NetSpeed Gemini Technical Reference Manual

Version: GEMINI-16.04.1

June 24, 2016

NetSpeed Gemini 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 Gemini Cache Coherent Interconnect.

Audience

This document is intended for users of NocStudio:

- NoC Architects
- NoC Designers
- SoC Architects

Prerequisite

Before proceeding, you should generally understand:

- Basics of Network on Chip technology
- AMBA interconnect standards

Related Documents

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

- NetSpeed NocStudio Gemini User Manual
- NetSpeed Gemini 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 Gemini Overview

NetSpeed Gemini is a cache-coherent, high-performance Network-on-chip (NoC) IP that is used for rapidly designing and analyzing highly efficient and scalable cache-coherent interconnects for a wide variety of SoCs. To quickly produce efficient high-performance cache-coherent NoC IPs, Gemini uses a requirements-driven design approach and an innovative directory-based design. Using Gemini, SoC architects can connect anywhere from 1 to 64 fully cache-coherent CPU clusters, GPUs or other compute units. It also supports 1 to 200 I/O coherent and non-coherent agents. Gemini is built upon following the following fundamental design principles.

1.1.1 Requirements driven approach

Gemini is based on a highly distributed architecture, where both the interconnect fabric and the coherency components can be scaled independently. Based on system requirements such as cache capacity and total coherent bandwidth, coherency components are added, customized and placed in the interconnect. The fabric itself can be designed and customized based on fine-grained requirements such as total and per-flow system bandwidth and chip layout. Gemini uses an algorithmic directory whose capacity scales linearly with number of caches and limits all unnecessary snoop traffic giving power efficiency and faster responses. Furthermore, the

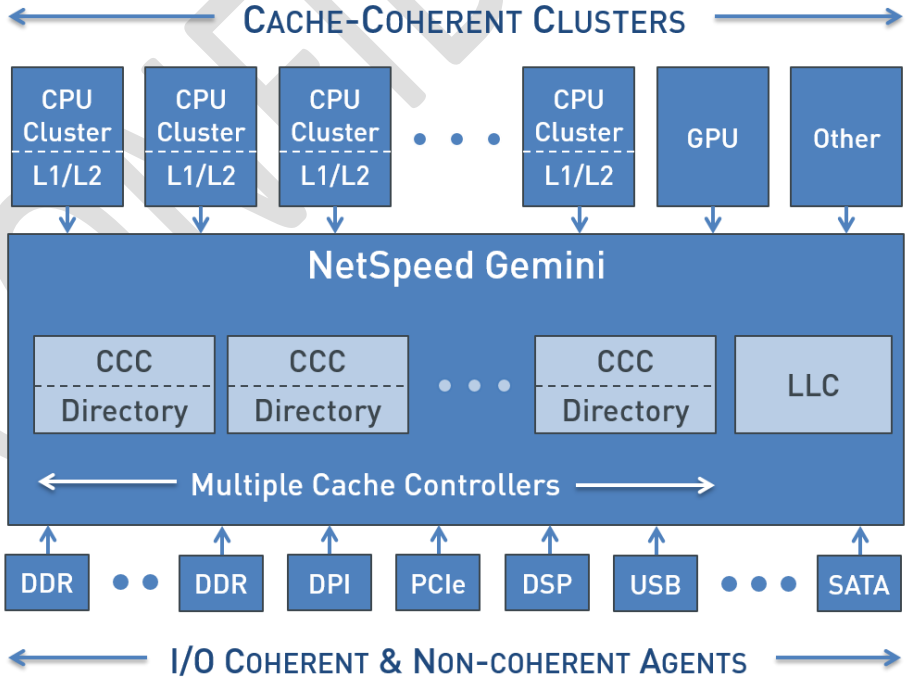


Figure 1: NetSpeed Gemini Overview

directory is partitioned in several slices and distributed across the fabric to provide unlimited scalability and flexibility. Another unique advantage of Gemini is that it can support both coherent and non-coherent traffic in the same design, which avoids the need to build multiple interconnects and inefficient bridges to connect them. Figure 1 illustrates the logical picture of a Gemini interconnect, where a variety of non-coherent and coherent masters and slaves are interconnected through a fabric which includes a distributed cache coherency directory and last level cache (LLC).

1.1.2 Physically aware latency optimized design

Gemini 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. Furthermore, Gemini's IP components can be partitioned, distributed and co-located near the coherent masters that typically utilize them, providing improved latency and power efficiency. 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, Gemini NoC topology, coherency and fabric components, and their placements are automatically computed using machine-learning and graph theory algorithms to optimize the design for area & power.

1.2 CONFIGURABILITY OPTIONS

NetSpeed Gemini provides user configurability and flexibility across multiple design dimensions. In addition to providing flexibility of number of ports, coherency participation, etc., significant configurability is also provided to the architect in defining the cache hierarchy and asymmetric traffic based on the system characteristics such as hit rates. Gemini is configured and optimized using NocStudio - an architecture exploration platform based on machine learning and graph theory algorithms. Using NocStudio, architects can design interconnects with high level specification and tradeoff power, performance and area with real-time feedback on SoC bandwidth and latency, coherency performance etc. from NocStudio SoC performance simulator.

1.3 INTERFACES

1.3.1 Ports and Port Protocols

Since Gemini is an extension of Orion, it supports all of the Orion functionality and port types. This means AXI4, AXI3, AXI-lite, AHB and APB port types can all be added to a Gemini configuration.

In addition to non-coherent port types, Gemini support 3 additional port protocols. These are ACE, ACE-lite, and ACE-lite+DVM.

1.3.2 ACE

The ACE interface protocol is used for fully cache coherent agents, like a CPU or a coherent GPU. ACE includes read and write channels, and their corresponding response channels. It also has channels for snoops and snoop responses. Finally, it has additional signals called RACK and WACK used as part of the coherency protocol to indicate response arrival.

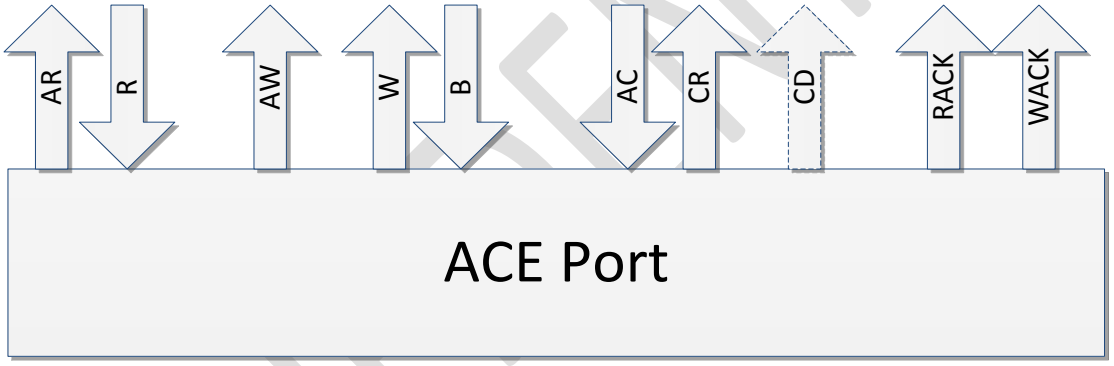


Figure 2: ACE Port

The figure above shows the ACE port, including the various channels.

Note that the CD channel, which is used for snoop data response, is an optional channel. Gemini allows an ACE port to be configured with or without the CD channel.

1.3.3 ACE-lite

The second standard port protocol in the coherency space is the ACE-lite port. This port is designed for agents without caches that want to perform reads and writes to the coherent address space. WriteUnique (or WriteLineUnique) allows writes to perform coherent snoops and invalidation/flushes of a cache line before writing to the line. This ensures writes will be seen by cache coherent devices since they'll have to read the new cache line value from memory. A ReadOnce command reads a copy of a line, but goes through the coherency mechanism to make sure that it sees the most recent copy.

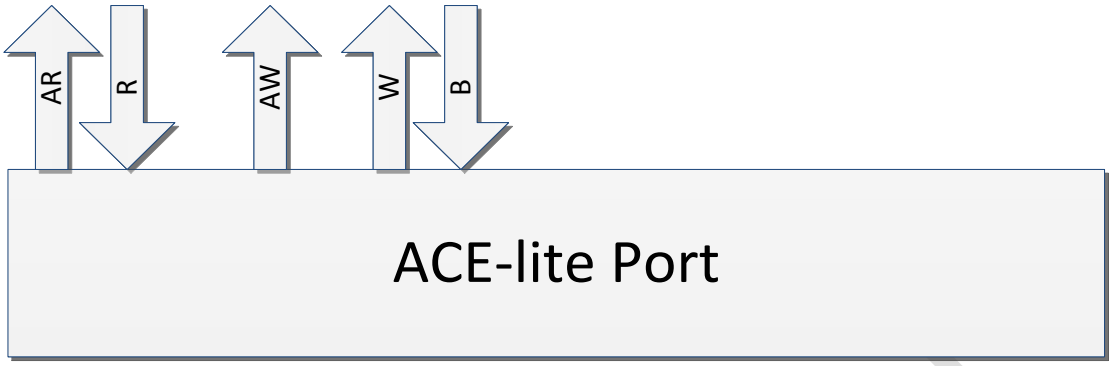


Figure 3: ACE-lite

The figure above shows the ACE-lite port interfaces. There is no snoop or snoop response channels, and no RACK or WACK signals.

1.3.4 ACE-lite+DVM

Unlike the ACE and ACE-lite channels, there is no explicit reference to the ACE-lite+DVM port protocol within the AXI4/ACE specification. This hybrid protocol looks similar to an ACE interface without the CD channel, but behaves differently.

This protocol is intended to be used by agents that have no caches but do have a built-in MMU. The MMU-400 or MMU-500 products are examples of this protocol.

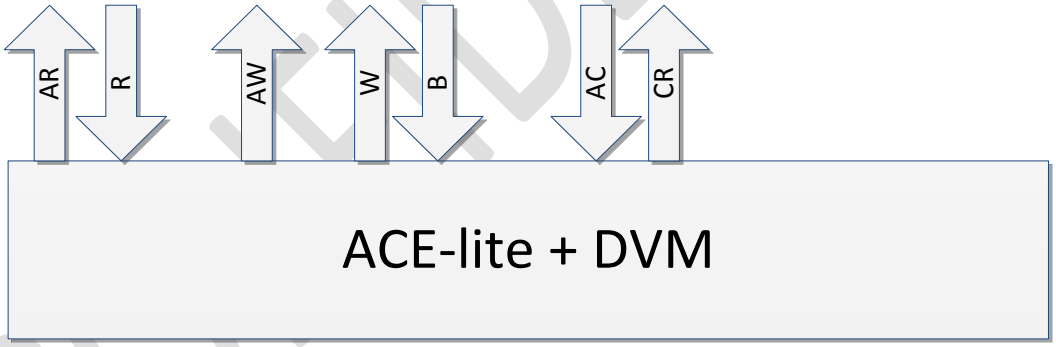


Figure 4: ACE-lite+DVM

As shown in the figure, this agent has the typical read and write paths of an ACE-lite agent, IO coherent reads and writes. It also has a snoop and snoop response channel (AC and CR), which are used only for DVM snoops since the agent never keeps a cached copy of a line. Since the DVM protocol does not use a CD bus (it never sends data with the snoop response), it is not included in this interface. It also has no need for the RACK or WACK signaling.

1.3.5 ACE-lite Converted Bridges

Some components use older non-coherent protocols, like AXI 4. While the components are built as non-coherent, they may want to participate in the coherency protocol using IO coherency.

Gemini supports conversion from some of these legacy protocols to ACE-lite. Eligible requests will convert simple reads and writes into ReadOnce and WriteUnique requests.

The following protocols can be converted to ACE-lite:

1. AHB-lite Master
2. AXI 4 Master
3. AXI 3 Master

The conversion hardware is built into the Gemini bridges and can be enabled during NoC construction using NocStudio. A bridge property called *cc_acelite_conversion* will convert the bridge to Ace-lite.

Bridges setup for ACE-lite conversion allow both IO-coherent and non-coherent requests to be made. For any address range that does not support coherency, the request will stay as ReadNoSnoop or WriteNoSnoop, with AxDOMAIN set to System Shareable or Non-Shareable, depending on the AxCACHE bits. When AxCACHE[3:2]==2'b00, the request is mapped to System Shareable. Otherwise, it is mapped to Non-Shareable.

For coherency ranges, the AxCACHE bits control whether a request will participate in the coherent protocol. If AxCACHE bits indicate non-cacheable (AxCACHE[3:2]==2'b00), the requests will be sent as WriteNoSnoop and ReadNoSnoop with AxDOMAIN = System Shareable.

If AxCACHE[3:2] indicates a cacheable space, the request will be converted to a ReadOnce or WriteUnique, with AxDOMAIN equal to Outer Shareable.

Note that in the non-converted bridges, these would be sent as Non-Shareable domain and WriteNoSnoop or ReadNoSnoop.

One exception to this conversion is when a request is sent with AxLOCK==1. Since ACE doesn't support the Exclusive functionality for ACE-lite, this request will be issued as ReadNoSnoop or WriteNoSnoop with either System or Non-Shareable domain.

1.3.6 Master and Slave ports

The port protocols described above (ACE, ACE-lite, ACE-lite+DVM, and legacy->ACE-lite converted) cannot be used uniformly as master or slave ports. The following table describes the limitations.

Table 1: Coherency Port Protocol Restrictions

Protocol	Master Port Type	Slave Port Type	Description

ACE	Yes	No	ACE port is used by coherent masters. ACE slave interface only exists on interconnect IP, so agents cannot use this as a slave port type.
ACE-lite	Yes	Yes	An ACE-lite slave can accept barriers, cache maintenance operations, as well as reads and writes.
ACE-lite+DVM	Yes	No	Like the ACE port, only a master can utilize this port protocol.
ACE-lite conversion	Yes	No	Only masters can convert to ACE-lite.

2 Functional Description: Coherency Components

2.1 Coherency IP Components

There are three coherency IP modules that are included in Gemini in order to handle the coherency functions of the chip. These IP modules must be instantiated as agents within the system and connected to the NoC like a typical agent. This section will describe these modules and their expected usage.

2.2 CCC (Cache Coherency Controller)

The CCC is the primary coherency control module. This is where coherent reads are sent to perform coherency lookups and to issue snoops if required. Since coherent requests are sent to the CCC before being sent to memory, the CCC should be instantiated near the most latency sensitive coherent agents. Typically these will be the CPUs.

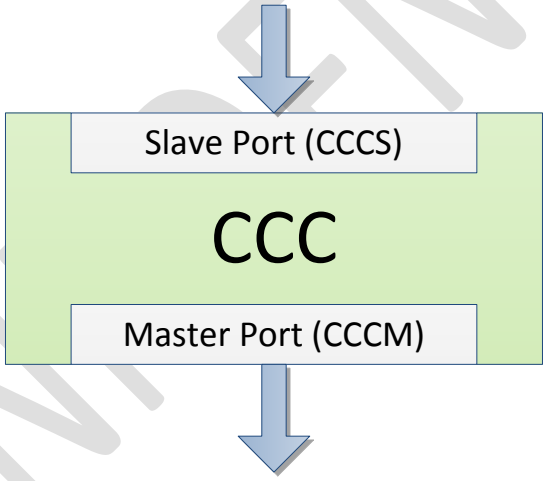


Figure 5: CCC Port Diagram

The figure above shows the CCC with two ports. One is the slave port, where the CCC accept coherent requests from CPUs or other coherent agents, and one is the master port, where the CCC issues reads or writes to memory or to a cache in order to satisfy a coherent request.

The slave port protocol for the CCC is called CCCS. The master port protocol is CCCM. The slave port should be configured to be near the latency sensitive coherent requesters such as CPU. The master port should be configured to be near the destination cache or memory.

2.2.1 Multiple CCCs

In some Gemini systems, multiple coherency controllers can be instantiated. Each CCC will be responsible for a subset of the address space. The assignment of address responsibility has

significant flexibility. Each can be assigned their own range or ranges of addresses, or they can share a set of address regions and slice the range between them. For instance, two CCCs could split an address range into even and odd cache lines, with one CCC handling the even lines and one CCC handling the odd lines.

There are several reasons for instantiating multiple coherency controllers.

2.2.2 Increased Coherent Bandwidth

A single coherency controller has limitations on the bandwidth it can handle. The data interface width at the CCC is 32 bytes wide. While read and write channels can both be active at the same time, the data width limit puts an upper bound on total number of cache lines that can be processed by a single controller. At 1GHz, this puts an upper bound of 32GB/s of read and 32GB/s of write bandwidth on a single CCC.

A different limitation of bandwidth is that coherent requests must access the directory RAM that store the coherent state of the system. Since this is a single-ported RAM, the access of the RAM can be a limit. Cache coherent reads can take two accesses (one for the lookup, and one for saving the new state). Writebacks/Evicts can also require two accesses (one for a lookup, one to save the new state). IO coherent requests will typically only have one access for the lookup, but can require an additional access if snoops are sent and state is modified.

Additional coherency controllers increase bandwidth approximately linearly. Two controllers will have 2x the bandwidth (2x the port bandwidth, and 2x the directory bandwidth). This assumes traffic is somewhat uniformly split between the two CCCs, which can be done by using lower order index bits.

2.2.3 Reducing Directory Size to Decrease Latency

The Directory is a RAM, and so will have an increased latency with more storage. Creating additional CCCs can split the directory size, potentially allowing a decrease in latency or an increase in bandwidth.

2.2.4 Minimizing Latency with Location

In some systems, multiple coherency controllers may be used by placing them in different locations within the chip, and assigning address spaces so that physically close agents will send the bulk of their traffic to the nearby coherency controller. This produces a NUMA (Non-Uniform Memory Access) type of chip design. Since the CCCs are located closer to the agents accessing them, system latency can be reduced.

2.3 IOCB (IO COHERENCY BRIDGE)

The IOCB is an IP block that is responsible for IO coherent accesses (WriteUnique, WriteLineUnique, and ReadOnce). It accepts requests from ACE-lite, ACE-lite+DVM and ACE-lite Converted bridges. It can be added with the *add_iocb* command in NocStudio.

Non-cached traffic is traffic issued by an agent that doesn't use a cache. This includes non-coherent traffic as well as IO coherent traffic. One of the key issues with these non-cached requests is the need for ordering of the requests. Sharing of data by multiple agents utilizes various sharing models that rely on the order in which request are completed. For instance, writes frequently need to be ordered to ensure that prior writes are visible to an agent before later writes. If a master writes data and then stores a flag to indicate that the data is valid, an agent that reads the flag must be sure that data is visible.

There is frequently a tradeoff between satisfying ordering requirements and achieving high bandwidth. This is because a common method of enforcing ordering between two requests is to wait for the first to complete before issuing the second. If ordering requirements are common, this can have a significant reduction in total bandwidth.

The IOCB is built to enable high bandwidth while allowing frequent ordering requirements. It is an IO coherency accelerator. It takes advantage of the inherent parallelism of coherency to prefetch coherent permissions for the requests. Once permissions are granted, it can commit the operations with high bandwidth without the need for serialization.

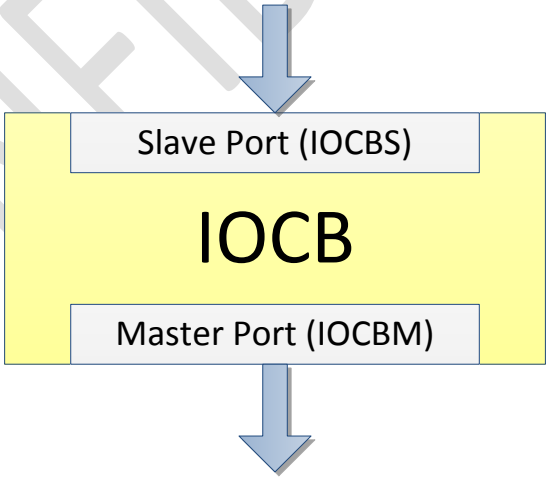


Figure 6: IOCB Port Diagram

The diagram above shows the IOCB agent and its two ports. The slave port accepts requests from IO coherent masters (ACE-lite, ACE-lite+DVM, ACE-lite converted). The slave port protocol is specified as IOCBS. The master port issues coherent requests to and from the CCC, and the protocol is specified as IOCBM.

2.3.1 Multiple IOCBs

A system can have more than one IOCB, just as it can have more than one CCC. There is no relationship between these, however. A system could have multiple CCCs and only one IOCB, or a single CCC and multiple IOCBs, or any combination.

While a CCC is assigned to an address range, the IOCB is assigned to masters. This means a master only ever connects to a single IOCB. If multiple IOCBs are added to the system, each master will be assigned to only one of them.

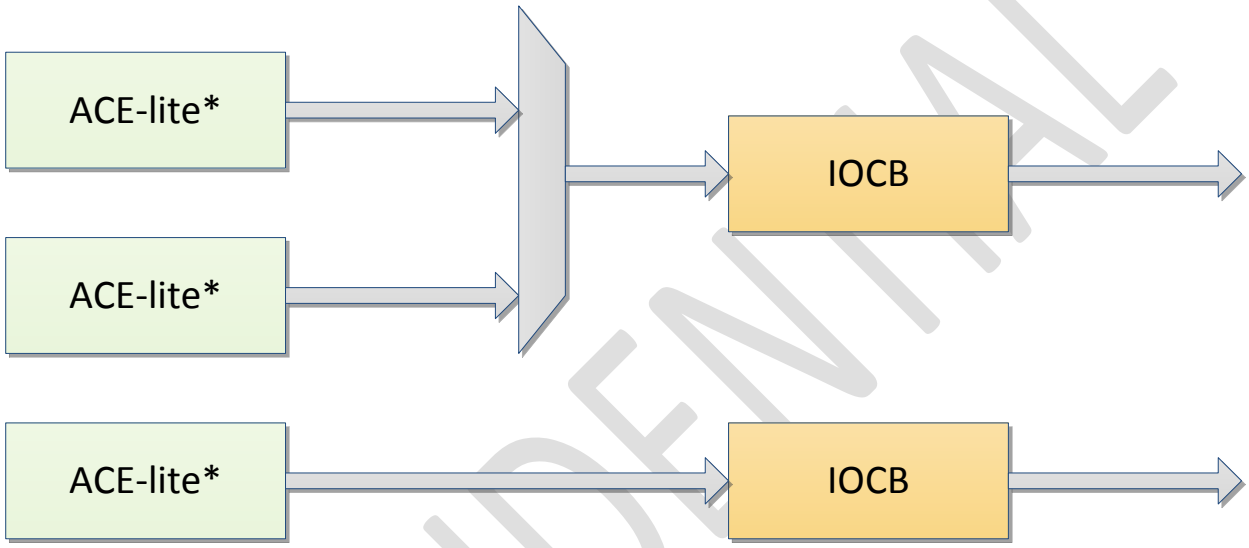


Figure 7: Multi-IOCB diagram

As the figure above shows, each master only sends traffic to a single IOCB, even when there are multiple IOCBs in the system. However, multiple masters can send to the same IOCB. Sharing the IOCB reduces total hardware costs and allows dynamic sharing of resources.

Just as in the case of the CCC, the IOCB has limits on its total bandwidth. The IOCB data ports are 32 bytes wide, so it can only transmit 32B of read data and 32B of write data per cycle. At 1GHz, this is 32GB/s of read and 32GB/s of write bandwidth.

2.4 DVM (Distributed Virtual Memory Controller)

Distributed Virtual Memory is an independent communication protocol that utilizes the ACE communications channels. The DVM module can be instantiated using the *add_dvm* command in NocStudio. Details of the DVM protocol can be found in the AMBA ACE specification.

DVM is used to broadcast and synchronize control packets for TLB invalidations, Instruction Cache invalidations, and similar requests. Generally it is used to maintain non-coherent hardware storage structures when the underlying data changes.

The interconnect has two primary functions related to DVM. The first is that when a DVM message is sent from a master, it is broadcast to all other DVM-enabled agents (ACE or ACE-lite+DVM). The broadcast is sent in the form of a snoop request to these other masters. The second function is a DVM synchronization processes, which includes sending synchronization snoops, gathering completion requests from the DVM agents, and eventually signaling back to the synchronizing master that the request has completed.

To satisfy the DVM protocol requirements, a single DVM agent must be instantiated within the network. This agent broadcasts the snoops, gathers responses, and replies to the original master. It also tracks synchronization requests and verifies that all DVM completion requests have been issued

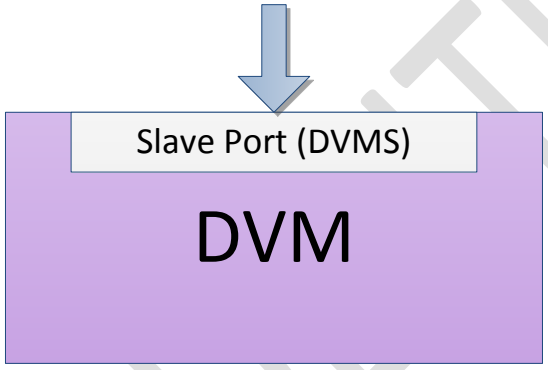


Figure 8: DVM module port diagram

As seen in the diagram above, the DVM has a single port which accepts DVM requests and issues corresponding snoops as needed. This slave port protocol is specified as DVMS.

Unlike CCC or IOCB, only a single DVM module can exist.

2.4.1 Systems without DVM

It is possible to construct a coherent system without DVM. This may be desirable when the coherent agents do not support the DVM protocol, so adding a DVM module would be unnecessary. To ensure that a lack of DVM is intentional, NocStudio requires the mesh property *cc_dvm_support* to be set to 'no'. If this property is not explicitly set to 'no', the NoC generation will signal an error indicating that the DVM module was not added.

2.5 DIRECTORY CONFIGURATIONS

Gemini utilizes a directory-based coherency protocol. This means that the coherency controller(s) have a storage structure that tracks which lines are in the various coherent caches. A directory is a kind of snoop-filter, and its job is to allow the coherency controller to locally determine the state of a cache line without sending snoops to the caches. This allows lower

latency accesses, reduced network bandwidth, reduced snoop bandwidth at the masters, and higher peak bandwidth of the system.

The directory stores cache line addresses, state information about each address, as well as information about which of the coherent caches the line is present in. The directory does not store cache line data. It is not a cache. It only contains address and state information.

The directory is built to use single-ported RAM arrays for the storage. The RAM implementation is not included with Gemini and must be provided by the customer. The RAM design details can be supplied to NocStudio during the Gemini construction so that the final RAM design can be swapped in seamlessly later.

2.5.1 Building Gemini with no Directory

It is possible to build a Gemini system with no directory. In this system, each request would have to issue snoops to every ACE master in the system, and wait for snoop responses before a reply can be returned. This can reduce or limit coherent bandwidth and will likely increase latency of requests, but it can save area for customers who need to minimize the footprint of the hardware.

When an ACE master is added in NocStudio, it can be marked as having no snoop filter support using the bridge property *cc_snoop_filter_support* . If snoop filter support is disabled, it means coherent requests will have to send snoops to each master. If all ACE masters are set up that way, no directory would exist at all.

The CCC is designed to be able to send one snoop to a single destination per cycle. This means that the coherent bandwidth can be significantly reduced if the number of ACE masters is large. If a system had 4 ACE masters, it could only process a request ¼ cycles at most, since it would take that long to send all of the snoops. To get more coherent bandwidth, more CCCs can be used. However, there is a finite amount of snoop traffic that an ACE master can receive. It can only accept one snoop per cycle because of the single snoop port. So coherent bandwidth has a fundamental limit with this snooping configuration.

2.5.2 Mixing Directory And Snoop Agents

Gemini allows a mix of coherent agents that do or do not use the directory. This can lead to more complex design tradeoffs.

When some agents are tracked by the directory, and other agents are snoop-only, the benefits of the directory has the potential to be significantly reduced. A directory provides more coherent bandwidth and lower request latency. But if snoops need to be sent to some agents on every request, latency will go up and bandwidth will be reduced and ultimately limited by the snoop ports. Without additional modifications, this combination would behave poorly.

To avoid these issues, Gemini can support the Inner Shareable and Outer Shareable domains. The behavior of the system obeys the following rules:

Table 2: Inner and Outer Shareable Domain Visibility

Requesting Agent	Behavior of Different Domain Type			
	Inner Sharable Domain		Outer Sharable Domain	
	Directory lookup?	Snoop sending to snoop-only agent?	Directory lookup?	Snoop sending to snoop-only agent?
ACE (Directory support)	Yes	No	Yes	Yes
ACE (Snoop only)	Yes	Yes	Yes	Yes

As the table shows, every access will look up the directory, since this incurs no latency or bandwidth cost. The snoop-only agents only get snooped under some conditions. Other requests from snoop-only agents will always send the snoops. But for directory supporting ACE agents, only their Outer Shareable accesses will send snoop. So directory agents can realize the benefits of a directory even when there are snoop-only agents by accessing regions as Inner Shareable.

