which bits to compare. For instance, to count on SOPs seen, Register 0 and 1 should both be set to 0x1.

The event is determined by a bit-wise comparison. The more mask bits used, the less likely a comparison will match and an event will be counted. If it is just looking for SOP, it will count the number of packets. If it looks for SOP and EOP, it will count the number of single-flit packet.

Partial fields can be compared by setting only some of the mask bits for a field. Packets with QoS values of 8-15 could be set by marking only one mask bit corresponding to the QoS bit 3.

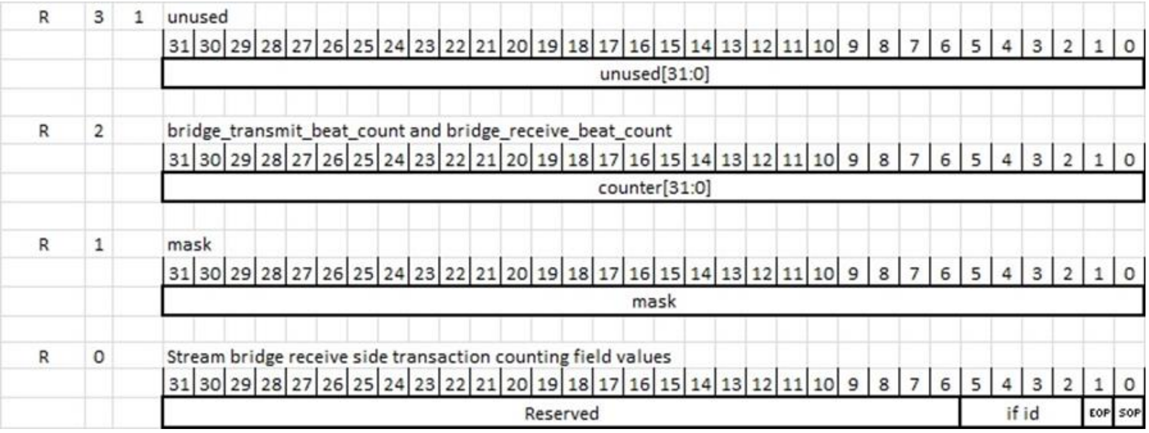
Reset value for Register 0: 0x00000000

Reset value for Register 1: 0x00000000

Reset value for Register 2: 0x00000000

Reset value for Register 3: 0x00000000

6.2.3 Bridge Receive Event Counter Control (R)



R	3	1	unused																															
			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
			unused[31:0]																															
R	2		bridge_transmit_beat_count and bridge_receive_beat_count																															
			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
			counter[31:0]																															
R	1		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																															
R	0		Stream bridge receive side transaction counting field values																															
			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
			Reserved																												if id				EOP	SOP

Figure 45 Bridge Receive Event Counter Control (R) register.

The event counter control registers can be used to count performance or debug events in the receive section of the streaming bridge. This is the portion of the bridge that accepts packets from the NoC and sends it to the host. There are 4 register in the event counter control register set.

Register 3 is currently unused.

Register 2 is the even counter itself. It is 64 bits wide. Whenever a selected event occurs in hardware, the event counter will increment. This register defaults to zero. It can be read at any time, and can be written to any value. When the counter reaches its peak value, it will roll over to zero and continue counting. The rollover condition can be set up to trigger an interrupt.

Register 1 is a mask register. It defaults to zero, meaning no events are counted by default. To enable counting, the user can write 1s to the appropriate bits.

The mask is used in combination with Register 0 to select the event(s) to count. Register 0 can be programmed with comparison fields. Register 1 mask will determine which bits to compare. For instance, to count on SOPs seen, Register 0 and 1 should both be set to 0x1.

The event is determined by a bit-wise comparison. The more mask bits used, the less likely a comparison will match and an event will be counted. If it is just looking for SOP, it will count the number of packets. If it looks for SOP and EOP, it will count the number of single-flit packet.

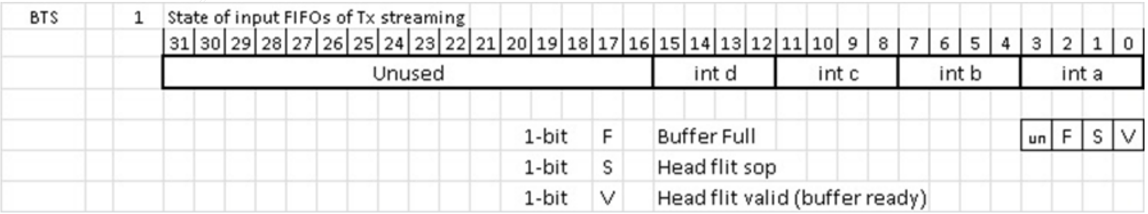
Reset value for Register 0: 0x00000000

Reset value for Register 1: 0x00000000

Reset value for Register 2: 0x0000000000000000

Reset value for Register 3: 0x00000000

6.2.4 Bridge Transmit FIFO Status (BTS)



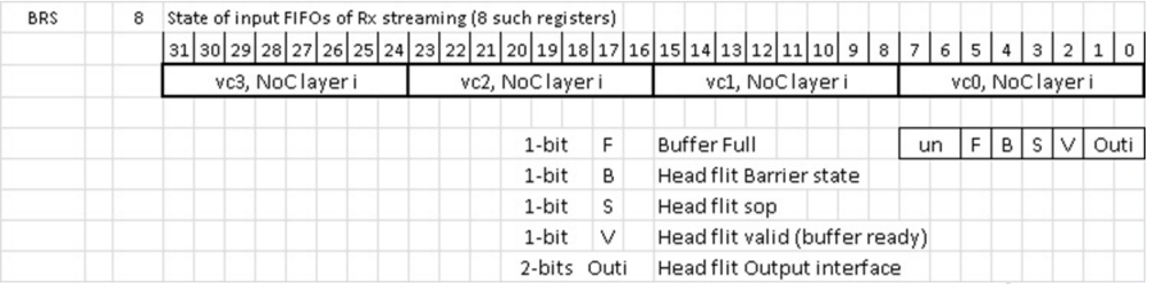
BTS	1	State of input FIFOs of Tx streaming																													
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Unused																int d				int c				int b				int a			
														1-bit		F		Buffer Full												un	F	S	V
														1-bit		S		Head flit sop															
														1-bit		V		Head flit valid (buffer ready)															

Figure 46 Bridge Transmit FIFO Status (BTS) register.

This register tracks the status of the bridge transmit FIFOs. There are up to 4 FIFOs, with one per interface in the streaming bridge. This is a read-only register.

Reset value: 0x00000000

6.2.5 Bridge Receive FIFO Status (BRS)



BRS 8 State of input FIFOs of Rx streaming (8 such registers)

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
vc3, NoC layer i								vc2, NoC layer i								vc1, NoC layer i								vc0, NoC layer i							
																								un		F	B	S	V	Outi	

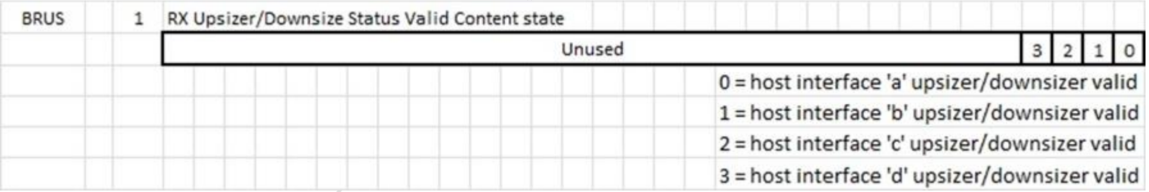
1-bit	F	Buffer Full
1-bit	B	Head flit Barrier state
1-bit	S	Head flit sop
1-bit	V	Head flit valid (buffer ready)
2-bits	Outi	Head flit Output interface

Figure 47 Bridge Receive FIFO Status (BRS) register.

These registers track the status of the bridges 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. This is a read-only register.

Reset value: 0x00000000

6.2.6 Bridge Rx Upsizer Status (BRUS)



BRUS 1 RX Upsizer/Downsize Status Valid Content state

Unused	3	2	1	0

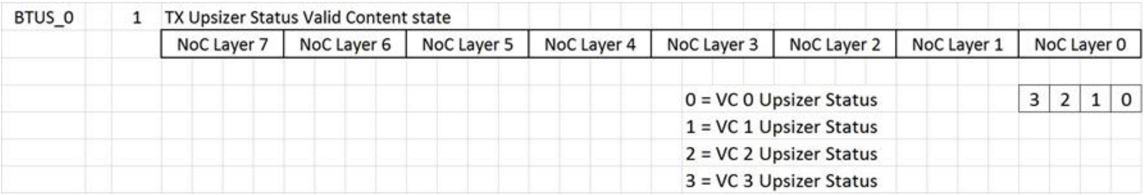
0 = host interface 'a' upsizer/downsizer valid
1 = host interface 'b' upsizer/downsizer valid
2 = host interface 'c' upsizer/downsizer valid
3 = host interface 'd' upsizer/downsizer valid

Figure 48 Bridge Rx Upsizer Status (BRUS) register.

