Ncore 3 System Architecture Specification rev 0.51

Product release: 3.4

Rev: 0.51, April 25, 2023

Release Information

Version	*Editor*	*Change*	*Date*
0.1	MF	Initial Document template created	10/24/2019
0.2	BB	Document template updated based on feedback	01/26/2023
0.3	SD	▪ Initial content based on: - Ncore 3 system arch specification v0.82 - Ncore 3.2 system arch specification ▪ Features section updated with link to JAMA requirements ▪ Performance monitoring section updated with 3.4 version rev 0.8 ▪ Error section updated with 3.4 version rev 0.84 ▪ Address map section updated with CSR access 3.4 version rev 0.6 ▪ QoS section created including 3.4 QoS improvement rev0.86 ▪ Connectivity architecture section updated with 3.4 version rev 0.86 ▪ Legato description added ▪ Multiported NCAIU section added with 3.4 version rev 0.72 ▪ Proxy cache section added with 3.4 version of Ncore 3 System Architecture Specification rev 0.51 ▪ System messages section added with 3.4 version of NCore 3 SysCmd architecture specification rev 0.56 ▪ Transport layer section updated with Width- & Rate-Adapter Support specification rev 0.5 ▪ Sharer promotion section added with Ncore Sharer Promotion Architecture specification rev0.82 ▪ PCIe support section added with PCIe interoperability and optimization specification ▪ End to end credit section added with Skid buffer architecture specification ▪ Section 5 updated with Adress decode enhancements specification ▪ Section 12 Security updated with CONC-10553: NS bit don't care for transactions to CSR and boot region ▪ Section 2 Ncore system description updated with CONC-9571: Engineering Version Id Register ▪ Adding TSR technical safety requirement from JAMA	02/7/2023
0.5	BB	Formatting; candidate for first review	04/03/2023
0.51	SD	Adding link to section answering TSRs	04/19/2023
Legend:	BB MF SD Xx	Bernard Bonardi Michael Frank Said Derradji Whoever else edited this document	

Confidential Proprietary Notice

document with Arteris IP, then the click-through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the agreement shall prevail.

Confidentiality Status

This document is Confidential and Proprietary. This document may only be used and distributed in accordance with the terms of the agreement entered into by Arteris IP and the party that Arteris IP delivered this document to.

Product Status

The information in this document is ***Preliminary***.

Web Address

http://www.arteris.com

- ***Table of Contents***

Table of Figures

Table of Tables

Preface

This preface introduces the Arteris® Network-on-Chip Hierarchical Coherency Engine Architecture Specification.

About this document

This technical document is for the Arteris Network-on-Chip Hierarchical Coherency Engine Architecture. It describes the subsystems and their function along with the system's interactions with the external subsystems. It also provides reference documentation and contains programming details for registers.

Product revision status

TBD

Intended audience

This manual is for system designers, system integrators, and programmers who are designing or programming a System-on-Chip (SoC) that uses or intend to use the Arteris Network-on-Chip Hierarchical Coherency System (ANoC-HCS).

Using this document

TBD

Glossary

The Arteris© Glossary is a list of terms used in Arteris© documentation, together with definitions for those terms. The Arteris© Glossary does not contain terms that are industry standard unless the Arteris© meaning differs from the generally accepted meaning.

Typographic conventions

italic

Introduces special terminology, denotes cross-references, and citations.

bold

Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

monospace

Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

monospace italic

Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name. monospace italic Denotes arguments to monospace text where the argument is to be replaced by a specific value. monospace bold Denotes language keywords when used outside example code.

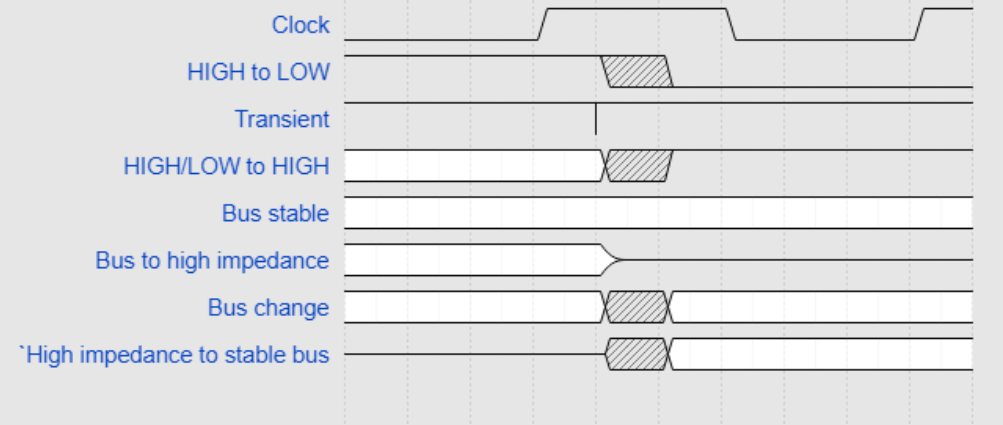
SMALL CAPITALS

Used in body text for a few terms that have specific technical meanings, that are defined in the Arteris® Glossary. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Timing diagrams

The following figure explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



Signals

The signal conventions are:

Signal level

The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means:

- HIGH for active-HIGH signals.
- LOW for active-LOW signals.

Lowercase n

At the start or end of a signal name denotes an active-LOW signal.

Additional reading

This book contains information that is specific to this product. See the following documents for other relevant information.

History of the World II, Mel Brooks.

1 Introduction

This chapter provides an overview of the Arteris Network-On-Chip Hierarchical Coherency Engine and its features.

1.1 About

The Arteris Network-On-Chip Hierarchical Coherency System (ANoC-HCS) is a high-performance, scalable, coherent NoC. It supports all types of traffics, including cache coherent, I/O coherent, and non-coherent traffic. It includes a transport layer and provides full state of the art cache coherency protocol layer to connect heterogeneous compute or IO nodes.

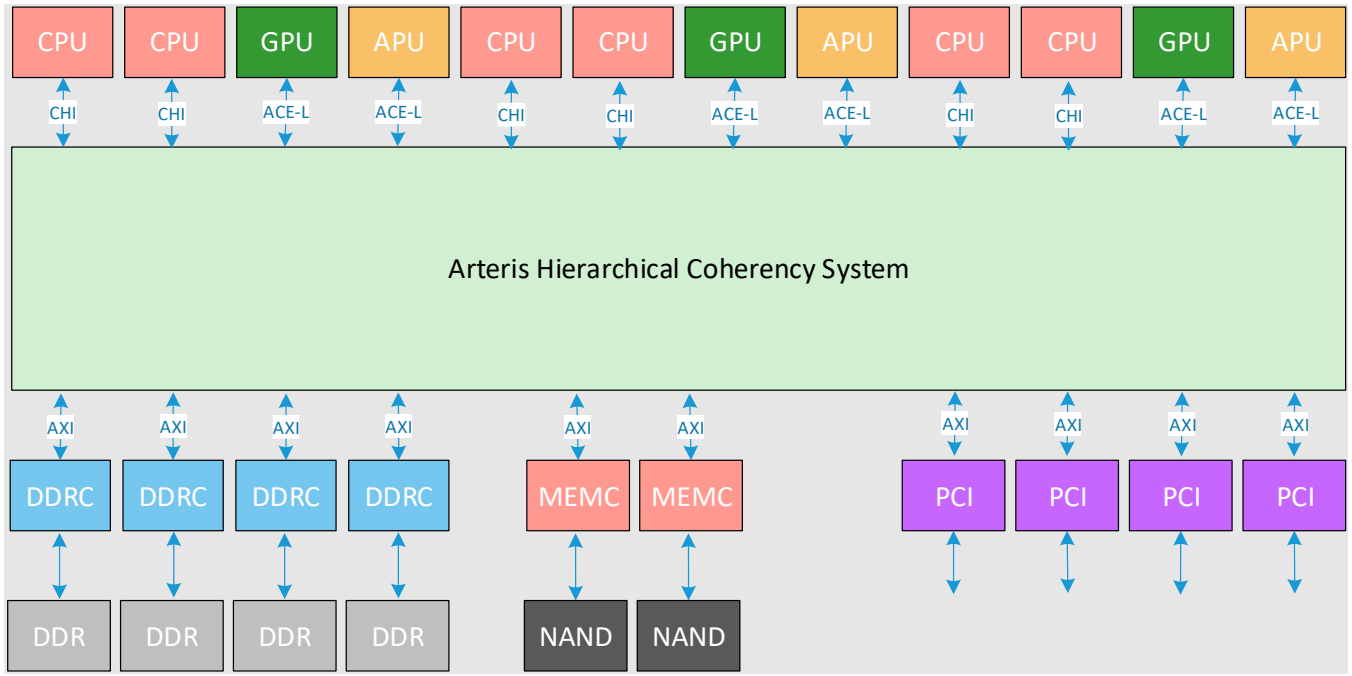


FIGURE 1-1 ANOC-HCS BLOCK DIAGRAM

The NoC is composed of various components, the cache protocol layer units are called Concerto units and the transport layer is also called CDN for control and data network. The NoC is highly configurable and the selected components depend of the SoC requirements.

1.2 Features

1.2.1 Compliance

Ncore 3.4 is compliant with the following interface standards, coherency architecture and related functionality.

It provides a native interface with CHI Protocol: Ncore supports Arm AMBA 5 CHI Issue A and B implementing the base features plus the following:

- Distributed Virtual Memory (DVM) transactions
- Exclusive transactions
- Far Atomic operations
    - Ncore 3 supports only cacheable coherent and non-coherent atomics to normal memory connected below a DMI. Ncore does not support atomic transactions to non-cacheable device memory or cacheable normal memory below a DII.
- Cache-stashing for data locality
- Direct data transfer for latency
- Separate Read Data and Comp Responses for direct memory transfer (DMT)
- Combined CompAck with WriteData
- Data width option 128/256 bits
- Up to 32 User bits (RSVDC field as User Placeholder)

does not support the following features:

- Retry mechanism
- Write stashes from CHI B
- CHI data check
- Eviction of clean lines
- Barrier transactions
- Big Endian support

It provides a native interface with AXI4/ACE/ACE-Lite/ACE-Lite-E Protocols: Ncore 3.4 supports Arm AMBA® AXI4 and ACE Protocol Specification issue E[1], implementing the following options:

- Data width option 64/128/256 bits
- Up to 32 User bits (User Placeholder)

does not support the following features

- Untranslated transaction identifiers
- Non-secure Access identifier

[1] ACE-Lite-E refers to the ARM AXI-ACE 0022H specification

- Near Atomic transactions
- Barrier transactions
- Persist CMO write transactions
- Big Endian support
- burst limitation with native interfaces AXI, ACE-Lite, ACE-Lite E and ACE.
    - Narrow bursts are limited to only a single beat
    - Fixed bursts are not supported
    - Wrap non modifiable burst which cross 64-byte boundary are not supported
- ACE-Lite E StashOnceShared and StashOnceUnique are not supported for non-shareable domain
- ACE-Lite E data check and poison is not supported
- ACE-Lite E user loop-back is not supported
- ACE-Lite E low power signaling is not supported
- ACE-Lite E device and non-cacheable atomics are not supported

With Distributed Virtual Memory: Ncore 3 **s**upports Arm's Distributed Virtual Memory architecture, including support for System MMUs (SMMUs), both Table Look-aside Buffer Unit (TBU) and Translation Control Unit (TCU). Ncore 3 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.

With Arm TrustZone: Ncore 3 supports the Arm TrustZone technical security model by handling the security bit when the TrustZone is enable on the interface. It supports Arm TrustZone technology specification used by some Arm processors that run secure and non-secure operating system on the same core. Indeed, the secure vs. non-secure indication is propagated and stored throughout the Ncore 3 system.

With Arm access permission: Ncore 3 supports legacy access permission signals. AxPROT[0] and AxPROT[2] are passed through within Ncore 3 for masters with AXI, ACE-Lite, ACE-Lite E, and ACE. In the case of CHI masters, the signal is tied off.

With Arm Q-channel: Ncore 3 implements power management features with two levels of clock-gating and ARM Q-channel support. The implementation shall comply with the AMBA® Low Power Interface Specification issue C while the P-channel is not supported. The two levels of clock gating are configurable. The first level of clock gating shall turn on and off local clock and it can be enabled during the synthesis while the second level of clock gating shall turn on and off the clock for an entire unit and it can be enabled during the configuration creation via a parameter.

1.2.2 Ncore features

- Ncore 3 supports simultaneous coherent inter-operation of ACE based and CHI-based processors or accelerators via front-end interfaces units with its proprietary messaging protocol Concerto revision C
- Ncore 3 provides a generic memory interface via DMI unit
- Ncore 3 provides a generic interface to peripheral via DII unit

- Ncore 3 provides an optional system memory cache with the following functionalities:
  - Atomic Engine to perfom atomic operations
  - Enhanced and flexible caching policies
  - Per-way SRAM mode for data arrays
  - Per-source way-reservation for allocation of data
- Ncore 3 supports an optional proxy cache at interface units for AXI, ACE-Lite and ACE-Lite-E
- Ncore 3 provides Microarchitecture enhancements to fully support Producer-Consumer ordering across the system, offering optimized PCIe compatibility.
- Ncore 3 provides an efficient and high performance transport interconnection (called Legato) for carrying Concerto C messages:
  - Low latency and high bandwidth
  - Deadlock-free routing of protocol messages
- Ncore 3 provides End-to-End Credit Management
- Ncore 3 also provides a dedicated transport interconnection solution for accessing Configuration and Status Registers in the system
- Ncore 3 provides a flexible address map facility
  - To Enable the implementation of a “single system image” covering all storage spaces in the system
  - To access both coherent and non-coherent regions in the system
  - To support multiple non-contiguous address ranges, including those accessed non-coherently, to be mapped to a single memory controller
  - To offer optional programmable registers to specify the map, which enable the possibility to implement the system address map at run time.
  - To allow the physical address space to be packed to a smaller footprint allowing use of fewer address bits and reducing board space occupied by memory devices
- Support for System Event and Coherency Messaging
- Ncore 3 provides Quality of Service with up to (QoS)

1.2.3 System/Chassis level features

Ncore 3 provides system/chassis level functionalities.

Trace and debug : Ncore 3 provides debug and trace features to allow to the customer to capture the Ncore 3 internal transactions. The functionalities below are supported:

- 1. Label: Identify and select transactions based on a set criterion like address, target etc.
- 2. Order: Identify temporal ordering of transactions, using a time stamp.
- 3. Capture: Take a snapshot of the labeled transaction.
- 4. Aggregate and access: Accumulate all captured transactions at a central location and provide SW access.
- 5. Analysis: Software is required to access the captured information and analyze it.

The debug and trace registers data shall be readable even if the system is hanged: a debug APB port is implemented to enable an alternate way to access all the Ncore 3 configuration and status registers.

Performance monitoring: Ncore 3 implements a performance monitoring framework and capabilities such as performance counters to determine the system performances. Performance counter units shall be added to the functional units (DCE, DVE, IOAIU, CHIAIU, DMI, DII blocks) and support tracking and reporting up to 31 events.

1.2.4 List of features added to NCore version 3.2

- Resiliency Enhancements:
    - Concerto mux enhancement: Adding missing bits into ECC calculation
    - Memory address protection
- CSTI network enhancement:
    - Limit CSTI access from CDTI initiator
- Processor Event support:
    - SysCo/SysAck messages
- Performance enhanced for PCIe traffic

1.2.5 List of features added to NCore version 3.4:

- Support for security and coherency based on GPRS
    - Security extension for the security level required to access GPR address regions
    - GPRS Access through single CAIU
    - Secure CSR update
- NCAIU capabilities:
    - Multiport NCAIU support
    - No interleaving on NCAIU data response
    - NCAIU interleaving only at 64B boundaries
    - NCAIU with proxy cache non-coherent access support
- Proxy cache capabilities:
    - Owner transfer
    - Sharer promotion
- DMI:
    - WTT/RTT threshold availble for better QoS
    - Optional Read data interleaving support
- Performance statistics enhancement
    - Periodic static collecting & reporting infra structure
    - Transport layer optimization
    - Periodic proxy cache statistics
    - Filter statistics

1.3 Requirements

TABLE 1-1: LINKS TO SECTIONS THAT CAPTURE THE TECHNICAL ANSWERS TO THE PRDs

PRD name	Link to section
NCR34-PRD-11 CHI Protocol Support	Compliance
NCR34-PRD-12 AXI4/ACE4/ACE5-Lite-E AIU Support	Compliance
NCR34-PRD-13 AXI AIU with Proxy Cache	Proxy cache
NCR34-PRD-14 DII and internal non-coherent connectivity	Distributed IO Interface (DII)
NCR34-PRD-15 Support of Flexible Coherent Memory Map	General Purpose Address Space
NCR34-PRD-16 System Clock target Frequency of 1.6GHz	Timing goals
NCR34-PRD-17 Power Management features	Power management
NCR34-PRD-19 Debug and Trace Features	Trace and Debug
NCR34-PRD-20 Performance Monitoring Feature	Performance Monitoring
NCR34-PRD-23 System Memory Cache Capabilities into Ncore	System cache
NCR34-PRD-24 Read Data Forwarding into Ncore	Transaction flows
NCR34-PRD-27 Support for different data widths	Topology optionsWidth Adapters
NCR34-PRD-28 Separate Data and Virtual Address Table Coherency	Distributed Virtual Memory System Engine (DVE)
NCR34-PRD-29 Ncore Security features	Security
NCR34-PRD-230 Distributed Memory Interface unit (DMI)	Distributed Memory Interface (DMI)
NCR34-PRD-236 QoS support in Ncore	Quality of service (QoS)
NCR34-PRD-202 Increase CAIU, IOAIU & DMI outstanding transactions amount	Performance goals
NCR34-PRD-203 PCIe ordering changes	PCI Express support
NCR34-PRD-204 Memory generic interface support	Distributed Memory Interface (DMI)
NCR34-PRD-205 Coherency requests support for power down	Attach/Detach Protocol
NCR34-PRD-206 Exclusive events reporting	Event Messaging Architecture
NCR34-PRD-208 Error handling feature enhancement	Error handling
NCR34-PRD-212 Increase of CHI-AIU and AXI-AIU units user bit limit	Compliance
NCR34-PRD-213 SRAM address protection	SRAM protection
NCR34-PRD-214 Limiting CSR access to one unit	Ncore CSTI Network
NCR34-PRD-215 Proxy cache AXI AIU interleaving only at 64 Byte boundaries	Read data interleaving mode
NCR34-PRD-216 Proxy cache support for owner transfer	Owner ship transfer
NCR34-PRD-217 Proxy cache support for sharer promotion	Sharer promotion
NCR34-PRD-218 AXI AIU with proxy cache non-coherent access support.	Proxy cache
NCR34-PRD-219 Multi-ported AXI AIU implementation support	Multi-ported NCAIU description
NCR34-PRD-220 Support for DMI WTT/RTT threshold	DMI Threshold support
NCR34-PRD-221 Support for performance monitors	Performance events
NCR34-PRD-222 No interleaving on AXI AIU data responses	Read data interleaving mode
NCR34-PRD-223 Support for security and coherency based on configuration register bit values	Definition of a Memory Address Map
NCR34-PRD-224 Connectivity interleaving	Removing connectivity
NCR34-PRD-226 Software credit management and skid buffer optimization	End to end credit

NCR34-PRD-227 Read/ Write dependency downstream the SMC to be removed to comply with standard	?
NCR34-PRD-228 Width and Rate adapter updates	CDTI Blocks
NCR34-PRD-229 Clock Domain Crossing on SMI and APB interfaces	Adapters automated insertion support

TABLE 1-2: LINKS TO SECTIONS THAT CAPTURE THE TECHNICAL ANSWERS TO THE TSRs

TSR ID	Description	Link to section
NCR34-TSR-174 NCR34-TSR-162 NCR34-TSR-175 NCR34-TSR-160 NCR34-TSR-176 NCR34-TSR-161	Memory, Transport and External interface ECC and parity logic shall be testable with a coverage >= 90%	
NCR34-TSR-170 NCR34-TSR-189 NCR34-TSR-171 NCR34-TSR-190 NCR34-TSR-172 NCR34-TSR-191	Snoop Filter, Proxy Cache and System Level Cache memories address and data shall be protected by Parity	SRAM protection
NCR34-TSR-150 NCR34-TSR-151 NCR34-TSR-153 NCR34-TSR-154 NCR34-TSR-157 NCR34-TSR-158	Snoop Filter, Proxy Cache and System Level Cache memories address and data shall be protected by ECC	SRAM protection
NCR34-TSR-184	Lock Steps Latent	Functional Safety
NCR34-TSR-186	Timeout Latent	
NCR34-TSR-148 NCR34-TSR-163 NCR34-TSR-149 NCR34-TSR-165 NCR34-TSR-152 NCR34-TSR-167 NCR34-TSR-155 NCR34-TSR-168 NCR34-TSR-179 NCR34-TSR-181	Functional unit shall be duplicated and output continuously compared	Functional SafetyFunctional Safety Controller
NCR34-TSR-169 NCR34-TSR-173	Transaction internally exchanged between agents or between agents and external associated units shall be protected by using parity	Transport protection
NCR34-TSR-156 NCR34-TSR-159	Transaction internally exchanged between agents or between agents and external associated units shall be protected by using ECC	Transport protection
NCR34-TSR-177	Configuration Registers Readback	Ncore CSTI Network

NCR34-TSR-178	Configuration Registers Protection	CSR register protection
NCR34-TSR-180	TP Timeout	?
NCR34-TSR-183	Fault Reporting Logic	Fault Checker
NCR34-TSR-185	Fault Reporting BIST	Fault Checker
NCR34-TSR-188 NCR34-TSR-182	Debug & Trace, BIST and APB port shall be isolated to be protected against unattended activation	Debug and BIST disable pin

2 Ncore System description

An Ncore 3 system is comprised of a collection of specialized, parameterizable, and configurable soft digital function blocks, connected via a fabric providing a light-weight transport protocol.

This chapter describes the structure of an Ncore 3 system.

2.1 Terminology

To provide coherency and other storage access services to the components of an SoC, an Ncore 3 system implements a proprietary protocol, called ***Concerto C Messaging Protocol (CCMP).*** This protocol is implemented via a set of messages exchanged between the various specialized function blocks that are the components of the Ncore 3 system. These blocks are called the Ncore 3 units.

Certain of Ncore 3 units, in addition to being connected to other Ncore 3 units also embody one or more additional interfaces to which components in the SoC, referred to as “external agents,” may connect to access the services of the Ncore 3 system. An external device that can originate a request, such as a storage access, into the Ncore 3 system is referred to as a “requester agent,” an “initiator agent,” or simply, an “agent.” An external device to which an Ncore 3 unit might transmit a request is referred to as a “target device,” or simply a “target.”

An agent or target device is expected to have one or more “native” interfaces by which it and the Ncore 3 system communicate with each other. A single native interface may be comprised of multiple transmit or receive sub-interfaces (from the point of view of the device). Associated with a native interface is the “native protocol,” which describes the rules of exchange of information over the interface. A unit of communication exchanged over the native interface is referred to as a “transaction.”

A single request transaction from an external agent translates into a sequence of CCMP messages internal to the Ncore 3 system plus additional transactions on external interfaces, as necessary.

CCMP messages are carried over an underlying light-weight, packetized transport interconnect, called the ***Concerto Transport Fabric (CTF)***. The interconnection fabric itself is oblivious of the message contents and their semantics. The transport fabric and all the Concerto C units connected via this fabric

are henceforth referred to in this document as the ***Concerto C Protocol Domain***. The protocol domain represents the totality of hardware elements that participate in the CCMP, both in terms of transporting its messages and of taking action based on them, thus being subject to CCMP semantics. When reference is specifically in relation to coherency, the protocol domain is also referred to as the ***coherency domain***.

Storage in an SoC is classified either as ***system memory*** or ***peripheral storage.*** System memory refers to the general purpose bulk memory, often implemented via DDR DRAM or SRAM in modern computer systems, used to store program code and data. Ncore 3 system enables access to system memory via specialized interfaces and ensures maintenance of coherence for those accesses when so demanded. Peripheral storage, on the other hand, is storage in the system that includes peripheral device based storage, particularly such as control and status registers. Because of its nature, coherency protocols cannot be applied to such type of storage.

Peripheral storage also is subject to different access ordering rules than coherent storage. For this reason, in Ncore 3, peripheral storage is accessed via a separate specialized interface.

Although this interface also supports connectivity to SRAMs and DDR DRAM, data residing in them cannot be accessed coherently.

2.2 Ncore units

The following is the list of types of Ncore 3 units and a short description of the function each type serves.

2.2.1 Agent Interface Unit (AIU)

AIUs offer means of connecting request originating agents into an Ncore 3 system. An AIU externally connects to an agent via an industry standard native interface. Internal to the Ncore 3 system, it connects to multiple other Ncore 3 units.

An AIU's primary purpose is for protocol translation from agent's native to Concerto C. The unit communicates with the external agent using its native protocol. The AIU embodies all the necessary functionality to support the native protocol. A request transaction causes the AIU to initiate a Concerto C message sequence to realize the semantic intent of the transaction.

An AIU can handle both coherent and non-coherent accesses to system memory and non-coherent accesses to peripheral storage.

Each AIU implements the System Address Map specified for the SoC. There can be more that one type of AIU in an Ncore 3 system.

2.2.2 Coherent-Agent Interface Units (C-AIU)

Ncore 3 offers C-AIUs to connect coherent agents into the system, typically individual processor or processor clusters, which include internal cache hierarchies. Each C-AIU offers one native interface to which an agent can be attached. A C-AIU supports one of the following three types of interfaces:

- ACE: ACE-AIU offers an ACE native interface to connect compatible processors and processor clusters which include coherency-capable cache hierarchies into the Ncore 3 coherency system.
- CHI-A: CHI-A-AIU offers a CHI-A native interface to connect compatible processors or processor clusters which include coherency-capable cache hierarchies into the Ncore 3 coherency system.
- CHI-B: CHI-B-AIU offers a CHI-B native interface to connect compatible processors or processor clusters which include coherency-capable cache hierarchies into the Ncore 3 coherency system.

For maintaining coherency over caches internal to the agent, a C-AIU is equipped with logic to snoop the agent via its native interface.

A C-AIU can handle both coherent and non-coherent accesses to system memory and to peripheral storage that might be issued by its agent.

2.2.3 Non-Coherent Agents'- Agent Interface Units (NC-AIU also called IO-AIU)

Ncore 3 offers AIUs to connect non-cache coherent agents, i.e., those that either don't have internal caches or have internal caches but whose contents are not required to be maintained coherent with respect to the backing store. This type of devices is also referred to herein as peripheral or IO devices. Thus the AIU that support such devices is referred to as an NC-AIU or NC-AIU. A single NC-AIU can provide connectivity into the system to one or more peripheral or IO agents.

Each NC-AIU offers one native interface to which an agent can be attached. An NC-AIU supports one of the following three types of interfaces:

- AXI-4: AXI-AIU offers an AXI-4 native interface to connect to one or more non-coherent agents.
- ACE-Lite: ACE-Lite-AIU offers an ACE-Lite native interface to connect to one or more non-coherent agents.
- ACE-Lite-E: ACE-Lite-E-AIU offers an ACE-Lite-E native interface to connect to one or more non-coherent agents.

While the peripherals or IO agents themselves are non-cache coherent, their accesses may pass through a System Memory Management Unit block (SMMU), which authenticates and translates requests initiated by agents. The SMMUs may cache copies of the address translation entries that are stored in

the main memory of the system. As such, these copies may be required to be maintained coherent using a special coherency protocol that is usually part of the main cache coherency protocol of the system. Maintaining copies of these translation entries involves snooping the SMMUs via the same interface by which the peripheral devices connect into the system. In particular, ACE-Lite and ACE-Lite-E interfaces and their associated protocols have this functionality, which the corresponding NC-AIUs also enable.

There can be zero or more NC-AIU units of each type in an Ncore 3 system.

2.2.4 Muti ported NCAIU

Multi ported NCAIU can have multiple identical native interfaces in power of 2s and a single SMI protocol transport connection (the number of SMI ports are determined by number of networks in the system). The Ncore 3 system sees the multi ported NCAIU as a single Ncore 3 unit. Following limitation applies to multi ported NCAIU:

- DVMs are not supported.
- Each port is address interleaved and must be identical.
- Individual powering down of ports is not supported.
- Proxy cache supports a max size of 1MB per port

It is architected as a wrapper around a single NCAIU core that is replicated.

2.2.5 Distributed Coherency Engine (DCE)

Cacheline copies are kept coherent by one or more ***Distributed Coherence Engine Units (DCEs)***.

Each cacheline address is associated with a single DCE, known as the ***Home DCE*** for that cacheline address. The Home DCE acts as the point of serialization for, and enforces coherence on, the given cacheline address. The ***serialization point*** refers to the point at which coherent accesses from all agents to a given address are serialized, establishing a ***coherence order*** for that address.

In a system configured with multiple DCEs, cacheline address are distributed across the DCEs using an *implementation-defined* mapping function, e.g. address bit interleaving or hashing, that routes coherent memory accesses to the appropriate Home DCE based on the cacheline address. Address interleaving can help reduce average utilization and thereby improve coherent access throughput in the system.

A DCE may implement a system directory partition for tracking and managing the contents of caching agent caches with respect to the addresses associated with the DCE. This allows the snoop messages to be sent to only the necessary AIUs. In theory, a directory may also be configured without a directory, effectively “broadcasting” snoop messages to all caching agents (Hybrid directory and broadcast configurations may alsobe possible). NCore 3 does not support broadcast configurations.

Unlike an AIU, a DCE does not connect to an external agent, and thus does not carry an external interface.

There can be one or more DCE units in an Ncore 3 system. If coherent accesses are being made in an Ncore 3 system, however, at least one DCE unit must be present in the system.

2.2.6 Distributed Memory Interface (DMI)

In a Concerto-based system, ***system memory*** is assumed to behave as "normal" storage in that a write to a given memory location updates the value stored at that location and a read from a memory location returns the last value written to that memory location. Furthermore, performing an access to system memory does not have side-effects.

Typically, system memory resides in off-chip DRAM storage, commonly in DDR memory devices, or in SRAMs that might be on- or off-chip.

In Ncore 3, system memory can be accessed ***coherently***. Coherency is maintained on the basis of a cacheline, which is a contiguous block of a power-of-two number of bytes that is uniquely identified by a ***cacheline address***, which is a system memory address aligned to the size, in bytes, of a cacheline.

Note that regions of system memory can also be accessed non-coherently.

System memory may be treated as being ***cacheable***, meaning an agent may buffer a copy of a cacheline, or a ***cacheline copy***, in its cache.

