NETSPEED SYSTEMS

NetSpeed Crux Technical Reference Manual

Version: CRUX-16.04.a

June 24, 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

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 partioning, etc., significant configurability is also provided to the architect in defining the NoC topology as well as other system level characteristics.

2 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 (AXI, ACE, etc.) and infered from system traffic information. The combined dependency specification is used to ensure full deadlock avoidance – both at the network-level and the protocol-level.

2.1 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 one of the other operations to complete. 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 1 shows the resources and dependencies for this example.

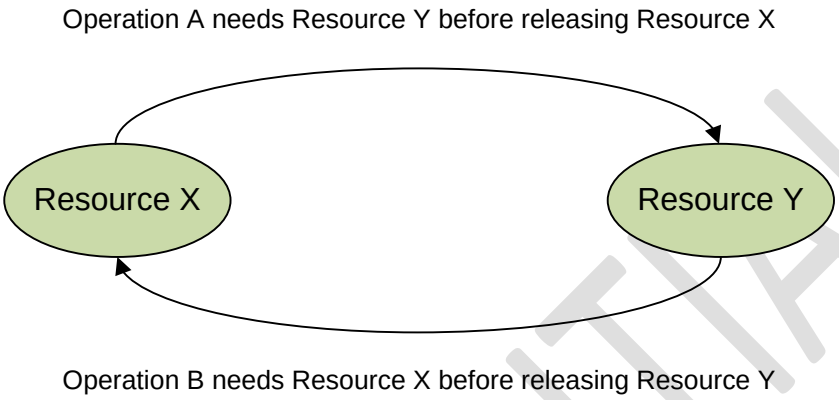


Figure 1. Simple Deadlock Graph with Two Resources

Figure 1 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 2 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.

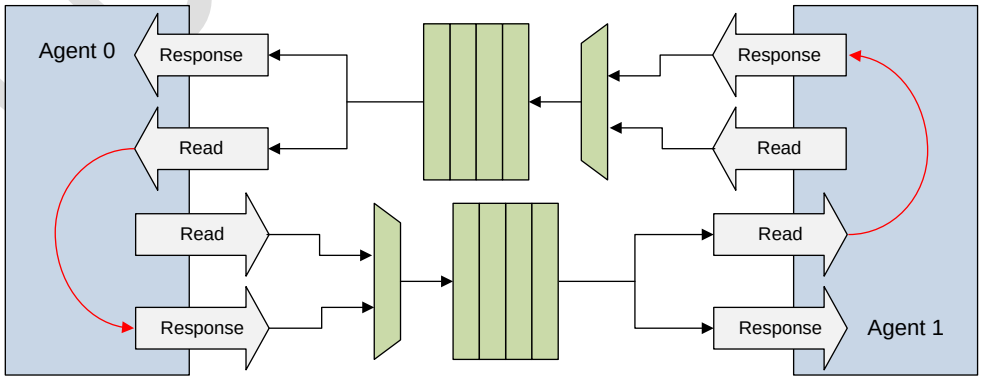


Figure 2. Simple Interconnect with Deadlock

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.

2.2 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 2 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 2, 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 3, NocStudio also generates a comprehensive system

dependency graph as well so that architects can crosscheck and ensure that there are no deadlocks.