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. This is a read-only register.

Reset value: 0x00000000

6.2.7 Bridge Tx Upsizer Status (BTUS)



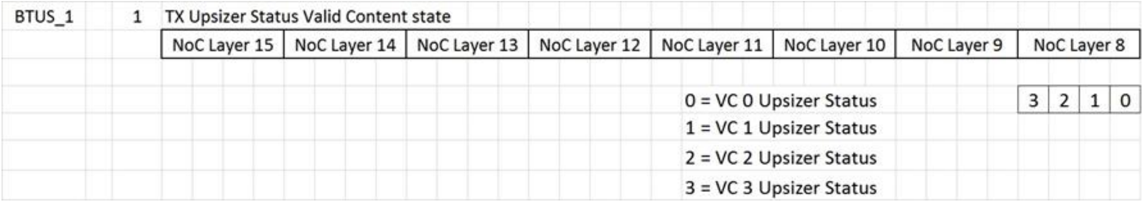
BTUS_0 1 TX Upsizer Status Valid Content state

NoC Layer 7	NoC Layer 6	NoC Layer 5	NoC Layer 4	NoC Layer 3	NoC Layer 2	NoC Layer 1	NoC Layer 0

3	2	1	0

0 = VC 0 Upsizer Status
1 = VC 1 Upsizer Status
2 = VC 2 Upsizer Status
3 = VC 3 Upsizer Status

Figure 49 Bridge Tx Upsizer Status (BTUS_0) register.



BTUS_1 1 TX Upsizer Status Valid Content state

NoC Layer 15	NoC Layer 14	NoC Layer 13	NoC Layer 12	NoC Layer 11	NoC Layer 10	NoC Layer 9	NoC Layer 8

3	2	1	0

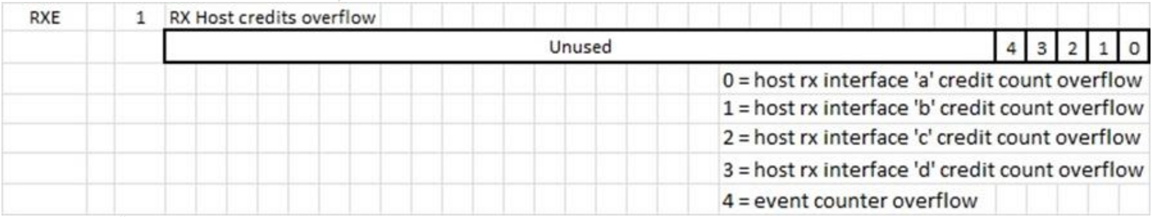
0 = VC 0 Upsizer Status
1 = VC 1 Upsizer Status
2 = VC 2 Upsizer Status
3 = VC 3 Upsizer Status

Figure 50 Bridge Tx Upsizer Status (BTUS_1) register.

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). They are read-only registers.

Reset value: 0x00000000

6.2.8 Receive Interrupt Status (RXE)



RXE 1 RX Host credits overflow

Unused	4	3	2	1	0

0 = host rx interface 'a' credit count overflow
1 = host rx interface 'b' credit count overflow
2 = host rx interface 'c' credit count overflow
3 = host rx interface 'd' credit count overflow
4 = event counter overflow

Figure 51 Receive Interrupt Status (RXE) register.

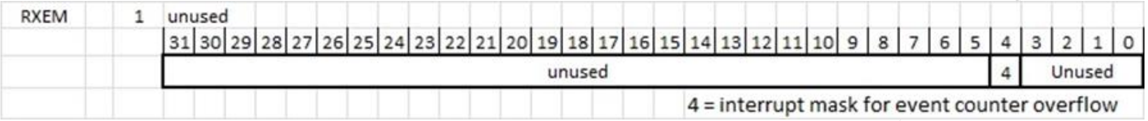
This register tracks the interrupt events in the receive portion of the streaming bridge. It resets to 0, but as these conditions occur, the corresponding bits are set to 1. This register can be read and can also be cleared by sending a write with bits set to 0 for the bits that should be cleared.

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.

Bit 4: Sets if the event counter overflows. This event can be masked so that no interrupt is sent on an overflow condition.

Reset value: 0x00000000

6.2.9 Receive Interrupt Mask (RXEM)



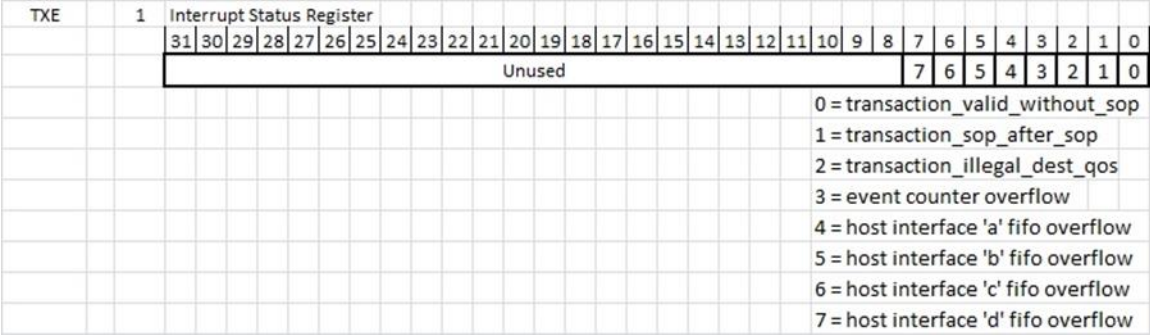
RXEM	1	unused																															
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		unused																											4	Unused			
																			4 = interrupt mask for event counter overflow														

Figure 52 Receive Interrupt Mask (RXEM) register.

This register is used to decide which of the error/interrupt events specified in the Receive 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. When bit 4 is set to 1, the corresponding interrupt event will not send an interrupt to the system.

Reset value: 0x00000010

6.2.10 Transmit Interrupt Status (TXE)



TXE	1	Interrupt Status Register																															
		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
		Unused																								7	6	5	4	3	2	1	0
																							0 = transaction_valid_without_sop										
																							1 = transaction_sop_after_sop										
																							2 = transaction_illegal_dest_qos										
																							3 = event counter overflow										
																							4 = host interface 'a' fifo overflow										
																							5 = host interface 'b' fifo overflow										
																							6 = host interface 'c' fifo overflow										
																							7 = host interface 'd' fifo overflow										

Figure 53 Transmit Interrupt Status (TXE) register.

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.

Bit 0: Sets if a transaction is initiated w/o SOP. This event will always trigger an interrupt and cannot be masked.

Bit 1: Sets if a SOP is received after SOP. This event will always trigger an interrupt and cannot be masked.

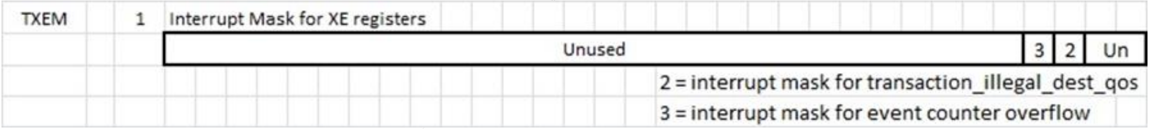
Bit 2: Sets if a transaction is received from bridge for which there is no entry present in the VC map, i.e. the destination and/or QoS is not supported. This is a decode error. This event can be masked to not send an interrupt, but the packet will be dropped in the bridge.

Bit 3: Sets if the event counter overflows. This event can be masked so that no interrupt is sent on an overflow condition.

Bit 4-7: 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.

Reset value: 0x00000000

6.2.11 Transmit Interrupt Mask (TXEM)



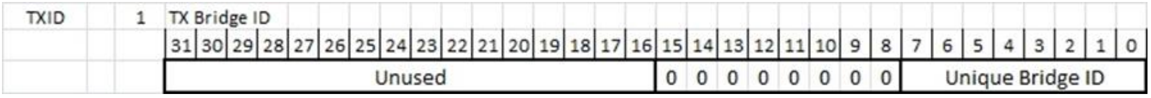
TXEM	1	Interrupt Mask for XE registers			
		Unused	3	2	Un
		2 = interrupt mask for transaction_illegal_dest_qos			
		3 = interrupt mask for event counter overflow			

Figure 54 Transmit Interrupt Mask (TXEM) register.