2.5.3 Possible Uses of Mixing Directory and Snoop-only Agents

There are a lot of different combinations that could be useful for mixing these systems. For instance, a GPU might have a very large coherent cache and keeping directory state of that cache could be expensive. In that configuration, the GPU could be marked as snoop-only. This would allow the GPU to still access coherent memory and view the CPU cache contents, but it would enable the CPUs to ignore the GPU in address regions it knows the CPU is not using.

Another possible use is in a system with an external link to a coherent subsystem. Since the coherent subsystem is not part of the chip and could change over time, it may be desirable to satisfy requests to that external system without directory overhead. But the latency costs for sending snoops may be substantial. Using the Inner Shareable domain would allow local agents to interact quickly while limiting the amount of global communication in the Outer Shareable domain.

2.5.4 Sizing the Directory

The first step in configuring the directories is to determine their capacity and associativity.

Any line present in a coherent cache must be tracked by the directory. If the directory doesn't have room for a new address, it will have to evict an existing address. To guarantee inclusivity,

this will result in a cache line flush being sent to the caches to remove the evicted address. This can lead to the caches being underutilized. This is called a directory conflict, and can result in a reduction in performance due to the underutilization of the caches. To avoid this, the directory needs to be sized appropriately.

The first recommendation is that the capacity of the directory is at least the capacity of the caches has to track. For instance, if there are 2 caches with 1MB each, the directory should be able to track 2MB of total cache capacity. Anything less than the cache capacity will result in underutilization of the caches.

Gemini supports significant heterogeneity in the system. It allows different coherent caches to be sized differently.

For example, one cache might have 2MB, another might have 512KB, while a third only has 128KB. The total cache capacity to be tracked is 2.625MB.

The directory utilizes a set-associative organization. This means it will have a power-of-2 number of sets, with some number of ways per set. This format puts a practical limitation on the granularity of the capacity. To ensure directory capacity >= cache capacity, the directory can be oversized if necessary. Configuring the sets and ways appropriately can minimize how much oversizing is necessary. For instance, a 2MB directory may be organized as 8 ways and 4096 sets. If 2.1 MB is needed, the directory can choose different options, such as 8 ways and 8192 sets (4MB), or 10 ways and 4096 sets (2.5 MB of cache). Varying the number of ways can increase capacity on a non-power-of-2 scale.

The capacity of the caches is not the only factor that should drive the sizing of the directory. Another similar factor arises from the fact that the directory is a shared data structure for all caches.

To truly guarantee that any combination of lines within the caches can fit within the directory, the directory would have to have the same indexing method as well as associativity of all of the caches. For instance, if 4 caches each have 8 ways, the directory would need a minimum of 4x8=32 associative ways to guarantee the lines in the caches can all fit in the directory. With less associativity, some combinations won't fit in the directory and conflicts can occur.

The Gemini directory is not intended to match the associativity or indexing method of the caches. Instead, it is implemented as a shared structure with limited associativity. It utilizes several innovations to reduce the likelihood of directory conflicts, as well as methods of reducing the cost of these conflicts when they occur. However, since conflicts are still possible, additional steps can be taken to decrease conflicts even more.

One available tool to avoid conflicts is to increase the directory capacity beyond the capacity of the caches. With a directory that has a larger capacity than the caches, there will be unused entries in the directory. This means that when the caches put added pressure on a set in the directory, the directory will be able to absorb additional lines without the need for a conflict. So to increase performance of the system, the directory can be slightly oversized.

There are two properties to set in order to size the directory. One describes the number of sets of the directory, while the other describes the associativity. Both are properties of the CCC host. The first is "cc_directory_associativity" and the second is "cc_directory_index_width".

The associativity must be between 8-16, and must be a multiple of 2. This is because the directory is organized as 2 tag arrays, with equal number of associative ways.

The index width is the log2 (#sets). A directory with 4096 sets will have an index width of 12 bits.

If not defined, these properties may be automatically set up using the setup_coherency command, described in a later section of this document.

2.5.5 Selecting Index and Tag Bits

Two related properties are available for customer to configure: cc_directory_index_bits and cc_directory_tag_bits. These properties specify the actual address bits that are used for index bits and for tag bits. This allows the use of higher order bits for indexing if desired.

However, the most common choice of index bits and tag bits will use the lowest order bits for indexing, and higher order bits for tagging. These properties will automatically be configured with the setup_coherency command, and therefore they should not normally be set by the user.

2.5.6 Directory RAM Specification

The directory storage is intended to be implemented using single-ported RAM arrays. RAM is needed because of the quantity of storage needed which can be significant. The RAM is single-ported in order to make it as generic as possible and easier to implement.

The directory RAM is organized as logical arrays. The actual RAM design could further divide these arrays, but logically they will be treated as two arrays.

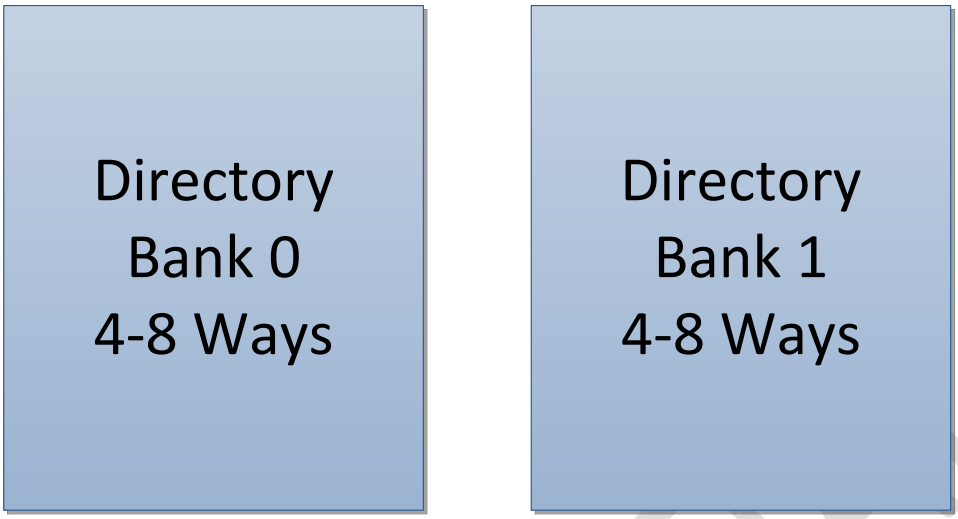


Figure 9: Directory organized as two tag arrays

The two RAMs are accessed in parallel during a lookup, but can be written individually. The index of the access can be different for the two RAMs in the same cycle, which requires that they be separate RAM instances.

While multiple ways are contained in the RAM, an entire row is accessed at a time. So the RAM design is not aware of the number of associative ways. It only sees an array of data blocks. The width of the data block is determined by NocStudio based on the required number of address bits, associativity, etc.

Since the RAM arrays are implemented by the customer, certain characteristics of the RAM must be provided to NocStudio to configure the hardware to access it correctly. The logic must know the latency of a read request, as well as the bandwidth of the RAM accesses. These are two different properties that can be set for the directory in the CCC host property list.

```
cc_directory_latency
cc_directory_bandwidth_denominator
```

The first value specifies the latency of the RAM lookup. The RAM design is expected to have a flopped input and a flopped output. The latency value describes the number of cycles it takes to perform the RAM lookup. A latency of 1 means that the RAM will take a full cycle to do it's lookup, but that data will actually arrive after two flop points. The following diagram shows a RAM latency of 1.

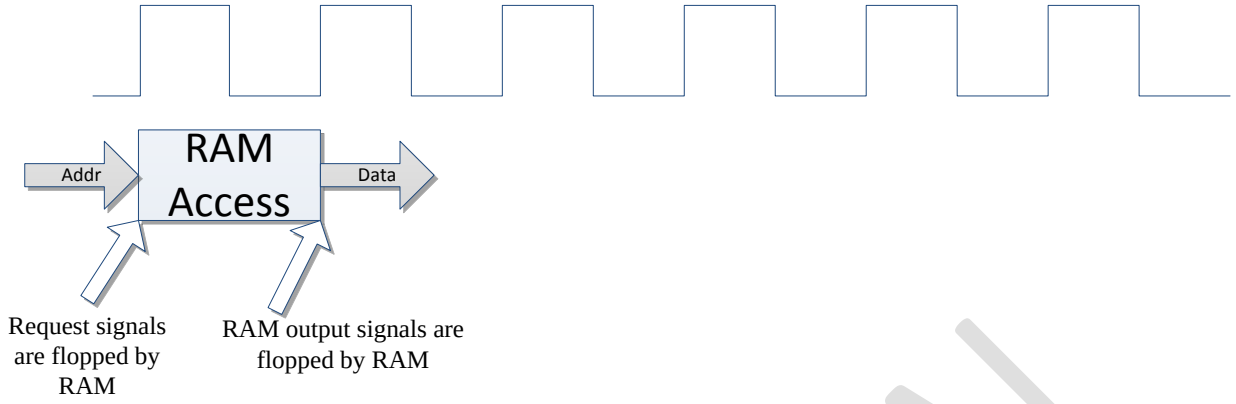


Figure 10: Example RAM latency == 1

As noted, when a RAM latency is set to 1, it means that if the request is sent on cycle N, the data is available during cycle N+2 since both inputs and outputs are flopped by the RAM.

The second property, "cc_directory_bandwidth_denominator", is used to indicate the bandwidth of the RAM. Depending on the frequency of the design and the size of the directory, it may be necessary to build the directory RAM with less bandwidth, such as only being able to perform an access once every other cycle.

This property is a bandwidth denominator. The bandwidth can be stated as 1 access/N cycles, where N is the value of this property. A value of 1 will indicate that a new access can happen each cycle. A value of 2 means one access every 2 cycles, etc. The maximum value is 4.

The following diagram shows a pipeline description of a directory RAM with a latency of 2 and a bandwidth denominator of 2.

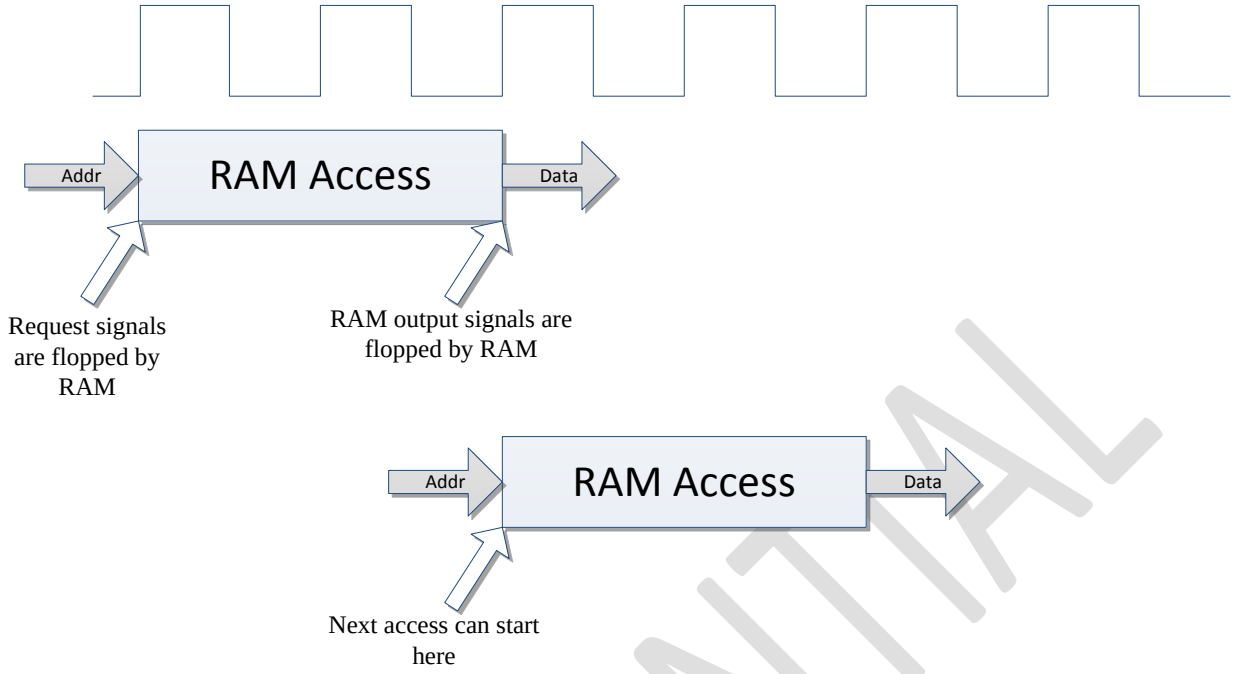


Figure 11: Example of 2 cycle latency, 1/2 cycle bandwidth

It is possible for the latency and bandwidth numbers to differ. A RAM might have a long latency due to travel time to the sub-arrays, but may have some ability to be pipelined.

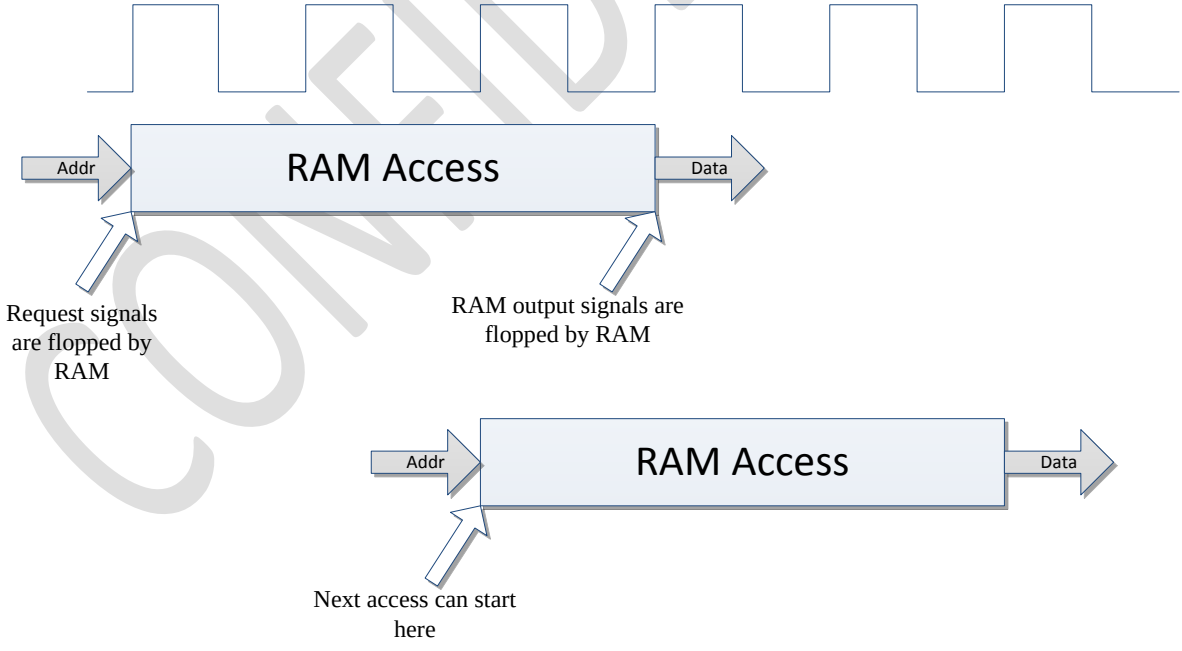


Figure 12: Example directory RAM with 3 cycle latency, 1/2 cycle bandwidth

As seen in the diagram above, the RAM may have a latency of 3 cycles, but a bandwidth of 1access every 2 cycles. With these two properties, the logic can adapt to most RAM design requirements.

While the directory has two logical arrays, the arrays must be implemented identically. That means the latency and bandwidth properties are common to both arrays.

2.5.7 ECC Support in Directory

The Directory supports ECC protection for the RAM. The ECC generation and checking happens in the CCC control logic, and the RAM itself only changes by having additional storage bits per entry. ECC can be added during configuration time with the following CCC host property:

```
cc_directory_ecc_enabled
```

NocStudio will calculate the minimal number of additional bits needed to support ECC, and add them to the RAM data width automatically. The approximate size of the ECC bits is log2(entry width)+1. So 150 bits of data would require 9 bits of ECC overhead.

The ECC algorithm allows for single bit error correction, and double bit error detection.

2.5.8 Parity Support in Directory

Instead of ECC protection of the Directory RAM, Gemini can configure the system to use Parity protection. While ECC can both detect single or double bit errors and correct single-bit errors, the Parity protection can only detect single-bit errors, and has no correction capabilities. It uses only a single additional bit, instead of the log2 bits needed for ECC, so this can provide some protection at a smaller cost.

The following CCC host property can be used to add parity to the directory.

```
cc_directory_parity_enabled
```

2.5.9 BIST and DFT Support for RAMs

Since the directory RAM is implemented by the customer, the BIST and DFT hooks for the RAM must also be provided by the customer.

A functional RTL model of the RAM is instantiated in Gemini RTL hierarchy. For the primary RAM functionality, the module within the CCC can be simply replaced with the more accurate customer RAM RTL module. However, to add additional hooks for BIST or DFT, additional signals must be made available to that module. To enable this, additional properties are created

within NocStudio to route additional signals through the RTL hierarchy to the RAM instance. These properties are cc_mem0_in_width, cc_mem0_out_width, cc_mem1_in_width, and cc_mem1_out_width.

A generic bit vector can be passed to and from each RAM, with the width of this vector specified by these properties. If DFT/BIST needs 15 signals coming from the top level to the RAM, and 1 output, the values can be set to 15 for the in_width, and 1 for the out_width properties.

2.5.10 Capacity Implications of CCC Slicing

When CCC slicing is used to increase bandwidth, it is usually not necessary for each directory instance to have the full capacity of the cache. Generally, the cache capacity can be divided by the number of slices. So a 2MB cache capacity could be implemented with 4 CCC slices, each with a directory capacity of .5MB.

There are two reasons for this. The first is when address slice bits are the same as the cache index bits, then it is guaranteed that only ¼ of the lines in the cache can reside in a single CCC slice. So if bits [7:6] are used for slicing and for indexing, then ¼ of the lines in the cache will have bits [7:6]==2'b01, so the CCC that is mapped by [7:6]==2'b01 will only need ¼ of the capacity.

Even if the cache uses different address bits, a random distribution of addresses will typically only need ¼ of the cache capacity in each of the 4 CCC slices. If each slice supported the full capacity, they would typically be only ¼ utilized.

So for most designs, multiple CCCs can collectively match the capacity of the caches. This means the cost of slicing the CCC is only the overhead control logic in the CCC, and the directory RAM which will tend to dominate the area will not need to increase.

2.5.11 Minimizing Storage

The directory entries are typically very small because they only hold the tag bits and some state bits. This means that the total RAM size is highly sensitive to each additional bit. For instance, a system with a 40-bit address size and 2MB of cache could have a directory with 8 ways and 4096 sets. This means 12 bits of index, 6 bits of offset, and the remaining 22 bits for tag. If 5 more state bits are needed, the total goes up to 27 bits per entry.

Because the number of bits is relatively small, removing any unneeded bits can have a significant impact on the area.

Removing a single bit can reduce the directory size by about 4% in this example.

There are two sets of bits that can usually be removed from the directory. The first is the address slice bits, when multiple CCCs are used in a sliced manner. If there are 4 CCCs, sliced using bits [7:6] of the address, then those two bits do not need to be included in either the index or tag of the directory.

The second common source of unused bits exists because of the address map. Typically, only a portion of the address space is coherent. Usually only the DRAM space, and sometimes a much smaller region used for RAM or ROM. This means that most of the address space is not coherent, and so the directory will never receive requests with these addresses. Since DRAM is typically placed near the bottom of the address map (near 0x0), there are usually higher order bits that are unused. For instance, in a 40-bit address, it may be that coherent address space always has Addr[39:36] equal to 4'b0000. This still leave 64GB of address space that can be used for coherent addresses. Removing these additional 4 bits can further reduce the directory size.

Since Gemini allows for a programmable address map, this reduction cannot happen automatically. Instead, the configuration must explicitly indicate that not all of the address bit will be used by the directory.

2.6 SNOOP FILTER SUPPORT

The normal Gemini model utilizes a directory, and expects ACE masters to provide snoop-filter support, as specified in section C10: Optional External Snoop Filtering in the AMBA AXI and ACE Protocol Specification. This means that when an ACE master replaces a cache line, it is expected to send an Evict request for any clean lines that it is dropping. This allows the directory to know which addresses it no longer need to track.

Gemini can also be configured to support ACE masters without the snoop-filter support (snoop-only agents), including a combination of both types.

Having snoop-only agents can impact both the latency and bandwidth of the coherent system. Latency is increased because coherent requests will always need to send snoops to the snoop-only agents and the round-trip plus processing delay will be added to the latency. Bandwidth is also reduced as snoops are sent one at a time from each CCC. Multiple snoop-only agents will reduce bandwidth by even more.

There may still be useful cases for snoop-only traffic. One example is when a master with a large cache, like a GPU, wants to be coherent but doesn't want to increase the size of the directory. Another example is an external port where the size of the cache may vary.

To allow these cases to exist without significant performance loss, it is possible to distinguish between Inner Shareable and Outer Shareable domains. When snoop-only agents are created,

Gemini will be automatically configured to create three Shareability domains: Inner-Shareable Snoop-Only; Inner-Shareable Directory Agents; Outer Shareable All Agents.

With these regions, directory supporting agents making Inner Shareable requests can benefit from the lower latency of the directory by not snooping the Snoop-only agents. When data may be shared with the snoop-only agents, the Outer Shareable domain can be used.

To specify a master as snoop-only, the master bridge property *cc_snoop_filter_support* can be set to 'no'.

2.7 ACE Master Cache Capacity

ACE masters can have their cache capacity specified using the bridge property *cc_cache_capacity*. This indicates, in MB, how much cache is supported by that ACE master. This value is only used when the ACE master has snoop filter support. In that case, this value indicates to NocStudio how much cache each agent has that needs to be tracked by the directory. This can be used to automatically size the directory for the system.

2.8 Speculative Fetch Control

Gemini provides an optional performance feature. When a coherent read is sent to the coherency controller, it has two choices.

The first choice is for the read to perform the directory lookup, send snoops if required, and wait for the snoop responses. If data was returned by any of the snoops (or by a WriteBack/WriteClean), it can be forwarded to the requesting agent without the need for making a request to LLC or Memory.

The alternative is to perform a speculative fetch. The speculative fetch issues a read to the next stage of the memory hierarchy (LLC or memory) as soon as possible, even before the directory has completed its lookup. This can significantly reduce the latency of the request since it doesn't have to wait for the directory access, snoops, or snoop responses. However, it may lead to memory reads that turned out to be unnecessary, wasting some memory bandwidth.

Since the speculative fetch reduces latency but potentially increases bandwidth costs, it should be selectively used. Typically it should be enabled for latency sensitive requests, and disabled for requests that are less latency sensitive in order to save the bandwidth costs.

The bridge property cc_axi4m_speculative_fetch can be set to yes or no. By default, ACE masters will be set to yes, as they are often latency sensitive. Other masters are set to no, as IO coherent agents are often more latency insensitive.

The speculative fetch value here only sets the default behavior for the interconnect. These values can be overwritten using programmable registers in the CCC(s).

2.9 CD Channel Enable/Disable

ACE protocol specifies that the CD channel (snoop data response) is an optional bus. An ACE master can be built without this channel, requiring any data to be sent back using a WriteBack command on the AW and W channels.

The property *cc_cd_channel_enabled* is used to specify whether the ACE master has a CD channel or not. By default, it is specified as "yes" since CD channel is usually included. Marking this as no will remove the CD channel signals from the bus.

2.10 CD Channel Width

If the CD channel present on an ACE master port, the data width of that channel can be independently controlled from the R and W channel widths, allowing a different snoop data response bus width. This is an interface property for the "crcd" channel, called data_width. The following is an example of how to set this property:

```
ifce_prop cpu0/mprt.crcd.out data_width 128
```

2.11 Selecting a Target IOCB

Each ACE-lite master (including ACE-lite+DVM, and ACE-lite converted) must connect to an IOCB to process WriteUnique, WriteLineUnique, and ReadOnce requests. A single IOCB can be shared by multiple masters. Multiple IOCBs may be used for increased bandwidth, lower latency, or other system tradeoffs. If so, each master needs to have its target IOCB specified.

The bridge property *cc_target_iocb* can be used to specify the target IOCB. By default, this value is marked as unassigned. In a system with a single IOCB, the agents will automatically be mapped to that IOCB. If multiple IOCBs are present, this property must be set for each master.

2.12 Supporting a Fast Tap

Under certain circumstances, Gemini can support a reduced latency mechanism called a fast tap. The fast tap is more direct connection between an ACE master and the coherency controller. By having a direct connection, it is able to skip some of the network overhead of the system. This overhead includes packetization, arbitration and travel through the network, as well as de-packetization. And that cost can exist for both the AR and R packets.

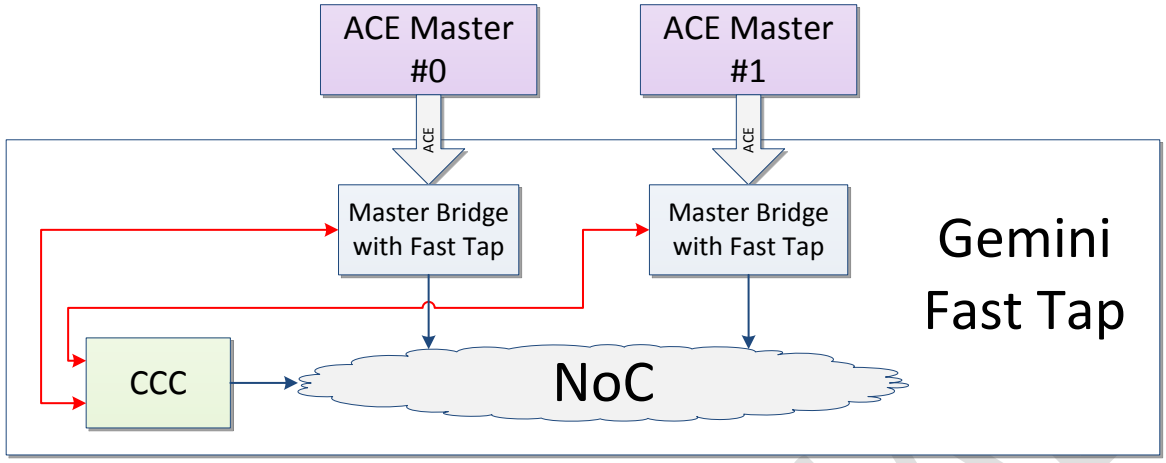


Figure 13: Fast Tap

The Fast Tap creates a dedicated path to the CCC and back, skipping packetization and de-packetization, as well as any arbitration steps. As seen in the diagram above, the fast tap connection creates additional dedicates paths to the CCC. Other traffic can be sent through the bridge to the normal paths through the NoC. The Fast Tap is only used for coherent reads and Cache Maintenance operations that target the CCC.

Fast Tap does require dedicated wires as well as some additional storage, so it does not come for free. However, for many systems, the latency reduction may be worth the additional costs.

Fast Tap has a number of restrictions.

1.) Only ACE masters can use the Fast Tap.
2.) Fast Tap is limited to at most 2 ACE masters.
3.) Fast Tap is limited to interfaces with a 16 byte R channel.
4.) Fast Tap can only exist in a system with a single CCC.

The property used to specify that an ACE master should utilize the fast tap is *cc_fast_tap_enabled*, which defaults to "no".

2.13 Request Splitting

NetSpeed Gemini will split all reads and writes that target CCC, IOCB, or LLC if the requests cross a natural 64-byte boundary if . These components process requests on a 64B cache line granularity, requiring any larger requests to be split into smaller requests that can be processed.

For other requests in the system, request splitting can be controlled by the axi4m_request_split_size bridge property of the master.

2.14 ACE Agent 64-byte Limitation

Due to an issue in the ACE specification, full ACE agents are restricted to coherent accesses within a 64-byte boundary. Coherent requests that cross a 64-byte boundary will be rejected by the NoC. This only applies to full ACE agents. AXI or ACE-lite agents have no such restriction and are instead limited by the 4KB AXI granularity requirement.

2.15 Write-Evict Handling

In Rev E of the ACE protocol specification, the Write-Evict requests is added to the protocol. This is to be used in conjunction with a last-level cache. A configuration without a last-level cache should avoid issuing Write-Evict requests, as they can cause unnecessary requests to the CCC or to memory. In ARM processors, this can usually be prevented by disabling UniqueClean Evictions. In cortex-A15, for instance, this can be found in the L2ACTLR register.

2.16 Bufferable Requests

NetSpeed Gemini does not provide early acknowledgments for write requests. If a request is marked as bufferable, Gemini will not treat it any differently from non-bufferable requests.