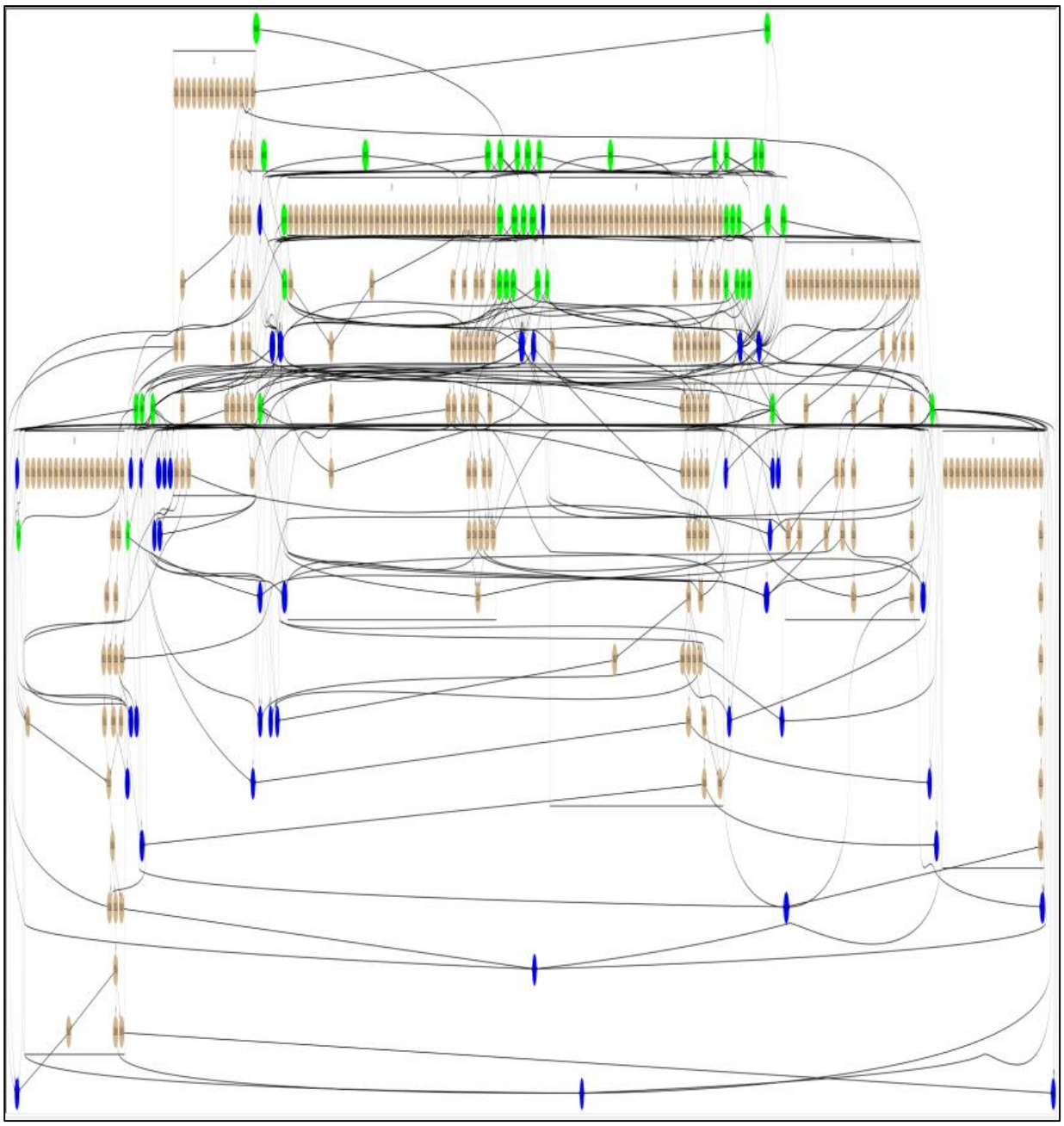


Figure 3 - System Level Dependency Graph

3 Security

3.1 Connectivity-Based Firewalls

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

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

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

3.2 Address Range Connectivity

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

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

3.3 Propagation of TrustZone Bit

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

In coherent systems, the AxPROT[1] bit is used as an additional address bit for coherent requests. This ensures that coherency IP treats non-secure and secure accesses to an address as if they were different addresses, preventing an access to one address from accessing the data in the other.

These mechanisms are supported in NetSpeed IP.

3.4 Selective Traffic Filtering

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

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

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

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

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

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

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

3.5 Programmable Address Map

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

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

3.5.1 Disabling Address Ranges

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

3.5.2 Changing the Address Map

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

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

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

3.5.3 Modifying Traffic Filtering

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

3.6 Interface Overrides

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

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

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

3.7 User Bits

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

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

4 Quality of Service Support

Quality of Service (QoS) is commonly known as a mechanism for ensuring a predictable performance in a network. In modern day SoCs, a comprehensive end-to-end QoS support is often required as it aims at fair allocation of bandwidth when multiple traffic flows contend for system resources. This requirement poses a fundamental technical challenges to SoC architects.

In NetSpeed IP, QoS can be defined easily through traffic specification. Each transaction specified by command add_traffic/ add_traffic_b contains a 4-bit QoS value (0-15). Each QoS value is then mapped to two attributes: a 2-bit priority (0-3), and an 8-bit weight (0-255).

QoS, together with its priority and weight, can be used to ensure the quality of a network through:

- Strict priority based allocation
- Weighted bandwidth allocation
- Rate limiting hosts
- Dynamic priority support

4.1 STRICT PRIORITY BASED ALLOCATION

In strict priority, a flow marked as high priority always preempts a low priority flow when they contend for the same resource.

Strict priority is the simpler to implement in a NoC, such as by using priority based arbitration at each router so that higher priority packets always win arbitration. However in this type of design, Head of Line (HoL) blocking may occur as low priority packets at a router's channel may block high priority packets at upstream routers. This can be addressed by ensuring that each priority class is in a different virtual channel. For systems with a large number of priority classes, the number of virtual channels may become a concern. NetSpeed IP achieves the best of both worlds and dynamically reuses virtual channels in a way that end-to-end priority enforcement without HoL blocking is enforced with a small number of virtual channels.

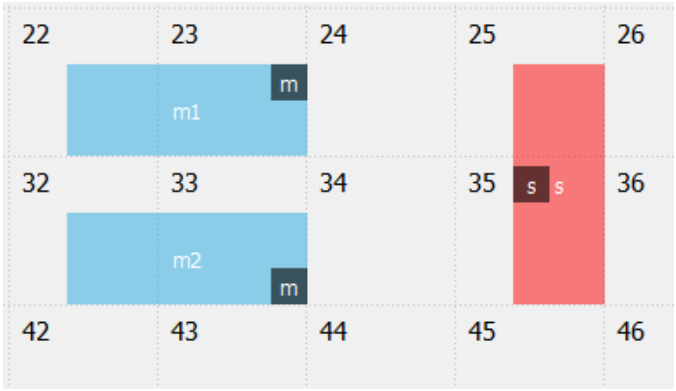
4.1.1 How to Define the Priority

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

4.1.2 Examples

Here are some examples on strict priority based allocation.

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



The relationship between priorities and various QoS id can be reported through command "qos_pri_map".

```
$ qos_pri_map
QoS  0, priority 0
QoS  1, priority 1
QoS  2, priority 2
QoS  3, priority 3
QoS  4, priority 0
QoS  5, priority 1
QoS  6, priority 2
QoS  7, priority 3
QoS  8, priority 0
QoS  9, priority 1
QoS  10, priority 2
QoS  11, priority 3
QoS  12, priority 0
QoS  13, priority 1
QoS  14, priority 2
QoS  15, priority 3
```

4.1.3 Example #1

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

```
add_traffic qos 0 rates 1 1 m2/m ar s/s
add_traffic qos 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	-	-	-	-									

4.1.4 Example #2

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

```
add_traffic qos 0 rates 1 1 m1/m ar s/s
add_traffic qos 3 rates 1 1 m2/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	-	-	-	-									

4.1.5 Example #3

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

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

4.2 WEIGHTED BANDWIDTH ALLOCATION

In weighted allocation policy, the resource bandwidth is divided among all contending flows based on a pre-specified set of weights.

Weighted bandwidth allocation in NoC and maintaining the work conserving property is a much more challenging problem. The bandwidth received at the destination from various sources depends on how arbitration is performed at the routers, the position of the sources and destinations of the flows within the NoC, and the state of other flows, specifically the current transmission rate of other flows. 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.