In Concerto C, system memory may only be accessed via the Ncore 3 unit called, ***Distributed Memory Interface Unit (DMIU)***.

Note that any system memory reached via a DMI is within the Coherency Domain of the CCMP.

There can be one or more DMI units per Ncore 3 system.

Multiple DMIs can be accessed in an address interleaved manner.

The DMI block offers an AXI interface to connect to a memory controller. The width of this AXI bus is parameterizable.

A DMI may optionally contain a cache, called ***System Memory Cache (SMC)***, that acts as the last-level cache in-line to memory. Being in-line to the memory controller, the SMC can help reduce Read latency and improve delivered bandwidth.

The *SMC* offers an option to include an ***Atomic Engine (AE).*** The *AE* supports the ***Far Atomic Operations*** (***FAO***s) defined in CHI-B and ACE-Lite-E interface architectures. Thus, in Ncore 3 *FAO*s are supported for all locations in *system memory* connected via the DMI.

The DMI also contains logic to remap addresses referenced in Concerto C messages to physical addresses that may allow ***address compaction*** on the AXI bus.

2.2.7 Distributed IO Interface (DII)

Besides system memory, a system can have additional storage that cannot be, or doesn't need to be, maintained coherently. Configuration and status registers in the system's peripheral devices are an example of this category. These registers can have side effects under Reads or Writes. Consequently, they cannot be prefetched as part of a cacheline. Such storage, thus, must be treated as being ***non-cacheable***, and therefore also ***non-coherent***. Peripheral device based buffers and peripheral SRAMs are other examples of this category. These may be well- behaved, allowing them to be accessed as ***cacheable*** but ***non-coherent*** storage.

In Ncore 3, this category of storage, labeled herein generally as ***peripheral device storage,*** is accessed via a unit called ***Distributed IO Interface (DIIU)***.

Coherency is not supported for any storage that is accessed via a DII. However, if it is well- behaved, such as SRAM, it can be accessed as being cacheable.

There can be multiple DIIs in an Ncore 3 system.

The DII block offers an AXI interface to connect to IO devices. The width of this AXI bus is parameterizable.

The DII also contains logic to remap addresses referenced in Concerto C messages to physical addresses that may allow ***address compaction*** on the AXI bus.

Ncore 3 should have at least one SysDII, which is special DII connecting with CSR network

2.2.8 Distributed Virtual Memory System Engine (DVE)

Just like application processes, peripheral devices cannot always be trusted as being free of malicious or erroneous behavior. And so, their accesses must be screened via MMUs akin to those that are used inside processors. These MMUs used to authenticate IO accesses into coherency domain are called IO MMUs (IOMMU) or System MMUs (SMMUs).

It is also advantageous, and common practice, for processor MMUs and SMMU to use the same page tables stored in system memory. When this is done, an SMMU's contents must be maintained consistent with those of a processor's MMU. This is achieved in Ncore 3 via a hardware function block called **Distributed Virtual Memory Management Engine (DVE)**.

There can be at most one DVE in the Ncore 3 system.

DVE include a module to capture traces from other units. Also, it would be connected to the user's device driver to deliver captured data through APB interface.

2.2.9 Global Register Block (GRB)

Global Register Block (GRB) connects only to the CSR network and hosts system identification registers.

2.3 Ncore System organization

The Architectural system Overview with different concerto units is shown in Figure 2-1. The figure depicts one or more instances of the various types of AIUs discussed above connected to the Concerto Transport Fabric.

The coherent processors or accelerators or clusters thereof are connected individually via a native interface offered by ACE, CHI-A, or CHI-B AIU. One or more non-coherent agents, typically peripheral devices that don't possess coherent cache hierarchies and carry interfaces such as AXI, may optionally be first connected via an external network to concentrate their traffic, and then be connected to an AXI-AIU. This traffic may optionally be intercepted by an SMMU before getting to the AIU. The SMMUs may optionally interface with an ACE-Lite or an ACE-Lite-E AIU, depending on the desired functionality. ACE-Lite-E interfaces enable peripherals to perform stashing and Far Atomic operations. The fabric is comprised of one or more independent transport networks that carry Concerto C message traffic that is generated in response to coherent and non-coherent accesses along with other types of transactions issued by agents to the AIUs. The connectivity capabilities, performance parameters, physical placement, and topology of these networks are customizable per the needs of the SoC.

The target of an agent's transaction could be system memory or peripheral storage. System memory is accessed via one or more DMI units. System memory address ranges can be interleaved across sets of DMIs.

One or more DII units are used to access all peripheral storage in the system.

The Ncore 3 system supports the ARM DVM architecture operations including required interactions with the SMMUs in the SoC. The DVE unit coordinates the execution of the DVM protocol.

A concerto unit can expose upto three different main interface types, they are classified as

- Native interface: this is for communication with other standard interface IPs
- SMI: this is for concerto protocol communication
- APB: this is for control and status register (CSR) communication

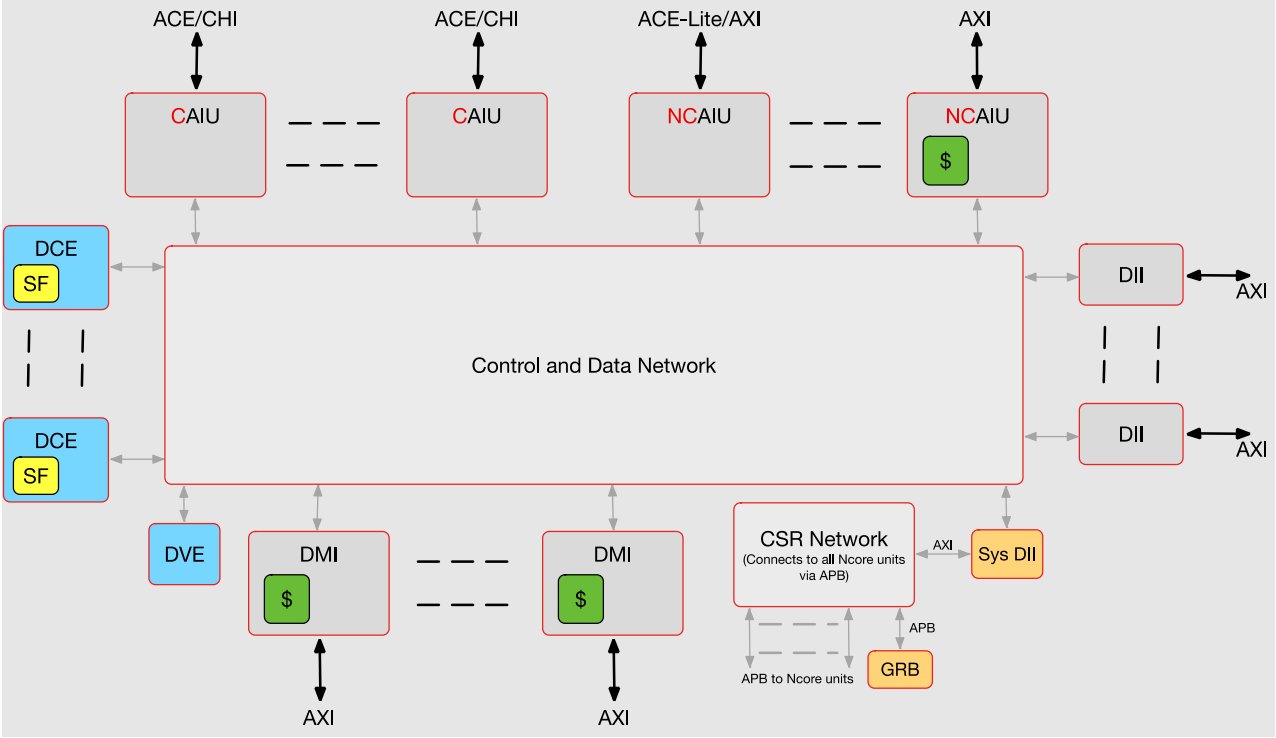


FIGURE 2-1 SYSTEM OVERVIEW

For the sake of brevity, the Figure 2-1 is simplified and does not show complete details of the CSR network and APB connections to all concerto units.

2.3.1 Multi-ported NCAIU description

The Top-level block diagram of a multi ported NCAIU is shown in Figure 2-2. As can be see, it is architected as a wrapper around a single NCAIU core that is replicated. The native interface (which can be AXI, ACE-Lite or ACE-Lite E) and NCAIU Core are replicated based on the parameter nNativeInterfacePorts. The multiport logic is responsible to make the NCAIU cores look like a single NCAIU to the CMUX (concerto mux), and in turn to the rest of the Ncore 3 system. The APB demux manages the APB connectivity to access each NCAIU core. See section 5.1 for more detail on Ncore 3 register region. The orange input ports are tie offs; note that there is:

- one identificator FUnit ID
- one Ncore register region identifier (NRRI)
- separate register page number (RPNs) depending on the number of ports.

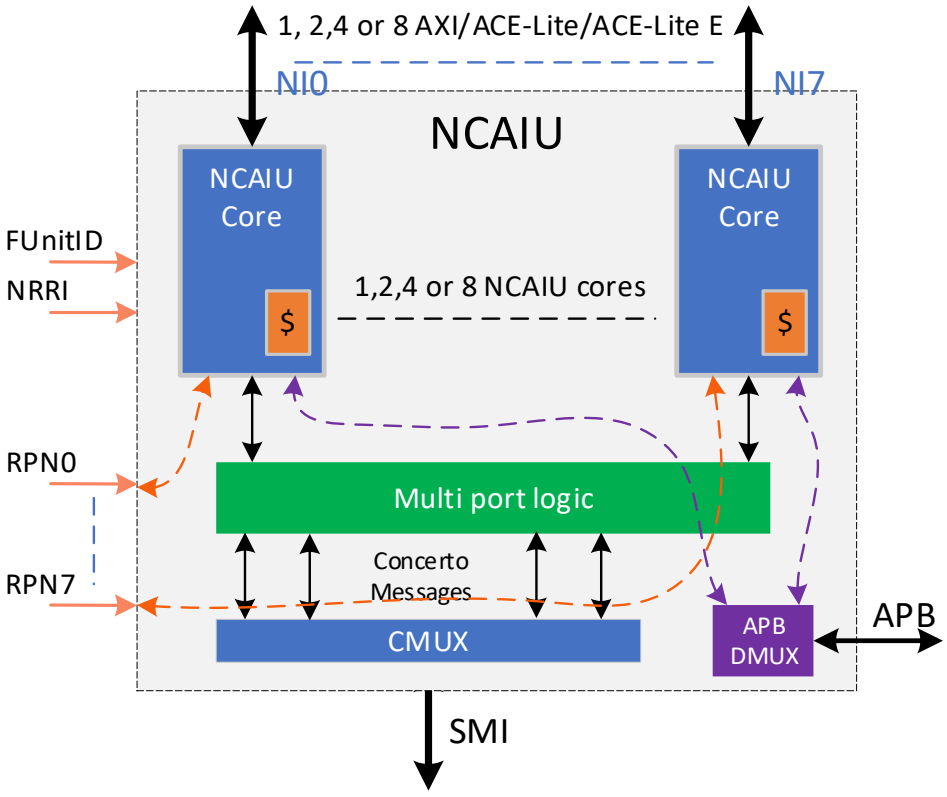


FIGURE 2-2 MULTI PORTED NCAIU

All NCAIU Cores must be identical. Parameters that are constrained based on the number of ports must be divided by the number of ports (nNativeInterfacePorts) before they are passed to the NCAIU core. The Native interface ports and the NCAIU cores are address interleaved based on the parameter aNcaiuIntvFunc. Depending on the implementation, address bits used for interleaving may or may not be present (stored) within NCAIU cores, in either case they are expected to be present on the native interface and the SMI interface.

Outgoing messages must have unique message ID. Ideally this message ID is nothing but the OTT index, as there are multiple NCAIU cores the message ID will need to be extended by appending NCAIU core identification bits at MSB. These bits are nothing but ordinal numbers identifying the NCAIU core, example for a 4-port NCAIU it will be 2 bits 00, 01, 10 and 11. Depending on the implementation either the NCAIU core can append the unique identifier bits or just append Zeros which may be replaced by the unique identifier bits within the Multiport logic.

The Multiport logic block is mainly responsible for muxing and demuxing concerto messages between the NCAIU cores and the CMUX. The outgoing messages are muxed out based on the type of Concerto

message. Incoming Concerto messages are demuxed based on the MSB bits of the message ID within the Concerto message, exception to this rule is the SnpReq message which is demuxed based on the interleaved address bits.

2.3.1.1 CSR address space and access support

Each NCAIU core supports one register page number RPN, this translates to upto 8 RPNs per multi-ported NCAIU depending on number of ports configured. All the RPNs of a single multiport NCAIU must be contiguous and within a single NRRI. All configured RPNs (NCAIU cores) must be accessed via a single APB port, a APB demux is used to support only one APB port on the multi-ported NCAIU.

Note: SW changes may be needed to support more than 4K address space per APB on the CSR network.

2.3.1.2 Trace and Debug Support

Each NCAIU core will implement its own trace trigger block and CSRs. The trace capture block will have to be implemented within the Multiport logic and must be controlled via CSRs in the first NCAIU core. To keep the NCAIU cores identical all of them will implement the trace capture registers.

2.3.1.3 High transport BW support

To support higher BW on the SMI ports software will need to implement a special wrapper. This wrapper will instantiate 2 multi-ported NACIUs as shown in Figure 2-3.

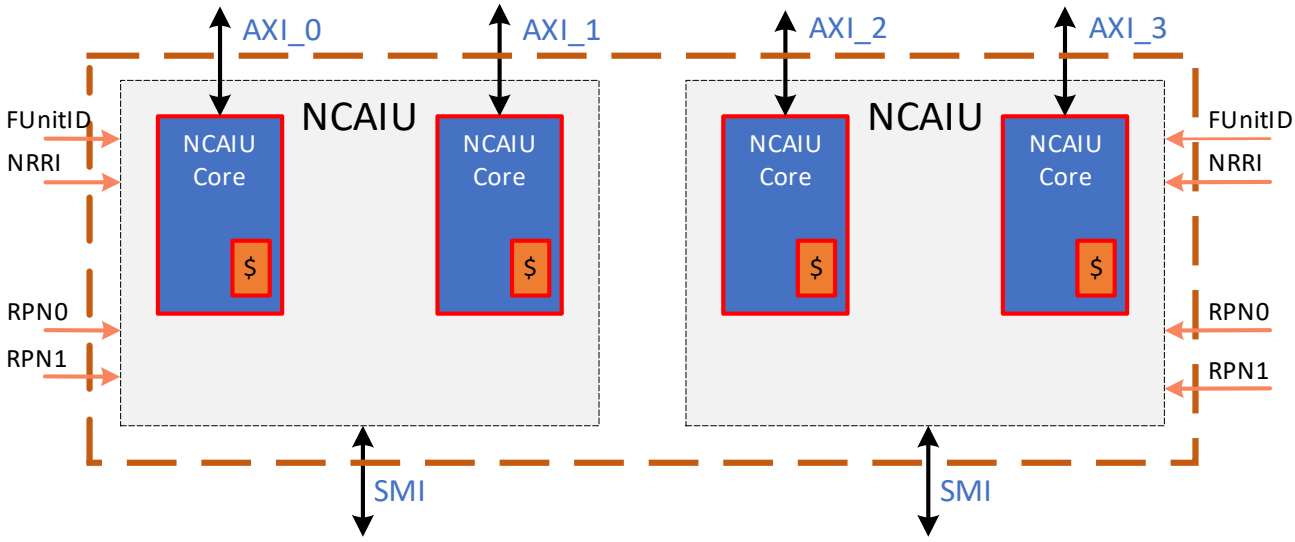


FIGURE 2-3 MULTI-PORTED NCAIU WRAPPER

2.4 Topology options

The transport layer offers the flexibility to create optimized topologies to find the best tradeoffs for each SoC configuration.

The list of datawidth options for each unit is the following:

- CHI AIU interface data width options: 128/256 bits
- ACE/AXI/ACE-Lite AIU interface data width options - 64/128/256
- DII interface data width selection - 64/128/256
- DMI interface data width selection - 128/256

As a reminder, the SMI interface width matches the corresponding ATP interface width because Ncore 3.4 supports header parallel style. All the data widths combinations shall be verified.

Ncore 3 does not impose restrictions on the choice of topologies but helps to ensure that the outcome is correct and efficient. It shall detect protocol and routing deadlocks.

2.5 Engineering version ID

The Engineering Version Identifier is a concatenation of two fields: {MPFHash[18:0], CHIP_ID[12:0]}. The CHIP_ID is obtained from a license file and the MPFHash is the last 19 bits obtained from the 128bit MD5 Hash."

3 Transport layer

Ncore 3 uses two types of transport interconnect: the Control and Status Transport Interconnect (CSTI) and the Control and Data Transport Interconnect (CDTI).

CSTI is the service interconnect for configuration and status register access to all Ncore 3 components. CDTI is the main protocol interconnect.

Ncore 3 components generate different set of control and data messages as described in Concerto protocol specification. These messages are mapped to one data network and at least two control network, these networks are referred together as CDTI. Separation of different messages onto independent transport resources avoids dependency loops inside the transport that would otherwise have deadlock issues and may also improve performance.

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. Data network supports three width options, 64, 128, and 256 bit. These options can be mixed and matched within a single data network.

For a single native request, Ncore 3 protocol issues multiple control messages and most likely a single data message. 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, a cache line read will consume 2 data network cycles. In this case to achieve maximum throughput three control networks must be used. This ensures the number of control network cycles required will be within the 2 data network cycles. Currently, Ncore 3 supports the following two 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.

3.1 Legato transport description

Ncore 3 units are interconnected via an underlying interconnection fabric, called Legato. It relies on the ***Concerto Transport Fabric (CTF)***. The CTF provides the transportation service for the CCMP messages. Each individual network is built out of communication components: switches, adapters, pipeline registers, and terminal blocks.

The components are connected amongst themselves via point-to-point, unidirectional links.

A network component may have multiple ports. Each port connects to exactly one link. A port is called an ***ingress port*** if the link connected to it points toward the component; it is called an ***egress port*** if the link connected to it points away from the component.

A component may have “primary” ports and “secondary” ports. Primary ports, ingress and egress, are defined to be the ones that are used for communicating information across the component during the normal operation of the network. It is assumed that secondary ports, ingress and egress, are strictly used for orthogonal and supplementary functions such as error reporting, component status inquiry, etc. A primary ingress port is considered to be connected to a primary egress port, if the component allows a message to travel from the said ingress port to the said egress port.

A link connects two network components. It connects an egress port on one component to an ingress port of another component. Information on the link travels over the link from the egress port to the ingress port, which is also the direction a message flows over the link. The direction of information flow over a link is the direction indicated for that link. Thus a link can only be “traversed” in the direction the information can flow over it. The two said network components are considered to be “connected” via the said directed link between them.

A ***link*** is a set of wires, perhaps punctuated by electrical buffers to help power the transmission of signals. It is used to transmit information from one of the pair of components it connects to the other. The link also includes signals that travel in the opposite direction that are used to control the flow of said information. In a graphical representation, the direction of the link indicates the direction of the flow of information over the link.

A path from network component A to network component B exists if there exists an alternating sequence of components and links, such that a message can travel from A to B. If this is true, component A is considered to be “connected” to component B via the path.

A single path defines a rudimentary CTF network. If two paths share a network component, they both belong to the CTF network that contains all the components and links used by those two paths. If a path shares a component of a CTF network, the path and the components used by it also belong to the said CTF network.

If a path does not share a component with a network, it does not belong to the said network.

Note that by definition, paths are unidirectional. A path from A to B does not necessarily guarantee a path from B to A.

A message enters a network via one ***terminal block*** and exits it via another *terminal block*. The terminal blocks are *directional*. A unit connects to a ***Transmit*** (***TX***) terminal block to send a message into the network, and it connects to a ***Receive*** (***RX***) terminal block to receive a message from the network. Figure 3-1 shows a path from a TX terminal block to an RX terminal block via a CTF network.

A CTF network, thus, is a collection of network components, and links connecting them, such that it provides paths from one defined set of terminal blocks to another defined set of terminal blocks. Figure 3-2 shows a CTF network offering multiple TX and RX terminal blocks to which an end-point units can connect to transmit and re ceive messages.

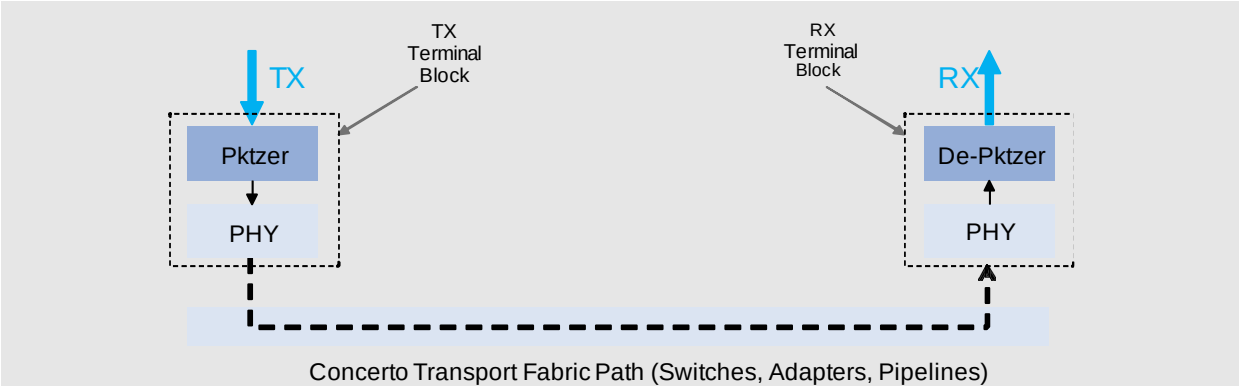


FIGURE 3-1 TERMINAL BLOCK TO TERMINAL BLOCK PATH THROUGH THE CTF

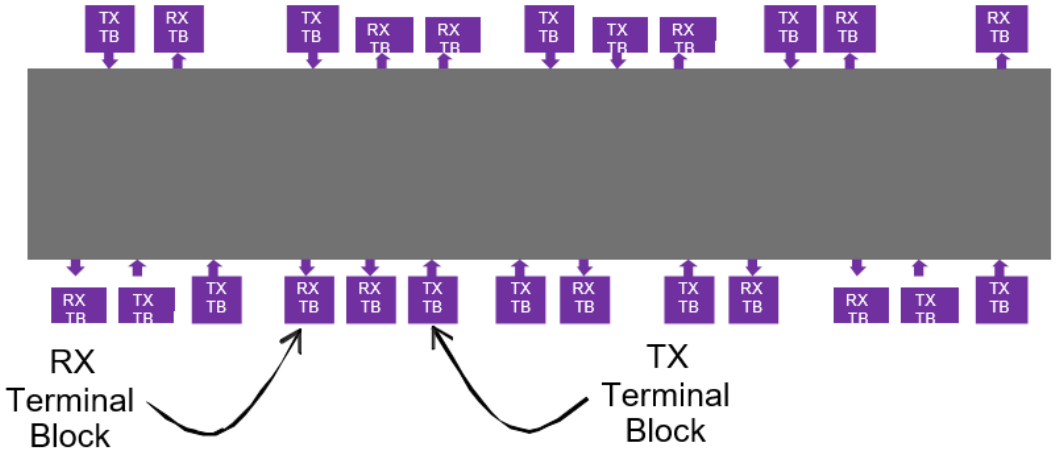


FIGURE 3-2 A CTF NETWORK OFFERING MULTIPLE TERMINAL BLOCKS FOR CONNECTING UNITS

A CTF network, in general, may offer more than one path between two terminal blocks.

A given CTF network is characterized by the subset of types of messages it carries. In general, a given physical path in a network may be used to carry multiple types of Concerto C Protocol messages. Distinct CTF networks carry distinct subsets of CCMP message types.

3.1.1 Attachment of Ncore units to CTF

Ncore 3 units use a CTF network to send or receive messages from other Ncore 3 units.

Figure 3-3 shows the connectivity between the Ncore 3 units and a CTF network. As can be seen, the sending Ncore 3 unit attaches to the TX terminal block of the CTF and the receiving Ncore 3 unit attaches to the RX terminal block.

An Ncore 3 unit A can send a message to another Ncore 3 unit B if a path exists from the *TX terminal block* to which A is connected to the *RX terminal block* to which B is connected.

A single terminal block, TX or RX, is used to connect to any single CTF network.

In general, an Ncore 3 unit may connect into a given CTF network via one or more TX and/or RX terminal blocks.

An Ncore 3 unit may also connect into multiple CTF networks. However, sets of terminal blocks used by an Ncore 3 unit to connect into distinct networks are distinct.

The interfaces between an Ncore 3 units and CTF terminal blocks are defined in detail in section 3.3.

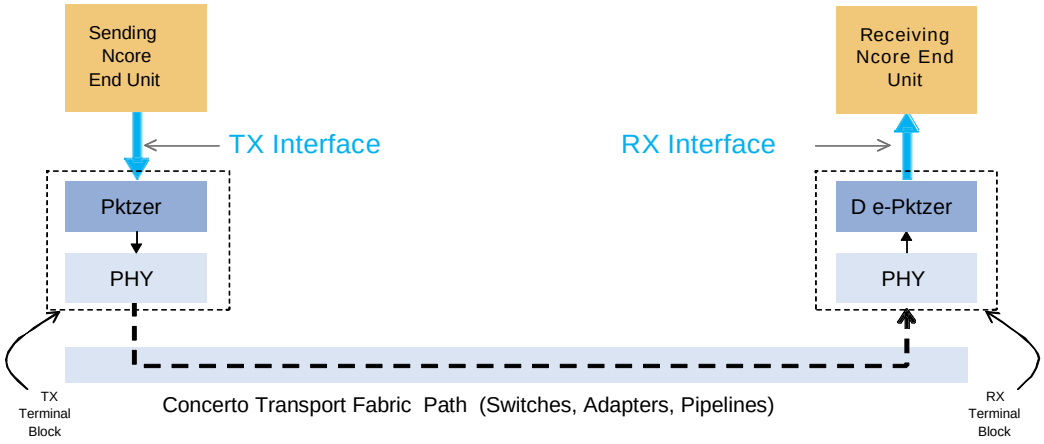


FIGURE 3-3 NCORE UNIT-TO-UNIT PATH VIA THE CTF

3.1.2 Structure of the Terminal Block

A ***terminal block*** connects on one side to another CTF network component via a link. On the other side, it offers a standardized interface to which an Ncore 3 unit is connected. The terminal block offers a ***Transmit interface (TX)*** to the Ncore 3 unit if the network link attached to the block points away from it. The terminal block offers a ***Receive interface (RX)*** to the Ncore 3 unit if the network link attached to the block points towards it. A terminal blocks thus represents an extremity, an end, or a "terminal" for the transport interconnects and the ***ATP protocol domain***. The domain is entered at a TX terminal block and exited at an RX terminal block. The Ncore 3 units that attach to the terminal blocks are thus referred to as ***end-units*** or ***terminal-units*** of the CTF networks.

Figure 3-4 (a) shows a diagram of the functionality of a Transmit Terminal block. It performs the following functions:

- **Interface Logic:** Accepts a message from the Ncore 3 unit with which it interfaces.
- **Route Calculation:** Calculates the Data-Link layer route to the target unit indicated in the message.
- This blocks may take into account the "Steering" field in the Concerto C Message Header in calculating the Route. It is not used in the current version of Ncore.
- **ATP Data-Link Packetization:** Creates a Data-Link layer packet from the message. Note that each Concerto C Message results in a single packet at the ATP DL layer.
- This block may take into account the TTier field in the Concerto C Message Header to pick the virtual channel to use for transmitting the packet along its link. Virtual channels are not used in the current version of Ncore 3.
- **ATP PHY TX Interface:** Transmits the packet into the CTF network via its link.

Figure 3-4 (b) shows the internal functionality of a receive terminal block. Performs the following functions:

- **ATP PHY RX Interface:** Accepts a Data-Link layer packet from its CTF link.
- **ATP Data-Link De-Packetization:** Extracts the message from the ATP packet format to forward it to the end-unit attached at this TerminalBlock.
- **Interface Logic:** Transmits the message to the Ncore 3 unit with which itinterfaces.

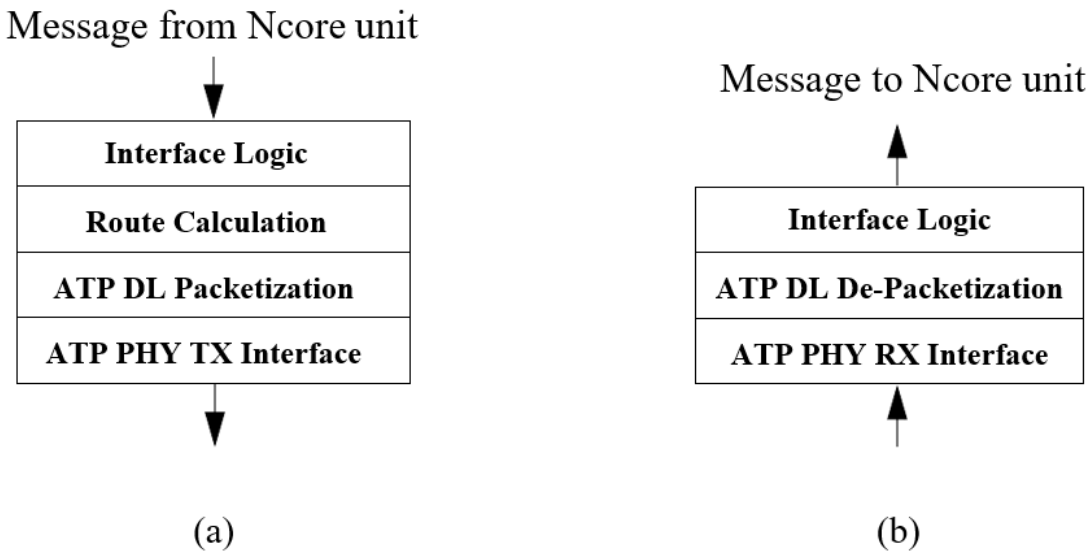


FIGURE 3-4 STRUCTURES OF TX AND RX TERMINAL BLOCKS.