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.

Reset value: 0x00000008

6.2.12 Tx Bridge ID (TXID)



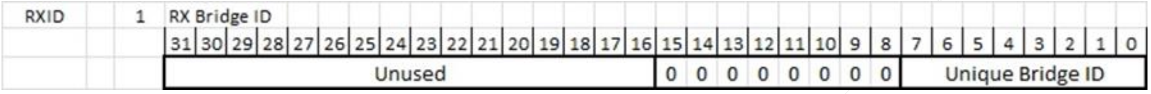
TXID	1	TX Bridge ID																																
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Unused																0	0	0	0	0	0	0	0	Unique Bridge ID							

Figure 55 Tx Bridge ID (TXID) register.

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.

- Unique bridge ID:
  - 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.

6.2.13 Rx Bridge ID (RXID)



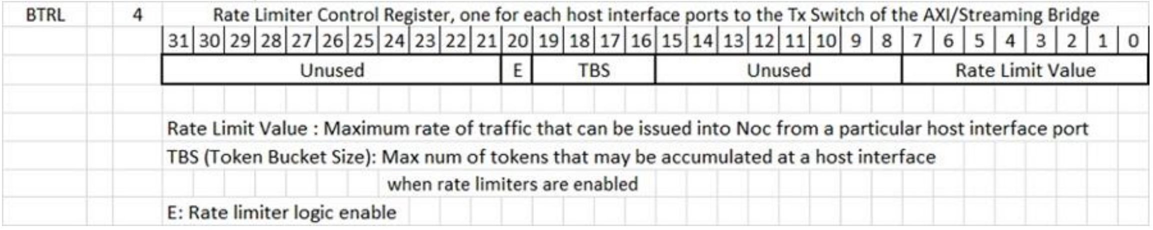
RXID	1	RX Bridge ID																															
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		Unused																0	0	0	0	0	0	0	0	Unique Bridge ID							

Figure 56. Rx Bridge ID (RXID) register.

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.

- Unique bridge ID:
  - 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 Tx side of the bridge.

6.2.14 Streaming TX Rate Limiter (BTRL)



BTRL	4	Rate Limiter Control Register, one for each host interface ports to the Tx Switch of the AXI/Streaming 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
		Unused											E	TBS				Unused								Rate Limit Value							

Rate Limit Value : Maximum rate of traffic that can be issued into Noc from a particular host interface port

TBS (Token Bucket Size): Max num of tokens that may be accumulated at a host interface when rate limiters are enabled

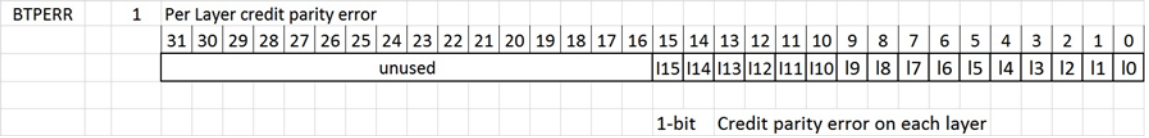
E: Rate limiter logic enable

Figure 57 Streaming TX Rate Limiter (BTRL) register.

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.

- 7:0 - Starting Weight, for traffic issue to the NoC from the host interface
- 19:16 - Max Count Value for Token. Anytime the token count is greater than zero, the host gets qualifies to inject message into NoC
- 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

6.2.15 Credit parity error on each layer (BTPERR)

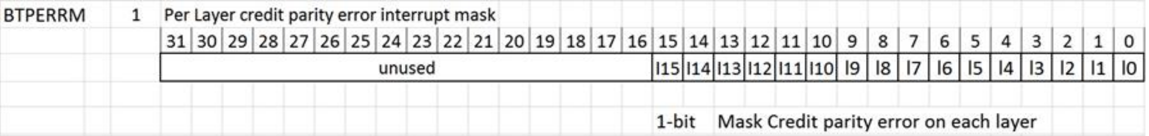


BTPERR	1	Per Layer credit parity error																															
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		unused																l15	l14	l13	l12	l11	l10	l9	l8	l7	l6	l5	l4	l3	l2	l1	l0
																		1-bit	Credit parity error on each layer														

Figure 58 Credit parity error on each layer (BTPERR) register.

Transmit bridge parity error status register. One register bits 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.

6.2.16 Interrupt Mask Credit parity error on each layer (BTPERRM)

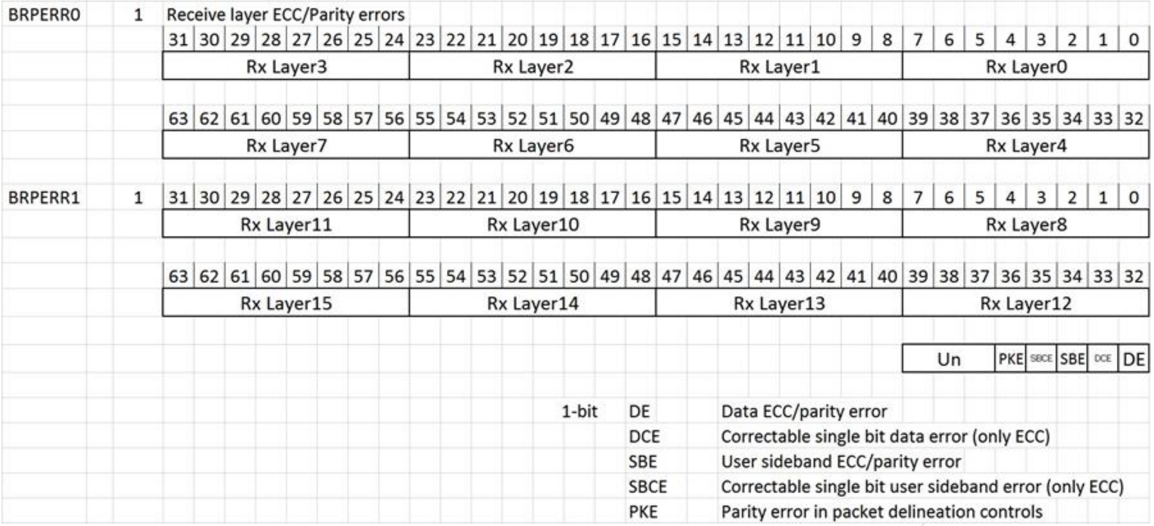


BTPERRM	1	Per Layer credit parity error interrupt mask																															
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		unused																l15	l14	l13	l12	l11	l10	l9	l8	l7	l6	l5	l4	l3	l2	l1	l0
																		1-bit	Mask Credit parity error on each layer														

Figure 59 Interrupt Mask Credit parity error on each layer (BTPERRM) register.

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.

6.2.17 Receive layer ECC/Parity errors (BRPERR)



BRPERR0 1 Receive layer ECC/Parity errors

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
Rx Layer3								Rx Layer2								Rx Layer1								Rx Layer0							

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
Rx Layer7								Rx Layer6								Rx Layer5								Rx Layer4							

BRPERR1 1

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
Rx Layer11								Rx Layer10								Rx Layer9								Rx Layer8							

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
Rx Layer15								Rx Layer14								Rx Layer13								Rx Layer12							

Un			PKE	SBCE	SBE	DCE	DE

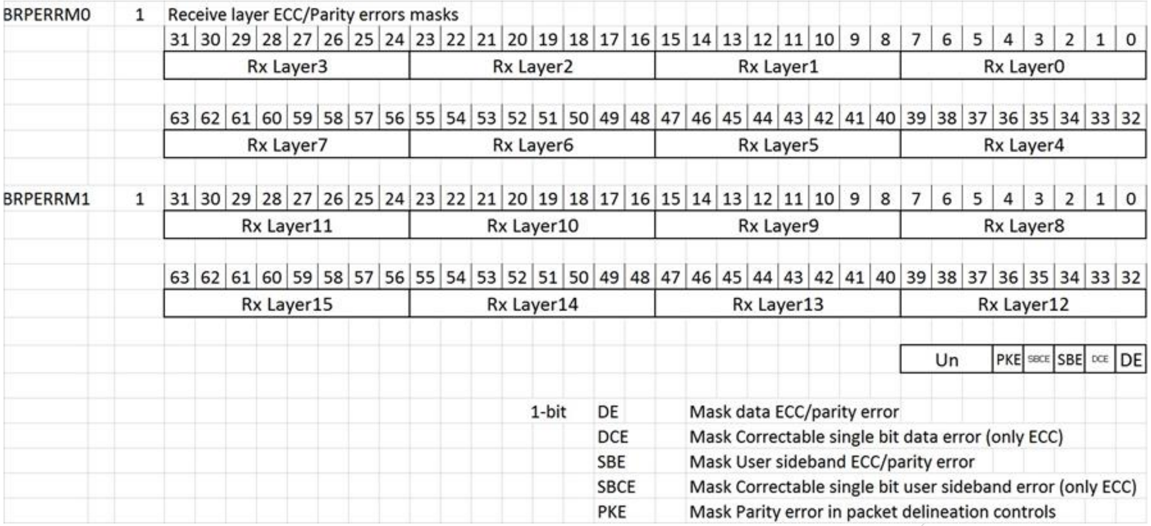
1-bit	DE	Data ECC/parity error
	DCE	Correctable single bit data error (only ECC)
	SBE	User sideband ECC/parity error
	SBCE	Correctable single bit user sideband error (only ECC)
	PKE	Parity error in packet delineation controls

Figure 60 Receive layer ECC/Parity errors (BRPERR) registers.