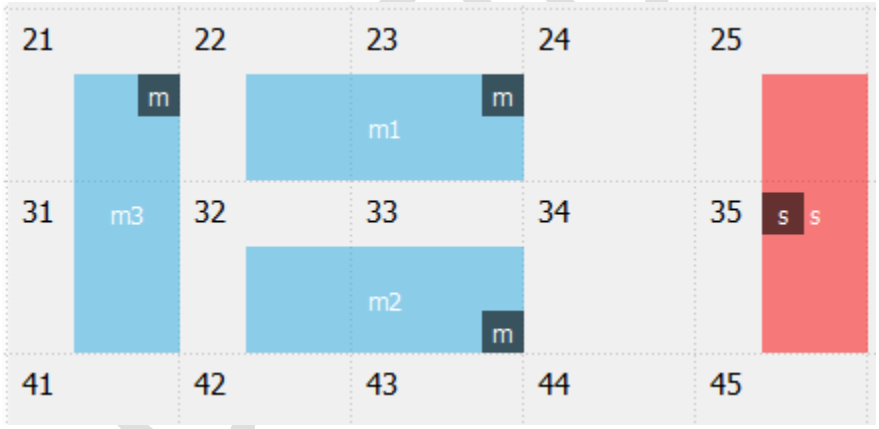
4.2.1 How to Define the Weight

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

4.2.2 Examples

Here are some examples on weighted bandwidth allocation.

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.

```
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
```

4.2.3 Example #1

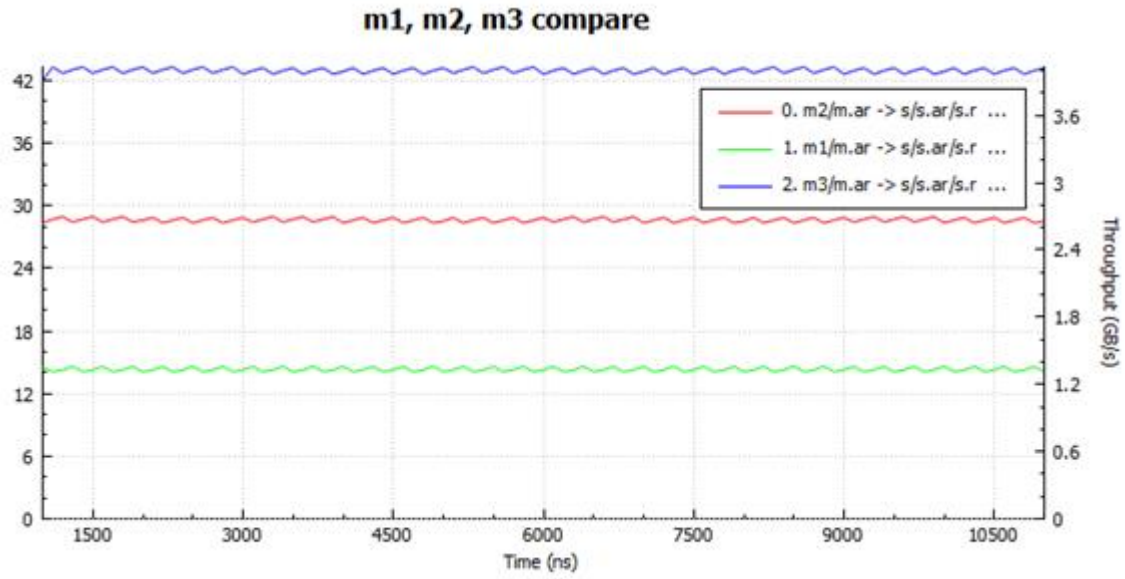
In a case where traffics from all 3 masters are contending for resources, the ratio of resulting bandwidth for each master bridge follows that of the weight:

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

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

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

This ratio can also be observed from the graph below.



4.2.4 Example #2

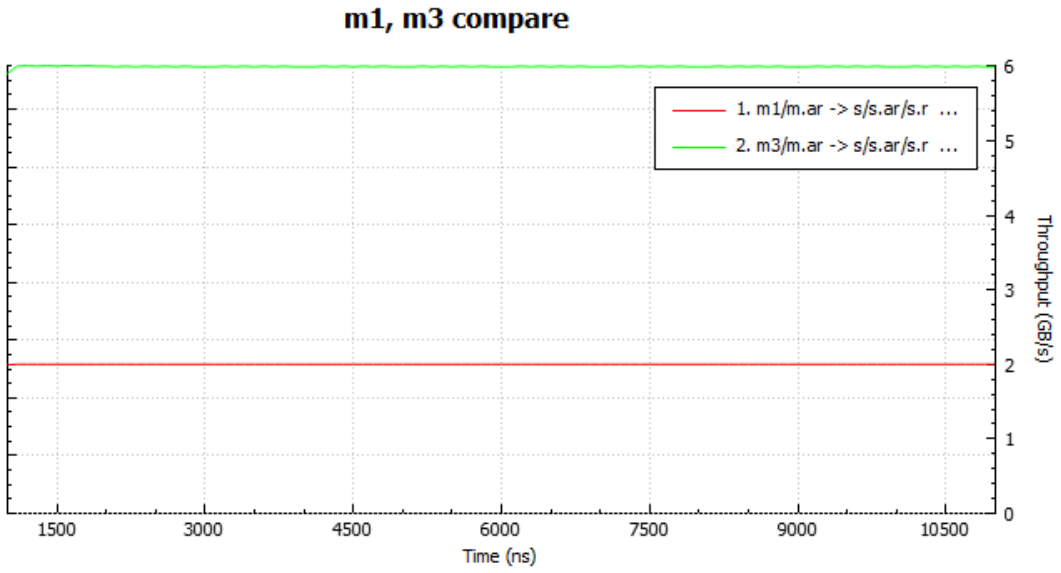
In a case where only 2 traffics are contending for resources, the ratio of resulting bandwidth for each of the 2 master bridges follows their defined weights:

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

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

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

This ratio can also be observed from the graph below.