3.2 Ncore Messaging over CTF

3.2.1 CCMP Message Formats

Two basic formats are used for CCMP messages:

- **A Data Message (DM)**
  - In addition to command and/or control information, a data message carries data that may be read from or written into storage locations.
- **A Non-Data Message (NDM)**
  - A non-data message carries only command and/or control information

Below are shown formats of NDM and DM:

NDM = <Concerto C Message Header> + <Non-Data Payload>

DM = <Concerto C Message Header> + <Non-Data Payload> + <Data Payload> + <Auxiliary Payload>

Exchange of each message over an SMI is performed “atomically,” meaning that one message is delivered completely before the next is transmitted, for both NDM and DM.

An NDM is delivered wholly in a single clock cycle.

For a DM, the Data Payload is allowed to be transferred in multiple segments or beats, each of which must be a power of 2 number of double-words (64 bits). Thus, if the size of the beat is 2 double words (8 bytes), or 16 bytes, 64 bytes of data will be transferred in 4 beats.

In a DM, the <Concerto C Message Header>, <Non-Data Payload>, and the first beat of <Data Payload> are all transmitted during the first cycle of DM transfer, and the remaining beats of data are transmitted during subsequent cycles.

In general, a data payload beat may also include auxiliary information payload, as shown above. This auxiliary information is on a per double word basis and is present for each double word in the beat. Typically, per double word, the following information is included:

- DW Byte Enables
- DW Protection bits
- DWId
- Poison
- User bits

The auxiliary bits for the data beat are right-appended to data beat.

3.2.2 Concerto C Message Header

All Concerto C messages have a common set of fields at their start. This set forms the common header for all Ncore C messages. Although the width of the fields can change from one Ncore 3 instance to the next, the order of fields is fixed as shown in Figure 3-5, giving all message a common header format.

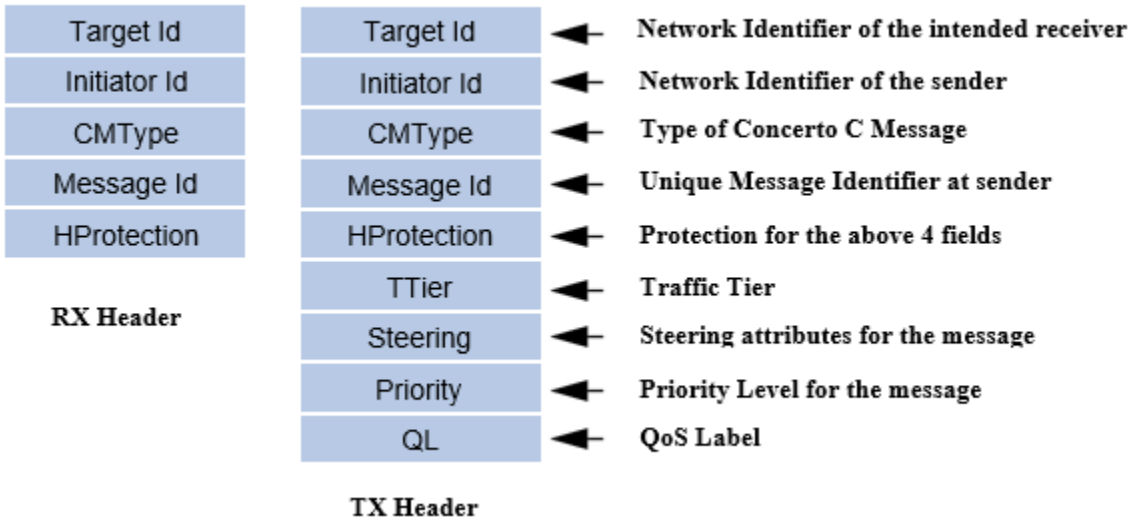


FIGURE 3-5 CONCERTO C MESSAGE HEADERS

Target Id, Initiator Id, CMType, Message Id, and HProtection are fields that are carried end-to-end by the message, and hence appear in the TX and RX headers in Figure 3-5.

There are other fields in the TX header that are not intended to be carried end-to-end, but are included in the header. It is via these fields that the Ncore 3 unit at the interface makes specific service requests of the transport service. The purpose of these fields is specified below:

- **TTier**: Traffic Tier. This field is used for deadlock avoidance purpose. Each type of Concerto C message is associated with an architected TTier value. Multiple types of messages may have the same TTier value.
- The transport network maps a TTier value to a virtual channel for each of the links over which the message is carried through the network. For a given message type, the virtual channel chosen can be different at different links. However, the following rule must be satisfied to ensure freedom for deadlocks.

Deadlock Prevention Rule: A message carrying a value V for TTier must not be permanently blocked by a message carrying a value greater than V. Satisfaction of this rule in the interconnect is mandatory to prevent deadlocks. Minimum value for this field is 0. Thus messages with TTier = 0 must not be blocked by any message with TTier > 0. This field is not intended to be treated as a Quality of Service attribute.

- **Steering**: This field influences the determination of the route taken by the message to reach its target. It is not used in the current Ncore 3 implementation.
- **Priority**: A higher priority message is given preference over a lower priority message in determining passage across network components.

- **QL**: QoS Label. This field is used to enable bandwidth provisioning and requirements. It is not used in the current Ncore 3 implementation.

The widths of the fields in the header and individual message types are build-time constants that are known to all Ncore 3 units. The widths of the fields in the message header are also passed into the terminal block. This enables the terminal block to correctly access these fields when it receives the message over the SMI.

Other fields in the message, which constitute the "body" of the message are not visible to the transport facility.

Similarly, the Ncore 3 unit receiving a message is able to identify and access all the fields in the header. Once it accesses the CMType field, it knows what type of message it has received.

That in turn allows the receiving unit to interpret all the fields of the message.

3.2.3 Packetized delivery of Concerto C messages

The exact delivery format of Concerto C message is outside the scope of this architecture. The only requirement placed by this architecture is that for multi-PHIT delivery of messages, the Concerto C Message header is replicated for each PHIT of transmission.

3.2.4 Formats of Concerto C Messages

Majority of Concerto C Message use the NDM format.

3.2.4.1 DMs

The following CCMP messages use DM format: DTR, DTW, DTWMrgMRD

3.2.4.2 NDMs

The following CCMP messages use NDM format:

CMDreq, SNPreq, MRDreq, HNTreq, STRreq, RBRreq, RBUreq, UPDreq, CCMDrsp, NCMDrsp, SNPrsp, DTWrsp, DTRrsp, HNTrsp, MRDrsp, STRrsp, UPDrsp, RBRrsp, RBUrsp, CMPrsp, CMErsp, and TRErsp.

3.3 CTF Messaging Interface

An Ncore 3 unit and a CTF terminal block communicate via standardized interfaces, called the ***Symphony Message Interface (SMI)***, across which the two blocks exchange messages.

An *SMI* can be of ***TX*** or ***RX*** kind, named with reference to the Ncore 3 unit which shares it with a *Terminal Blocks* of the CTX. These two interfaces are shown in Figure 3-6.

The *SMI* is defined to be a synchronous interface, and any resynchronization to another clock is done outside the SMI interface.

The messages exchanged across the SMIs are Concerto C Messaging Protocol messages.

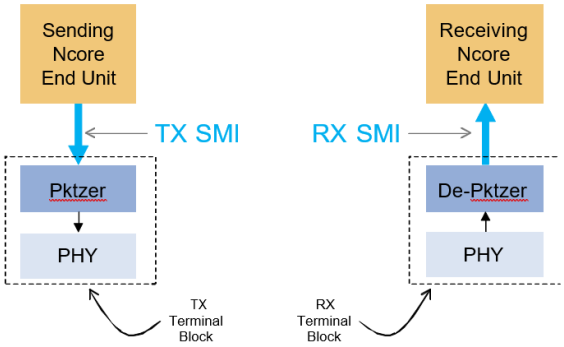


FIGURE 3-6 TX AND RX SMI INTERFACES

3.3.1 Transmission of Concerto C messages overCTF

As shown in the Figure 3-7, after receiving a message from the Ncore 3 unit over the TX SMI, the transmit terminal block packetizes the message and transmits it via its link into the CTF network. On the other side, a receive terminal block receives a packet from its link, extracts the message from it, and transmits it to the Ncore 3 unit connected via the RX SMI.

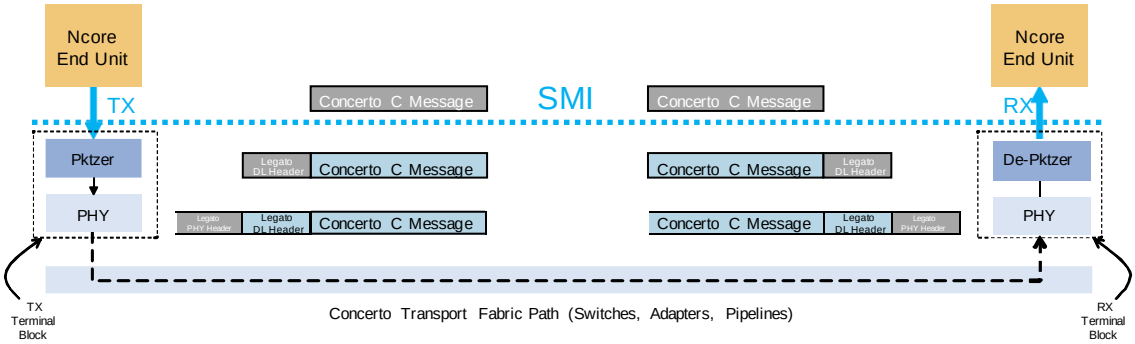


FIGURE 3-7 TX AND RX SMI INTERFACES

3.3.2 SMI Interface signals

Table 3-1 shows the signals for a transmit side SMI. The interface shown is a combined interface that allows sending DMs and NDMs.

An NDM-only TX SMI has a reduced signal list: DPPresent, DPValid, DPLast, DPayload, and DPAux signals are not included. For a DM-only TX SMI, DPPresent bit can be asserted permanently (tied to 1).

While Legato offers VC support, virtual channels are not used in NCore 3.

TABLE 3-1 SMI COMBINED TX INTERFACE

Signal Name	Direction (Ncore unit- centric)	Description
Ready	Input	Indicates willingness to accept transfer
Valid	Output	Indicates a Valid transfer
NumNDPBits	Output	Number of ND-payload bit. This equals length of NDM in bits. This helps define the minimum sized packet for the message.
NDPayload	Output	The Non-data payload of the Concerto C Message
DPPresent	Output	Data payload present in this message
DPValid	Output	Data payload valid in this cycle
DPLast	Output	Asserted for the last beat of data payload.
DPayload	Output	Data payload beat. 1, 2, 4, or 8 Double words.
DPAux	Output	Auxiliary bits. <Num Aux bits per DW> *<Num DWs Per Beat>

Table 3-2 shows the signals for a receive side SMI. The interface shown is a combined interface that allows receiving DMs and NDMs.

TABLE 3-2 SMI COMBINED RX INTERFACE

Signal Name	Direction (Ncore unit-centric)	Description
Ready	Output	Indicates willingness to accept transfert
Valid	Input	Indicates a Valid transfer at VC
NDPayload	Input	The Non-data payload of the Concerto C Message
DPPresent	Input	Data payload present in this message
DPValid	Input	Data payload valid in this cycle
DPLast	Input	Asserted for the last beat of data payload.
DPayload	Input	Data payload beat. 1, 2, 4, or 8 Double words.
DPAux	Input	Auxiliary bits. <Num Aux bits per DW> *<Num DWs Per Beat>

An NDM-only RX SMI has a reduced signal list: DPPresent, DPValid, DPLast, DPayload, and DPAux signals are not included.

For a DM-only RX SMI, DPPresent bit can be asserted permanently (tied to 1).

3.3.3 Mapping Messages to SMI fields

This section specifies how CCMP message contents are mapped onto SMI fields at the TX SMI and appear on the SMI fields at the RX SMI.

3.3.3.1 NDM TX

For an NDM, the entire NDM is mapped bit 0-aligned into the NDPayload field of the TX SMI interface. That is,

TX_SMI.NDPayload ⇐ <Concerto C Message Header> + <Non-Data Payload> (left aligned)

The message header, being at the beginning of Concerto C message is always located at the start of the NDPayload field of the SMI, TX.

3.3.3.2 NDM RX

At the RX SMI interface, the whole NDM is wholly extracted left-aligned from the NDPayload field of the SMI interface.

<Concerto C Message Header> + <Non-Data Payload> ⇐ RX_SMI.NDPayload (left aligned extraction)

The message header, being at the beginning of Concerto C message is always located at the start of the NDPayload field of the SMI, RX.

3.3.3.3 DM TX

For a DM, the Message Header and the Non-Data Payload portions of the DM are mapped left-aligned into the NDPayload field of the TX SMI interface. That is,

TX_SMI.NDPayload ⇐ <Concerto C Message Header> + <Non-Data Payload> (left aligned)

The message header, being at the beginning of Concerto C message is always located at the start of the NDPayload field of the SMI, TX.

The Data Payload portion of DM is mapped into the DPayload field of the TX SMI, on a per beat basis. That is,

TX_SMI.DPayload ⇐ <DM.Data Payload>

Note: It is required that the number of DWs supported in the TX_SMI match that for the DM being transmitted over it.

The Auxiliary Payload portion of DM is mapped into the DPAux field of the TX SMI, on a per beat basis. That is,

TX.SMI.DPAux ⇐ <DM.Auxiliary Payload>

3.3.3.4 DM RX

At the RX SMI interface, the Message Header and the Non-Data Payload portions of the DM is extracted left-aligned from the NDPayload field of the SMI interface.

<Concerto C Message Header> + <Non-Data Payload> ⇐ RX_SMI.NDPayload (left aligned extraction)

The message header, being at the beginning of Concerto C message is always located at the start of the NDPayload field of the SMI, RX.

The Data Payload portion of DM is extracted from the DPayload field of the RX SMI, on a per beat basis. That is,

<DM.Data Payload> ⇐ RX_SMI.DPayload

Note: It is required that the number of DWs supported in the TX_SMI match that for the DM being received over it.

The Auxiliary Payload portion of DM is extracted from the DPAux field of the RX SMI, on a per beat basis. That is,

<DM.Auxiliary Payload> ⇐ RX.SMI.DPAux

3.4 Ncore CDTI Network

The CDTI provides the transportation service for the CCMP messages. The CDTI may be comprised of one or more independent networks or interconnects, and individual network is built out of communication components: switches, adapters, and terminal blocks.

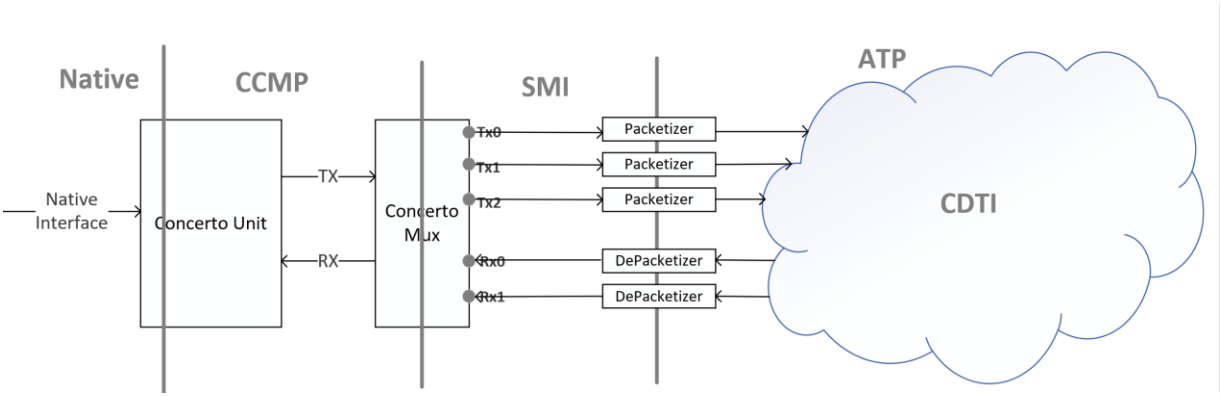


FIGURE 3-8: CCMP COMMUNICATION THROUGH THE CDTI

Concerto_Mux is playing an important role as a terminal between CCMP and ATP:

- Protocol conversion from CCMP to SMI
- Network mapping: CCMP includes large number of message types, and these will be delivered into ATP using different network considering performance and deadlock avoidance. The example in Figure 3-8 shows message mapping into three Tx networks and two Rx networks.
- ECC/Party check is implemented
- Transport error is detected, and the results will be added at CMStatus.

Transport error is detected only when Resiliency is enabled.SMI (Symphony Message Interface) is messaging interface that transfers messages with or without data (Ndp_, Dp_). Information that is needed to determine the route of the packets from Source to the Destination in the CDTI is added at this layer. This layer is also responsible for traffic management across the fabric.

NOTE: Flow control of SMI message is based on valid-ready handshakes.

NOTE: The SMI is defined to be synchronous interface, and any resynchronization to another clock is done outside the SMI interface (In CDTI).

Packetizer and Depacketizer are format/framing of packets/flits for actual transmission of control and data bits over the CDTI networks. (ATP) Legato components are based on this ATP protocol.

3.4.1 CDTI Blocks

There are four types of blocks used in the networks of the CDTI: switches, pipe, clock and width adapters. For the detailed configuration capability of the Adapters, refer NCore 3 parameter specification.

3.4.1.1 Adapters automated insertion support

Maestro shall support automated insertion of width adapters. When source and destination of a network segment have different width, the decision shall be based on:

- nInputWidth = {switch, FUnit} transmitter.width
- nOutputWidth = {switch, FUnit} receiver.width

The automatically generated WidthAdapter shall be customer configurable by changing the default settings of the following parameters:

- nDepth (default = 0) to configure additional buffer stages
- boolPipelined (default = false)

Maestro shall support user configurable insertion and removal of RateAdapters

- UI shall provide a means to select a network connection between two FUnits or an FUnit and a switch

The manually inserted RateAdapter shall be configurable by UI

- nDepth (default = TxnSize) to configure buffer stages
- nDepth shall be derived from the network segment where the user chose to insert the adapter
- When a user attempts to insert a rate adapter on a segment connecting a WidthAdapter output to a receiver, Maestro shall offer to parametrize the widthAdapter to increase depth instead (do we need a forced override to insert a RateAdapter?)
- When a user attempts to insert a rate adapter in front of a WidthAdapter, Maestro shall issue a warning, this is useless and only adds latency, recommend to parametrize the width adapter instead
- Future versions of the RateAdapter may support different different clock domains for input and output ports

TABLE 3-3 INSERTION RULES FOR ADAPTERS

	Input < Output	**Input = Output**	**Input > Output**	**Description**
Type	Width Adapter	Rate Adapter	Width/Rate Adapter	Adapter type depends on the interface configuration
Rule	Automatic Insertion	Insertion by User Input	Automatic Insertion	When input and output do not have the same width, a Width Adapter will be required
Configurability	**Automatic**	**Automatic**	**Automatic**	

	• Insertion = Yes • nInputWidth • nOutputWidth	• Insertion = No	• Insertion = Yes • nInputWidth • nOutputWidth	
	User • boolPipelined • nDepth[1]	**User** • Insertion • nWidth • nDepth • boolPipelined	**User** • boolPipelined • nDepth[1]	
nDepth	**Automatic** 1 x nInputWidth + 1 x nOutputWidth		**Automatic** ≥ 1 x nInputWidth	Automatic insertion will alway use the minimum size required for the functionality
	User + n x nOutputWidth	**User** parameter based on rate difference ≥ n x nWidth	**User** + n x nOutputWidth	User may configure additional storage in Maestro's UI
Notes:	1. Optional, additional buffer stages for rate adaptation			

3.4.1.2 Switches

Ncore 3 supports sym_switch and sym_buf_switch.

- Sym_switch: This is a worm hole switch that provides no buffering or pipelining at all in the switch. There is a check put in that looks for logic loops between sym_switches because of cycles in the topology. These cycles can be broken with a sym_pipe_adapter.
- Sym_buf_switch: Buffered switches are switches that at a minimum buffer at the input of the switch but can also buffer in various places along the data path.
- All switches support path pruning.
- Ncore 3 is only using round-robin arbitration policy.

3.4.1.3 Clock Adapters

- sym_async_adapter: This is a unidirectional adapter that sits between two ATP links that run off different clocks. The two clocks can be edge aligned or not. It is also being used for power domain control. Users can change the buffer depth of this async adapter. Also, user could configure the depth of synchronizer using system parameter.
- chi_async_adapter: This is a unidirectional adapter that sits between CHI interfaces that run off different clocks. The buffer depth is calculated referring the credit of chi-interface. User could configure the depth of synchronizer using system parameter.
- sym_rate_adapter: A rate adapter is a unidirectional adapter that sits between two ATP links that run off different clocks and the ATP link's clock entering the block is slower than the ATP link's clock exiting the block. The purpose of the rate adapter is to eliminate bubbles in a packet on the fast side of the adapter.

3.4.1.4 Pipe adapters

- sym_pipe_adapter: The pipe adapter is a unidirectional adapter that places storage along an ATP link and can be used as a pipe stage or FIFO. The ATP links going in and going out have identical definitions. User could configure the size of this pipe adapter. Also, sym_pipe_adapter has two modes: circular and FIFO. They're functionally equivalent but affect timing and power. Circular is lower power but has worse timing on the output. Default should be circular.

3.4.1.5 Width Adapters

Ncore 3 architecture supports different widths of networks between agents (64, 128, 256 bits).

Connections between receivers and transmitters with different widths require a WidthAdapter.

A WidthAdapter converts a sequence of phits belonging to a packet arriving from a narrow interface to the wide interface.

This avoids using only part of the wide output interface's bandwidth, which would propagate downstream. A WidthAdapter will assemble a wider phit by storing:

- at least one phit entry of the width of the outgoing port
- one entry with the difference in width between the input and the output port

A WidthAdapter will introduce additional latency m:

- m = (nOutputWidth/nInputWidth) + ord(boolPipeline) If pipelining enabled

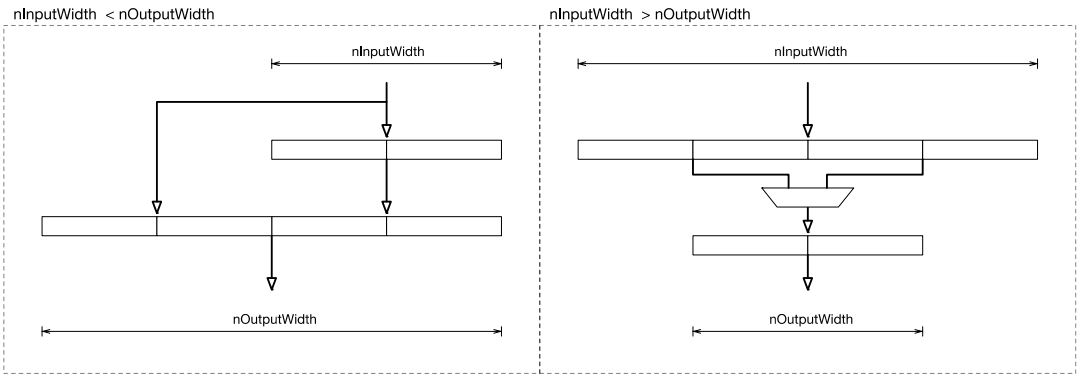


FIGURE 3-9: INTERNAL STRUCTURE OF WIDTHADAPTER

A WidthAdapter will introduce additional bubbles into the down stream traffic.

A WidthAdapter converting from wide interface to narrow interface may use a single wide entry to hold a phit while breaking it down into a stream of consecutive, narrow phits.

A WidthAdapter shall track up to 4 transactions and detect the boundary between packets having a different TxnID

3.4.1.6 Rate adapter

A RateAdapter will be used when a packet, consiting of multiple phits, may contain bubbles. The rate adapter's function is, to aggregate temporally separated pieces/phits of a transaction, and retransmit them as consecutive sequence to a downstream receiver. The Legato interconnect does not support transmission of flits belonging to different transactions. Rate adapters may be used to level out fluctuations in input rate, even when the arrival rate ≥ departure rate for a short time, at the cost of increased buffering.

- Rate adapters always have the same width on the input and the output port
- A rate adapter implements a FiFo-Queue where the first phit of a packet (flit) will not signal valid to the downstream receiver until the entire packet has been assembled in the queue.
- A rate adapter has to implement sufficient storage to hold at least one full packet - n buffer entries organized as width bits
- number of entries n = txn_size/port_width

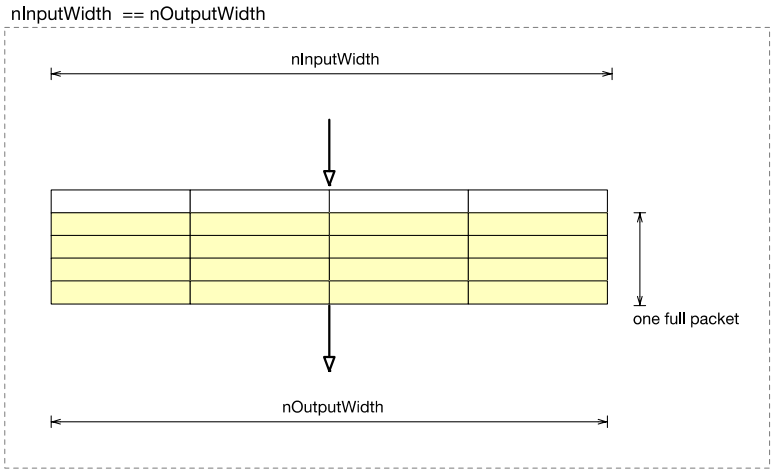


FIGURE 3-10: INTERNAL STRUCTURE OF RATE ADAPTER

Pipeline support shall be supported (improved timing), adding one additional storage entry of width bits to receive the first phit for the next transaction.

Additional entries may be specified if the designer desires to optimize bursty traffic in front of a congested switch. This will support more than a single transaction to be forwarded in an uninterrupted burst.

A rate adapter will introduce additional latency of m cycles:

- m ≥ number of phits per transaction + 1

A width adapter shall track up to 4 transactions and detect the boundary between packets having a different TxnID.

3.4.1.7 Combined Width/Rate adapter

Figure 3-11 shows an example of a WidthAdapter that has been parametrized to add buffer stages.

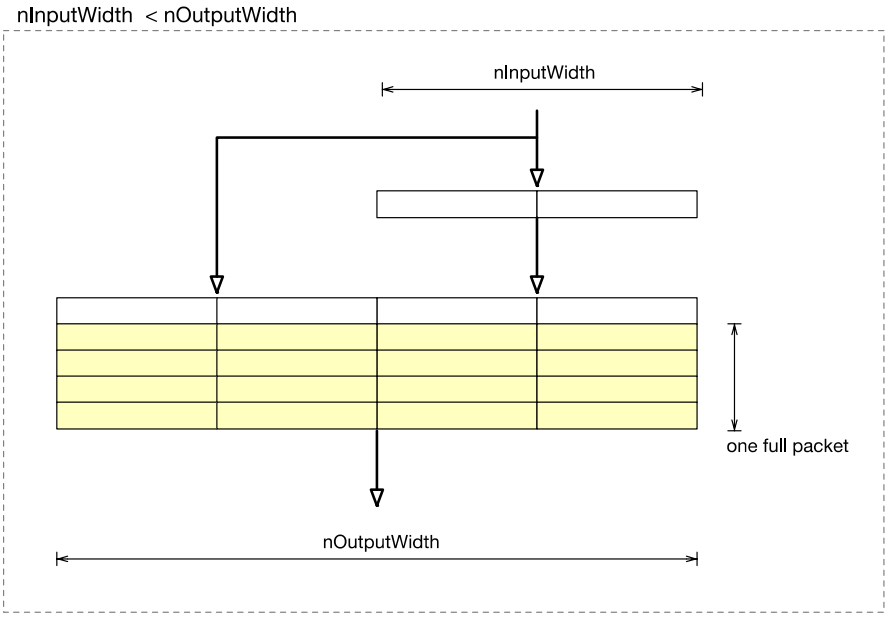


FIGURE 3-11: WIDTHADAPTER WITH RATEADAPTATION

3.4.2 Mapping CDTI to SMI field

CDTI connects to other concerto units via a Symphony Messaging Interface (SMI), which encapsulates concerto messages as shown inTable 3-4.

TABLE 3-4 SMI TO CONCERTO MESSAGE MAPPING

SMI NDP Mapping	
SMI	**Concerto Message**
ndp_targ_id	{target_id, fPortId}
ndp_src_id	{initiator_id, fPortId}
ndp_steer	steering
ndp_msg_tier	t_tier
ndp_msg_qos	ql
ndp_pri	priority
ndp_len	wNdpBits parameter
ndp_ndp	See cpr/concMsg. This can include m_prot if SMI message protection is enabled. Note wMProt is based on the NDP width per message type.
ndp_dp_present	1 for DTWReq and DTRReq, otherwise 0
ndp_msg_id	message_id
ndp_msg_user	{Header protection bits (if enable SMI header protection) }
ndp_msg_err	0 (unused)
SMI DP Mapping	
SMI	**Concerto Message**
dp_last	last
dp_data	data
dp_user	NUM_DW_PER_BEAT x {dbad, dwid, dprot(if enable SMI data protection), be, user}

3.4.3 Mapping CDTI to ATP field

Ncore 3 supports only Header Parallel packet style.

- Header Parallel means the first beat of a packet contains both header and data information and if subsequent data beats are needed, the following beats contain no header. The width of the ATP link in the header parallel style is set by the width of the header and width of the Data Phit. No Extra packetization latency added in this style.
    - The interface width is sum of header packet payload and data payload. That is, ATP interface width will match with SMI interface width.
    - This is mainly to maximize performance while spending more area.

The ATP link is defined by the following parameters:

- Maximum packet size.
- Flit Data Width
- Packet Style
- Header parallel (referred to as parallel)
- Header serial (referred to as serial)
- The below is ATP signals in the network.

TABLE 3-5: ATP SIGNAL IN THE NETWORK

Signal	I/O	Description	Width	Optional	Parameterizable	Note
Valid	O	1'b1 Indicates the present beat has valid data from master	Number of VCs	No	Yes	1 in NCore 3
Ready	I	1'b1 Indicates slave can accept beat this cycle	Number of VCs	No	Yes	1 in NCore3
Fir	O	1'b1 indicates first beat of packet.	1	No	No	

st						
Last	O	1'b1 indicates last beat of packet.	1	No	No	
Bus	O	Contains the contents of the packet	Variable	No	Yes	
Pressure	O	The pressure of the present beat	Variable	Yes	Yes	Not in Core3
Prot	O	The protection bits of the output of the present beat	Variable	Yes	Yes	Not in Core3
Prob_b	I	The protection bits of the input of the present beat	Variable	Yes	Yes	Not in Core3

3.4.4 Connectivity mapping

This section goes over CDTI connectivity of different Ncore 3 units. It specifies how the connectivity must be optimized to take advantage of Ncore 3 unit and port interleaving commonality. Furthermore, it also specifies when certain connectivity can be removed by choice and how Ncore 3 units should report run time errors in these scenarios. The objective here is to optimize the network and remove any possible unused connectivity by the configuration.