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.

6.2.18 Receive layer ECC/Parity errors Masks (BRPERRM)



BRPERRM0 1 Receive layer ECC/Parity errors masks

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

Rx Layer3 | Rx Layer2 | Rx Layer1 | Rx Layer0

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

Rx Layer7 | Rx Layer6 | Rx Layer5 | Rx Layer4

BRPERRM1 1

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

Rx Layer11 | Rx Layer10 | Rx Layer9 | Rx Layer8

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

Rx Layer15 | Rx Layer14 | Rx Layer13 | Rx Layer12

Un | PKE | SBCE | SBE | DCE | DE

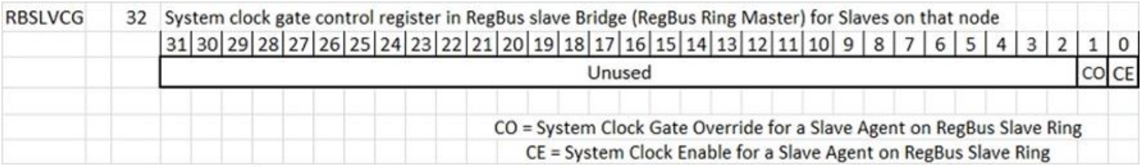
1-bit	DE	Mask data ECC/parity error
	DCE	Mask Correctable single bit data error (only ECC)
	SBE	Mask User sideband ECC/parity error
	SBCE	Mask Correctable single bit user sideband error (only ECC)
	PKE	Mask Parity error in packet delineation controls

Figure 61 Receive layer ECC/Parity errors Masks (BRPERRM) registers.

Mask registers for receive bridge parity error interrupts from register BRPERR0 and BRPERR1. 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.

6.3 REGBUS BRIDGE REGISTERS

6.3.1 System Clock Gate Control Register in Regbus Slave Bridge (Regbus Ring Master) for Slaves on that node (RBSLVCG)



RBSLVCG 32 System clock gate control register in RegBus slave Bridge (RegBus Ring Master) for Slaves on that node

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

Unused | CO | CE

CO = System Clock Gate Override for a Slave Agent on RegBus Slave Ring

CE = System Clock Enable for a Slave Agent on RegBus Slave Ring

Figure 62 System Clock Gate Control Register in RegBus Slave Bridge (RegBus Ring Master) for Slaves on that node (RBSLVCG).

7 Appendix A: NetSpeed Streaming Interface Protocol

This section describes NetSpeed Streaming Interface Protocol (NSIP) and streaming bridge that connects a streaming host port device to the NoC via streaming bridge. NSIP provides a flexible and simple interface to the SoC IPs by allowing up to four bi-directional interfaces at each host port. Each interface uses separate data bus and credit based flow control signals and can receive and transmit messages simultaneously. Each message may be composed of multiple data beats. In each cycle, a single data beat is transmitted or received at an interface, containing a fixed number of data bits equal to the interface data width. The data width of the transmitting and receiving interfaces are independently configurable and must be power of two multiple of their cell size.

Cell size can be a system-wide constant indicating a common denominator for data sizes. Upsizing and downsizing of data happens on power-of-2 multiples of the cell size. For a system with more than one cell size, where interface data width is specified in terms of one of the cell sizes, and communication only happens between interfaces that share a cell size, the cell size property can be set to -1 to indicate that NocStudio should allow multiple cell sizes. Each physical network (a.k.a. layer) can only support one cell size.

Outgoing message data from each host port's TX interface are packetized into NoC packets for injection into the NoC and ejected packets from the NoC are de-packetized into message data beats delivered to the destination RX interface. A single NSIP host port may have both transmitting and receiving interfaces, in the following sections we describe the transmitter and receiver interface behaviors.

7.1 NSIP Transmitter

A host port with transmitting NSIP interfaces can use up to four independently flow controlled interfaces, named a, b, c, and d. Each TX interface has its own specified data width which is a power of 2 multiple of the system wide specified cell size. Each TX interface may transmit messages that are composed of one or more data beats – each data beat is transmitted in a single cycle. Data beats are always equal to the data width of the TX interface and are marked with SOP and EOP bits indicating the start of a message and end of the message, respectively. Multiple messages may not interleave on a TX interface, however there may be any number of idle cycles on a TX interface between successive data beats of same of different messages. Furthermore, any combination of the RX interfaces may receive data beats simultaneously.

NSIP protocol supports a 4-bit QoS that can be transmitted along with each transmitted message. The QoS indicates the Quality-of-Service level of the message and must stay the same for every data beat of the message. In NetSpeed IP, each QoS levels are isolated from each other and has a 2-bit associated priority. Furthermore, each QoS level may have an associated weight at every host port. In NetSpeed NoC IP, the weights are programmable.

A transmitting host port's interface is allowed to transmit to any other host port's interface. Therefore, each transmitted message must also include an 8-bit destination id, indicating a system wide identifier for each destination host port in the system, and 2-bits indicating the destination interface. These 10-bits must stay the same for all data beats of a message. Host port identifiers are system wide unique values (it can be user assigned for each host port in NocStudio environment; if unassigned for a host port then NocStudio automatically assigns unique identifiers).

For flow control, each TX interface uses a separate flow control. The flow control is credit based and occurs at each interface between the transmitting host port and the streaming bridge. For each interface, streaming bridge contains a flow control FIFO for storing the transmitted data beats at the TX interface. The TX host port is expected to maintain a credit counter per interface and is allowed to transmit data beats only when it has credits available for the respective interface. Upon transmitting a data beat, the host port is expected to decrement its credit counter value. When streaming bridge reads data beats from an interface FIFO, it returns the credit back to the TX host port for the respective interface, which is expected to increment the interface's credit counter by one. This is illustrated in Figure 67.

NSIP protocol does not allow interleaving of data beats of different packets at a TX interface.

7.1.1 NSIP TX Signals

The following tables describe the interface pins on a streaming bridge that connects to a streaming host port. Configuration, clocks and reset signals are separately described in the next section.

<Bridge name> is a unique bridge name, which connects a specific host port on a specific host to the NoC. <N> identifies the interface on a streaming bridge; it is one of {a, b, c, d}.

Table 9 NoC Streaming Bridge TX signals from Streaming Host port to NoC

Signal Name	Direction	Width
Host port egress interface * to NoC		

host_dest_hp_*<Bridge name>*_<N>	IN	8-bit (destination host port id)
host_dest_int_*<Bridge name>*_<N>	IN	2-bit (destination interface id)
host_xfer_qos_*<Bridge name>*_<N>	IN	4-bit
host_beat_valid_*<Bridge name>*_<N>	IN	1-bit
host_beat_sop_*<Bridge name>*_<N>	IN	1-bit
host_beat_eop_*<Bridge name>*_<N>	IN	1-bit
host_beat_data_*<Bridge name>*_<N>	IN	Power of 2 x cell_size based on NocStudio specification.
noc_host_credit_inc_*<Bridge name>*_<N>	OUT	1-bit

7.1.2 NSIP TX Timing Diagram

Figure given below shows an example timing diagram of three single-beat transactions at TX Bridge