4.3 RATE LIMITING HOSTS

While the above two QoS mechanisms are sophisticated and effective, they have costs and limitations. The traffic isolation and priority mechanism requires multiple virtual channels. The weighted QoS mechanism works best with multiple masters and a single target.

NetSpeed NoC also 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 enabled 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.

4.3.1 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.

4.3.2 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.

4.3.3 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.

4.3.4 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 16-bit therefore the rate granularity is $1/(2^{16})$.

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.

4.3.5 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.

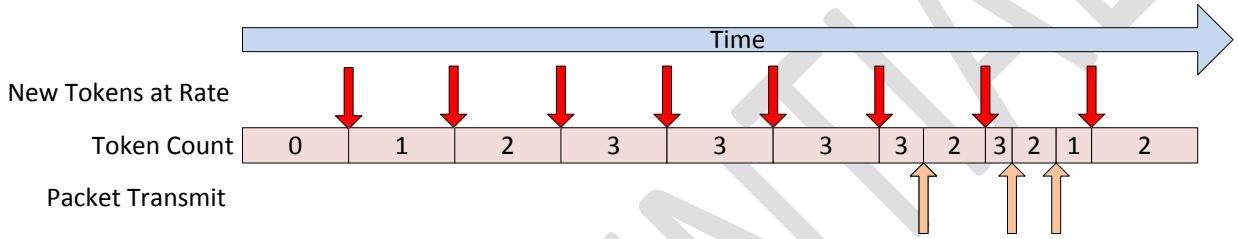


Figure 4 - Token Count increases at specified rate. Packet transmit decrements count.

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.

4.3.6 Specifying A Rate

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

In hardware, the rate is specified as a 16-bit number. The value of the number follows this equation, where N is the programmed value.

rate = $N/(2^{16})$

The hardware implements the rate calculation as a 16-bit adder where the overflow bit is used as the token arrival bit. This provides significant granularity for specification of the rate limit. For example, to specify a rate of 1 token every 5 cycles (or 20%), N should be specified as 13107(decimal) or 0x3333. When added together 5 times, the value will nearly reach approximately 216, so one packet can be sent every 5 cycles.

4.3.7 Token Bucket Sizing

The token bucket size allows for an agent to accumulate tokens over a window of time. This allows the agent to issue a burst of requests over a smaller window, while still being limited in the long run. The larger the bucket, the larger the short-term burst can be.

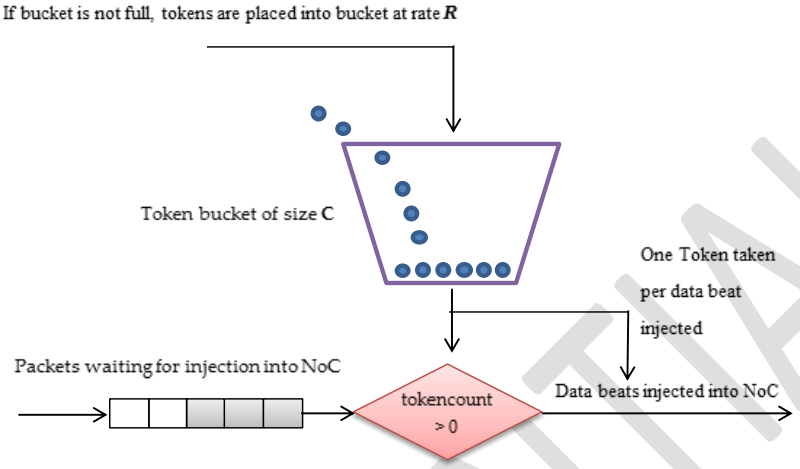


Figure 5 - Token Bucket

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 16. In NocStudio, this value is programmed using the interface property rate_limit_bucket_size.

4.3.8 Token Usage

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

4.3.9 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
15:0	Rate Limit Value, for traffic issue to the NoC from the host interface
19:16	Bucket Size. This indicates the maximum number of token that may be accumulated at an interface when rate limiters are enabled

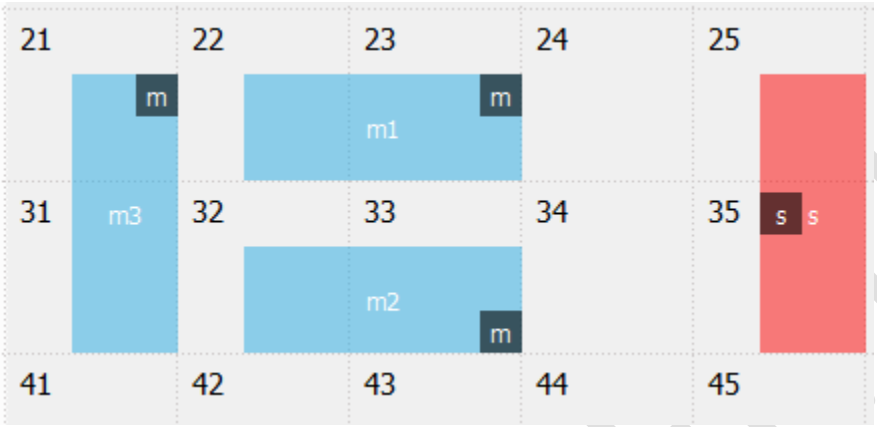
20	Rate Limit logic enable. Rate limiter logic is used for arbitration only when the enable bit of the rate limiter configuration register is set to 1

Table 1 - Rate limiter Configuration Register

4.3.10 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.

4.4 DYNAMIC PRIORITY SUPPORT FOR ISOCHRONOUS TRAFFIC

4.4.1 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. Instead, it is common to treat this traffic as 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.

