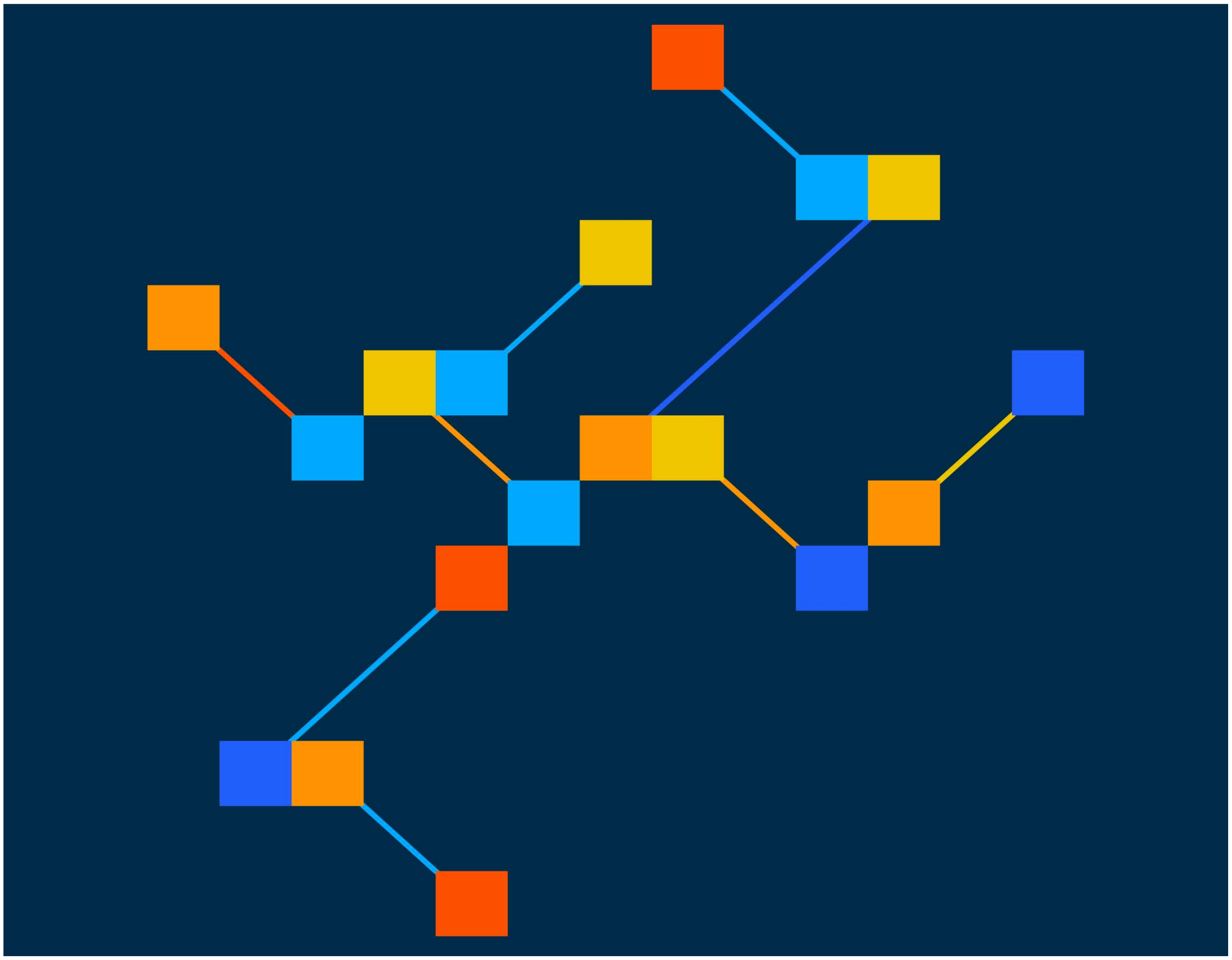
ARTERIS IP

Reference Manual
Ncore 3.6



arteris.com

Doc Name	Revision	Date	Description
100011_RM_3.6.4	1.0_A.1	Sep 2024	Release Update
100011_RM_3.6.3	1.0_A.1	Jun 2024	Release Update
100011_RM_3.6.2	1.0_A.1	Mar 2024	Release Update
100011_RM_3.6.1	1.0_A.1	Jan 2024	Release Update
100011_RM_3.6	1.0_A.1	Nov 2023	Initial Release

Contents

1 Ncore 3 Introduction

1.1 Terminology

The following terms have a specific meaning in this document.

Table 1. Terms and Abbreviations

Term	Definition
BIST	Built-In-Self Test
BR	Boot Region
CAIU	Coherent Agent Interface Unit
CDTI	Control and Data Transport Interconnect
CMD	Command Message
CSTI	Control and Status Transport Interconnect
ERG	Exclusive Reservation Granule
DCE	Distributed Coherence Engine
DII	Distributed IO Interface
DMI	Distributed Memory Interface
DTR	Data Read Message
DTW	Data Write Message
DVE	Distributed Virtual Memory Engine
DVM	Distributed Virtual Memory
GPAS	General Purpose Address Space
GRB	Global Register Block
MIG	Memory Interleave Group
MRD	Memory Read Message
NCAIU	Non-coherent Agent Interface Unit
NRS	Ncore Register Space
NoC	Network on Chip
PoS	Point of Serialization
RB	Request Buffer
RPN	Register Page Number

Table 1. Terms and Abbreviations

Term	Definition
SMC	System Memory Cache
SMI	System Management Interface
SNP	Snoop Message
SF	Snoop Filter
STR	State Reply Message
UPD	Update Message
VC	Virtual Channel

1.2 System Overview

A system based on the Ncore interconnect consists of a set of master agents and a set of slave agents that communicate through the Ncore interconnect. A master agent is an external logic block that initiates read and write transactions to a particular physical address. Typical examples of a master agent include processor clusters, graphics processing units, accelerators, or peripheral devices. A slave agent is an external logic block that services read and write transactions for a given set of physical addresses. Typical examples of a slave agent include memory or cache controllers or peripheral devices.

Master agent transactions are characterized based on their access semantics, which may be classified as either non-coherent or coherent. For a given physical address, a non-coherent transaction directly accesses the data in the corresponding memory location or peripheral device. Specifically, a non-coherent read transaction transfers data from a memory location, while a non-coherent write transaction transfers data to a memory location. Alternatively, for a given physical address, a coherent transaction may access the data either in the corresponding memory location or in one of the caches of the system. Specifically, a coherent read transaction transfers data values from the most recent copy, either in the memory location or in a cache, while a coherent write transaction transfers data to a memory location and invalidates any other cached copies.

A coherent transaction is further classified as a fully-coherent transaction or an IO-coherent transaction. In the former case, the master agent performing the coherent transaction obtains a copy that is required to remain coherent with respect to coherent transactions from other master agents. In the latter case, the master agent obtains a copy that is not required to remain coherent.

The same memory location may be accessed with transactions that have different access semantics; however, the Ncore interconnect does not enforce coherence in these cases and does not provide any ordering between transactions to the same memory location with different access semantics.

A master agent is classified by the transactions the agent is capable of issuing:

- A fully-coherent agent, e.g., a processor cluster may issue fully-coherent, IO-coherent, and non-coherent transactions.

- An IO-coherent agent e.g., a graphics processing unit may issue IO-coherent and non-coherent transactions.
- A non-coherent agent e.g., a typical DMA engine only issues non-coherent transactions.

Collectively, fully-coherent agents and IO-coherent agents are known as coherent agents. Furthermore, fully-coherent agents are also classified as caching agents, which incorporate caches that store coherent copies of data and implement snooping interfaces on which coherent operations are presented. IO-coherent and non-coherent agents may implement caches; however, these caches store only non-coherent copies of data. Data in non-coherent caches may be kept coherent with data in coherent caches using software coherence methods.

A slave agent is classified by how the agent services access that result from read and write transactions. A memory agent provides access to memory locations, and accesses to the physical addresses serviced by a memory agent have the following properties:

- A read access returns the last value written, and in particular, multiple read accesses to the same memory location return the same value unless an intervening write access has changed the value at the memory location
- Read and write accesses do not cause side-effects

The memory locations serviced by a memory agent may be accessed by a master agent using coherent or non-coherent transactions, and if those memory locations are accessed with fully-coherent transactions, copies of data from a memory agent can be cached and are kept coherent by the Ncore interconnect. To access a memory agent, the Ncore interconnect issues memory read transactions and memory write transactions, which perform read accesses and write accesses respectively.

A non-memory agent provides access to other forms of system functionality, and accesses to the physical addresses serviced by a non-memory agent have the following properties:

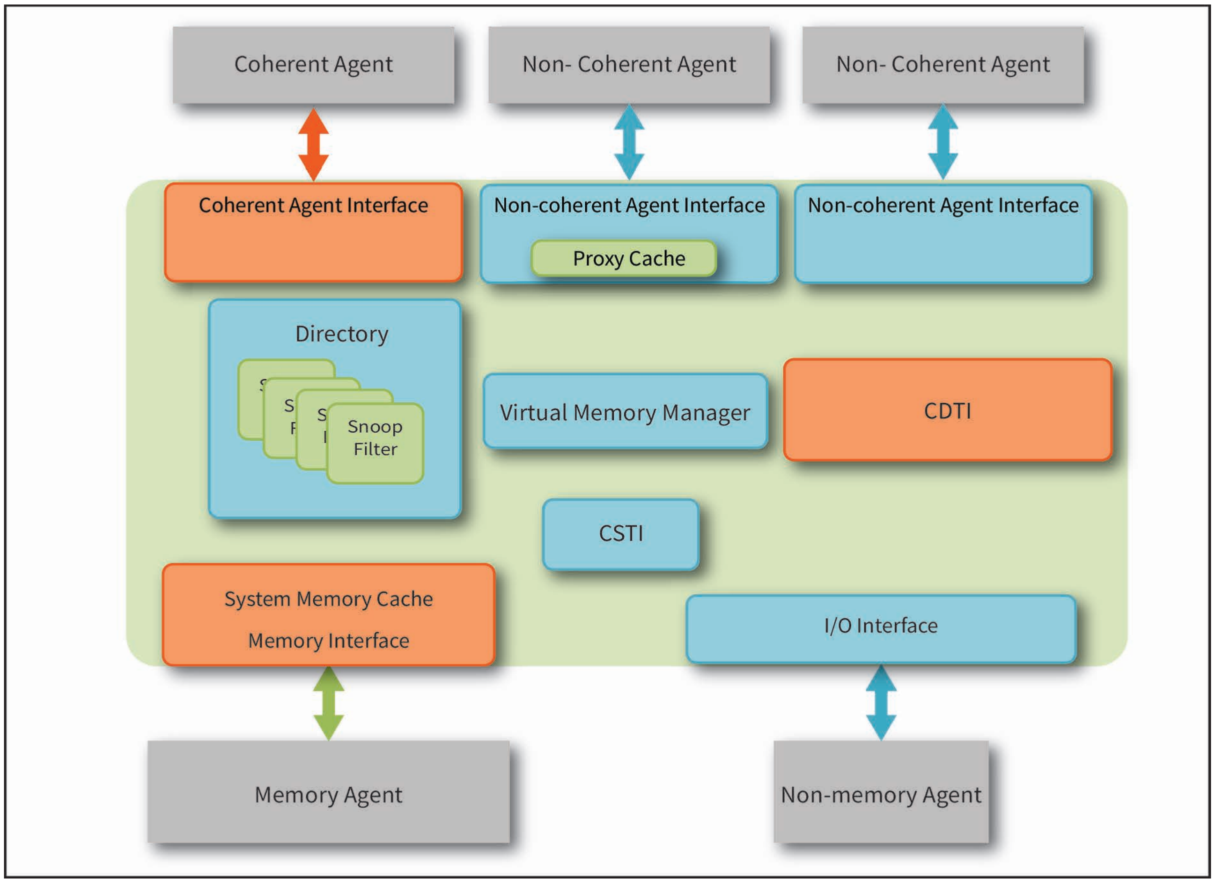
- A read access may not return the last value written, and multiple read accesses to the same memory location may return different values
- Read and write accesses may cause side-effects, e.g., raising an interrupt or initiating a hardware state machine

The system functionality provided by a non-memory agent may be accessed by a master agent using non-coherent transactions only. Copies of data from a non-memory agent are not cached and cannot be kept coherent.

1.2.1 Functional Overview

Figure 1 illustrates a set of functional components in the Ncore system. This functional view of an Ncore system describes the functional components and their relationship to each other. Agents communicate with an Ncore system address space through various interface components. A coherent agent accesses via a coherent agent interface, while a non-coherent agent accesses via a non-coherent agent interface. Correspondingly, a memory agent is accessed via a memory interface and non-memory agent is accessed via an IO interface.

Figure 1. Ncore 3 Functional Overview



A memory interface may be configured with a dedicated system memory cache, which caches data based on configurable policies. In a memory interface configured with a system memory cache, any transaction first accesses the system memory cache, and, depending on transaction type and the configuration of the system memory cache, the memory interface may service the transaction from the cache or perform a memory access. System memory caches may be configured to support additional features such as way partitioning, scratchpad, and atomic transactions.

The Directory enforces coherence on memory granules known as directory cache lines and serializes coherent transactions to the same directory cache line. The directory also implements one or more snoop filters that track the cache state of caching agents, and each caching agent is assigned to a single snoop filter. A caching agent may be a fully-coherent agent or an non-coherent agent interface configured with a proxy cache, described below. The type of coherent transaction and the result of the snoop filter access determines which coherent operations need to be performed and the number of snoops that will be issued.

A non-coherent agent interface may be configured with a fully-coherent proxy cache which caches data on behalf of the non-coherent agent. A transaction first accesses the proxy cache. On a proxy cache hit, the transaction is serviced using the proxy cache. On a proxy cache miss, the transactions result in either a fully-coherent or an IO-coherent transaction downstream. The decision is based on the original transaction attributes. If the data should be allocated in the proxy cache, a fully-coherent transaction is issued or else an IO-coherent transaction is issued.

A virtual memory manager handles all DVM traffic within the Ncore system and it also processes DVM commands.

Communication among components occurs via transport interconnect. A transport interconnect is a communication fabric that transports information from one component to another without altering the transmitted information or changing the semantics of the original transaction.

Ncore contains two transport interconnect structures characterized by the functionality that they provide.

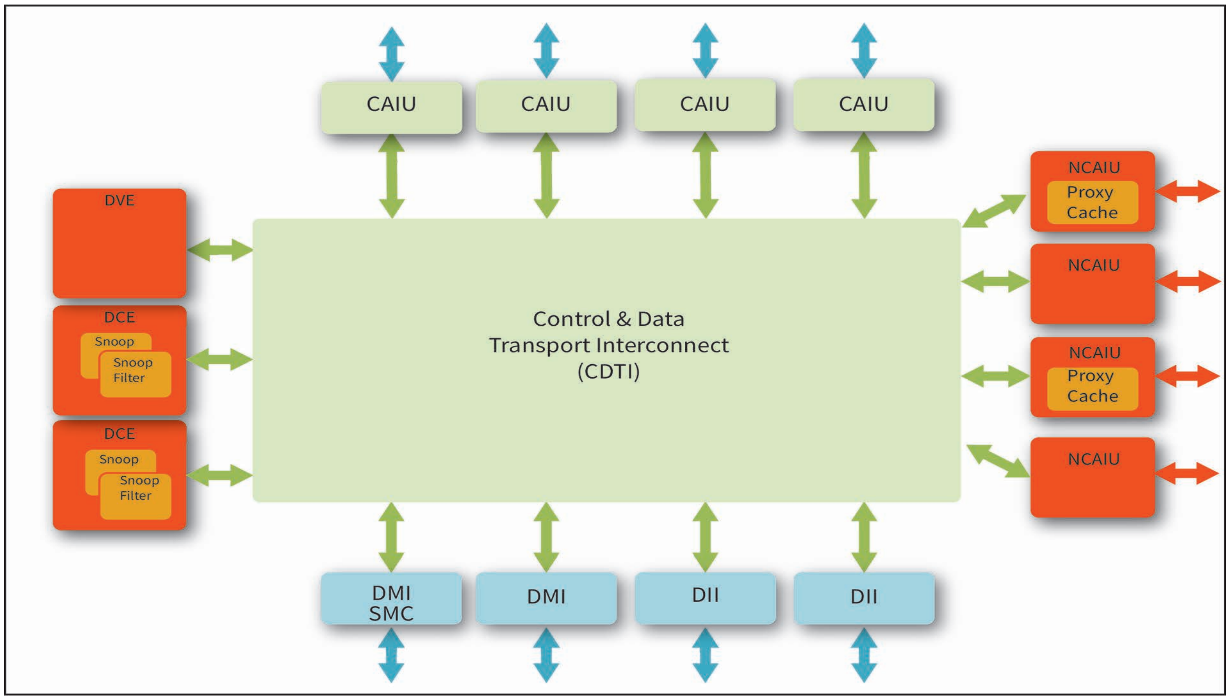
- Control and data transport interconnect (CDTI) is responsible for exchanging coherent and non-coherent control and data information between all components of the Ncore system.
- Control and status transport interconnect (CSTI) provides access to the control and status registers in the Ncore system.

1.2.2 Structural Overview

The Ncore functionality described in the previous section is represented by a set of structural units connected by the transport interconnects. The structural view of an Ncore interconnect describes these structural units and their connectivity illustrated in Figure 2. Each functional component has an analogous structural unit as follows:

- Coherent agent interface -> Coherent agent interface unit (CAIU)
- Non coherent agent interface -> Non-Coherent agent interface unit (NCAIU)
- Directory -> Distributed Coherence Engine (DCE)
- Memory Interface -> Distributed memory interface (DMI)
- IO Interface -> Distributed IO interface (DII)
- Virtual memory manager -> Distributed Virtual memory Engine (DVE)

Figure 2. Ncore 3 Structural View



Coherent Agent Interface Unit

A CAIU can be configured to implement one of three fully-coherent interface protocols, ACE, CHI-B, or CHI-E.

Non-Coherent Agent Interface Unit

An NCAIU can be configured to implement one of three non-coherent or IO-coherent interface protocols; AXI, ACE-Lite, and ACE5-Lite. The AXI interface can be configured through the GPRRS registers to access coherent or non-coherent address space. If an NCAIU instantiates a proxy cache, it must be configured to support coherent accesses.

Distributed Coherence Engine

A DCE implements directory structures to track the location of data. An Ncore system must be configured with at least one DCE. Depending on system transaction bandwidth multiple DCEs may be initiated. DCE instances may be physically distributed and logically interleaved - the address bits used for interleaving must be chosen during system configuration.

Distributed Memory Interface

A DMI provides an AXI master interface to normal memory for coherent and non-coherent transactions. Multiple DMI may be implemented. They can be distributed and may address interleaved within the Ncore system for bandwidth and physical placement. The number of instances and caching capabilities is chosen during configuration. Different interleaving options can be specified during configuration, and one of the specified options can be selected via registers at run time.

Distributed Virtual Memory Engine

The DVE is a virtual memory manager within the Ncore system. The current version of Ncore allows only one instance of DVE to manage DVM capable agents.

Distributed IO Interface

A DII provides an AXI master interface to IO devices for non-coherent requests.

2

Ncore 3 Protocol

2.1 Cache Coherency Protocol

Ncore components communicate and enforce cache coherence via the Arteris cache coherence protocol, which defines several protocol transactions, each of which accesses a single directory cache line, classified as follows:

- A coherent read transaction fetches a coherent copy of data for an agent and may or may not invalidate copies in other caching agents
- A coherent clean transaction transfers any modified data from a caching agent to memory and may or may not invalidate the copies in other caching agents
- A coherent write transaction transfers any modified data from a caching agent to memory, invalidating copies in other caching agents
- A memory update transaction transfers modified data from a caching agent to memory but does not invalidate the copies in the other caching agents

At a high level, each native transaction received by a CAIU or an NCAIU will be translated into one or more protocol transactions, based on the alignment and length of the original coherent transaction. Protocol transactions will complete in accordance with the requirements of the original native protocol, including sequencing responses from multiple transactions, if necessary.

Each transaction consists of one or more protocol messages, divided into protocol control messages and protocol data messages exchanged between pairs of Ncore components. For brevity the following section refers to CAIU and NCAIU as AIU. Protocol control messages are classified as follows:

- A command message (CMD) initiates a coherent or non-coherent transaction. CMDs flow between AIU and DCE/DMI/DII/DVE
- A snoop message (SNP) transmits coherence operations to caching agents and consists of a request and response phase. It flows between DCE/DVE and AIU
- A memory read message (MRD) performs a non-speculative memory read. It flows between DCE and DMI
- A request buffer message (RB) handles memory write buffer allocation. It flows between DCE and DMI
- A state reply message (STR) communicates transaction progress information to the requesting agent and consists of a request and response phase. It flows between DCE/DMI/DII and AIU

- An update message (UPD) communicates cache state information to a directory. It flows between AIU and DCE

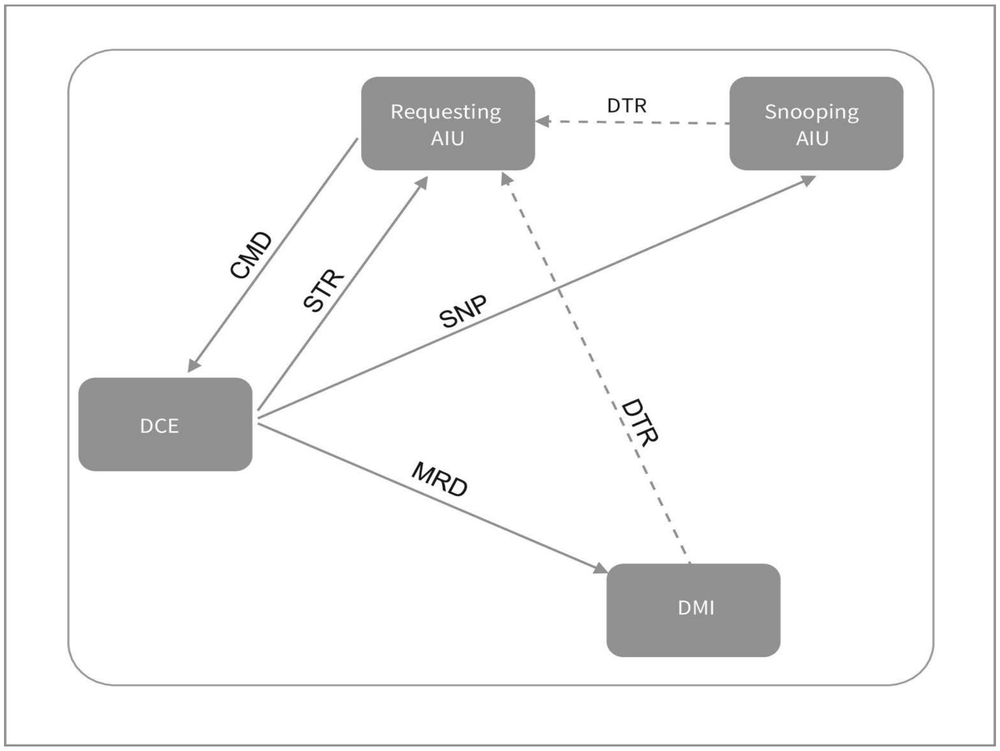
Protocol data messages are classified as follows:

- A data read message (DTR) transfers data to the requesting agent. It flows between AIU/DMI/DII and AIU
- A data write message (DTW) transfers data from the requesting agent to memory or device. It flows between AIU and DMI/DII/DVE

The sequence of protocol messages associated with a native transaction depends on the type of protocol transaction and the state of the snoop filters.