In the following description, CN0 is for control network 0, CN1 is for control network 1, CN2 is for control network 2 and DN is for data network.

The mapping for the control and data network in four network, 3CN1DN configuration is shown in Table 3-6(without optimizations). Three networks, 2CN1DN configuration is achieved by combining CN1 and CN2

TABLE 3-6 SYSTEM CONTROL AND DATA NETWORK MAPPING

Unit	Control Network 0 (CN0)		Control Network 1 (CN1)		Control Network 2 (CN2)		Data Network (DN)	
	Tx Msg	Rx Msg	Tx Msg	Rx Msg	Tx Msg	Rx Msg	Tx Msg	Rx Msg
CAIU - CHI								
	CmdReq	StrReq	StrRsp	CmdRsp			DtwReq/ DtwDbgReq	DtrReq
	SysReq	SnpReq	SnpRsp	DtrRsp			DtrReq	
		SysReq	DtrRsp	CmpRsp				

Unit	Control Network 0 (CN0)		Control Network 1 (CN1)		Control Network 2 (CN2)		Data Network (DN)	
	Tx Msg	Rx Msg	Tx Msg	Rx Msg	Tx Msg	Rx Msg	Tx Msg	Rx Msg
			SysRsp	DtwRsp/ DtwDbgRsp				
				SysRsp				
CAIU - ACE								
	CmdReq	StrReq	StrRsp	CmdRsp			DtwReq/ DtwDbgReq	DtrReq
	SysReq	SnpReq	SnpRsp	DtrRsp			DtrReq	
	UpdReq	SysReq	DtrRsp	CmpRsp				
			SysRsp	DtwRsp/ DtwDbgRsp				
				SysRsp				
				UpdRsp				
NCAIU (ACE-Lite/ACE-Lite E with DVM)								
	CmdReq	StrReq	StrRsp	CmdRsp			DtwReq/ DtwDbgReq	DtrReq
	SysReq	SnpReq	SnpRsp	DtrRsp (only ACE-Lite E)			DtrReq (only ACE-Lite E)	
			DtrRsp	DtwRsp/ DtwDbgRsp				
				SysRsp				
NCAIU (AXI with proxy cache)								
	CmdReq	StrReq	StrRsp	CmdRsp			DtwReq/ DtwDbgReq	DtrReq
	SysReq	SnpReq	SnpRsp	DtrRsp			DtrReq	
	UpdReq	SysReq	DtrRsp	DtwRsp/ DtwDbgRsp				
			SysRsp	SysRsp				
				UpdRsp				
NCAIU (AXI without proxy cache and ACE-Lite/ACE-Lite E without DVM)								
	CmdReq	StrReq	StrRsp	CmdRsp			DtwReq/ DtwDbgReq	DtrReq
			DtrRsp	DtrRsp (only ACE-Lite E)			DtrReq (only ACE-Lite E)	
				DtwRsp/ DtwDbgRsp				
DCE								
	StrReq	CmdReq	CmdRsp	SnpRsp	MrdReq	MrdRsp		
	SnpReq	SysReq	SysRsp	StrRsp	RbrReq	RbrRsp		
	SysReq	UpdReq	UpdRsp	SysRsp	RbuRsp	RbuReq		
DII								
	StrReq	CmdReq	NCCmdRsp	StrRsp			DtrReq	DtwReq
			DtwRsp	DtwDbgRsp			DtwDbgReq	
				DtrRsp				
DMI								
	StrReq	CmdReq	NCCmdRsp	StrRsp	MrdRsp	MrdReq	DtrReq	DtwReq
			DtwRsp	DtwDbgRsp	RbrRsp	RbrReq	DtwDbgReq	
				DtrRsp	RbuReq	RbuRsp		
DVE								
	StrReq	CmdReq	NCCmdRsp	StrRsp				DtwReq/ DtwDbgReq
	SnpReq	SysReq	CmpRsp	SnpRsp				
			DtwRsp/ DtwDbgRsp					
			SysRsp					

Connectivity between different Ncore 3 units is shown inTable 3-7. The top row and the left most column specify the Ncore 3 unit names, the intersection point specifies the networks that connect the two units. Intersection points that are empty signifies that there are no connections between the corresponding units.

TABLE 3-7 CONNECTIVITY MAPPING

	CAIU	NCAIU	DCE	DII	DMI	DVE
CAIU	DN CN1	DN CN1	CN0 CN1	CN0 CN1 DN	CN0 CN1 DN	CN0 CN1 DN
NCAIU	DN CN1	DN CN1	CN0 CN1	CN0 CN1 DN	CN0 CN1 DN	CN0 CN1 DN
DCE	CN0 CN1	CN0 CN1			CN2	
DII	CN0 CN1 DN	CN0 CN1 DN				
DMI	CN0 CN1 DN	CN0 CN1 DN	CN2			
DVE	CN0 CN1 DN	CN0 CN1 DN				

Detailed TX and RX connectivity between Ncore 3 units for different networks are shown in Table 3-8, Table 3-9, Table 3-10 and Table 3-11. The top row and the left most column specify the Ncore 3 unit names, the intersection point specifies the connectivity. The direction of connectivity is from column name perspective, i.e. the port names TX/RX specifies the connectivity on the column unit.

TABLE 3-8 CN0 TX RX CONNECTIVITY MAP

	CAIU CHI	CAIU ACE	NCAIU AXI W Cache	NCAIU AXI No Cache	NCAIU ACE-Lite	NCAIU ACE-Lite-E	DCE	DII	DMI	DVE
CAIU CHI							TX/RX	TX/RX	TX/RX	TX/RX
CAIU ACE							TX/RX	TX/RX	TX/RX	TX/RX
NCAIU AXI W Cache							TX/RX	TX/RX	TX/RX	
NCAIU AXI No Cache							TX/RX	TX/RX	TX/RX	
NCAIU ACE-Lite							TX/RX	TX/RX	TX/RX	TX/RX(if DVM both)
NCAIU ACE-Lite-E							TX/RX	TX/RX	TX/RX	TX/RX(if DVM both)
DCE	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX				
DII	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX				
DMI	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX				
DVE	TX/RX	TX/RX			TX/RX (if DVM both)	TX/RX (if DVM both)				

TABLE 3-9 CN1 TX RX CONNECTIVITY MAP

	CAIU CHI	CAIU ACE	NCAIU AXI W Cache	NCAIU AXI No Cache	NCAIU ACE-Lite	NCAIU ACE-Lite-E	DCE	DII	DMI	DVE
CAIU CHI	TX/RX	TX/RX	TX/RX	TX	TX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX
CAIU ACE	TX/RX	TX/RX	TX/RX	TX	TX	TX	TX/RX	TX/RX	TX/RX	TX/RX
NCAIU AXI W Cache	TX/RX	TX/RX	TX/RX	TX	TX	TX	TX/RX	TX/RX	TX/RX	TX
NCAIU AXI No Cache	RX	RX	RX				TX/RX	TX/RX	TX/RX	TX
NCAIU ACE-Lite	RX	RX	RX				TX/RX	TX/RX	TX/RX	TX/RX(if DVM)
NCAIU ACE-Lite-E	TX/RX	RX	RX				TX/RX	TX/RX	TX/RX	TX/RX(if DVM)
DCE	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX				
DII	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX				TX
DMI	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX				TX
DVE	TX/RX	TX/RX	RX	RX	TX(if DVM)/RX	TX(if DVM)/RX		RX	RX	

TABLE 3-10 CN2 TX RX CONNECTIVITY MAP

	CAIU CHI	CAIU ACE	NCAIU AXI W Cache	NCAIU AXI No Cache	NCAIU ACE-Lite	NCAIU ACE-Lite-E	DCE	DII	DMI	DVE
CAIU CHI										
CAIU ACE										
NCAIU AXI W Cache										
NCAIU AXI No Cache										
NCAIU ACE-Lite										
NCAIU ACE-Lite-E										
DCE									TX/RX	
DII										
DMI							TX/RX			
DVE										

TABLE 3-11 DN TX RX CONNECTIVITY MAP

	CAIU CHI	CAIU ACE	NCAIU AXI W Cache	NCAIU AXI No Cache	NCAIU ACE-Lite	NCAIU ACE-Lite-E	DCE	DII	DMI	DVE
CAIU CHI	TX/RX	TX/RX	TX/RX	RX	RX	TX/RX		TX/RX	TX/RX	RX
CAIU ACE	TX/RX	TX/RX	TX/RX	RX	RX	RX		TX/RX	TX/RX	RX
NCAIU AXI W Cache	TX/RX	TX/RX	TX/RX	RX	RX	RX		TX/RX	TX/RX	RX
NCAIU AXI No Cache	TX	TX	TX					TX/RX	TX/RX	RX
NCAIU ACE-Lite	TX	TX	TX					TX/RX	TX/RX	RX
NCAIU ACE-Lite-E	TX/RX	TX	TX					TX/RX	TX/RX	RX
DCE										
DII	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX				RX
DMI	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX	TX/RX				RX
DVE	TX	TX	TX	TX	TX	TX		TX	TX	

3.4.5 Connectivity Optimization

This section goes over connectivity optimizations that must be applied to remove connections from the full connectivity described in Connectivity mapping section.

3.4.5.1 Removing connectivity

Depending on the system requirements a customer may choose to design a system where only a subset of AIUs may talk to only a subset of DIIs, this includes CSR configuration DII. Following is required

SW requirements:

- Provide GUI/TCL way to delete connections (all routes for all messages) between any AIUs and DIIs with following restriction
  - If an AIU is specified as CSR access capable AIU where parameter fnCsrAccess is set, then the customer must not be able to delete connection to CSR configuration DII
- Make sure the parameter hexAiuDiiVec is set correctly once the connections are deleted

HW requirements:

- If a transaction gets decoded to a DII to which the AIU is not connected based on port tie offs associated with parameters hexAiuDiiVec; then report an error as address decode error with additional information as specified for error type code 0x7

3.4.5.2 AIU – DCE Connectivity Optimization

In cases where interleaving address bit commonality is present between an AIU group (AIU ports marked as interleaved) and DCE interleaving then the connection between them must be optimized.

- If interleaving granularity is same and all address bits match, then optimization must be implemented i.e. connectivity becomes one to one
- If interleaving granularity is same and necessarily all address bits do not match but at-least one or more bit at the same index match, then optimization must be implemented. The level optimization depends on number of address bits that match.
- If interleaving granularity is same and all address bits do not match, then no optimization is required

Detailed examples for different combinations of 2-way and 4-way interleaving are shown inTable 3-12. In the example AIUs and DCEs are numbered as follows for 4-way interleaving

- 00 → AIU0, DCE0
- 01 → AIU1, DCE1
- 10 → AIU2, DCE2
- 11 → AIU3, DCE3

TABLE 3-12 AIU DCE INTERLEAVING EXAMPLE

Scenario	AIU interleaving address bits	DCE interleaving address bits	Optimization	Comment
2-way interleaved AIU and 2-Way interleaved DCE	X	X	AIU0 → DCE0 AIU1 → DCE1	One to one completely optimized connectivity
	X	Y	AIU0 → DCE0, DCE1 AIU1 → DCE0, DCE1	No optimization complete cross bar
4-way interleaved AIU and 4-way interleaved DCE	X, Y	X, Y	AIU0 → DCE0 AIU1 → DCE1 AIU2 → DCE2 AIU3 → DCE3	One to one completely optimized connectivity
	X, Y	X, A	AIU0 → DCE0, DCE1 AIU1 → DCE0, DCE1 AIU2 → DCE2, DCE3 AIU3 → DCE2, DCE3	Partially 1 to 2 optimized as the MSB bit is same and the LSB bit is not same
	X, Y	A, Y	AIU0 → DCE0, DCE2 AIU1 → DCE1, DCE3 AIU2 → DCE0, DCE2 AIU3 → DCE1, DCE3	Partially 1 to 2 optimized as the LSB bit is same and the MSB bit is not same
	X, Y	A, B	AIU0 → DCE0, DCE1, DCE2, DCE3 AIU1 → DCE0, DCE1, DCE2, DCE3 AIU2 → DCE0, DCE1, DCE2, DCE3 AIU3 → DCE0, DCE1, DCE2, DCE3	Complete cross bar, both bits are different.
2-way interleaved AIU and 4-Way interleaved DCE	X	X, A	AIU0 → DCE0, DCE1 AIU1 → DCE2, DCE3	Partially 1 to 2 optimized as the MSB bit is same
	X	A, X	AIU0 → DCE0, DCE2 AIU1 → DCE1, DCE3	Partially 1 to 2 optimized as the LSB bit is same
	X	A, B	AIU0 → DCE0, DCE1, DCE2, DCE3 AIU1 → DCE0, DCE1, DCE2, DCE3	No optimization complete cross bar
4-way interleaved AIU and 2-Way interleaved DCE	X, Y	X	AIU0 → DCE0 AIU1 → DCE0 AIU2 → DCE1 AIU3 → DCE1	2 to one optimization as the MSB bit matches
	X, Y	Y	AIU0 → DCE0 AIU1 → DCE1 AIU2 → DCE0 AIU3 → DCE1	2 to one optimization as the LSB bit matches
	X, Y	A	AIU0 → DCE0, DCE1 AIU1 → DCE0, DCE1 AIU2 → DCE0, DCE1 AIU3 → DCE0, DCE1	No optimization complete cross bar 1) only LSB bit matches 2) none of the bits match

SW requirements:

- Make sure the parameter hexAiuDceVec is set correctly once the connections are optimized
- Compute nDceConnectedCas and nAiuConnectedDces

HW requirements:

- If a transaction gets decoded to a DCE to which the AIU is not connected based on port tie offs associated with parameter hexAiuDceVec; then report an error as address decode error with additional information as specified for error type code 0x7

3.4.5.3 AIU/DCE – DMI Connectivity Optimization

In cases where interleaving address bit commonality is present between an AIU group (AIU ports marked as interleaved) / DCEs and DMI interleaving, then the connection between them must be optimized.

- If interleaving granularity is same and all address bits match, then optimization must be implemented i.e. connectivity becomes one to one
- If interleaving granularity is same and necessarily all address bits do not match but at-least one or more bit at the same index match, then optimization must be implemented. The level optimization depends on number of address bits that match.
- If interleaving granularity is same and all address bits do not match, then no optimization is required

In the case of DMI commonality must be considered across the different interleaving options specified, if there is an intersection then optimization can be done if not then no optimization applies.

SW requirements:

- Make sure the parameters hexAiuDceVec, hexAiuDmiVec and hexDceDmiVec are set correctly once the connections are optimized
- Compute nDmiConnectedDces and nDceConnectedDmis

HW requirements

- If a transaction gets decoded to a DCE/DMI to which the AIU/DCE is not connected based on port tie offs associated with parameters hexAiuDceVec, hexAiuDmiVec and hexDceDmiVec; then report an error as address decode error with additional information as specified for error type code 0x7

3.4.5.4 AIU – AIU Connectivity Optimization

In cases where interleaving address bit commonality is present between interleaved AIUs, then the connection between them must be optimized based on following:

- AIUs within a single interleaved group must have all connectivity between them optimized.
- AIUs across interleaved groups; if the interleaved address bit/s are not same for any two or more AIUs then all connectivity between these AIUs must be optimized. This is irrespective of the granularity of interleaving

Detailed Example is shown in Figure 3-12.

In group 0, AIU0 and AIU1 are 2 way interleaved with bit 7. As they are within a single group, they do not have any connectivity between them.

In group 1, AIU2, AIU3, AIU4 and AIU5 are 4 way interleaved with bit 7,8. As they are within a single group, they do not have any connectivity between them.

As there is commonality between the two groups, connections are optimized as shown where AIU0 talks with AIU2 and AIU3 as they share the interleaving bit 7 with a value of 0, same applies between AIU1 and AIU4, AIU5.

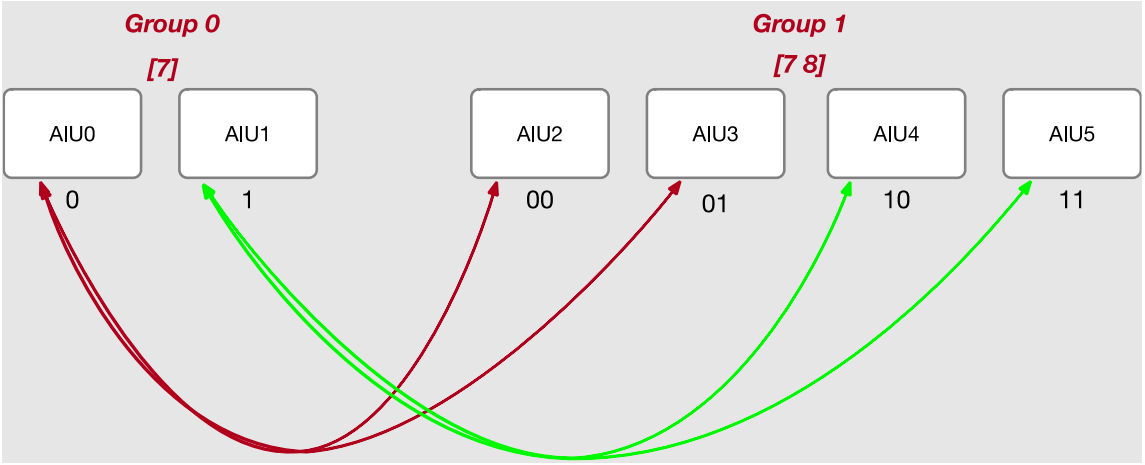


FIGURE 3-12: AIU TO AIU CONNECTIVITY

3.4.5.5 Snoop filter optimizations

The interleaving presents an opportunity to optimize snoop filter sharer vector. Details of possible optimization are discussed in Opens section. At this time these optimizations will not be implemented by Ncore 3, apart from following restrictions

1. Interleaved AIUs within a group must be assigned to the same Snoop filter. This restriction makes sense as it is expected that all agents connecting via interleaved ports within a group will have the same or shared cache structure.
2. Number of ways within a snoop filter must be limited to multiple of 4, with a max value of 32. This restriction is to reduce possible combinations.

3.4.5.6 Credit Optimization

This section goes over credit optimizations that must be implemented to compliment the connectivity optimizations. Credits in question here are build time defined credits specified for DCE i.e., nDceRbCredits and

nAiuSnpCredits. Other credits do not get affected as they will be SW defined at run time. Credit parameter specification stays the same with following changes in software and hardware.

SW changes:

- nDceRbCredits are used to derive the coherent write buffer size in DMI. The size of this buffer must be limited to the sum of nDceRbCredits of only the DCEs that are connected to the DMI.
- nAiuSnpCredits are used to derive the depth of the STT table (nSttCtrlEntries) in AIUs. The size of this STT table must be limited to sum of nAiuSnpCredits of only the DCEs that are connected to the AIU

HW changes:

- Implement RbCredit counters up to the number specified by the parameter nDceConnectedDmis. The offset value for Rb credits must be used from the DCE port tie off parameter hexDceDmiRbOffset.
- Implement snoop credit counters up to the number specified by the parameter nDceConnectedCas. (This is optional, we can defer it to reduce RTL change, note that this change may help in timing)

Note: AIUs have command credits counters, these can be optimized based on actual targets (DCEs, DMIs, DIIs) connected to the AIU.

3.4.5.7 Opens

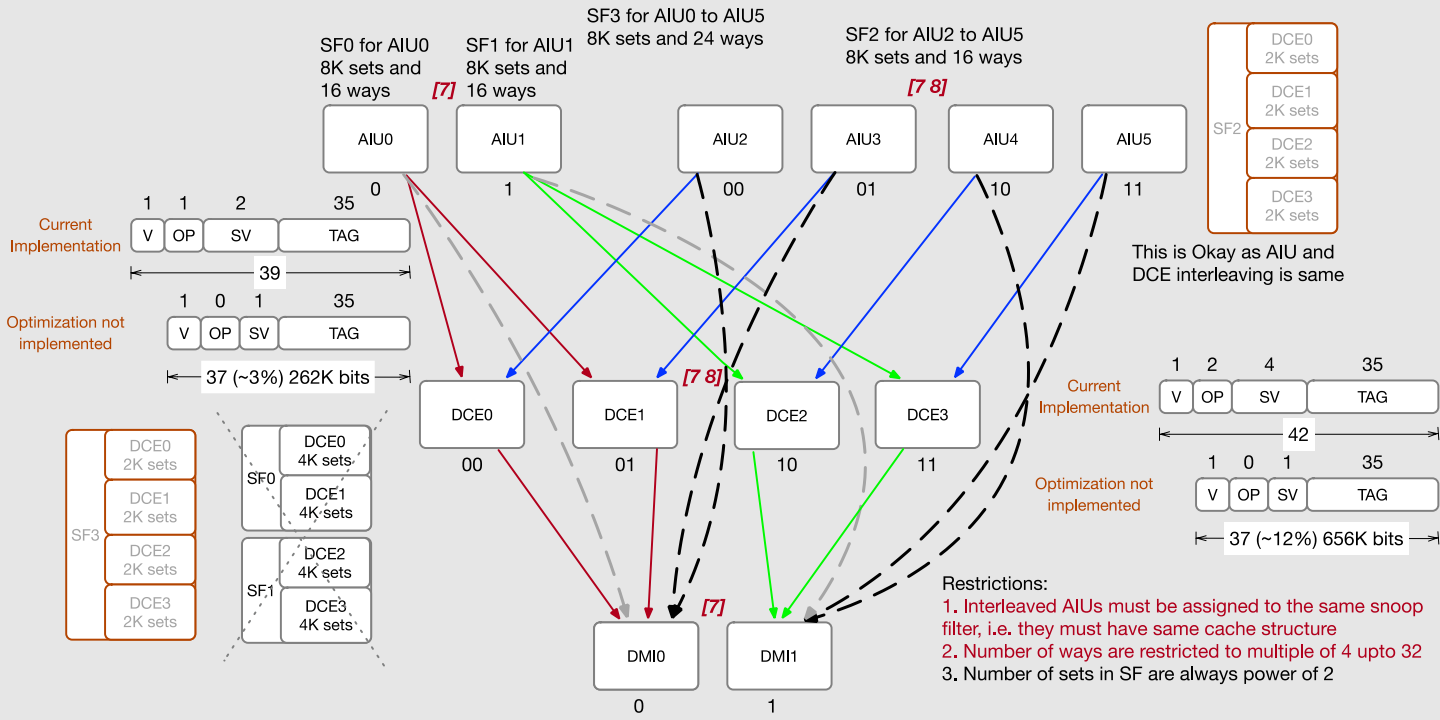


FIGURE 3-13: CDTI IMPLEMENTATION

3.5 Ncore CSTI Network

The CSTI network is dedicated to access configuration, status and error registers in Ncore 3. It is composed of two sub networks:

1. Request network
2. Response network

The CSTI network shall ensure that data written on configuration registers can be read back to confirm correct operation. It is required to implement safety measures.

Similarly to CDTO network, the CSTI network implements parity or ECC checks.

3.5.1 CSTI Blocks

In addition to the four blocks listed in the CDTI network, the CSTI network adds two more blocks

3.5.1.1 AXI initiator

There is one AXI initiator that is tied to a special DII unit using an AXI interface that is used to initiate all transactions into the CSTI.

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

The AXI initiator is 32 bits wide.

3.5.1.2 APB target

The widths of all the CSTI blocks are fixed at 32 bits, which is a width that is not available in the CDTI network. Please refer to Legato section for details on these components.

Every block in Ncore 3 that contains a register that can be read or written will have an APB port on it. The APB target block connects this port to the CSTI network. It converts requests from the AXI Initiator to APB transactions on the APB interface returns the responses to the AXI Initiator. All 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.

APB targets are 32 bits wide.

3.6 Addressing in CTFs

3.6.1 Fabric Unit Identification

When an Ncore 3 unit sends a message to another Ncore 3 unit, the receiver of the message must be uniquely identified. For this purpose, each Ncore 3 unit in the Ncore 3 platform system is also identified by a *globally unique identifier* value called the ***Fabric Unit Id (FUID)***. This unique identifier is used as part of the Initiator Id or Target Id in the header of a Concerto C message.

There is no specific requirement on the way the units are assigned identifiers.

Example:

FUIDs in an Ncore 3 system are numbered from 0 to (*N*-1), where *N* equals the total number of Ncore 3 units, and each unit, based on its type, is numbered within this name space as follows:

- 0 to (*A*-1): AIUs, where *A* equals the number ofC-AIUs
- *A* to (*A*+*B*-1): NC-AIUs, where *B* equals the number ofNC-AIUs
- (*A*+*B*) to (*A*+*B*+*D*-1): DCEs, where *D* equals the number of DCEs
- (*A*+*B*+*D*) to (*A*+*B*+*D*+*M*-1): DMIs, where *M* equals the number of DMIs
- (A+B+D+M) to (A+B+D+M+I-1): DIIs, where I equals the number of DIIs
- (*A*+*B*+*D*+*M*+*I*): DVE. An Ncore 3 system is expected to have 1 DVE.

Note: *A*+*B*+*D*+*M* +I+1 equals *N*.

Primary input: Each Ncore 3 unit must implement the ability to receive its FUIDs as a primary input. Primary input means input using strap pins (tie-off signal), to get input value not at configuration time but at runtime. The main purpose of this is to support stamping.

FUIDBase + FUNITOffset: As described earlier Ncore 3 could have more than one network. Also, we could stamp Ncore 3 network using FUIDs. In that case, to support unique ID to each Ncore 3 unit, we are using FUnitBase and FUIDOffset.

- FUIDBase is FUID for each network.
- FUIDOffset is supplied as a parameter (or as a tie-off signals) to provide offset for the network to provide unique ID.

  NOTE: FUIDOffset = 0 in NCore 3

3.6.2 Message Target Identification

An Ncore unit, in general, may connect to a given CDTI network via one or more terminal, TX and/or RX, and to one or more of CDTI networks, as necessary.

The Target Id field in the header of the message must uniquely identify a destination RX terminal block within a CDTI network. To achieve that, the Target Id field is defined to be a concatenation of two subfields:

- Target Unit Id: A unique identifier for the unit within the Ncore system. (FUnitId)
- Target Port Id: A unique identifier associated with the RX SMI interface of the said unit on which the message will be received. Target Port Ids are assigned integer values (0, 1, 2, 3, etc.) starting at 0. Thus, if a unit carries one or more RX SMIs, then there must be a Target Port Id of0.

If the unit attaches to a network via multiple RX SMIs, then the Target Port Id is used to disambiguate between those ports.

The Target Port Id field is 0 or more bits long, as necessary. Its length is determined by the maximum of the minimum number of bits needed by any of the units connected to the network to uniquely identify all of its RX ports (= MAX ($\log_2$(# RX ports for a network unit)). This length is applied to all the Target Ids for that network.

Thus, for example, if a network has one unit with 2 RX ports, one with 3 RX ports, and all others with 1 RX port, the width of the Target Port Id field must be 2 bits, as needed by the unit with 3 RX ports.

3.6.3 Message Initiator Identification

The message header also includes the Initiator Id. It is provided so that target unit can send a response message back to the initiator.

Like the Target Id, Initiator Id is also a concatenation of two sub-fields:

- Initiator Unit Id: A unique identifier for the unit within the Ncore system.
- Initiator Port Id: A unique identifier associated with the TX SMI interface of the said unit on which the message is beingtransmitted.

The Initiator Port Id field is 0 or more bits long, as necessary. Its length is determined by the maximum of the minimum number of bits needed by any of the units connected to the network to uniquely identify all of its TX ports (= MAX ($\log_2$(# TX ports for a network unit)). This length is applied to all the Initiator Port Ids for that network.

4 Protocol Layer

In a shared memory multiprocessor containing caches, coherency support is necessary to ensure correctness of computations. In addition, co computation requires the enforcement of consistency among values of multiple data locations accessed. This chapter describes the support for coherency and consistency within an Ncore 3 system.

4.1 Introduction

SoCs include function blocks that initiate storage accesses. These function blocks are called ***agents***. Agents can include ***coherent agents*** such as processors or processor clusters that incorporate cache hierarchies that have to be kept coherent, as well as ***non-coherent agents*** that do not themselves hold copies of data that have to be kept coherent, but nonetheless may make coherent accesses to system memory and thereby do interact with coherent agents and access contents of their cache hierarchies. Both coherent and non-coherent[2] agents can also make storage accesses that do not require enforcement of coherency, as well as other types of requests.

SoC function blocks that don't initiate storage accesses but might receive storage accesses and might fulfill them are referred to as ***target devices***.

Ncore 3 enables implementations of cache coherency subsystems for SoCs that can support coherent accesses, as well as non-coherent accesses via them; latter to account particularly for providing efficient and full-function connectivity for agents that might use a common pathway to issue both types of accesses and for connecting devices, such as memory, that may be the targets for both types of accesses.

Ncore 3 units implement a messaging protocol called the ***Concerto C Messaging Protocol (CCMP)*** to execute storage accesses and other requests, and their semantic actions.

Only a subset of messages in CCMP and associated semantics are utilized in enforcement of coherency in Ncore 3. This subset is referred to as ***Concerto Coherency Protocol (CCP)***.

Agents in an SoC are connected via their ***native interfaces*** to ***Agent Interface Units (AIUs)*** in the Ncore 3 systems[3]. The agents make their requests over their native interface. The AIUs translate these into CCMP messages. These original messages result in subsequent messages being exchanged among appropriate NCore units.

---

[2] IO and peripheral devices are non-coherent devices.

[3] For example, a processor may connect via CHI interface or a peripheral device might connect via an AXI interface.

All Ncore 3 units, AIUs, DCEs, DMIs, DIIs, and DVEs, participate in the CCMP are thus considered part of the ***CCMP-domain***.

Those units involved in the protocol activity related to coherency enforcement are also considered to be part of the ***coherency-domain***. These typically include AIUs, DCEs, and DMIs.

CCMP also supports maintenance of coherence among ***MMU TLBs***. That coherency domain includes AIUs and the DVE.

4.2 Storage Classification

Ncore 3 categorizes the totality of storage within an SoC into two classes:

- System memory, and
- Peripheral storage

Typically, system storage is implemented via Dynamic or Static RAM devices. Peripheral storage typically includes Control and Storage registers within peripheral devices, although it might include buffers and RAM devices.

In a Concerto C based system, ***system memory*** is refers to storage that is ***well-behaved***.[4] For such storage a write to a given memory location updates the value stored at that location and a read from a memory location returns the last value written to that memory location. Furthermore, performing an access to system memory does not have side-effects.