2.17 Barrier Handling

NetSpeed Gemini supports barriers in the ACE specification in multiple ways. For most barriers, the bridge will simply wait for prior requests to complete, which occurs when the BRESP or RRESP packet returns. For coherent memory barriers (Inner Shareable or Outer Shareable), barriers sent from ACE-lite or ACE-lite+DVM agents will be forwarded to the NetSpeed IOCB, to allow greater performance. ACE master bridges locally complete barriers sent from ACE masters.

Since barriers are not broadcasted to slave devices, any early acknowledgments from the slaves will allow the barriers to complete.

2.18 Read Data Interleaving

AXI protocol allows read requests to have their data responses interleaved by the slave and through the network. NetSpeed Gemini does not support this. If slaves do interleave data, a data de-interleaver can be added to the slave's bridge in NocStudio.

2.19 Command Mapping

In the ACE protocol, there are many different command variants. This section describes how these different commands are processed by Gemini. The table below shows which destination the command is sent to be processed.

ARSNOOP	ARDOMAIN	ARBAR	Transaction Type	Target
4'b0000	00, 11	x0	ReadNoSnoop	Slave
4'b0000	01, 10	x0	ReadOnce	IOCB or CCC*
4'b0001	01, 10	x0	ReadShared	CCC
4'b0010	01, 10	x0	ReadClean	CCC
4'b0011	01, 10	x0	ReadNotSharedDirty	CCC
4'b0111	01, 10	x0	ReadUnique	CCC
4'b1011	01, 10	x0	CleanUnique	CCC
4'b1100	01, 10	x0	MakeUnique	CCC
4'b1000	00,01,10	x0	CleanShared	CCC
4'b1001	00,01,10	x0	CleanInvalid	CCC
4'b1101	00,01,10	x0	MakeInvalid	CCC
4'b0000	01,10	01	Coherent Memory Bar	IOCB or local bridge**
4'b0000	01,10	11	Coherent Sync Bar	local bridge
4'b0000	00,11	x1	Non-coherent Bar	local bridge
4'b1110	01, 10	x0	DVM Complete	DVM
4'b1111	01,10	x0	DVM Message	DVM

AWSNOOP	AWDOMAIN	AWBAR	Transaction Type	Target
3'b000	00, 11	x0	WriteNoSnoop	Slave
3'b000	01, 10	x0	WriteUnique	IOCB or CCC*
3'b001	01, 10	x0	WriteLineUnique	IOCB or CCC*
3'b010	01,10	x0	WriteClean	CCC

3'b010	00	x0	Non-Share WriteClean	Slave
3'b011	01,10	x0	WriteBack	CCC
3'b011	00	x0	Non-Share WriteBack	Slave
3'b100	01,10	x0	Evict	CCC
3'b101	01,10	x0	WriteEvict	CCC
3'b101	00	x0	Non-Share WriteEvict	Slave
3'b000	01,10	01	Coherent Memory Bar	IOCB or local bridge**
3'b000	01,10	11	Coherent Sync Bar	local bridge
3'b000	00,11	x1	Non-coherent Bar	local bridge

* ACE masters send these requests to CCC. ACE-lite, ACE-lite+DVM, ACE-lite converted bridges send the request to IOCB.

**Barriers are sent to IOCB from ACE-lite or ACE-lite+DVM agents. For ACE agents, they are locally completed.

In the tables above, there are 5 possible destinations for a request to be sent to. Non-coherent requests can target the destination slave, without going through other coherency components. Some other requests target the IOCB, which can make request to the CCC if necessary. Some request will go straight to CCC for processing. DVM requests will go to the DVM module for processing. And finally, some of the barrier instructions will be satisfied locally at the bridge.

Note that a configuration may have multiple slaves, multiple CCCs, etc. Additional address map information is used to identify which specific agent a request will be sent to. Additionally, if an address is not mapped to a device, a decode error will be generated in the local bridge.

2.20 Coherency Connect/Disconnect

ACE masters and ACE-lite+DVM masters can receive snoops from CCC or DVM. To shut down one of these ports, the coherency IP must be alerted that the agent wants to be disconnected from there coherency protocol so they can stop sending snoop requests to that agent. To provide this functionality, these agents have a connect/disconnect mechanism for coherency.

These agents can be configured in three ways in NocStudio. They can be configured to have a control pin that selects whether the agent is connected or disconnected from the coherency, or they can be controlled through registers.

If pins are configured, the NoC will create two new signals to the port called SYSCOREQ and SYSCOACK. The SYSCOREQ is driven by the master to connect to the coherency protocol. The SYSCOACK provides a status response. These signals are part of a 4-state handshake. At reset, the agent starts disconnected, and SYSCOREQ and SYSCOACK are deasserted. While disconnected, the agent cannot issue coherent requests or DVM requests. When it wants to connect, it will assert the SYSCOREQ signal. It must then wait for the SYSCOACK signal to indicate that the connection has been made. The SYSCOREQ should not be deasserted until SYSCOACK is raised.

Once connected, the agent can receive snoops and may issue coherent requests including DVM requests.

If the agent decides to disconnect, it must go through a disconnect process. The first thing is must do is stop making new coherent requests. The ACE master must then flush its caches of any modified lines. This is because once it is disconnected, no other agents will be able to snoop it and any data it has will be lost. Once the cache is flushed, and all CopyBacks have received their responses, the agent can signal that it wants to disconnect by deasserting the SYSCOREQ. Even after SYSCOREQ is deasserted, the agent must still accept snoops and respond to them appropriately. This is because there may already be snoops outstanding in the network, and they must be successfully completed.

Eventually, the SYSCOACK will indicate that the coherent agent has successfully disconnected and that no new snoops are outstanding. At this point, the agent can power down or reset without breaking the rest of the coherent system.

The coherent agent can instead be set up to have register control of the coherency connect/disconnect mechanism. Instead of having pins to control this, the master bridge will have internal registers that generate or syn the SYSCOREQ and SYSCOACK signals. To disconnect a connected agent, a register write to the SYSCOREQ register of that bridge should be made. When the SYSCOACK register match the SYSCOREQ value, the change has taken place. Like the pins, SYSCOREQ shouldn't change until after the SYSCOACK is matching it.

If this register control mode is select, the user can configure an agent to come out of reset either connected or disconnected. If connected, the SYSCOREQ signal will start of asserted, and the connect sequence will start immediately. If disconnected, the agent will start of disconnect and only become connected when the SYSCOREQ register is written.

The coherency connect/disconnect mechanism can be used for powering down an agent. It can also be used for error handling, to allow an agent do be reset without disrupting the coherent system.

2.21 ECC ALGORITHM

If the directory is configured to have ECC (Error-Correcting Code), the IP will implement a customized ECC algorithm. Additional bits will be added to the directory RAM array width to hold ECC information. The directory control logic will handle generating ECC values and checking the directory read results to confirm that there is no error.

The ECC algorithm uses a hamming code with an additional parity bit, sometimes referred to as SECDEC (single error correction, double error detection). The algorithm adds the ECC checkbits to the protected data block, so all bits are protected with single-bit correction, and double-bit dectection.

The hardware supports a register mechanism to directly access the directory RAM, including the ECC checkbits. It supports multiple variants, including a method to take an existing directory entry and flip one or more bits before writing it back into the array. This can be used to test ECC logic within the system.

The ECC detection and correction can also be disabled via register access.

2.22 INVALIDATION ENGINE

The directory supports an invalidation engine that can invalidate the full content of the directory. The engine runs through every set of the array and overwrites all bits in that set to zero, including ECC bits if present. No stale content will be left in the directory after the invalidation engine has completed.

The invalidation engine is triggered on the deassertion of reset. While executing the invalidation, any coherent requests that need to access the directory will be stalled until the sequence has completed.

The invalidation engine is built to issue writes to both directory arrays in parallel, to speed up the time to completion. A directory set contains multiple entries, all of which are invalidated in a single write.

The invalidation engine can also be triggered by a register access. This must be carefully controlled. The invalidation will cause a loss of coherent state information, so any agents holding a copy of a line will no longer be tracked. To avoid this, all caches should be invalidated if the directory is invalidated.

While the invalidation engine will overwrite the directory content, any outstanding requests may modify the directory state after the invalidation engine has completed. To avoid this effect, all requests to the CCC must be completed and the CRT entries should be checked to confirm there is no outstanding requests.

3 Cache Hierarchy Configurability

Gemini supports a flexible cache hierarchy, allowing the architect significant control of the design. A cache hierarchy is the organization of the caches, coherency components, and memories. It is defined by the connectivity of these components, as well as which addresses are handled by each component.

A simple example of a cache hierarchy is shown below:

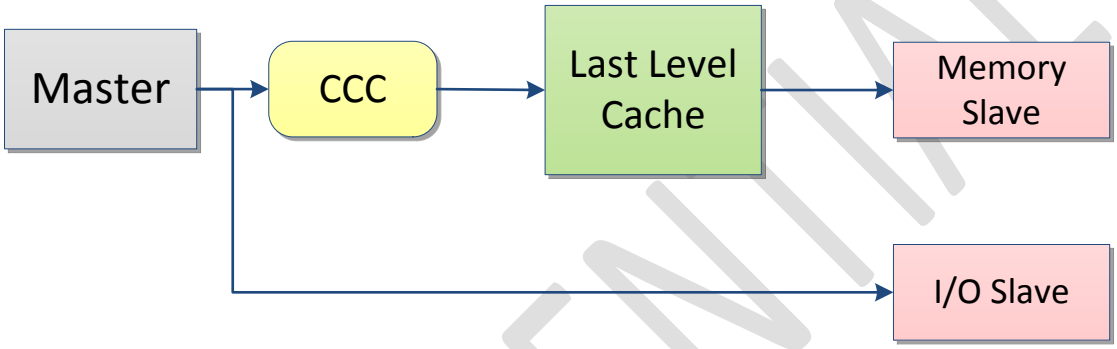


Figure 14: Coherent and Non-coherent Devices

In the diagram above, a master is connected to two slave devices. One is memory slave that is intended to support coherency. The other is an I/O slave that accepts only non-coherent requests. As seen in the diagram, the coherent memory is connected to the master through a coherency controller and a last level cache. For the I/O slave, non-coherent requests can be sent directly from the master to the slave.

3.1 Asymmetric Hierarchies

The cache hierarchy can be asymmetric within the same system.

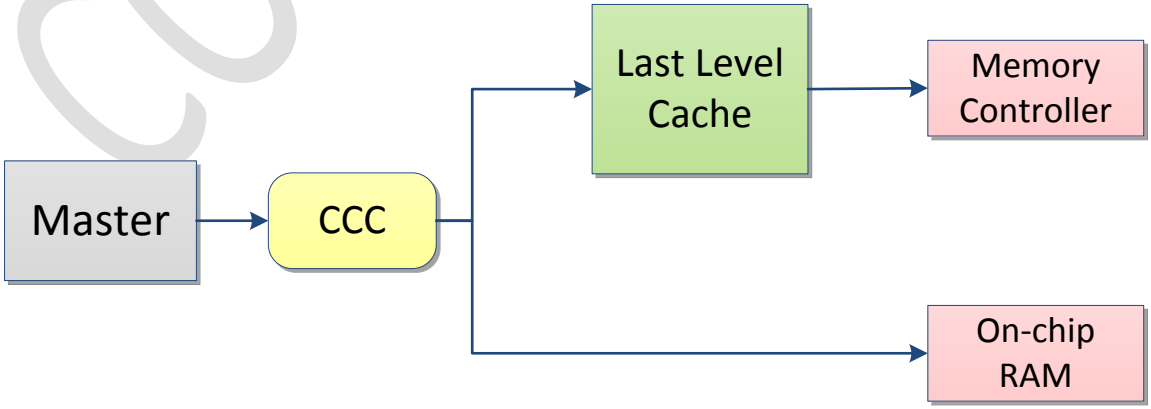


Figure 15: Asymmetric Cache Hierarchy

In the diagram above, both slaves are enabled for coherency, but only one of them utilizes the cache. Since one of the slaves is an on-chip RAM, it provides no benefit for storing data in the LLC since the on-chip RAM can be accessed just as quickly. It also allows the LLC to improve performance more by only caching lines with a long memory latency.

3.2 Different Ranges for the same Slave

This asymmetry can also apply to the same slave, but allowing different memory ranges to be treated differently.

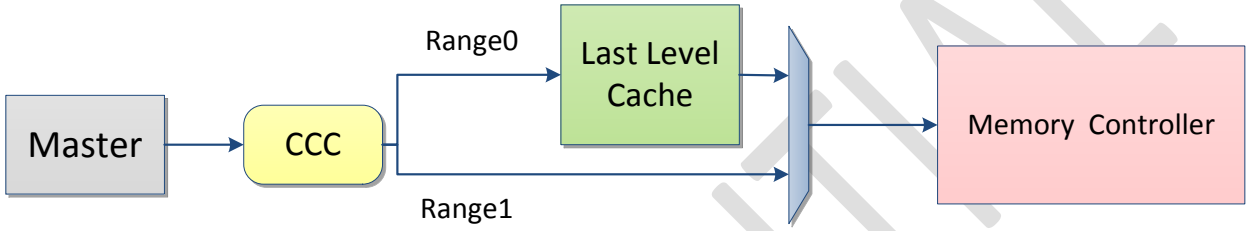


Figure 16: Different ranges of to the same memory controller

In the diagram above, different ranges of the same memory controller can be handled differently, with one range going through a cache, while the other range goes directly from the CCC to the memory controller.

3.3 Adding Slices for CCC or LLC

For systems with larger bandwidth requirements, it may be necessary to utilize multiple CCCs or LLCs to increase bandwidth. One common method for supporting this is to take an address range and slice it into equal parts, with each CCC or LLC responsible for one of the parts. If requests are well distributed to the different slices, bandwidth will increase proportional to the number of components. Having 4 instances of a cache can get 4x the bandwidth of a single instance.

The slicing function uses specified address bits to assign responsibility to each component. Slicing can be done using a power-of-2 number of slices. This allows a simple decode of address bits.

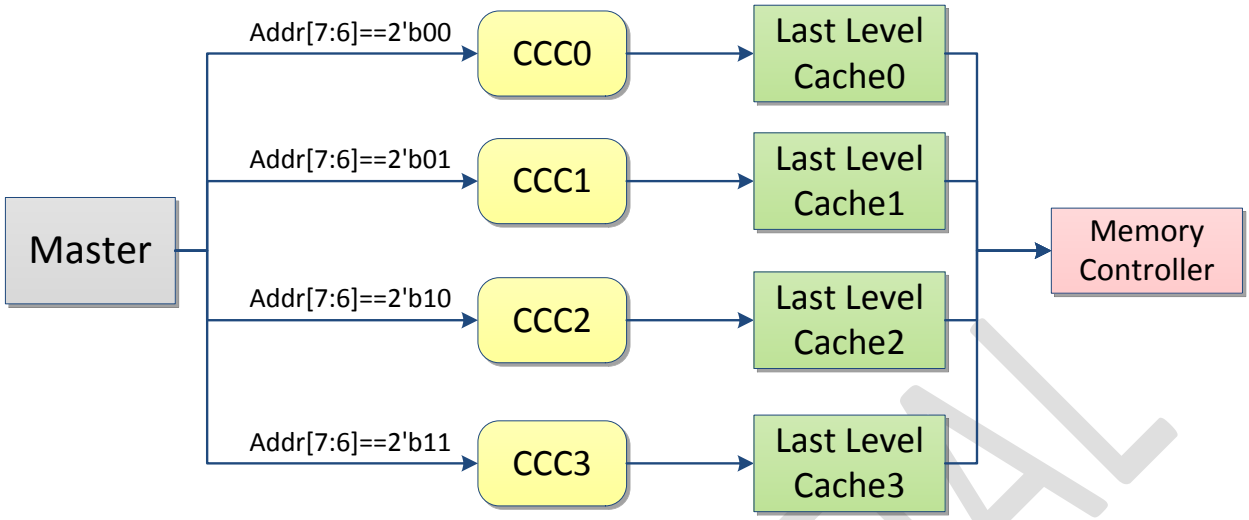


Figure 17: Uniform sliced address space

In the figure above, the memory controller address space is split into 4 regions, dividing by address bits 7 and 6. Each address region is handled by a separate CCC and a separate LLC. Each CCC slice handles ¼ of the memory addresses, as does each LLC slice. Since both the CCCs and the LLCs are sliced with the same bits, connectivity is limited. Each CCC slice only has a connection to a single LLC slice.

3.4 Different Slicing between CCC and LLC

In the example above, slicing was done in a uniform way between the CCCs and LLCs. This may be an useful design method, but Gemini supports a larger variety of cache hierarchies. The number of CCCs and the number of LLC can vary.

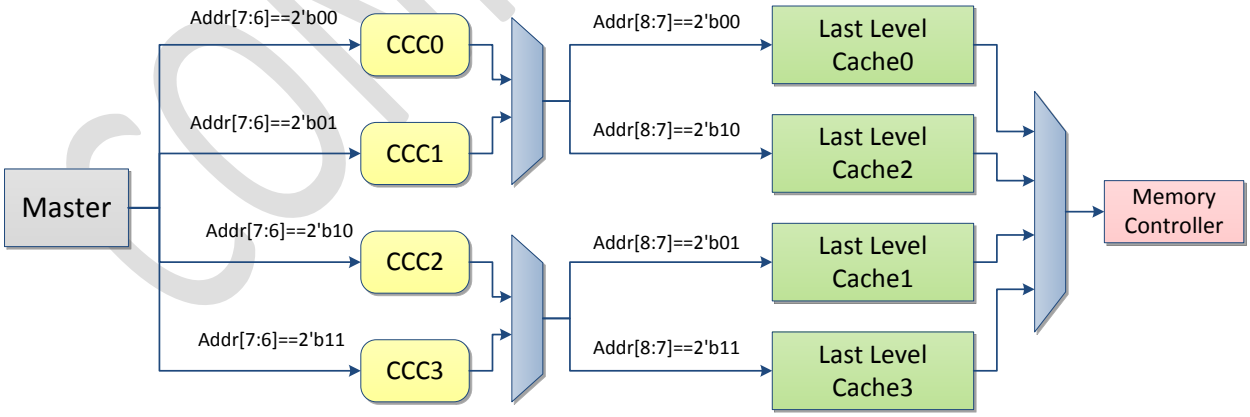


Figure 18: Different Slicing between CCC and LLC

The figure above shows 4 CCCs and 4LLCs, but with a different connectivity. CCC0 and CCC1 can each talk to LLC0 and LLC1, while CCC2 and CCC3 talk to LLC2 and LLC3. This configuration can happen when slice bits are chosen in a different manner.

3.5 Different Number of CCCs and LLCs

Since slicing can be different between CCCs and LLC, Gemini can also support a different number of CCCs and LLCs in the same cache hierarchy. This can be useful if the bandwidth requirements are different or if physical location requires agents to be physically distributed.

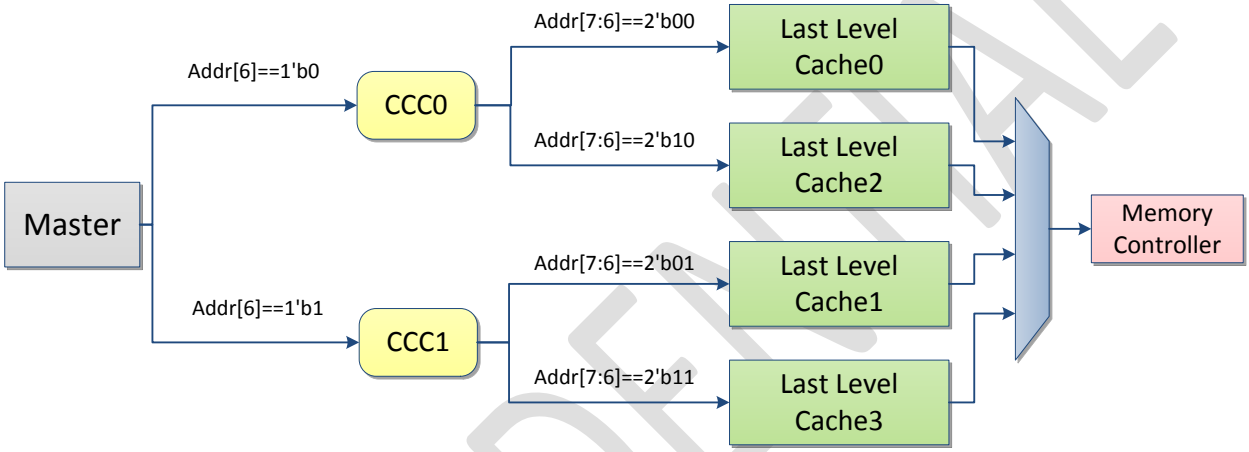
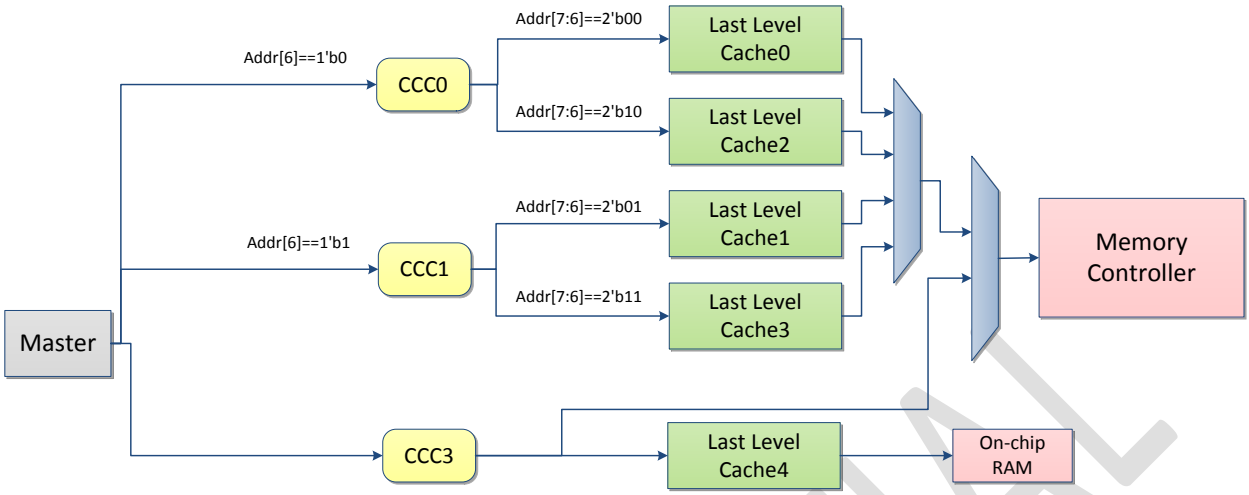


Figure 19: Different number of CCCs and LLCs

In the figure above, there are only 2 CCCs while there are 4 LLCs in this cache hierarchy. Additionally, each CCC only talks to two LLCs.

3.6 Combining These Design Choices

Complex cache hierarchies can be built with these various controls, as seen below.



3.7 NON-COHERENT ACCESS POINT

Gemini's cache hierarchy support allows for the LLC or Cache to be added as either a Coherent Cache or as a Memory Cache.

A Coherent Cache is a cache that is only accessible by coherent requests. The CCC sends requests to the LLCs, but non-coherent traffic skips the CCC and the LLC and talks directly to memory.

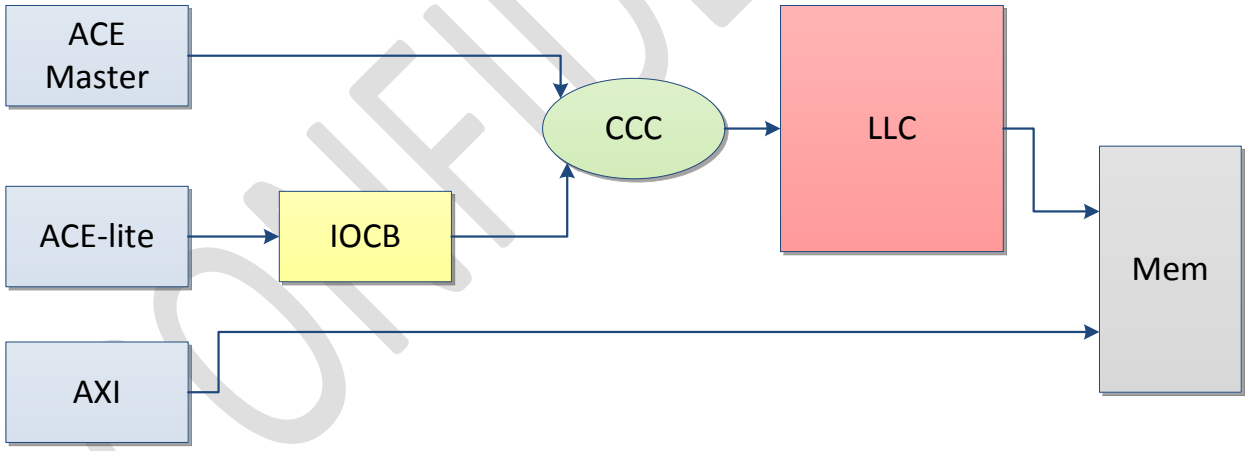


Figure 20: Example Coherency Cache hierarchy

In the diagram above, coherent traffic is sent to IOCB and CCC, which can send coherent requests to the LLC. Non-coherent requests, as embodied by the AXI agent's traffic, skip past the coherent system and the LLC, and get sent directly to the memory port.

An alternative cache hierarchy design uses a Memory Cache. A memory cache is a cache that is accessible by all traffic, including coherent and non-coherent. The following diagram shows an example memory cache system.

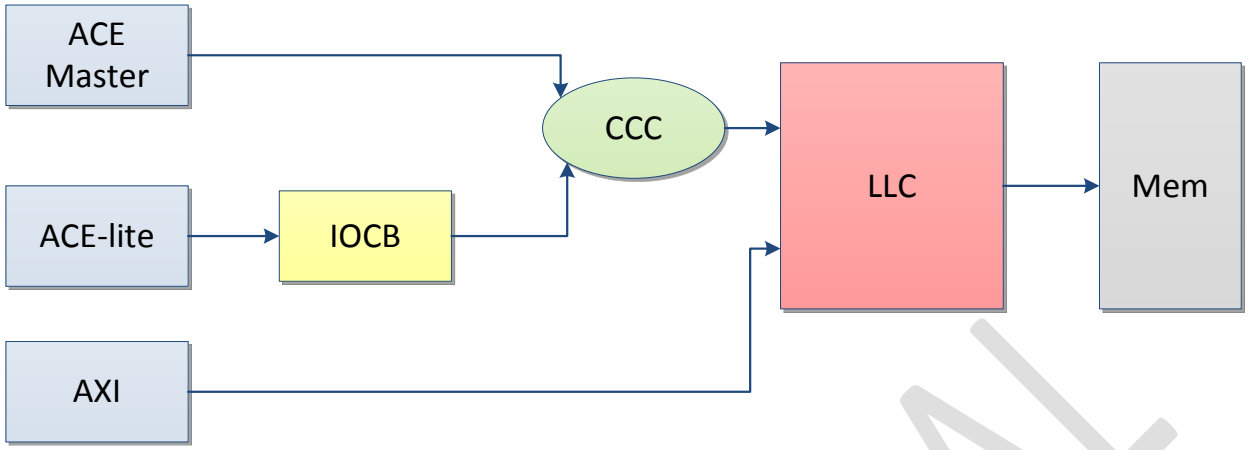


Figure 21: Example Memory Cache hierarchy

There are several tradeoffs between a coherent cache and a memory cache, and choosing the right option for a system can create a significant differentiation.

The decision of a memory cache vs. a coherent cache is decided during construction of the Gemini system. The add_llc_group or add_cache_group commands have a memory cache attribute that can be set. If set, the cache will act as a memory cache, and non-coherent traffic to that range will be sent to the LLC. If the attribute is not set, the default will be coherent cache behavior, where only the CCC talks to the LLC.

4 Functional Description: System Interconnect

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

Figure 22 illustrates the basic building blocks of the NetSpeed System Interconnect architecture.