2.1.1 Coherent Read Request

Figure 3. Coherent Read Request



In a coherent read request:

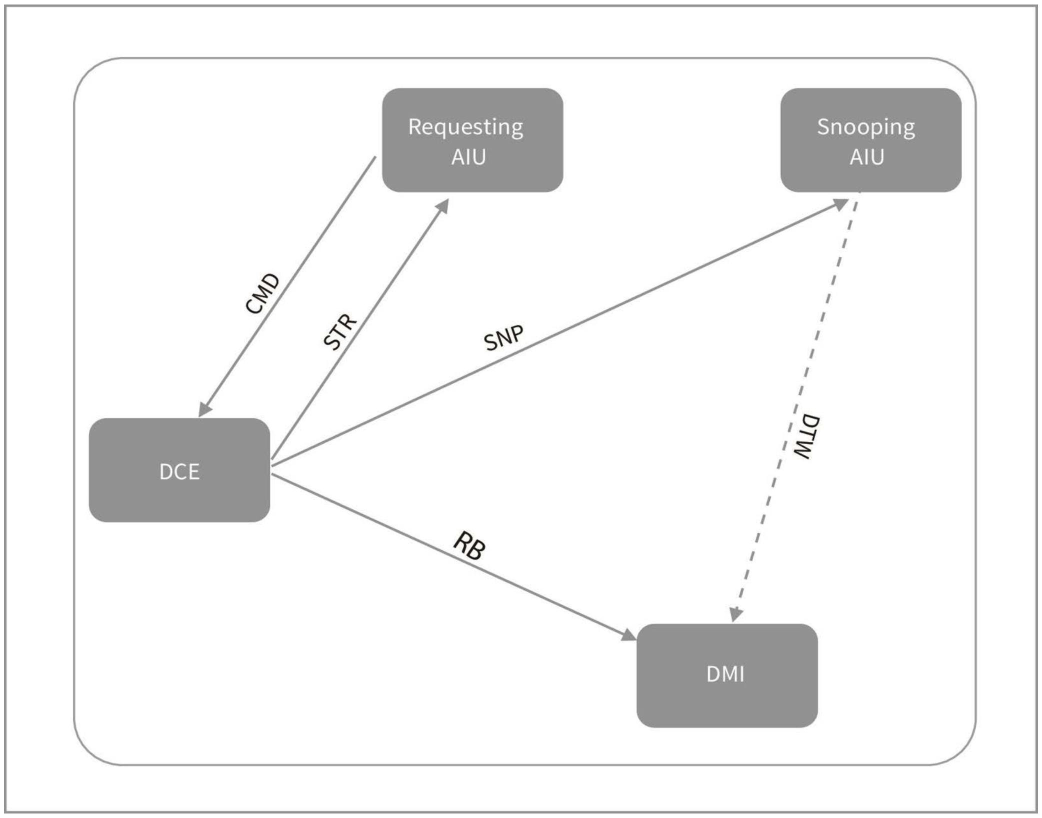
- CMD message is issued by an AIU to a DCE.
- In DCE, the arriving transaction is ordered with respect to other requests to the same cache line. Then the snoop filters will be accessed to determine what SNP messages, if any, need to be issued to AIU to enforce coherence. The request phase of the SNP message indicates the type of coherence operation to be performed, while the response phase of the SNP message indicates the result of the coherence operation.

- Depending on the state of the cache line in the snooped caching agent, the snooped agent forwards data to the requesting agent by issuing a DTR message in response to the SNP message.
- If, based on the state of the snoop filters, no SNP messages are issued or, based on the responses to the SNP messages, no snooped agent will provide data, the DCE issues an MRD message to a DMI.
- If configured with a system cache, the DMI will first check its cache. On a hit data will be returned from the system cache. If no system cache has been instantiated or the system cache does not hold a copy of the requested cache line, the DMI will issue a memory read transaction downstream and data will be provided in a DTR message, once it arrives from memory.

Once all SNP message responses have been gathered and consolidated, the DCE issues an STR message to the requesting AIU. The response phase of the STR message indicates completion of the transaction to DCE and is the final ordering event in the processing sequence. This concludes the processing of a transaction and any request ordered behind this transaction will proceed.

2.1.2 Coherent Clean Request

Figure 4. Coherent Clean Transaction



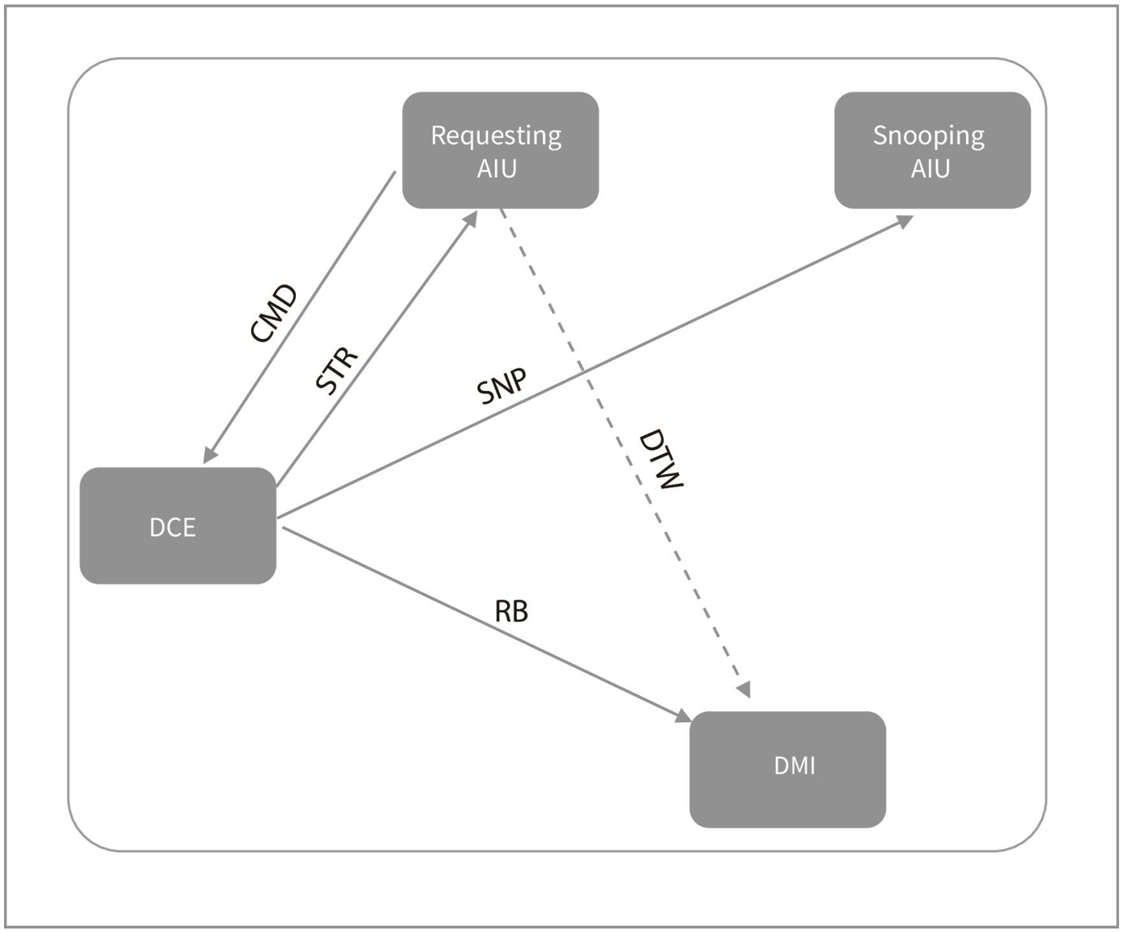
In a coherent clean request:

- CMD message is issued by an AIU to a DCE. In the DCE, a coherent clean transaction behaves similarly to a coherent read transaction, and SNP messages are issued to the appropriate AIUs.
- Additionally, RB is issued to DMI. In this case, however, if a caching agent has a modified copy of data, the AIU issues a DTW message to DMI to update the memory with the most recent value.

Once memory has been updated and all the SNP message responses have been gathered, the DCE issues and completes an STR message.

2.1.3 Coherent Write Request

Figure 5. Coherent Write Request



In a coherent write request:

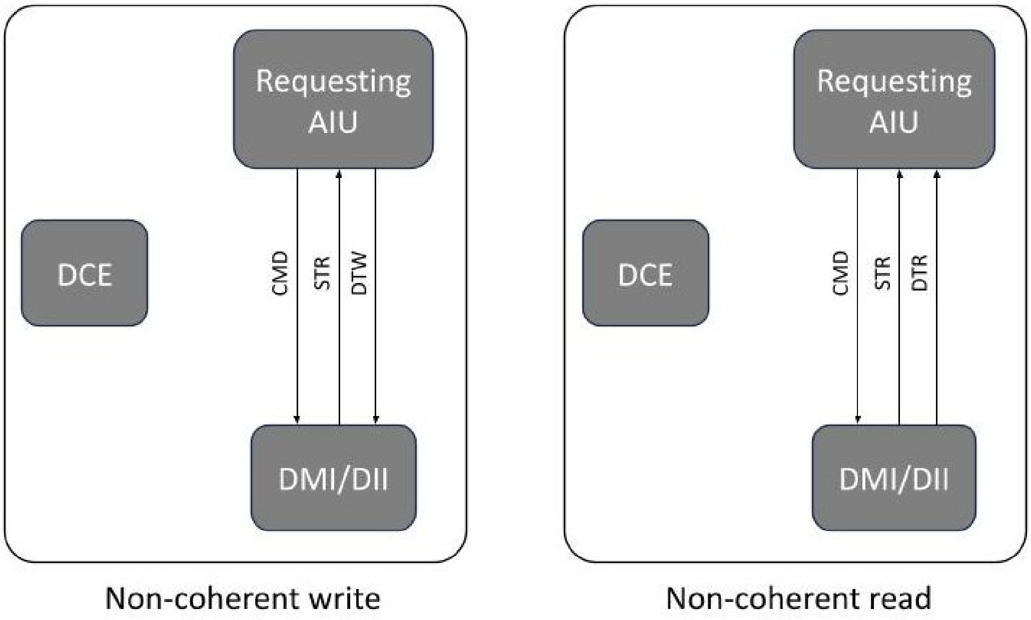
- CMD message is likewise issued to a DCE.
- DCE effectively performs a coherent clean transaction to ensure that memory has been updated and that all copies of data have been invalidated.

- At that point, the directory issues the STR message, and upon receiving the STR message, the AIU issues a DTW message to a DMI to update the memory with the write value.

Upon receiving the DTW message, the DMI performs a memory write transaction, and once the memory has been updated, the STR response phase is performed by DCE. Finally, all hazards are removed and transactions are ordered behind this request may proceed.

2.1.4 Non-Coherent Read and Write Requests

Figure 6. Non-Coherent Read and Write Requests



In a non-coherent read request:

- A CMD message is issued to either a DMI or DII depending on the memory address.
- The DMI/DII then issues a STR message and forwards data to the requesting agent by issuing a DTR message.

In a non-coherent write request:

- A CMD message is issued to either a DMI or DII depending on the memory address.
- The DMI/DII issues a STR message and upon receiving the STR message, the AIU issues a DTW to the DMI/DII to update the memory with the write value.

2.2 Supported Protocols

Currently, Ncore supports only the following AMBA protocols:

- CHI
- ACE
- ACE-Lite
- ACE5-Lite
- AXI 4

2.2.1 CHI Support

Currently, Ncore supports CHI-B and CHI-E protocols. A mixed system with both CHI-E and CHI-B is supported with the following limitations:

- CHI protocol credits and Retry mechanisms are not supported.
- System cache allocation is based on memory attribute rather than the LikelyShared attribute.
- Ncore implements trace mechanism.
- CHI-B : DataCheck is not supported.
- Only little-endian is supported for CHI-B atomic operations.
- CHI-B, CHI-E : WriteUniqueStashPtl and WriteUniqueStashFull are not supported.
- CHI-B, CHI-E : valid StashNID values must reference configuration fabric unit IDs. Only CHI-B and CHI-E, excluding the requester, can be valid Stash Target.
- CHI data interleaving limited to maximum 4 simultaneous transactions.
- CHI-E : No data source information.
- CHI-E : No stash group support.
- CHI-E : No persistence group support.
- CHI-E : No DBID order support.
- CHI-E : No split transaction support.
- CHI-E : No MTE.
- CHI-E : No MPAM.

2.2.2 AXI & ACE Support

Currently, Ncore supports AXI4, ACE, ACE-Lite & ACE5-Lite with the following limitations:

- Barriers are not supported.
- Burst Restrictions:
    - Narrow transfers are not supported for bursts of more than one transfer.
    - All ACE coherent transactions must be 64B or less.
- ACE5-Lite valid StashNID values must reference configuration fabric unit IDs.
- ACE5-Lite StashOnceShared and StashOnceUnique are not supported for non-shareable domain.
- ACE5-Lite data check and poison is not supported.
- ACE5-Lite user loop-back is not supported.
- ACE5-Lite low power signaling is not supported.
- ACE5-Lite device and non-cacheable atomics are not supported.
- AXI4 Fixed burst is not supported.
- AXI4 read data interleaving is not supported at DII.

2.2.3 Distributed Virtual Memory Support

Ncore does not perform conversion between DVMv8 and DVMv7 messages. As a result, if a system includes coherent agents capable of issuing DVMv8 transactions and coherent agents capable of receiving DVMv7 transactions, a bridge to translate DVMv8 transactions to DVMv7 transactions must be implemented. See the "Conversion from DVMv8 to DVMv7 format" section in the ACE specification for additional information. Ncore DVM support can be configured at DVM 8, DVM 8.1, or DVM 8.4. Only ACE agents, CHI agents, and ACE-Lite-E-DVM agents (i.e., ACE-Lite-E that has AC channel) can issue DVM commands to Ncore.

NOTE: Ncore can be configured to not support any DVM operation. If a DVM transaction is issued to Ncore configured to not support any DVM operation, that transaction could lead to a potential hang due to the lack of snoop credits.

2.2.4 TrustZone Support

Ncore supports the TrustZone security model. The secure vs. non-secure indication is propagated and stored throughout the Ncore system.

2.2.5 Access Permission Support

Legacy access permission signals AxPROT[0] and AxPROT[2] are passed through within Ncore for masters with AXI, ACE-Lite, ACE5-Lite, and ACE. In the case of CHI masters, the signal is tied off.

2.2.6 Coherent Exclusive Monitor Support

Ncore supports exclusive monitors by creating a basic monitor for each core in each DCE and a configurable number of tagged monitors in each DCE. Each basic monitor implements the behavior described in the "Minimum PoS Exclusive Monitor" section in the ACE specification, and each tagged monitor implements the behavior described in the "Additional address comparison" section.

The number of processors/threads performing an exclusive sequence must be specified per CAIU. In the case of ACE-CAIU the ARID and AWID bits that identify the core performing an exclusive access sequence must be specified.

2.2.7 Non-coherent Exclusive Monitor Support

Ncore can be configured with non-coherent Exclusive Monitors in the DMI and DII. When Ncore is configured with non-coherent Exclusive Monitors in the DMI or DII, Ncore implements up to eight Monitors in each DMI and DII. The following restrictions apply to NC exclusive transactions:

- Exclusive Reservation Granule of only 64 bytes is supported. ERG is the minimum region that can be tagged for exclusive accesses by an exclusive monitor.
- In the case of mixed coherent/non-coherent address space, coherent access will not affect the state of the exclusive monitor.
- DMI and DII AXI-ID width will be limited to a min size of AIU AXI-ID/LPID width.

When Ncore is not configured with exclusive monitors in the DMI or DII, non-coherent Exclusive Monitors usually reside at a shared target, for example an endpoint. Non-Coherent exclusive transactions are transported and IP blocks connected to DMI/DII that are required to implement non-coherent exclusive monitors. Ncore may modify the AXI ID of non-coherent exclusive transactions but will make sure they are unique. Following restrictions apply to NC exclusive transactions:

- All non-coherent exclusive transactions should be non-cacheable request with visibility attribute set as system visible.
- Non- coherent exclusive transactions that end up as hit in SMC will result in unpredictable behavior, cacheable and non-cacheable address space which include exclusive transactions must not be mixed.
- Exclusive Reservation Granule of only 64 bytes is supported. ERG is the minimum region that can be tagged for exclusive accesses by an exclusive monitor.
- DMI and DII AXI-ID width will be limited to a min size of AIU AXI-ID/LPID width.
- The Exclusive monitor at the target connected to DMI or DII must meet following minimum requirements:
  - Set the monitor reservation only on exclusive read and track both AXID and Address.

  - Clear the monitor reservation on any write accesses to the reserved address.
  - If multiple monitors are present then pick the monitor that is not reserved, if all monitors are reserved then randomly pick a monitor and update it.

2.2.8 Ordering Support

CHI specified request ordering and endpoint ordering is honored with the following restrictions:

- All normal memory access via DMI are endpoint ordered where the end point is cache line sized, i.e., 64 bytes.
- IO accesses via DII honor endpoint ordering based on the largest endpoint specified at system build time, i.e., all accesses which have endpoint ordering attribute set and fall within the specified largest endpoint size will be ordered.

ACE ordering is based on AXI-ID where transactions with the same AXI-ID are processed one after another.

AXI, ACE-Lite, and ACE5-Lite ordering based on AXI-ID is honored, depending on the address space and CSR settings. These settings define how transactions are processed at two different points in Ncore:

- AXI-ID overwrite setting defines how a transaction is processed at the top of Ncore i.e., an NCAIU. The setting is available for both read and write channels individually, by setting the appropriate CSR for an address region an AIU can be configured to either issue transactions in order based on the AXI-ID or ignore the AXI-ID and issue them as soon as possible. This setting does not violate native interface protocol.
- Policy setting defines how a transaction is processed at the bottom of Ncore i.e., at DII/DMI. There are two policy options available:

  a. Endpoint Order: All transactions are issued and completed in order. DII honors endpoint ordering based on the largest endpoint specified at system build time, i.e., all accesses that fall within the specified largest endpoint size will be ordered. DMI honors endpoint ordering based on the cache line size of 64 bytes, i.e., all accesses that fall within the cache line will be ordered.

  b. Relaxed Order: All transactions are issued and completed in order. DII and DMI make sure that transactions to the same address are ordered to avoid RAW, WAR, and WAW hazards.

PCIe Ordering Support

Ncore communicates with a PCIe Bridge/root complex using AMBA protocols like AXI, ACE-Lite, and ACE5-Lite. These protocols do not carry PCIe semantics like posted and non-posted transactions. Ncore supports PCIe ordering in for ACE-Lite/ACE5-Lite connected agents by providing following guarantees.

- Transactions with same AxID within a channel (Read/Write) are ordered, expectation here is that the PCIe agent is sending transactions that require ordering with the same AxID.

- Ncore guarantees that writes can make forward progress by enabling paths to by-pass reads whenever needed within the NOC. These by-passes will make sure RAW, WAR and WAW hazards are avoided as per AXI/ACE-Lite/ACE5-Lite protocols.

Further discussion on PCIe support within Ncore requires an understanding of the bridge/root complex that is interfacing with Ncore, refer to the specific IP user/data manual.

A Producer consumer PCIe use case is shown in Appendix A, in section, *Producer consumer example scenario*.

2.2.9 Quality of Service Support

Ncore implements simple priority arbitration-based QoS. The 4-bit QoS value for an incoming transaction is mapped to one of 8 QoS buckets (3-bit value). This bucket value is used for arbitration at all points in the system where requests accumulate in a buffer or transaction table (CAIU, NCAIU, DCE, and DMI). Note that coherency serialization rules override this priority arbitration.

Ncore implements a QoS Starvation prevention mechanism that guarantees forward progress for low QoS transactions that may be stuck behind higher QoS transactions. This mechanism is implemented at all QoS priority arbitration points in the system.

The implementation of Starvation prevention is based on the number of competing transactions making forward progress. If a transaction has not been able to make forward progress at an arbitration point, while a specified number of competing transactions made forward progress, this transaction is marked as starved and will be allowed to make forward progress on the next arbitration cycle.

The starvation threshold value is specified as a default at build time and can be modified for each Ncore unit CAIU, NCAIU, DCE and DMI individually via CSRs. Depending on the system configuration, which includes transaction table resources, credits, pipeline stages in the network and the traffic profile, the threshold value can be specified, which in turn will determine the maximum bandwidth for high priority (real time traffic) transactions in a loaded system.

Set up `CAIUQOSCR`, `XAIUQOSCR`, `DCEUQOSCR0`, `DMIUTQOSCR0` qos Eventthreshold field to set the activation level of the starvation. `DMIUTQOSCR0` entries are reserved for high priority hard real time traffic.