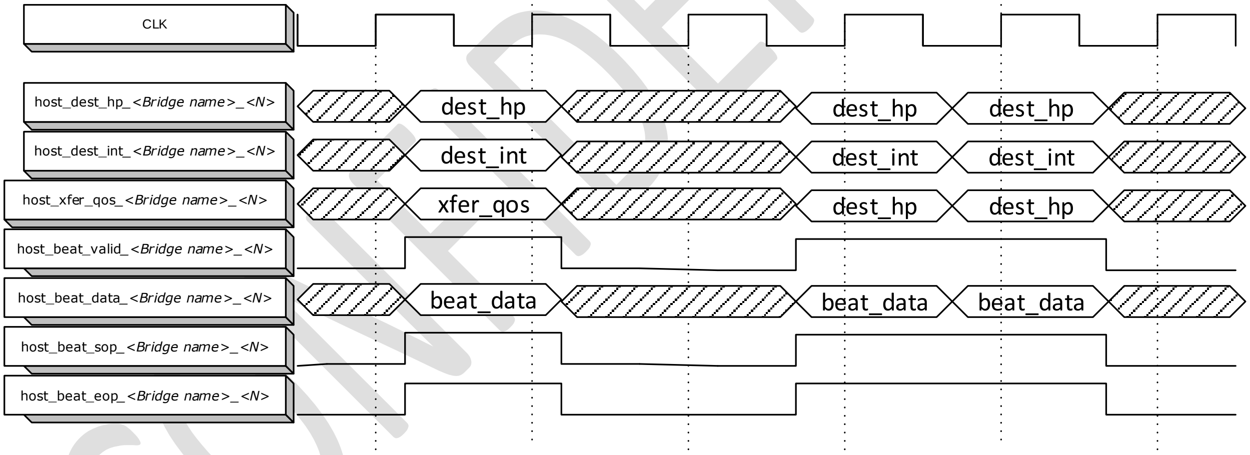

CLK
host_dest_hp_<Bridge name>_<N>
host_dest_int_<Bridge name>_<N>
host_xfer_qos_<Bridge name>_<N>
host_beat_valid_<Bridge name>_<N>
host_beat_data_<Bridge name>_<N>
host_beat_sop_<Bridge name>_<N>
host_beat_eop_<Bridge name>_<N>
dest_hp
dest_hp
dest_hp
dest_int
dest_int
dest_int
xfer_qos
dest_hp
dest_hp
beat_data
beat_data
beat_data

Figure 63: Timing of Single beat transactions at TX Bridge

Figure given below shows an example timing diagram of multi-beat transactions at TX Bridge

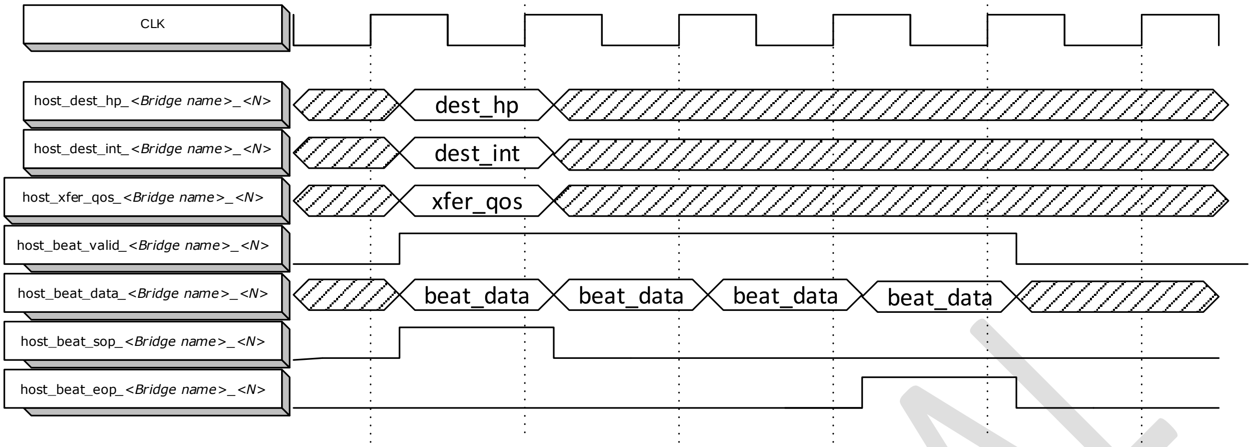

CLK
host_dest_hp_<Bridge name>_<N>
host_dest_int_<Bridge name>_<N>
host_xfer_qos_<Bridge name>_<N>
host_beat_valid_<Bridge name>_<N>
host_beat_data_<Bridge name>_<N>
host_beat_sop_<Bridge name>_<N>
host_beat_eop_<Bridge name>_<N>
dest_hp
dest_int
xfer_qos
beat_data
beat_data
beat_data
beat_data

Figure 64: Timing of Multi beat transactions at TX Bridge

7.2 NSIP Receiver

A host port with receiving NSIP interfaces can use up to four independently flow controlled interfaces, named a, b, c, and d. Just like TX interfaces, each RX interface has its own specified data width which is a power of 2 multiple of its cell size. Each RX interface may receive messages that are composed of one or more data beats – each data beat is received in a single cycle. Data beats are always equal to the data width of the RX interface and are marked with SOP and EOP bits indicating the start of a message and end of the message, respectively. Multiple messages may not interleave on a RX interface, however there may be any number of idle cycles on a RX interface between successive data beats of same of different messages. Furthermore, any combination of the RX interfaces may receive data beats simultaneously.

The 4-bit QoS value is not available at the RX interface; thus the QoS at TX interfaces is intended only for the NoC consumption. Furthermore, the 8-bit destination identifier and 2-bit interface is not available at the RX interfaces, as these are implicit at each RX interface. Just like TX interfaces, each RX interface uses a separate credit based flow control between the receiving host port and the streaming bridge. However, unlike TX interfaces, the flow control FIFO for each RX interface is present at host side within the host port, in which the data beats received by the host port are stored. At the NoC side, the streaming bridge maintains a credit counter per RX interface and is allowed to send data beats to a host port RX interface only when it has credits available for the respective interface; credit counter is decremented by one every time a data beat is sent. When host port reads data beats from one of its RX interface FIFOs, it is expected to return the credit back to the streaming bridge for the respective interface, which increments the interface's credit counter by one. This is illustrated in Figure 67.

NSIP protocol does not allow interleaving of data beats of different messages at a RX interface.

7.2.1 NSIP RX Signals

Following table describes the signals at the host port side of a rx streaming bridge.

Table 10 NoC Streaming Bridge RX signals from NoC to Streaming Host port

Signal Name	Direction	Width
Host port ingress interface * from NoC		
noc_beat_valid_*<Bridge name>*_*<N>*	OUT	1-bit
noc_beat_sop_*<Bridge name>*_*<N>*	OUT	1-bit
noc_beat_eop_*<Bridge name>*_*<N>*	OUT	1-bit
noc_beat_data_*<Bridge name>*_*<N>*	OUT	Power of 2 x cell_size based on NocStudio specification.
noc_bridge_credit_inc_*<Bridge name>*_*<N>*	IN	1-bit

7.2.2 NSIP RX Timing Diagram

Figure given below shows an example timing diagram of single-beat transactions at RX bridge

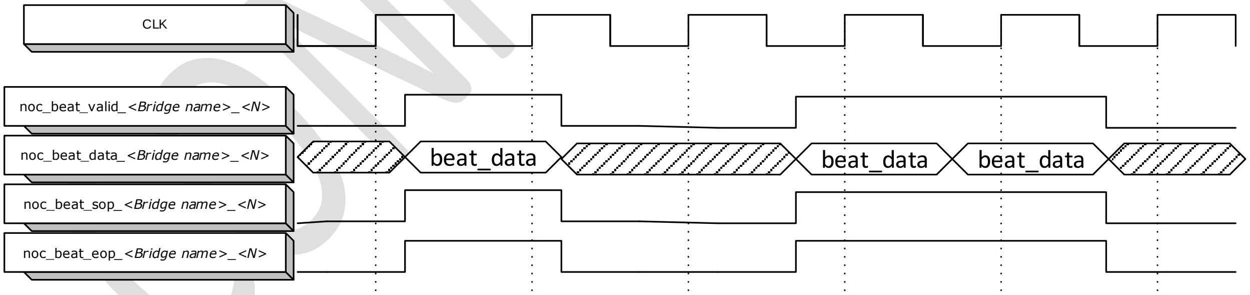

CLK
noc_beat_valid_<Bridge name>_<N>
noc_beat_data_<Bridge name>_<N>
noc_beat_sop_<Bridge name>_<N>
noc_beat_eop_<Bridge name>_<N>
beat_data
beat_data
beat_data

Figure 65: Timing of Single beat transactions at RX Bridge

Figure given below shows an example timing diagram of multi-beat transactions at RX Bridge