Peripheral storage is not guaranteed to be well-behaved. As such, data values in such locations may not be persistent, e.g. two reads without an intervening write may not return the same value, or a read may not return the last value written. Additionally, a read or write access may cause side-effects.

In Concerto C, coherency is enforced only on system memory locations, although these locations may also be accessed via Concerto C protocol without requiring that coherence be enforced during such accesses.

System memory is divided into equal-sized data granules called ***cachelines***, on which coherence is maintained. Each cacheline consists of a contiguous block of a power-of-two number of bytes and is uniquely identified by a ***cacheline address***, which is a system memory address aligned to the size, in bytes, of a cacheline. An agent may buffer a copy of a cacheline, or a ***cacheline copy***, in its cache.[5]

---

[4] Well-behaved storage is also referred to as “normal” storage.

[5] Depending on the context, the term cacheline may refer to a container for data granule or the data granule itself.

Memory accesses are classified relative to the size and alignment of a cacheline. A memory access that operates on all the bytes within a cacheline-aligned data granule is a ***full-cacheline access***. A memory access that operates on a, possibly non-contiguous, subset of the bytes within a cacheline-aligned data granule is a ***partial-cacheline access***. Finally, a memory access that operates on bytes in two or more cacheline-aligned data granules is classified as a ***multiple-cacheline access***.

Concerto C does not support multi-cache line accesses or those that span a cacheline address boundary. Ncore 3 supports 64-Byte cache lines.

4.3 Coherency protocol features

4.3.1 Cacheline states

The data that can be kept coherent is located in system memory. CCP maintains coherency among copies of a cacheline in agent caches and its value in system memory ensuring that the latest value assigned to the data granule is never lost.

The CCP allows a coherent agent to hold a partial cacheline, but for the sake of the coherency the protocol doesn't distinguish it from the agent holding a full copy of the cacheline.

The CCP supports the following properties for cachelines in caches within an agent:

- **Invalid vs. Valid:** An ***invalid*** cacheline copy *must* not be accessed by a caching agent, while a ***valid*** cacheline copy may be accessed by a caching agent. A caching agent may consume cacheline data from a valid cacheline copy but may only modify cacheline data depending on the other properties of the cacheline copy.
- **Owned vs. Non-owned:** An ***owned*** cacheline copy is a valid cacheline copy that identifies an agent as the ***cacheline owner***, who, depending on the other properties of the cacheline copy, is responsible for forwarding data in response to any snoop messages and for updating memory on a replacement. A ***non-owned*** cacheline copy is a valid cacheline copy that identifies an agent as a ***cacheline sharer***, who has no responsibilities with respect to the protocol. There is at most one owned cacheline copy in the system at any time.
- **Shared vs. Unique:** A ***shared*** cacheline copy is a valid cacheline copy that may be valid in another agent's cache. A ***unique*** cacheline copy is a valid cacheline copy that *must* not be valid in any other agent's cache. A caching agent may modify cacheline data only if its copy of the cacheline is unique, and that modification may occur without any additional protocol transactions. There is at most one unique cacheline copy in the system at any time, and a unique cacheline copy implies the cacheline copy is also owned.
- **Clean vs. Dirty:** A ***clean*** cacheline copy is a valid cacheline copy that is consistent with respect to the latest copy of data, which may be present in either another agent's cache or system memory. A ***dirty*** cacheline copy is a valid cacheline copy that represents the latest copy of data and is known to be inconsistent with respect to system memory. Upon modifying a clean cacheline copy, a caching agent changes the state to dirty, and upon replacing a dirty cacheline copy, a

caching agent must update memory. There is at most one dirty cacheline copy in the system at any time, and a dirty cacheline copy implies the cacheline copy is also owned.

A cache line can actually be held in a cache in a state that is a combination of the above states. Examples are Shared-Clean, Unique-Clean, and Shared-Dirty.

A cacheline state may have another property when it is held in a cache.

- **Empty vs. Partial vs. Full:** A cacheline may be held validly in a cache such that the cache hierarchy contains all of its bytes, only some of its bytes, or none.

This state adds two more legal combination states of Unique-Clean-Empty, Unique-Dirty-Partial. These are not explicitly represented in Concerto C coherency protocol, but are supported effectively none-the-less by simply treating them as Unique-Clean and Unique-Dirty, respectively.

4.3.2 Supported coherency protocol classes

Not all the 16 possible states based on the above state variables associated with a cache line are meaningful or legal within the CCP. But a large variety of well known coherency protocol classes, such as MESI, MOESI, etc. can be crafted by further restricting the possible legal states and transitions between them to smaller subsets. The following table shows the protocol created by limiting the set of permitted states to be a subset of legal states.

TABLE 4-1:SUBSET OF PROTOCOL CLASSES

Protocol Class	UD	UC	OD	OC
MSI	Yes	No	No	No
MFSI	Yes	No	No	Yes
MOSI	Yes	No	Yes	No
MOSIF	Yes	No	Yes	Yes
MESI/MEI	Yes	Yes	No	No
MESIF	Yes	Yes	No	Yes
MOESI	Yes	Yes	Yes	No

4.3.3 Elements of Coherency Management

Cacheline copies are kept coherent by one or more ***Distributed Coherence Enforcement Units (DCEs)***. Each cacheline address is associated with a single DCE, known as the ***Home DCE*** for that cacheline address. The Home DCE acts as the ***point of serialization*** for, and enforces coherence on, the given cacheline address. In a system configured with multiple DCEs, cacheline address are distributed across the DCEs using an Ncore-instance specific mapping function.

Ncore 3 implements a ***directory-based coherency protocol***. A DCE may implement a system ***directory partition*** for tracking and managing the contents of caching agent caches with respect to the addresses associated with the DCE. The directory can be configured to track the presence of a cacheline in various caches in the system as well as a specific cache which might hold the cacheline in an owner state.

When an AIU receives a coherent request from an agent, the AIU creates an appropriate CMDreq message and sends it to a DCE unit. The DCE unit consults the directory to determine which of the coherent agents must be snooped and sends the snoop messages to the corresponding AIUs.

The AIU receiving the snoops, snoop their coherent agents in turn and return their individual snoop responses. The DCE uses the responses to update its directory to reflect the new cache states.

The directory partition implemented by the DCE acts as a ***snoop filter***, allowing the DCE to minimize the number of snoops that are sent out.

The CCMP primarily operates as a ***Write-invalidate*** style coherency protocol. Such a protocol requires that an update to a coherently maintained location by an agent results in invalidation of other copies of the cacheline that contains that location.

The CCMP also employs a ***Write-Update*** style coherency protocol to efficiently support ***Write-stashing operations*** that may be issued on ACE-Lite-E and CHI-B native interfaces that allow data to be deposited directly into processor or system caches without first being written into memory.

4.4 Proxy cache

An NCAIU with an AXI interface may also be configured to instantiate a Proxy Cache that can satisfy native agent transaction from non-coherent agent.

Upon receiving a native agent transaction, an NCAIU configured with a proxy cache decode the memory type of the native agent transaction and breaks the native agent transaction into cacheline-sized cacheline transaction. Expressed on the AxCACHE signal, the memory type of the native agent transaction determines the visibility requirement of the cacheline transactions with respect to the proxy cache and memory. The proxy cache supports Normal Write-through (WT) and Normal Write-back (WB) memory types; however, Device and Non-Cacheable memory types are not supported. As a result, all native agent transactions received by an NCAIU configured with a proxy cache perform an IO cache lookup that determines whether or not the cache contains a copy of cache line.

This section describes the changes to caching model of Ncore proxy cache. Two changes are covered in this document.

1. Owner ship transfer
2. Pseudo exclusive proxy cache

The general caching model stays as is, which is MOESI. No new parameters are introduced but the NcMode parameter is deprecated in this release. It must be set to 0 coherent mode when proxy cache is enabled.

4.4.1 Proxy cache allocation and visibility policies

Proxy cache allocation policy is controlled only by AxCache value of the transaction. Table 4-2 gives details on how proxy cache allocates transactions and sets visibility attributes based on AxCache values. If a cache line is allocated in the cache, then the concerto command going out on SMI must go with allocation attribute set to no allocate i.e. 0.

TABLE 4-2 CACHE ALLOCATION & VISIBILITY

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®/No(W)	Early	Write through no allocate / read allocate
0111	®(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

4.4.2 Transactions native to concerto

This section goes over how native transactions are translated to concerto transactions after proxy cache look up. Table 4-3 shows different cache look up actions. Note that in Table 4-3 “X” denotes don’t care. **SZ** denotes size of the transaction which can be partial (ptl) or full. **CS** denotes current state of the cache which can be SC (shared clean), SD (shared dirty or owned), UC (unique clean) and UD (unique dirty). **AP** denotes allocation policy where NA is no allocate and AL allocate, this is determined based on Table 4-2. **NS** denotes possible next state of the cache after the transaction finishes. **Conc Msg** specifies the message to be issued into the Ncore system.

TABLE 4-3 CACHE LOOKUP ACTIONS

Transaction	SZ	CS	AP	NS	Conc msg	Notes
Read	X	IX	NA	IX	CmdRdNITC	Miss no allocate (no cache case)
			AL	SC/SD/UC/UD	CmdRdVld	Miss allocate
		SC	X	-	-	Cache hit
		SD	X	-	-	Cache hit
		UD	X	-	-	Cache hit
		UC	X	-	-	Cache hit
Write	Ptl	IX	NA	IX	CmdWrUnqPtl	Miss no allocate (no cache case)
			AL	UD	CmdRdUnq	Miss allocate
		SC	X	UD	CmdRdUnq	Cache hit upgrade
		SD	X	UD	CmdRdUnq	Cache hit upgrade, drop data if SD is not lost in the interim and just upgrade to UD. (Data could be form memory and thus stale)

		UC	X	UD	-	Cache hit
		UD	X	UD	-	Cache hit
	Full	IX	NA	IX	CmdWrUnqFull	Miss no allocate (no cache case)
			AL	UD	CmdMkUnq	Miss allocate
		SC	X	UD	CmdMkUnq	Cache hit upgrade
		SD	X	UD	CmdMkUnq	Cache hit upgrade
		UD	X	UD	-	Cache hit
		UC	X	UD	-	Cache hit

For Write Ptl cache hit upgrade cases, current states SC or SD issues CmdRdUnq instead of CmdClnUnq. This is done to avoid the creation of UCE and UDP state, in the case where an intervening snoop invalidates the SC and SD state.

If a cache is not present, then the native commands are mapped to concerto commands using the row where allocation (AP) policy AP is NA and the next state (NS) is IX

The updated proxy cache must support access to DII, these transactions are non-coherent. Native read and write transactions hitting DII/DMI GPRS with NC bit set are simply translated to CmdRdNc and CmdWrNc respectively.

NcMode parameter will be deprecated and only the GPRS configuration will be honored.

Cache allocation and look up is followed based on AxCache bits for the transaction. Partial Miss allocate cases must be converted to no allocate case for non-coherent transactions. Non-coherent cache lookup actions are shown in Table 4-4. Note that final state transition for allocating read is always UC and on write is UD. This is done to make sure dirty write data is evicted back to the memory; the read data does not need to be evicted back to the memory if it is clean. This architecture does not grantee coherency/consistency if software mixes coherent and non-coherent transactions within the same address space without taking proper precautions.

TABLE 4-4 NON-COHERENT CACHE LOOKUP ACTIONS

Transaction	SZ	CS	AP	NS	Conc msg	Notes
Read	Ptl	IX	NA	IX	CmdRdNc	Miss no allocate (no cache case)
			AL	IX	CmdRdNc	Partial allocate converted no allocate on miss
		SC	X	-	-	Cache hit
		SD	X	-	-	Cache hit
		UD	X	-	-	Cache hit
		UC	X	-	-	Cache hit
	Full	IX	NA	IX	CmdRdNc	Miss no allocate (no cache case)
			AL	UC	CmdRdNc	Miss allocate
		SC	X	-	-	Cache hit
		SD	X	-	-	Cache hit
		UD	X	-	-	Cache hit
		UC	X	-	-	Cache hit
Write	Ptl	IX	NA	IX	CmdWrNcPtl	Miss no allocate (no cache case)

			AL	IX	CmdWrNcPtl	Partial allocate converted no allocate on miss
		SC	X	UD	-	Cache hit
		SD	X	UD	-	Cache hit
		UC	X	UD	-	Cache hit
		UD	X	UD	-	Cache hit
	Full	IX	NA	IX	CmdWrNcFull	Miss no allocate (no cache case)
			AL	UD	-	Miss allocate
		SC	X	UD	-	Cache hit
		SD	X	UD	-	Cache hit
		UD	X	UD	-	Cache hit
		UC	X	UD	-	Cache hit

Evictions of non-coherent transactions allocated to proxy cache do not need update commands to be sent to DCE, if they are sent, it results in extra command traffic on the network and does not have functional impact. Coherent transactions allocated to proxy cache do need update commands to be sent to DCE, they help in reducing snoop traffic by updating the snoop filter at the expense of extra command traffic on the network, if they are not sent then it may result in more snoop traffic in the network.

A CSR control bit must be implemented in the cache control register to enable/disable update commands to DCE. This bit must affect update commands in general irrespective of the original allocating transaction is coherent or non-coherent.

4.4.3 Proxy cache state transitions

Proxy cache state transitions are detailed in this section. Table 4-5 gives details of state transition at the proxy cache when a DTR is received from the Ncore system. Table 4-6 gives details of state transition based on the snoop received at the proxy cache. Table 4-6 also gives details regarding different DtrReq and DtwReq that may be issued. The UP column refers to unique presence, here “X” means don’t care, “Y” means a UP value of 11 with a match on MPF3 field or UP value of 01 and “N” means UP value of 11 with no match on MPF3 field. Currently UP values of 00 and 10 are reserved, they can be treated as “N” case but DCE should not be sending them out and are illegal for this release. Cases that are not mentioned in the tables are not legal.

TABLE 4-5: PROXY CACHE DTRREQ STATE TRANSITION

Transaction	CS	DtrReq	NS	Notes
Read	IX	DtrDataInv	IX	Data not cached
		DtrDataSCln	SC	
		DtrDataSDty	SD	
		DtrDataUCln	UC	
		DtrDataUDty	UD	

Write partial	IX/SC/SD	DtrDataUCln or DtrDataUDty	UD	Merge partial data and cache
Write full	IX/SC/SD	X	UD	Replace data with write data

TABLE 4-6:CACHE SNPREQ STATE TRANSITIONS

SnpReq	CS	NS	UP	RV	RS	DC	DT[1]	DT[0]	DtrReq	DtwReq
SnpClnDtr	IX	IX	X	0	0	0	0	0	-	-
	SC	SC	N	1	1	0	0	0	-	-
	SC	SC	Y	1	1	0	1	0	DtrDataSCln	-
	SD	SD	Y	1	0	0	1	0	DtrDataSCln	-
	UC	SC	Y	1	1	0	1	0	DtrDataSCln	-
	UD	SD	Y	1	0	0	1	0	DtrDataSCln	-
SnpNoSDInt	IX	IX	X	0	0	0	0	0	-	-
	SC	SC	N	1	1	0	0	0	-	-
	SC	SC	Y	1	1	0	1	0	DtrDataSCln	-
	SD	SD	Y	1	0	0	1	0	DtrDataSCln	-
	UC	SC	Y	1	1	0	1	0	DtrDataSCln	-
	UD	SD	Y	1	0	0	1	0	DtrDataSCln	-
SnpVldDtr	IX	IX	X	0	0	0	0	0	-	-
	SC	SC	N	1	1	0	0	0	-	-
	SC	SC	Y	1	1	0	1	0	DtrDataSCln	-
	SD	SC	Y	1	1	1	1	0	DtrDataSDty	-
	UC	SC	Y	1	1	1	1	0	DtrDataSCln	-
	UD	SC	Y	1	1	1	1	0	DtrDataSDty	-
SnpInvDtr	IX	IX	X	0	0	0	0	0	-	-
	SC	IX	N	0	0	0	0	0	-	-
	SC	IX	Y	0	0	0/1[a]	0/1[a]	0/1[b]	If (UP = 01) DtrDataUCln	If (UP =11) DtwDataCln
	SD	IX	Y	0	0	0/1[a]	0/1[a]	0/1[b]	If (UP = 01) DtrDataUDty	If(UP=11) DtwDataDty
	UC	IX	Y	0	0	1	1	0	DtrDataUCln	-
	UD	IX	Y	0	0	1	1	0	DtrDataUDty	-
SnpNitc	IX	IX	X	0	0	0	0	0	-	-
	SC	SC	N	1	1	0	0	0	-	-
	SC	SC	Y	1	1	0	1	0	If (TOF=CHI/AXI) DtrDataInv else DtrDataSCln	-

SnpReq	CS	NS	UP	RV	RS	DC	DT[1]	DT[0]	DtrReq	DtwReq
	SD	SD	Y	1	0	0	1	0	If (TOF=CHI/AXI) DtrDataInv else DtrDataSCln	-
	UC	UC	Y	1	0	0	1	0	DtrDataInv	-
	UD	UD	Y	1	0	0	1	0	DtrDataInv	-
SnpNitcCI	IX	IX	X	0	0	0	0	0	-	-
	SC	IX	N	0	0	0	0	0	-	-
	SC	IX	Y	0	0	0	1	0	DtrDataInv	-
	SD	IX	Y	0	0	0	1	1	DtrDataInv	DtwDataDty
	UC	IX	Y	0	0	0	1	0	DtrDataInv	-
	UD	IX	Y	0	0	0	1	1	DtrDataInv	DtwDataDty
SnpNitcMI	IX	IX	X	0	0	0	0	0	-	-
	SC	IX	N	0	0	0	0	0	-	-
	SC	IX	Y	0	0	0	1	0	DtrDataInv	-
	SD	IX	Y	0	0	0	1	0	DtrDataInv	-
	UC	IX	Y	0	0	0	1	0	DtrDataInv	-
	UD	IX	Y	0	0	0	1	0	DtrDataInv	-
SnpClnDtw	IX	IX	X	0	0	0	0	0	-	-
	SC	SC	X	1	1	0	0	0	-	-
	SD	SC	X	1	1	0	0	1	-	DtwDataDty
	UC	UC	X	1	0	0	0	0	-	-
	UD	UC	X	1	0	0	0	1	-	DtwDataDty
SnpInvDtw / SnpUnqStsh / SnpStshUnq	IX	IX	X	0	0	0	0	0	-	-
	SC	IX	X	0	0	0	0	0	-	-
	SD	IX	X	0	0	0	0	1	-	DtwDataDty
	UC	IX	X	0	0	0	0	0	-	-
	UD	IX	X	0	0	0	0	1	-	DtwDataDty
SnpInv/SnpInvStsh	X	IX	X	0	0	0	0	0	-	-
SnpStshShd	IX	IX	X	0	0	0	0	0	-	-
	SC	SC	X	1	1	0	0	0	-	-
	SD	SD	X	1	0	0	0	1	-	DtwDataDty
	UC	SC	X	1	1	0	0	0	-	-
	UD	SD	X	1	0	0	0	1	-	DtwDataDty

Notes:

a. Set if UP == 01

b. Set if UP ==11 and target matches

4.4.4 Sharer promotion

The previous implementation of NCore (Ncore3.2) in certain scenarios red data from DMI/DRAM even when a copy of the cacheline is available in the cache upstream. This specifically happened when snoop filter ends up tracking a cacheline as only sharer and no owners. This section describes the changes implemented to promote one of the sharers as an owner for snooping and data retrieval purposes.

4.4.4.1 Unique Presence

This section goes over the unique presence field in snoop request messages Table 4-7 gives details of different encodings of the field.

TABLE 4-7: UNIQUE PRESENCE FIELD

UP	Name	Description
00	Reserved	future use for Null filters.
01	Unique Presence	This is the only copy in the system as per the Snoop filter. The snooping AIU maybe able to upgrade the Dtr type being issued to the requesting AIU
10	Reserved	Reserved
11	Unique Permission	The AIU identified in the MPF3 field is authorized to send DtrReq to the requesting AIU

4.4.4.1.1 UP = 00

This option maybe used in future for a null filter implementation where snoops are issued one at time i.e., the first caching agent's SnpReq must receive a SnpRsp before the next caching agent's SnpReq is issued. In this case any snooped AIU is authorized to issue a DtrReq. Depending on the type of request, DCE may stop issuing more SnpReq, if the Snooped AIU already sent out a DtrReq.

4.4.4.1.2 UP = 01

Unique Presence option maybe set only when one unique copy of the is cache line is present in the system. The snooped AIU in this case may upgrade the Dtr type to be issued from shared to unique. This is a legacy option that will be carried forward, benefit of this option is not quantified.

4.4.4.1.3 UP = 10

This option is reserved, with this new change in architecture the older functionality of this option is merged into UP = 11 option. This is done to simplify specification and design.

4.4.4.1.4 UP = 11

Unique Permission option is set to identify the AIU that is authorized to send the DtrReq. The Identified AIUs Funit ID is specified in the MPF3 field. If an owner is present in the system, then the owner is identified in the MPF3 field, if there are no owners and one or more sharers are present then one of the sharers is identified in the MPF3 field. The Snooped AIU is authorized to send the DtrReq if its own Funit ID matches the Funit ID in the MPF3 field of the SnpReq.

4.4.4.2 DCE Details

Table 4-8 shows details on how the UP and MPF3 fields need to be set for different snoop request. X here denotes don't care, Implementation may set it zero or any random value.

TABLE 4-8: DCE UP AND MPF3 FILED SETTING

SnpReq	Condition	UP	MPF3	Notes
SnpClnDtr / SnpNoSDInt / SnpVldDtr /	Null filter	00	X	Future use
	Unique Owner/Sharer	01	X	AIU may upgrade to Unique Dtr
	Owner is identified Multiple sharers	11	Owner	
	Owner not identified Multiple Sharers	11	Find first Sharer	One of the sharer is promoted as owner
SnpInvDtr*	Null filter	00	X	Future use
	Unique Owner/Sharer	01	X	Issue DTR as per current spec of Snoop response map
	Owner is identified Multiple sharers	11	Owner	One owner and Multiple sharers are present in the system, issue DTW to DMI instead of DTR, In this case SnpRsp DC must be '0' and DT must be 2'b01
	Owner not identified Multiple Sharers	11	Find first Sharer	One of the sharers is promoted as owner, multiple sharers are present in the system, issue DTW to DMI instead of DTR , In this case SnpRsp DC must be '0' and DT must be 2'b01
SnpNitc/ SnpNitcMI / SnpNitcCI	Null filter	00	X	Future use
	Unique Owner/Sharer	01	X	
	Owner is identified Multiple sharers	11	Owner	
	Owner not identified Multiple Sharers	11	Find first Sharer	One of the sharer is promoted as owner
SnpInvDtw / SnpClnDtw / SnpInv / SnpInvStsh/ SnpUnqStsh / SnpStshUnq / SnpStshShd	Null filter	00	X	Future use
	Unique Owner/Sharer	01	X	AIU may do a DtwDataCln
	Owner is identified Multiple sharers	11	X	No DTR for this snoop so MPF3 is don't care
	Owner not identified Multiple Sharers	11	X	No sharer promotion in this case

*Currently DCE issues SnpInvDtw for Read Unique when there is more than one agent with the copy of CL. This will need to change so that DCE issues SnpInvDtr for all Read Unique cases with UP settings as specified in the Table 4-8 corresponding changes need to be made in AIUs. Motivation behind this change is to improve Read Unique performance when there is more than one copy of the CL in the system. The implementation of using SnpInvDtw may cause the DMI read when data is clean.

4.4.4.3 AIU Details

Table 4-9 gives details on different actions the AIUs need to take based on the UP and MPF3 field in different SnpReq.

TABLE 4-9: AIU UP ACTION

SnpReq	UP	MPF3	CHI AIU RetToSrc	NCAIU with Proxy Cache	ACE AIU	Action
SnpClnDtr / SnpNoSDInt / SnpVldDtr / SnpInvDtr / SnpNitcMI / SnpNitcCI/ SnpNitc	00	X	Not set	-	-	Future use
	01	X	Set	-	-	AIU may upgrade to Unique DtrReq based on the Snoop response mapping specification
	11	Match	Set	Provide data even if in SC state	Provide data if provided by the ACE agent	AIU must send out the appropriate DtrReq based on the snoop response mapping specification (for SnpInvDtr this may be a DTW based on Table 4-2)
	11	No Match	Not set	Do not provide data	Drop data if provided by the ACE agent	AIU must not send out the DtrReq
SnpInvDtw / SnpClnDtw / SnpInv / SnpUnqStsh / SnpStshUnq / SnpStshShd	00	X	Not set	-	-	Future use
	01	X	Not set	-	-	AIU may do a DtwDataCln based on the Snoop response mapping specification
	11	X	Not set	-	-	No DTR for this snoop
	11	X	Not set	-	-	No DTR for this snoop

4.5 Read data interleaving mode

Read data can be returned with data 'beat' interleaved. I.e., if the databus is less than a cacheline, which is always the case, two or more lines can be beat interleaved.

The proxy cache shall support the mode when AXI AIU with proxy cache does not interleave read data below the cache line boundary of 64 Bytes. It is required for compatibility with agents which does not support interleaving. The mode may incur some performance penalty, since potential bubbles between the cacheline transfer beats are not filled.

This mode is selectable by a configuration parameter NoCacheLineInterleave, please refer to the parameters specification for more detail on the parameter.

Read data can be returned with data interleaved for different AXIDs. This allows the agent databus to be fully utilized. However, not all agents can support return data interleaving for different AXIDs. Accordingly, a parameterized mode sets the IOAIU so read data is not interleaved for different AXIDs. This means that when a read request for a given AXID is sent, all the data for that AXID request must be

transferred completely before data for another AXID can be returned to the IOIAU agent.
The mode will be implemented by returning read data in agent request order.
This mode will likely incur a performance penalty, since bubbles will likely occur on the return data bus.

4.6 System cache

The system memory cache (SMC) allows DMI(s) to buffer cache lines and service requests without accessing 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 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). 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. The SMC can be configured as scratchpad. The SMC supports way partitioning.

Allocation policy

SMC in general uses the native interface (CHI, ACE, ACE-Lite, ACE-Lite- E & AXI) allocation hint to allocate to the cache. These hints can be overwritten by a configuration setting where following type of request may individually be disabled from allocation:

- Write request with clean data
- Write request with dirty data
- All read request data
- All write request data
- Disable cache allocation for WriteUnique from CHI/ACE processors

Allocating a new cache line may cause a valid cache line to be replaced based on a configurable replacement policy. The proxy 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). 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.

Also, the SMC allocates read depending on the memory attribute signals and originator (NCAIU with Proxy Cache yes or not).

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 partition can be configured via configuration and status registers. Portion of the cache reserved for single or multiple Agent identifiers (Agent = AIU)

- Feature Build Time Enabled
  - Number of Agent IDs (register pairs)
- Boot time configured on a per way basis
  - Register Pair
  - Agent ID
  - Reserved ways for this Agent ID
- Partitioned Ways are reserved to Agent IDs for allocation

Scratchpad mode

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 part of the cache as scratchpad on a per way basis.

Only contiguous ways starting from Way `0` can be configured as scratchpad.

The configuration is performed at boot time, and configured on a per way basis

4.7 Transaction flows

This section goes over different transaction flows within the Ncore 3 system. The flow diagram consists of two AIUs, which can be any possible AIU type, one DCE and one DMI (which can be treated as DII in

the case of non-coherent transactions). Responses are shown in orange and data is shown in red. Dashed purple arrows show dependencies at the unit for initiation of the message. The arrival time show in just an example and is dependent on network delay which is not captured here. Note that time flows top to bottom.

4.7.1 Non-Coherent transactions

Non-coherent transaction flow does not involve DCE. AIUs directly communicate with DMI / DII depending on the identified target. Non-Coherent read transaction flow is shown in Figure 4-1 and the write transaction flow is shown in Figure 4-2. Though the diagrams only show DMI the same flow applies to DII. StrRsp and DtrRsp are just used for message ID release and have no other significance in DMI / DII.

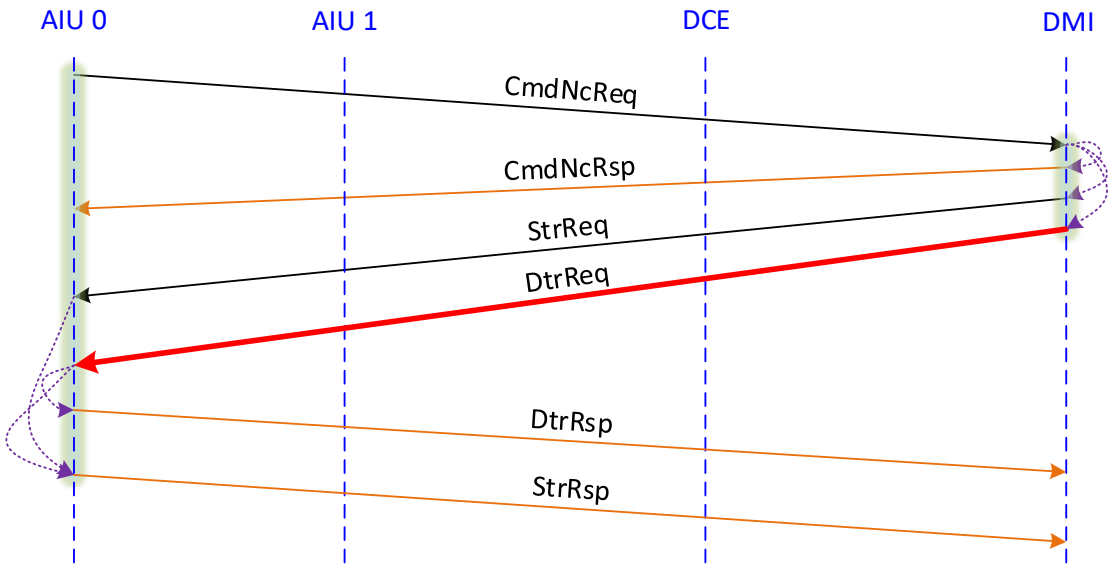


FIGURE 4-1: NC READ TRANSACTION FLOW

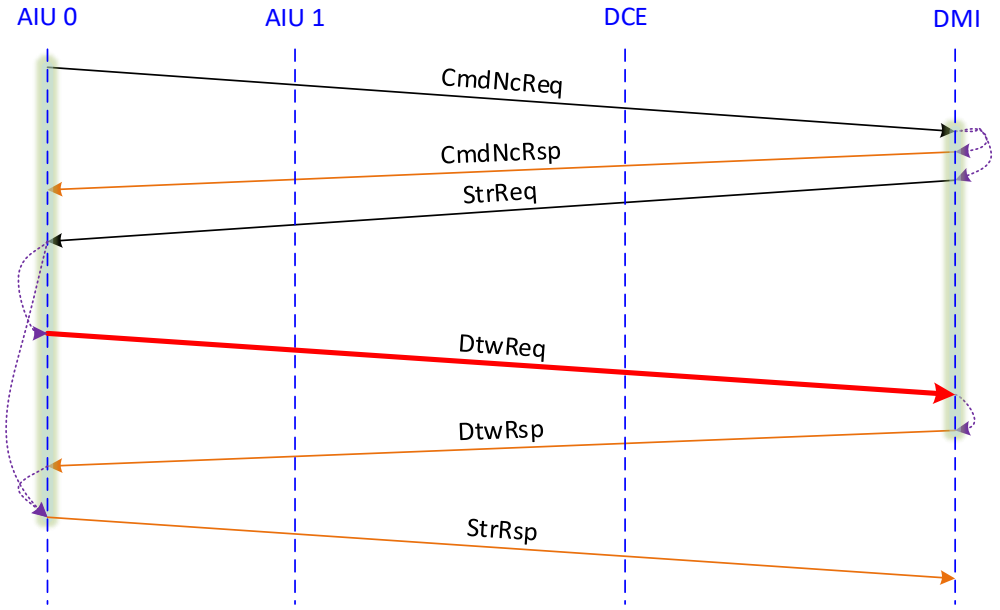


FIGURE 4-2 NC WRITE TRANSACTION FLOW

Non-coherent atomic transaction flow is shown in Figure 4-3. The flow is valid for CmdWrAtm, CmdRdAtm, CmdSwapAtm & CmdCompAtm. Note that the Dtr flow shown will not be present in the case of Store atomics (CmdWrAtm).