QoS Map to Verilog Example

The following example shows a 4-bit Native interface QoS value map to 16'h Verilog format for Priority 0 to 7. Value 0 has the highest priority and value 7 has the lowest priority.

```
"qosMap": [

      "16'hC000",   //qosMap[0]: QoS: 15 - 14 -> Priority: 0

      "16'h3000",   //qosMap[1]: QoS: 13 - 12 -> Priority: 1

      "16'h0C00",   //qosMap[2]: QoS: 11 - 10 -> Priority: 2
```

```
"16'h0300",   //qosMap[3]: QoS: 9 - 8 -> Priority: 3
"16'h00C0",   //qosMap[4]: QoS: 7 - 6 -> Priority: 4
"16'h0030",   //qosMap[5]: QoS: 5 - 4 -> Priority: 5
"16'h000C",   //qosMap[6]: QoS: 3 - 2 -> Priority: 6
"16'h0003"    //qosMap[7]: QoS: 1 - 0 -> Priority: 7
```

2.2.10 WFE Wakeup Event Signaling Support

Ncore implements support for WFE events. Ncore provides sideband signals to connect to the DSU EVENTI and EVENTO signals. Ncore also supports broadcasting and event to all connected coherent agents (CAIUs) whenever a coherent exclusive monitor is cleared.

2.2.11 Coherency transition signaling Support

Ncore CAIUs implements SYSCOREQ and SYSCOACK protocol as per the ARM specification for CHI interface. It must be used to transition a CAIU in and out of coherency. The complete procedure is described in the *Reset and Initialization* section.

2.2.12 Read Data Interleaving

The DMI has a reorder buffer which can enable read data interleaving support on the AXI interface. This buffer is implemented in SRAM and includes ECC and & parity protection on the data and address.

3 System Attributes

3.1 Snoop Filters

Ncore uses snoop filters to reduce the system command bandwidth consumed to enforce coherence by eliminating unnecessary coherency operations. The system may be configured with one or more snoop filters, and each caching agent must be assigned to one and only one snoop filter. As a result, each snoop filter can be configured to match the caching characteristics of the associated caching agent.

A tag filter is a set-associative structure that tags entries by physical address. It stores cache line validity and ownership information about its associated caching agents so that only the necessary coherency operations are performed on those caching agents. Each caching agent is represented by a bit that indicates whether the caching agent is a sharer and one of the caching agents, represented by a pointer field, may be designated as the owner of the cache line. A sharer caches a valid, clean, non-unique copy of data, while an owner caches a valid copy of data that may be either dirty or unique. For a coherent transaction that requires a valid copy of a cache line, only the owner is snooped, and for a coherent transaction that requires a unique copy of a cache line, the owner and each sharer are snooped.

In a system that is configured with multiple snoop filters, only one caching agent may be designated the owner. The capacity of a tag filter is determined by specifying the number of sets and the degree of associativity (i.e., number of ways), and the product of both determines the maximum number of cache lines that can be tracked by the tag filter. Because the tag filter must be fully inclusive of the tracked caches, the total number of entries in the tag filter must be greater than or equal to the number of cache lines that can be stored in those caches. Ideally, the number of sets and degree of associativity should match the tracked caches. One way to achieve this is to split the tag filters per agent to match each agent number of set and associativity. Otherwise, to do so typically implies that the degree of associativity of the snoop filter equals the sum of the degrees of associativity for the tracked caches. Often, this is prohibitive from an implementation cost perspective, so an acceptable rule of thumb for most general purpose workloads using only 1 snoop filter for all caching agent, is to configure the snoop filter with a degree of associativity between 8 and 16 and with a capacity of 1.5 to 2 times the sum of entries within all tracked caches.

Regardless of the configuration, a tag filter will maintain inclusion by performing replacements when necessary. In particular, if an entry must be allocated in the tag filter, and there are no available (invalid) ways in the required set, one entry will be chosen at random for replacement, and the directory will issue the recall request to the caching agents that host a copy of this cache line, forcing them to invalidate their copies and update memory as necessary. The directory then uses the newly freed entry for the allocated cache line address.

To reduce the number of replacements issued under certain workloads, a tag filter can be configured with a set of victim buffers that acts as a FIFO of replaced tag filter entries. An entry's position in the FIFO represents its relative age, i.e., the head of the FIFO is the oldest entry and the tail is the youngest. When an entry is replaced from the tag filter, the entry is written to the tail of the FIFO, and if the FIFO is full, a replacement is issued for the entry at the head of the FIFO to make room for the new entry. An entry is transferred from the victim buffers back to the tag filter if a later coherent transaction matches the entry or if a matching set entry is invalidated in the associated snoop filter due to a later coherent transaction.

3.2 Addressing and Memory Regions

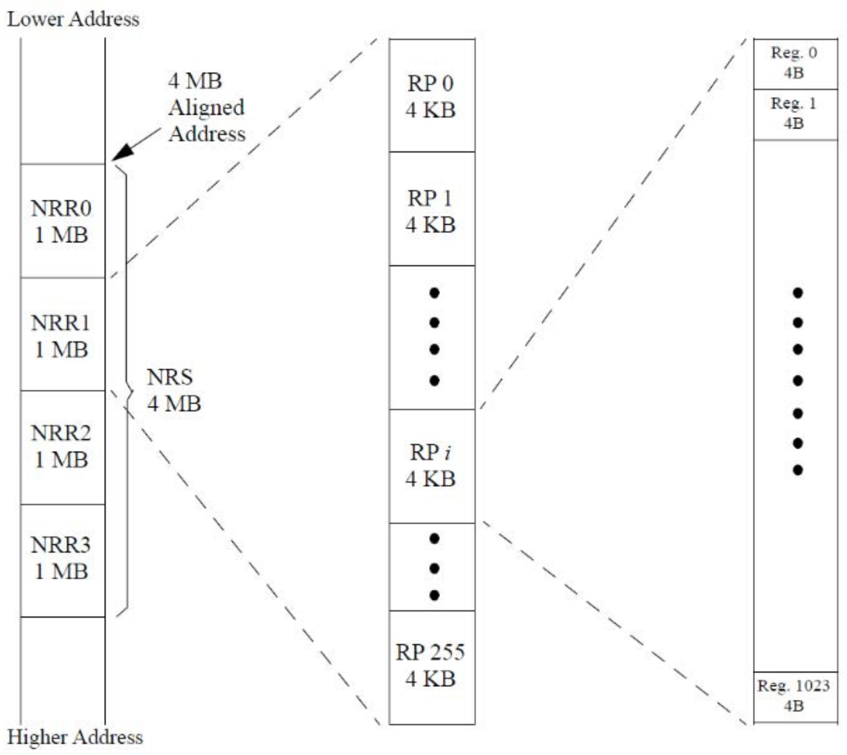
Ncore address map is categorized into three main spaces:

- Ncore Register Space (NRS)
- General Purpose Address Space (GPAS)
- Boot Region (BR)

Each space may contain one or more address regions.

NRS is peripheral storage space used for Ncore unit configuration and status registers (CSR). Organization of NRS is shown in Figure 7.

Figure 7. NRS organization



NRS may contain one or more Ncore register region (NRR). Each NRR is of 1MB and is further divided into 256 register pages (RP) of 4KB. Each Ncore component is associated with an RP and may implement up to 1024, 4B (32bit) registers.

Currently, Ncore systems only support one NRR. The number of valid RPs within this NRR is equal to the total number of Ncore components configured in the system. Table 2 shows the assignment of RPs to different Ncore components.

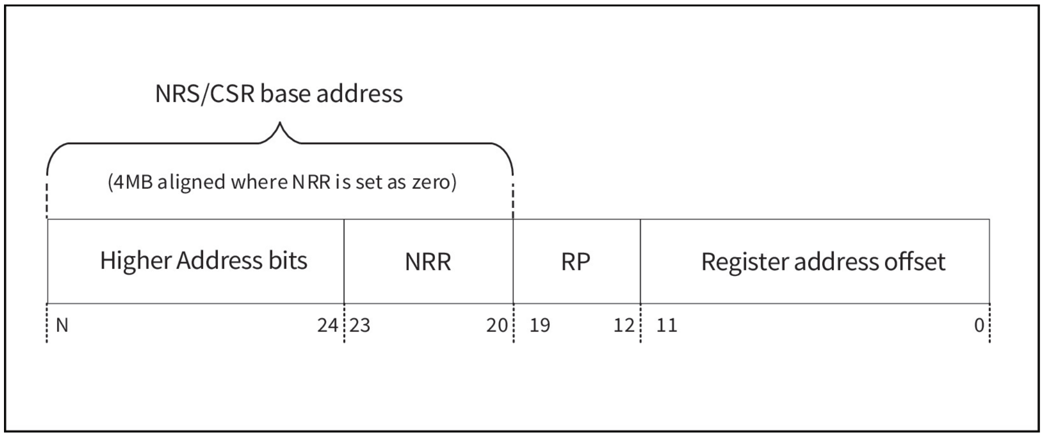
- nCAIUs refers to the number of CAIUs in the system and so forth for different Ncore components.
- nDII is always equal to the total number of DIIs configured plus one. The additional DII (i.e., the last DII) is the Ncore system DII that is mapped to the NRS.
- nDVE and nGRB is always one.

Table 2. RP assignment to Ncore Components

Ncore component	RPNs
CAIUs	RPN0 to RPN(nCAIU-1)
NCAIUs	RPN(nCAIU) to RPN(nCAIU+nNCAIU-1)
DCEs	RPN(nCAIU+nNCAIU) to RPN(nCAIU+nNCAIU+nDCEs-1)
DMIs	RPN(nCAIU+nNCAIU+nDCEs) to RPN(nCAIU+nNCAIU+nDCEs+nDMIs-1)
DIIs	RPN(nCAIU+nNCAIU+nDCEs+nDMIs) to RPN(nCAIU+nNCAIU+nDCEs+nDMIs+nDIIs-1)
DVE	RPN(nCAIU+nNCAIU+nDCEs+nDMIs+nDIIs)

The base address of this region must be specified at Ncore system build time and must be 4MB aligned. Figure 8 shows the address distribution.

Figure 8. Address distribution



General purpose address space is for general purpose use. It can support multiple number of address regions. The number of regions needed within a system must be specified at build time, at least one region is required. The regions can only be configured via CSR accesses and must be configured by software at boot up. They can be mapped to either normal memory (DMI) or a peripheral device (DII). The size of these regions must be a power of 2 within the range of 4 KB and 32 TB (achieved via interleaving). The base address configured must be aligned to the size of the region.

Boot address space is to be used for system boot up. Current Ncore systems only support one boot address region. This region can be mapped to either normal memory (DMI) or a peripheral device address space (DII). The size of this region must be a power of 2 within the range of 4 KB and 8 TB. The base address of this region must be specified at Ncore system build time and must be aligned to the size of the region.

Address decoding is priority based. NRS has the highest priority followed by general purpose address space and lastly boot address space. If an access matches higher priority address space then lower priority address space matches are ignored, and the higher priority space is selected for access. Within an address range multiple address region matches are treated as errors; this specifically applies to general purpose address space. An access will result in an error if multiple address region matches are detected within a single address range or no matches are detected to any address space.

3.2.1 Interleaving

Ncore supports interleaving across snoop filters, system caches, and initiator groups. An initiator group can be formed by creating a group of initiators. All of the initiators comprising an initiator group must be identical. When DMIs, DCEs and Initiator Groups are all interleaved using the same bits, Maestro is able to dramatically reduce the connectivity of the networks in the design. This capability enables higher access bandwidth and eases physical design of the chip. Snoop filters are implemented within DCE and can be interleaved/distributed across 1 to 8 instances of DCE. System caches are implemented within DMI.

DMIs can be interleaved/distributed as a Memory Interleave Group (MIG) of 1, 2, 4, 8, or 16 DMIs. Multiple memory interleave groups form a Memory Interleave Group Set (MIGS). For example, if a system has 7 DMIs, DMI0 to DMI6, they can be combined into 3 different memory interleave groups:

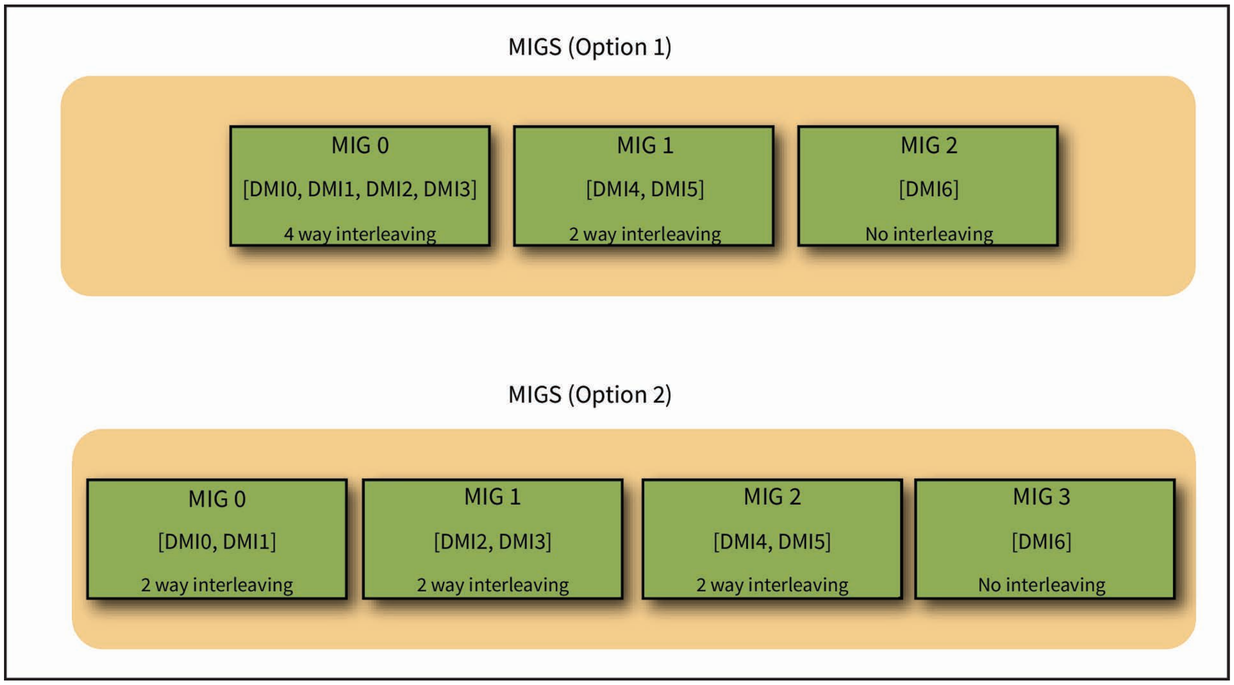
1. Group 0 of four DMIs (DMI0, DMI1, DMI2, DMI3)
2. Group 1 of 2 DMIs (DMI4 and DMI5)
3. Group 2 of one DMI - i.e., DMI6

Three groups together are referred to as a Memory Interleave Group Set. Alternatively, the MIGS may be configured as follows:

1. Group 0 of two DMIs (DMI0, DMI1)
2. Group 1 of two DMIs (DMI2, DMI3)
3. Group 2 of two DMIs (DMI4 and DMI5)
4. Group 3 of one DMI - i.e., DMI6

The groups are illustrated in Figure 9.

Figure 9. MIGS Options



Instance and Set Selection

DCE and DMI component instances, system memory cache sets, proxy cache sets and snoop filter sets are selected as a function of address bits. The method for calculating the selection index is configurable, and the selection index is based on simple interleaving, in which the selection index is determined by the value of a set of N address bits, known as the primary selection bits. The value of selection index bit n equals the value of the address bit defined by primary selection bit n. For example, if primary selection bit `0` is specified as address bit 6, the value of selection index bit `0` equals the value of address bit 6.

Each address bit in the set of primary selection bits must be unique, i.e., an address bit must appear only once in the set of primary selection bits. The address bits must be chosen from a range whose upper bound is less than the size of the system address and whose lower bound is greater than or equal to the size of a cache line. For example, if the size of the system address is 4GB and the size of the directory cache line is 64B, the highest address bit that can be selected is 31 and the lowest address bit is 6. Address bits less than the size of the directory cache line comprise the directory cache line offset, which are bits `0` through 5 in this example.

An identifier is computed by taking the value of the selection index modulo the number of items being selected. To select an instance on a component, an instance identifier is calculated. To select a set in a system memory cache, a proxy cache, or a snoop filter, a set identifier is calculated. In the case of snoop filter and system memory cache the final set index is computed by concatenating the instance identifier with the set identifier, i.e., the instance identifier represents the most significant bits of the set index, while the set identifier represents the least significant bits. The address bits in the set of primary selection bits that determine the instance identifier must not appear in any of the sets of primary selection bits that determine the set identifier.

Additional constraints that apply to various components are described in more detail below.

Snoop Filter, System Memory Cache and Proxy Cache and Initiator Groups

The Set selection is additionally constrained as following:

- The number of sets per instance must be a power-of-two integer
- The number of primary selection bits must equal the binary logarithm of the number of sets.

Distributed Memory Interface Selection

DMI selection is additionally constrained as follows:

- The number of interleaved instances may be either 1, 2, 4, 8, or 16.

The number of primary selection bits must equal the binary logarithm of the number of interleaved instances.

At Ncore system build time, up to 2 set interleaving address bits can be specified for 2, 4, 8, and 16 interleaving options. One of the specified options can be selected via a configuration register.

Distributed Coherency Engine Selection

DCE selection is additionally constrained as follows:

- The number of instances may be either 1, 2, 4, 8, or 16
- The number of primary selection bits must equal the binary logarithm of the number of interleaved instances.
- The primary selection bits must be chosen from the address bits common to all snoop filters and above the cache line offset

Initiator Group Interface Selection

- The number of interleaved instances may be either 1, 2, 4, 8 or 16
- The number of primary selection bits must equal the binary logarithm of the number of interleaved instances
- All initiators in an initiator group must be identical.

3.3 NCAIUs

A non-coherent agent interface unit (NCAIU) can be configured to serve in many different roles.

A Multi-ported AXI NCAIU is an NCAIU which can be configured with 2, 4, or 8 identical AXI4 agent ports. Transaction requests arriving at a Multi-ported NCAIU will be address interleaved across all the ports, according to user specified bits. Each NCAIU core will see a mutually exclusive set of addresses. If a proxy cache is specified then it will implement a set of interleaved proxy cache instances, each associated with a given AXI port. The size of each proxy cache instance can be configured from 4kB to 1Mb in powers of 2. All proxy cache instances will be the same size.

An NCAIU with a proxy cache can access coherent and non-coherent traffic. An address-based coherency mode exists with a coherency bit added to the address map. This allows the AXI NCAIU to access both the DMI coherent address space as well as the non-coherent address space of the DII.

To enable support for legacy agents that do not support read data interleaving, interleaving for the NCAIU can be turned off. In this mode, data returned from the cache will return as non interleaved. This setting is only relevant if the agent's databus interface is less than 64 bits. When interleaving is disabled, data from the network is stored in the data buffer and only forwarded to the agent when full lines have been received. All requests for a given AXID is issued before requests for succeeding AXIDs is provided. This mode will also disable SMI bypasses.

3.4 Caches

An Ncore system can be configured with two different types of caches:

1. Proxy cache: Optional cache configured within an NCAIU
2. System memory cache: Optional cache configured within a DMI

3.4.1 Proxy Cache

The NCAIU with an AXI interface can be configured with proxy cache. This allows NCAIUs to buffer data coherently with respect to other coherent caches in the system. Requests received by the NCAIU access the proxy cache, and if data is present in the cache, the request is serviced from the cache. If not, the cache may be filled, based on the allocation policy indicated by the requests AxCache bits. If a proxy cache miss results in a cache allocation, then the NCAIU performs a fully-coherent request within Ncore. If the proxy cache miss does not result in a cache allocation, the NCAU performs an IO-coherent request within Ncore.

An NCAIU with a proxy cache can access coherent and non-coherent traffic. An address-based coherency mode exists with a coherency bit added to the address map. This allows the AXI NCAIU to access both the DMI coherent address space as well as the non-coherent address space of the DII.

To enable support for Legacy agents which do not support read data interleaving, interleaving for the NCAIU can be turned off. In this mode, data returned from the cache will be returned non-interleaved. This setting is only relevant if the agent's databus interface is less than 64 bits.

The proxy cache supports the MOSI (Modified, Owned, Shared and Invalid) cache state model. A cache line in the modified state is ***unique*** (i.e., the only cached copy in the caching agent caches) and ***dirty*** (i.e., inconsistent with respect to the next level of the memory hierarchy). A cache line in the owned state may not be unique (i.e., a valid copy may be present in another caching agent cache) and may be dirty. A cache line in shared state may not be unique and is presumed to be clean. Finally, a cache line in the invalid state is not present in the cache.

The capacity of the proxy cache is determined by specifying the number of sets and the degree of associativity (i.e., number of ways), and the product of both determines the number of cache lines that are buffered in the cache.

The proxy cache allocates cache lines based on AxCache signaling on the AXI interface. Allocation and visibility mapping is shown in the table below. Visibility here refers to when Ncore issues the response for the writer request, Early means Ncore issues the response as soon as the request leaves Ncore via a DMI or DII, Late means it waits for the response from the downstream agent.

Table 3. AxCache Allocation & Visibility Mapping

AxCache	Allocation	Visibility	Notes
`0000`	No	Late (system)	Device Non-bufferable
`0001`	No	Early	Device bufferable
`0010`	No	Late (system)	Normal Non-cacheable, Non-bufferable
`0011`	No	Early	Normal Non-cacheable, Bufferable
`0110`	Yes(R)/No(W)	Early	Write through no allocate / read allocate
`0111`	Yes(R)/No(W)	Early	Write back no allocate / read allocate
`1010`	No(R)/Yes(W)	Early	Write through write allocate
`1011`	No(R)/Yes(W)	Early	Write back write allocate
`1110`	Yes	Early	Write through read & write allocate
`1111`	Yes	Early	Write back read and write allocate

Three cache line replacement policies are supported as described in "Cache line Replacement Policy".

3.4.2 System Memory Cache

The system memory cache (SMC) allows DMI(s) to buffer cache lines and service requests without accessing the next level of memory hierarchy. The SMC represents a lower level cache in the memory hierarchy relative to those in the caching agents and NCAIU. SMC is a pseudo-exclusive cache i.e., it is primarily an exclusive cache, which increases total available cache on the chip but there are certain scenarios where SMC may hold a copy of cache line that also resides in the cache hierarchy above SMC. An example scenario is when a dirty owner evicts a shared cache line.