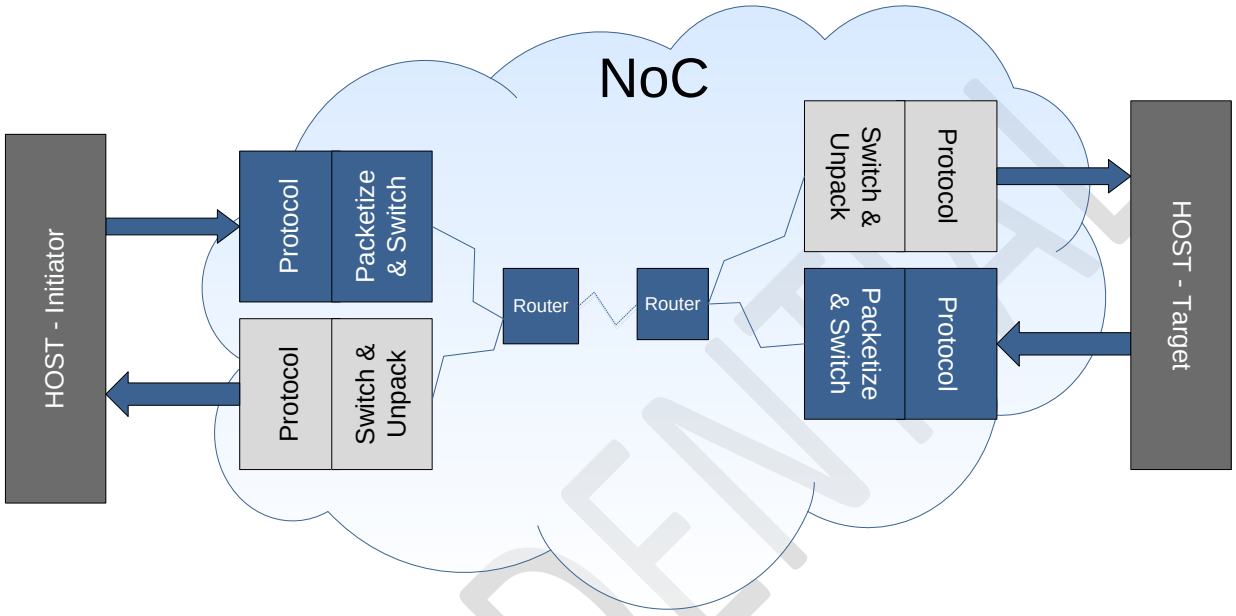


Figure 22. Bridge and Router Functions in the NoC

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

A 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.

4.1 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 23 illustrates a full 2D-mesh interconnect, with a router at each mesh cross point.

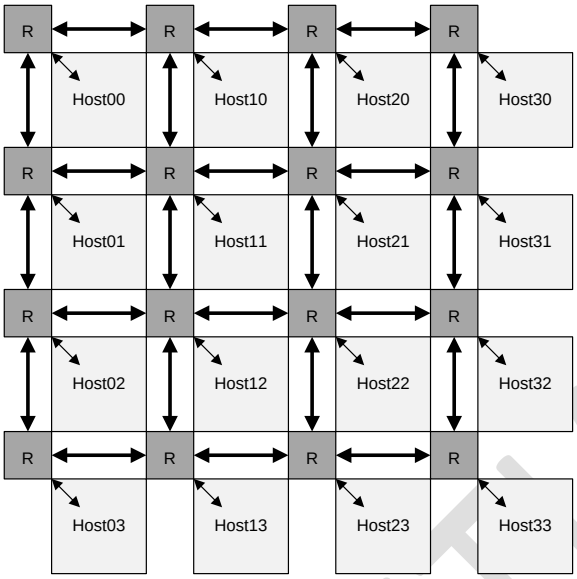


Figure 23. A 4x4 Homogeneous Grid or Mesh Interconnect

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 23. 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 23. 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 24 **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.

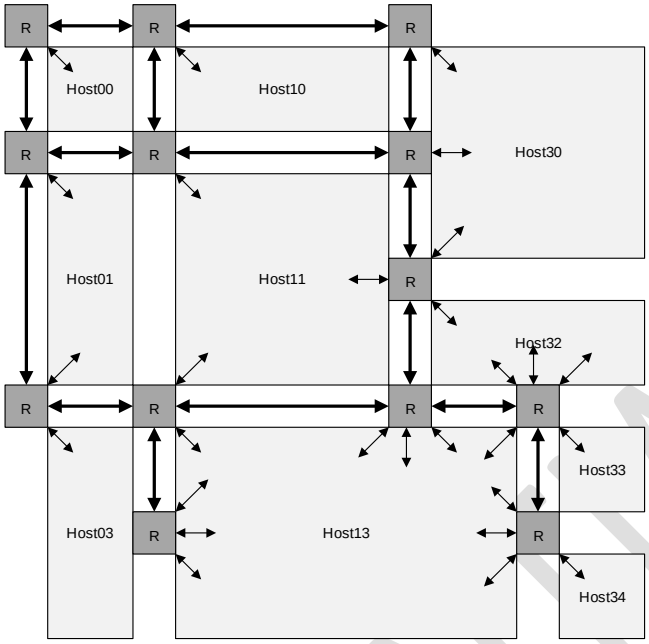


Figure 24 A 4x4 heterogeneous grid or mesh interconnect

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 24 **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.

4.1.1 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.

4.1.2 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.

4.2 ROUTERS

Figure 25 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.

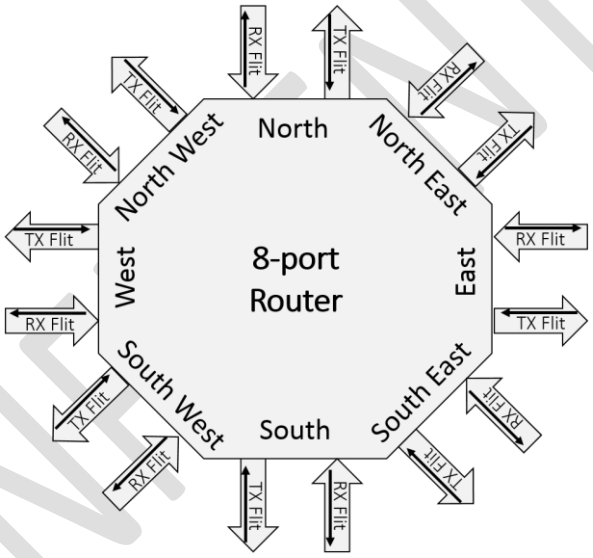


Figure 25. Schematic Symbol of a NocStudio RTL-Library Router Component

4.3 BRIDGING FROM HOST TO NOC

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

Bridge parameters and properties are assigned by NocStudio based on the high-level specification of traffic and hosts. Some bridge properties are made visible to the user. Those

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

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

4.3.1 AXI4 Master Bridge

Figure 26 shows a block diagram of the NetSpeed AXI4 master bridge. The master bridge contains a layer for AXI4 protocol processing, and a switch layer that communicates with routers in up to sixteen NoC layers.

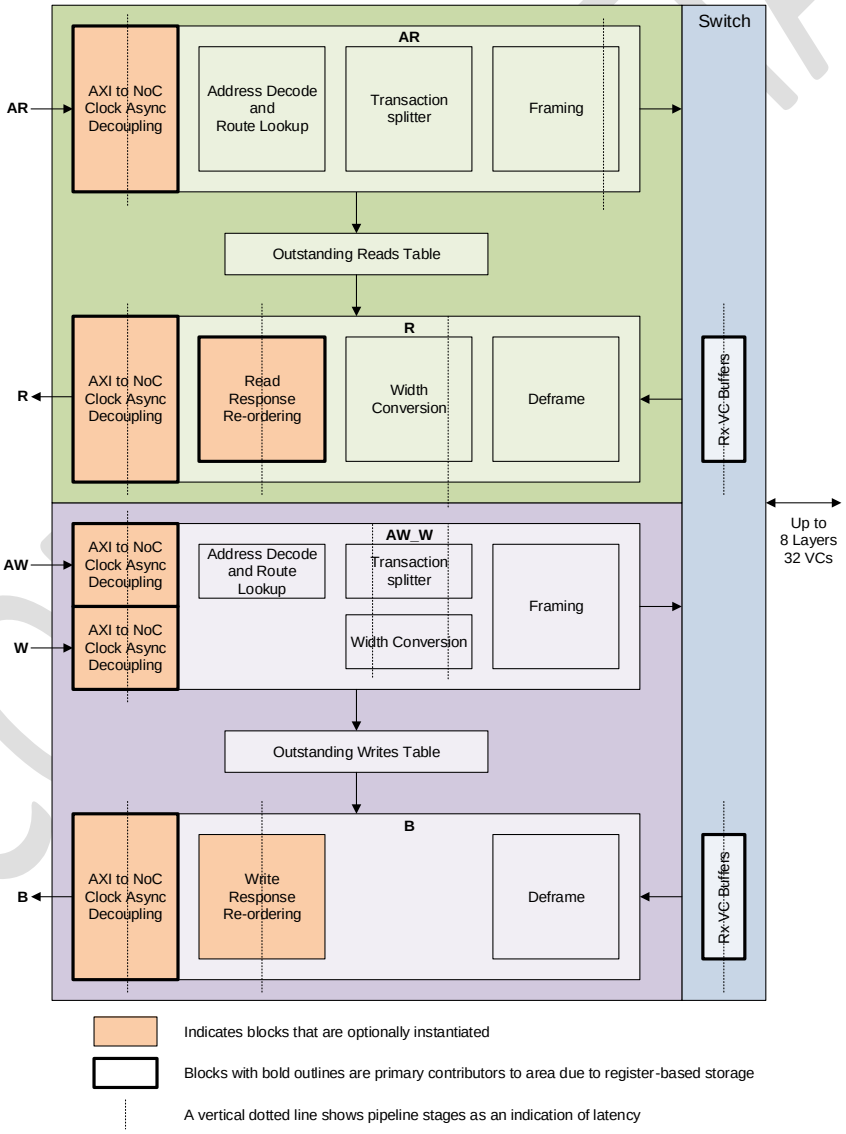


Figure 26. AXI4 Master Bridge

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

4.3.1.1 AXI4 Channels and NoC

- The master bridge provides an AXI4 slave interface to a host AXI4 master port. Five standard AXI4 channel—read command (AR), write command (AW), write data (W), read response (R) and write response (W)—are supported.
- Up to sixteen physical NoC layers with four virtual channels each are available to transport the AXI4 channels. These are allocated by NocStudio based on QoS, bandwidth, deadlock, and other requirements.
- The AR, AW, and W channels are packetized and transmitted on NoC layer virtual channels and packets are received from NoC layer virtual channels for R and B channels.
- A write command from an AW channel and the corresponding write-data burst from a W channel are combined into a single packet. AW could be sent as the side band of the write packet for higher write bandwidth, or could be sent as a header in-band with the write packet for lower area.

4.3.1.2 Decoding and Routing

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

4.3.1.3 *Flow control*

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

4.3.1.4 *AXI4 Specific Features*

- The master bridge supports a configurable number of outstanding transactions on the read and write channels.
- Logic for ordering R and B responses processed out-of-order by the network for higher performance can be optionally instantiated on the master bridges. Responses to the master are returned in the order that the requests were received.
- INCR transactions are split at specific address boundaries based on certain rules described in section 1.9
- FIXED transactions are split into multiple single beat INCRs
- WRAP transactions different from 64-byte, 32-byte, or 16B cache-line size are considered a fatal error, and handling by the NoC is not guaranteed. An interrupt is raised to indicate this fatal error.
- Split transaction responses are transparently coalesced to match the original master command. Optionally, read response segments of a split transaction can be interleaved with transactions with different AID
- R and W data channels are processed based on the commands supporting width conversion when communicating with AXI slaves having different AXI data widths.

4.3.1.5 *Errors and Stalls*

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

4.3.2 AXI4 Slave Bridge

Figure 27 shows a block diagram of the NetSpeed AXI4 slave bridge. The slave bridge contains a layer for AXI4 protocol processing, and a switch layer that communicates with routers in up to sixteen NoC layers.

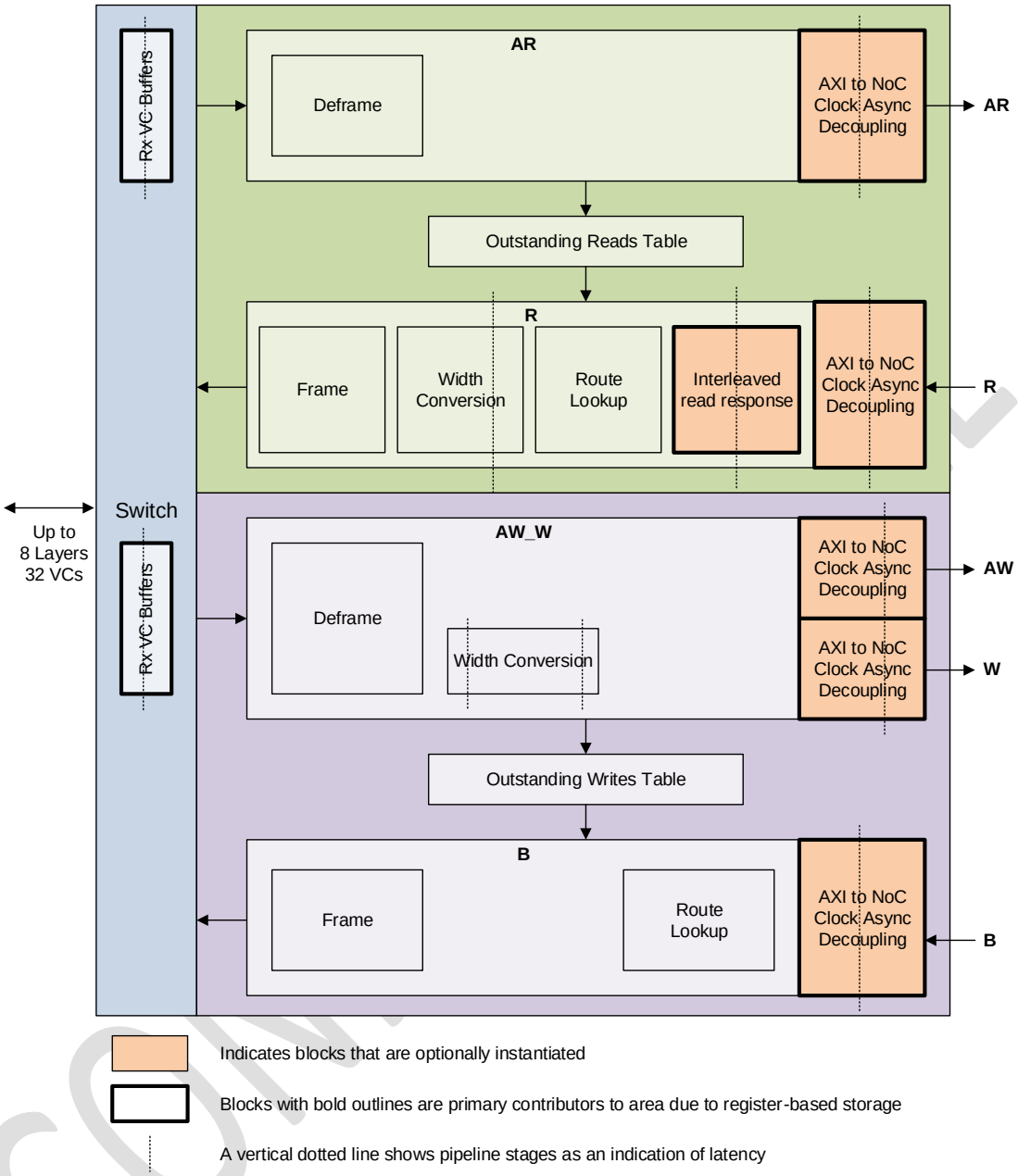


Figure 27. AXI4 Slave Bridge

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

4.3.2.1 AXI4 Channels and NoC

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

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

4.3.2.2 Decoding, Routing, and Flow Control

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

4.3.2.3 AXI4 Specific Features

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

4.3.3 AXI3 Master and Slave Bridge

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

4.3.4 AXI4-Lite Slave Bridge

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

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

Figure 28 shows a block diagram of the AXI to AXI4-Lite Bridge. On the ingress side, the bridge is compliant with the AXI4 interface specification. On the egress side it is compliant with the AXI4-Lite interface specification. To simplify address definition and decode of downstream bridges, this bridge can be configured to send the AxREGION bits. Note that AxREGION bits are not part of AXI4-Lite interface. The bridge can also be configured to pass ARSIZE to allow narrow read access on the slave. Narrow write accesses can be done using WSTRB and does not require AWSIZE.

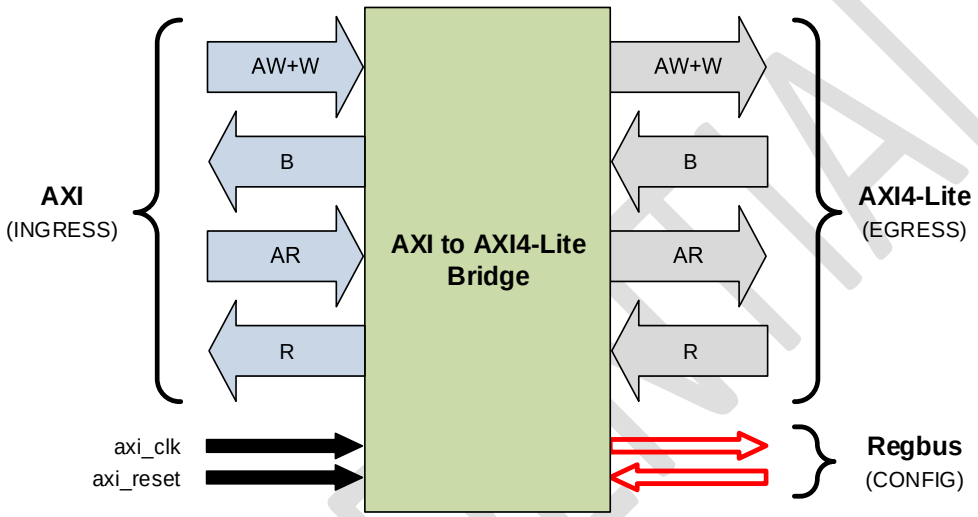


Figure 28. AXI to AXI4-Lite Bridge

4.3.5 APB Bridge

Figure 29 shows a block diagram of the AXI to APB bridge. The AXI to APB bridge provides the capability to attach legacy APB slaves to AXI4 masters. This layer is used with the AXI4 slave bridge to form a NoC to APB bridge. The bridge translates incoming AXI4-Lite transactions into APB transactions. It has the following features:

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

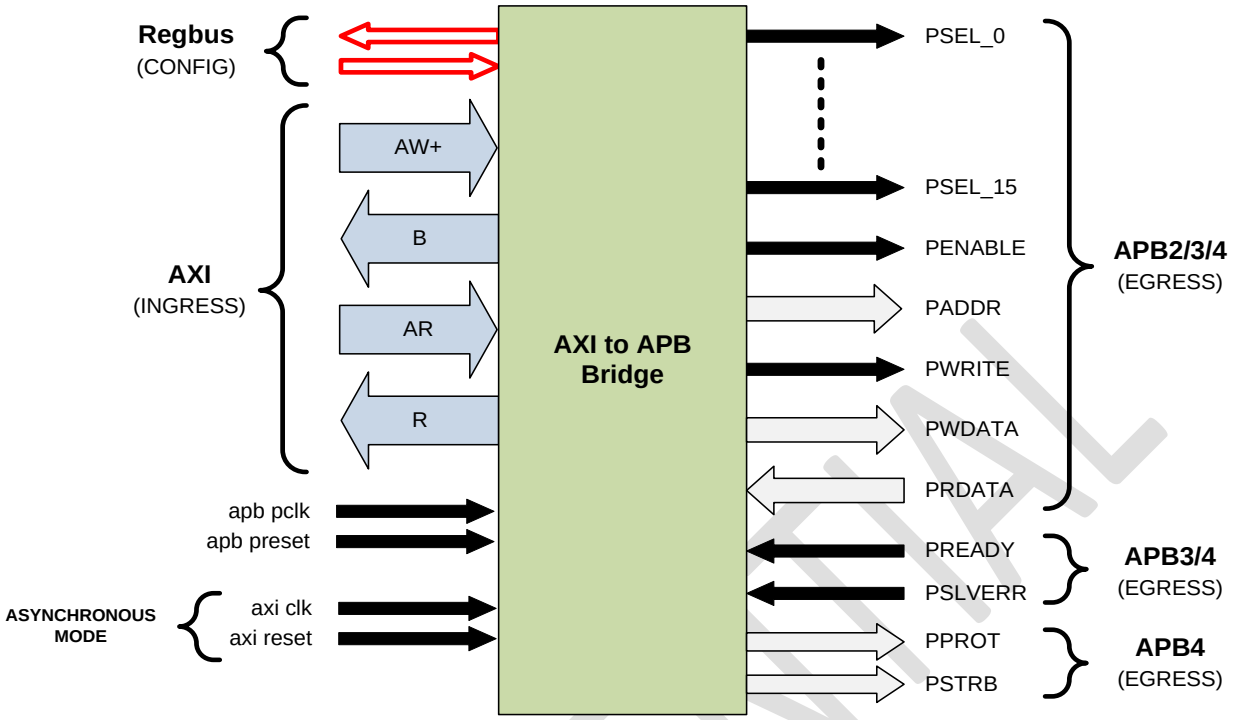


Figure 29. AXI to APB Bridge

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

4.3.6 AHB-Lite Master Bridge

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

Figure 30 shows a block diagram of the AHB-Lite to AXI master bridge. The following summarizes the features:

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

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

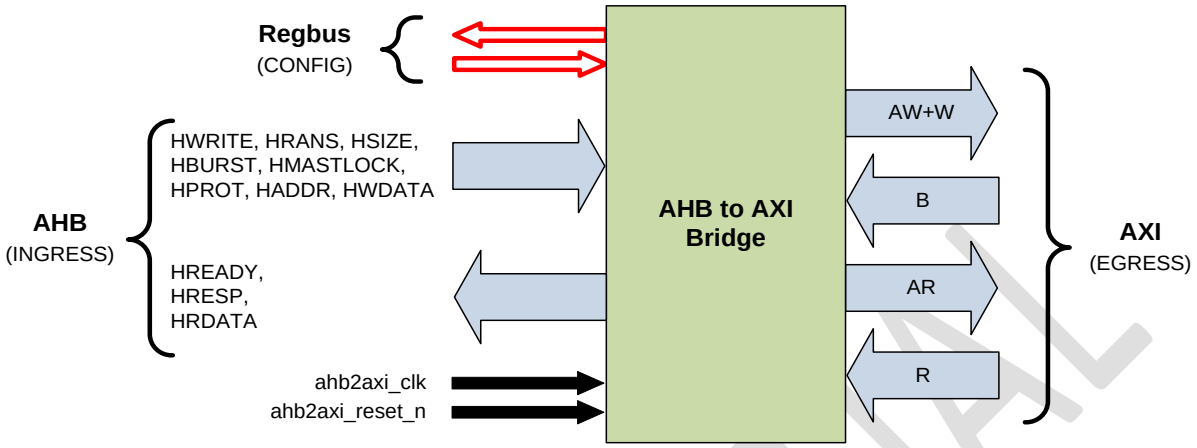


Figure 30. AHB-Lite to AXI Master Bridge

4.3.7 AHB-Lite Slave Bridge

Figure 31 shows a block diagram of the AXI to AHB-Lite bridge. The following summarizes the features:

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

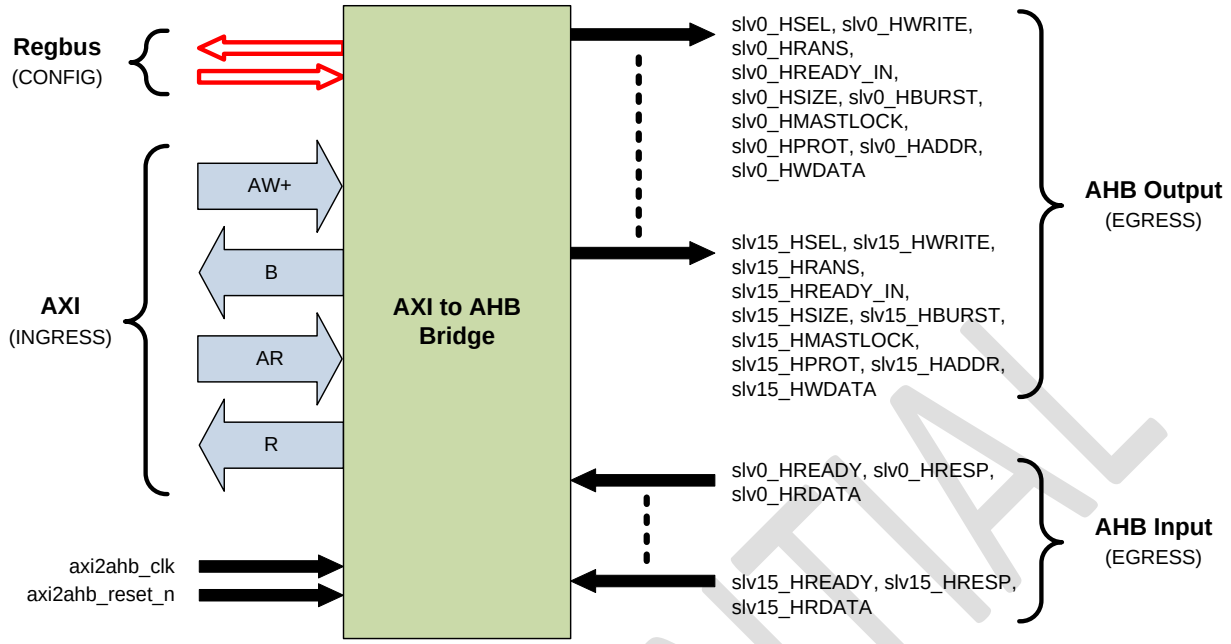


Figure 31. AXI to AHB-Lite Bridge

4.4 SHARED INTERFACE BRIDGE

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

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

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

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

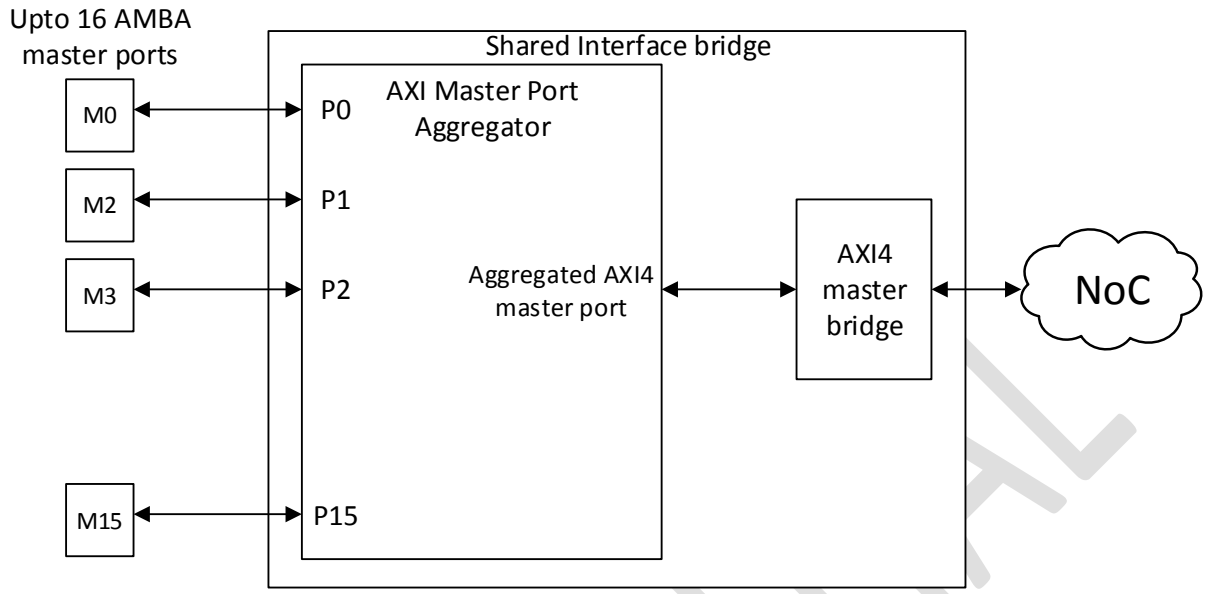


Figure 32 Shared Interface Bridge

4.5 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 33.

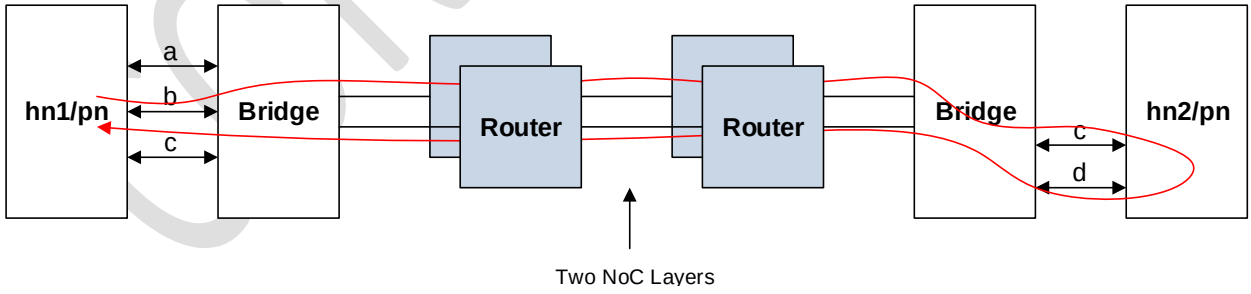


Figure 33. Information Transport in the NoC

4.5.1 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).