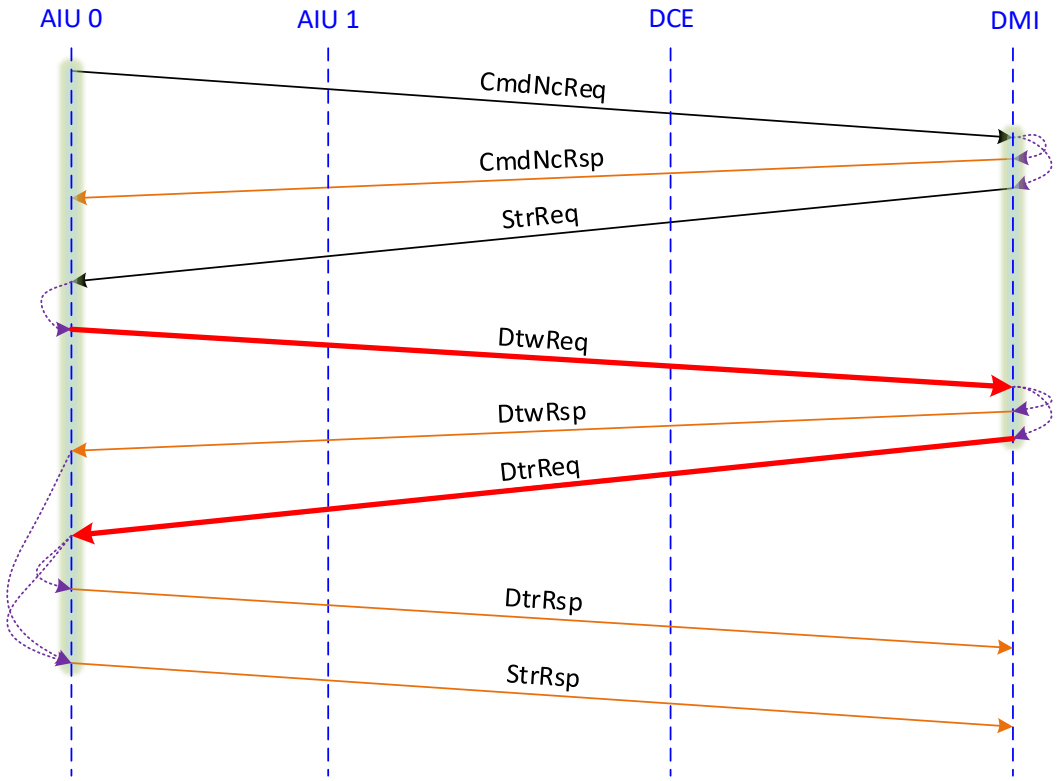


FIGURE 4-3 NON-COHERENT ATOMIC TRANSACTION FLOW

4.7.2 Coherent Read and CMO transactions

In the following cases AIU0 is treated as the requesting AIU and AIU1 as the snooping AIU.

Read transaction flow with snoop data is shown in Figure 4-4. The flow is valid for CmdRdVld, CmdRdUnq, CmdRDNitc and CmdRdNitcMI. Note that in the later two cases the Dtr must be a clean Dtr while in the former two cases it can be either clean or dirty.

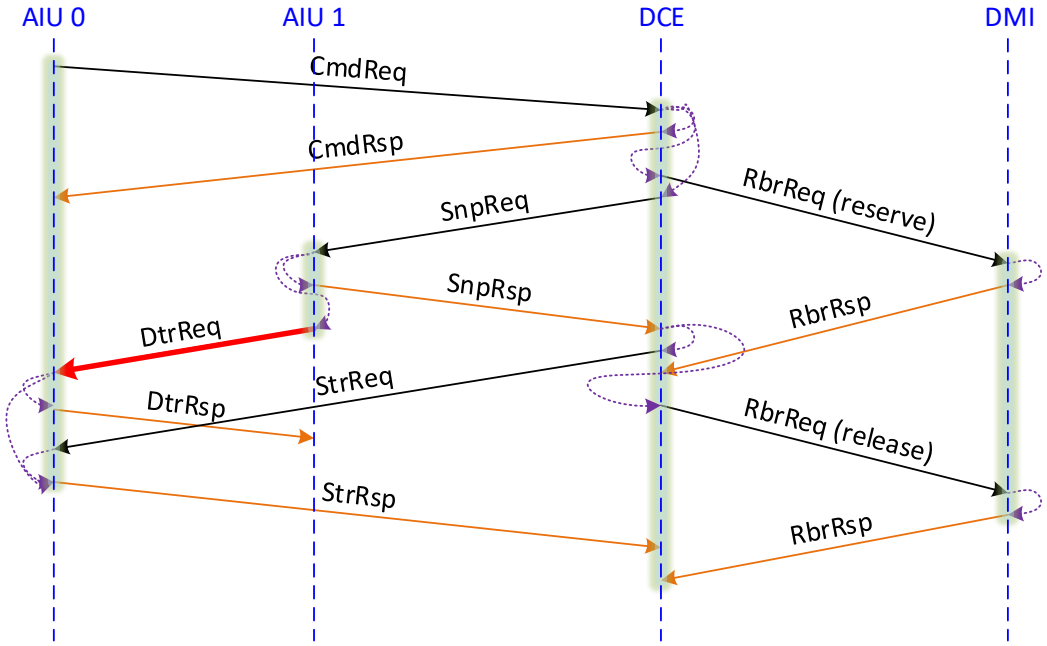


FIGURE 4-4 READ WITH SNOOP DATA

Read transaction flow where snooping AIU does not provide the data and the data is read form DMI is shown in Figure 4-5. The flow is valid for CmdRdVld, CmdRdUnq, CmdRdCln, CmdRdNitc, CmdRdNitcCI and CmdRdNitcMI. Note that in the first two cases the Dtr can either be clean or dirty in rest of the cases it must be clean.

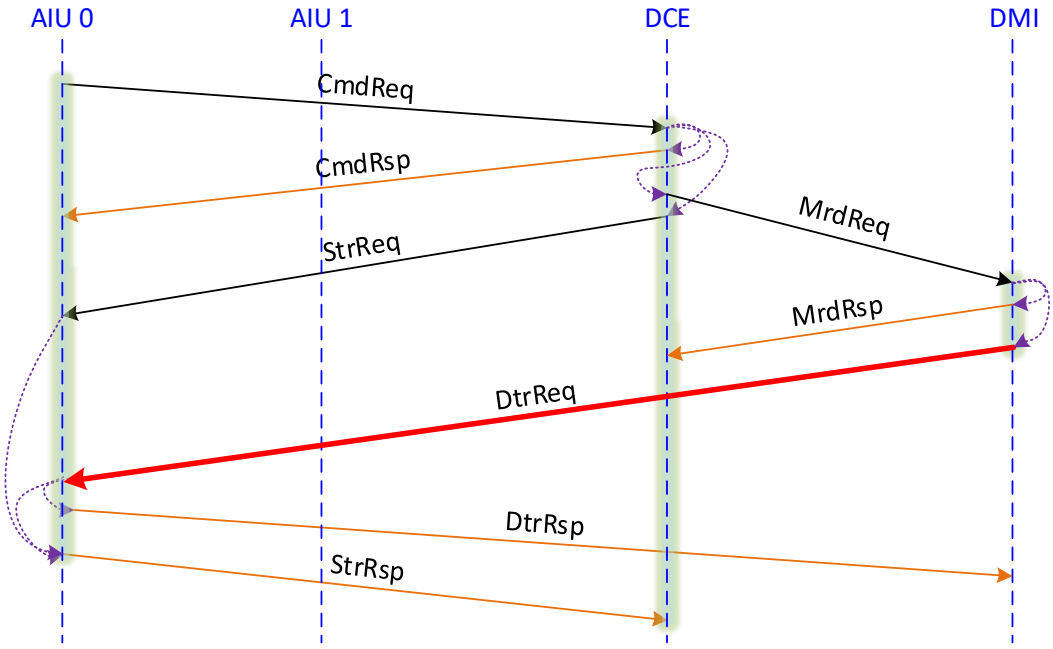


FIGURE 4-5 READ WITH DMI DATA

Read clean transactions with snoop data flow is shown in Figure 4-6. The flow is valid for CmdRdCln and CmdRdNitcCI. The Dtw shown is issued to write dirty data back to memory as the requesting AIU cannot accept dirty data. The Dtw is not required in cases where ownership is retained by the processor providing the data or the data is already clean. Note that the RbuReq and RbuRsp are independent credit update messages and are not tied to the deallocation of the associated request in DCE.

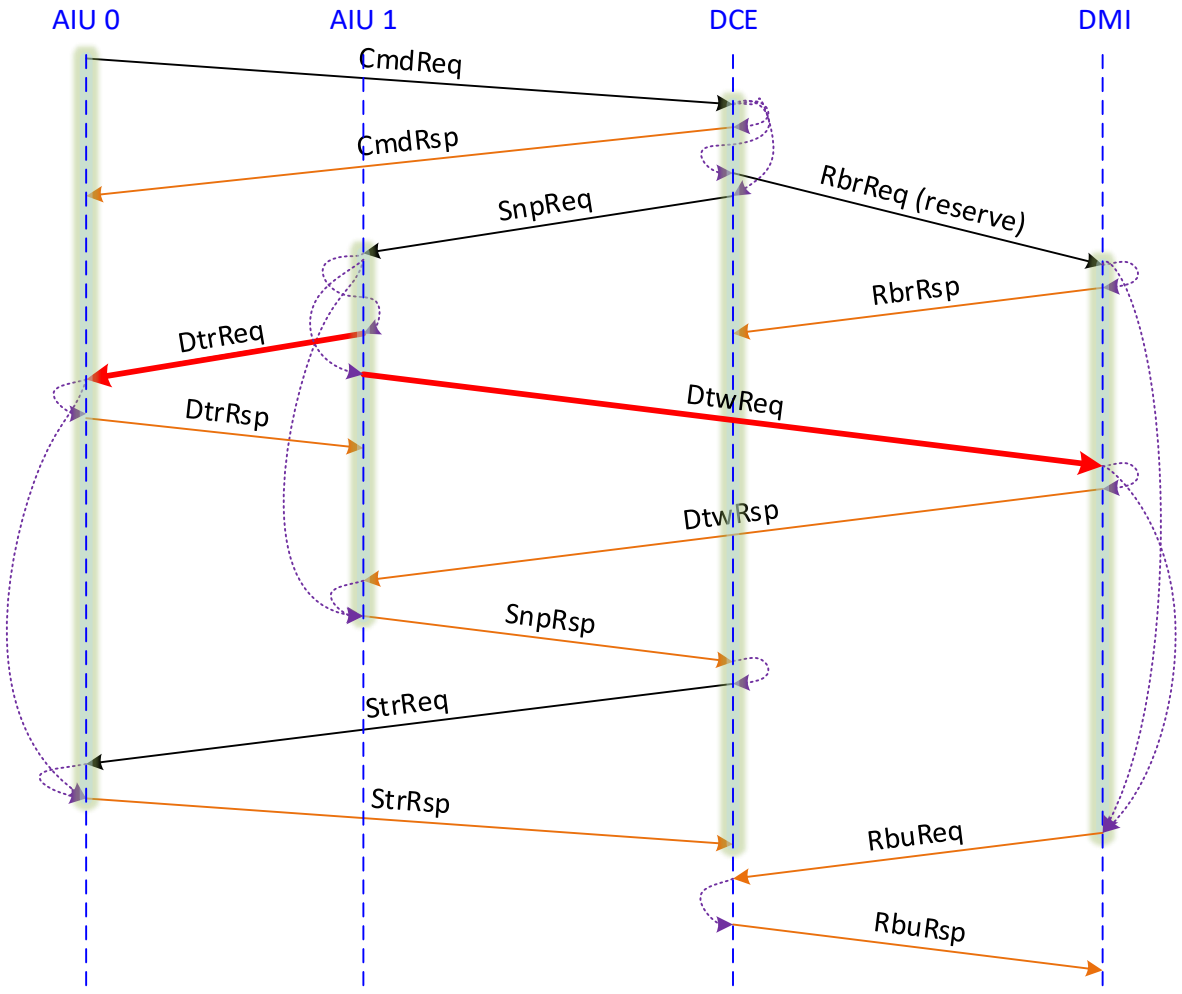


FIGURE 4-6 READ CLEAN WITH SNOOP DATA

Read with no snoop transaction flow is shown in Figure 4-7. The flow is valid for all read commands CmdRdVld, CmdRdUnq, CmdRdCln, CmdRdNitc, CmdRdNitcCI and CmdRdNitcMI. Note that in the first two cases the Dtr can either be clean or dirty in rest of the cases it must be clean.

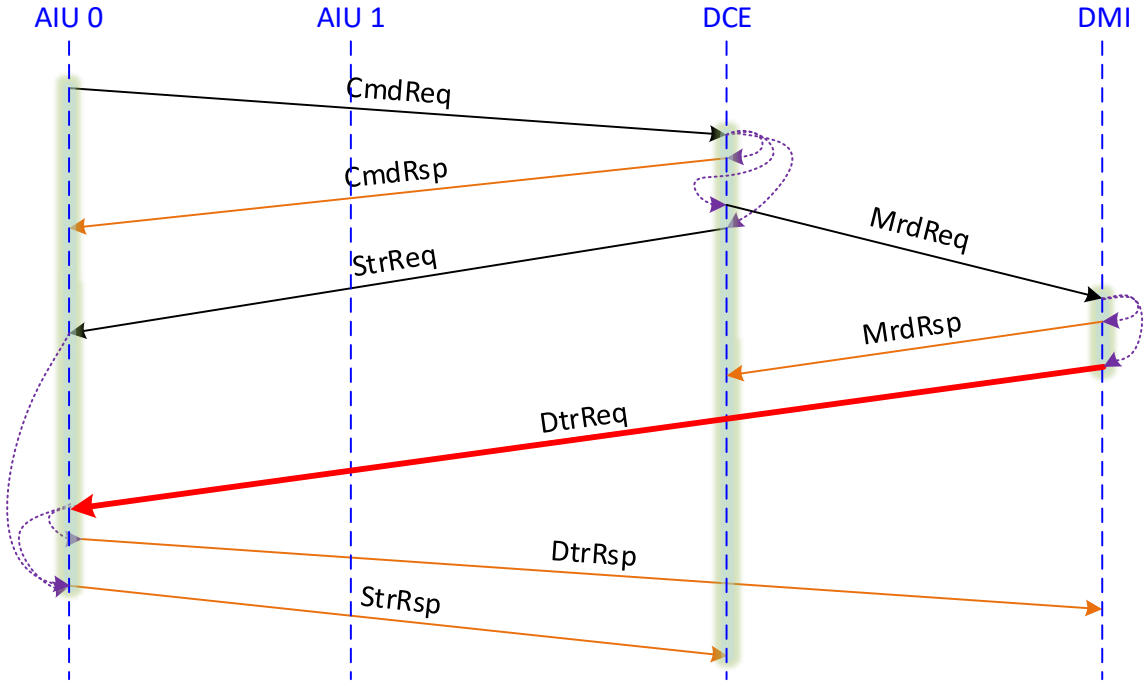


FIGURE 4-7 READ WITH NO SNOOPS

Clean operations (data less) with dirty snoop data transaction flow is shown inFigure 4-8. The flow is valid for CmdClnUnq, CmdWrAtm, CmdRdAtm, CmdSwapAtm, CmdCompAtm, CmdMkUnq, CmdClnInv, CmdMkInv, CmdClnVld & CmdClnShPsist. Note that Mrd is issued by DCE only in the last four cases to enforce CMO at the SMC after receiving snoop responses. Furthermore, in current Ncore Arch specification coherent atomic transactions are treated as clean transactions by DCE. Please refer to 4.7.4 for coherent atomic transaction processing.

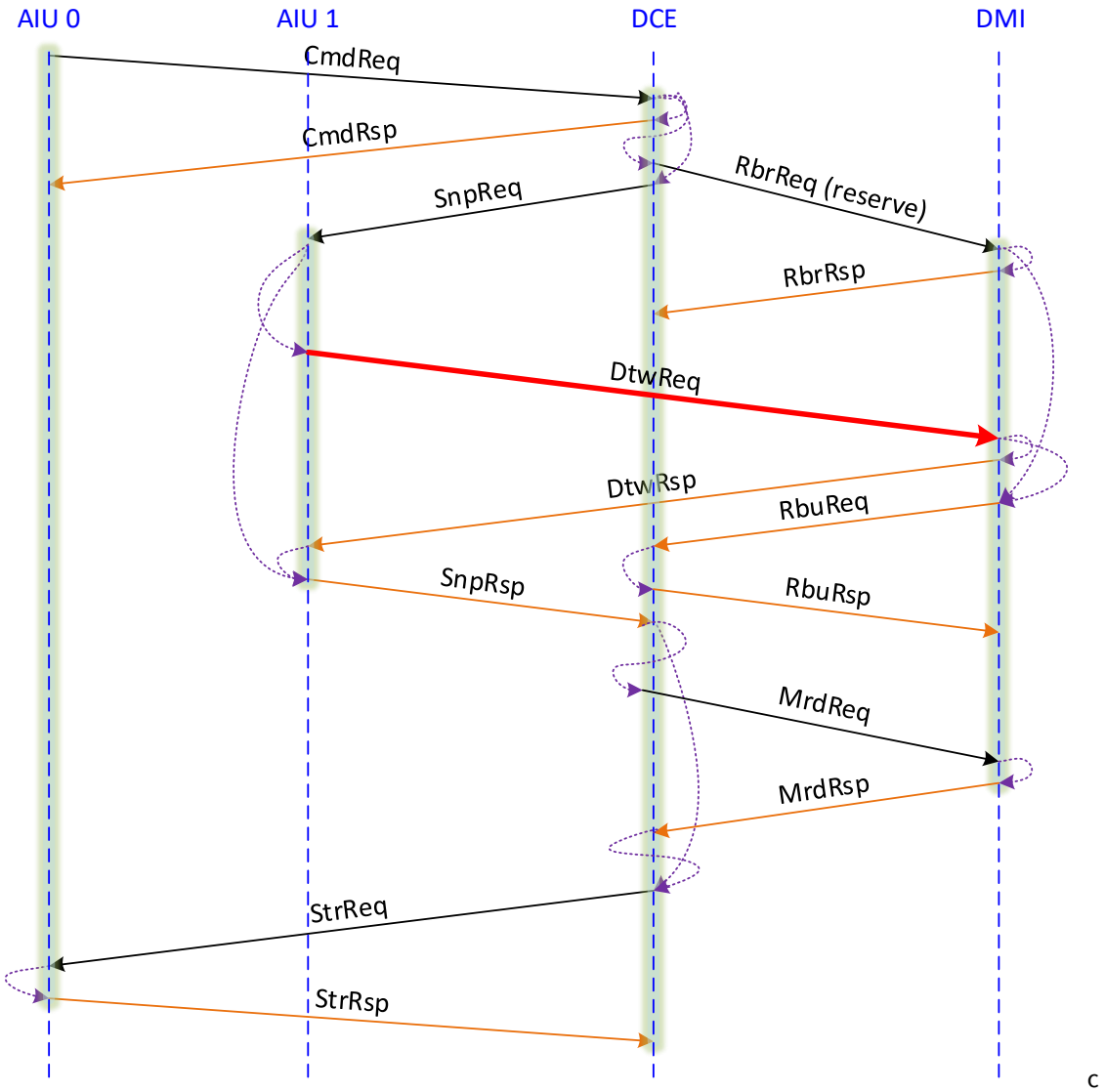


FIGURE 4-8 CLEAN WITH DIRTY SNOOP DATA

Invalidate transaction flow is shown in Figure 4-9. The flow is valid for CmdClnInv (agents with clean data) & and CmdMkInv.

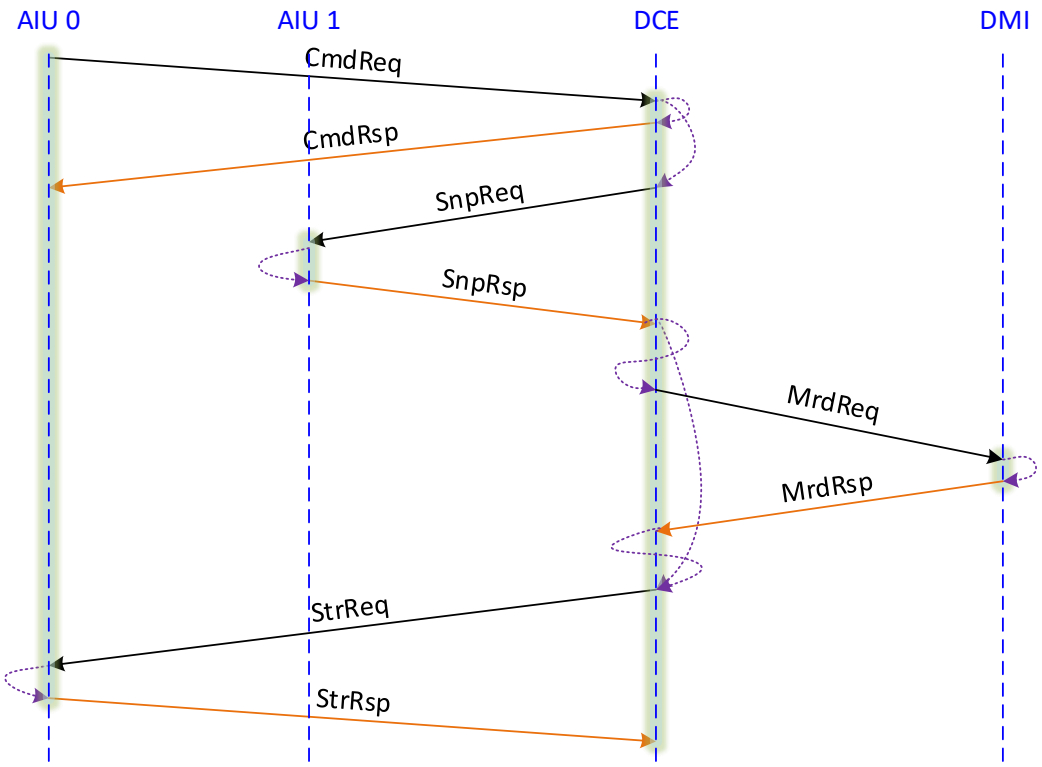


FIGURE 4-9 INVALIDATE TRANSACTION FLOW

Read when snooped processor is in partial state transaction flow is shown in Figure 4-10. This flow is valid for all read commands when the owner is in unique partial state. Note that DtwMrgMrdReq processing in DMI differs in the case of stash commands where the DtwRsp is delayed till after DtrRsp is received by DMI.

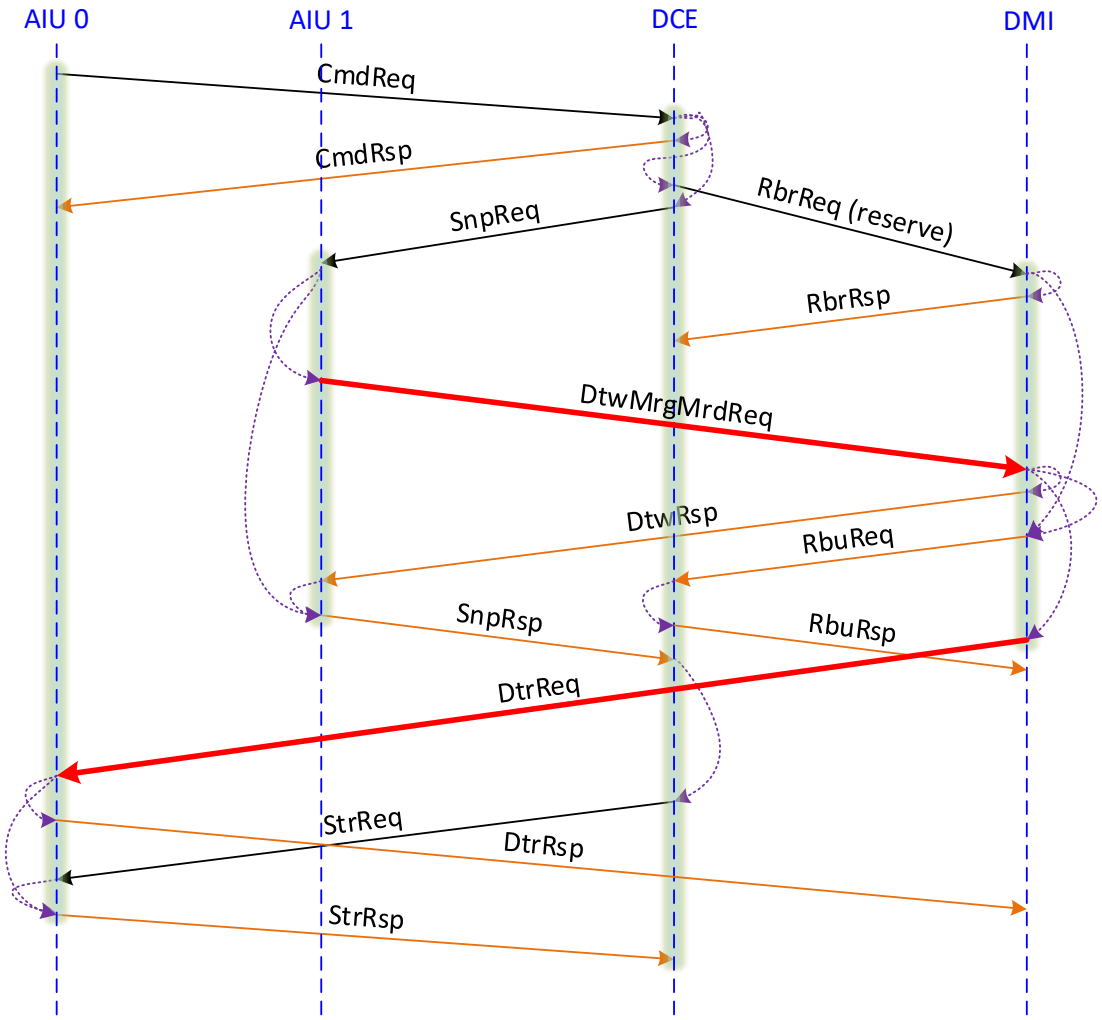


FIGURE 4-10 READ WHEN OWNER IN UNIQUE PARTIAL STATE

4.7.3 Coherent write transactions

Write line unique transaction flow is shown in Figure 4-11. This flow is valid for CmdWrUnqPtl and CmdWrUnqFull, note that DtwReq secondary is optional in the case of CmdWrUnqPtl and will not exist in the case of CmdWrUnqFull. Transaction flow for CHI AIU CmdWrBkPtl, CmdWrBkFull, CmdWrClnPtl, CmdWrClnFull, CmdWrEvict and CmdEvict is shown in Figure 4-12. Note that CmdEvict will not have an associated Dtw. In the case of ACE AIU WriteBack/WriteClean and AXI AIU with proxy cache evictions are treated as non-coherent writes to DMI. WriteClean will retain a clean copy of the line in the initiator's cache, no Update command will be sent to DCE. WriteBack will invalidate the line in the initiator's cache and must be followed by an Update command to DCE, this is shown in Figure 4-13.