SMC supports a three-state cache model, a cache line may either be in an ***invalid*** state (i.e., not present in the cache) or ***clean*** state (i.e., present and consistent with respect to the next level of the memory hierarchy) or ***dirty*** state (i.e., present and inconsistent with respect to the next level of the memory hierarchy). SMC in general uses the native interface (CHI, ACE, ACE-Lite, ACE5-Lite, and AXI) allocation hint to allocate to the cache. These hints can be overwritten by a configuration register setting; where following types of request may individually be disabled from allocation into SMC.

- WriteUnique request from CHI/ACE processor
- Write request with clean data
- Write request with dirty data
- All read request data
- All write request data

The capacity of the SMC is determined by specifying the number of sets and the degree of associativity (i.e., number of ways), and the product of both determines the number of cache lines that are buffered in the cache. SMC supports three cache line replacement policies, these are described in the cache line replacement policy section below.

SMC supports the following optional features: Scratchpad, Way Partitioning, Cache line Replacement Policy, and Address Hashing.

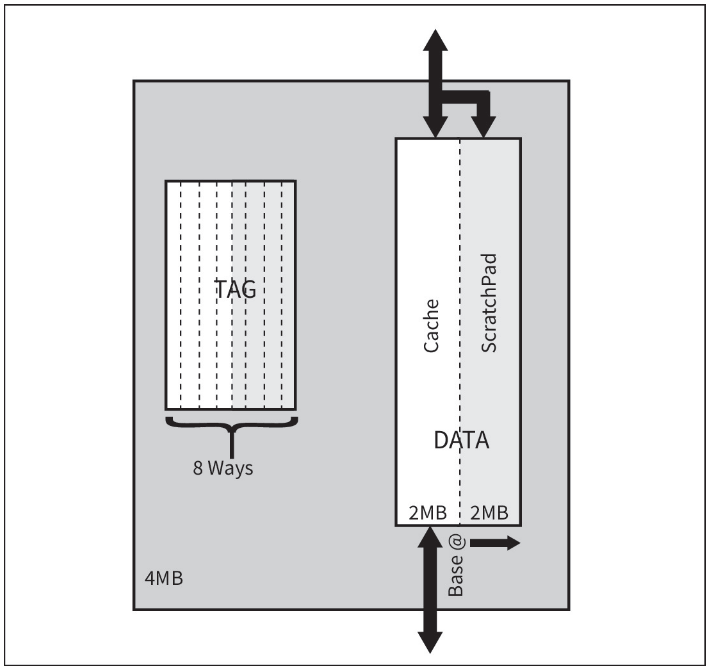
Scratchpad

Scratchpad can be thought of as a standalone, low latency memory which is separate from the cache. This memory can be direct mapped to a configurable address range. Accesses to this address range will be directed only to the scratchpad and will not be seen by the caching partition of SMC or the downstream memory.

SMC can be configured at initialization to partition one part of the cache as scratchpad on a per way basis. Figure 10 illustrates a 4 MB SMC with 8 ways that has 4 of its ways configured as scratchpad, i.e., 2 MB as scratchpad and 2 MB as regular cache.

NOTE: When a way is configured as scratchpad, all the sets associated with that way will be configured as scratchpad. Only contiguous ways starting from way 0 can be configured as scratchpad.

Figure 10. Scratchpad RAM



Way Partitioning

Way partitioning capability enables reservation of a portion of the cache for a single or multiple CAIUs/NCAIUs, for the purpose of brevity CAIU/NCAIU is referred to as AIU in this description. SMC can be configured to partition the cache on per way basis. Note that, when a way is reserved, all the sets associated with that way will be reserved for the configured AIU. When way partitioning is enabled, all incoming request with configured AIUs will only allocate to the enabled way(s). No other AIUs requests will be able to allocate to the enabled way(s); furthermore, all incoming requests will be able to hit on all the cache ways, irrespective of the AIU it is from. Way partitioning can be configured via configuration and status registers.

Cache line Replacement Policy

Allocating a new cache line may cause a valid cache line to be replaced based on a configurable replacement policy. The cache will search for an invalid entry in the set to which the cache line maps. If an invalid entry exists, that entry is selected for the new cache line, but if an invalid entry does not exist, the cache selects a cache line for replacement using either a random policy or a not-recently-used policy (NRU) or a pseudo least-recently-used policy (PLRU). The random policy uses a pseudo-random pointer to choose a valid cache line for replacement, while the NRU policy tracks cache lines that have been recently used (i.e., recently accessed) and uses the pseudo-random pointer to choose a valid cache line for replacement from the set of cache lines that have not been recently used.

The recently used status is reset once all entries in a set are deemed recently used. The PLRU policy uses a binary decision tree – where the leaf nodes represent the cache lines in a set - to find a node currently accessed, and updates the binary decision tree to point to the node that is farthest away from the node currently accessed. As a result, the PLRU policy uses the binary decision tree to approximate the relative age order of cache lines.

Address Hashing

A good address hashing function enables the uniform distribution of cache entries. This reduces address conflicts, resulting in less cache entries evictions. This also reduces memory bank and index collisions, allowing higher memory throughput. This also reduces traffic congestions and hotspots. Ncore implements hashing through the use of primary and secondary selection bits. Primary selection bits are used to specify and access the sets of the cache. Secondary selection bits can be specified to enable an XOR-based permutation of each primary selection bit by a set of secondary selection bits.

If secondary selection bits are specified then each selection index bit is equal to the logical XOR of the corresponding primary selection bit and the list of associated secondary selection bits. If secondary selection bits are not specified then the selection index bit is equal to the primary selection bit. The number of secondary selection bits and the number of primary selection bits must match and the primary and secondary selection bits cannot overlap.

3.5 Credits and Resources

To alleviate congestion and manage resources in CDTI, credit counters are implemented for certain protocol message types. Credits are explicitly assigned to CMD, SNP, MRD, and RB messages. Remaining messages such as, STR, DTR, and DTW are implicitly controlled by credits defined for other messages. For example, DTW messages are limited by RB messages.

The credit values must be configured based on round trip latencies. Initiating Ncore component implements credit counters to track messages it initiates. Similarly, target Ncore component implements recourses to accommodate all messages it can receive.

For each CAIU/NCAIU, following credit counters are implemented:

- CMD message credit counter per connected instance of DCE
- CMD message credit counter per connected instance of DMI
- CMD message credit counter per connected instance of DII
- CMD message credit counter for DVE; this limits the number of DVM commands. This is implemented in all CAIUs only.

For each DCE, the following credit counters are implemented:

- SNP message credit counter per connected instance of CAIU/NCAIU
- MRD message credit counter per connected instance of DMI
- RB message credit counter per connected instance of DMI

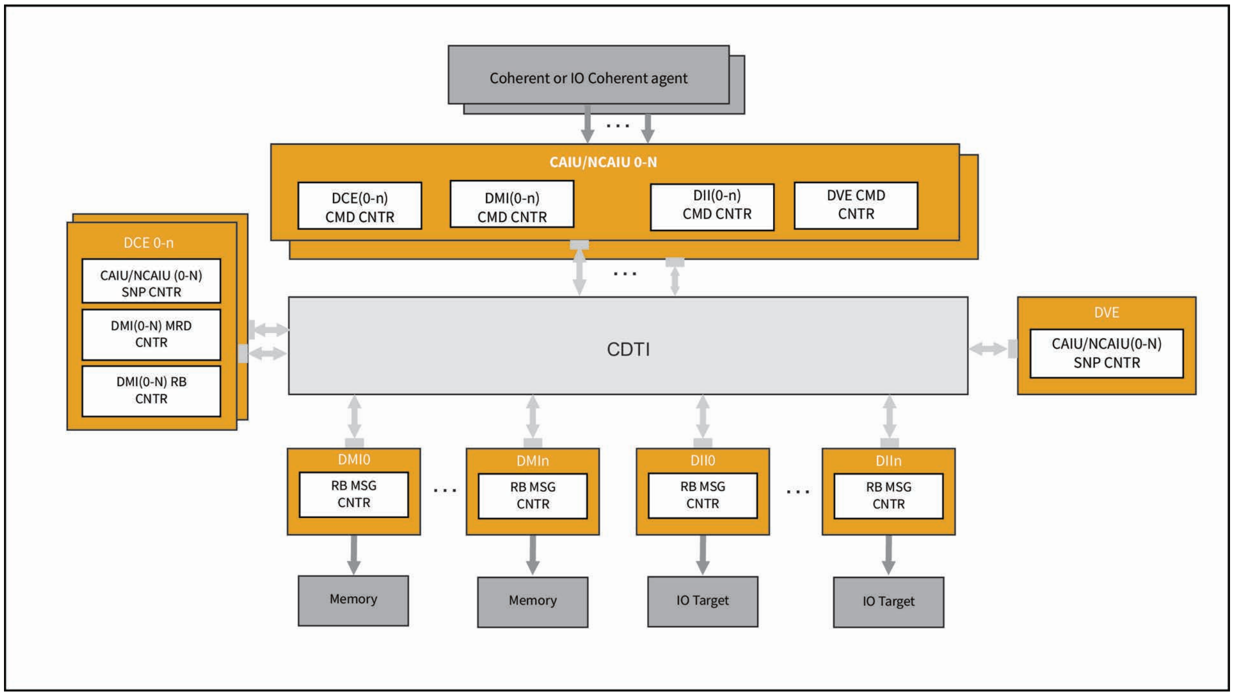
For each DMI/DII, the following counter is implemented:

- RB message counter

For DVE, the following credit counter is implemented:

- SNP message counter per DVM capable instance of CAIU/NACIU

Figure 11. Credit Counters Overview



For each CAIU/NCAIU:

- Snoop transaction table: this transaction table is sized based on SNP message credits.
- For each connected DCE/DMI/DII, a CMD message credit register in CAIU/NCAIU must be configured with respect to the following rules:
    - The sum-up of Number of credits configured in each AIU connected to the same DCE/DMI/DII must be less or equal to this DCE/DMI/DII CMD message skid buffer size. Otherwise the target DCE/DMI/DII skid buffer can be overloaded and that can lead to a deadlock.
    - The minimum number of credits to be configured in a CHI/DCE is 1.
    - The minimum number of credits to be configured in a IOAIU is 2.
    - The maximum number of credits to be configured in a CAIU/NCAIU is 31.
    - If a CAIU/NCAIU is not used, the number of credits can be set to 0.

For each DCE:

- For each connected DMI, a Mrd message credit register must be configured with respect to the following rules:

- The sum-up of credits configured in each DCE connected to the same DMI is less or equal to this DMI Mrd SkidBuffer size. Otherwise the target DMI skid buffer can be overloaded and that can lead to a deadlock.
- The minimum number of credits is 1.
- The maximum number of credits to be configured in a DCE is 31.

For each DMI, the RB control entries table is sized based on connected DCEs' RB message credits.

In addition to above resources, all Ncore components are configured with additional resources to achieve required throughput. These resources are configured based on round trip latencies.

The CMD message skid buffer in the DMI, DCE, and DII and the MRD message skid buffer in the DMI share the same architecture. These skid buffers are comprised of two parts, a fixed size, limited skid buffer supporting arbitration and an overflow FIFO storage buffer. The arbitration buffer supports sizes of either 4, 8, 16, 32, 48, 64, 128, 192, or 256 entries. The storage buffer is sized by the size of the arbitration buffer plus {0, 4, 8, 16, 32, 64, 96, 128, 160, 192 224, 128, 256}. The storage buffer size can not exceed 320. At run time, the maximum number of credits each flow may use is programmable and can be optimized based on differences in round trip latency or bandwidth requirements.

For each CAIU/NCAIU, the following resources are implemented:

- Outstanding Transaction Table Control Entries : This limits the number of outstanding transactions this CAIU/NCAIU can have at a given time.

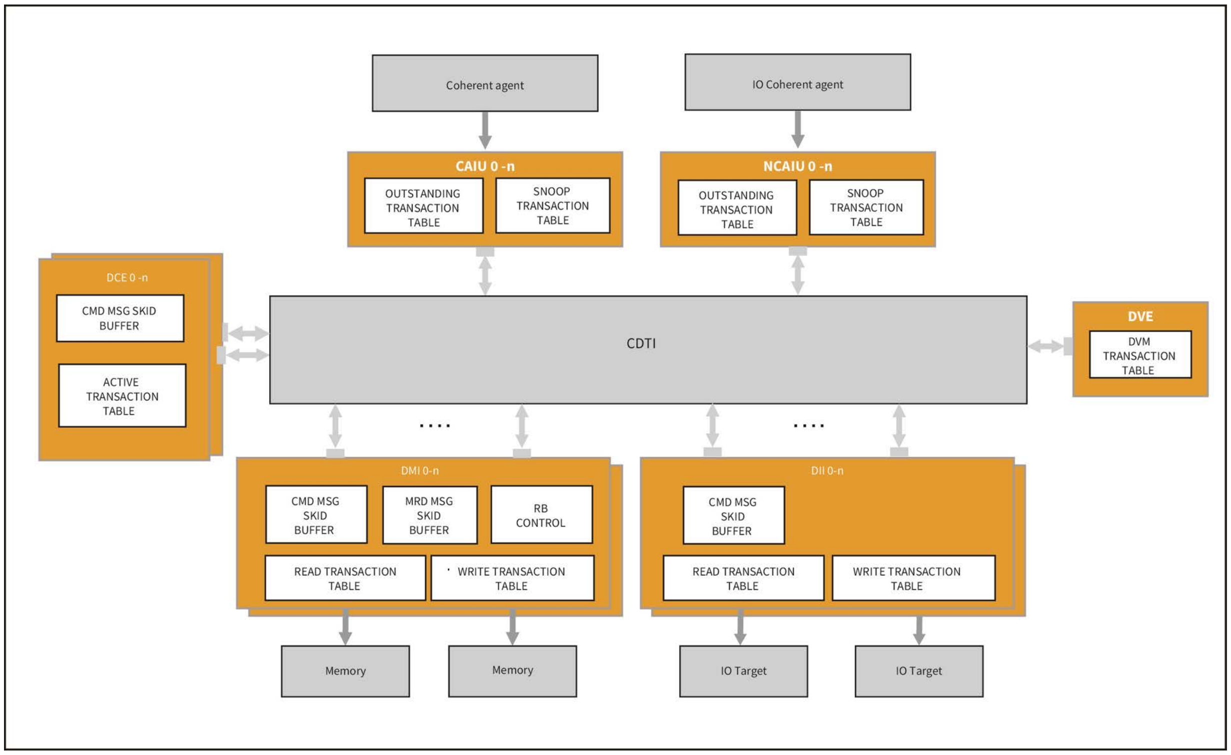
For each DCE, the following resources are implemented:

- Active Transaction Table Control Entries : This limits the number of active transactions each DCE can have at a given time.

For each DMI/DII, the following resources are implemented:

- Read Transaction Table Control Entries : This limits the number of read transactions each DMI/DII can have outstanding on the AXI interface.
- Write Transaction Table Control Entries : This limits the number of write transactions each DMI/DII can have outstanding on the AXI interface.

Figure 12. Resource Overview



3.6 Storage Protection

Storage structures, such as snoop filters, proxy/system cache tag and data arrays, and transaction table data buffers, are configured with the following protection schemes:

- None - no protection provided
- Parity Entry - a parity bit per entry; error detection capability for single bit errors, but no error correction capability
- SECDED Entry - single error correction, double-bit error detection code per entry; error detection and correction capability

In this context, an *entry* represents a snoop filter entry, a proxy/system cache tag entry, or a single data beat in a data transfer into or out of a proxy/system cache data entry or a transaction table data buffer entry.

Parity protection enables the hardware to detect single-bit errors in an entry, but all errors are uncorrectable errors. SECDED protection enables the hardware to detect double-bit errors in an entry and to detect and correct single-bit errors in an entry. With SECDED protection, double-bit errors are uncorrectable errors, and single-bit errors are correctable errors. Each unit implements resources for logging and signaling uncorrectable and correctable errors separately. Each Ncore unit outputs two interrupt signals: a combined uncorrectable interrupt signal and a combined correctable interrupt signal.

The SECDED protection algorithm is implemented with a Hamming code.

3.7 Power Management

Ncore power management capabilities include two levels of clock gating and Q channel support.

3.7.1 Clock Gating

The first level clock gating is enabled by synthesis tools, for example Synopsys® Design Compiler. A second level of clock gating may be inserted, on a per Ncore unit basis. This level gates the clock for the complete unit when no active transactions are within that unit.

3.7.2 Q Channel

Ncore will create a Q channel interface per clock domain which can be used to clock gate the desired clock domains. This being a coherent system, for each coherent agent’s native interface the SYSCOREQ and SYSCOACK hand shake protocol, if available, and a software-controlled bring-up/tear-down sequence must be followed. The sequence is described in section 5.2.

3.8 Transport Interconnect

Ncore 3 uses two types of transport interconnect: the CSTI and CDTI.

CSTI is the service interconnect for configuration and status register access to all Ncore components. CDTI is the main protocol interconnect.

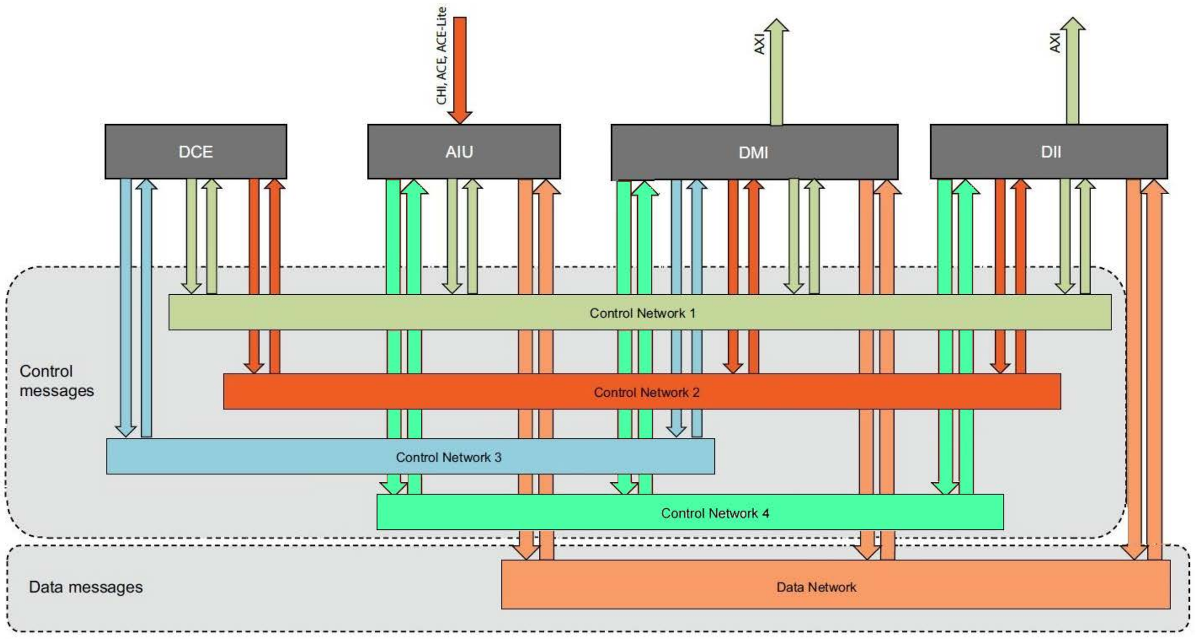
Ncore 3 components generate different set of control and data messages as described in section, “Cache Coherency Protocol”. These messages are mapped to one data network and at least two control networks, these networks are referred together as CDTI. Separation of different messages onto independent transport resources improves performance and breaks dependency loops inside the transport that may cause deadlocks.

The number of control networks are determined based on floor plan and throughput requirements. Control network width is determined based on system address width and other protocol control information that needs to flow on it. The width of the Control network width is not user selectable. Data network supports three width options, 64, 128, and 256 bits. These options can be mixed and matched within a single data network.

For a single native request, the Ncore protocol will issue multiple control messages and most likely a single data message, as described in the “Cache Coherency Protocol” section. Depending on the width of the data network, one or more control network can be chosen to achieve the required throughput. For example, if the data network is 256 bits wide, a cache line read will consume 2 data network cycles. In this case, three control networks must be used to achieve maximum throughput. This ensures the number of control network cycles required will be within the 2 data network cycles. Currently, Ncore supports the following three options:

- Three networks, one data and two control networks, one for requests and another for responses
- Four networks, one data and three control networks, one for requests, another for responses and lastly a dedicated network for messages between DCEs and DMIs.
- Five networks, one data and four control networks, one for requests, two for responses and lastly a dedicated network for messages between DCEs and DMIs. This is illustrated in Figure 13.

Figure 13. Five Interconnect Option



Each Ncore component has one service port that connects to the CSTI and multiple protocol ports that connect to CDTI. Number of ports connecting to CDTI for each Ncore component is as follows:

- CAIU/NCAIU has one data egress/ingress port and two control egress/ingress ports
- DCE has two control egress/ingress port and one dedicated control egress/ ingress port for DMI
- DMI has one data egress/ingress port and three control egress/ingress ports, one of the control egress/ingress port is dedicated to DCE
- DII has one data egress/ingress port and two control egress/ingress ports
- DVE has one data ingress port and two control egress/ingress ports

The control ports may be combined within CDTI depending on the choice of number of control interconnects.

Topology and bus width for different networks within CDTI and CSTI can be designed depending on floorplan and throughput requirements.

3.8.1 Topologies

The transport is created during the "Architectural Design" phase, in the Maestro software. During this phase, Maestro renders one or more NoC implementations to fulfill the requirements set forth in the specifications. NoC design primitives - such as interfaces, switches, links, adapters - are instantiated, connected and parameterized to achieve the design objectives.

The primary task in this phase is the choice of a network topology to connect all components together, as well as the assignment of the routes along which traffic will travel.

The main objectives are:

- Instantiating components that will functionally connect all communicating endpoints
- Ensuring those components are sized to support the required bandwidths
- Connecting them in a way that offers the lowest possible latency for critical traffic
- Choosing routes that are as direct as possible for critical traffic, and otherwise distribute the traffic load to minimize contention hot spots
- Ensuring that the routes do not interfere with each other to create deadlocks

Maestro permits manual implementation of these tasks (with tool assistance) or, automatic implementation for designs that are suitably regular.

