# Functional Description: Non-Coherent Components

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

## Bridging from Host to NoC

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

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

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

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

### AXI4 Master Bridge

Figure 12 shows a block diagram of the CFG 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 1. AXI4 Master Bridge

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

#### 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.

#### 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.

#### 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.

#### 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.

#### 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.

### AXI4 Slave Bridge

Figure 13 shows a block diagram of the CFG 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 2. AXI4 Slave Bridge

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

#### 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

#### 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.

#### 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.

### 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.

### AXI4-Lite Slave Bridge

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

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

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



Figure 3. AXI to AXI4-Lite Bridge

### APB Bridge

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

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



Figure 4. 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.

### AHB-Lite Master Bridge

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

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

* AHB-Lite mirrored master interface.
* Data widths of 32-bit, 64-bit, and 128-bit. Address widths of 32-bit and 64-bit.
* Writes are bufferable or non-bufferable depending on HPROT[2].
* HPROT[2] can be overridden to force non-bufferable write behavior.
* Single read outstanding.
* Configurable number of outstanding bufferable writes.
* 1KB address boundaries per AHB transaction, and the NoC splits transactions into 64-byte chunks.



Figure 5. AHB-Lite to AXI Master Bridge

### AHB-Lite Slave Bridge

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

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



Figure 6. AXI to AHB-Lite Bridge

### Shared interface bridge

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

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

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

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



Figure 7 Shared Interface Bridge

### Reorder Bridge

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



Figure 8: Reorder Bridge NoC Component

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

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

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

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

#### Reorder bridge benefits

##### Reducing Reorder Buffer Entries

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

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

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

##### Reducing AID table storage

The AID table has three major functions:

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

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

##### Reducing Address Map Storage

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

#### Using the reorder bridge

##### Instantiating the Reorder bridge

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

##### Address Map and Security Limitations

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

#### Limitations and implications of using the reorder bridge

##### Unwanted Connectivity Security Hole

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

##### Security Filtering Differences

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

##### Power Management (Fence/Drain)

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

##### Timeout Handling

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

### Configurable Slave Block

#### Architecture

The Configurable Slave is an AXI agent that can be added to an AMBA NoC as a functionality supplement. This module can be configured at design time with different kinds of on-chip storage, such as SRAM or register files. This on-chip storage is wrapped with control logic allowing straightforward instantiation within an AMBA system.



Figure 9: Configurable slave System View

The Configurable slave looks like any other slave agent, connecting to the NoC through a slave bridge. Multiple configurable slave components can be added to the network to increase bandwidth or to have lower latency by moving storage closer to a particular source.



Figure 10: Configurable slave in Gemini System

The Configurable slave can also fit into a Gemini system either as a non-coherent slave, or as a coherent slave. A coherent slave means CCC will track addresses for this slave and handle cache coherency functions. The slave would then act as a backing store for the coherent range.

The configurable slave can be designed with multiple storage components for. It can instantiate SRAM arrays, Register Files, or Flip Flops, or a combination of them. Each storage component can be sized independently of the others. Multiple instances of an array can be used for banking, either to increase bandwidth or to limit the size of each array instance.



Figure 11: Configurable Storage

One of the storage types is SRAM. This is a single-ported SRAM. The size of each SRAM array must be power-of-2 number of array elements. Each array can have variable latency and bandwidth, depending on the implementation of the SRAM (these values are given as part of the specification). Since bandwidth may be a limit, an option is provided to allow a power-of-2 number of banks of SRAM, allowing parallel accesses for increased bandwidth. Banking will be based on the bus width of the slave. So, a 128bit data path will bank on 128bit granularity, so consecutive addresses will access different banks.

For smaller structures, or some non-power-of-2 size storage, cache line (64B) storage blocks can be added using either flops or Register Files.

Banking of arrays happens on the granularity of the data bus width and will use the next lowest order bits depending on the size of the banking.

##### Data Bus Width

The bus data width of the configurable slave is configurable. Larger widths can be used to increase bandwidth. The bus data width also defines the width of the logical arrays. So, an SRAM will have a data width matched to the bus width. Configurable slave can support data width of 4, 8, 16, 32, or 64 bytes.

##### Address Bus Width

The address bus width of the configurable slave is also configurable. This can allow address bit truncation to reduce area and to avoid needing to manipulate the address before sending to the slave.