4.5.2 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.

4.5.3 NoC Channel Width and Heterogeneity

Flits are carried in NoC router channels, which are restricted to certain widths. Each flit carries overhead, and the number of flit payload bits varies with the flit size. Unless a flit is an EOP flit, the number of payload bits is restricted to a power-of-two multiple of **cell_size**. That value is fixed in NocStudio for the entire NoC project (the NoC project **cell_size** can be modified with the **mesh_prop** command). EOP flits can contain any integer multiple of the **cell_size** payload bits. The value of **cell_size** must be selected based on the host-port signal properties to maximize the packing efficiency of the bridges. Figure 34 illustrates a packet that is organized as two large flits or four small flits.

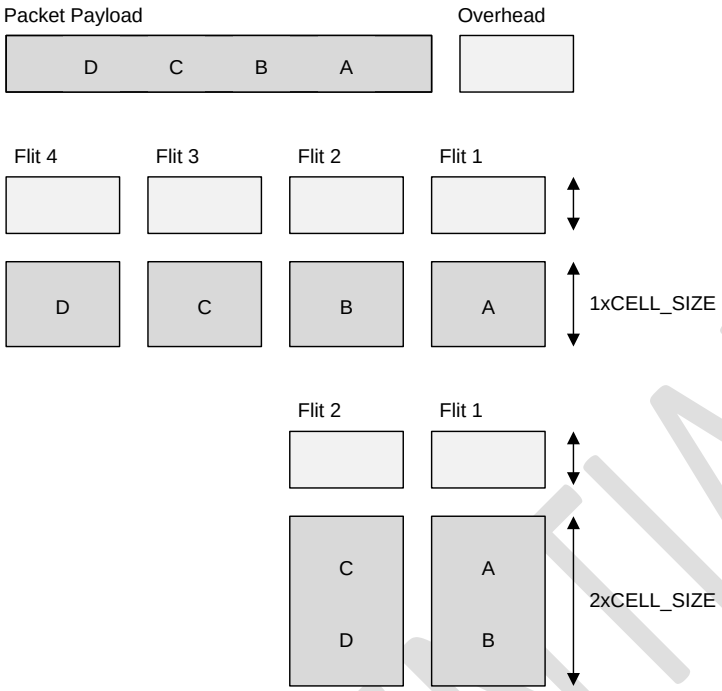


Figure 34. NoC Packet Organized in Two Different Flit Sizes

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

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

Even if the NoC channels are sized at power-of-two multiples of **cell_size**, non-power-of-two flits can be combined if an EOP flits is in the set of flits being combined. The resulting EOP flit can

contain a non-power-of-two multiple of flits. When an EOP flit moves from a wider channel to narrower channel it gets divided into the correct number of smaller flits. Figure 35 illustrates merging and dividing flits under various channel-width scenarios.

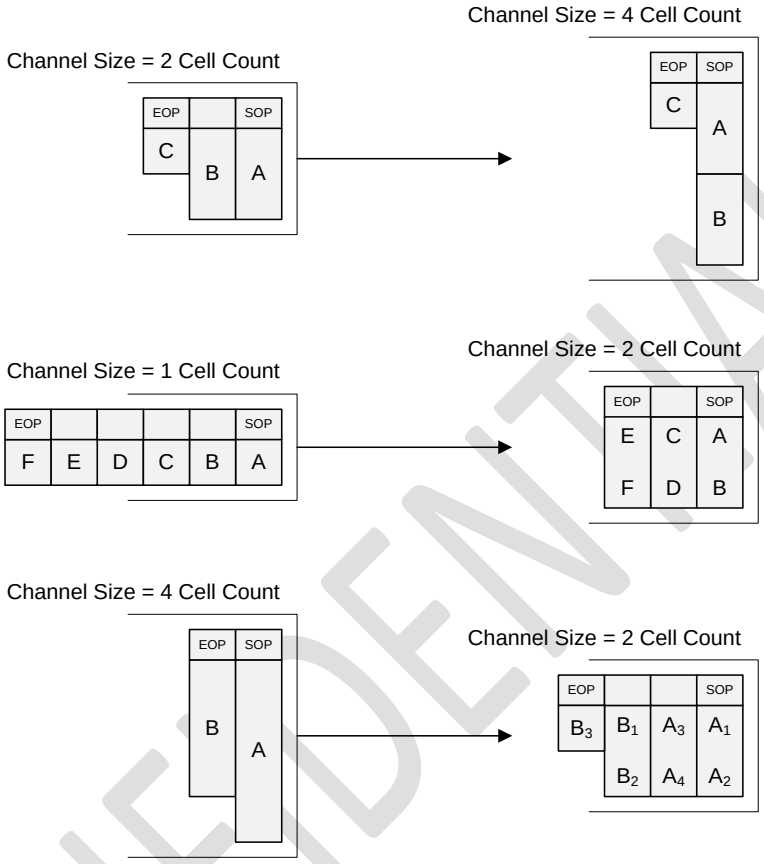


Figure 35. Merging and Dividing Flits with Various Channel Widths

Figure 36 shows how various router paths might have to merge and divide different flit sizes.

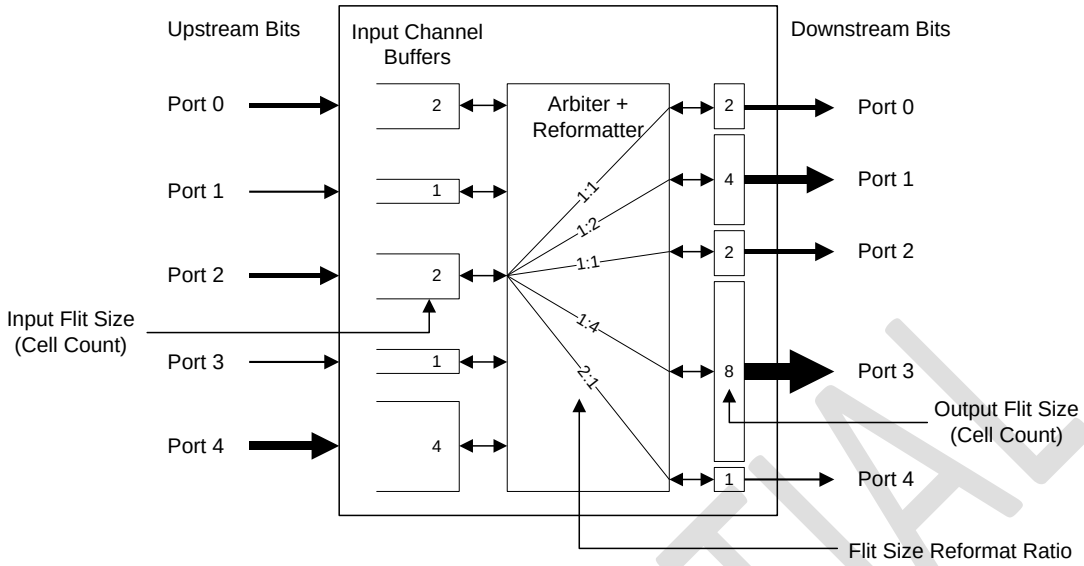


Figure 36. Combinations of Flit Merging and Division at Router Ports

4.5.4 Determining NoC Channel Widths

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

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

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

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

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

4.6 CLOCKING

4.6.1 Clock Domains and Clock Crossing

Figure 37. Multiple Clock Domains and Clock Crossings 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 Figure 37, the hosts in dark blue and the bottom left portion of the chip have been assigned the *other* clock domain. In the bottom right is an *apbbr* clock domain at 200 Mhz.

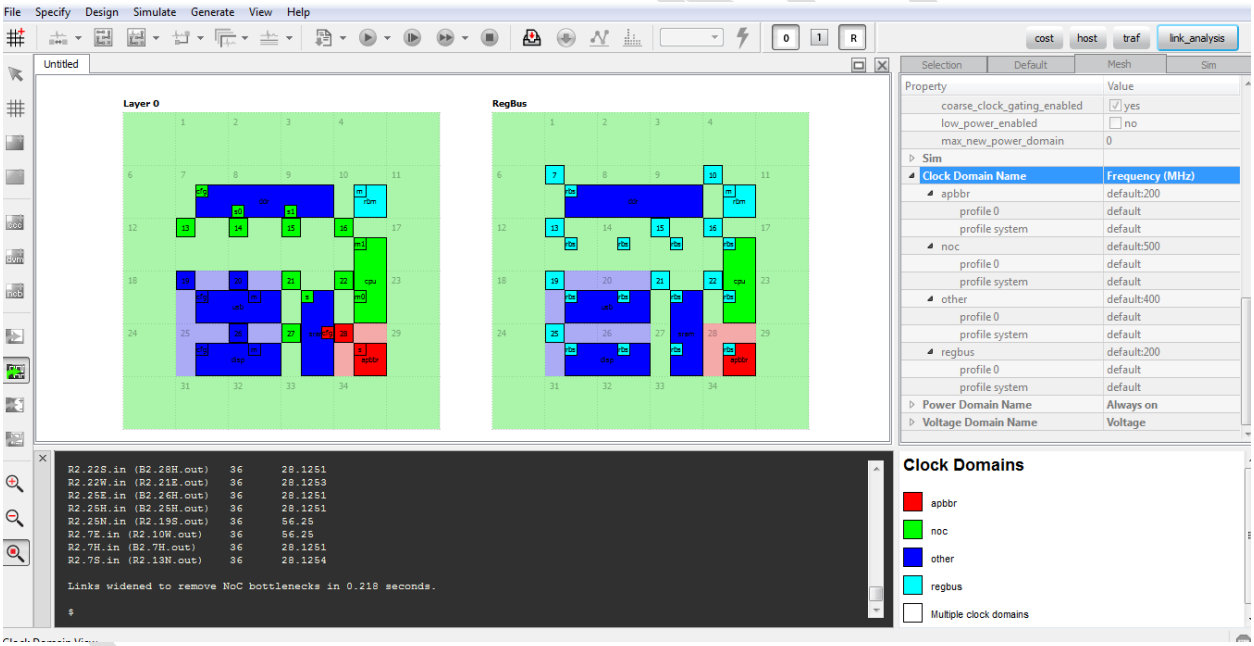


Figure 37. Multiple Clock Domains and Clock Crossings

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 37, 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 alignment 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 relationship as well as a phase relationship. To achieve phase alignment, it is expected that the source of the ratio clocks will be the same.

The N:1 and 1:N clock crossers can actually support a range of ratios. N can be 1, 2, 3 or 4. This allows the faster clock to run at either the same frequency, or up to 4x faster. To determine the actual ratio, a training sequence occurs after reset which determines the exact ratio of the clocks.

If the clock ratio needs to change after reset, a retraining sequence can be initiated through a register access to the bridge in question. The retraining sequence must be carefully controlled as the clock crossing will be unreliable until the training has been completed. To retrain, follow these steps:

- Prevent requests from transmitting across the interface. If software cannot prevent agents from sending accesses to that interface, the NoC address maps can be programmed to disable access from a master, or to a slave.
- Once all requests across that interface has stopped, the clock can be changed.
- The training sequence should be triggered by performing a write to the training register in the bridge module for that interface.
- Poll on the training register status to determine when it is complete.
- Re-enable traffic across the interface.

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.

4.6.2 In-Link Domain crossing

Any router-router or router-bridge link of the NoC can be configured to enable asynchronous clock domain crossing. This structure partitions the NoC link into two independent domains hierarchically. 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 paritions, which can be included into different module groups according to domains.

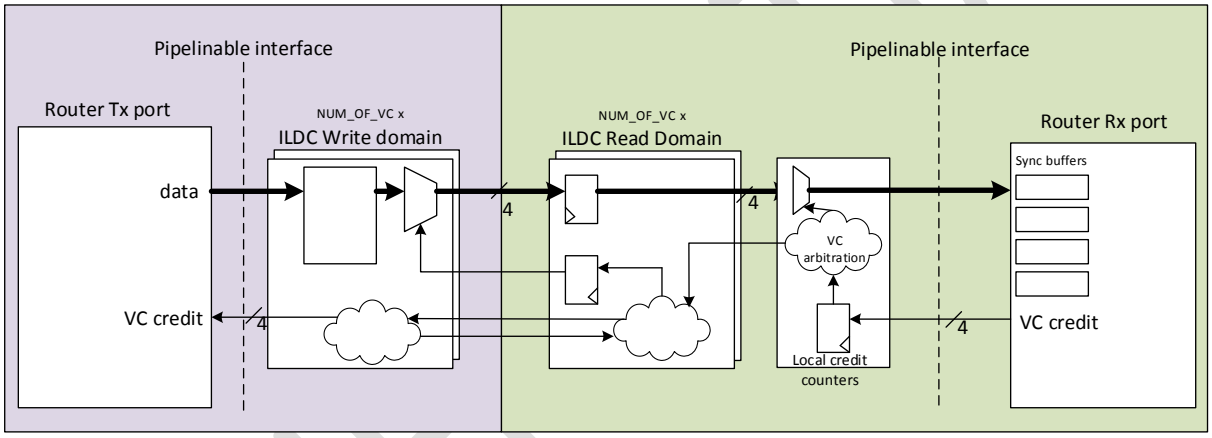


Figure 38 In-Link Domain crossing structure

4.6.3 Clock Gating

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

1. **System-level clock gating:** An external system controller gates the entire NoC clock. This is controlled by system software of firmware.
2. **Coarse-grained clock gating:** This is done at the NoC-element level based on its activity.
3. **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. Fine-grain clock gating is usually done by the synthesis tool, exploiting logic-level opportunities to insert local clock gating at the flop level. System-level and coarse-grain clock gating are enabled using the NocStudio property:

```
mesh_prop coarse_clock_gating_enabled yes
```

By default this property is set to **no**. When set to **yes**, the NoC RTL will have primary outputs to enable system-level and coarse-grain clock gating.

Figure 39. Clock Gating as Implemented in NetSpeed NoC shows NetSpeed NoC clock gating.

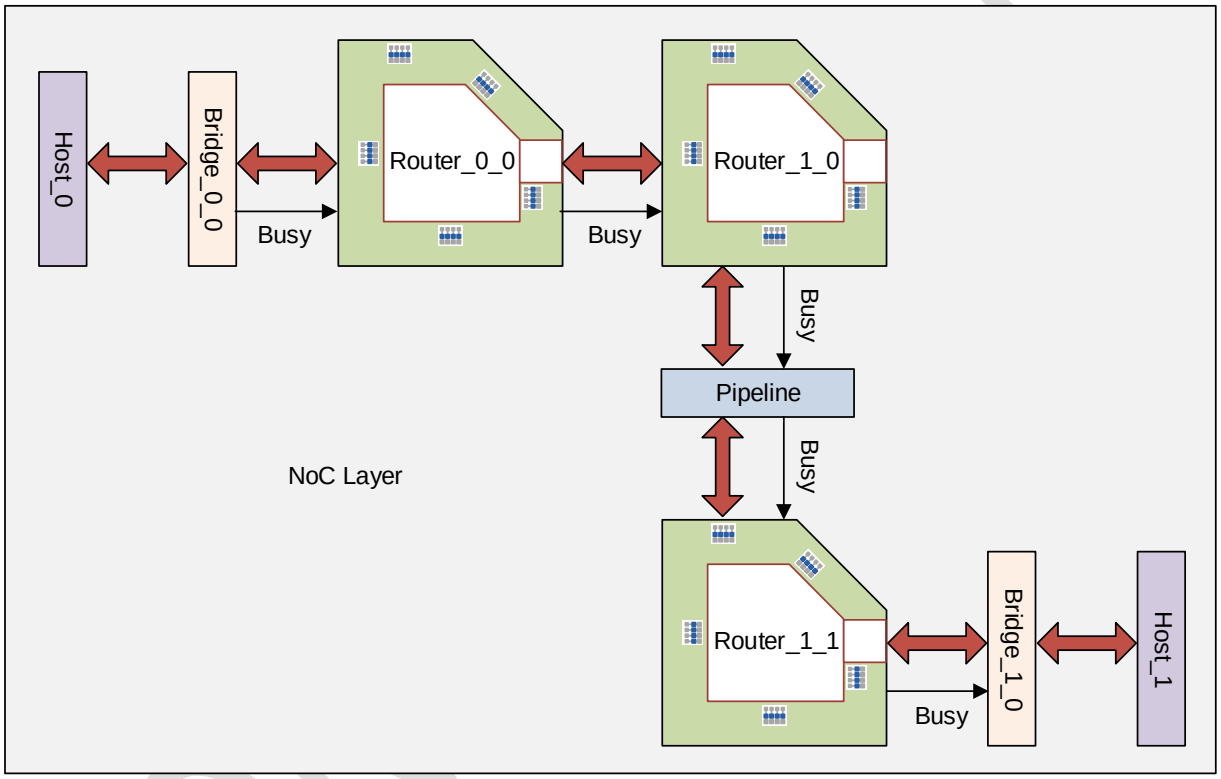


Figure 39. Clock Gating as Implemented in NetSpeed NoC

4.6.3.1 System-Level Clock Gating

System-level clock gating uses a **system_*_clk_en** pin with each bridge or router clock input. This enable is controlled by an external system-clock controller that can be controlled by firmware. The **system_*_clk_en** can be used to control the NoC clock during the boot sequence, power management, etc. No communication with the NoC can occur when the clock gating signal is toggled and sufficient cycles must be allocated for the signal to propagate to all NoC element inputs.

4.6.3.2 *Coarse-Grained Clock Gating*

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

- Routers
- AXI Bridges
- Pipeline stages inserted by NocStudio
- Regbus routers

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

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

A clock-gated NoC element awakes from its clock-gated state when a flit bound for the element is detected at a neighboring element. Because NoC elements can have registered or unregistered outputs, their internal pipelines differ and therefore have a different wake latency. An extra latency cycle can be incurred each time a NoC element wakes. Each element has a hysteresis counter value that determines the number of inactive cycles needed for a router or bridge to clock-gate itself. This value 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.

Coarse-grained clock gating is controlled by hardware, but a software-controlled override signal is provided to disable this feature. The override signal is generated from the node's RingMaster element Clock Gating Override register. Override control is present in each NoC element.

The default hysteresis value required for clock gating is 100 cycles. NocStudio can be used to modify the hysteresis values of individual elements.

Clock-gating overrides and hysteresis values can be programmed on silicon using the Regbus.

Regbus layer elements are also clock gated, using Regbus-specific system-control signals.

4.7 Address Maps and Configurability Options

4.7.1 Address Maps

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

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

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

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

4.7.2 Address Relocation

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

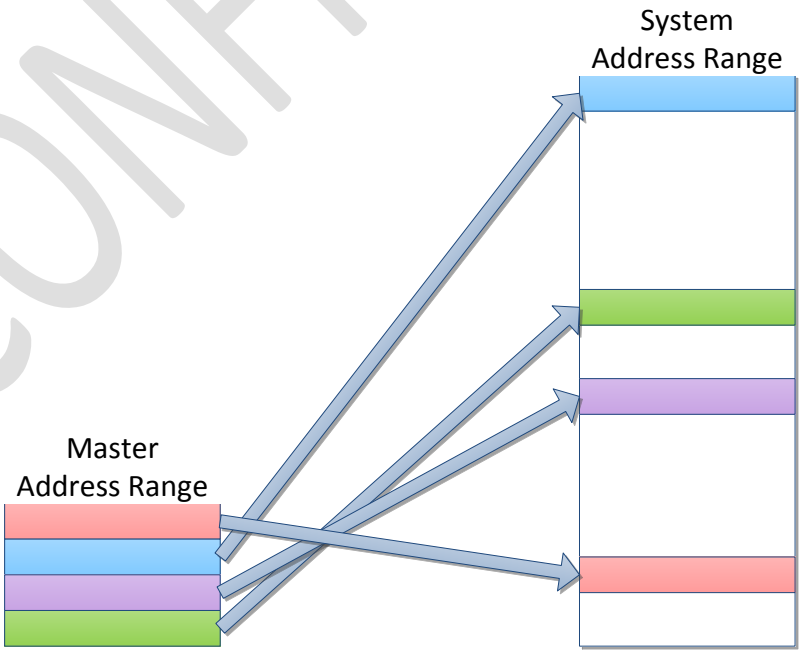


Figure 40: Master with small address space can access regions in a much larger address space

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

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

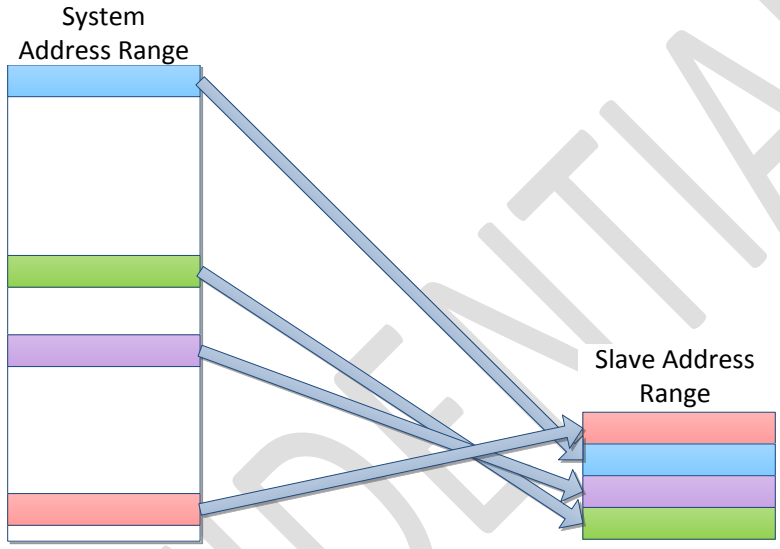


Figure 41: Multiple system address ranges can be compressed to appear as a contiguous slave region

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

4.7.3 Hash Function

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

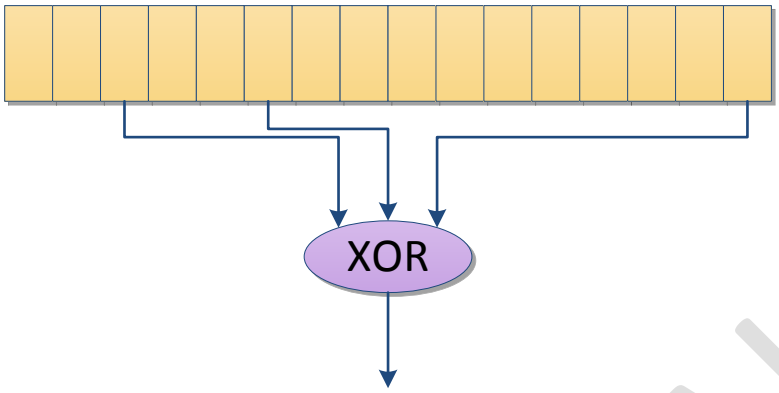


Figure 42: Simple XOR hash function

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

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

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

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

4.8 AID HANDLING AND TRANSACTION ORDERING

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

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

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

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

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

4.9 SPLITTING OF AMBA TRANSACTIONS

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

a. Coherent transactions from ACE and ACEL masters are split at 64B boundary
b. Trasactions sent to slaves providing interleaved read responses are split at 64B boundary
c. Transactions from a master bridge configured to have a read reorder buffer are split at 64B boundary
d. Transactions from a master bridge with traffic to a slave supporting virtualized command interface or traffic to 'Reorder-Bridge' are split at 64B boundary
e. Non-coherent transactions are split at granularity which can be configured at each master bridge. Default value of this is 1024B
f. If the programmed address ranges on a master bridge have lower granularity than the configured split size, then the split size is overridden to match the smallest address range granularity

When transactions are split at a master bridge, responses returning for the split segments have to be merged. When re-order buffer is enabled, it is also used to coalesce the split response

segments. Alternativel, the bridge can be configured to return read response segments from different AIDs in an interleaved manner.

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

4.10 Width Conversion

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

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

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

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

AxCACHE[1] marks transactions as modifiable or non-modifiable. Modifiable transactions provide greater flexibility in NetSpeed NoC to transport and modify transactions passing through the system for greater performance. Non-modifiable transactions are honored. However some transactions marked as non-modifiable will still be subjected to modification, for example if width conversion operation requires that for functional correctness.

4.11 Virtual slave interface

It is common to want a slave device to have multiple interfaces so that different kinds of traffic can be sent without interference. For instance, a memory controller may want to have an interface for normal traffic, as well as one for traffic with real-time requirements, such as display or audio. Even the normal traffic may be sub-divided into latency sensitive flows such

as CPU traffic, and other traffic that is much less sensitive to latency. To avoid head-of-line blocking issues, these various traffic classes need to have separate interfaces to the memory controller or other slave. If they share an interface, then one traffic class can block the others. Multiple physical interfaces can get very expensive because of the number of pins required, as well as the number of physical wires that have to be transported to the slave. Virtual interfaces can reduce both problems.