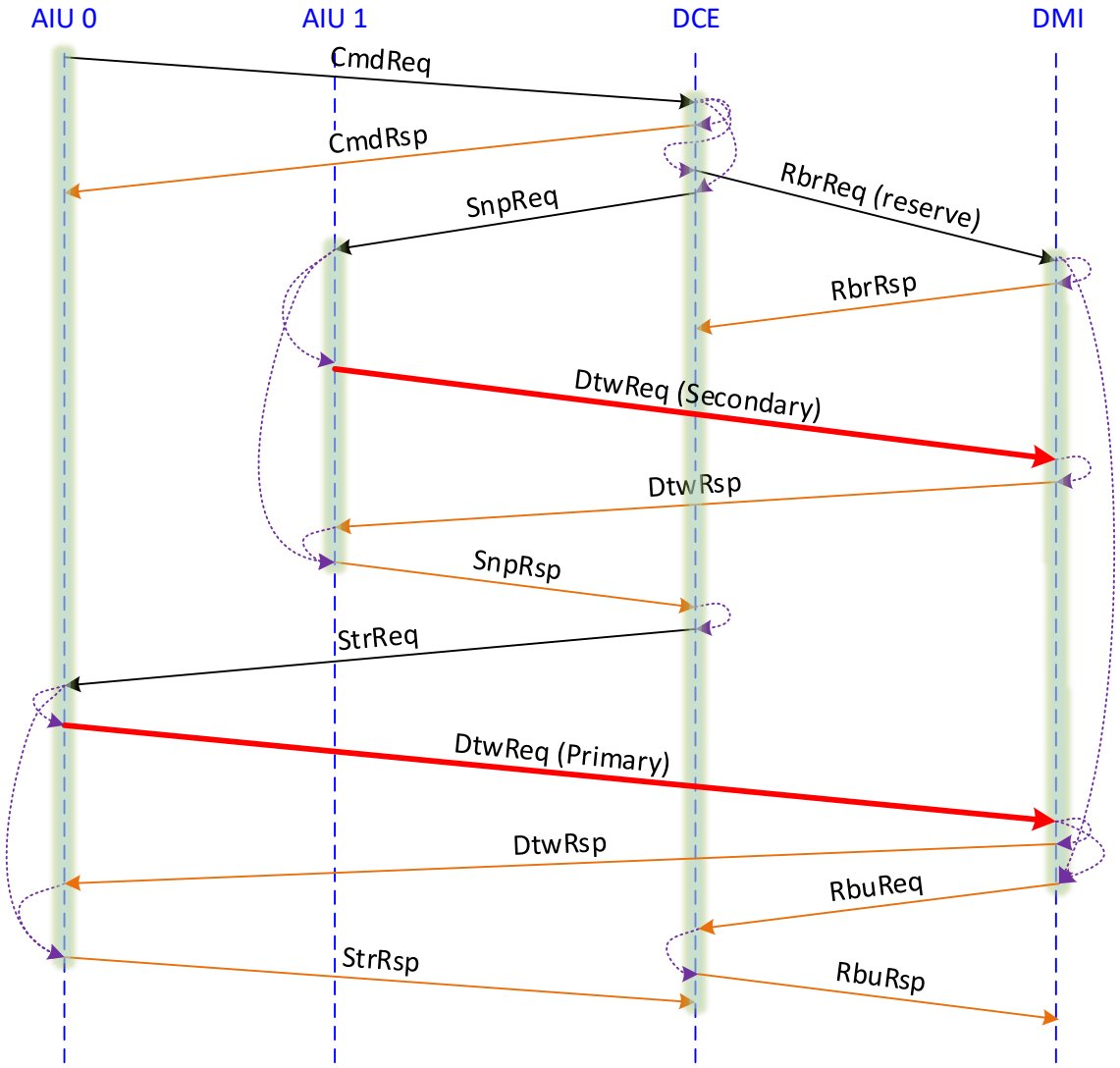


FIGURE 4-11 WRITE UNIQUE TRANSACTION FLOW

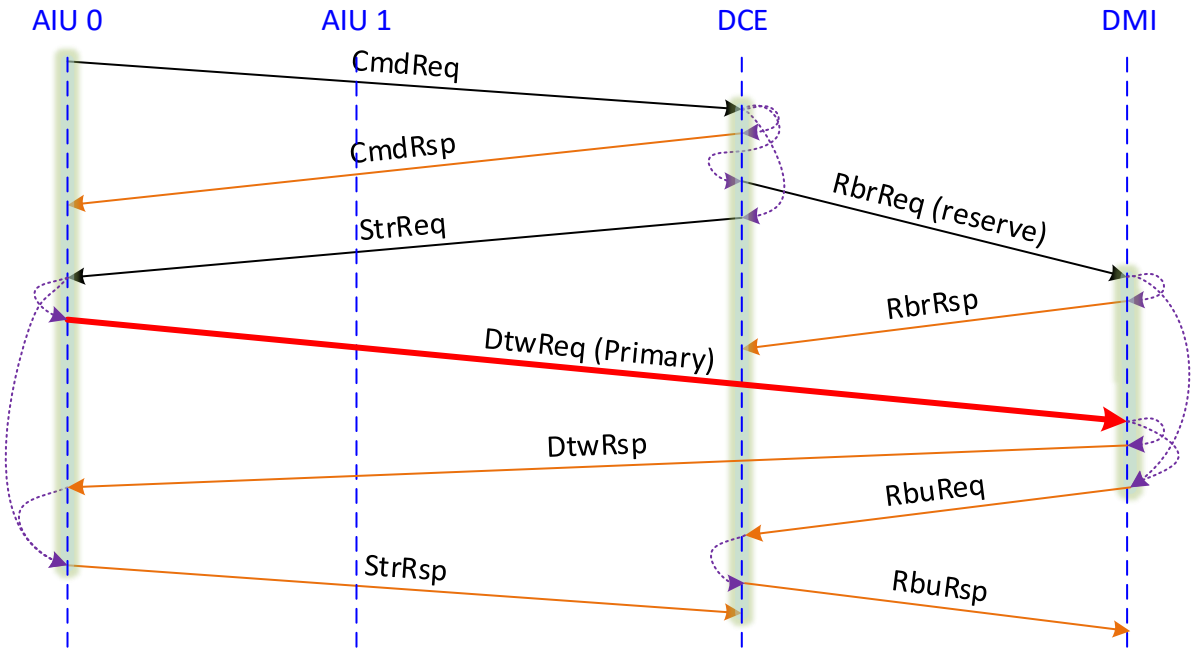


FIGURE 4-12 CHI COHERENT WRITE TRANSACTION FLOW

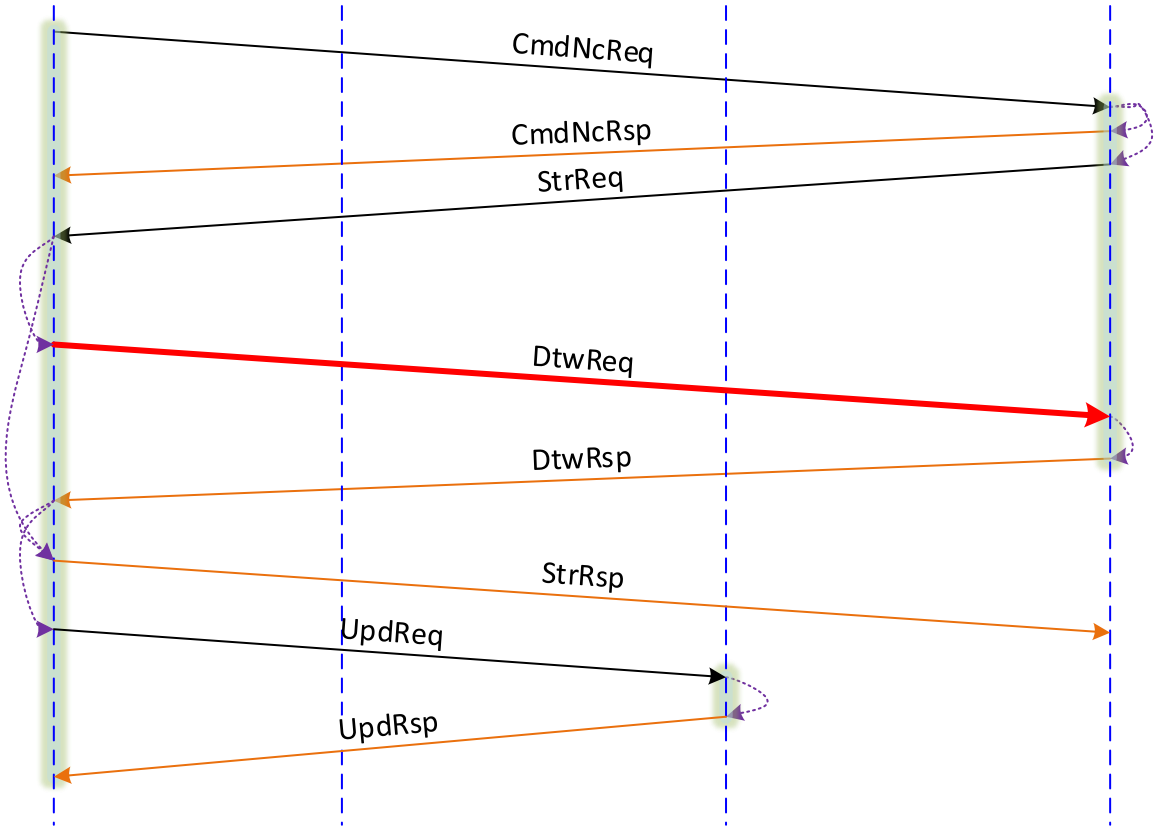


FIGURE 4-13 ACE WRITE BACK TRANSACTION FLOW

4.7.4 Coherent atomic transactions

Coherent atomic transactions are broken into two parts, coherent clean part which is done by DCE and non-coherent atomic part which is done by DMI. The flow for this transaction and its dependencies are shown in Figure 4-14. The flow is valid for CmdWrAtm, CmdRdAtm, CmdSwapAtm & CmdCompAtm. Note that the Dtr flow shown will not be present in the case of Store atomics (CmdWrAtm).

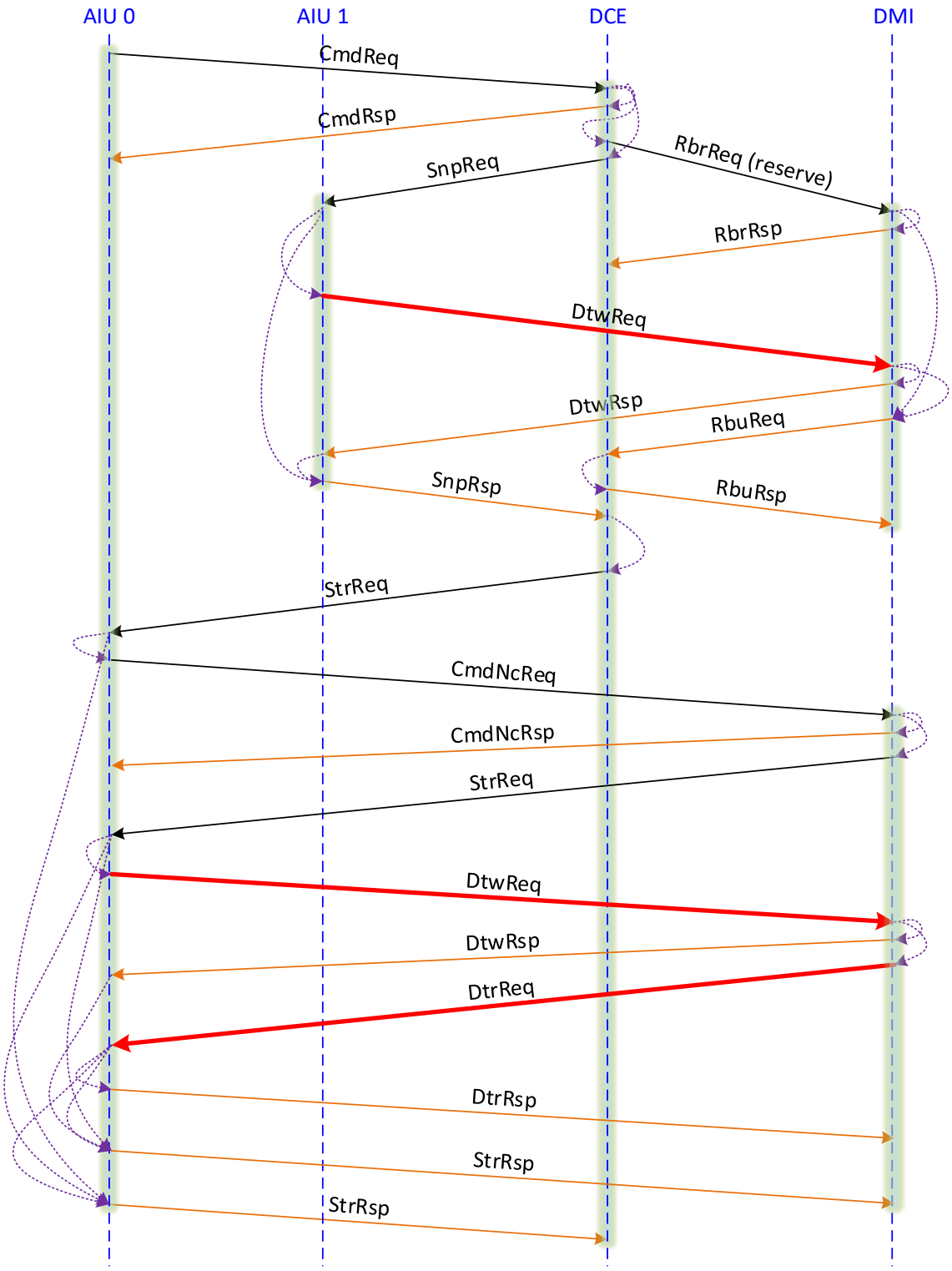


FIGURE 4-14 COHERENT ATOMIC TRANSACTION

4.7.5 Stash transactions

In the following cases AIU1 is treated as the requesting AIU; AIU2 as the snooping AIU and AIU 0 as the target AIU.

Write stash full transaction flow when target is identified and accepts the stash is shown in Figure 4-15. The flow when the target does not accept the stash is shown in Figure 4-16. Note that this flow is same for the case where target is not identified, with the exception that the target Snoop to AIU0 will not be present, furthermore it also applies to write stash partial when target is not identified. Write stash partial when target is identified and accepts the stash is shown in Figure 4-17.

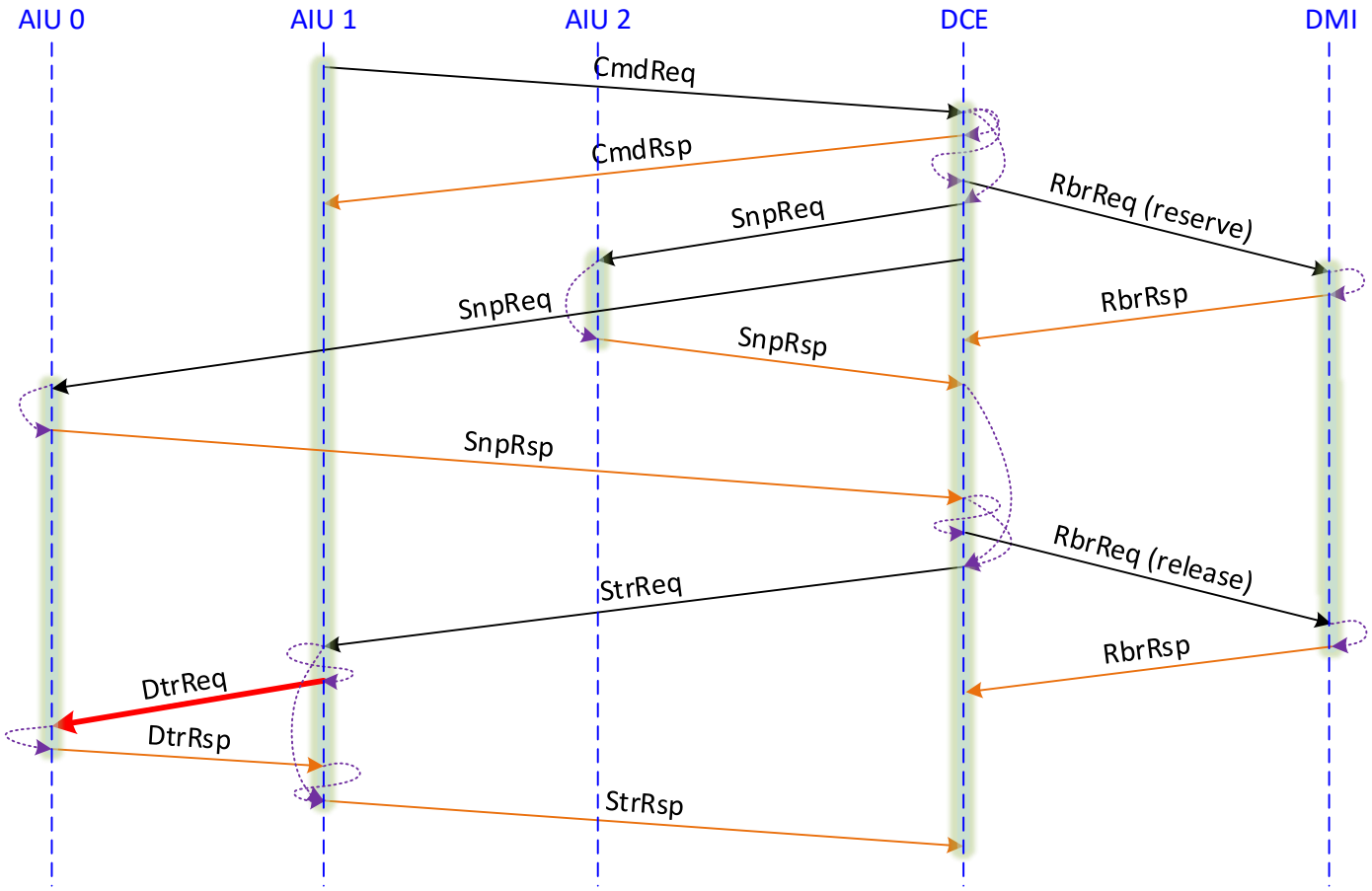


FIGURE 4-15 WRITE STASH FULL ACCEPT TRANSACTION FLOW

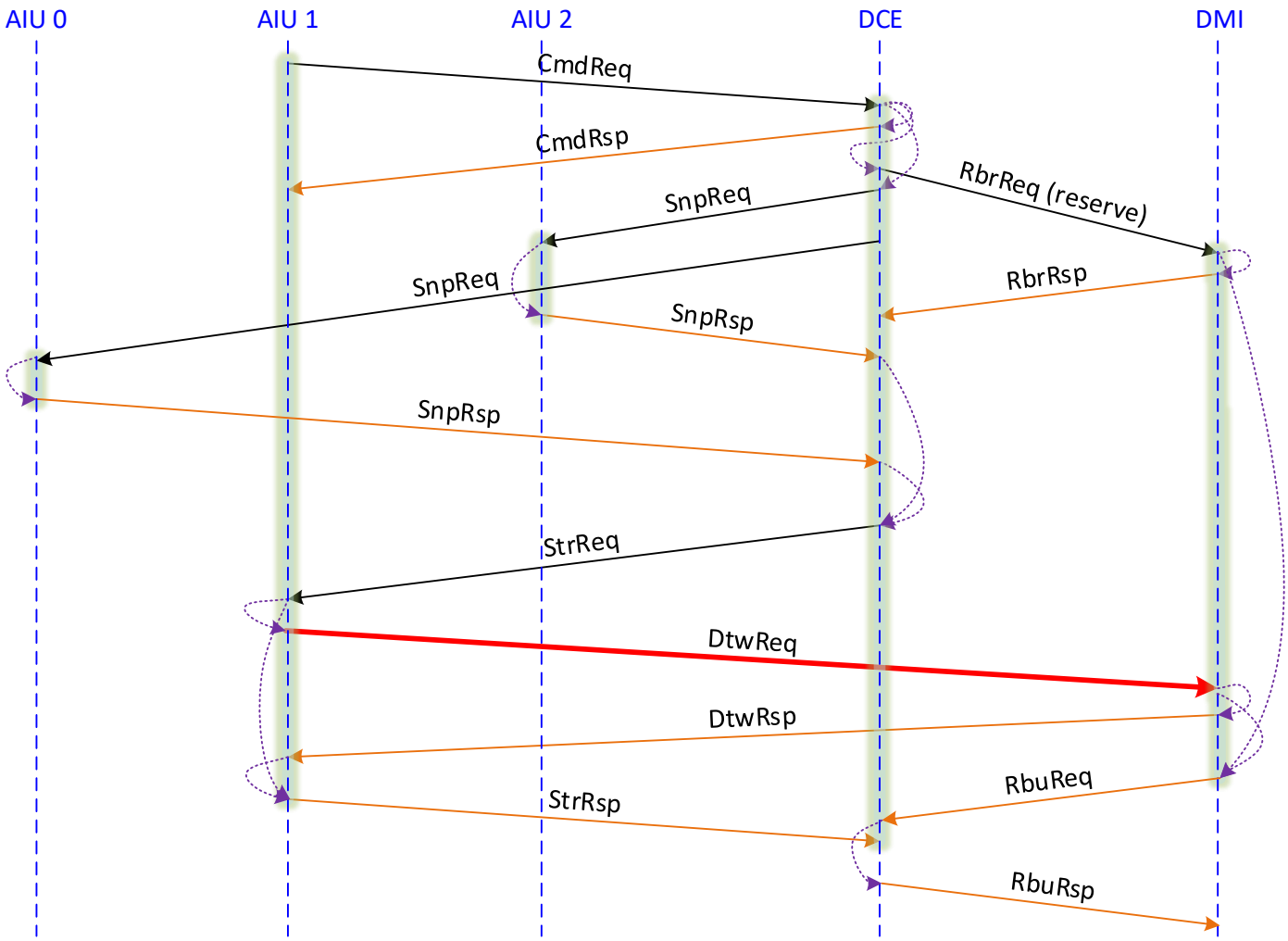


FIGURE 4-16 WRITE STASH DECLINE TRANSACTION FLOW

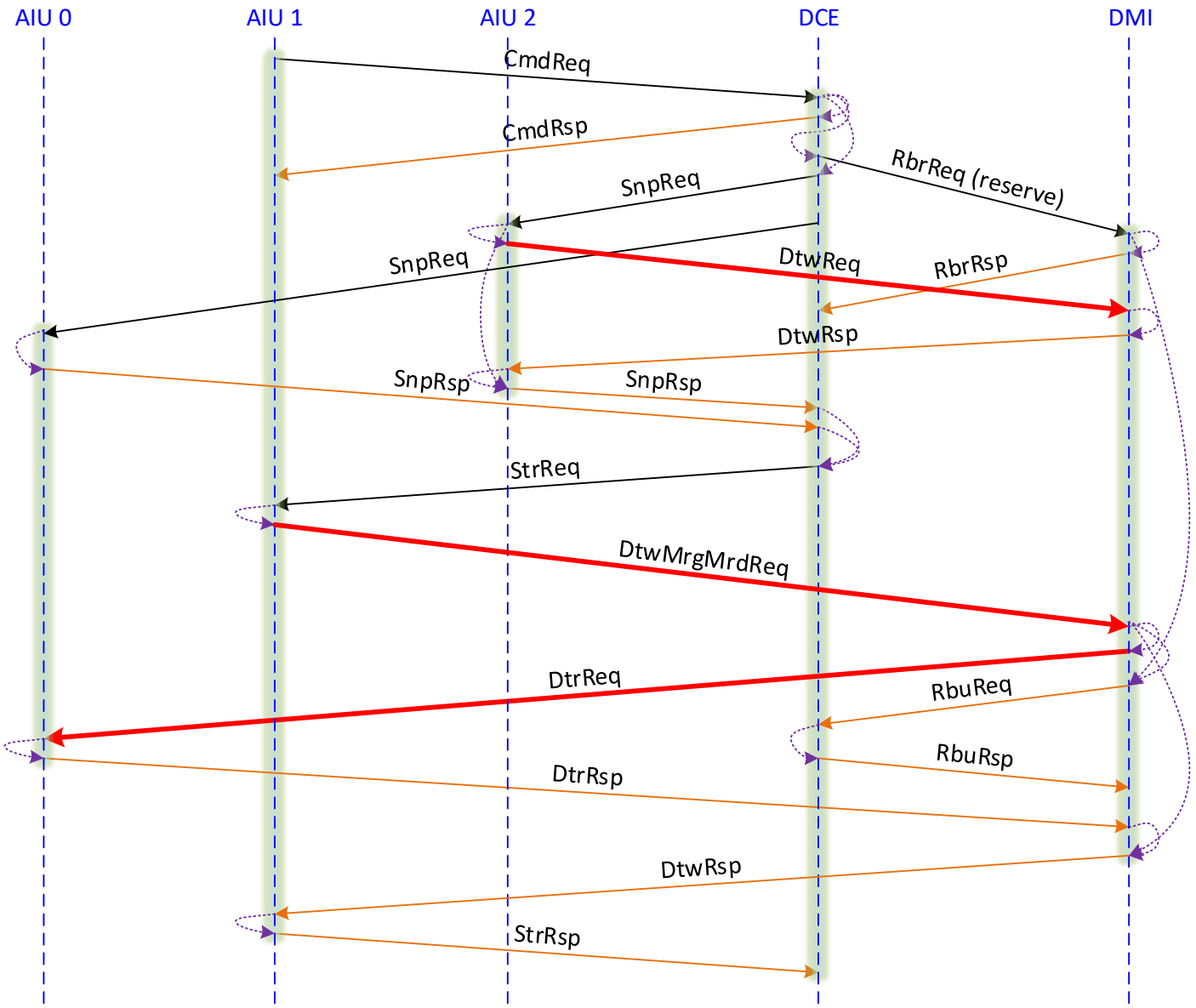


FIGURE 4-17 WRITE STASH PARTIAL ACCEPT FLOW

Read Stash decline flow is shown in Figure 4-18. Note this is different than target not identified case that no snoop is sent to target and no MRDPref is going to be sent out to DMI for the declined request. Read stash target not identified flow is shown in Figure 4-19. Read Stash accept flow is shown in Figure 4-20, when Read stash is accepted, if there's a peer with dirty copy of the data, the data is first written to DMI,

and a MRDReq will be sent out by DCE after the write completes. (the read after write sequence is guaranteed by DTWReq -> DTWRsp -> SNPRsp -> MRDReq). If no peer is providing the data, data will be directly read from DMI.

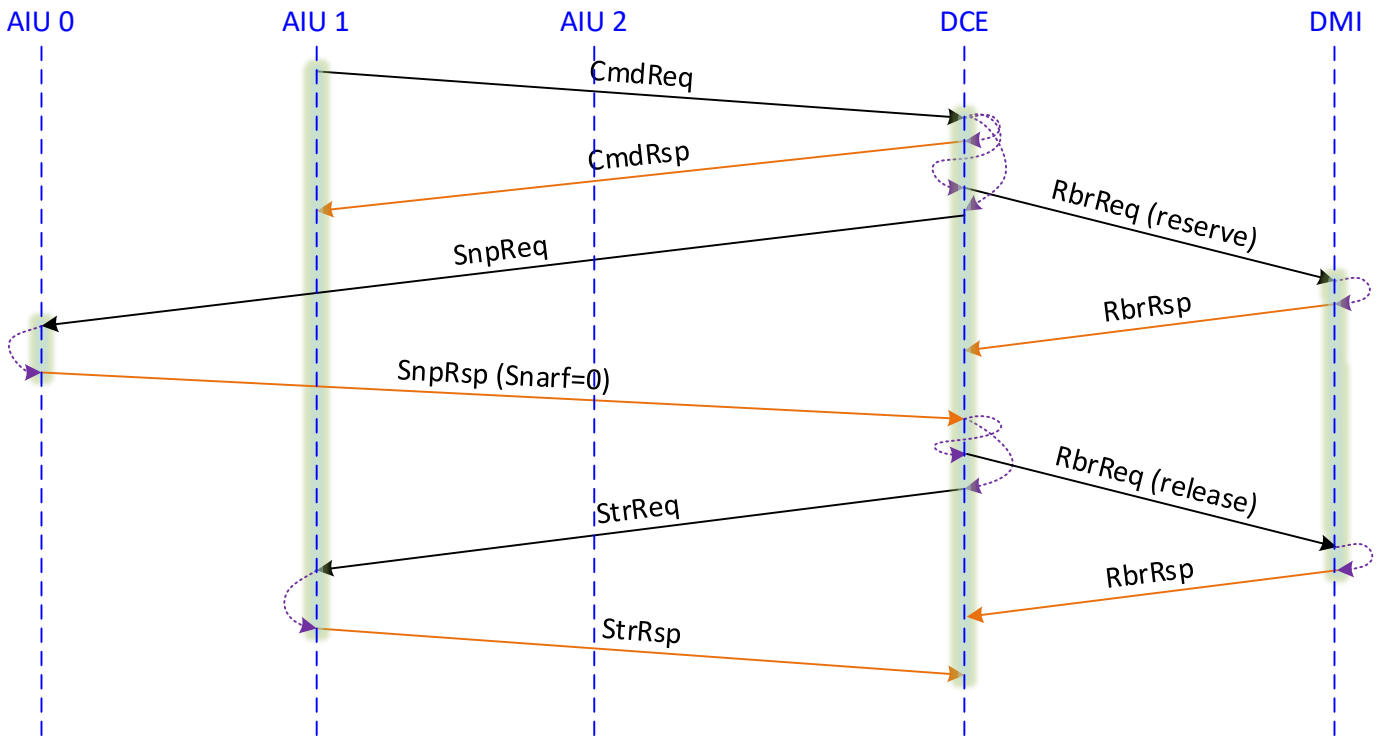


FIGURE 4-18 READ STASH DECLINE FLOW

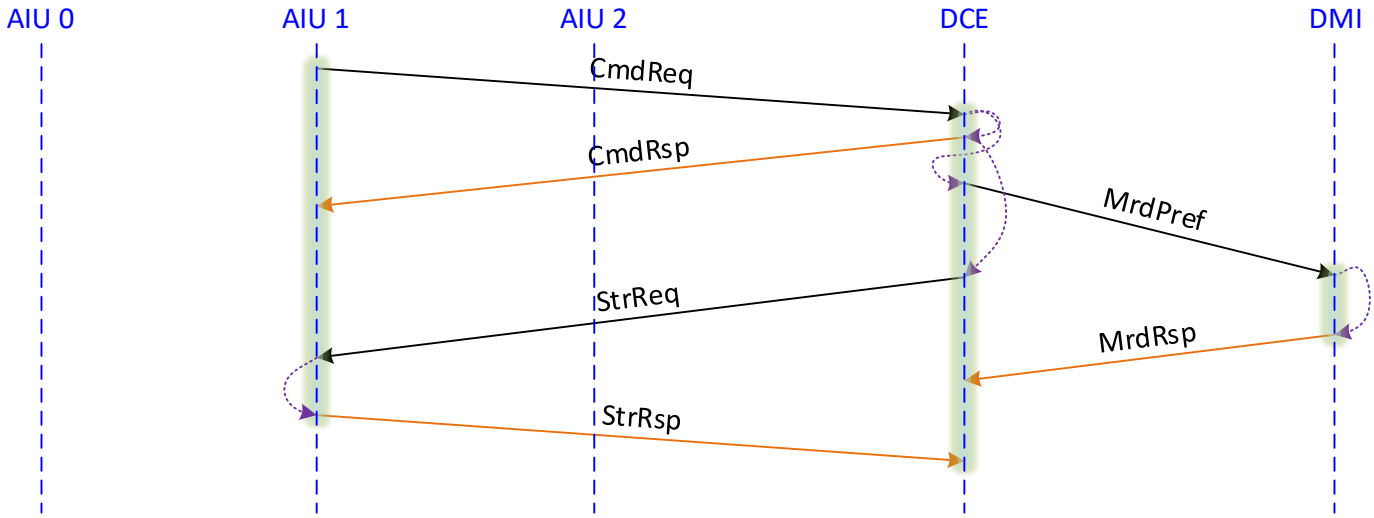


FIGURE 4-19 READ STASH TARGET NOT IDENTIFIED FLOW

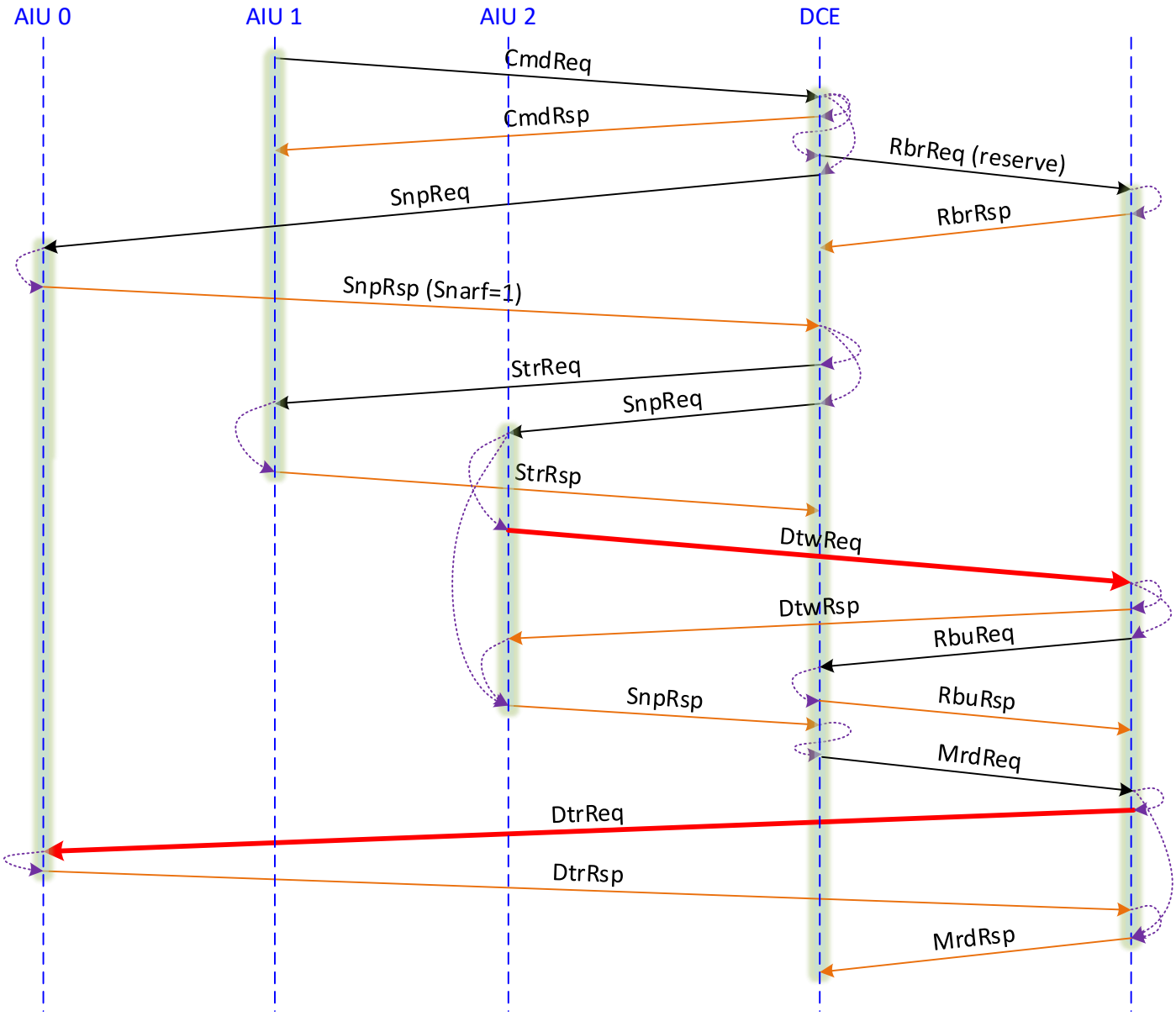


FIGURE 4-20 READ STASH ACCEPT FLOW

4.7.6 Prefetch transactions

Prefetch transaction flow does not involve DCE. AIUs directly communicate with DMI depending on the identified target. Prefetch transaction flow is shown in Figure 4-21. The Prefetch command fills the SMC if the line is not present alternatively if the line is already present in the SMC, resources are not available or if there is a collision then it just gets acknowledged. Note that prefetch command may collide with Mrd of the same address as it does not serialized by DCE.