##### Exclusive Monitors Option

The configurable slave can optionally have AXI Exclusive monitors. This option will default to off, so less hardware will be instantiated. The user must configure this option in NocStudio to enable it.

The exclusive monitors will be set up based on NocStudio definitions of the agents, which means it can fully populate as many Exclusive monitors as needed by the system, including those needed for logical agents (i.e., sub-agents).

The Exclusive monitors are set up during an Exclusive ReadNoSnoop command (ARLOCK==1). A store by another agent to the same 64B address will reset the validity of the monitor. An Exclusive WriteNoSnoop (ARLOCK==1) will perform a conditional store. If the monitor has been cleared before the conditional store is processed, the store will be canceled.

The Exclusive Monitors are designed to track 64B of data. If different semaphores are located within the same 64B block of data, any successful modification of the line will clear the monitors tracking that address.

##### Bandwidth

The configurable slave is designed to break up bursts into single beat operations and issue them in order to the storage modules. The AR and AW/W interfaces provide a very small amount of storage to allow request to arrive, arbitrate, and send requests to the storage modules with the beat-specific address.



Figure 12: Logical view of arbitration

As shown in the diagram above, reads and writes arbitrate together to send a single beat-sized request to the storage components. This means that the configurable slave shared bandwidth between reads and writes. A 32B interface will only support 32B per cycle of bandwidth, shared between reads and writes.

When a read or write wins the arbitration, it will continue until it has completed. The configurable slave does not interleave the requests, to avoid unexpected behavior where a read to the same address as a write returns partially new and partially old data.

##### Clock Domain

The configurable slave runs at a single frequency for all internal components. There is no clock crossing per individual storage entry. The slave is expected to run at NoC frequencies, but like any slave on the NoC, it can be configured to run at a different clock with clock crossing within the NoC.

##### ECC Protection

The Configurable Slave can support ECC protection on each storage bank. The ECC will protect data based on the data width of the slave. So, a 32B bus data width will have ECC protection of 32B, which can use 10 bits. Smaller bus width will end up with more ECC storage bits for the same total capacity.

Table 1: ECC Overhead based on bus width

|  |  |  |
| --- | --- | --- |
| Data Width | ECC Bits per Data Width | ECC percentage per 64B |
| 512 Bits | 11 | 2.1% |
| 256 Bits | 10 | 3.9% |
| 128 Bits | 9 | 7.0% |
| 64 Bits | 8 | 12.5% |
| 32 Bits | 7 | 21.9% |

The ECC protection forces a functional change as well. If partial data writes are sent to the configurable slave storage with ECC protection, the Configurable Slave will perform a read-modify-write in order to set the ECC bits correctly. This can reduce the bandwidth of a storage array, so ECC enabled storage should have additional banking. If partial writes are rare, additional banking may be unnecessary.

The ECC algorithm is SEC/DED (single-bit error corrects, double-bit error detect).

##### Storage Element

The following values must be specified for each storage module:

Table 2: Storage component parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | SRAM | Register File | Flops | Description |
| # of banks | Y | Y | Y | Default 1. Can be used to have multiple banks |
| Total Capacity | Y | Y | Y | All banks must be power of 2 number of entries. |
| ECC Protection | Y | Y | Y | Enables ECC protection. Will increase storage to hold ECC bits. |
| Read Latency | Y | Y | Y | For SRAM, this latency includes RAM latency, and any other cycles needed for transportation to and from array. For flops and RF, this only includes the transportation latency. |
| Bandwidth | Y | - | - | For SRAM, this indicates the throughput |
| mem\_info\_in | Y | Y | - | This parameter allows a set of input signals to be routed to the array through the top-level module. These are typically used for BIST or other DFT signals. |
| mem\_info\_out | Y | Y | - | This parameter allows a set of output signals to be routed to the array through the top-level module. These are typically used for BIST or other DFT signals. |
| Address range | Y | Y | Y | Specify address ranges for these slaves |

##### CSR -- Control and Status Registers

Please refer to Programmers’ Model chapter for register definition. For design specific register set, please refer to the noc\_reference\_manual.html and noc\_regsiters.csv generated from NocStudio.

#### Micro-Architecture

This section describes the configurable slave microarchitecture.

##### AR Channel

A small amount of logic sits in the AR channel to handle flow control and deal with address modification for burst requests.