4.4.2 Two Priority Level Specification

The dynamic priority support for isocronous 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.

One intended use of this feature is to support isochronous traffic classes, such as display traffic. Display traffic would be given its own traffic class and two priorities. It should be given a low priority as its default value. When the display engine starts falling behind in its data prefetching, it can change the dynamic priority select control to switch to the alternative priority, which should be set to a high priority. This will allow the traffic to start as low priority but switch to high priority as the real-time requirement approaches.

4.5 MAPPING AMBA QOS VALUES

The QoS discussed thus far in this section is NoC QoS.

There's another type of QoS-- AMBA QoS, which can be specified by AXI masters. A mapping between AMBA QoS and NoC QoS 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 QoS value of 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 QoS value of 3. This gives the master the ability to send different QoS traffic to the same destination. However, to guarantee ordering at the point of QoS transition, traffic is serialized between the two different QoS streams.

```
add_traffic  amba_qos {0 1 2 3} qos 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} qos 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 QoS value for that interface channel.

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 6. 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.

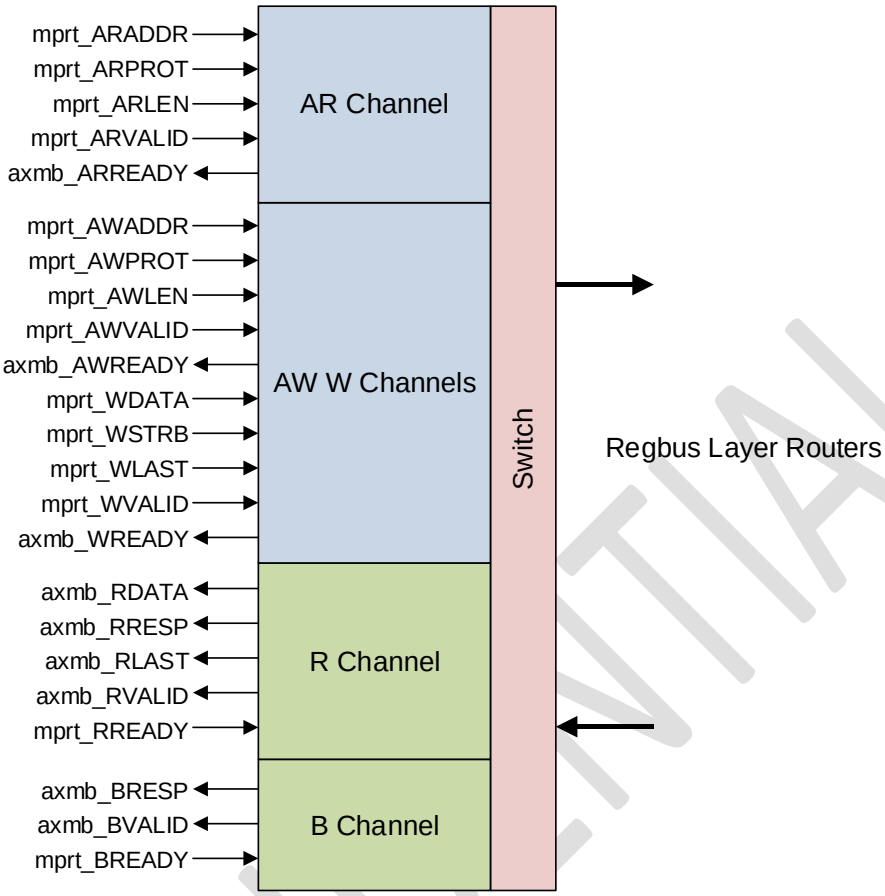


Figure 6. Regbus Master Bridge

5.1.2 The Regbus Layer

As shown in Figure 7, 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.

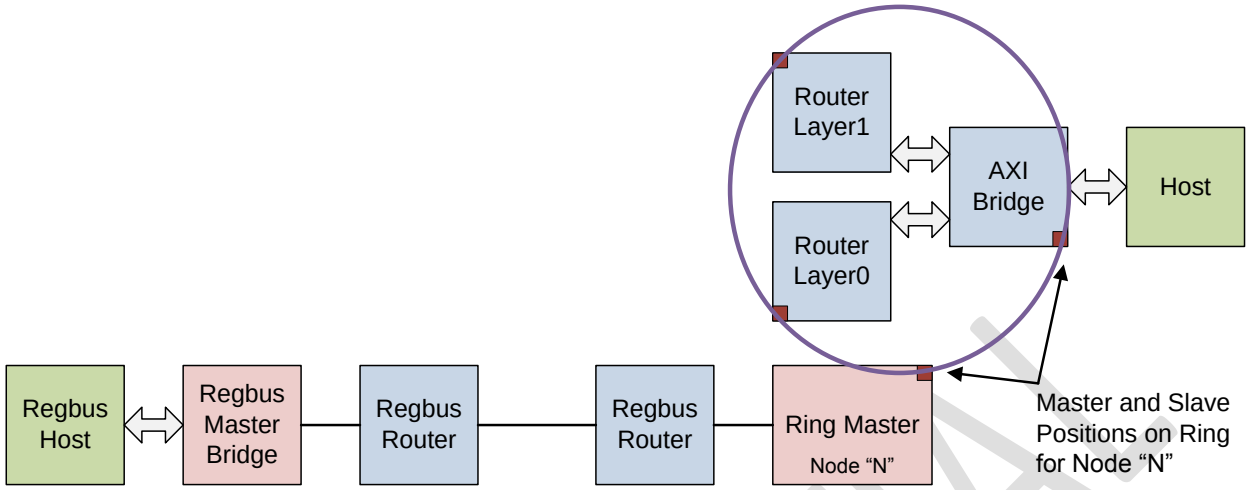


Figure 7. Regbus Layer Communication

By default the Regbus master bridge, Regbus routers, and RingMaster all run at a common frequency. Hoever, 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 8 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.

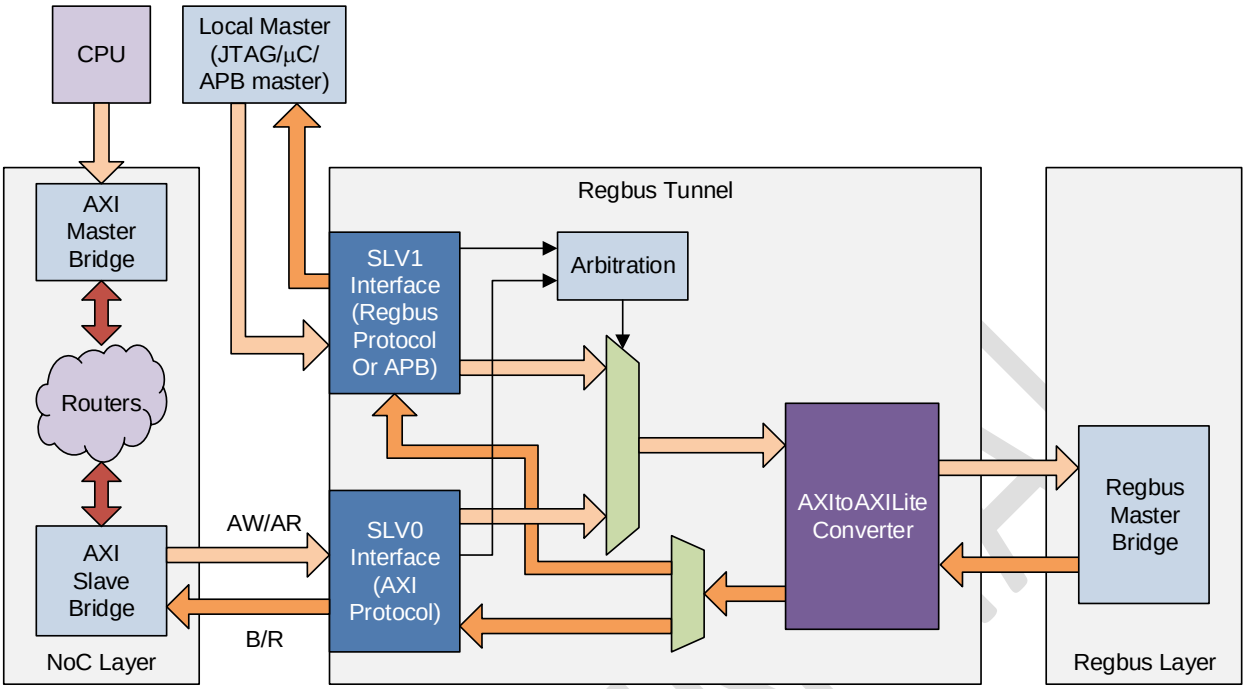


Figure 8. Regbus Tunnel Connects Primary NoC Layer to Regbus Layer

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.

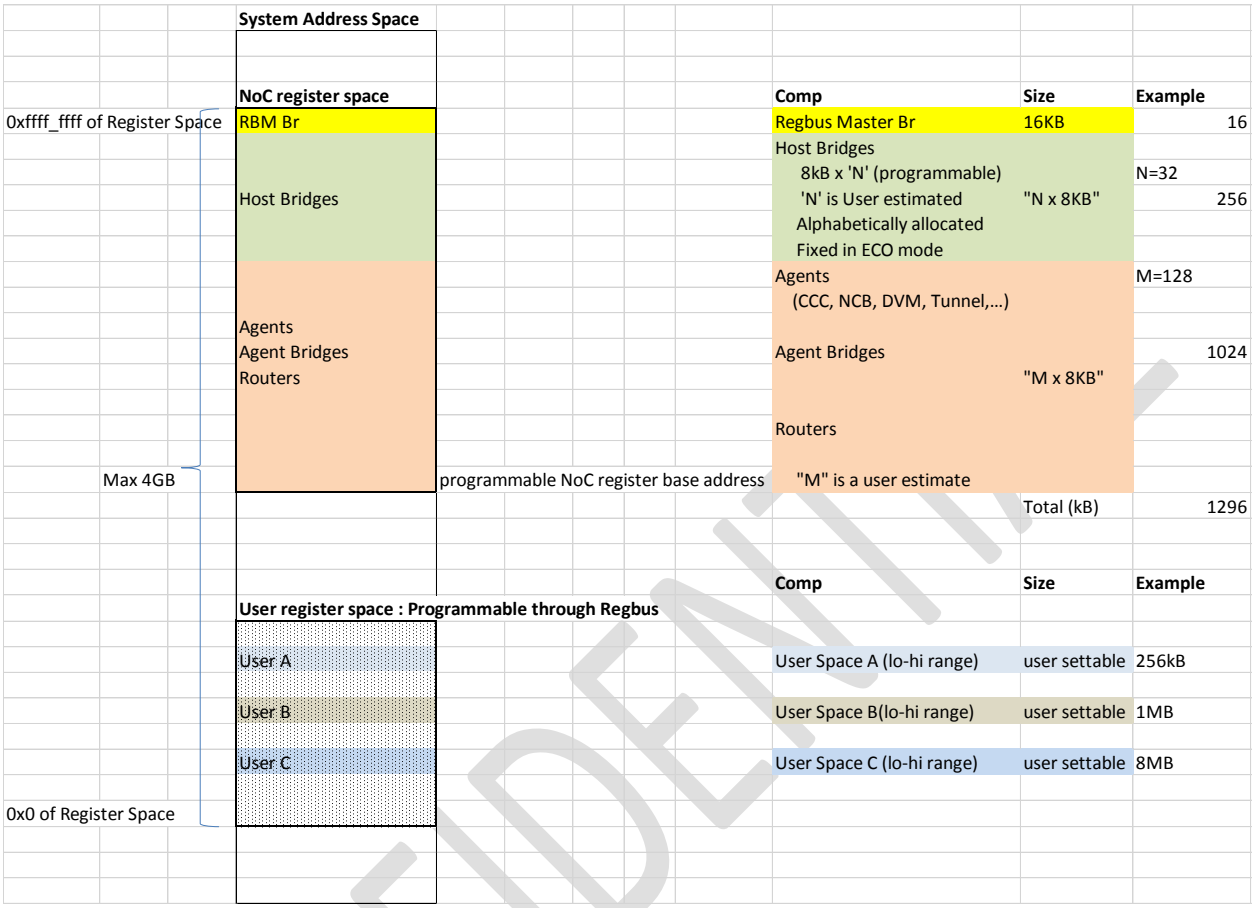


Figure 9: Regbus Address Map

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 2: 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 3: 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 4: Response table for NoC Register Accesses

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

64-bit access to defined 32-bit register	Okay