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

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

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

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

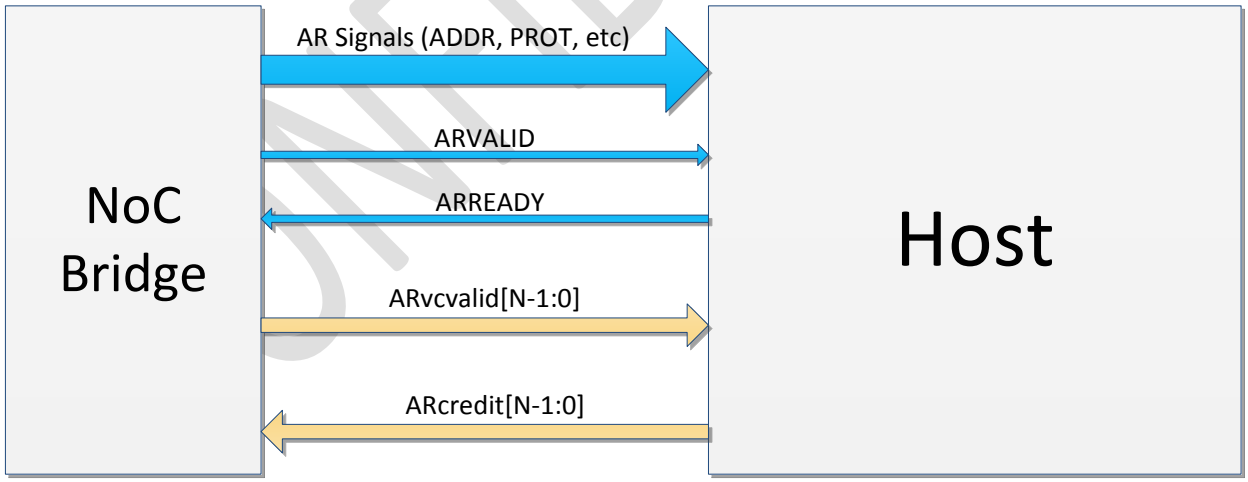


Figure 43: AR interface additions

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

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

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

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

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

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

Virtual channels on AW,W channels are shown below

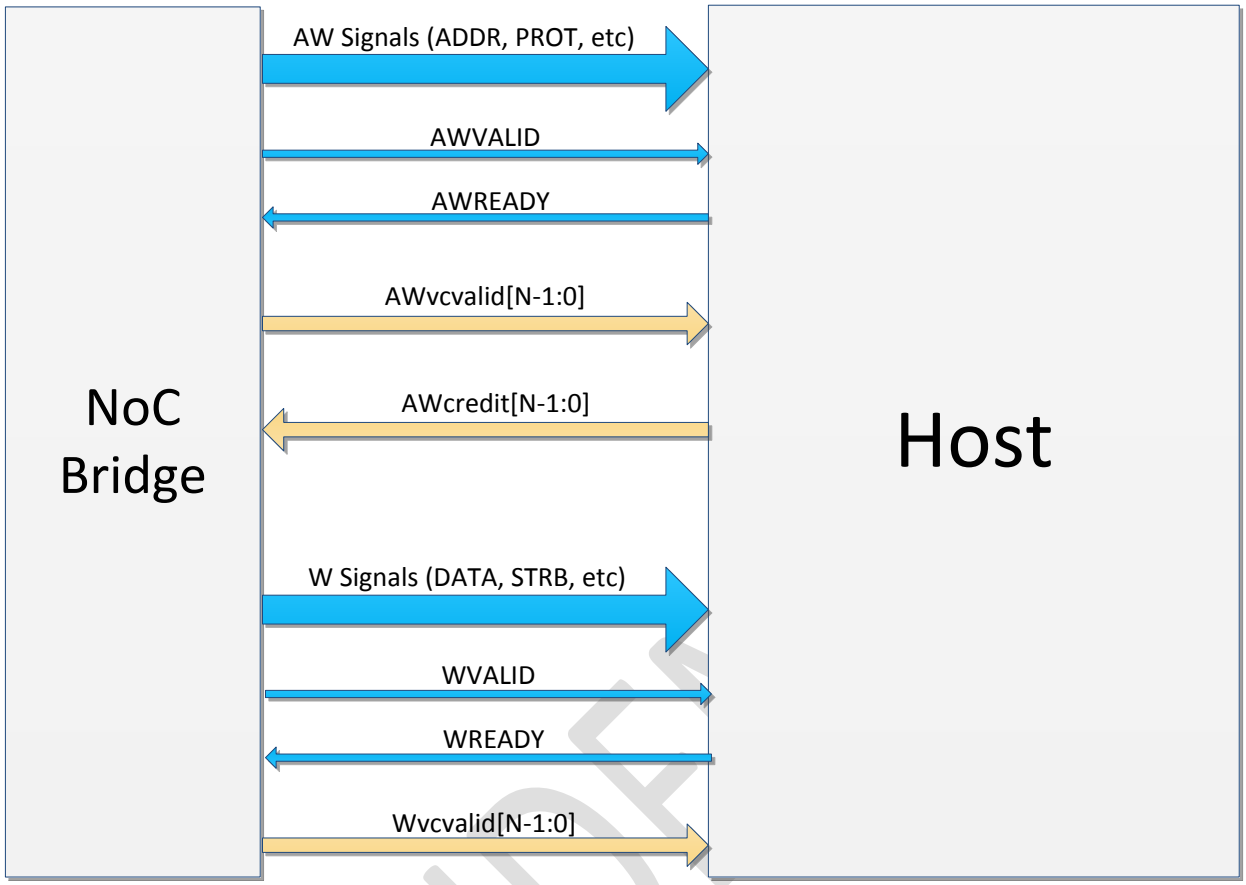


Figure 44 AW-W interface additions

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

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

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

4.12 INTERRUPT

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

5 Low Power Support

5.1 NetSpeed Power Management Architecture

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

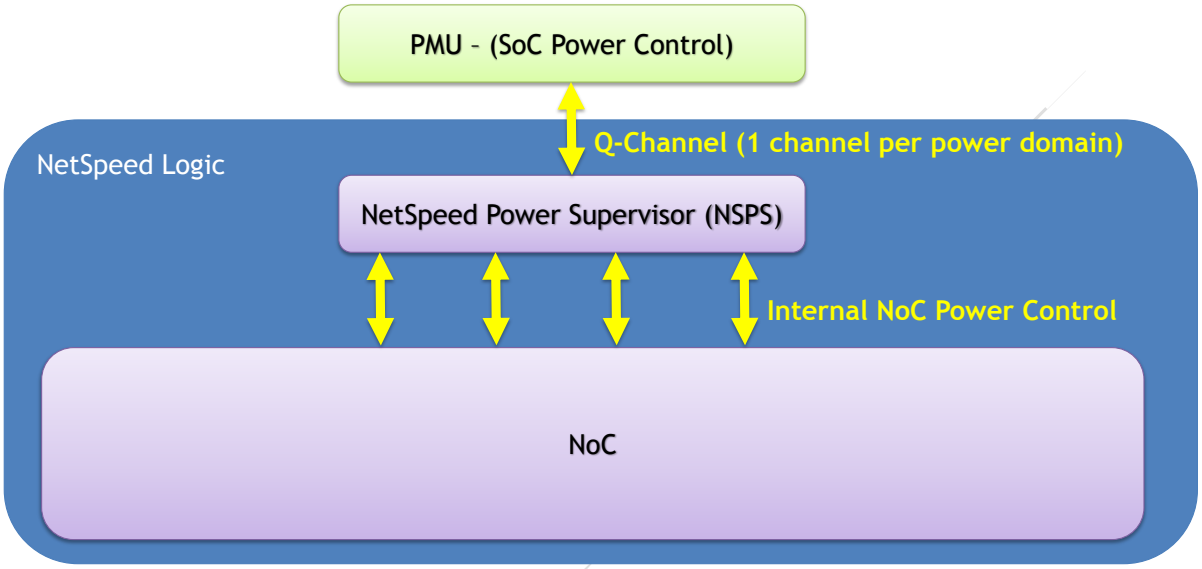


Figure 45 NetSpeed Power Management Architecture

5.1.1 PMU (SoC Power Management Unit)

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

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

5.1.2 NetSpeed Power Supervisor

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

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

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

5.1.3 NoC Element Power Management Logic:

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

5.2 Power Domains and Relationships to Clock and Voltage Domains

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

5.2.1 Clock Domains

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

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

5.2.2 Multiple Voltage Domains

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

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

5.3 Low Power Signaling Interface Between PMU and NSPS

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

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

5.3.1 Power Down Sequence

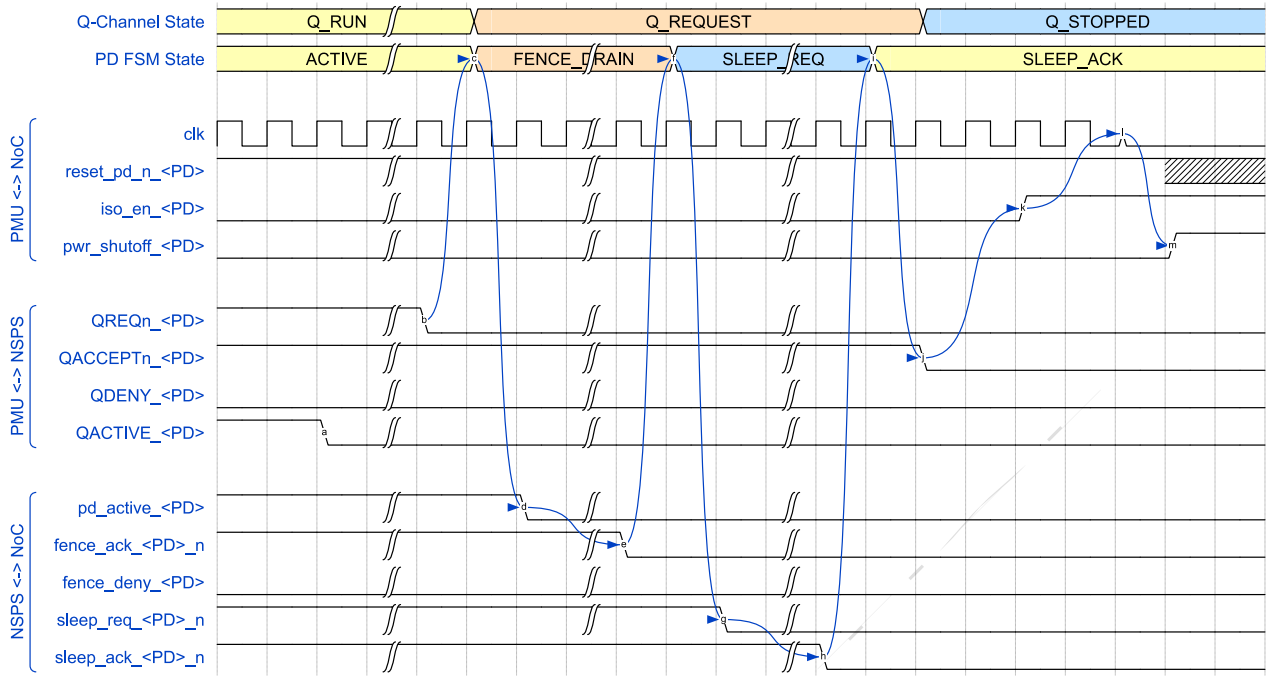


Figure 46 Power Down Waveform Sequence

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

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

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

5.3.2 Power Up Sequence

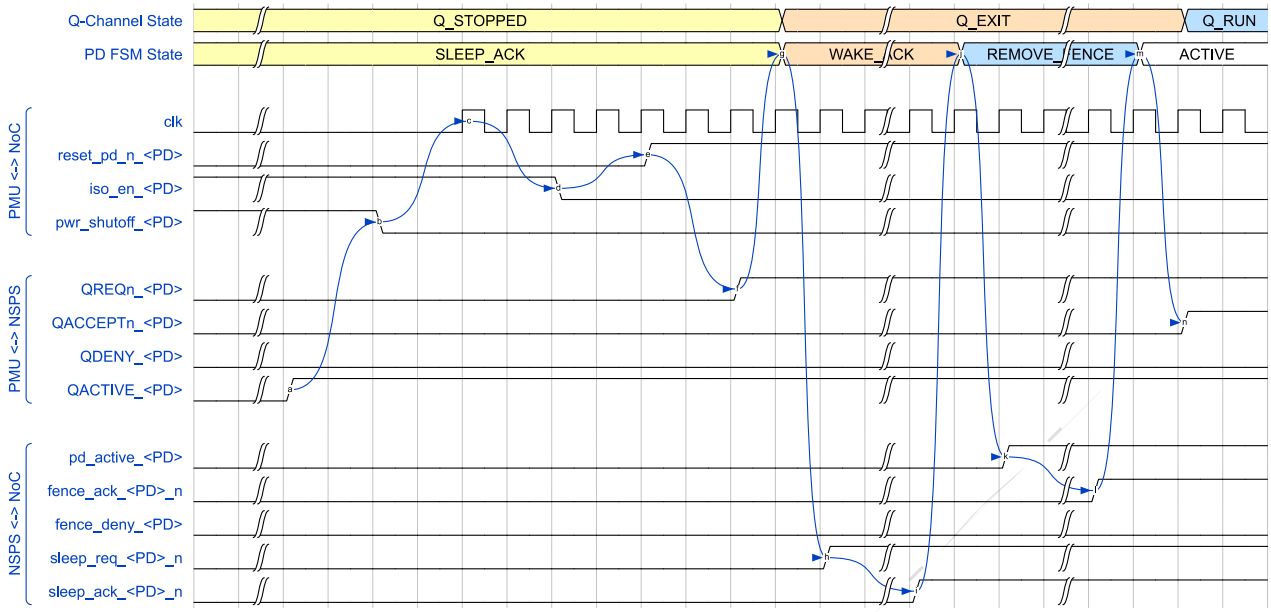


Figure 47 Power Up Sequence Waveforms

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

5.4 Fencing and Draining

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

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

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

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

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

5.4.1 Fencing Behavior

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

1. DECERR response: completed immediately by the master bridge with a DECERR status.
2. Auto-Wake Request: transaction will be held in local storage in the master bridge while it signals NSPS to request any blocking power domains to be woken by asserting QACTIVE.

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

In addition to the master bridge setting, NocStudio maintains a property for each power domain which indicates whether or not it is wakeable. This is set via an optional argument to the *add_power_domain* command, or it may also be updated via the *set_power_domain_auto_wakeup* command. Only when all power domains that are blocking a particular transaction are wakeable will auto-wake requests be generated. If any power domain in this list is not wakeable, the response is forced to become a DECERR.

5.4.2 Fencing and QDENY

When a power down request (*QREQn_<PD>* 1->0) is received for an auto-wake capable power domain, if there are any pending transactions in the NoC that depend on that domain, the NSPS will reject the power down request by asserting *QDENY_<PD>* instead of initiating fencing and draining. The PMU must return Q-channel to the Q_RUN state by raising *QREQn_<PD>*, and it must wait for all activity that depends on the target power domain to complete before a power

down request will be accepted (though it may retry repeatedly while waiting). This behavior holds regardless of the fencing response type configuration at the master bridges (*axi4m_autowake_enable*).

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

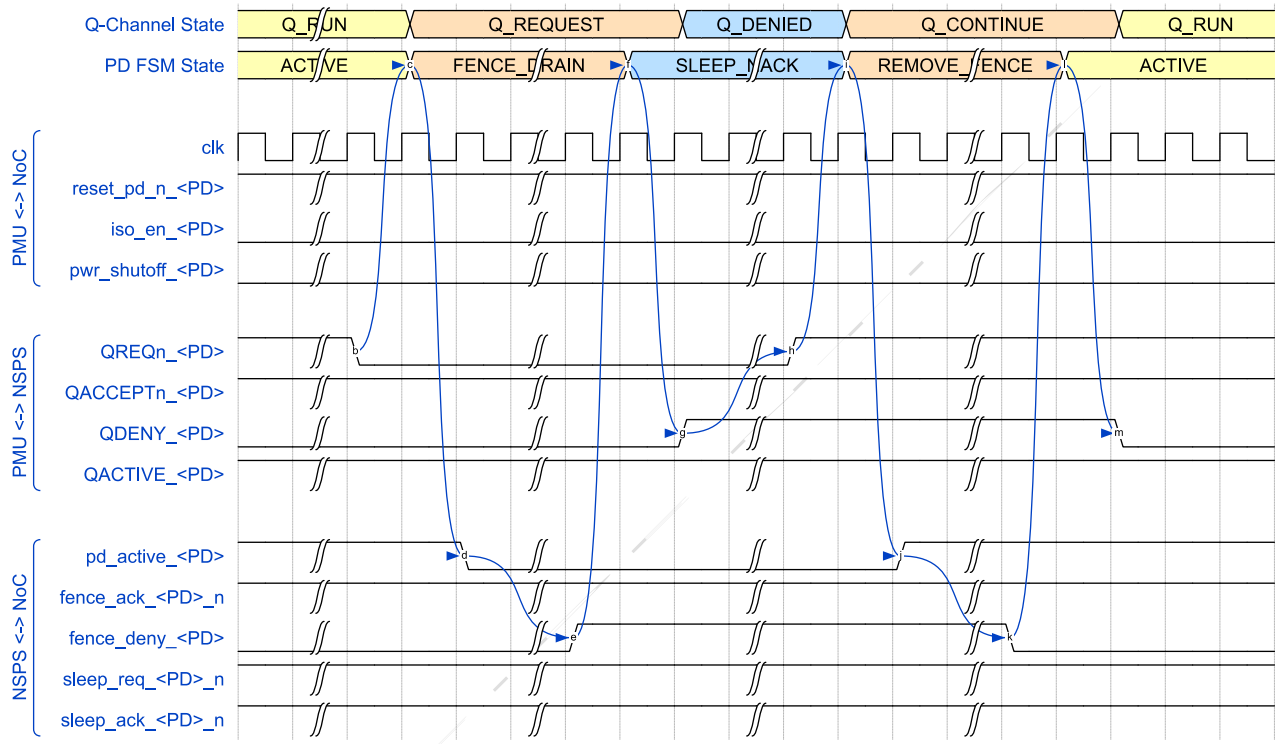


Figure 48 QDENY Waveform Sequence

5.5 Low Power Signals

5.5.1 NetSpeed Power Supervisor Interface to PMU

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

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

Signal Name	Source	Destination	Purpose

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

Table 3 NSPS to PMU Interface

5.5.2 PMU to Power Domain Interface

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

Signal Name	Source	Destination	Purpose
reset_n_<PD>	PMU	NoC elements	Per power domain cold reset. This is asserted to establish clean state following the first power-on event for the power domain. Point to multipoint, subject to clock/voltage domain crossing treatment.
reset_pd_n_<PD>	PMU	NoC elements	Per power domain warm reset. This is asserted to restore clean state after a power removal/restoration sequence. Point to multipoint, subject to clock/voltage domain crossing treatment.

iso_en_<PD>	PMU	isolation clamps	Defined in ns_soc_ip.cpf, this controls isolation clamps that are inferred where outputs of the power domain connect to inputs that exist in other power domains. Clamping function is enabled when this signal is driven high.
pwr_shutoff_<PD>	PMU	power gates	Defined in ns_soc_ip.cpf, this controls power gates for the power domain. Power is removed when this signal is asserted high.

Table 4 PMU to Power Domain Interface

5.6 BRIDGE LOGIC IN HOST POWER DOMAIN

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

5.6.1 AHBLM Compound Bridge

For the AHB Lite master bridge, protocol conversion logic translates the host interface protocol to AXI4 which is driven to an AXI4 master bridge. In this case, due to protocol restrictions described further in section 5.7, the conversion logic must live in the host's power domain. When the bridge's *power_domain* property is set to a different power domain than the *power_domain_host* property, the boundary between these domains appears between the protocol converter and the AXI bridge.

5.6.2 Voltage Domain Boundaries between Host and Bridge

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

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

5.7 AHB Lite Master Bridge (AHBLM)

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

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

5.8 Always On Power Domains

Always on power domains are those for which the *power_domain_always_on* property is set to "yes." The fundamental expectation is that logic in such power domains is powered before or coincident with other power domains in the NoC (i.e., at initial start-up or boot time), and that it remains powered continuously during operation of the device. They are defined in CPF power intent files so that signal transitions between them and other power domains can be recognized and properly handled. However, these power domains do not have power gating (*pwr_shutoff_<PD>*) or isolation (*iso_en_<PD>*) logic or signaling inferred in the CPF files as such logic is unnecessary. Q-channel interfaces and the associated NSPS power control logic are not present for always on power domains.

5.8.1 Clock Gated Only Power Domains

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

5.9 Gemini Low Power

Gemini low-power support has additional requirements and considerations. This section will describe these in more details.

5.9.1 Coherent IP and their Master Ports must share a power domain

The CCC, IOCB, and LLC hosts must share a power domain with their master port bridges. This is to ensure that these agents can always issue a request to the master port without having to check if the master bridge is powered on, or to go through a sequence to power it on.

5.9.2 Coherency Connect/Disconnect

ACE masters and ACE-lite+DVM masters must go through a special sequence in order to be powered down. Each of these agents can receive snoops from either CCC or DVM. In order to shut-down, they must get into a state where they no longer need to accept snoops and the CCC and DVM know that they shouldn't send them. This is implemented in the coherency connect/disconnect protocol. Only when these agents are successfully disconnected can they be powered down.

When these agents attempt to connect to coherency, the CCC, DVM and all paths to and from those agents must be either powered up, or allowed to be powered up with an auto-wake. If one of these elements was in a decode-error type of power domain and not powered up, the connect sequence would get stuck and the agent could hang. Either software needs to be very careful about controlling these power domains and ensuring they are available before any ACE or ACE-lite+DVM is powered on, or they system should choose to place these elements in an auto-wake domain.

ACE-lite, ACE-lite+DVM, and ACE-lite-converted agents all send their IO coherent requests to IOCB. When the requests arrive at IOCB, if it is not already connected to coherency, it will start the process of connecting. Just as with the ACE agents, the CCC must be powered on or auto-wake before the IOCB can connect. This means it must be available before any IO coherent traffic can be issued, or the system could hang.

5.9.3 DVM Power Domain as Decode Error

The DVM protocol has unique properties. DVM requests cannot handle decode-errors like other requests. The DVM sync request, for instance, requires that a snoop is sent from the DVM to indicate the synchronization has taken place. If the DVM were powered down and was in a decode-error type of power domain when a DVM request is made, the master could hang. For this reason, either the PMU must ensure the DVM is powered up before any ACE or ACE-lite+DVM agents are enabled, or the DVM must be part of an auto-wake domain.

5.9.4 IOCB Power Down

The IOCB can be powered down whenever it is inactive, as it doesn't hold state. This makes it a useful candidate for an auto-wake power domain. If the IOCB is requested to power down while it still has requests outstanding, it will reject the power down request with a QDENY.

5.9.5 CCC Power Down

The CCC stores directory state for coherent masters and so it can't be turned off at random times, even if no traffic is outstanding. CCC can only be powered off when all coherent masters

are disconnected from it, including IOCB. When there are no connected agents, any information in the directory will be stale, and the CCC can be powered down. If the CCC is part of an auto-wake domain and is requested to shut down, it will produce a QDENY to reject the power down if it detects that there are connected agents.

5.9.6 LLC Power Down

The LLC has unique power domain requirements. It holds a significant amount of data in the cache. It may also be configured as a ScratchPad RAM. In order to power down without data loss, the cache must be flushed first.

To flush the LLC of dirty data, the LLC ways must all be disabled for allocation, to avoid new lines being added to the cache. After that, a flush engine must be initiated that pushes all dirty data to memory, invalidating the cache as it goes. The invalidation is to ensure that write requests don't hit in the cache and modify a line that was already cleaned. Once the flush engine is completed, and traffic is disabled to the LLC, those LLC ways should be marked as disabled.

If any of the ways are set up as ScratchPad RAM, the LLC will reject a power down sequence. This is because the data will be lost. If the data is no longer needed, software should reprogram those ways as disabled. Power down of the LLC can only happen if all ways are disabled as cache and disabled as RAM. This tells the LLC that no data is currently stored.

If the LLC is part of an auto-wake domain, it will reject any power down request until the entire cache is disabled.

6 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.

6.1 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 49 shows the resources and dependencies for this example.

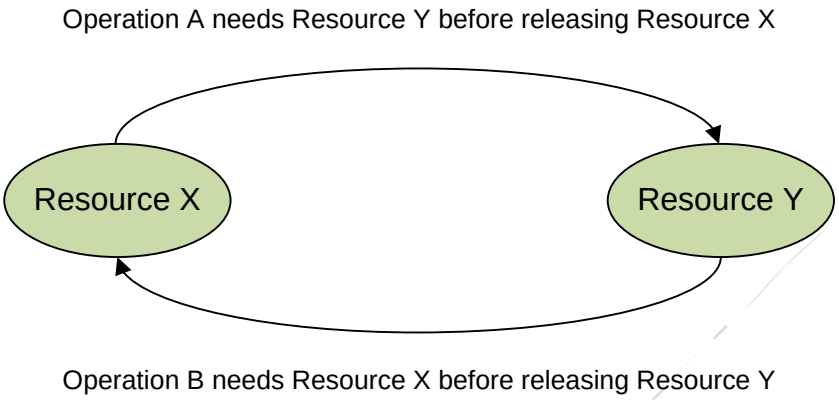


Figure 49. Simple Deadlock Graph with Two Resources

Figure 49 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 50 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.

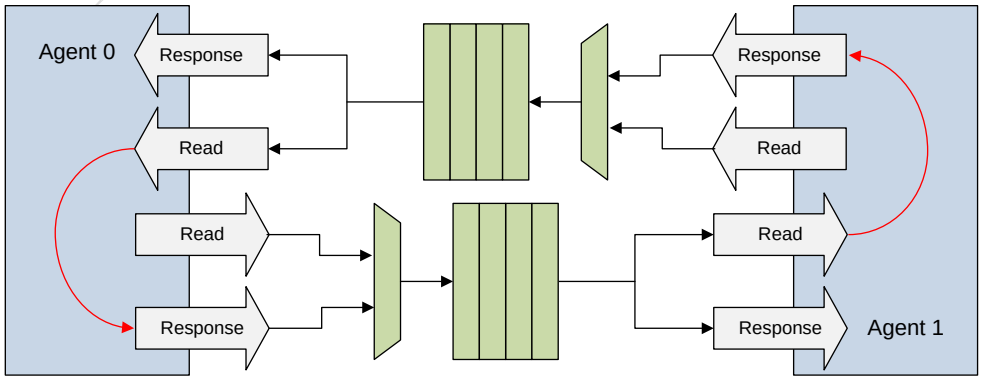


Figure 50. Simple Interconnect with Deadlock

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.

6.2 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 50 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 50, 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 51, NocStudio also generates a comprehensive system dependency graph as well so that architects can crosscheck and ensure that there are no deadlocks.

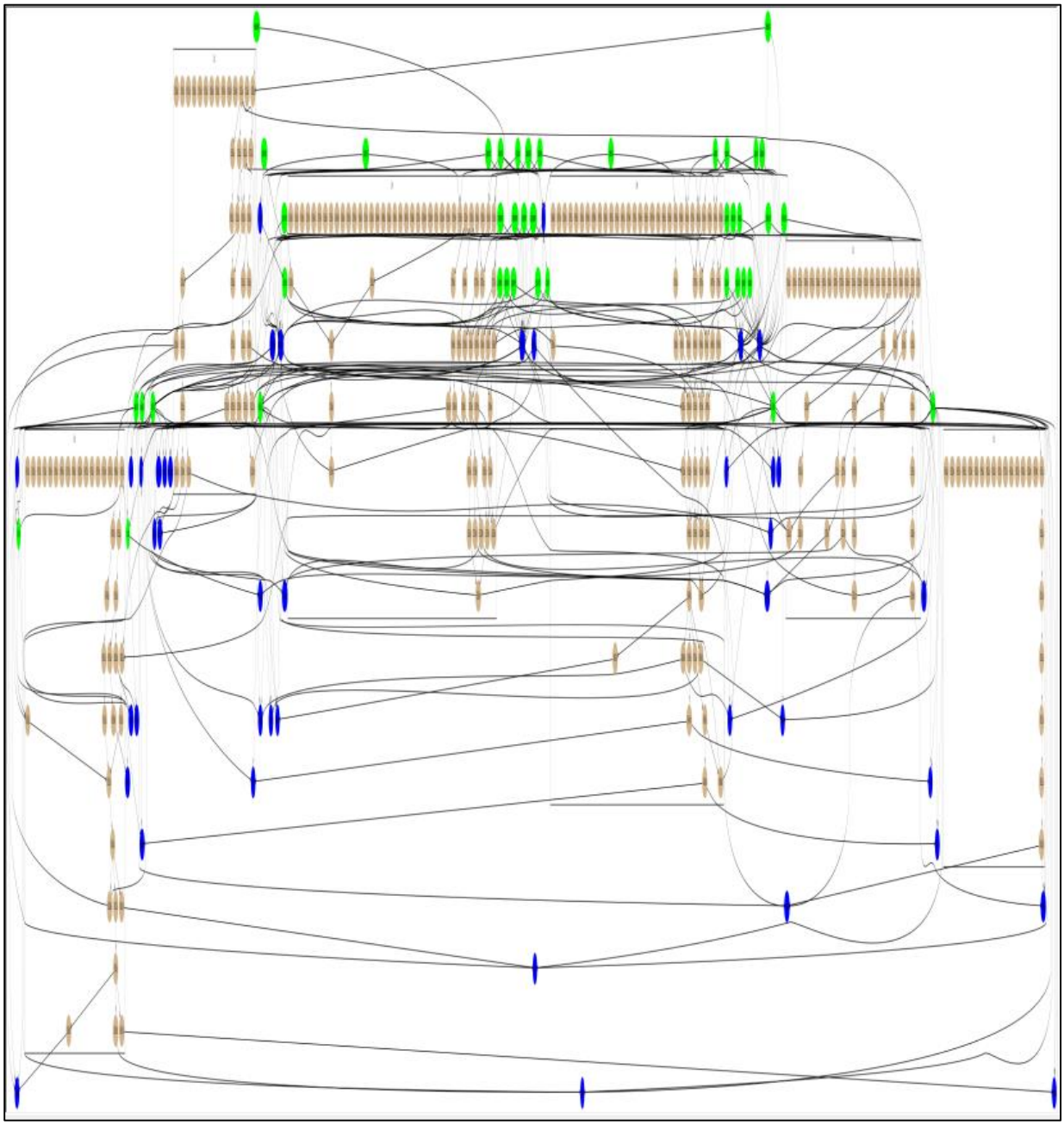


Figure 51 - System Level Dependency Graph

7 Security

7.1 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.

7.2 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.

7.3 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.

7.4 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.

7.5 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.

7.5.1 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.

7.5.2 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.

7.5.3 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.

7.6 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.

7.7 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.

8 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

8.1 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.

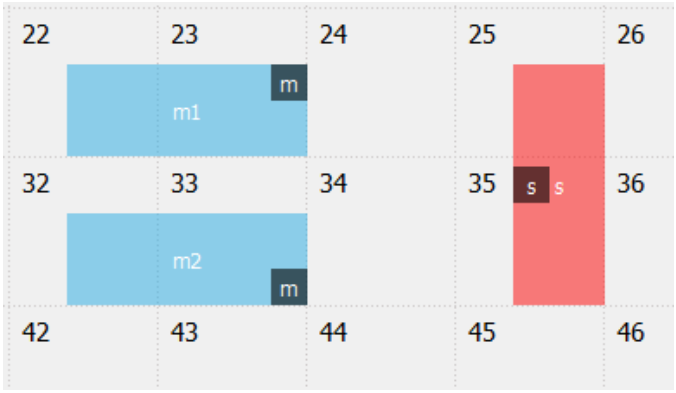
8.1.1 How to Define the Priority

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.