3.8.2 Manual Topology Generation

Topologies and routes can be manipulated manually, while automating tedious and error-prone subtasks. An example flow might be as follows:

1. Maestro automatically creates Ncore units based on user definition of the native interfaces (Native interfaces are AXI/ACE/CHI interface)
2. Any number of switches that need to be implemented in the NoC can be defined graphically or by TCL scripts. The designer then connects them (graphically or via TCL) to describe routes. This process can be iterated as needed. Switches are placed into clock and power domains, and the data width is configurable
3. Whenever the connectivity spans clock boundaries or data width boundaries (for the data network), Maestro automatically adds appropriate adapters to implement the clock or packet size conversion
4. Other primitives can be manually added to satisfy functional or non-functional objectives; for example, extra buffering elements (to improve performance), or pipeline stages (to facilitate timing closure)

Maestro does not impose restrictions on the choice of topology but helps ensure the outcome is correct and efficient

3.8.3 Packet Definitions

Both the CDTI and CSTI networks are packet based. The definition of the packet is determined by the super set of requirements for all the blocks tied to a particular network. Depending on the network, packets can be multiple beats and a given network can carry packets with different numbers of beats, anywhere from 1 to 8 beats.

3.8.4 CDTI Blocks

There are five blocks used in the networks of the CDTI. Three of them can be manipulated manually:

- Switches
- Pipe Adapters
- Rate Adapters

Two of them are automatically inserted by Maestro after a customer is done manipulating the topology:

- Clock Adapters
- Width Adapters

3.8.5 Switches

Switches can be parameterized in five ways:

1. Number of ports
2. Depth of internal buffering
3. Routes Taken
4. Data width of a switch (Only in the data network)
5. The clock driving a switch

Maestro allows switches to be broken apart and merged from the initial topology it creates after socket and route definition. The maximum allowed number of input and output ports is 16 each, but this may be exceeded temporarily while editing the topology.

While the topology is being manipulated, if multiple paths between two Ncore units exist for a given route, Maestro allows the customer to select which path to take.

The data width for a switch can be set to 64, 128, or 256 bits wide.

The depth of a buffer placed on all the input ports can be set to 0, 2, 4, 8, 12, 16, 24 or 32. The depth represents a beat of a packet. The buffer can hold multiple packets or parts of packets.

The switch uses round robin for arbitration.

3.8.6 Pipe Adapters

Pipe adapters are used to fix timing issues should the link between two blocks travel a far distance. Three parameters can be configured for pipe adapters:

1. Depth

While the width and clock of a pipe adapter cannot be directly modified in the Maestro GUI, it can be controlled indirectly by prepending or appending it to an existing block. The inserted adapter will directly assume the width and clock configuration from that block.

The depth of a pipe adapter may be set to 0, 1, 2 or 3. The depth represents active beats in a packet. The pipe adapter can hold multiple packets or parts of packets.

3.8.7 Clock Adapters

Maestro will insert a clock adapter between any two blocks that are driven by different clocks and connected by a direct link.

If two blocks in the topology are connected by a link and they are connected to different clocks, Maestro inserts a clock adapter between the two blocks.

Maestro allows you to globally set the number of synchronizing registers used when crossing clock domains. Acceptable values are 2,3 or 4. This value sets the number of registers used in the circular FIFO as 2*(sync depth+1) + 1 raised to the nearest even number.

Maestro supports the insertion of clock adapters on the ATP interface, the APB interface as well as the SMI interface.

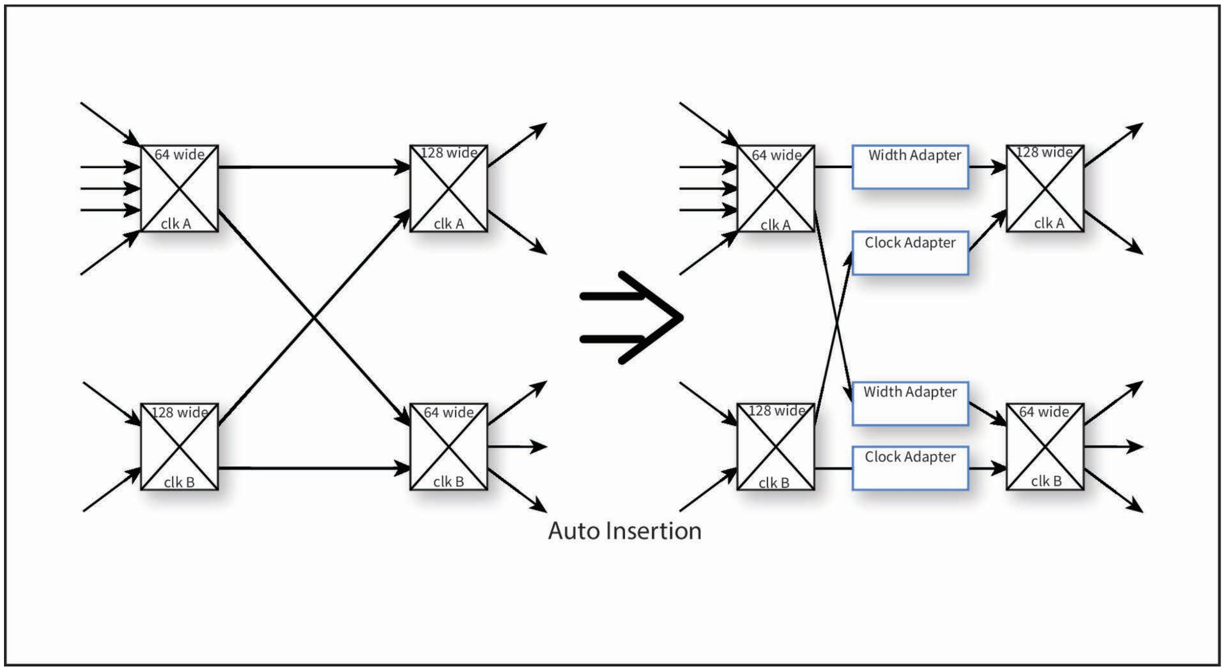
Synchronizing Register Depth	Circular FIFO Depth
2	8
3	10

3.8.8 Width Adapters

If two blocks in a topology are connected by a link and they have different widths, Maestro inserts a width adapter.

If the two connected blocks have both, different widths and different clocks, then Maestro will insert both a clock adapter and width adapter. The order of insertion for concatenated adapters will be chosen to maximize performance.

Figure 14. Auto Adapter Insertion



3.8.9 Rate Adapters

The user may insert rate adapters to the topology in front of width adapters when increasing the bus width size. The rate adapter will prevent partially filled data packets from entering the network to avoid wasting transfer bandwidth.

3.8.10 CSTI Network

The CSTI network is used to access path for configuration, status, and error registers. It is automatically generated by Maestro and is composed of two sub networks:

1. Request network
2. Response network

3.8.11 CSTI Blocks

The CSTI network uses switches and adds three more blocks:

1. AXI initiator
2. APB Port
3. APB targets

The widths of all CSTI blocks are fixed at 32 bits.

By default, all the AXI Initiators can access the APB targets. The Maestro GUI allows selectiono of only one AXI Initiator to enable access via this Initiator to the

APB targets. A dedicated register can be programmed to promote additional AXI Initiators to access the APB targets.

3.8.12 AXI Initiator

A selected AXI slave agent is tied to a special DII (the system DII) and forwards all CSR transactions into the CSTI.

This AXI agent has two ports into the CSTI network, an outgoing request port and an incoming response port.

3.8.13 APB port

Alternatively, an APB port can be used for accessing the Control and Status registers [refer to chapter 3.10.3 Secondary APB port support for CSR network].

The APB Port complies with APB issue 4 and Ncore supports additional functionalities listed below:

Transaction protection

Ncore CSR registers are only accessible via Secure Transactions and the protection signal PPROT[1] signal must be asserted low.

The additional APB PPROT[0] and PPROT[2] protection signals are don't care for Ncore 3.6.

Sparse data transfer

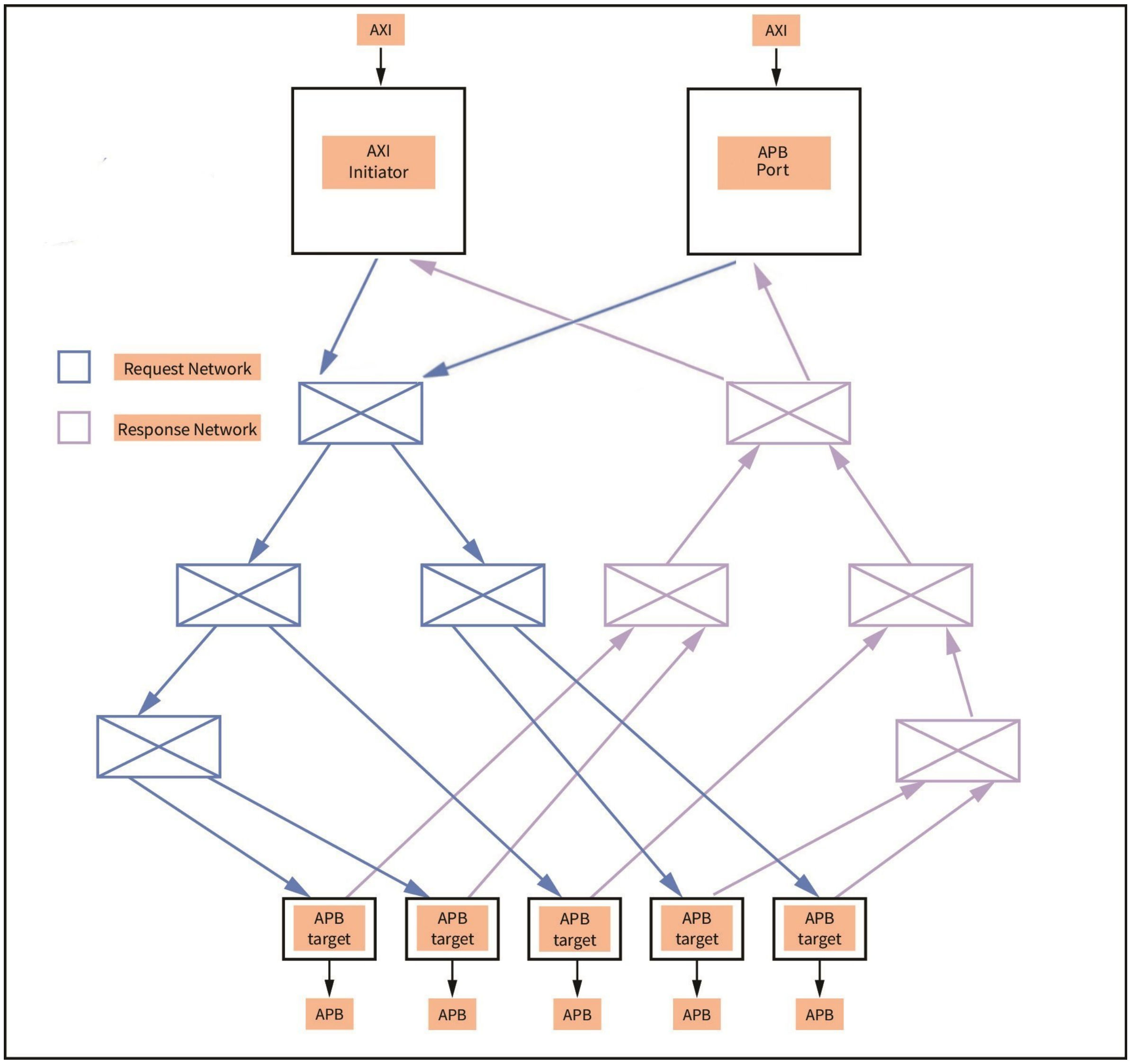
Ncore does not support partial writes for CSR accesses. During a write operation, the APB PSTRB[3:0] signals must be asserted high.

3.8.14 APB Target

Every Ncore block that contains control and status registers exposes an APB port that is compliant to APB issue 4. An APB target block connects this port to the CSTI network. It receives requests from the AXI initiator, converts them to APB transactions on the APB interface and returns responses to the AXI Initiator. All APB transactions generate responses, so when a write completes the APB Target sends a completion response back to the AXI initiator.

The APB Target has two ports into the CSTI network, an incoming request port and an outgoing response port.

Figure 15. CSTI Network with AXI Initiator and APB Port masters



3.8.15 Deadlock Analysis

The generated topology needs to be deadlock-free. Maestro employs a couple of techniques to ensure this.

- Protocol-level deadlocks:
  - Protocol-related deadlocks are avoided by enforcing:
    - A separation of message classes, which are mapped onto two or more networks. For each configuration, messages are segregated to avoid cyclic dependencies.
    - A safe allocation of end-to-end message credits. This allocation is designed to ensure that endpoints will always have sufficient buffer-

ing to receive in-flight messages, to prevent logjams back propagating across the interconnect.

Maestro automatically checks for deadlocks in generated topologies. If the user inadvertently configures a deadlock-prone system, the checker will alert to the possibility of cycles in the system. In this case, RTL generation will not be allowed.

3.9 Error Management

In Ncore, an error is defined to be a deviation from correct or expected system behavior. An undetectable error is a deviation that hardware is unable to detect, while a detectable error is a deviation hardware is able to detect. Detectable errors may be further classified as:

- correctable errors, if the hardware is able to correct the deviation upon its detection
- uncorrectable errors, if the hardware is unable to correct the deviation.

3.9.1 Correctable Errors

Ncore detects and processes correctable errors when it is configured with SRAMs that are SECDED protected, these are classified into two categories:

- Correctable data errors: these are errors on SRAM accesses that store data such as data SRAMs in caches
- Correctable address errors: these are errors on SRAM access that store address information, such as tag memories in caches.

In addition when Ncore is configured with Resiliency it is able to detect and correct errors on the network. For more details, refer to *Chapter 4*.

3.9.2 Uncorrectable Errors

Ncore detects and processes multiple uncorrectable errors, these are classified as follows:

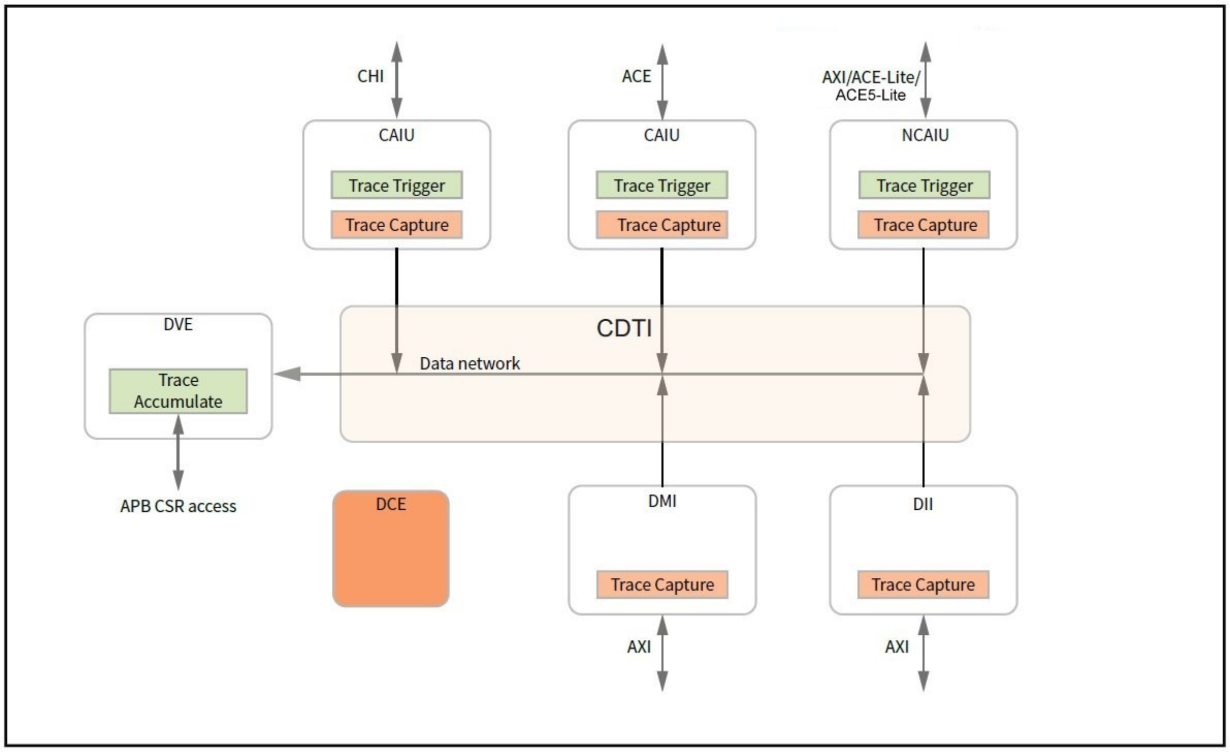
- Uncorrectable data errors : These are errors on SRAM accesses to memories that store data, such as data SRAMs in caches. When this error is detected, the cache line will be marked as poisoned within the SRAM and the poison attribute will be returned on subsequent access to the cache line. The request will then complete, propagating poison information on the native interface. For a CHI interface the appropriate error signaling method is defined in the standard, the poison bit will be set on the data response. For ACE, ACE-Lite, and AXI, error signaling is provided by the byte enables for poisoned data beats will be de-asserted. If enabled, an interrupt will be asserted by the component that detected the error.
- Uncorrectable address errors : These are errors on SRAM accesses to memories that store address information, such as tag SRAMs in caches. When this error is detected, the request will be completed with an error message on the native interface, if enabled, an interrupt will be asserted by the component that detected the error.

- Address map uncorrectable errors : These are errors on accesses that either hit multiple configured address regions within GPAS or do not hit any configured address regions. the request is completed with an error message on the native interface. If interrupts are enabled, it is asserted at the component that detected the error.
- Native uncorrectable errors : These are errors that are propagated into Ncore by external native interfaces such as AXI. If the original request does not complete, then the error is propagated, and the original request is completed with error messages on the native interface. If interrupts are enabled, it is asserted at the Ncore component that observes the error on its Native interface.

3.10 Debug and Trace

Ncore implements a debug and trace capability which adds three blocks, Trace Trigger, Trace Capture, and Trace Accumulate shown in the following figure. This capability is built on top of the existing networks and does not add any additional network wires.

Figure 16. Debug and Trace Capability



3.10.1 Trace Trigger

Trace trigger adds the capability of marking a Native Interface transaction that is to be traced, It is present in each AIU. There are two classifications of marking a transaction to be traced.

1. Master initiated trace, this applies to CHI and ACE5-Lite interfaces, which have trace signal capability on the interface
2. Ncore initiated trace, this applies to all AIUs in Ncore, here Ncore provides CSRs which can be configured to mark desired transactions as trace transactions.

Master Initiated Trace

CHI and ACE5-Lite interfaces have trace signaling capability, the master can identify transactions that are to be traced by appropriate interface signaling. Ncore honors this signaling and meets the interface requirements. In the case of ACE5-Lite the more conservative approach is taken where all responses on the native interface propagate the trace signal, furthermore in the case of write channel the trace signaling on W channel is ignored and trace signaling on AW is considered

Ncore Initiated Trace

All AIUs in the Ncore system including those that support trace signaling have the capability to initiate transaction tracing using internal CSRs. An incoming transaction on the native interface is compared with the trace CSR settings, if there is a match the transaction is marked to be traced. Multiple numbers of CSR sets can be present as specified at build time. Different trace options are described below. In a single set of registers, all the options below can be enabled together, and they are checked with an AND option — i.e., all enabled options should match the incoming transaction. Between register sets it is an OR operation.

- **Trace Signal**: Trace transactions that are marked by the native interface, if trace signaling capability is available on the Native interface.
- **Address Range**: Trace transactions that match the specified address range. One address range can be specified per register set.
- **Op Code**: Trace transactions that match the specified Op codes. Up to 4 op codes can be specified at a time per register set.
- **Memory Attribute**: Trace transactions that match the specified memory attributes on the native interface. This is MemAttr for CHI and AxCache for AXI/ACE/ACE-Lite/ACE5-Lite. One memory attribute can be specified per register set.
- **User Bits**: Trace transactions that match the specified user bits on the native interface. One user bit value can be specified per register set.
- **Target Type**: Trace transactions that match the specified target type DMI/DII. One target type can be specified per register set.

3.10.2 Trace Capture and Accumulate

Trace capture samples and packages packets that are marked to be traced. These packets are accumulated in the Trace Accumulate buffer and can be read out using CSR reads. The size of accumulate buffer is configurable at build time. Contact your AE for details on how to decode captured trace information.

3.10.3 Secondary APB port support for CSR network

If a system hang occurs, the user cannot dump the Debug and Trace register values from the AXI Initiator via the CSR network. An APB port is implemented by default to allow alternate access to all the Ncore Configuration and Status Registers which allows analysis of the Debug and Trace registers content in order to isolate the root cause of the system hang. This port may be restricted for debug purposes only. If the same register is accessed concurrently via this debug APB port and the AXI Initiator CSR access, then the effect on the CSR is undefined. Ncore does not guarantee any ordering between the two different access points.

3.11 Performance Counters

Each Ncore component can be configured to implement either 4 or 8 performance counters. These are 64 bit counters capable of counting up to two event at a time. These events may be single bit event or multi bit event. Following counter modes are available:

1. Normal count: In this mode the counter counts both the events together, if both the events are asserted then it is counted as two, if one event is asserted then it is counted as one
2. AND count: In this mode the counter counts one if both the events are asserted.
3. XOR count: In this mode the counter counts one only if one event is asserted and the other is de asserted.
4. Instantaneous count: In this mode the counter provides the instantaneous value of the multi bit event as is.