32-bit access to defined 64-bit register	Okay. Each half of the 64-bit register can be accessed using 32-bit access
32-bit access to non-existing register address	Decode Error
64-bit access to non-existing register address 64-bit access to an address which is aligned to 32-bits Read access to secure register with AxPROT[1] = 1 Write access to secure register with AxPROT[1] = 1 Read/Write access to non-secure register with any AxPROT[1]	Decode Error Decode Error No read performed. 0 data and decode error response is returned No write performed. Decore error response is returned 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 5: 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

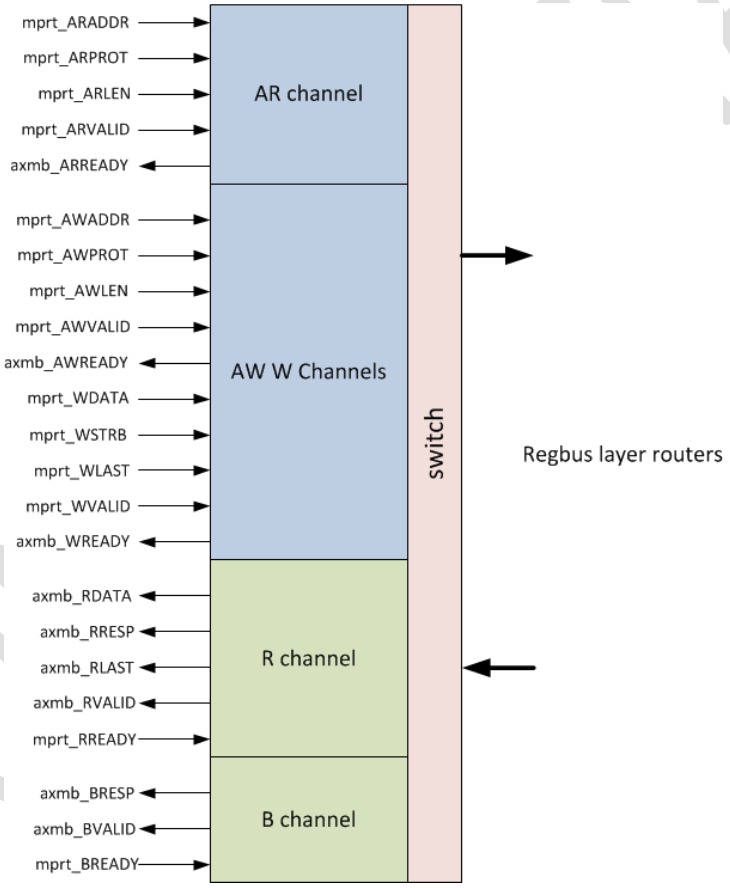


Figure 10: Register bus master bridge

The list of input signals is specified below:

Table 6: 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

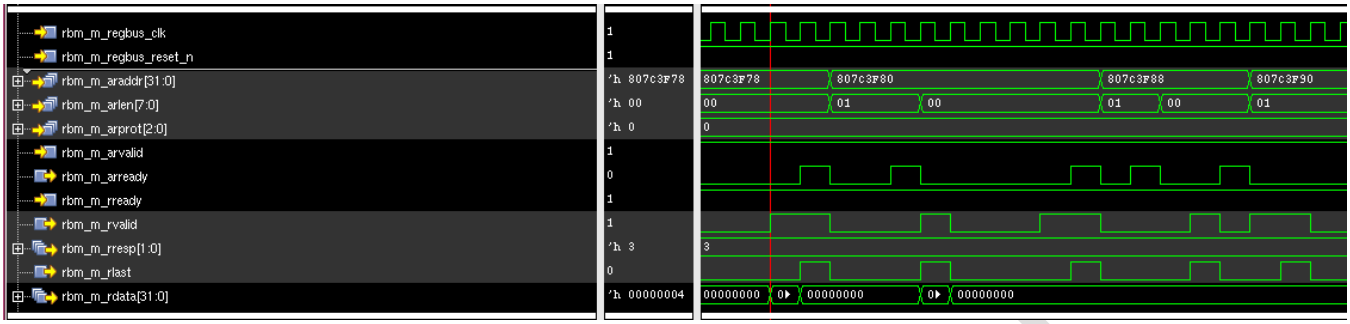


Figure 11: Waveform showing read requests and responses at the register bus master interface

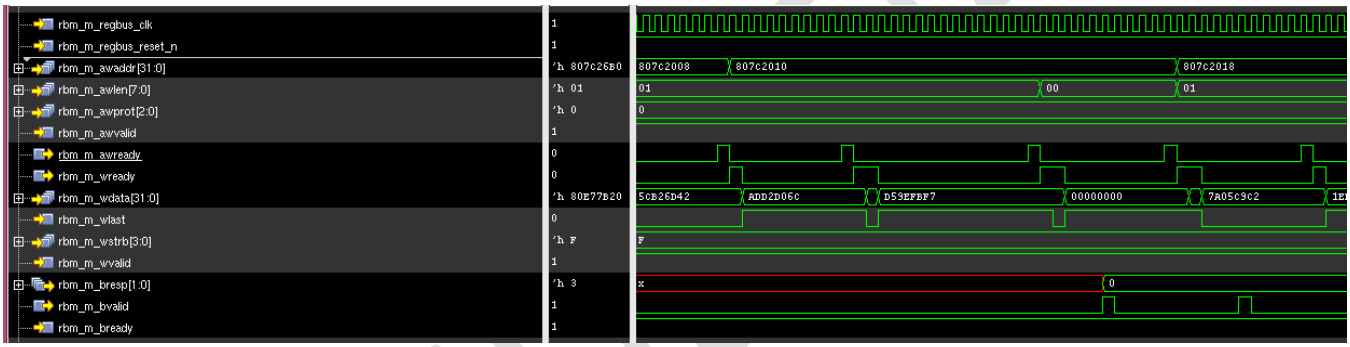


Figure 12: Waveform showing write requests and responses at the register bus master interface

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 7: 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

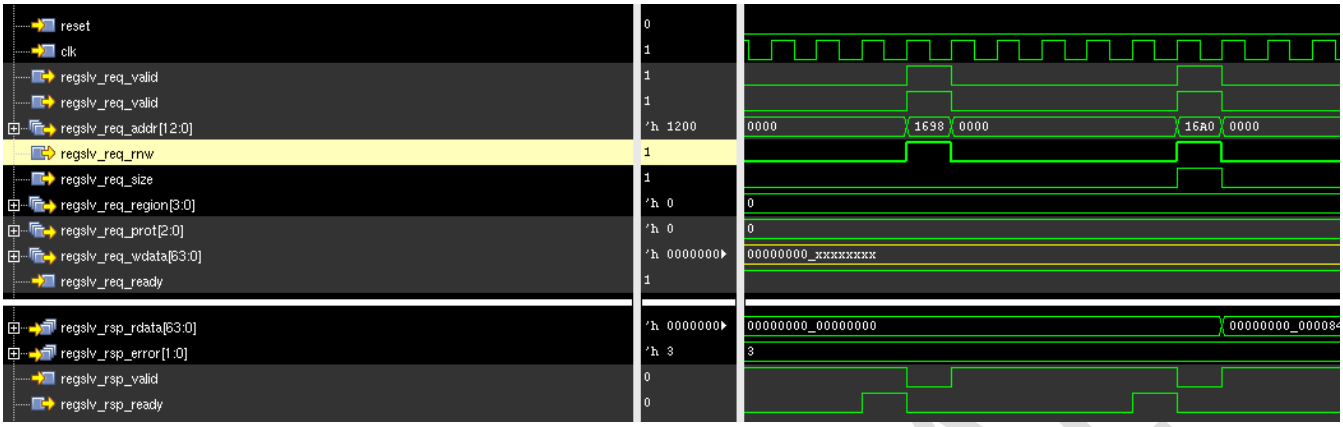