8.1.2 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
```

8.1.3 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	-	-	-	-									

8.1.4 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	-	-	-	-									

8.1.5 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	-	-	-	-									

8.2 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.

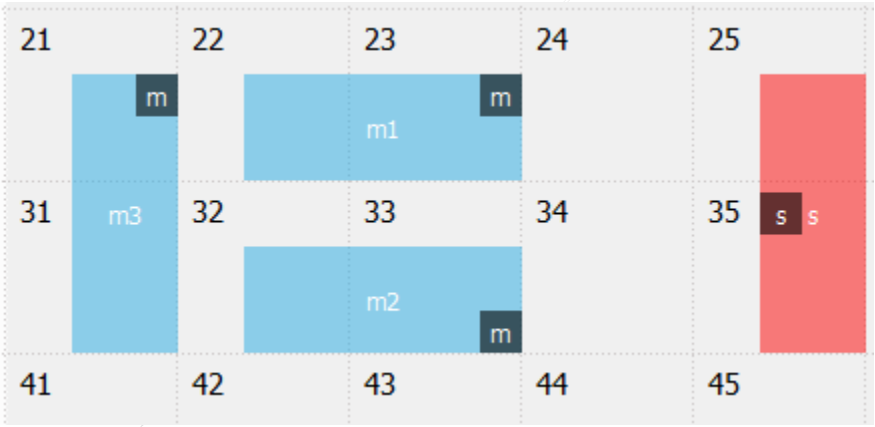
8.2.1 How to Define the Weight

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

8.2.2 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
```

8.2.3 Example #1

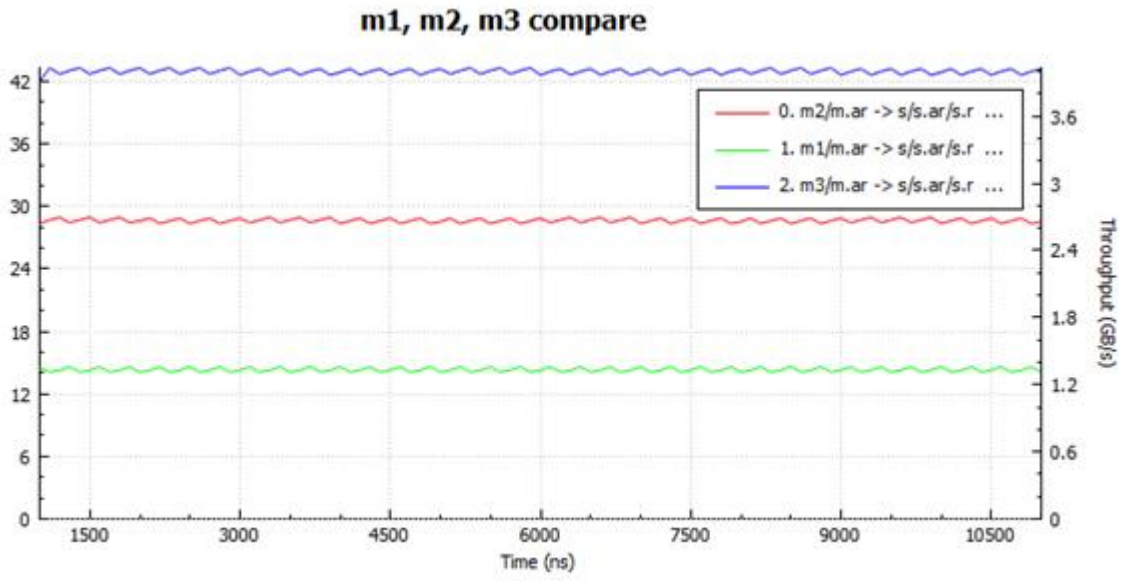
In a case where traffics from all 3 masters are contending for resources, the ratio of resulting bandwidth for each master bridge follows that of the weight:

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.



8.2.4 Example #2

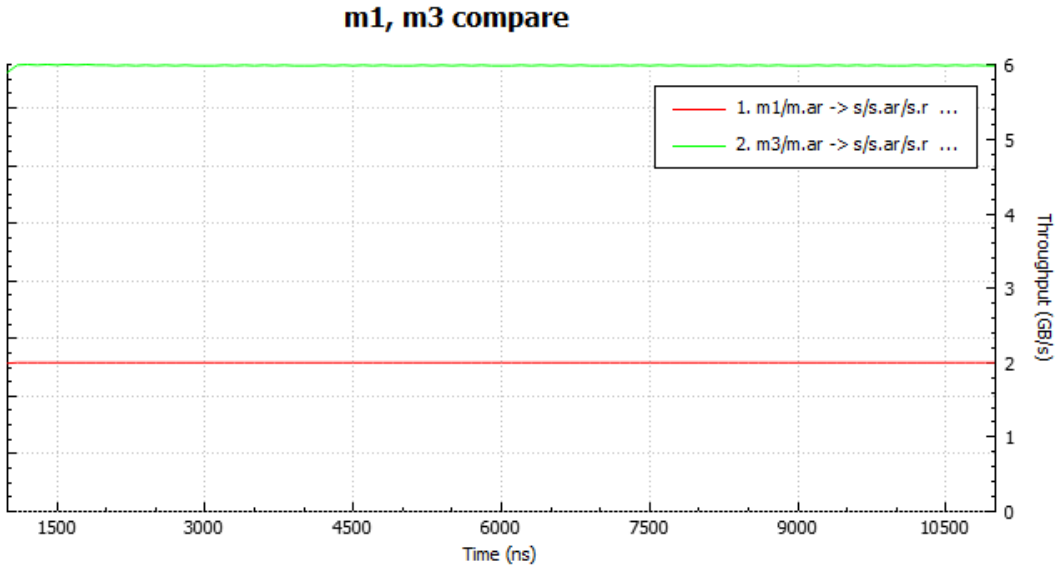
In a case where only 2 traffics are contending for resources, the ratio of resulting bandwidth for each of the 2 master bridges follows their defined weights:

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.



8.3 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.

8.3.1 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.

8.3.2 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.

8.3.3 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.

8.3.4 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.

8.3.5 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.

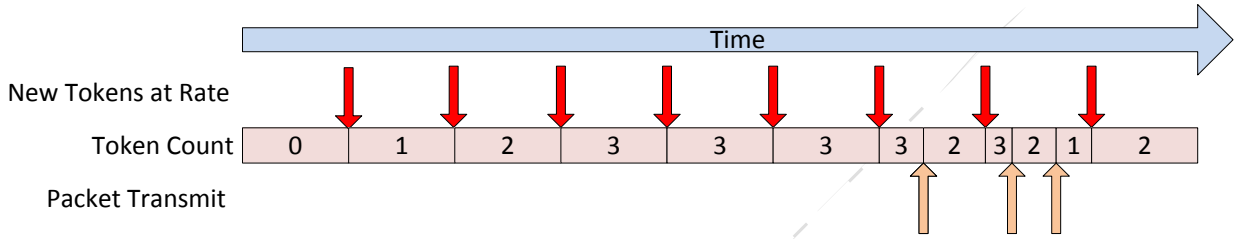


Figure 52 - Token Count increases at specified rate. Packet transmit decrements count.

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.

8.3.6 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.

8.3.7 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.

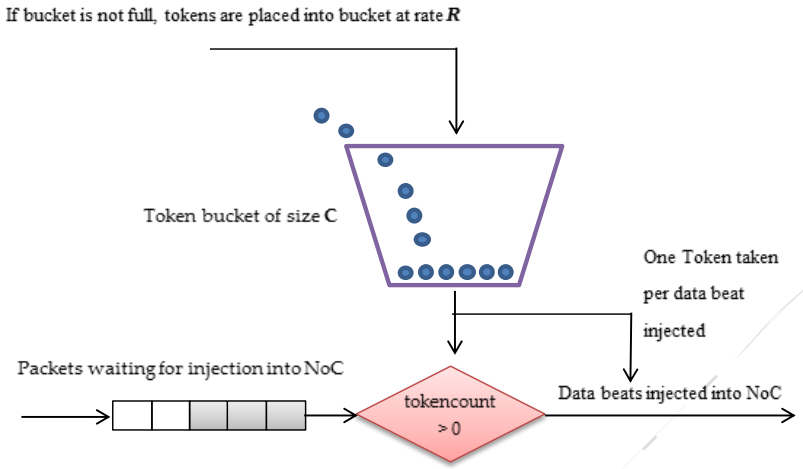


Figure 53 - Token Bucket

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.

8.3.8 Token Usage

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

8.3.9 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

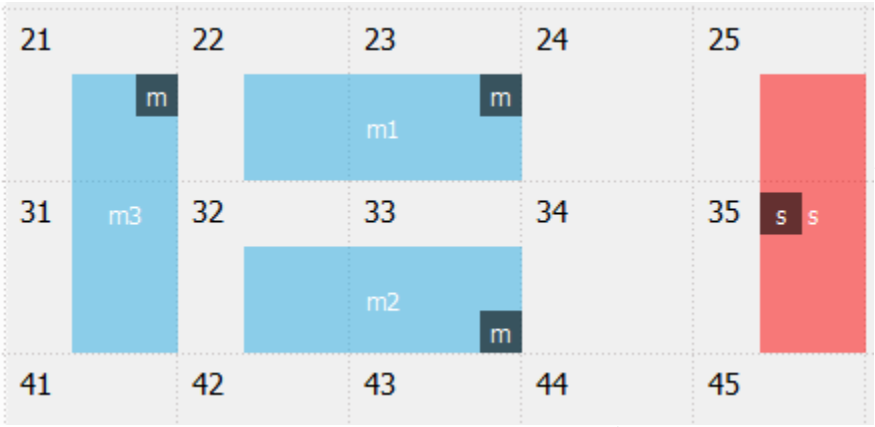
19:16	Bucket Size. This indicates the maximum number of token that may be accumulated at an interface when rate limiters are enabled
20	Rate Limit logic enable. Rate limiter logic is used for arbitration only when the enable bit of the rate limiter configuration register is set to 1

Table 5 - Rate limiter Configuration Register

8.3.10 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.

8.4 DYNAMIC PRIORITY SUPPORT FOR ISOCHRONOUS TRAFFIC

8.4.1 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.

8.4.2 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.

8.5 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.

9 NoC Serviceability: Regbus Layer

9.1 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.

9.1.1 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 54. 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.

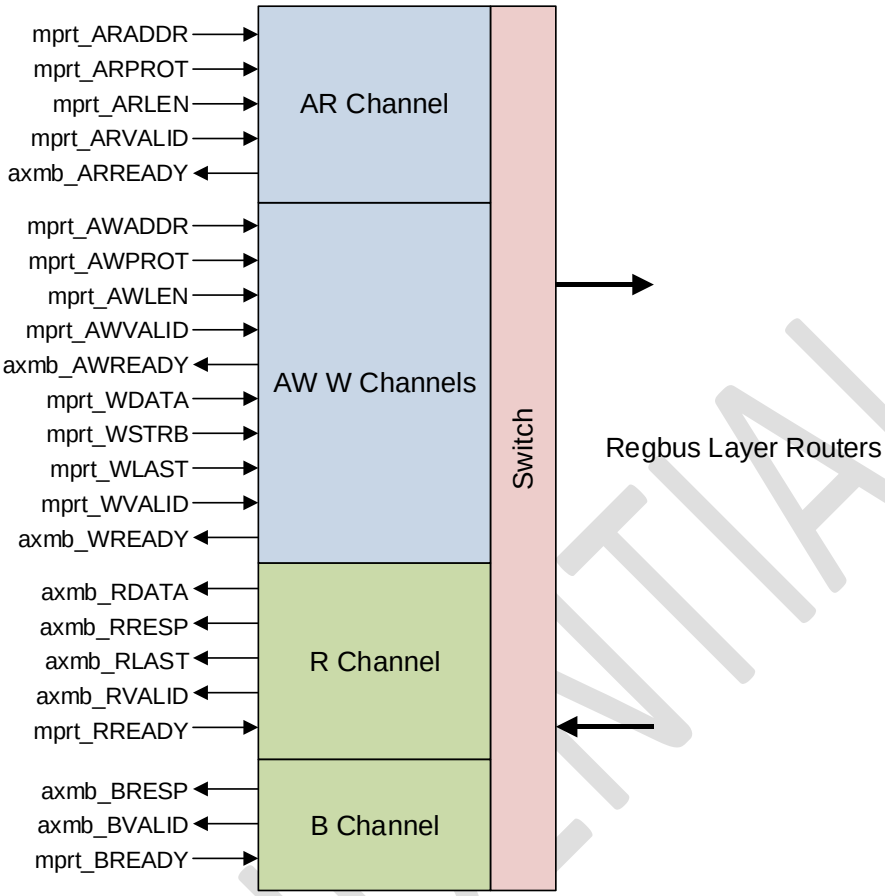


Figure 54. Regbus Master Bridge

9.1.2 The Regbus Layer

As shown in Figure 55, 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.

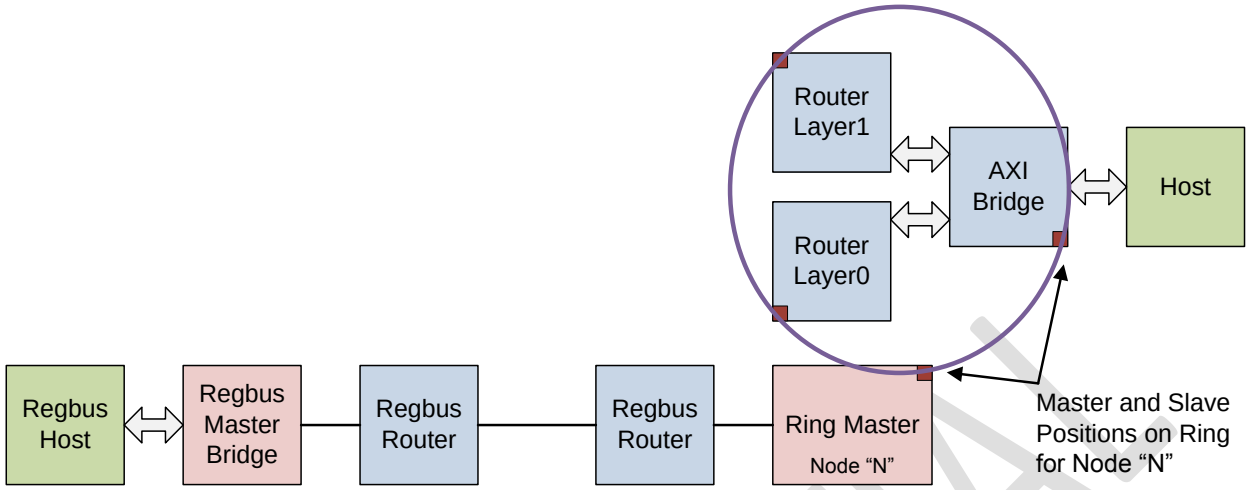


Figure 55. Regbus Layer Communication

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.

9.1.3 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 56 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.

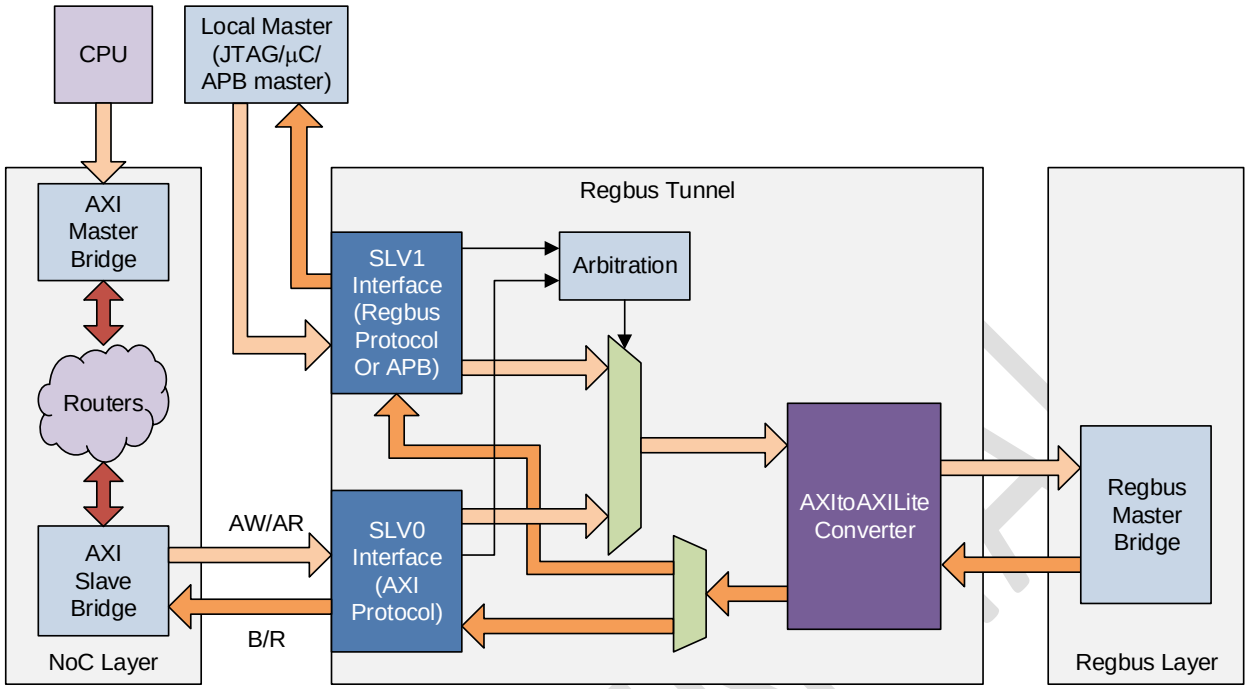


Figure 56. Regbus Tunnel Connects Primary NoC Layer to Regbus Layer

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.

9.1.4 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.

9.1.4.1 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.

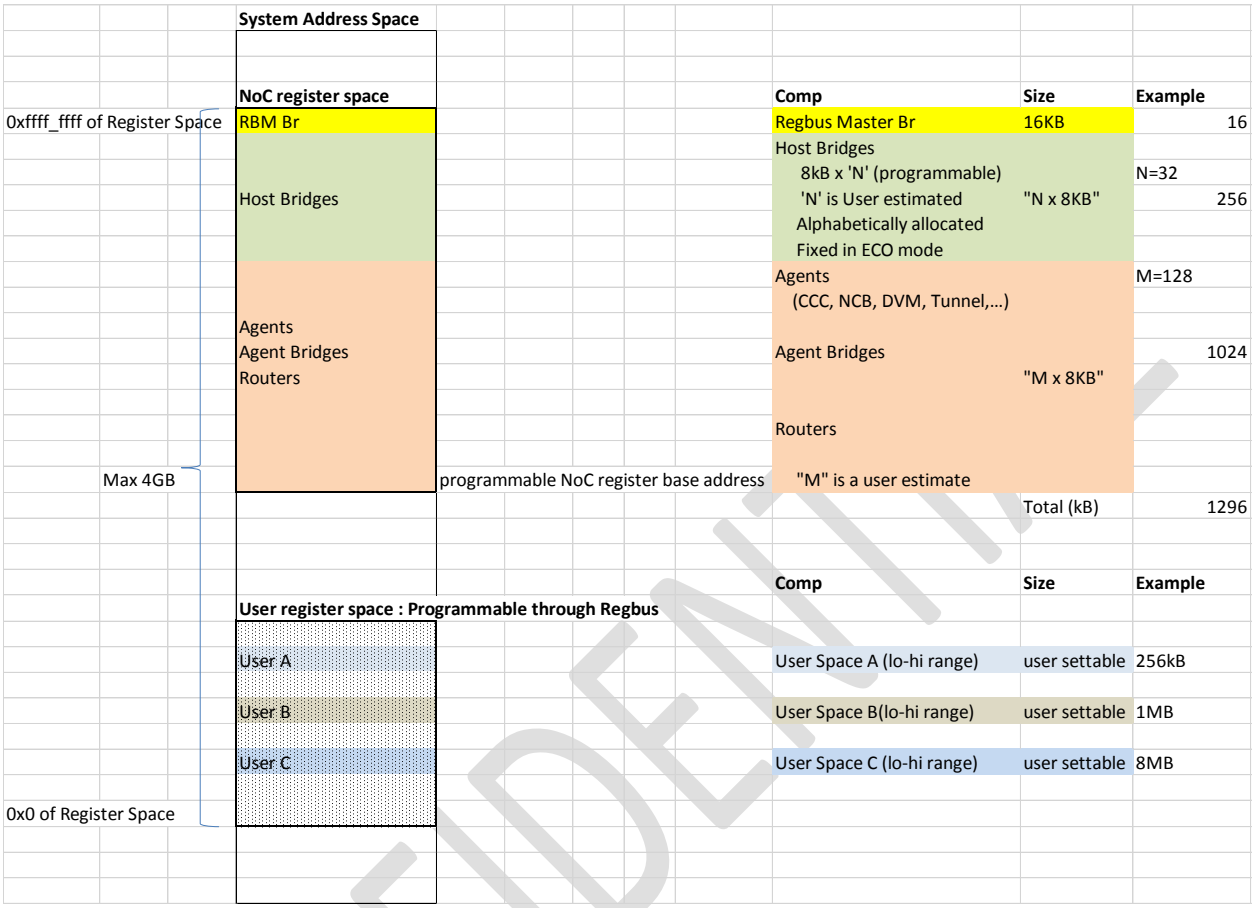


Figure 57: Regbus Address Map

9.2 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 6: 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 7: 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

9.3 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 8: 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

9.4 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 9: 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

9.5 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

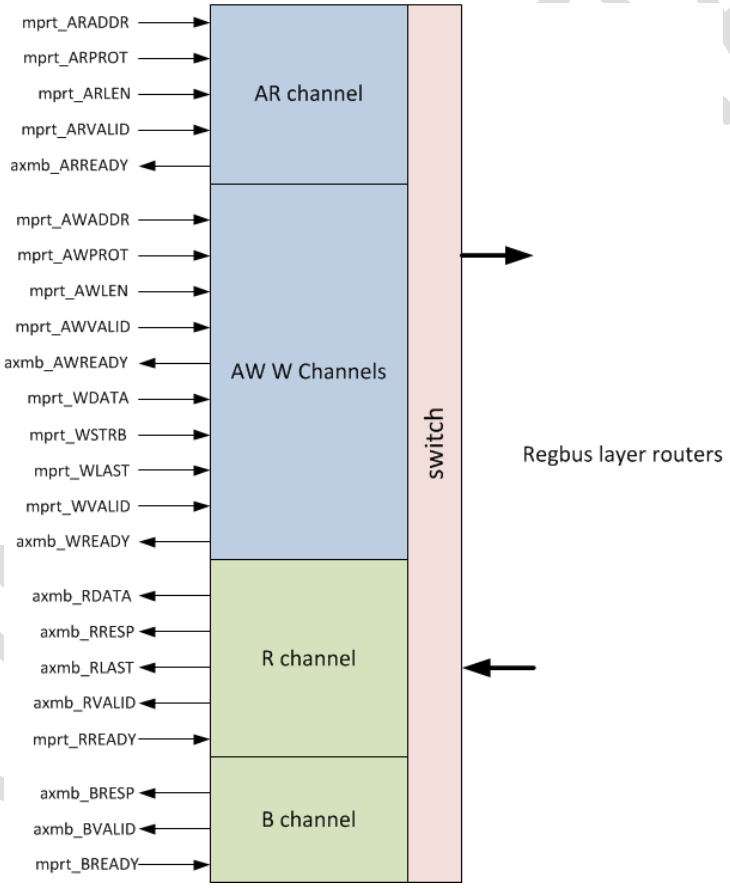


Figure 58: Register bus master bridge

The list of input signals is specified below:

Table 10: 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

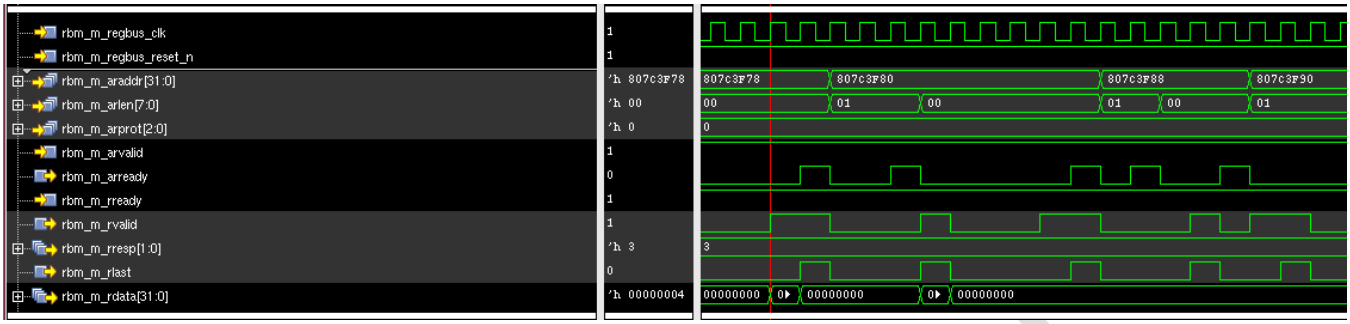


Figure 59: Waveform showing read requests and responses at the register bus master interface

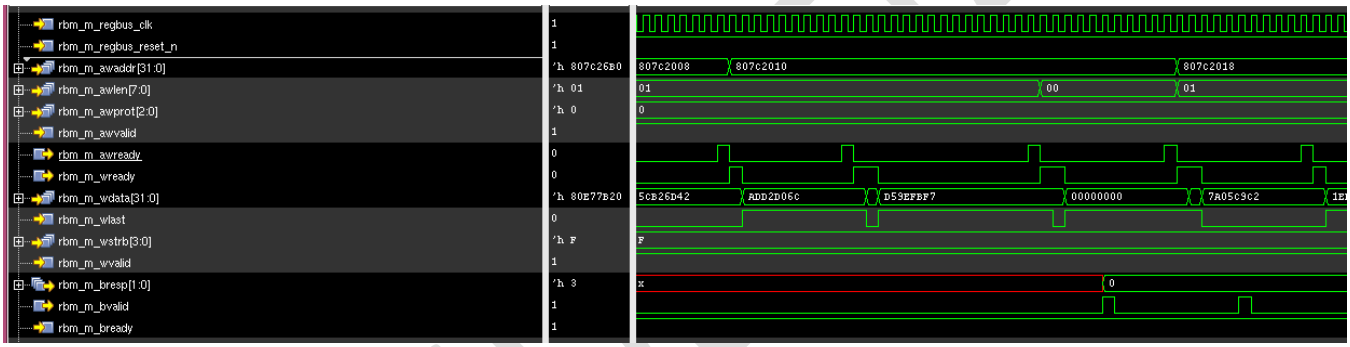


Figure 60: Waveform showing write requests and responses at the register bus master interface

9.6 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.

9.7 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.

9.8 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 11: 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

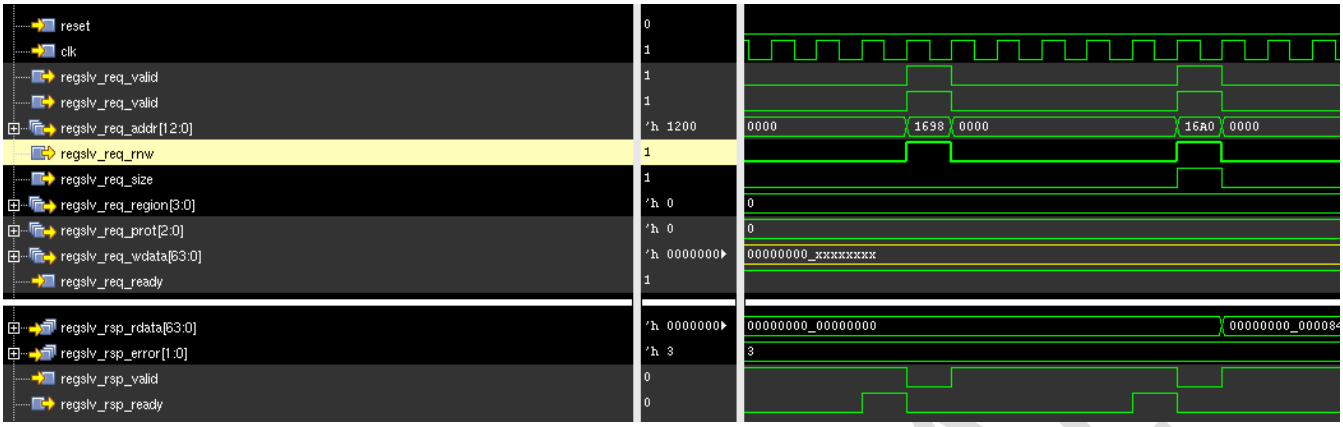


Figure 61 : Waveform showing ring slave read requests and responses (4B and 8B)

Figure 61 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.