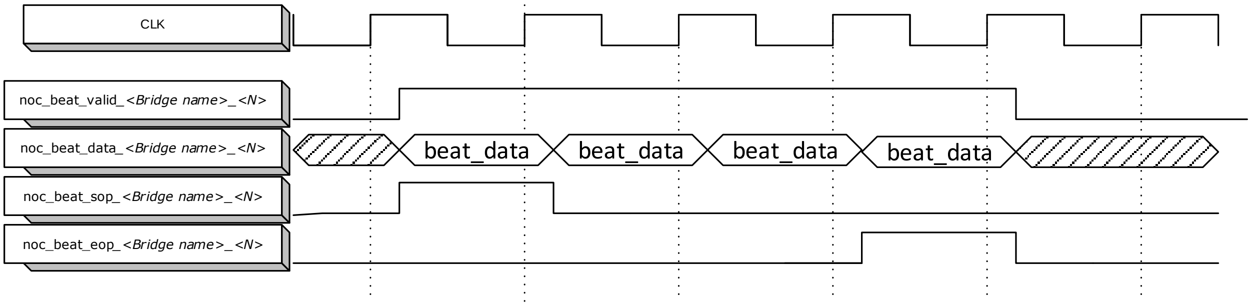

CLK
noc_beat_valid_<Bridge name>_<N>
noc_beat_data_<Bridge name>_<N>
noc_beat_sop_<Bridge name>_<N>
noc_beat_eop_<Bridge name>_<N>
beat_data
beat_data
beat_data
beat_data

Figure 66: Timing of Multi beat transactions at RX Bridge

7.3 Credit Based Flow Control at TX and RX Interfaces

In this section we illustrate the credit based flow control logic at TX and RX interfaces of NSIP host port in Figure 67. While the overall flow control logic is identical at TX and RX interfaces, the logic is partitioned between credit control logic and flow control FIFO differently at the RX and TX interfaces. At TX interface host port is responsible for containing and managing the credit counters for each interface and NoC contains and manages the flow control FIFOs. At RX interfaces roles are reversed.

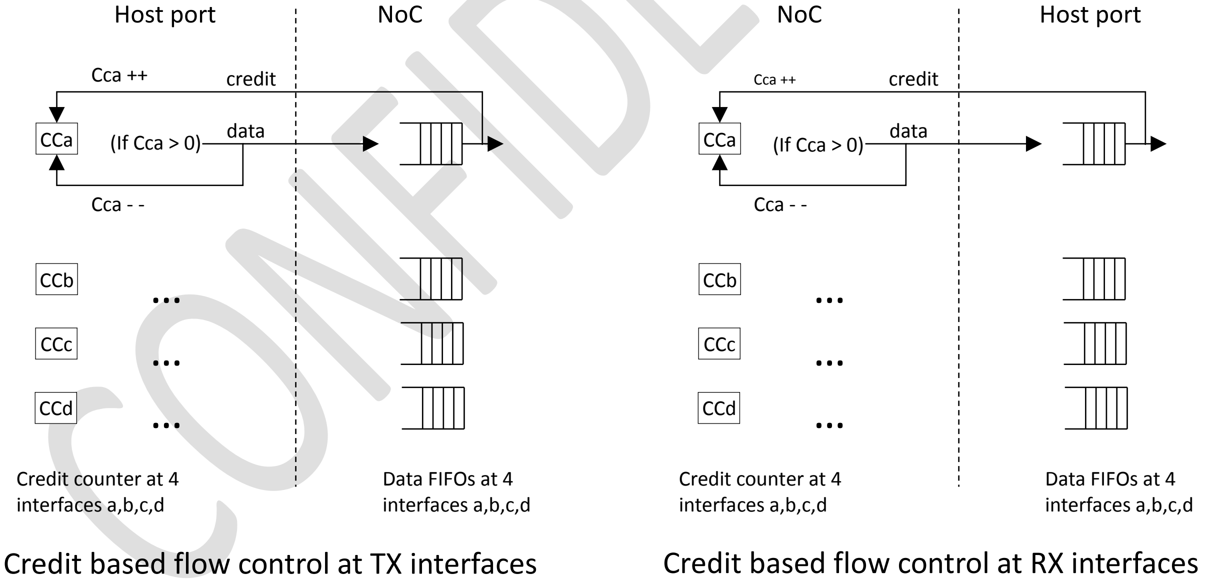

Host port
NoC
Cca ++
credit
CCa
(If Cca > 0)
data
Cca - -
CCb
CCc
CCd
...
Credit counter at 4 interfaces a,b,c,d
Data FIFOs at 4 interfaces a,b,c,d
Credit based flow control at TX interfaces
NoC
Host port
Cca ++
credit
CCa
(If Cca > 0)
data
Cca - -
CCb
CCc
CCd
...
Credit counter at 4 interfaces a,b,c,d
Data FIFOs at 4 interfaces a,b,c,d
Credit based flow control at RX interfaces

Figure 67: Credit based flow control at TX and RX interfaces of NSIP agents.

Note that the at TX interfaces, the credit control logic is present at the host port while at RX interfaces, the flow control FIFO is present at the host port

Figure given below shows the timing diagram of multi-beat transactions with Credit flow control at TX Bridge

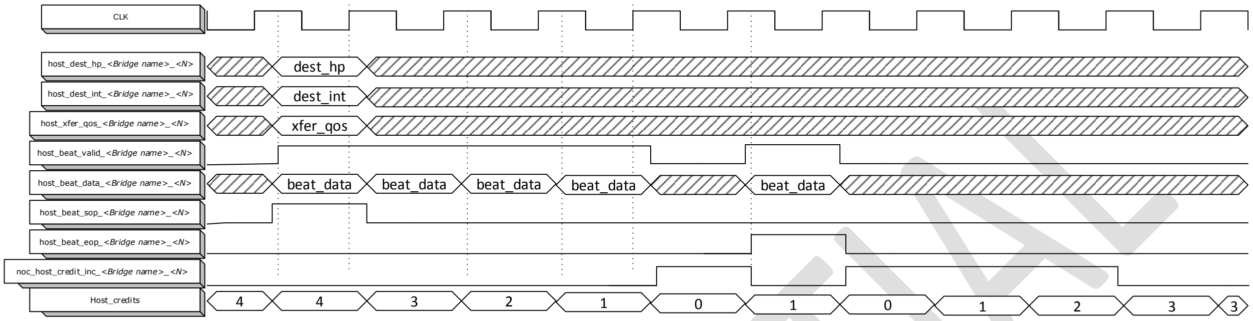

CLK
host_dest_hp_<Bridge name>_<N>
host_dest_int_<Bridge name>_<N>
host_xfer_qos_<Bridge name>_<N>
host_beat_valid_<Bridge name>_<N>
host_beat_data_<Bridge name>_<N>
host_beat_sop_<Bridge name>_<N>
host_beat_eop_<Bridge name>_<N>
noc_host_credit_inc_<Bridge name>_<N>
Host_credits
dest_hp
dest_int
xfer_qos
beat_data
beat_data
beat_data
beat_data
beat_data
4
4
3
2
1
0
1
0
1
2
3
3

Figure 68: Timing of Multi beat transaction with credit flow control at TX Bridge

Figure given below shows the timing diagram of multi-beat transactions with Credit flow control at RX Bridge

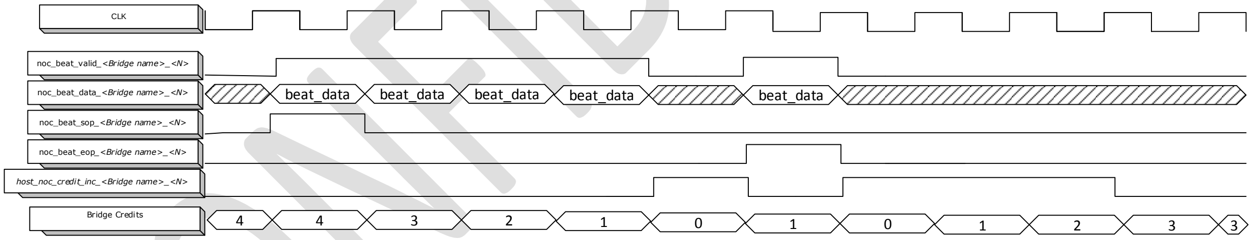

CLK
noc_beat_valid_<Bridge name>_<N>
noc_beat_data_<Bridge name>_<N>
noc_beat_sop_<Bridge name>_<N>
noc_beat_eop_<Bridge name>_<N>
host_noc_credit_inc_<Bridge name>_<N>
Bridge Credits
beat_data
beat_data
beat_data
beat_data
beat_data
4
4
3
2
1
0
1
0
1
2
3
3

Figure 69: Timing of Multi beat transaction with credit flow control at RX Bridge

7.4 Width Ratios and Conversion Summary

Following list summarizes the streaming interface data width restrictions and constraints on the RX and TX communicating interfaces.

- Each host port interface's data width must be power of two multiple of cell_size.
- The minimum and maximum width of interfaces is 1x cell_size and 64x cell_size, respectively.