Figure 13 : Waveform showing ring slave read requests and responses (4B and 8B)

Figure 13 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.

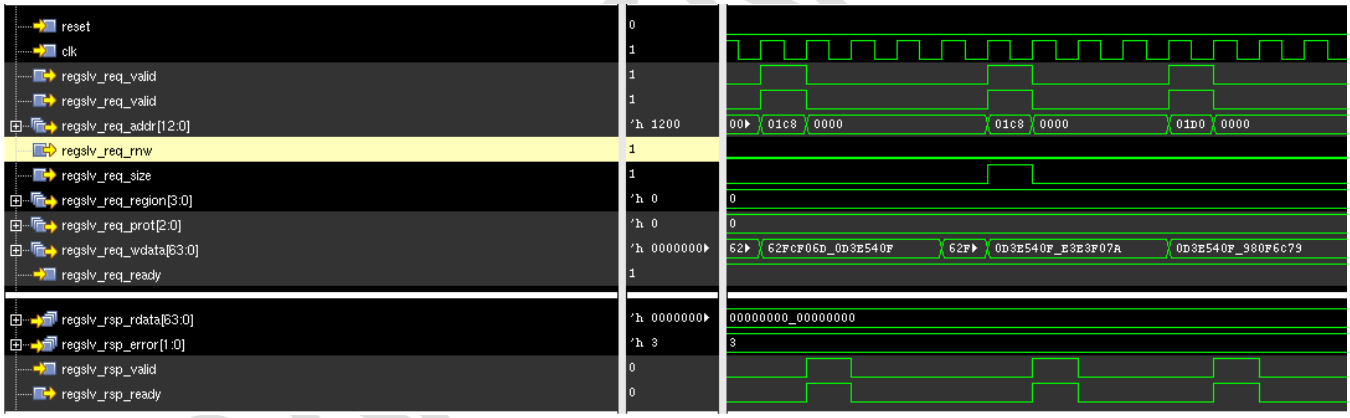


Figure 14 : Waveform showing ring slave write requests and responses (4B and 8B)

Figure 14 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.

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, in brief, the regsisters available at SoC designers. Complete details on the NoC registers can be found in the custom NoC reference manual generated by NocStudio.

6.1 STREAMING BRIDGE REGISTERS

6.1.1 QoS Profile Data (P)

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

6.1.2 Bridge Receive FIFO Status (BRS)

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

6.1.3 Bridge Rx Upsizer Status (BRUS)

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.

6.1.4 Bridge Tx Upsizer Status (BTUS)

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.

6.1.5 Tx Bridge ID (TXID)

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.

6.1.6 Rx Bridge ID (RXID)

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.

6.1.7 Streaming TX Rate Limiter (BTRL)

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.

2670 Seely Ave

Building 11

San Jose, CA 95134

(408) 914-6962

http://www.netspeedsystems.com