Each counter reports count via 2 registers one is a 32-bit count register that reports lower 32 bits and another configurable register which supports

1. Capturing the upper 32 bits of the count
2. Used as an accumulation register for IIR filter.
3. Use as max/saturation value for accumulation events

3.11.1 CAIU Performance Events

Event #	Width	Name	Description
1	1	SMI 0 Tx Stall event	Counts every cycle when valid is set and ready is low
2	1	SMI 1 Tx Stall event	Counts every cycle when valid is set and ready is low
3	1	SMI 2 Tx Stall event	Counts every cycle when valid is set and ready is low
4		Reserved	
5	1	SMI 0 Rx Stall event	Counts every cycle when valid is set and ready is low
6	1	SMI 1 Rx Stall event	Counts every cycle when valid is set and ready is low
7	1	SMI 2 Rx Stall event	Counts every cycle when valid is set and ready is low
8		Reserved	
9	1	ACE AW stall event	Counts every cycle when valid is set and ready is low (Does not apply to CHI AIU)
10	1	ACE W stall event	Counts every cycle when valid is set and ready is low (Does not apply to CHI AIU)
11	1	ACE B stall event	Counts every cycle when valid is set and ready is low (Does not apply to CHI AIU)
12	1	ACE AR stall event	Counts every cycle when valid is set and ready is low (Does not apply to CHI AIU)

Event #	Width	Name	Description
13	1	ACE R stall event	Counts every cycle when valid is set and ready is low (Does not apply to CHI AIU)
14	1	ACE AC stall event	Counts every cycle when valid is set and ready is low (Does not apply to CHI AIU)
15	1	ACE CD stall event	Counts every cycle when valid is set and ready is low (Does not apply to CHI AIU)
16	1	ACE CR stall event	Counts every cycle when valid is set and ready is low (Does not apply to CHI AIU)
17	1	CmdReq WR event	Count all write commands that are associated with data, refer CCMP for all related commands.
18	1	CmdReq RD event	Count all read commands that are associated with data, refer CCMP for all related commands.
19	1	SnpRsp event	Count all snoop responses that trigger data transfer read or write.
20	8	Active OTT entries	Number of active Outstanding Transaction Table (OTT) entries
21	3	Reserved	
22	3	Captured SMI packets	Number of SMI packets captured by trace capture logic
23	3	Dropped SMI packets	Number of SMI packets dropped by trace capture logic
24	1	Address Collisions	Count number of address collisions on incoming transactions
25	3	Interleaved Data	Count max number of active interleaved data transactions.
26	1	Agent event counter	Counts number of events triggered by the native agent
27	1	NoC event counter	Counts number of events triggered by the NoC
28		Reserved	
29		Reserved	
30	1	Div 16 counter	Divide by 16 free running counters
31	1	Number of QoS starvations	Number of times QoS starvations occurred

3.11.2 NCAIU Performance Events

Event #	Width	Name	Description
1	1	SMI 0 Tx Stall event	Counts every cycle when valid is set and ready is low
2	1	SMI 1 Tx Stall event	Counts every cycle when valid is set and ready is low
3	1	SMI 2 Tx Stall event	Counts every cycle when valid is set and ready is low
4		Reserved	
5	1	SMI 0 Rx Stall event	Counts every cycle when valid is set and ready is low
6	1	SMI 1 Rx Stall event	Counts every cycle when valid is set and ready is low

Event #	Width	Name	Description
7	1	SMI 2 Rx Stall event	Counts every cycle when valid is set and ready is low
8		Reserved	
9	1	ACE-Lite/AXI AW stall event	Counts every cycle when valid is set and ready is low
10	1	ACE-Lite/AXI W stall event	Counts every cycle when valid is set and ready is low
11	1	ACE-Lite/AXI B stall event	Counts every cycle when valid is set and ready is low
12	1	ACE-Lite/AXI AR stall event	Counts every cycle when valid is set and ready is low
13	1	ACE-Lite/AXI R stall event	Counts every cycle when valid is set and ready is low
14	1	ACE-Lite AC stall event	Counts every cycle when valid is set and ready is low.
15	1	ACE-Lite CD stall event	Counts every cycle when valid is set and ready is low.
16	1	ACE-Lite CR stall event	Counts every cycle when valid is set and ready is low.
17	1	CmdReq WR event	Count all write commands that are associated with data, refer CCMP for all related commands.
18	1	CmdReq RD event	Count all read commands that are associated with data, refer CCMP for all related commands.
19	1	SnpRsp event	Count all snoop responses that trigger data transfer read or write.
20	8	Active OTT entries	Number of active Outstanding Transaction Table (OTT) entries
21		Reserved	
22	3	Captured SMI packets	Number of SMI packets captured by trace capture logic
23	3	Dropped SMI packets	Number of SMI packets dropped by trace capture logic
24	1	Address Collisions	Count number of address collisions on incoming transactions
25		Reserved	
26	1	Agent event counter	Counts number of events triggered by the native agent
27	1	NoC event counter	Counts number of events triggered by the NoC
28		Reserved	
29		Reserved	
30	1	Div 16 counter	Divide by 16 free running counters
31	1	Number of QoS starvations	Number of times QoS starvations occurred

Proxy Performance Events

Event #	Width	Name	Description
32	1	Cache read hit	
33	1	Cache write hit	
34	1	Cache snoop hit	

Event #	Width	Name	Description
35	1	Cache eviction	
36	1	Cache no ways to allocate	
37	1	Cache fill stall	
38	1	Cache read stall	
39	1	Cache write stall	
40	1	Cache replay	
41	1	Cache read miss	
42	1	Cache write miss	
43	1	Cache snoop miss	
44		Reserved	
45		Reserved	
46		Reserved	
47		Reserved	
48		Reserved	
49		Reserved	
50		Reserved	
51		Reserved	
52		Reserved	
53		Reserved	
54		Reserved	
55		Reserved	
56		Reserved	
57		Reserved	
58		Reserved	
59		Reserved	
60		Reserved	
61		Reserved	
62		Reserved	

3.11.3 DMI Performance Events

Event #	Width	Name	Description
1	1	SMI 0 Tx Stall event	Counts every cycle when valid is set and ready is low
2	1	SMI 1 Tx Stall event	Counts every cycle when valid is set and ready is low

Event #	Width	Name	Description
3	1	SMI 2 Tx Stall event	Counts every cycle when valid is set and ready is low
4	1	SMI 3 Tx Stall event	Counts every cycle when valid is set and ready is low
5	1	SMI 0 Rx Stall event	Counts every cycle when valid is set and ready is low
6	1	SMI 1 Rx Stall event	Counts every cycle when valid is set and ready is low
7	1	SMI 2 Rx Stall event	Counts every cycle when valid is set and ready is low
8	1	SMI 3 Rx Stall event	Counts every cycle when valid is set and ready is low
9	1	AXI AW stall event	Counts every cycle when valid is set and ready is low
10	1	AXI W stall event	Counts every cycle when valid is set and ready is low
11	1	AXI B stall event	Counts every cycle when valid is set and ready is low
12	1	AXI AR stall event	Counts every cycle when valid is set and ready is low
13	1	AXI R stall event	Counts every cycle when valid is set and ready is low
14		Reserved	
15		Reserved	
16		Reserved	
17	1	DtwReq event	Counts every cycle when valid DtwReq is accepted into DMI.
18	1	DtrReq event	Counts every cycle when valid DtrReq is transmitted out DMI.
19		Reserved	
20	8	Active WTT entries	Number of active WTT entries
21	8	Active RTT entries	Number of active RTT entries
22	4	Captured SMI packets	Number of SMI packets captured by trace capture logic
23	4	Dropped SMI packets	Number of SMI packets dropped by trace capture logic
24	1	Address Collisions	Count number of address collisions on incoming transactions
25	1	Number of Merge events	Count number of DtwMergeMrd messages
26	1	Number of system visible Txn	Count number of system visible transactions
27		Reserved	
28		Reserved	
29		Reserved	
30	1	Div 16 counter	Divide by 16 free running counters
31	1	Number of QoS starvations	Number of times QoS starvations occurred

SMC Performance Events

Event #	Width	Name	Description
32	1	Cache read hit	
33	1	Cache write hit	
34	1	Cache CMO hit	Operations like cleanInvlidate etc
35	1	Cache eviction	
36	1	Cache no ways to allocate	
37	1	Cache fill stall	
38	1	Cache read stall	
39	1	Cache write stall	
40	1	Cache replay	
41	1	Cache read miss	
42	1	Cache write miss	
43	1	Cache CMO miss	
44		Reserved	
45		Reserved	
46		Reserved	
47		Reserved	
48		Reserved	
49		Reserved	
50		Reserved	
51		Reserved	
52		Reserved	
53		Reserved	
54		Reserved	
55		Reserved	
56		Reserved	
57		Reserved	
58		Reserved	
59		Reserved	
60		Reserved	
61		Reserved	
62		Reserved	

3.11.4 DII Performance Events

Event #	Width	Name	Description
1	1	SMI 0 Tx Stall event	Counts every cycle when valid is set and ready is low
2	1	SMI 1 Tx Stall event	Counts every cycle when valid is set and ready is low
3	1	SMI 2 Tx Stall event	Counts every cycle when valid is set and ready is low
4		Reserved	
5	1	SMI 0 Rx Stall event	Counts every cycle when valid is set and ready is low
6	1	SMI 1 Rx Stall event	Counts every cycle when valid is set and ready is low
7	1	SMI 2 Rx Stall event	Counts every cycle when valid is set and ready is low
8		Reserved	
9	1	AXI AW stall event	Counts every cycle when valid is set and ready is low
10	1	AXI W stall event	Counts every cycle when valid is set and ready is low
11	1	AXI B stall event	Counts every cycle when valid is set and ready is low
12	1	AXI AR stall event	Counts every cycle when valid is set and ready is low
13	1	AXI R stall event	Counts every cycle when valid is set and ready is low
14		Reserved	
15		Reserved	
16		Reserved	
17	1	DtwReq event	Counts every cycle when valid DtwReq is accepted into DII.
18	1	DtrReq event	Counts every cycle when valid DtrReq is transmitted out DII.
19		Reserved	
20	8	Active WTT entries	Number of active Write Transaction Table (WTT) entries
21	8	Active RTT entries	Number of active Read Transaction Table (RTT) entries
22	3	Captured SMI packets	Number of SMI packets captured by trace capture logic
23	3	Dropped SMI packets	Number of SMI packets dropped by trace capture logic
24	1	Address Collisions	Count number of address collisions on incoming transactions
25		Reserved	
26		Reserved	
27		Reserved	
28		Reserved	
29		Reserved	
30	1	Div 16 counter	Divide by 16 free running counters
31		Reserved	

3.11.5 DCE Performance Events

Event #	Width	Name	Description
1	1	SMI 0 Tx Stall event	Counts every cycle when valid is set and ready is low
2	1	SMI 1 Tx Stall event	Counts every cycle when valid is set and ready is low
3	1	SMI 2 Tx Stall event	Counts every cycle when valid is set and ready is low
4		Reserved	
5	1	SMI 0 Rx Stall event	Counts every cycle when valid is set and ready is low
4	1	SMI 1 Rx Stall event	Counts every cycle when valid is set and ready is low
7	1	SMI 2 Rx Stall event	Counts every cycle when valid is set and ready is low
8		Reserved	
9		Reserved	
10		Reserved	
11		Reserved	
12		Reserved	
13		Reserved	
14		Reserved	
15		Reserved	
16		Reserved	
17		Reserved	
18		Reserved	
19		Reserved	
20	8	Active ATT entries	Number of active ATT entries
21		Reserved	
22		Reserved	
23		Reserved	
24	1	Address Collisions	Count number of address collisions on incoming transactions
25	1	SF hit	Snoop filter hit (either owner or sharer) count
26	1	SF miss	Snoop filter miss (neither owner nor sharer) count
27	1	SF recall	Snoop filter recall transaction count
28	1	Snoop rsp miss	Snoop response reports miss
29	1	Snoop rsp Owner transfer	Snoop response Ownership transfer

Event #	Width	Name	Description
30	1	Div 16 counter	Divide by 16 free running counters
31	1	Number of QoS Starvations	Number of times QoS starvations occurred

3.11.6 DVE Performance Events

Event #	Width	Name	Description
1	1	SMI 0 Tx Stall event	Counts every cycle when valid is set and ready is low
2	1	SMI 1 Tx Stall event	Counts every cycle when valid is set and ready is low
3	1	SMI 2 Tx Stall event	Counts every cycle when valid is set and ready is low
4		Reserved	
5	1	SMI 0 Rx Stall event	Counts every cycle when valid is set and ready is low
4	1	SMI 1 Rx Stall event	Counts every cycle when valid is set and ready is low
7	1	SMI 2 Rx Stall event	Counts every cycle when valid is set and ready is low
8		Reserved	
9		Reserved	
10		Reserved	
11		Reserved	
12		Reserved	
13		Reserved	
14		Reserved	
15		Reserved	
16		Reserved	
17		Reserved	
18		Reserved	
19		Reserved	
20	8	Active STT entries	Number of active Snoop Transaction Table (STT) entries
21		Reserved	
22	1	Captured DtwDbgReq packets	Number of DTW Debug (DtwDbg) packets captured by trace accumulation logic
23	1	Dropped DtwDbgReq packets	Number of DtwDbg packets dropped by trace accumulation logic
24		Reserved	
25		Reserved	
26		Reserved	
27		Reserved	
28		Reserved	
29		Reserved	
30	1	Div 16 counter	Divide by 16 free running counters
31		Reserved	

3.11.7 Bandwidth Counters

Ncore provides Bandwidth counters in AIUs, DMIs and DIIs. These counters count in 64 Byte data accuracy and are divided by 16 accuracies of the clock frequency. BW counting can be filtered based on FunitID or user bits, this can be configured in CSRs xBCNTFR and xBCNTMR

- In AIUs count the following:
- Read data bandwidth, this refers to the data read into the master/cache
- Write data bandwidth, this refers to the data written out of the master/cache, this may include evicted data.
- Snoop data bandwidth, this refers to the data moved out of the master/cache due to snoops

In DMIs/DIIs the count is as follows:

- Read data bandwidth, this refers to the data read from DMI/DII. In the case of DMI the data may be read from SMC if present.
- Write data bandwidth, this refers to data being written into DMI/DII. In the case of DMI the data may be written into SMC if present.

BW counting is achieved by setting xCTCR0 as follows:

- Counter control to 3b100 i.e., use as 32 bit counter
- SSR count to 3b100 i.e., use as 32 bit counter
- Count first event to one of the following events depending on what is to be counted
  - CmdReq WR event: to count AIU write BW
  - CmdReq RD event: to count AIU read BW
  - SnpRsp event: to count Snoop BW
  - DtwReq event: to count DMI/DII write BW
  - DtrReq event: to count DMI/DII read BW
- Count Second event to divide by 16 clock cycle event

Once the counter is enabled the following is reported:

- xCNTSR: number of divide by 16 clock cycles
- xCNTVR: number of 64 Bytes of data

The above information can be used to calculate the effective BW. The counter must be disabled before reading them in to get the correct BW. Note that this will be approximate BW as all transactions that were counted may necessarily not be 64 Bytes transactions and the time accuracy is divided by 16.

3.11.8 Latency Histogram Counters

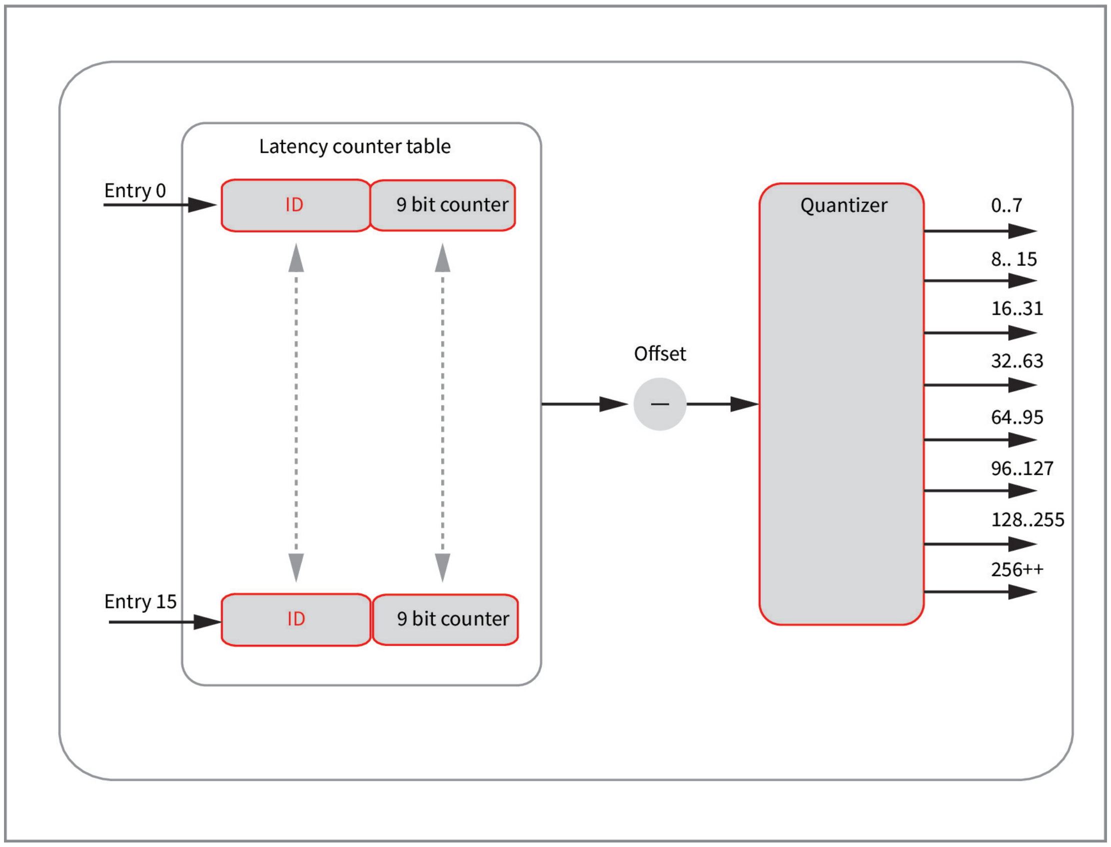
Ncore provides latency counters in AIUs, DMIs and DIIs for either reads or writes. Latency is reported back as a binned histogram. It is an optional feature that is controlled via a unit level parameter, as described in the Parameters section. At any given time, it can be configured to report only one type of latency i.e., read latency or write latency. The latency reported at AIUs is from Ncore system preceptive, i.e., it is reporting the complete lifetime of the transaction within Ncore. The latency reported at DMI/DII is more a closer representation of latency seen at the Native AXI interface.

Read latency includes all transitions defined by CCMP/Native interface as a read transition that provides data to the requester. Write latency includes all transactions defined by CCMP/Native interface as a write transaction that provides data to be written at the slave. Stash transactions are not included.

Logic for generating the latency histogram is implemented as a separate block, as shown in Figure 17. Whenever a transaction of the selected type is allocated to the transaction table of the AIU, it is also allocated to the latency counter table. In the case of DMI/DII, the transaction will be allocated when it is available to be issued on the Native interface. The ID represents the transaction table ID and the counter is a 9-bit counter. The counter will be started as soon as the entry is allocated. While all entries within the latency counter table are occupied, any newly arriving, qualified transaction will be ignored and not counted towards the latency histogram.

The counter increments every 2, 4, 8 or 16 clock cycles based on the CSR configuration. If the counter hits its maximum value, it will saturate until deallocation, the saturated value will be used for latency histogram binning. In an AIU an entry in the latency counter table will be deallocated at approximately the same time the corresponding entry is deallocated from the transaction table. In the case of DMIs/DIIs, an entry in the latency counter table will be deallocated at approximately the same time the corresponding response is received from the native interface. The configured offset value will then be subtracted from the entry’s counter value and the result will be limited towards zero. The quantizer uses the offset corrected value as the index to determine which histogram bin must be incremented.

Figure 17. Latency Histogram Block



To create a single histogram the first 4 consecutive counter registers xCNTCR must be configured as follows:

- Counter control to 3b100 i.e., bit bin counter
- SSR count to 3b100 i.e., 32 bit bin counter
- Count first event is don't care in this case
- Count Second event is don't care in this case

xLCNTCR must be configured as follows:

- Latency prescale to desired prescale value
- Read/Write latency to desired read or write latency to be counted
- Latency bin offset to desired offset value
- Latency count enable to enable latency counting

xMCNTCR must be configured as follows:

- Local Count Clear to 1b1 and clear all local counters (this will clear all local counters other than the histogram counters)
- Local Count Enable to 1b1 and enable all local counters

After elapsed number of transactions or time or set Local Count Enable to 1b0 to stop the counters

8 histogram bins can be read from the 4 consecutive counter registers:

- xCNTSR0: Bin 1
- xCNTVR1: Bin 2
- xCNTSR1: Bin 3
- xCNTVR2: Bin 4
- xCNTSR2: Bin 5
- xCNTVR3: Bin 6
- xCNTSR3: Bin 7

4

Resiliency

4.1 Functional Safety Overview

Resiliency is a feature that enables Functional Safety, fault tolerance and other high reliability features in customer designs. It utilizes user configurable protection techniques and component duplication within the Ncore interconnect, with a fault controller to monitor and log faults detected. Also, supplied is a Built-In Self Test (BIST) engine that enables checking of fault detection logic. This chapter discusses the architecture of the resiliency features provided in Ncore.