Figure 13: AR Channel

As requests come in on the AR channel, a small skid buffer accepts the requests from the interface. If the skid buffer is full, the ARREADY will backpressure the NoC.

For a burst read, the initial address of the request must be modified for the other beats. Logic must support WRAP and INCR type of transfers. It should also handle narrows, for future functionality. To track the burst values, a beat count is needed.

##### AW/W Channel

Like the AR channel, the AW+W channel has an interface module that deals with flow control and burst requests.



Figure 14: AW Channel

A skid buffer is need for AW and W channels. Flow control to the AWREADY and WREADY signals are dependent on the skid buffers being empty.

Unlike reads, a request to the arbiter is conditional on W channel having available data.

Like AR channel, the AW channel can support WRAP, INCR, and narrow requests. The address needs to be modified based on the current beat count.

##### R Channel

The read response channel has two FIFOs. The RID is written to immediately on the issuing of a read beat, and saves the MSTID, RID, and RLAST for that beat. The Read Response FIFO stores the read



Figure 15: R Channel

The read response FIFO entries must be pre-allocated for each beat of a read. There is a bypass path around the read response FIFO to allow for a reduced latency response, when possible. The bypass only happens with read response FIFO is empty.

##### B Channel

The Write Response channel accepts writes responses from the components or decode error module and issues the response onto the interface.



Figure 16: B Channel Logic

As responses come in, there is a merging function for write responses. Each beat of a burst will get a response, but only the last beat will send the write response. During that window, if any errors are detected, the errors will be combined in the response and returned with the write response.

To issue the first write, a write response buffer must be pre-allocated. Additional writes for a burst do no allocate new entries. The last write must be indicated to the write response channel control to indicate when it can send.

##### Arbiter

As requests come in from AR or AW, they arbitrate to access the storage modules. The arbitration is a simple round-robin arbitration, with a modification to stick with a source until the request has completed.



Figure 17: Arbiter

If a write is selected, the entire write will complete before the next read will be selected. The same is true for read bursts. This ensure multi-copy atomicity of the requests.

##### Lookup Pipeline

The pipeline of the lookup varies depending on the type of access.



Figure 18: Access Pipeline

As requests come into the pipeline, they arbitrate and mux. The winning request performs a lookup of the AXI Exclusive monitors, as well as the Address Map. The Address Map is used to decide which storage structure to access.

For AXI Exclusive reads, the access to the monitors just marks the corresponding monitor as valid.

For all writes, the write request has to compare against all of the monitors. If an address match occurs, and the write is successful, the other agent’s monitors will be reset. If the write is an Exclusive store, it will conditionally execute based on the status of its own monitor.

##### Configurability

The highly configurable part of this agent is the ability to configure it to have any number of the available components, with any of the configurable sizes. Additional components could be made available in future revisions.

Components are added to the slave via NocStudio. Multiple components of the same type can be added, allowing useful improvements. A RAM might not have sufficient bandwidth on its own, but two RAMs can be added with interleaving of them based on some low order bit (often interleaving would be done on the bus data width granularity, so a 16B bus would have 16B banks, allowing consecutive accesses to spread across the data banks. Generally, bandwidth can be increased via banking.

Multiple banks can also allow for non-power-of-2 sized arrays. A 96KB RAM can be constructed with a 64KB RAM and a 32KB RAM.

###### Address Map

An address map is used during the access pipeline to determine which component is being accessed. If no component is targeted, a decode error module will be selected, and a response will be generated from there.

###### Response Latency

As requests run through the access pipeline, they will be sent to the various components. In order to allow this parallelism, while still maintaining response order, each component will ensure that responses are returned in a fixed amount of time so that they always return in order. A parameter will specify how long the longest latency is, and each component will match that latency.

Write requests to RAMs or RFs may actually take longer to complete, because they might have to do read-modify-write sequences. This means the component may stay busy long after the response has returned.

###### Security Check

Each component will be assigned a security requirement (secure, non-secure, or either). This will allow regions to be security controlled. If an access to a range doesn’t match the security requirement, it’ll get a decode error and not access the component.

##### Decode Error

When a request arrives to an unmapped region, the request will be processed by a decode error module instead of one of the components.

The decode error module will response to reads and writes as if it is a component but will mark the response as a decode error. Like components, the response latency must match the other components.