- Interfaces may be marked as single beat or multi beat.
- A single beat interface must always receive or transmit single beat messages.
- A single beat interface must only communicate with single beat interfaces.
- A multi beat interface must only communicate with multibeat interfaces.
- Any two host port interfaces marked as single beat may communicate with each other only if they have same data width.
- Any two host port interfaces marked as multibeat may communicate with each other irrespective of their data width.
- A host port may have any combination of up to four TX and up to four RX active interfaces, and each interface may have any allowed width and single beat property.

The allowed communication between different interfaces marked as single beat or multibeat and having different widths are illustrated between in Figure 70.

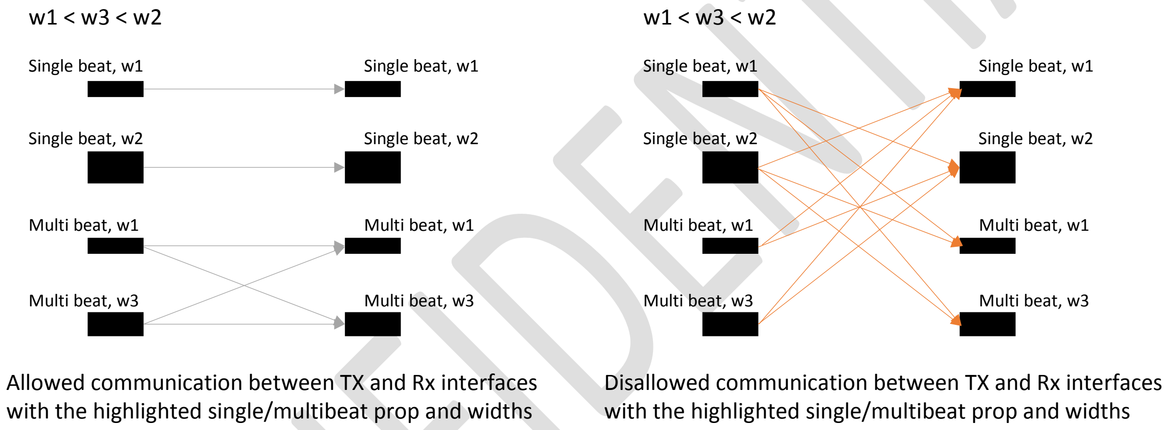


Figure 70 Four TX interfaces (left) and four RX interface (right) that may and may not communicate with each other based on interface widths and single beat prop

When a TX interface communicates with a RX interface with different width (none of them can be single beat), TX message data beats undergo width conversion before being delivered at the RX interfaces. If TX interface is wider than the RX interface then, each TX data beat is split into r data beats, where r is the interface width ratio, width (TX)/width (RX). This number is always a power of 2 value as interface widths are always a power of 2 value. When TX interface is narrower than the RX interface then beginning from the SOP data beat, each successive data beats of up to *t* beats of a message at the TX interface are combined into a single data beat delivered at the RX interface, where *t* is the interface width ratio width(RX)/width(TX). If the number of TX data beats modulo t is not zero, then the last succession of combined TX data beats would contain less than t TX data beats. In this case, valid TX beats are combined and the invalid beat locations have random content.

Below is an example to illustrate the width conversion of a message from a narrow TX interface to a wide RX interface, assuming that the TX message is composed of seven data beats and width ratio is four.

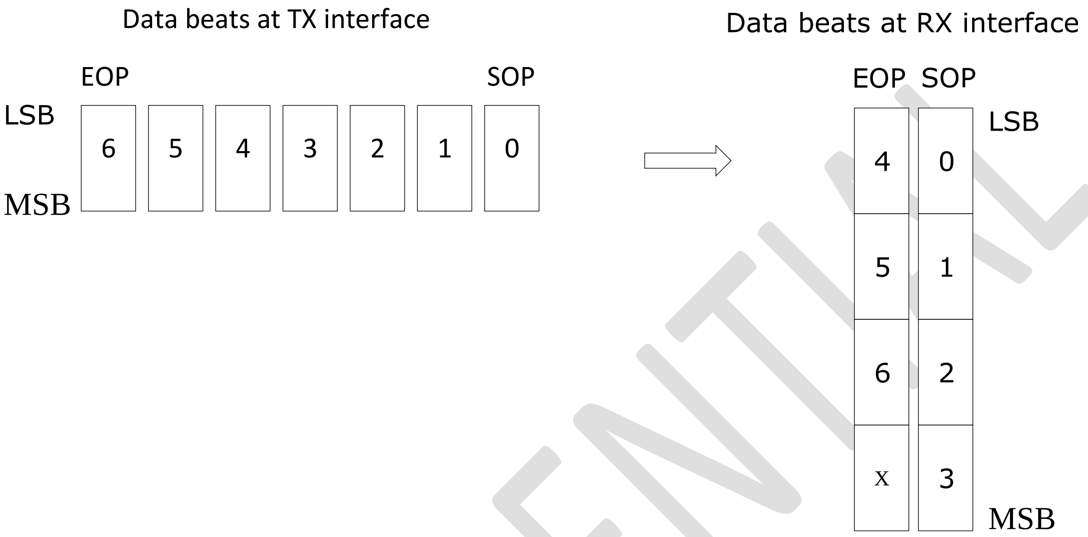


Figure 71 Illustration of width conversion of data beats of a message in NSIP protocol from a narrow TX interface to a wide RX interface

Wide TX to narrow RX width conversion is identical.

7.5 Ordering Requirements

All message with the same QoS value transmitted from a host port's TX interface to another host port's RX interface must be delivered in the transmitted order at the RX interface.

Messages with different QoS values between same source and destination host ports interfaces are not required to be ordered.

7.6 NetSpeed Streaming Bridge Overview

This section describes the NetSpeed Streaming Bridge functionality and how it is used in NocStudio to interface with streaming host port devices and how communication between various streaming interfaces is specified.

At TX side, streaming bridge maps the transmitted messages at TX interfaces by host ports to NoC layers and VCs by looking up a VC mapping table (a VC mapping tables exists at each transmitting streaming bridge). The bridge also performs the route computation and provides the needed infrastructure for the end-to-end QoS (fixed priority as well as the weights) enforcement. At RX side, streaming bridge simply delivered the NoC messages to the host port interfaces. A high level diagram of streaming bridge is shown in Figure 72. There are up to four TX and up to four RX interfaces to the host port. These interfaces are named a, b, c, and d. On the NoC side, a streaming bridge may connect to up to 8 NoC layers, *i.e.* 8 routers ports, a router from each layer. Each router port may have up to 4 VCs indicated by vc0, vc1, vc2, and vc3. TX part of the bridge sends data from host port to NoC, and RX part does the opposite.

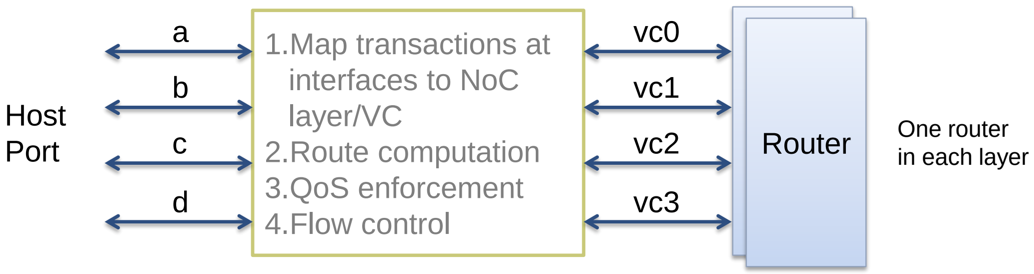


Figure 72 NetSpeed streaming bridge Interfaces to host port and to NoC

7.6.1 Functions of Transmitting (TX) Streaming Bridge

A transmitting streaming bridge performs the following functions.

1. Maps outgoing messages from host port TX interface to NoC layer and VC.
2. Determine route for the message.
3. Packetizes the message into NoC packets.
4. Implements QoS functions, both strict priority and weighted bandwidth.
5. Implements the credit based flow control at the host port TX interfaces.
6. Implements the flow control at the router side.
7. Performs arbitration between the interfaces if they contend for the same VC and/or NoC layer.