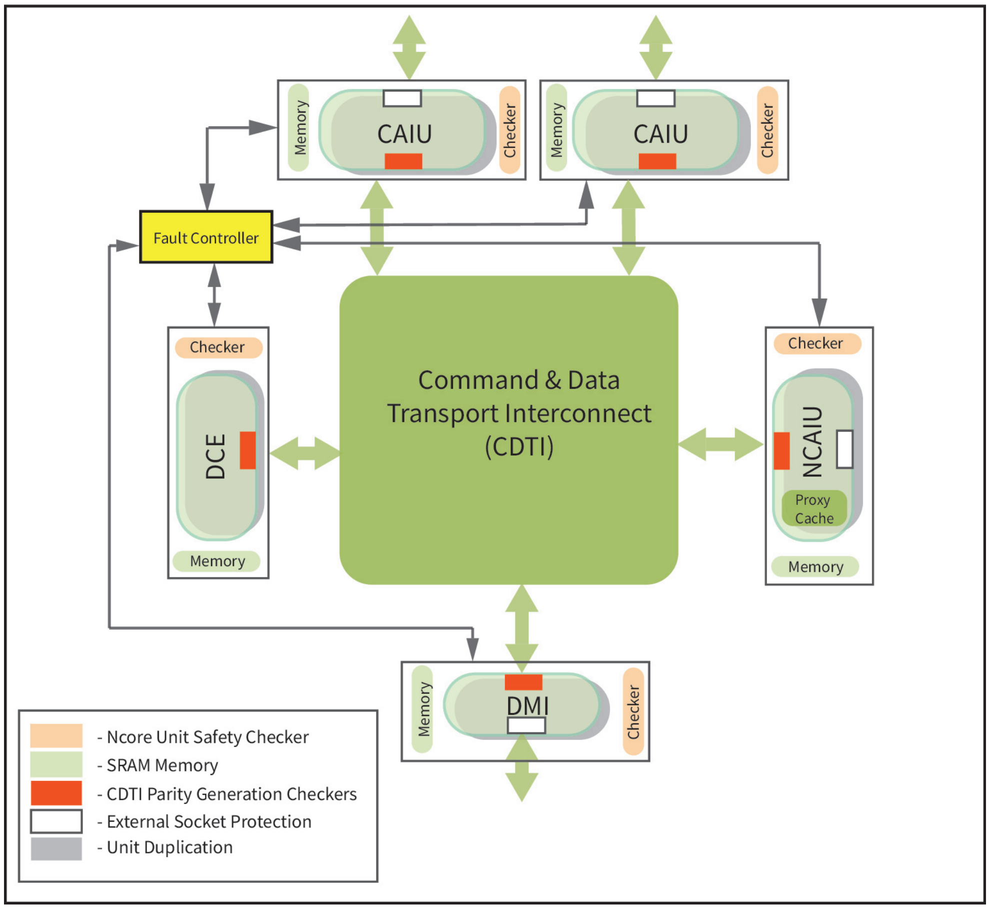
The resiliency feature in Ncore is optional, and when enabled, is implemented in addition to other configured Ncore features. Furthermore, Resiliency does not interfere or integrate with other Ncore features. In particular, control and status registers related to resiliency are independent of Ncore units control and status registers and they can be read and written by a secure master through a dedicated APB interface. Similarly, the fault reporting feature in resiliency is independent of the error reporting mechanisms.

Figure 18 below illustrates a high-level overview of the different elements of Ncore enabled with the resiliency option. There are six separate capabilities:

1. Component logic duplication. This capability replicates the logic for all units inside a Ncore component (i.e., CAIU, NCAIU, DCE, DMI, DII, and DVE) but shares a common copy of all internal memory structures. The replicated logic operates in lock-step with the original component logic and it can be configured to operate one to four clock cycles apart. Furthermore, additional "checking" logic - known as a Checker - is created to ensure that the original and replicated logic are operating in lock-step. As shown in Figure 18, there is a checker implemented for each unit of the Ncore component.
2. CDTI protection. Messages transported through the CDTI are protected. The protection type can be configured to either ECC or parity.
3. External interface protection. This capability enables a designer to source or terminate data protection signals on selected external CAIU, NCAIU, DMI, and DII interfaces. There are two parts to this feature. The first is the capability to augment the signals on an existing external interface with additional user-defined signals, and the second part is the creation of an "in-line" RTL module, per interface, that may be custom modified with user-created logic. This allows for the support of data and control signals protection schemes from third party IP providers.
4. Timeout reporting. This capability reports a fault if a transaction does not complete within the configured number of clock cycles.
5. SRAM memory protection. SRAM memory used by an Ncore component is protected for both address and data. The protection can be configured to be either ECC or parity.

6. Fault controller instantiation. This component aggregates faults reported by component checkers and drives an interrupt signal to notify the host system that a fault in the coherent interconnect has occurred and all the fault controller Control and Status registers are protected by parity. The fault controller hosts a register interface which allows software to identify the source of detected faults. Additionally, the fault controller embeds BIST to verify periodically that the logic of the fault reporting unit in each Ncore component is fully functional to ensure all faults are captured and reported correctly.

Figure 18. Ncore Elements with resiliency



4.2 Functional Safety Elements Description

Component Logic Duplication

Component logic duplication is applied to all Ncore components in a resiliency enabled interconnect. The two identical modules are identified as functional component and checker component. Component duplication, when enabled, applies to all Ncore components (CAIU, NCAIU, DCE, DII, DMI, and DVE). This feature creates logic for two identical RTL modules of the logical component, excluding any SRAM memories. If SRAM memories are used by a component, they must be protected either by ECC or parity, using the protection setting. The two identical modules are a functional component and checker component.

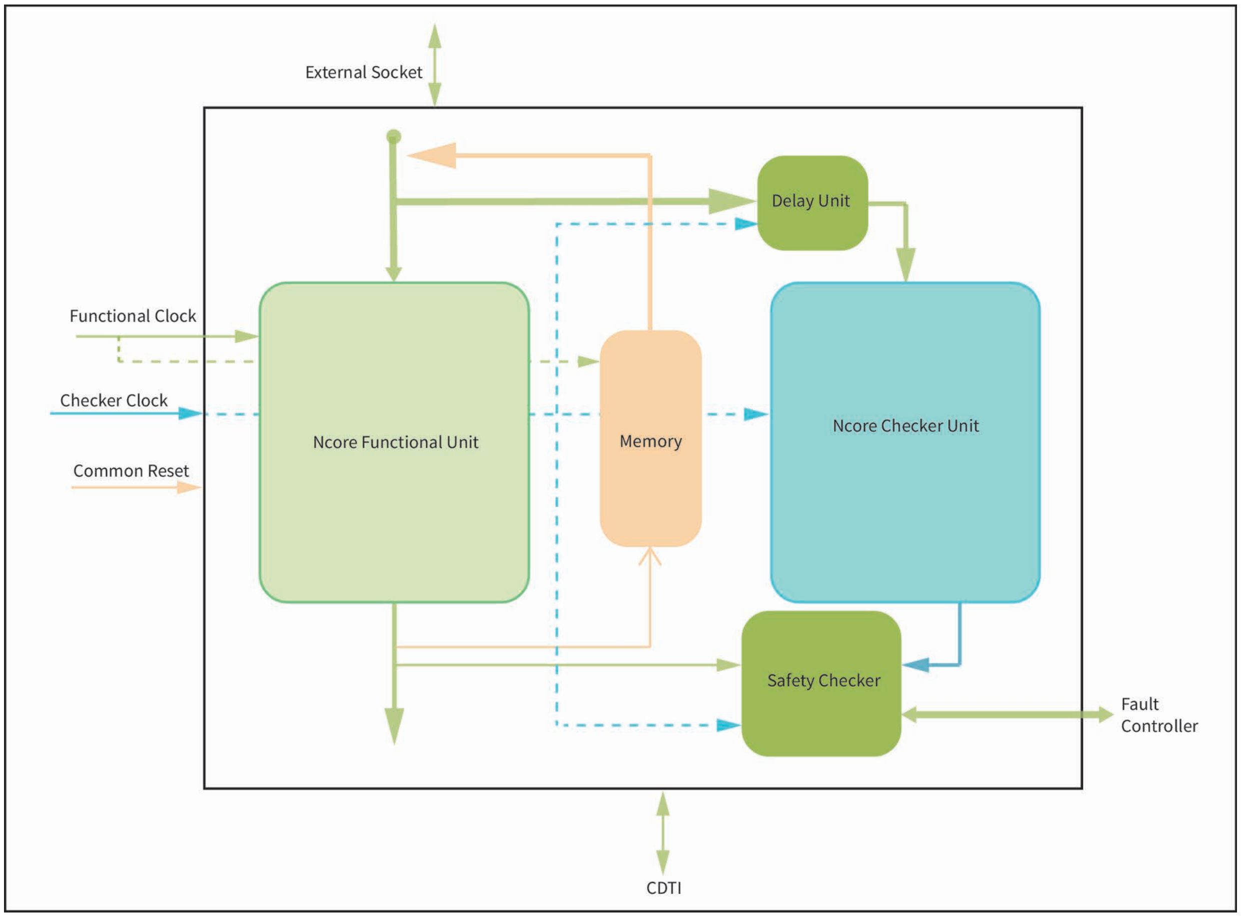
The checker component receives a version of the same inputs as the functional component, delayed by one to four clock cycles. The safety checker module receives the functional component's outputs and delays them by the same number of one to four clock cycles before comparing them with the checker component's outputs. Any discrepancy will be considered a fault. Faults detected will be logged and reported to the fault controller as mission fault. Once detected, the fault will remain sticky inside the checker component status register until a BIST sequence clears it.

To detect latent faults in the fault reporting logic, the safety checker module implements two identical comparator trees checking simultaneously that the output of the checker component is identical to the delayed version of the output of the functional component. A latent fault is reported if the outputs of the two comparator trees do not match.

There are two clock inputs at the top level of the duplicated component; the functional clock and the checker clock. The functional clock drives the functional component, memory, delay module and safety checker, while the checker clock drives the checker component. The two clocks should be identical and balanced. Having two different clock inputs enables the detection of problems in one of the clock trees.

The following figure illustrates the Ncore component duplication.

Figure 19. Ncore Component duplication



CDTI Message Protection

Messages transmitted over the CDTI are protected using either ECC or parity. The protection is applied to the data and address parts of the message, and also to some control information. The protection logic is duplicated as a byproduct of component duplication.

Each component generates protection information for messages entering the CDTI and checks protection information of all messages exiting the CDTI. Correctable ECC errors increment the correctable error counter. When this counter meets the set threshold, it generates a fault signal to the fault controller. Uncorrectable errors are considered as faults which are reported to the fault controller.

External Interface Protection

The External Interface Protection is an optional feature which can be enabled on individual interfaces. Once enabled, the tool permits the definition of a set of arbitrary signals at this interface, in addition to the CHI ACE or AXI signals. These signals are configurable in name, size, and direction. The intended use model is to allow protected agent interfaces that provide data and control signals ECC and/or parity to be properly defined.

During RTL generation, for this component, an empty RTL module with proper interface signals such as, clock, reset, user defined signals and appropriate fault signals are created.

Users may add their own protection logic in this empty RTL module and connect the logic to provided fault signal outputs. The fault signals are propagated to the fault controller. Ncore interface timing constraints apply on the interface signals to and from this module to internal Ncore logic. This module is also part of the component duplication.

SRAM Memory Protection

SRAM memory used by a component is protected both for address and data. The protection type can be configured to either ECC or parity. Correctable ECC errors increment the correctable error counter. When this counter reaches the set threshold, it generates a fault signal to the fault controller. Uncorrectable ECC errors are considered as faults which are reported to the fault controller.

Unattended activation protection

The unattended activation protection is enabled to prevent, in mission mode, the write accesses to the control registers below by a faulty system:

1. BIST control register in the fault controller.
2. Debug and Trace control registers in the component but the DCE.
3. All the component control registers through the APB port.

The system should follow the sequence below to execute the BIST and enable the APB port write access:

1. Set the unattended activation protection signal.
2. Write to the Ncore component control registers through the APB port or the fault controller BIST control register.
3. Reset the unattended activation protection signal.

NOTE: Same sequence should be used for enabling the Debug and Trace feature. However, this feature should only be enabled in the lab and always disabled in mission mode.

Control and Status register access protection

The access to the Control and Status registers can be protected by limiting the read and write operation to only one CAIU or NCAIU of the Ncore. In mission mode, additional CAIU/NCAIU can be promoted to access the Control and Status registers as well.

NOTE: An NCAIU with AXI interface and set in coherent mode cannot be selected to access the Control and Status registers.

Fault Controller

The fault controller monitors and logs faults reported by all safety checkers in the system. It has a Built-In Self Test (BIST) Finite State Machine (FSM) that communicates with the safety checkers and is used for key-on check of the fault reporting logic, as well as clearing mission and latent faults. The BIST FSM can detect stuck-at-faults. The BIST FSM is based on 9 states in addition to the idle/reset state:

1. Reset safety checkers fault Data Flip Flop (DFF). The fault controller expects all mission and latent faults from the safety checkers to go to '0'.
2. (a) Force the safety checkers mission fault functional comparator output to ‘1’. The fault controller expects all mission faults from the safety checkers to go to '1'.

   (b) Reset safety checkers fault Data Flip Flop (DFF). The fault controller expects all mission faults from the safety checkers to go to '0'.
3. (a) Force the safety checkers latent fault functional comparator output to ‘1’. The fault controller expects all latent faults from the safety checkers to go to ‘1’.

   (b) Reset safety checkers fault Data Flip Flop (DFF). The fault controller expects all latent faults from the safety checkers to go to ‘0’.
4. (a) Force the safety checkers mission and latent fault functional comparator output to ‘1’. The fault controller expects all mission and latent faults from the safety checkers to go to '1'.4.

   (b) Reset safety checkers fault Data Flip Flop (DFF). The fault controller expects all mission and latent faults from the safety checkers to go to ‘0’.

5. BIST indicates the Ncore components to conduct the Timeout test. The fault controller expects all Timeout uncorrectable errors from the functional and checker Ncore components to go to ‘1’.

Note: If no error is reported after 2^{14} clock cycles, a BIST error is asserted.

6. Reset Timeout fault DFF. The fault controller expects to see all Timeout uncorrectable errors from the Ncore components to go to '0'; and is now back to idle mode.

The BIST sequence can either be executed automatically or by single stepping via the fault controller BIST control register. The use model is as follows: after a fault detection, clear the fault by executing all the steps of the sequence automatically. If the BIST fails, it is possible to refine the diagnostic by executing each step one by one, to isolate which step fails. The BIST status register reports the steps that are completed by setting the corresponding bit to ‘1’ and the steps that failed by setting the corresponding bit to ‘1’.

The Fault controller must be assigned a clock identical to one of the Ncore components that will be the last clock to go off and first to go on. It should be in a power domain that will be the last domain to power down and the first power domain to power up. In an Ncore interconnect with multiple clock and power domains, where some components might go to a power down mode, the clock for the fault controller must be chosen so that it's in the power domain that will be the last one to go OFF and the first one to go ON.

NOTE: In the current implementation, it is not possible to run the BIST sequence if some power domains are down.

Therefore, in an Ncore interconnect with multiple power domains, if some of the domains are OFF, only fault detection will be provided, and the whole system must be reset to clear the fault.

All fault and BIST signals between fault controller and the safety checker are treated as asynchronous signals, and they are synchronized at both destinations. These signals may need level shifters, depending on the different power domains that go across in the system.

5 System Software Guidelines

The following sections discuss the various requirements for system software.

5.1 Reset and Initialization

Before processing any requests, each Ncore component must be transitioned into the operational state, which is achieved once the following requirements have been met:

- The supply voltage has been raised to an operational level
- The clock source is generating a stable clock signal
- The clock frequency is less than or equal to the maximum frequency allowed for the given operational voltage level
- The reset signal has been asserted and has been de-asserted after the required minimum number of cycles (16 cycles of the slowest clock in the system) if any of the following conditions occurred:
  - The supply voltage dropped below the logic or memory retention level
  - The clock source was not generating a stable clock signal
  - The clock frequency was greater than the maximum frequency allowed for a given operational voltage level

NOTE: The reset is asynchronous assert and synchronous de-assert.

Ncore components may be configured to have memories. These memories are initialized by an internal state machine on de-assertion of reset.

Once all Ncore Components are in operational state, the boot/initialization processes described below need to be followed.

Boot/Initialization

An agent connected to CAIU/NACIU may be designated as boot agent. This agent is expected to make the first request into the Ncore system. This request is expected to be a non-coherent request. The boot-up sequence to be followed is:

- Perform CSR access and go through the Ncore component discovery as described in "Ncore Component Discovery"
- Configure required memory and peripheral address spaces as described in "Address space configuration"

- Configure error registers as needed to enable required error detection and reporting as described in *Error Detection and Reporting configuration*.
- If preset, configure and enable all needed SMCs as described in “SMC configuration”.
- If preset, configure and enable all needed Proxy caches as described in “Proxy cache configuration”.
- Read register bit DCEUSFMAR[MntOpActv] to verify the DCE Snoop Filter RAM is initialized.
- Enable/disable EVENTI and EVENTO port as described in *WFE Wakeup Event Signaling Support* if present with XAIUUTCR, CAIUUTCR, and DCEUTCR EventDisable field.
- Transition CAIUs to Coherent state by the SYSCOREQ and SYSCOACK interface if present or Ncore CSRs to trigger the internal mechanism, this is described in *CAIU Coherence transition mechanism*.
- Optional: the timeout has already default value but depending of the traffic and design implementation, we recommend to customize correctly the timeout register value by step of 4,000 clocks cycles of:
  - xAIUTOCR: transaction timeout value. In case of huge AXI burst or high traffic on many agents, we recommend reviewing this value.
  - xAIUEHTOCR: XAIU Event Handshake Timeout value depending on your design reaction time.
  - xAIUSEPTOCR & xAIUSCPTOCR are internal Ncore timeout that do not need to be updated.
- Optional: Set up quality of service QOS as described in *Quality of Service Support*.

If the Ncore system is configured with a BR then the above steps can be part of the boot code, in this case BR can be the first access to load the boot code.

- Restriction: access to the Boot Region (BR) memory space only with non-coherent transactions.

Ncore Component Discovery

Ncore component discovery involves following steps at the designated CAIU/NCAIU. All read & write operations here are at the NRS base address. Complete register map configuration is available in the reports folder when RTL is generated.

- Read register at RP `0x0` and register address offset `0x0` i.e., at address bits [19:`0`] as zero; this will read xIDR register of the AIU providing the following information:
  - AIUs register page number (RPN)
  - Ncore register region identifier (NRRI), currently set to zero
  - Ncore unit ID, this uniquely identifies an Ncore unit within its type
- Read register GRBUNRRUCR at NRRI identified in the previous register, RP `0x`FF and address offset `0x`FF8. This register will identify the number of

different Ncore components which include AIUs, DCEs, DMIs, DIIs, and DVEs. RPs to Ncore component mapping is shown in (NRS section 3.2).

- Step through all Ncore component xIDR registers (register address offset `0x0`) and xINFO registers (register address offset `0`xFFC) starting form RPN `0`.

NOTE: Register access is restricted to NRS region access (xAIUNRSAR[0] set to 1 the agent can access to NRS region) , only secure access (for AXI, ACE request, AxPROT[1] attribute set to 0 and for CHI request, NS attributes attribute set to 0), 4-byte-aligned address, 4-byte size, non-coherent-address space, device-transaction access (as per AxCache bits on AXI/ACE, or SnpAttr bit on CHI). For CHI, the Order bits must indicate Endpoint order.

Any other access format will return an error. Access to an undefined RPN will return an error. Reserved registers within a valid RPN are treated as read as zero and write ignore.

Address space configuration

All AIUs and DCEs in the Ncore system must be configured with the same peripheral and memory address space configuration. This involves two main configuration steps as following:

- DMI interleaving configuration, this requires configuring the following two registers.
  - xAMIGR must be configured to select desired MIGS, by default MIGS `0` is selected.
  - xMIFSR must be configured to select desired interleaving function i.e., address bits used for DMI interleaving, by default option `0` is selected.
- GPAS configuration, this requires configuring following 3 registers per address region:
  - xGPRAR must be configured with attributes of the address region, these include.
- Home unit type (HUT), this specifies type of memory. System memory is mapped to a DMI and peripheral memory is mapped to a DII.
- Home unit identifier (HUI), this specifies target DMI or DII. In the case of DII the Nunit ID must be specified and in the case of DMI MIG number within the selected MIGS in xAMIGR must be specified.
- Size, this is specified as a binary number from `0` to 40 from which the region size is calculated as (size of IG) * 2^(size+12) bytes. Here size of IG is the number of DMIs within the selected MIG. It is always 1 for DII.
  - xGPRBLR must be configured with lower order address bits 43:12 of the base address of the region.
  - xGPRBHR must be configured with higher order address bits of the base address of the region.
- ReadID/WriteID, this is applicable only to NCAIUs and specifies ordering override. When these bits are set individually or together respective read or write channel Same AXI-ID ordering is ignored when transactions are issued

into the Ncore system by the AIU. AXI-ID ordering protocol at the Native interface is honored

- Policy, this is applicable only to NCAUs and specifies the ordering policy followed by DMI/DII at the bottom of the Ncore. For details on the policies, refer to the section, *Ordering Support*.
- If present, configure address translation registers <specify register name> in DMI/DII.

An example address space configuration is shown in Appendix A.

Error Detection and Reporting configuration

Following steps must be followed to configure error detection and reporting for each Ncore unit/component:

- Enable uncorrectable error detection by setting the appropriate error detection enable bits in xUEDR register of all Ncore units.
- If desired enable uncorrectable error interrupts by setting the appropriate error interrupt enable bit in xUEIR register of all Ncore units.
- Configure the correctable error control register xCECR to enable correctable error detection, interrupt and update the threshold as needed.
- If desired timeout error threshold can be updated or disabled at xTOCR register of all Ncore units.

Proxy cache configuration

Following steps must be followed to configure and enable proxy cache:

- Read register XAIUPCISR to verify both tag and data RAM are initialized.
- Configure register XAIUPCTCR to enable the proxy cache in two steps:
  - Enable cache look up by setting LookupEn
  - Enable cache allocation by setting AllocEn

SMC configuration

Following steps must be followed to configure and enable SMC.

- Read register DMIUSMCISR to verify the tag RAM is initialized.
- Configure register DMIUSMCAPR to specify desired cache allocation policy.
- Perform an "Initialize all entries" maintenance operation for the data array by writing the appropriate values to DMIUSMCMCR.
- Read Maintenance Operation Active bit in DMIUSMCMAR until the bit equals zero.
- If the DMI was configured with scratchpad support, then as desired enable the scratchpad as described in “Scratchpad configuration”.
- If the DMI was configured with way partitioning capability, configure registers DMIUSMCWPCRx to set up way partitioning.
- Configure register DMIUSMCTCR to enable the SMC in two steps:
  - Enable cache look up by setting LookupEn
  - Enable cache allocation by setting AllocEn

Scratchpad configuration

Following steps must be followed to configure and enable the scratchpad:

- Configure DMIUSMCSPBR0/1 with desired scratchpad base address.
- Configure DMIUSMCPCR0 with desired number of ways minus 1 to be used as scratchpad. Maximum value is limited by the total number of ways configured in the cache.
- Configure DMIUSMCPCR1 with scratchpad size. The size must be specified in number of cache lines and must be an integer multiple of the number of sets in the cache, times the desired number of ways for scratchpad size, minus 1.
- Configure DMIUSMCPCR0 to enable scratchpad by setting ScPadEn.

CAIU Coherence transition mechanism

There are two ways to transition a CAIU into coherent state:

CHI CAIU can transition to coherent state via the SYSCOREQ and SYSCOACK protocol signaling as specified by ARM.