#### Combined Bridge Optimization

The initial version of the Configurable slave uses an AXI port, allowing it maximum flexibility in how it connects into a system. This means that to connect to the network, it will usually talk through a slave bridge.



Figure 19: Configurable slave talks to Slave Bridge

There is additional overhead for this kind of solution. The slave bridge has to send requests to the slave in AXI format, which is not very useful in a system which processes one request at a time in order. The slave bridge also tracks IDs to the slave, to allow for responses to return out of order. And all of this causes additional latency for reads and writes to the slave.

An alternative is directly adding the slave functionality into a combined Slave + Slave Bridge module.



Figure 20: Combined Slave and Bridge

This will have lower latency, and smaller area. This will be an optional feature when it is available, allowing a choice between the combined slave+bridge, or the two.

## Address Maps and Configurability Options

### 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.

### 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 21: 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 22: 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.

### 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 23: 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. From 1904 release, a slave can be multiple targets of the same hash function. For example, a two-bit hash decodes to 4 targets and we could map 3 of those targets to be the same slave. 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.

## AID Handling and transaction ordering

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

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

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

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

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

## 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.

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

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

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

## Width Conversion

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

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

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

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

AxCACHE[1] marks transactions as modifiable or non-modifiable. Modifiable transactions provide greater flexibility in CFG 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.

## Virtual slave interface

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

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

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

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

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



Figure 24: 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 and W channels are shown below



Figure 25 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.

## Interrupt

The NoC can produce one or more interrupt signals, using the mesh\_prop interrupt\_mode. NocStudio defaults it to “single”. The interrupt(s) can be triggered on various error conditions, including ECC error, decode errors, or illegal commands. Each NOC element internally can assert its own interrupts. As shown in Figure 26, there may be up to 4 interrupt output coming out the master bridge (1) non-fatal interrupt (2) fatal interrupt (3) Combined fatal-non-fatal interrupt and (4) security violation interrupt. During NOC design, SOC architect and system security architect decide how interrupt handler manages the interrupts when asserted, then configure the NOC to merge all interrupts or expose individual interrupt wire to the top level and user can deal with at SOC level.

Status

Mask

Fatal interrupt

Non-Fatal interrupt

Security mask

Security status

Security violation interrupt

Single interrupt

Non-secure registers

Secure registers

mesh\_prop security\_interrupt\_enable yes

Figure 26 master bridge interrupt signals

In single interrupt mode, the system software (interrupt handler) can read the status registers from every NOC element (bridge, router and agent) to decide the exact trigger of the interrupt event. Depending on the complexity of the NOC regbus layer and how fast the regbus layer is running at, the time elapsed may exceed the required response time for a given application. This is when user move from single merged interrupt mode to expose\_module mode. This eliminates the big OR-gate inside the NOC and SOC designer can design a priority decoder before feeding the wire to the interrupt controller.

In designs where functional safety requires, the NOC can be enabled with ECC / Parity at various levels. Please refer to the Safety and Reliability chapter for supported features. Depending on error condition, there could be parity error, single-bit correctable or double-bit non-correctable ECC errors, or plain simple address decoder or slave response errors, the NOC can be configured to separate the fatal and non-fatal interrupts for interrupt handler to better manage these error conditions.

In modern world SOC designs, security (or firewall violation) can be an extremely sensitive topic. When required user can set “mesh\_prop securiry\_interrupt\_enable yes” in the design, NocStudio would extract these security related status and mask bits and put them in a separated secure registers inside the master bridge as in CFG NOC architecture the address decoder and security checks in implemented inside the master bridge with the exception of the configurable slave block.

Figure 27 illustrates how the OR-gates are constructed depending on mesh\_prop setting in a design. The “interrupt\_mode” determines the wiring for general non-fatal and fatal interrupt while “security\_interrupt\_enable” expose separate interrupts for security violations.

hfn.v

Master bridge 0

Master bridge 1

Master bridge N

1. mesh\_prop interrupt\_mode single

(2) mesh\_prop interrupt\_mode expose\_module

(3) mesh\_prop interrupt\_mode fatal\_and\_non\_fatal

1. mesh\_prop security\_interrupt\_enable yes

(4) mesh\_prop interrupt\_mode expose\_module\_fatal\_and\_non\_fatal

1. mesh\_prop security\_interrupt\_enable yes

AND ( (2) OR (4) )

Figure 27 High Fan-out net module

## Restrictions

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