7.6.2 Functions of Receiving (RX) Streaming Bridge

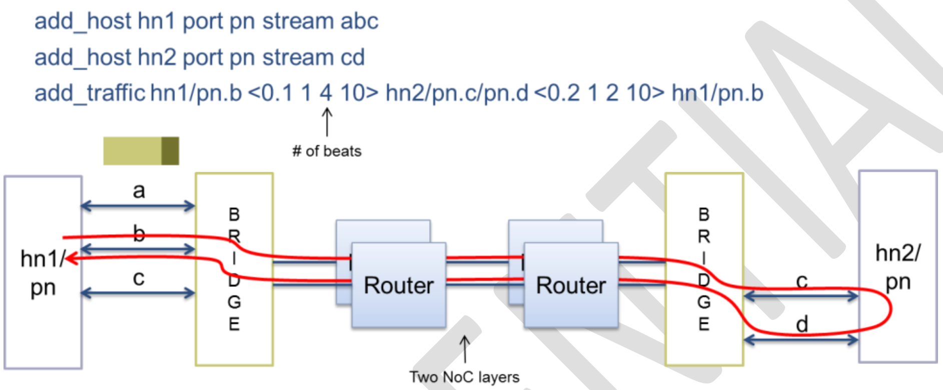
A receiving streaming bridge performs the following functions.

1. De-packetizes NoC packets into messages and deliver at the host port RX interfaces.
2. Implements the flow control at the router side.
3. Implements the credit based flow control at the RX host port interfaces.

4. Performs arbitration between VC and/or NoC layer if they contend for the same interface.

7.7 Adding Streaming Bridge and Traffic in NocStudio

Below is an illustration of an example of adding hosts with streaming host ports with different number of interfaces and using *add_traffic* to add a request and response messages between the host port's interfaces.



```
add_host hn1 port pn stream abc
add_host hn2 port pn stream cd
add_traffic hn1/pn.b <0.1 1 4 10> hn2/pn.c/pn.d <0.2 1 2 10> hn1/pn.b
```

Figure 73 Illustration of a NoC with streaming host ports and traffic between them

Here two hosts named hn1 and hn2 are added. Each has a single host port named *pn* of protocol type stream, and they have a,b,c and c,d interfaces, respectively. Traffic is added which consists of a request and response. hn1/pn makes a request message at interface b to destination hn2/pn's interface c. hn2/pn responds with a response message at interface d which is sent to the interface b of hn1/pn. Notice that the response message may be sent by and destined to a different interface than the interface from where the request message has arrived from. Host hn2 in this case must be able to determine the correct destination and put this information in the response message so that NoC can deliver it correctly. Alternatively, such traffic may be added as two separate add_traffic commands, one for request and one for response, as shown below:

```
add_traffic hn1/pn.b 0.1 1 4 10 hn2/pn.c
add_traffic hn2/pn.d 0.2 1 2 10 hn1/pn.b
add_dep hn2/pn.c hn2/pn.d
```

Notice that here an additional ***add_dep*** command is used to indicate to NocStudio that the second message is a response of the first message, and that the second message may backpressure the first message. add_dep command must be used to correctly specify the dependencies between various interfaces of streaming bridges to ensure that the resulting NoC constructed by NocStudio is deadlock free.

7.8 STREAMING BRIDGE SPECIFICATION

This section describes the Streaming bridge interfaces in greater details, and the functional specification of various streaming bridge components.

7.8.1 Streaming Host Port Requirements

Following are requirements, and expected behavior of the host port interfaces connecting to a streaming bridge:

- Streaming host ports using the streaming bridges to communicate with each other can use up to four separate physical interfaces supported by the bridge. The interfaces are named a, b, c and d, and the list of active interfaces of a host port may be provided with the **add_host** command. Each interface can be uni- or bi-directional. Whether an interface is bidirectional or not depends on the traffic specification, *i.e.* if traffic transaction hops only leave (arrive at) an interface then it becomes an output (input) only interface, else it remains bi-directional.
- The width of each direction of an interface can be set to any power-of-two multiple of the cell size and can be viewed or modified using the **bridge_prop** command. If a NoC supports multiple cell sizes (**cell_size** = -1), the interface must be a power-of-two multiple of its cell size. An interface may be marked as single beat using **ifce_prop** is_single_beat yes command; if an interface is marked as single beat then it may not send or receive a multi-beat message and may not communicate with multibeat interfaces.
- Two communicating interfaces must have same is_single_beat property.
- If two communicating interfaces are single beat, then their width must be the same. They may have different width only if they are not marked as single beat.
- When two communicating non-single-beat interfaces have identical width, the transmitted data beats at a TX interfaces are delivered in order at the destination interface and the numbers of transmitted and received data beats are equal.
- When two communicating non single beat interfaces have different widths, the ratio of their widths, r, a power of two value, determines the number of beats in received message. Going from a wider interface to a narrower interface, each wider data beat is divided into r narrower data beats during delivery at the destination interface; the LSBs of the wider

flit form the first narrow flits which are delivered before the MSBs. Going from a narrow to wide interface, up to r transmitted data beats are combined into a single data beat delivered at the RX interface; the first narrow beat becomes the LSBs of the first wider data beat delivered at the RX interface. During combining, if an EOP arrives and there are not r data beats to combine then the combined data beat at RX interface is padded.

- Any source streaming host port's interface may communicate with any destination streaming host port's interface, as long as they have identical is_single_beat property.
- Each injected message must have the destination bridge id and interface id and a 4-bit QoS, each of which must stay constant for all data beats of the message at the TX interface. A TX interface can only send messages with those destination bridges and interface ids and QoS values that are specified in the **add_traffic**. Any other messages will be dropped.
- For streaming host ports, each transaction hop or message must be explicitly specified using **add_traffic** and there is no response automatically associated with any request.
- An interface may transmit or receive traffic of multiple priorities and weights, which is indicated in the message using the 4-bit QoS signal at the TX interface. Refer to QoS section for more details on this signal. At the bridge, each 4-bit QoS value maps to a 2-bit priority and 8-bit weight.

7.8.2 Injection and Ejection Port VC Width Computation

Following rules are generally (but not always) used by NocStudio to determine the width of the VCs at the injection and ejection ports to which the streaming bridge is connected.

- The width of injection/ejection port VCs of a router to which Streaming interfaces are connected are determined based on the Streaming interface data widths and how VCs and interfaces are connected.
- At RX side, if a router' ejection port sends data to a single interface then the interface width and the port and VC width are the same.
- At RX side, if a router' ejection port sends data to multiple interfaces then the width a VC on the port will be equal to the width of the widest interface to which it sends data.
- At TX side, if a single interface sends data to a router' injection port then the interface width and the port and VC width are the same.
- At TX side, if multiple interfaces send data to a router' injection port then the width of a VCs on the port will be equal to the width of the widest interface from which it received data.

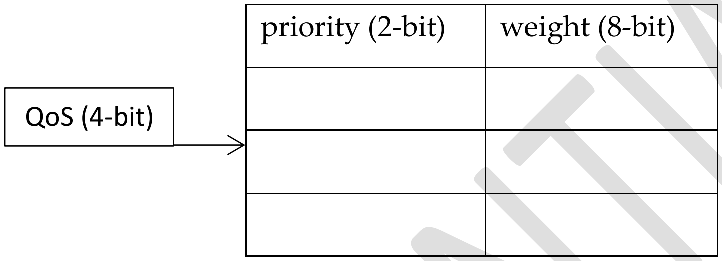
7.9 QoS

There are two types of QoS attributes that can be assigned to the messages; the 4-bit QoS signal along with a message is used to describe these two for the message.

1) strict priority
2) weighted bandwidth allocation

There is a fixed mapping between 4-bit QoS and 2-bit priority at every host port; by default, the 2-LSB bits of QoS forms the priority of the message. This mapping can be modified in NocStudio using **bridge_prop** command.

The QoS is used to determine the VC and NoC layer, therefore it is used as one of the indices into the VC mapping table at TX bridges. VCs and routes are allocated only for those QoS values that were specified in the traffic profile using **add_traffic**. A message that was not specified using **add_traffic** will cause a protocol error at the bridge.



With each QoS, there is an associated weight value too. The reset time weight value at each streaming bridge is configurable via NocStudio using **bridge_prop** command. Later the weight values must be re-configured via regbus. Alternatively, one may modify the weight of by using different QoS values for the messages, assuming that they are added via **add_traffic** and they have different configured weights.

7.9.1 Programmable Configurations

The weight for each QoS level is programmable via regbus. Initial weight values at reset time are configured via NocStudio **bridge_prop** command.

7.9.2 VC Mapping table and QoS priority

VC Mapping table at TX wide of streaming bridge is configured based on the destination host port and interface and QoS value of various messages at the source host port and interfaces. Only the messages that are added via **add_traffic** command are supported. Any other message will cause a protocol error. The VC mapping table is designed such that the resulting logic is optimized based on the VC and NoC layers of various messages, QoS values and the destination bridge and interface id.

VC mapping table is not configurable; it is set during the NoC design time.

2670 Seely Ave
Building 11
San Jose, CA 95134
(408) 914-6962
http://www.netspeedsystems.com