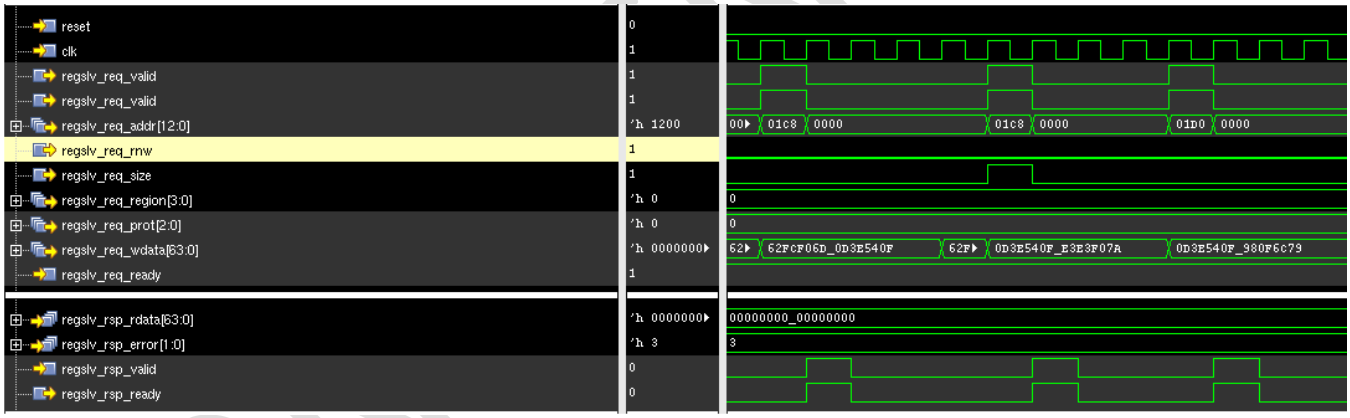


Figure 62 : Waveform showing ring slave write requests and responses (4B and 8B)

Figure 62 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.

10 Programmers Model

NetSpeed NoC delivers a rich set of registers used for NoC control, debug and performance monitoring of the NoC. This section describes, 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.

10.1 STREAMING BRIDGE REGISTERS

10.1.1 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.

10.1.2 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.

10.1.3 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.

10.1.4 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.

10.1.5 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.

10.1.6 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.

10.1.7 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.

10.2 AXI Master Registers

10.2.1 Base Address Register (AM_ADBASE) and Address Mask Register (AM_ADMASK)

These registers specify the base addresses and masks of different slave ranges accessible from this master. One base, mask, and reloc register set per address range assigned to the master. These registers can be individually designated as read-only or read-write based on NocStudio property assigned to address ranges.

10.2.2 Slave Address Relocation Register (AM_ADRELOCSLV)

Register used to relocate a master address to slave address.

10.2.3 System Address Relocation Register (AM_ADRELOCSYS)

Register used to relocate a master address to system address.

10.2.4 Timeout Configuration Register (AM_TOCFG)

This register is used to configure response timeouts.

10.2.5 Check Outstanding to Specified Slave Register (AM_OSSLV)

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

10.2.6 Configurations Register (AM_CFG)

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

10.2.7 Status Flags Register (AM_STS)

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

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

10.2.8 Bridge ID Register (AM_BRIDGE_ID)

This register holds the unique identifier assigned to the master bridge.

10.2.9 NoC Version ID Register (AM_NOCVER_ID)

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

10.2.10 Status and Error Register (AM_ERR)

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

10.2.11 Slave ID of Timed Out Requests Register (AM_TOSLVID)

AR slvid and AW slvid fields indicate slave IDs to which a read, write response timeout was detected. Note that slvid encoding is not same as the bridge ID of the slave. NocStudio provides a table mapping the slvids to the actual slave ports accessible from the master bridge.

10.2.12 Local Read Decode Error Address Register (AM_ERA)

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

10.2.13 Local Write Decode Error Address Register (AM_EWA)

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

10.2.14 Interrupt Mask Register (AM_INTM)

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

10.2.15 Address Capture Value Register (AM_CADDR)

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

10.2.16 Address Capture Mask Register (AM_CADDRMSK)

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

10.2.17 Command Capture Control Register (AM_CCMD0)

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

10.2.18 Command Capture Mask Register (AM_CCMDMSK0)

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

10.2.19 Count of Captured Events Register (AM_CNTR0)

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

10.2.20 Number of Captured Commands over which Latency is to be Measured Register (AM_LATNUM0)

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

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

10.3 AXI SLAVE REGISTERS

10.3.1 Status Flags Register (AS_STS)

This register holds the slave bridge status bits.

10.3.2 Bridge ID Register (AS_BRIDGE_ID)

This register holds the unique identifier assigned to the slave bridge.

10.3.3 Status and Error Register (AS_ERR)

This register is the status and error register.

10.3.4 Interrupt Mask Register (AS_INTM)

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

10.4 AHB2AXI Converter Registers

10.4.1 Control Register (AHBM_CTL)

This is the control register for AHB master bridges.

10.4.2 Status Register (AHBM_STS)

This is the status register for AHB master bridges.

10.4.3 Interrupt Mask Register (AHBM_IM)

This is the Interrupt Mask Register for AHB master bridges.

10.5 AXI2AHB Converter Registers

10.5.1 Control Register (AHBS_CTL)

This is the control register for AHB slave bridges.

10.5.2 Status Register (AHBS_STS)

This is the status register for AHB slave bridges.

10.5.3 Interrupt Mask Register (AHBS_IM)

This is the Interrupt Mask Register for AHB slave bridges.

10.6 APB Registers

10.6.1 Bridge Version Register (APBSLV_BRIDGE_VERSION)

This register holds the version number of the bridge.

10.6.2 Bridge ID Register (APBSLV_BRIDGE_ID)

This register holds the unique identifier assigned to the bridge.

10.6.3 Slave Sleep Status Register (APBSLV_SLVS_SLEEP_STATUS)

This register holds the slave's sleep status.

10.7 DVM Host Registers

10.8 DVM Active Vector Register

This register indicates active DVM agents in the system. The DVM IP module is parameterized for a maximum number of agents supporting DVM in the system. This parameterization is taken care of by NocStudio. When reset is de-asserted, the DVM IP module expects that all specified DVM agents in the system are active. This is reflected in this readable, writable ACTIVE_VECTOR register. If, due to low power mode, or other reasons, a DVM agent in the system is shut down, the DVM IP module needs to be made aware of this. To do this, the bit position of the DVM agent in this ACTIVE_VECTOR register should be set to 0.

On seeing a 0 for an agent in the ACTIVE_VECTOR, the DVM IP module ensures that no snoops are sent to it, and does not wait for snoop responses or completions from this agent.

10.8.1 DVM Error/Fault Log Register

This register logs DVM agents returning snoop responses.According to the AMBA spec on Distributed Virtual Memory transations, when an agent in the system receives a DVM transaction, it must respond in one of two ways.

1. If it can perform the requested action, it must respond with CRRESP = 0b00000.
2. If it is unable to perform the requested action, it must respond with CRRESP = 0b00010.

The FAULT_LOG register logs which of the agents responded with CRRESP=0b00010. It is a sticky register, so once set, it remains set until cleared by writing 0 to that bit.The maximum number of DVM agents supported is 256. The entire fault log can hence take from one to four 64-bit registers. Unused bits within the 64-bit registers are tied to 0.Each agent, or host, in the system is assigned a corresponding bridge ID, based on the bridge it is connected to. The fault log register is based on this bridge ID.

10.8.2 DVM Status Register (DVM_STS)

This register indicates the "busy" or non-idle state of the internal logic within the DVM IP module. When all traffic has been serviced, and no new transactions are received, the bitwise-AND of these bits will be 0, indicating a state of quiescence.

10.8.3 DVM Agent Disable Status (DVM_AGENT_DISABLE_STATUS)

This status register is used to indicate that a change to the active agent vector has taken effect. When an agent is removed from the active agent vector, snoops targeting that master may still be queued or inflight. While snoops are outstanding, the master must continue to operate to satisfy snoop requests. This register indicates that all snoops to the disabled agents have completed and no new snoops will be issued to those agents.

The status indicates whether any snoops outstanding at the time of the last modification of the DVM active vector register(s) have all completed or not. A value of 1 indicates that the snoops have all completed and it is safe to power off or otherwise disable the masters that were removed from the active vector. A value of 0 indicates that snoops are still outstanding and the master must continue to operate and accept snoops.

10.9 CCC Host Registers

10.9.1 CCC Speculative Fetch Vector Register

This register controls the speculative fetch behavior of the Cache Coherency Controller. Speculative fetch means that the CCC will issue reads to memory or next level of cache before the directory lookup is performed or the snoops are sent. This allows a lower latency for these reads in the case of a directory miss. If snoops need to be sent for this request, data may be retrieved from a caching agent. This means the speculative fetch may be increasing memory bandwidth in case where it wouldn't have needed to send a read.

This set of registers specifies a bit vector where each bit corresponds to a master agent on the NoC. When requests from this agent arrive at the CCC, a comparison with the bit vector will determine whether speculative fetch is currently enabled for that agent.

10.9.2 CCC Active Agent Vector Register

This register specifies which of the ACE masters are currently active and capable of receiving snoop requests. This register contains a bit vector where each bit corresponds to an ACE master bridge. The Bridge ID determines which bit corresponds to each bridge.

This vector can be used to disable snoop-only ACE agents, or to power off an ACE agent that may still have stale directory content. Note that ACE CPUs will often leave stale directory content.

10.9.3 CCC ECC Disable Register

This control register can be used to disable ECC correction and detection, if the IP is configured to have ECC. If no ECC is present, the control register doesn't do anything. All other bits are unused.

10.9.4 CCC Directory Hash Bypass Register

This control register allows the directory to avoid index hashing. The directory is designed with two different indexing mechanisms. For way 0, the index bits specified in the configuration are used directly from the address. For way 1, additional address bits XORed with the index bits. This hashing is used to reduce the probability of conflicts within the directory.

The hash bypass register will skip the hashing that is performed for way 1. This can allow a more straightforward debug. It can also be used with the indirect access control registers to enable a more intuitive access of the RAM array.

A change of this bypass will change the lookup mechanism of the RAM. Since it changes the location where addresses can reside, this bypass should only be used when the directory is invalid or during debug operations. If changed during normal operation, coherency will likely be violated.

10.9.5 CCC Agent Disable Status Register (CCC_AGENT_DISABLE_STATUS)

This status register is used to indicate that a change to the active agent vector has taken effect. When an agent is removed from the active agent vector, snoops targeting that master may still be queued or inflight. While snoops are outstanding, the master must continue to operate to satisfy snoop requests.

This register indicates that all snoops to the disabled agents have completed and no new snoops will be issued to those agents. The status indicates whether any snoops outstanding at the time of the last modification of the ccc_active_vector register(s) have all completed or not. A value of 1 indicates that the snoops have all completed and it is safe to power off or otherwise disable the masters that were removed from the active vector. A value of 0 indicates that snoops are still outstanding and the master must continue to operate and accept snoops.

10.9.6 CCC LLC Control Register (CCC_LLC_CONTROL)

When the CCC is connected to a Last Level Cache, it has some unique programmable features that allows the two to interact more closely. This register allows programmable control of these features.

The CM bit is the enable for the Cache Maintenance Operation propagation. A Cache Maintenance Operation flushes or invalidates a cache line from the coherency domain. This can allow interaction with non-coherent devices that access the slave directly. Since a Last Level Cache can hold additional data, it may be necessary to flush or invalidate lines from the LLC to memory. When this register bit is set, the CCC will propagate any Cache Maintenance Operations to the LLC, where the cache can take the specified action. When this bit is set to zero, the Cache Maintenance Operations will not be propagated.

The WE bit is the enable for Write Evict propagation. In ACE protocol rev E, the WriteEvict command was created in order to write clean data to a downstream cache, such as the Last Level Cache. This allows the cache to only allocate a line when it is dropped by one of the ACE masters, allowing a better utilization of RAM storage. When the WE bit is set to 1, the WriteEvict will be propagated from the CCC to the LLC. If set to 0, the WriteEvict will drop the data and only update the CCC directory to indicate that the master has given up its copy of the line.

10.9.7 CCC Directory Invalidation Control/Status Register (CCC_DIRECTORY_INV)

This is a control register that can be used to invalidate the entire directory. Setting this register will kick off a hardware engine that will block normal coherent traffic and invalidate all entries of the directory.

The register also acts as a status register. When the control bit is set, it triggers the hardware state machine. The value of that register will stay high until the state machine completes. At that point, it will automatically transition to 0. Since coherent traffic will be blocked until the invalidation sequence has completed, it is not always necessary to check the status of this register.

Writing a value of 1 will trigger the invalidation engine, and it will transition to 0 when completed. All other bits are unused, and the default value is 0.

10.9.8 Directory RAM Indirect Access Control Register (CCC_INDIRECT_ACCESS_TRIG)

This registers is the indirect access trigger. Indirect access is a mechanism that allows register-based access to the directory RAM. This can be used for testing RAM bits or reading content on an error condition.

The indirect access is based on a content+trigger mechanism. For writes, the content register is written first to accumulate the data that should be written. Once the content is ready, the trigger register is used to kick off the hardware write mechanism. For reads, the trigger register kicks off a read, and provides data by placing the result into the content registers where it can be accessed.

10.9.9 CCC Speculative Fetch Vector Register (CCC_INDIRECT_RAM_CONT)

This is the indirect access RAM content register. Its use is conjunction with the indirect access trigger register. On an indirect read, data is written to this register. On an indirect write, content from this register is written into the RAM. On a read-modify-write, content from this register is used for the XOR function.

10.9.10 CCC Event Counter Mask Control Register

This register is used to program the event counter. Each bit of this register enables the performance counter to increment if the event occurs. A value of 1 for a bit indicates that this event should be counted.

10.9.11 CCC Event Counter Count Value Register

This register holds the current event count. As selected events occur, the count will increase. When this register rolls over, it can generate an interrupt if the interrupt mask is set correctly.

10.9.12 CCC Coherent Request Tracker Status Register (CCC_CRT_STATUS)

This register tracks the current state of the Coherency Request Trackers. These are the state-machines that track coherent request from the time they are started to the time they are completed. This can be used to determine if there are requests that have made it to the CCC but are not done being processed.

10.9.13 ECC Error Information Register (CCC_ECC_INFO)

This register monitors ECC errors and saves information for potential debug support. The register holds two kinds of information. It keeps a count of all ECC errors that have been detected in a 16 bit ECC counter.

10.9.14 CCC Interrupt Mask Register

This register is used for determining what kind of events can trigger an interrupt from the CCC. The three events it can control are:

- 0: Multi-bit ECC error
- 1: Single-bit ECC Error
- 2: Event Counter Overflow

10.9.15 CCC Interrupt Status Register

This interrupt is a status register that tracks the interrupt generating events. This includes multi-bit ECC error, single-bit ECC error, and event counter overflow. When these events occur, this register is updated and will hold the bit value until cleared. It can be cleared by writing to the register. To allow per-bit clearing control, the write value should use a value of 1 when it doesn't want to make a change, or a value of 0 when it wants to clear.

10.10 LLC Host Registers

10.10.1 LLC Class Allocate Registers (LLC_CLASS_ALLOC)

The class allocation control registers are used to specify which way groups (a group of 4 associative ways) can be written to. Each master in the system belongs to an LLC class, and each class allocation control register indicates which way groups that class of agents can allocate into.

These registers can be used to provide dedicated associativity for different agents or groups of agents. The default value of these registers indicates that all ways are accessible by all agents, with a value of one indicating allocation is allowed.Setting the value to zero will disable allocation for an agent.

It is permissible to turn off allocation for all ways, which will prevent any accesses from that class from allocating into the cache.

Note that the llc_global_alloc register can override these values. If global allocation is disabled for a way group, none of the agents can allocate into those ways regardless of what the llc_class_allocate registers indicate.

10.10.2 LLC Global Allocate Register (LLC_GLOBAL_ALLOC)

This register controls whether lines can be allocated into a way group by any agent.

If a way group is disabled from allocation in this register, no agents can allocate even if the llc_class_allocate registers are set. This register is used as part of a sequence to remove ways from use by the cache for either Scratchpad RAM usage, or for power gating. By removing allocation ability, a flush engine can remove the existing contents of the line without fear that new entries will be added during or after the flush.

10.10.3 LLC Cache Way Enable Register (LLC_CACHE_WAY_ENABLE)

This register indicates whether a way group is enabled for cache access. If enabled, a cache lookup will read the associated Tag values and perform an address comparison. If disabled, the Tags won't be accessed and the contents of the Tags won't be compared.

10.10.4 LLC Scratchpad Ram Way Group Enable Register (LLC_RAM_WAY_ENABLE)

This register is used to enable cache ways to be used as a Scratchpad RAM instead of a cache. The register indicates which of the way groups should be used as a RAM instead of a cache. It is possible to use some of the LLC as a cache, and some as a RAM, by selecting which way groups are used by each. By default, this register is set to 0 so that all ways are used as cache. To set this register, the lines must be removed from cache usage by the llc_cache_way_enable register. Any cache contents should be flushed before enabling the RAM mode.

10.10.5 LLC Scratchpad Ram Way Group Security Register (LLC_RAM_WAY_SECURE)

This register allows the Scratchpad RAM to have trust-zone security checking. Each way group can be individually controlled. If the security bit is set, only secure accesses (those with AxPROT[1] set to secure) can access that address. Non-secure accesses will be responded to with an error, and an interrupt will be triggered if the interrupt is enabled.

10.10.6 LLC Scratchpad RAM System Address Base Register (LLC_RAM_ADDRESS_BASE)

This register indicates the system address offset of the RAM mode. The address range should always be allocated as the full size of the LLC capacity rounded up to a power of 2, and the address offset must be programmed to a naturally aligned address for that size. The default value of this register is the address range base specified during NoC construction.

10.10.7 LLC Tag Invalidation Control Register (LLC_TAG_INV_CTL)

The Tag Invalidation Control register triggers a state machine that will invalidate the contents of one or more way groups of the cache. The register has one bit per way group, and the bit vector written into this register will invalidate the corresponding ways. A value of 1 will indicate that the corresponding way group should be invalidated. A value of 0 will indicate that the way group should not be invalidated. This per-way control allows portions of the cache to be powered down and restarted later, with the ability to reset just the ways that were powered down and powered back on.

A write to the register will kick off the invalidation engine, invalidating the specified ways. A read of the register will indicate whether the invalidation is in progress. When the invalidation engine has completed, the bit vector will transition to a value of zero. So a read value of zero will indicate that the state machine has completed. The reset value of this register is zero. An invalidation sequence should be completed before a second sequence is requested.

10.10.8 LLC Data Bank Invalidation Control Register (LLC_DATA_INV_CTL)

This register controls a state machine that can invalidate the contents of data array banks. Each way group has a corresponding bit in this register. When the register is written, the invalidation engine will kick off and invalidate the data for each of the specified way groups. Writing a value of 1 to a bit indicates that the corresponding way group should be invalidated. Writing a value of 0 to a bit indicates that the content of that way group shouldn't be invalidated.

The register can be read to determine the current status of the data bank invalidation sequence. When hardware has completed the invalidation sequence for a way group, it will change the value of that register bit from 1 to 0. If the entire register has a value of 0, then the invalidation engine has completed. The reset value of this register is zero.

Invalidation of the data array is needed when switching between cache mode and scratchpad RAM mode, since the RAM mode allows direct access to the data. Any secure data that was stored in the cache may be visible to RAM mode accesses unless it is invalidated first. Similarly, if security permissions are removed for the Scratchpad RAM, the prior contents should be invalidated before removing the security check.

An invalidation sequence should be completed before a second sequence is requested.

10.10.9 LLC Way Flush Control Register (LLC_WAY_FLUSH)

This register controls a state machine that flushes specified way groups of the cache. The intent of this engine is to remove all content from the specified ways, pushing any dirty data that may exist to memory. It also invalidate clean lines. The Way Flush engine should be run while the llc_cache_way_enable is still on for those ways so the contents are still accessible, but the llc_global_alloc register should have disabled the way for allocation. This ensure that as lines are removed from the cache, they won't unintentionally get added again. Clean lines are invalidated to ensure that dirty line writes do not write into the way groups being flushed.

Writing the register will kick off the flush engine. If the write value specifies a bit value of 1, then that way group will be flushed. If the bit value written is zero, that way will not be flushed. When the sequence is completed, hardware will transition the bit values to zero. A register value of 0 indicates the state machine has complete. The default value for this register is zero.

A flush sequence should be completed before a second sequence is requested.

10.10.10 LLC ECC Disable Control Register (LLC_ECC_DISABLE)

This register allows ECC to be disabled for either the Data arrays or the Tag arrays. These are independently controlled. A bit value of 1 indicates that ECC is disabled. A bit value of 0 indicates ECC is enabled, if present. The register value resets to value 0, meaning ECC is enabled.

10.10.11 LLC ECC Tag Error Status Register (LLC_ECC_TAG_INFO)

This is a status register that tracks ECC errors that occur in the Tag array. The register will track the number of ECC errors, as well as whether single-bit or double-bit errors have been detected. If the SB bit is set, at least one single bit error has been detected. If the DB bit is set, at least one double-bit error has been detected.

Additionally, the register tracks information about the first error detected. It stores the index of the tag array that had the error, as well as the way group. If a double-bit error occurs after a single-bit error has already been recorded, the double-bit error will overwrite the content of the register. This is because double-bit errors are fatal, and the information about how a fatal error is more important that the information about a non-fatal error.

10.10.12 LLC ECC Data Error Status Register (LLC_ECC_DATA_INFO)

This is a status register that tracks ECC errors that occur in the Data array. The register will track the number of ECC errors, as well as whether single-bit or double-bit errors have been detected. If the SB bit is set, at least one single bit error has been detected. If the DB bit is set, at least one double-bit error has been detected.

Additionally, the register tracks information about the first error detected. It stores the index of the tag array that had the error, as well as the way group. It also tracks which half of the cache line failed, which is needed to identify the sub-bank that failed. If a double-bit error occurs after a single-bit error has already been recorded, the double-bit error will overwrite the content of the register. This is because double-bit errors are fatal, and the information about how a fatal error is more important that the information about a non-fatal error.

The register can be read for status, but can also be written. If the SB and DB bit are written with zeros, the sampling of the first detected error will happen as described above.

10.10.13 LLC RAM Content Register (LLC_INDIRECT_RAM_CONT)

This is the indirect access RAM content register. It is used in conjunction with the indirect access trigger register. On an indirect read, data is written to this register. On an indirect write, content from this register is written into the RAM. On a read-modify-write, content from this register is used for the XOR function.

Since the RAM data width may be larger than 64 bits, multiple registers are used to hold the data. Any bits beyond the data width are unused.

10.10.14 LLC Indirect RAM Access Trigger Register (LLC_INDIRECT_TRIGGER)

This register is the indirect access trigger. Indirect access is a mechanism that allows register-based access to the RAM arrays. This can be used for testing RAM bits or reading content on an error condition. The indirect access is based on a content+trigger mechanism. For writes, the content register is written first to accumulate the data that should be written. Once the content is ready, the trigger register is used to kick off the hardware write mechanism. For reads, the trigger register kicks off a read, and provides data by placing the result into the content registers where it can be accessed.

10.10.15 LLC Event Counter Value Register (LLC_EVENT_COUNTER)

This register is an event counter. When the event counter mask control enables certain events to be counted, they will increment this counter. When the counter overflows, in can produce an interrupt if the interrupt mask is enabled. This can be used to trap to software after a number of specified events has occurred. The counter can be read or written. Writing the value can

initialize the counter to a larger value which can speed up the point at which counter will overflow and the interrupt will be triggered.

10.10.16 LLC Event Counter Mask Control Register (LLC_EVENT_COUNTER_MASK)

This register is used to program the event counter. Each bit of this register enables the performance counter to increment if the event occurs. A value of 1 for a bit indicates that this event should be counted. If multiple bits are set to 1, the logic will only count if all of the corresponding events occur on the same cycle. This allows for combinations of events, such as a cache miss that causes an eviction. The events that can be counted are all related to cache accesses and occur in the same cycle of the pipeline, so they can be combined easily.

When an event satisfies all of the requirements, the llc_event_counter register will be updated. A value of 1 indicates that the event is selected for counting. A value of 0 the event is not selected. By default, this register will be set to 0 for all bits, indicating no event counting should occur.

10.10.17 LLC Interrupt Mask Register (LLC_EVENT_INTERRUPT_MASK)

This register is used for determining what kind of events can trigger an interrupt from the LLC.

10.10.18 LLC Interrupt Status Register (LLC_EVENT_INTERRUPT_ERR)

This is a status register that tracks the interrupt generating events. This includes multi-bit ECC error, single-bit ECC error, RAM mode disallowed accesses, and event counter overflow. When these events occur, this register is updated and will hold the bit value until cleared. It can be cleared by writing to the register. To allow per-bit clearing control, the write value should use a value of 1 when it doesn't want to make a change, or a bit value of 0 when it wants to clear.

Appendix A: AMBA Protocol Support

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

10.11 ACE / AXI4 Feature Adoption

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

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

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

10.12 AMBA Signal Adoption

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

10.12.1 Global AXI Signals

Signal	ACE
ACLK	AXI port clock
ARESETn	Active low reset

10.12.2 Write Address Channel Signals

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

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

10.12.3 Write Data Channel Signals

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

10.12.4 Write Response Channel Signals

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

10.12.5 Read Address Channel Signals

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

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

10.12.6 Read Data Channel Signals

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

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

10.12.7 Snoop Address Channel Signals

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

10.12.8 Snoop Response Channel Signals

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

10.12.9 Snoop Data Channel Signals

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

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

10.12.10 Acknowledge Channel Signals

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

10.13 AXI4-Lite Feature Adoption

AXI4 Lite Feature	NetSpeed AXI NoC Support
Ports	AXI4-Lite master and slave ports can connect to NetSpeed AXI NoC. Intercommunication between AXI4 agents and AXI4-Lite agents is

	supported, conversions are performed by the NoC
Conversion	INCR transactions of length > 1 from AXI masters to AXI4-Lite slaves, are split into multiple AXI4-Lite transactions of 32-bit or 64-bit size. Responses from the slave are converted back to the format expected by the AXI master. WRAP requests sent to an AXI4-Lite slave will result in a SLVERR response
Transfer Length	AXI4-Lite is restricted to single beat transactions
Data Width	32-bit and 64-bit AXI4-Lite interfaces are supported
Address Width	Support range of address width is 14-bit to 60-bits
Transfer size	AXI4-Lite does not support narrow transfers and all transactions are of full data width
AID	No ID bits are present in AXI4-Lite. Multiple outstanding transactions are supported, however all transactions must be ordered.

10.14 AXI3 Feature Adoption

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

10.15 AHB-Lite Feature Adoption

AHB-Lite Feature	NetSpeed AXI NoC Support
Version	AHB-Lite master and AHB-Lite slave devices can connect to NetSpeed AXI NoC.
Data Width	32, 64, 128 bits
Address Width	Range of 14 – 60 bits
HSIZE	8, 16, 32, 64, 128-bit transfer size
HMASTLOCK	Locked transfers are not supported. HMASTLOCK will be ignored.
HBURST	All standard specified modes supported
HSELx	Up to 16 AHB-Lite slaves can connect to an AHB-Lite slave bridge of the NoC. Slaves on a given slave bridge can have different address and data width but common endian format

Response delay	On AHB-Lite master, bufferable transactions will be provided early response and non-buffer transactions will receive response from the end point.
Endian format	Each AHB-Lite interface on NetSpeed NoC can be configured to handle little endian or big endian format
Write strobes	AHB-Lite interface does not support write strobes. However, write transactions from AXI masters to AHB-Lite slaves can use partial write strobes. Empty write strobes results in no AHB access and OK response.
Address alignment	AXI masters can perform write accesses to AHB slaves with any address alignment. Read transactions from AXI masters with unaligned addresses are changed to aligned addresses. Read is performed to the aligned address and response can optionally be marked with SLVERR based on a programmable register bit.

10.16 APB Feature Adoption

APB Feature	NetSpeed AXI NoC Support
Version	APB 2/3/4 slaves can be connected to NetSpeed AXI NoC
Conversion	INCR transactions from AXI masters are broken down into multiple single beat APB transactions at APB slave bridge. Responses from the slaves are converted back to the format expected by the AXI master. WRAP requests sent to an APB slave will result in a SLVERR response
Data Width	32-bit APB slave devices are supported
Address Width	Address is fixed at 32-bit
PSELx	Up to 16 APB slave devices can be connected to single APB slave bridge. Each APB slave is identified by a REGION associated with its address range
PREADY	Supported on APB 3, 4 slaves to allow slave to extend an APB transfer
PPROT	Supported on APB 4 slaves
PSLVERR	Supports slave error from APB 3, 4 slave devices and remaps appropriately to response sent back to master
Address alignment	All read transaction address from AXI masters to APB slaves must be aligned to 32-bits Narrow read or write transactions sent to APB slave are modified to be full prior to conversion to APB, i.e. any AxSIZE from masters is changed to 2 Write transaction address to APB 2, 3 slaves must be aligned to 32-bit (unaligned access results in SLVERR response). APB 4 slaves can be sent unaligned write transaction addresses.
PSTRB	Supported on APB 4 slaves to allow partial byte updates during write transfers. Write transactions from AXI masters to APB 2, 3 slaves must not use partial

	write strobe. Partial write strobes result in SLVERR response? Empty write strobes results in no access and OK response.

2670 Seely Ave

Building 11

San Jose, CA 95134

(408) 914-6962

http://www.netspeedsystems.com