CHI CAIU agent that does not have the SYSCO singling, ACE CAIU, Proxy cache NACIU and DVM capable NCAIU can transition to coherent state via the provided NCORE CSRs in the AIUs.

- Read the xTAR register to confirm the current state of the AIU.
- Set SysCoAttach in xTCR to start the transition of the AIU into coherent state.
- Poll the xTAR register SysCoAttached field to confirm that the AIU is in the coherent state.

Transition to the detached state using CSR must follow the following steps:

- Read the xTAR register to confirm the current state of the AIU.
- Set SysCoAttach to '0' in xTCR to start the transition of the AIU into detached / Non-coherent state.
- Poll the xTAR register SysCoAttached field to confirm that the AIU is in the detached/Non-coherent state.

5.2 Q channel clock gating sequence

The Q channel clock gating sequence is as follows:

1. Software must go through the processes of turning off all traffic to the Ncore components that are within the clock domain and are going to be clock gated. Steps included are:
    a. Stop all coherent traffic at the native interface of the target Ncore components.
    b. Flush and disable all caches above the native interface or within the target Ncore components.
    c. Any CAIUs, Proxy cache and DMV capable NCAIUs must be transitioned to non-coherent/detached state either via SYSCOREQ/SYSCOACK interface signaling or via xTAR/XTCR registers.
    d. Stop all at the native interface of the target Ncore components.
2. Trigger the CMU (Clock gating Management Unit) to send a Q channel request which includes the following:
    a. REQn is asserted on the Q Channel Interface.
    b. Ncore component will assert ACCEPTn and accept the request once it is not busy (no active transactions are pending in the Ncore component).

The software or CMU is required to implement a timeout counter to cover cases where the Ncore component may not accept the power down request. In this case ACCEPTn will not be asserted, at timeout REQn must be de-asserted (Clock gating aborted) and that Ncore component stays in up state.

In the Clock gated state the Ncore components will not block any transactions and have no special behavior. Depending on the system design on successful completion of Q channel request the Ncore subsystem clock can be turned off. The ACTIVE signal indicates if the block is busy and this signal can toggle at any time during normal operation.

The wake-up sequence is as follows:

1. Enable the clock.
2. De-assert REQn.
3. Go through the process of transitioning any CAIUs, Proxy cache, and DVM capable NCAIUs to coherent state.
4. Start normal traffic.

Initial system power up with Ncore clock domains in state is not supported.

A Appendix

A.1 System Address Map

This section describes an example setup for Ncore address space. Before proceeding further, read *Addressing and Memory Regions* and *Interleaving* capabilities of Ncore. The following build time configurations are assumed.

Table 20.

Description of parameter	Value	Notes
Number of GPRAS	6	
DCE interleaving	30:39	4 way interleaving at 512 MB
Number of DMIs	6	They are named DMI0 to DMI5
MIGS (option 1)	{[DMI0, DMI1, DMI2, DMI3], [DMI4, DMI5]}	2 groups, first one is 4 way interleaved second one is 2 way interleaved.
MIGS (option 2)	{[DMI0, DMI1],	
[DMI2, DMI3],		
[DMI4],		
[DMI5]}	4 groups, first 2 are 2 way interleaved and the last 2 are single DMIs - i.e., no interleaving	
2 way interleaving (options 1)	9	512Byte interleaving
2 way interleaving (options 2)	8	256Byte interleaving
4 way interleaving (options 1)	7,6	64B interleaving
4 way interleaving (options 2)	13,12	4KB interleaving
Number of DIIs	1	
Boot region base address	0x0	
Boot region map	DII	Mapped to the single DII in the system

For simplicity CSR address space and ordering configurations are not shown. This example is for CAIU that does not consider the ordering configuration and the bits must be set as zeros. The address map registers for a single GPRAS in detail is shown in Table 21. For a detailed description of each field, refer to the CSR description document. Figure 22 includes a brief description.

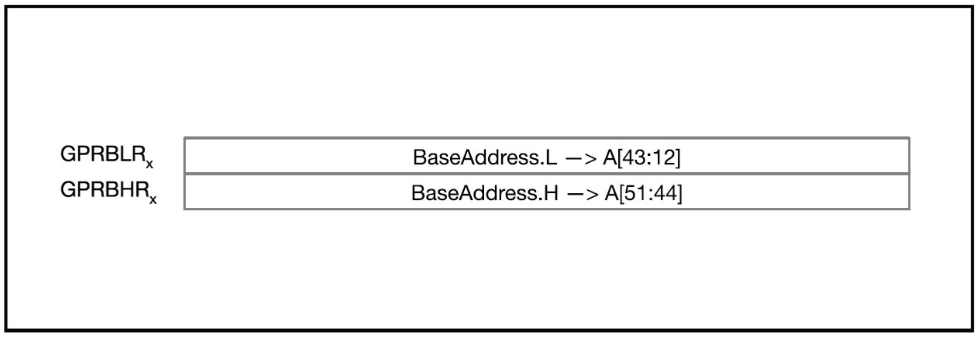
Table 21. GPRAS Address

Bit	Name	Description
31	Valid	This bit indicates if the region is valid. 0: Invalid mapping 1: Valid region mapping read-write none 0x0
30	HUT	This bit indicates the Home Unit Type. 0: System Memory 1: Peripheral Storage read-write none 0x0
29:25	Rsvd	Reserved

24:20	Size	Size = (IG) x $2^{(\text{Size} + 12)}$ bytes
		IG = 1, 2, 4, or 9

19:14	Rsvd	Reserved
13:9	HUI	This bit indicates the Home Unit ID: DII -> 0..number of (DII -1) DMI -> MIG number withing the MIGS
8	Rsvd	Reserved
7:6	NSX	This bit indicates the NSX security level required to access this region. NSX [0] is mapped to NS(non-secure). NSX [1] is mapped to NSE(non-secure extension, reserved for future use) read-write none 0x0
5	NC	Non-coherent. 0: Use coherent mode as indicated by incoming transaction. 1: Enforce non-coherent transaction read-write none 0x0
4:3	Policy response	Policy response policy read-writer 0x0
2	WriteID	Write ID no ordering by ASID read-write none 0x0
1	ReadID	ReadID No ordering by ARID, read-write none 0x0
0	Hazard Ignore	This field indicates a binary number from 0 to 31 from which the region's size is calculated as (Size of IG)*2 ^(Size + 12) bytes read-write none 0x0.

Figure 22.GPRAS Address



The example has 6 GPRAS out of which first 5 are configured as follows

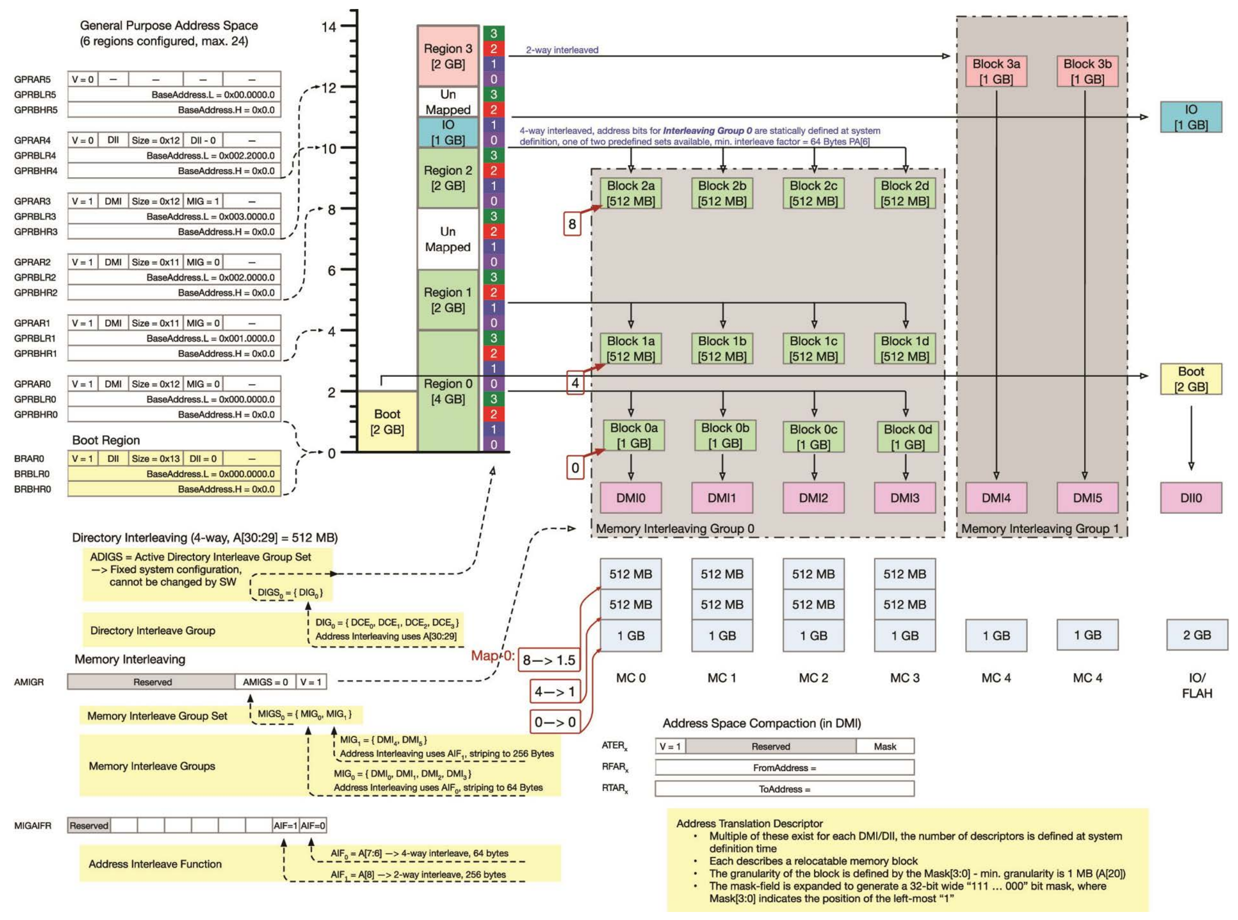
- GPRAS0
  - Mapped to MIG 0 which is four ways interleaved and uses option 0 of 64byte interleaving
  - The memory size is specified as 4GB, size value in CSR is 0x12 which calculates to 1GB multiplied by 4 way interleaving.
  - Start address is 0
  - This region remaps Boot address space
- GPRAS1
  - Mapped to MIG 0 which is four ways interleaved and uses option 0 of 64byte interleaving
  - The memory size is specified as 2GB, size value in CSR is 0x11 which calculates to 512MB multiplied by 4 way interleaving.
  - Start address is 4GB
- GPRAS2
  - Mapped to MIG 0 which is four ways interleaved and uses option 0 of 64byte interleaving
  - The memory size is specified as 2GB, size value in CSR is 0x11 which calculates to 512MB multiplied by 4 way interleaving.
  - Start address is 8GB
- GPRAS3
  - Mapped to MIG 1 which is tow ways interleaved and uses option 1 of 256byte interleaving
  - The memory size is specified as 2GB, size value in CSR is 0x12 which calculates to 1GB multiplied by 2-way interleaving.
  - Start address is 12GB
- GPRAS4
  - Mapped to DII 0, Note that DIIs cannot be interleaved
  - The memory size is specified as 1GB, size value in CSR is 0x12 which calculates to 1GB.
  - Start address is 10GB
  - This region remaps Boot address DII to other IO space

In addition, the figure also shows address compaction mapping for DMI0 where the base address 0 is mapped to 0, 4GB is mapped to 0x1GB, and 8GB is mapped to 1.5GB giving a total address space of 2GB. Figure 24 illustrates the address translation mechanism.

The address map for the example system is shown in Figure 23.

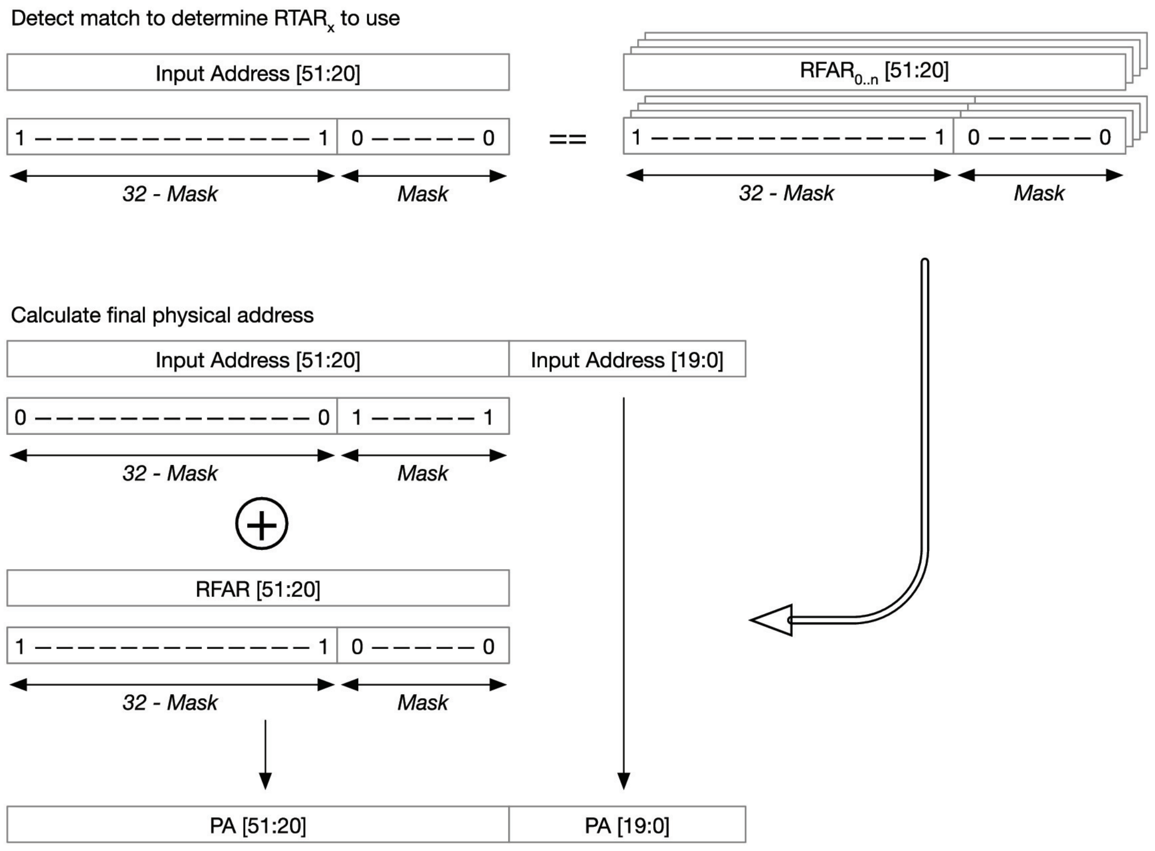
Descriptions in yellow are build time settings like boot region, interleaving for DCE, MIGS etc. In this example the Boot region is overwritten by a different address region after boot up. As shown in the following figure, the boot region is mapped to a DII, it can be mapped to a DMI provided that it will always have to be one of the memory interleaving groups within MIGS0.

Figure 23.Address Map


General Purpose Address Space
(6 regions configured, max. 24)
GPRAR5 V = 0 – – – –
GPRBLR5 BaseAddress.L = 0x00.0000.0
GPRBHR5 BaseAddress.H = 0x0.0
GPRAR4 V = 0 DII Size = 0x12 DII - 0 –
GPRBLR4 BaseAddress.L = 0x002.2000.0
GPRBHR4 BaseAddress.H = 0x0.0
GPRAR3 V = 1 DMI Size = 0x12 MIG = 1 –
GPRBLR3 BaseAddress.L = 0x003.0000.0
GPRBHR3 BaseAddress.H = 0x0.0
GPRAR2 V = 1 DMI Size = 0x11 MIG = 0 –
GPRBLR2 BaseAddress.L = 0x002.0000.0
GPRBHR2 BaseAddress.H = 0x0.0
GPRAR1 V = 1 DMI Size = 0x11 MIG = 0 –
GPRBLR1 BaseAddress.L = 0x001.0000.0
GPRBHR1 BaseAddress.H = 0x0.0
GPRAR0 V = 1 DMI Size = 0x12 MIG = 0 –
GPRBLR0 BaseAddress.L = 0x000.0000.0
GPRBHR0 BaseAddress.H = 0x0.0
Boot Region
BRAR0 V = 1 DII Size = 0x13 DII = 0 –
BRBLR0 BaseAddress.L = 0x000.0000.0
BRBHR0 BaseAddress.H = 0x0.0
14
12
10
8
6
4
2
0
Region 3 [2 GB]
Un Mapped
IO [1 GB]
Region 2 [2 GB]
Un Mapped
Region 1 [2 GB]
Region 0 [4 GB]
Boot [2 GB]
3 2 1 0
2-way interleaved
4-way interleaved, address bits for *Interleaving Group 0* are statically defined at system definition, one of two predefined sets available, min. interleave factor = 64 Bytes PA[6]
Block 3a [1 GB]
Block 3b [1 GB]
IO [1 GB]
Block 2a [512 MB]
Block 2b [512 MB]
Block 2c [512 MB]
Block 2d [512 MB]
8
Block 1a [512 MB]
Block 1b [512 MB]
Block 1c [512 MB]
Block 1d [512 MB]
4
Boot [2 GB]
Block 0a [1 GB]
Block 0b [1 GB]
Block 0c [1 GB]
Block 0d [1 GB]
0
DMI0
DMI1
DMI2
DMI3
DMI4
DMI5
DII0
Memory Interleaving Group 0
Memory Interleaving Group 1
512 MB
512 MB
1 GB
MC 0
MC 1
MC 2
MC 3
MC 4
MC 4
2 GB
IO/ FLAH
Map 0:
8—> 1.5
4—> 1
0—> 0
Directory Interleaving (4-way, A[30:29] = 512 MB)
ADIGS = Active Directory Interleave Group Set —> Fixed system configuration, cannot be changed by SW
$DIGS_0$ = { DIG_0 }
Directory Interleave Group
DIG_0 = { DCE_0, DCE_1, DCE_2, DCE_3 } Address Interleaving uses A[30:29]
Memory Interleaving
AMIGR Reserved AMIGS = 0 V = 1
Memory Interleave Group Set
$MIGS_0$ = { MIG_0, MIG_1 }
Memory Interleave Groups
MIG_1 = { DMI_4, DMI_5 } Address Interleaving uses AIF_1, striping to 256 Bytes
MIG_0 = { DMI_0, DMI_1, DMI_2, DMI_3 } Address Interleaving uses AIF_0, striping to 64 Bytes
MIGAIFR Reserved AIF=1 AIF=0
Address Interleave Function
AIF_0 = A[7:6] —> 4-way interleave, 64 bytes
AIF_1 = A[8] —> 2-way interleave, 256 bytes
Address Space Compaction (in DMI)
$ATER_x$ V = 1 Reserved Mask
$RFAR_x$ FromAddress =
$RTAR_x$ ToAddress =

Address Translation Descriptor
- Multiple of these exist for each DMI/DII, the number of descriptors is defined at system definition time
- Each describes a relocatable memory block
- The granularity of the block is defined by the Mask[3:0] - min. granularity is 1 MB (A[20])
- The mask-field is expanded to generate a 32-bit wide "111 … 000" bit mask, where Mask[3:0] indicates the position of the left-most "1"

Figure 24. Address Translation



A.2 Producer consumer example scenario

A simple producer-consumer scenario is described here. The producer-consumer scenario assumes that the Data and the Flag are sent with the same AxID. This is setup by configuring two GPRARs at producer with different order settings, one with relaxed order setting for data and another with strict order setting for Flag/MSIX Interrupt.

Relaxed Order setting for data:

Write ID override set as '1'

Read ID override set as '1'

Policy set as '10' "Relaxed order"

Strict Order setting for flag:

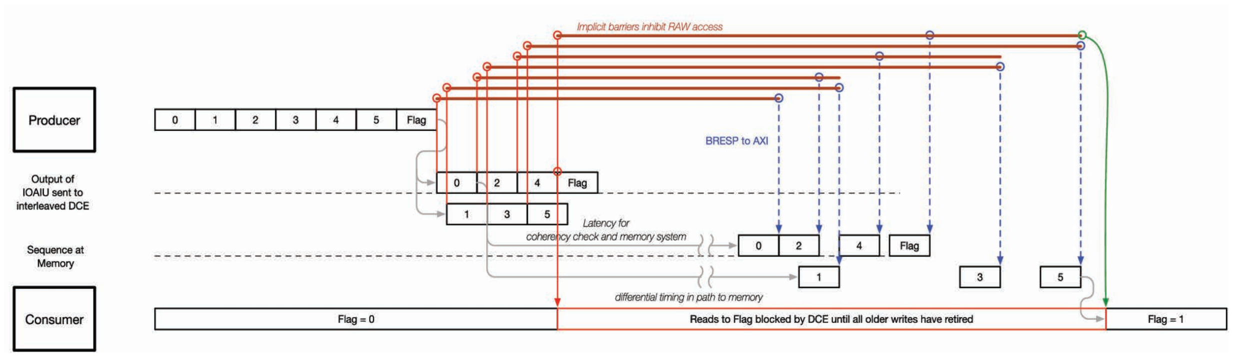
Write ID override set as '0'

Read ID override set as '0'

Policy set as '11' "endpoint order"

The above distinction with different GPRAs guarantees that data is processed as soon as possible by ignoring the AxID ordering for internal Ncore processing purposes, this achieves maximum possible bandwidth for data. While the Flag/MSIX is processed after the data has reached point of serialization as Flag will hit the strict GPRA. The worst case scenario with race conditions between two interleaved DCEs and memory path, is shown in Figure 25. As described in the figure Ncore uses implicit barriers that inhibit RAW accesses and guarantees that the consumer sees the new data once it sees that the flag is updated.

Figure 25.Order Write


Implicit barriers inhibit RAW access
Producer
0
1
2
3
4
5
Flag
BRESP to AXI
Output of IOAIU sent to interleaved DCE
0
2
4
Flag
1
3
5
Latency for coherency check and memory system
Sequence at Memory
0
2
4
Flag
1
3
5
differential timing in path to memory
Consumer
Flag = 0
Reads to Flag blocked by DCE until all older writes have retired
Flag = 1