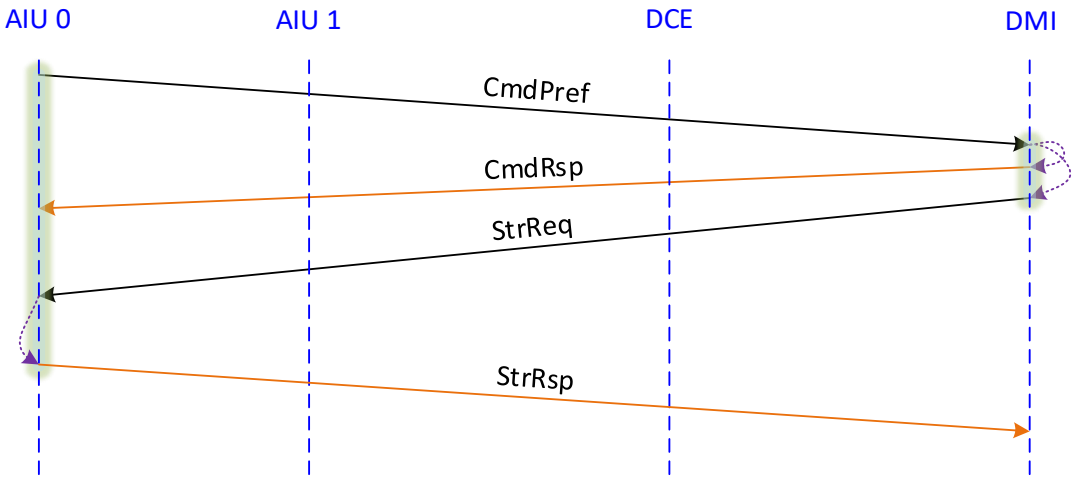


FIGURE 4-21 PREFETCH TRANSACTION FLOW

4.8 Ncore units native transactions mapping

										Can be seen in:				
Native Name	**Concerto Name**	**AR/AW**	**AxBar**	**AxAtop**	**AxCache**	**AxDomain**	**AxSnoop**	**Ptl Access**	**Room in Cache**	**AXI4**	**AXI - Cache**	**ACE-Lite**	**ACE**	**CHI**
ReadNoSnoop	ReadNonCoherent	R	0	X	X	0,3	0x0	X	X					
ReadOnce	ReadNITC	R	0	X	X	1,2	0x0	X	(0x0 | Not Proxy Cache)					
ReadShared	ReadValid	R	0	X	X	1,2	0x0,0x1	X	X					
ReadClean	ReadClean	R	0	X	X	1,2	0x2	X	X					
ReadNotSharedDirty	ReadNotSharedDirty	R	0	X	X	1,2	0x3	X	X					
ReadUnique	ReadUnique	R	0	X	X	1,2	0x7	X	X					
CleanShared	CleanValid	R	0	X	X	0,1,2	0x8	X	X					
CleanSharedPersist	CleanSharedPersist	R	0	X	X	0,1,2	0xa	X	X					
CleanUnique	CleanUnique	R	0	X	X	1,2	0xb	X	X					
MakeUnique	MakeUnique	R	0	X	X	1,2	0xc	X	X					
CleanInvalid	CleanInvalid	R	0	X	X	0,1,2	0x9	X	X					
MakeInvalid	MakeInvalid	R	0	X	X	0,1,2	0xd	X	X					
ReadOnceCleanInvalid	ReadNITCCleanInv	R	0	X	X	1,2	0x4	X	X					
ReadOnceMakeInvalid	ReadNITCMakeInv	R	0	X	X	1,2	0x5	X	X					

DVM Message	DVMMessage	R	0	X	X	1,2	0xf	X	X					
ReadOnce	ReadNITC					**Default Read**								
WriteUniquePtlStash	WriteStashPtl	W	0	X	[1]	1,2	0x8	X	X					
WriteUniqueFullStash	WriteStashFull	W	0	X	[1]	1,2	0x9	X	X					
StashOnceShared	LeadCCHShared	W	0	X	[1]	0,1,2	0xc	X	X					
StashOnceUnique	LoadCCHUnique	W	0	X	[1]	0,1,2	0xd	X	X					
Atomic Transactions	WriteAtomic	W	0	[5:4] = 1	X	X	X	X	X					
Atomic Transactions	ReadAtomic	W	0	[5:4] = 2	X	X	X	X	X					
Atomic Transactions	SwapAtomic	W	0	0x30	X	X	X	X	X					
Atomic Transactions	CompareAtomic	W	0	0x31	X	X	X	X	X					
WriteNoSnoop	WriteNonCohFull	W	0	X	X	0,3	0x0	0x0	X					
WriteNoSnoop	WriteNonCohPtl	W	0	X	X	0,3	0x0	0x1	X					
ReadUnique	ReadUnique	W	0	X	X	1,2	0x0	0x1	0x1					
WriteUniquePtl	WriteUniquePtl	W	0	X	X	1,2	0x0	0x1	0x0					
WriteUniqueFull	MakeUnique	W	0	X	X	1,2	(0x1 | AXI)	X	0x1					
WriteUniqueFull	WriteUniqueFull	W	0	X	X	1,2	0x1	X	X					
WriteClean	WriteNonCohPtl	W	0	X	X	0,1,2	0x2	0x1	X					
WriteClean	WriteNonCohFull	W	0	X	X	0,1,2	0x2	0x0	X					
WriteBack	WriteNonCohPtl	W	0	X	X	0,1,2	0x3	0x1	X					
WriteBack	WriteNonCohFull	W	0	X	X	0,1,2	0x3	0x0	X					
Evict	CmdEvict	W	0	X	X	1,2	0x4	X	X					
WriteEvict	WriteNonCohFull	W	0	X	X	0,1,2	0x5	X	X					
WriteUniqueFull	WriteUniqueFull					**Default Write**		0x0						
WriteUniquePtl	WriteUniquePtl							0x1						

4.9 End to end credit

Ncore 3 units uses an end-to-end credited protocol scheme.

Each credit represents a free entry at the receiver and a sender needs to have at least one credit available before sending a message.

The receiver in each target will be configured by Maestro to implement sufficient buffering (skid buffer) to store messages arriving from all communicating initiators.

The communication requirements are determined based on:

- Address map support required by the SoC use case
- Interleave factor - directories and memories can be interleaved to improve performance

A more detailed discussion including the definition and manipulation of the interconnect matrix can be found in section 3.4.4 Connectivity mapping.

The number of buffers is calculated by Maestro, based on the number of initiators, and for each initiator the number of allowed flows is defined as a design configurable parameter. A single parameter for an initiator defines the number of supported flows between that initiator and **all** targets. Therefore:

- All skid buffers associated with an initiator have the same depth

The max. bandwidth available to an agent depends on the

- Number of transactions it can have in flight
- The roundtrip latency for sending the request and receiving a response back
- The amount of data moved per transaction request

The Ncore 3 architecture only supports a single, design time defined parameter for each initiator. The user must determine the number of outstanding transactions based on the longest latency that initiator has to any agent it communicates with. As a result, this

- Leads to oversizing the total buffer in target (DCE/DMI/DII)
- Allocation is fixed at design time – buffer space may be assigned even for agents that will not communicate in the real system (e.g. due to address map configuration)

4.9.1 Previous Skid buffer architecture

Each target implements a skid buffer. This structure consists of a n-entry deep queue to store arriving requests and an arbiter that picks the oldest request, given a certain priority/QoS level. The implementation of an age buffer for the arbitration grows with the square of the number of entries. The data for each request occupies about 145 bits in the queue storage.

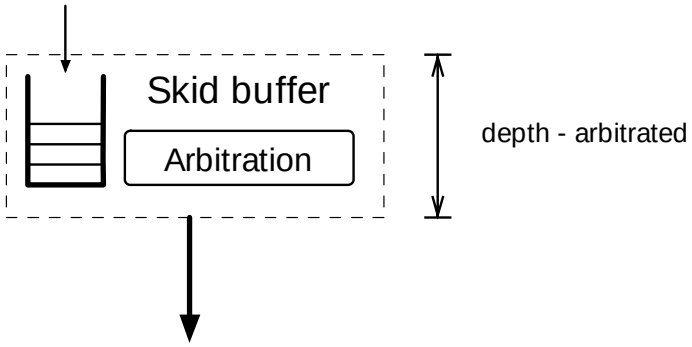


FIGURE 4-22: SKID BUFFER OVERVIEW

The physical size of the skid buffer is highly dependent on the total number of entries and grows quadratically with the number of entries.

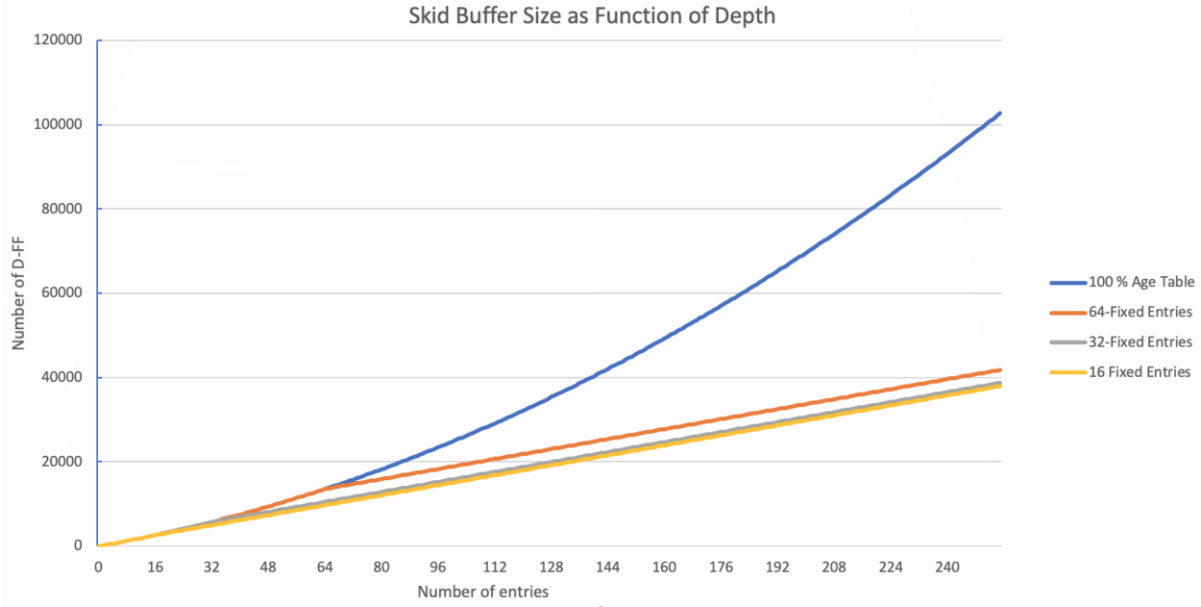


FIGURE 4-23: SKID BUFFER AS FUNCTION OF DEPTH

The logic structure to implement the arbitration is growing proportionally to the age buffer size and the complexity leads to significant loss in performance (speed).

As the total useful buffer is determined by an agent's bandwidth and latency – there is no use in frontloading a DMI with 100's of GB worth of traffic – an upper bound of skid buffer size for a target agent can be determined by looking at the total BW supported in steady state operation, the temporal distribution of requests and the number of interconnect buffers in the request path (queueing model interarrival time λ, Little's Law, service time μ).

4.9.2 Skid buffer optimization

The following chapters will describe several modifications to the previous implementation to address the problem of skid buffer explosion. Each proposal covers a different aspect of the issue and the combination of them will improve the power and area efficiency of Ncore and increase the flexibility of each design for the future.

4.9.2.1 Command Buffer Partitioning

The following issues of the previous skid buffer are addressed by this change:

- The previous implementation of Ncore implements the command buffers as a single, monolithic queue where arriving command requests are stored as they arrive from initiators and retrieved based on their age and priority
- The logic structure (age buffer) attached to this queue grows with the square of the number of entries
- In steady state traffic, the skid buffer will use only some part of the entire structure

The new command buffer architecture have the following requirements.

The physical implementation of the new command buffer shall consist of two parts:

- A fixed size, limited skid buffer - This limits depth of (arbitration) visible entries – only these entries will require arbitration in age-buffer (growing with $O(N^2)$)
- An overflow buffer to maintain total depth, by adding a FiFo in front of skid buffer – most efficient implementation, only needs D-FF/SRAM for txn-data storage

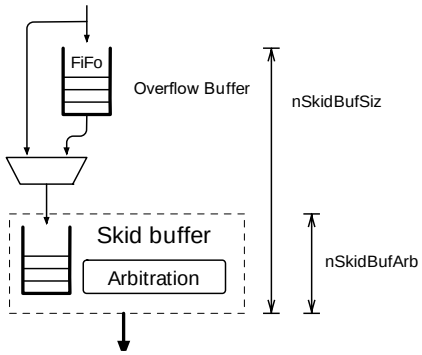


FIGURE 4-24: SKID BUFFER PARTITIONING

Figure 4-24 shows how the skid buffer shall be partitioned into an arbitration visible window at the bottom and an overflow buffer at the top. Two parameters are introduced to describe the depth of the entire skid buffer (nSkidBufSiz) and the size of arbitration window (nSkidBufArb). Please refer to the parameter specification for more details.

Static credit budgeting leads to overprovisioning of buffer space. Indeed, the number of credits actively used may be limited by SW scheduling – For some applications, the agents' credit may be adjusted by SW (driver) before kicking off active burst. For dynamic credit management overflow buffer can be small – just to protect against backup. Finally, the command buffer shall have sufficient depth to store

all credited transactions – for example during overlapped activity on multiple agents – without backing up traffic into interconnect (danger of deadlock). In steady state operation, the command buffer is only partially filled.

4.9.2.2 Software initialization of credit counters

What issues will be addressed by this change:

- The previous implementation of Ncore allocates a fixed amount of buffer space for each flow
- The allocation is fixed at design time and may not be changed
- The assignment of credits for an initiator treats all targets to the same amount of credits, independent of differences in round trip latency and BW requirements

The Ncore 3 new architecture have the following requirements:

1. The value of credits shall be programmable at each initiator
    a. For each reachable target, a CSR shall provide a field to modify the allocation of credits for a flow
    b. Maestro shall provide path vectors (calculated based on connectivity) to support generation of the required Credit Control Registers:
        i. For each CAIU:

Name	Elements	Description
boolDcePath[i]	# of DCE in system	a boolean, indicating a command path exists from CAIU to DCE[i]
boolDmiPath	# of DMI in system	a boolean, indicating a command path exists from CAIU to DMI[i]
boolDiiPath	# of DII in system	a boolean, indicating a command path exists from CAIU to DII[i]

        ii. For each NCAIU:

Name	Elements	Description
boolDmiPath	# of DMI in system	a boolean, indicating a command path exists from CAIU to DMI[i]
boolDiiPath	# of DII in system	a boolean, indicating a command path exists from CAIU to DII[i]

        iii. For each DCE:

Name	Elements	Description
boolDmiPath	# of DMI in system	a boolean, indicating a command path exists from CAIU to DMI[i]

    c. The CSR shall provide a means to determine if a communication path between this initiator and a specific target exists
    d. The max. number of credits available for a flow shall be programmable at runtime
2. A mechanism to determine the configured skid buffer size for each target shall be provided by implementing a read-only CSR
3. The initial allocation (at power-on-reset) shall be determined as follows:

a. If a Boot Region has been configured at an initiator, the non-coherent target defined by the MIG used for the initial boot region shall be initialized to 2 credits
b. If a Boot Region has been configured at an initiator, the target(s) associated with the system DII shall receive two credits - this needs to consider possible interleaving (unlikely to be used, but possible if DMI used to host the boot region?)
c. all other credit counts in CAIU/NCAIU/DCE shall receive 0 credits - the boot loader firmware shall determine the available budget for each agent by reading the Skid Buffer Size Register (SBSR) for each

4. This description does not apply to credits allocated to snoop transactions or DV messages

The implementation is described hereafter.

The format of the credit control field shall be 8-bits wide:

Bit	Type	Counter State	Description
7:5	RO	000: Normal Operation	Counter is operating normally, some outstanding credits
		001: Empty (== 0)	All credits have been used - should be a transient state during which no additional transactions may be sent
		010: Negative (< 0)	All credits have been used - this state would normally be reached when SW reduces the credit limit to a smaller value by an amount that exceeds the currently available credits. This is a transient state, during which no transactions may be sent. As outstanding credits arrive, the counter will increment through zero to allow requests to resume
		011: Reserved	Should never be observed - Error
		100: Full (== Limit)	No outstanding transactions, all credits are back
		101: Reserved	Should never be observed - Error
		110: Reserved	Should never be observed - Error
		111: No Connection	Special signature to indicate a non-existing connection between this initiator and the target represented by this control field. Maestro implements a connectivity map where some routes may not be required, this code allows firmware to recover the implemented connectivity map between initiators and targets
		Credit Limit	
4:0	RW	0 .. 31	This 5-bit value may be programmed to set the credit limit available for this connection. It is the user firmware's responsibility to set this value correctly, so that the total number of credits assigned between all initiators does not exceed the available buffer space

The credit counters:

The credit counter shall implement a two's complement counter using 6 bits. The counter shall be updated:

- when a credit is consumed - decrement counter
- when a credit is returned by receiving a response - increment counter

- when the CreditLimit is written - add the value calculated by newCreditLimit - oldCreditLimit. This value may be negative if the new value is smaller than the old value, effective decreasing the available credit

All 3 events may occur in the same clock cycle!

At reset the counter shall be cleared to zero or preset to a positive value.

Any counter value ≤ 0 indicates that no credits are available and requests must not be sent.

The Credit Control Registers (CCR) in CSR space:

- Ncore architecture may eventually support up to 32 DCE/DMI/DII and we shall plan our register map accordingly
- Each agent will be assigned an ordinal number withing its group (0..31)
- For AIU/NCAIU, each 32-bit Funit.CSR[index] shall pack credit control fields for all agent types (DCE, DMI, DII) using the same index, starting at the least significant byte, for example:
    - AIU[x].CCR[0] = {0xE0, DII_CCF, DMI_CCF, DCE_CCF}
    - AIU[x].CCR[1] = {0xE0, DII_CCF, DMI_CCF, DCE_CCF]

The most significant field in the register has been reserved for future expansion

- CCR registers shall be starting at offset 0xC00 within each unit's local address window
- The number of registers created shall be determined by the unit with the highest index
- Partially filled slots in registers shall be initialized with 0xE0, indicating to firmware that either no unit exists with this index or that the connection between this initiator and the target, mapped into this slot, is not implemented (no path exists)

The Skid Buffer Size Info Register (SBSIR) in CSR space:

Each unit implementing a skid buffer at its frontend shall announce the implemented size of the skid buffer in a CSR read-only register or a read-only field in an existing register.

The register shall be using this format:

Bit	Type	Meaning	Description
7:0	RO	SkidBufArb (0 .. 255)	Number of entries visible to arbitration in s split skid buffer configuration, value must be ≤ SkidBufSiz This value is supplied by a new parameter: **nSkidBufArb**
15:8	RZ	Reserved (0)	Keep for future expansion
25:16	RO	SkidBufSize (.. 1023)	Total Size of skid buffer implementation - this value may be identical to SkidBufArb if no split skid buffer has been implemented This value is supplied by a new parameter: **nSkidBufSize**
30:26	RZ	Reserved	Keep for future expansion

31	RO	Valid = 1	Valid bit, indicates presence of the register

The CSR address space for DCE/DMI/DII has an empty slot at 0xFF0 which shall be used to implement this register, DMI shall implement an additional register at 0xFE0 to implement a register for MRD buffer space information.

4.9.3 Software Support (Maestro)

New parameters are required to configure skid buffers at each target. For DCE and DII only one parameter set is required, DMI implements two skid buffers, one for coherent, the other for non-coherent traffic. Each FU shall have a private version of their parameters to support individual customization.

4.10 Exclusive monitor

4.10.1 Coherent Exclusive support

CHI specifies Ex Load as ReadClean, ReadNotSharedDrity & ReadShared. These translate to CmdRdCln and CmdRdVld in Concerto. Ex Store is specified as CelanUnique which translates to CmdClnUnq in concerto.

On receiving Ex Load CHI-AIU always responds back with pass when data is transferred.

The DCE implements a basic monitor and a configurable number of tagged monitors. (A DCE may be configured with zero tagged monitors.) The basic monitor consists of a valid bit for each Aiu funit ID and LpID pair, i.e. one bit for each processor in the system, while each tagged monitor consists of a basic monitor and an associated cacheline address field. The valid bit for each type of monitor

indicates whether a given processor is performing an exclusive access sequence, while the cacheline address field captures, or tags, the cacheline address for a given sequence. A tagged monitor is in-use if one or more processors are performing an exclusive access on the tagged address, i.e. any valid bit is set in the tagged monitor, and available if no processor is performing an exclusive access on the tagged address, i.e. no valid bits are set. At reset, the valid bits in the basic monitor and in any tagged monitors are set to the invalid state.

When DCE receives an Ex Load an address cam is done on all active tagged monitors and either one of the following three scenarios will get executed:

- If there is an address match corresponding processors ID valid bit is set in that monitor and the valid bits of the processor are cleared in other tagged monitors and basic monitor.
- If there is no address match and an empty tagged monitor is available, then that monitor is loaded with the address and the corresponding processors ID valid bit is set. Furthermore, the valid bits of the processor are cleared in other tagged monitors and basic monitor.
- If there is no address match and no empty tagged monitors are available, the corresponding processors ID valid bit for basic monitor is set. Furthermore, the valid bits of the processor are cleared in other tagged monitors.

Note that above actions of clearing the processor ID bits in other monitors ensures that at any given time a processor can do only one exclusive per DCE (in other words address region specified by DCE interleaving).

When DCE receives an EX Store an address cam is done on all active tagged monitors and either one of the following scenarios will get executed:

- If the address matches and the valid bit for that processor in the tagged monitor is set, then the valid bit for the processor is set in the basic monitor. Furthermore, the valid bit for the processor is cleared in the matching tagged monitor. In this case, the exclusive store passes.

- If the address matches and the valid bit for that processor in the tagged monitor is not set, then the valid bit for the processor is set in the matching tagged monitor. Furthermore, the valid bits for the processor are cleared in all other tagged monitors and the basic monitor. In this case, the exclusive store fails.
- If the address does not match and the valid bit for that processor in the basic monitor is set, then the valid bit for the processor remains set in the basic monitor. Furthermore, the valid bits for the processor are cleared in all other tagged monitors and the valid bits of other processors within the basic monitor are cleared. In this case, the exclusive store passes.
- If the address does not match and the valid bit for that processor in the basic monitor is not set and at least one tagged monitor is available, then that tagged monitor is selected, the valid bit for the processor is set in the tagged monitor, and the address is tagged in that monitor. Furthermore, the valid bits for the processor are cleared in all other tagged monitors and the basic monitor. In this case, the exclusive store fails.
- If the address does not match a tagged address and the valid bit for that processor in the basic monitor is not set and no tagged monitors are available, then the valid bit for the processor is set in the basic monitor. Furthermore, the valid bits for the processor are cleared in all tagged monitors. In this case, the exclusive store fails.

In the case when an exclusive message passes DCE completes the exclusive store as expected by sending out required snoops and updating the snoop filter. DCE will also set the ExOkay bit in the StrReq.

In the case where exclusive message fails DCE completes the transaction but does not send out any snoops (that may have been required) or change the snoop filter status. DCE will also not set the ExOkay bit in the StrReq.

Note that this implementation restricts one exclusive operation per processor per address region. here the address region is defined by the interleaving done across DCEs.

4.10.2 Non coherent exclusive support

Ncore 3 does not implement non-coherent exclusive monitors. Non-coherent exclusive transactions from an initiator connected to an AIU are forwarded to the intended target connected to either DMI or DII. The target is required to implement exclusive monitors. 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 64bytes 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 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.

Ncore units must implement following

- AIUs (CHI_AIU & IO-AIU)
    - Must forward the exclusive (ES/LOCK) signal on to the NCread & NCwrite transactions
    - Must forward the original (LPID/AXI-ID) on to the MPF2 field for NCread & NCwrite transactions
    - Must set VZ for all non-coherent exclusive transactions
    - For Read Exclusive AIU must pick up exclusive status from DtrReq CM status bit 0 (set for Ex-Okay)
    - For Write Exclusive AIU must pick up exclusive status from DtwRsp CM status bit 0 (set for Ex-Okay)
- DMI
    - Must forward ES from NCread and NCwrite transactions on to the AXI interface
    - Must forward original (LPID/AXI-ID) from the MPF2 field on to the AXI interface AXI-ID field
    - If a non-coherent exclusive hits the cache then DMI must not set CM status bit 0 for either DtrReq or DtwRsp
    - If a non-coherent exclusive misses the cache then DMI must set CM status bit 0 based on the Brsp for DtwRsp and Rrsp for DtrReq
- DII
    - Must forward ES from NCread and NCwrite transactions on to the AXI interface
    - Must forward original (LPID/AXI-ID) from the MPF2 field on to the AXI interface AXI-ID field
    - Must set CM status bit 0 based on the Brsp for DtwRsp and Rrsp for DtrReq

4.11 PCI Express support

4.11.1 Assumptions

PCIe supports multiple interleaved transactions, sharing a common transport medium. The different transaction types are listed in Table 4-10 (Transaction Type column). The PCIe root complex in an SoC would convert these transactions into ACE-Lite protocol when interfacing to an Ncore 3 IOAIU.

Depending on the address space selected and the semantics of the transaction, Ncore 3 would then propagate these to one of 3 types of functional interfaces:

- coherent memory --> DCE
- non-coherent memory --> DMI
- non-coherent memory and device transactions --> DII
- The PCIe IP ACE-Lite bus master will only use ReadNoSnoop, WriteNoSnoop, ReadOnce and WriteUnique transactions

TABLE 4-10: EXPECTED PCIE TRANSACTION TO AXI MAPPING

Item	Transaction Type	Mode	Target	ACE-Lite Transaction - AxCACHE[3:0]
1	Memory Read	Non-Posted	DCE[6]	normal, cacheable, bufferable
2			DMI	normal[7]
3			DII	normal, non-cacheable[8]
4	Memory Write	Posted	DCE	normal, cacheable, bufferable
5			DMI	normal, non-cacheable, bufferable
6			DII	normal, non-cacheable, bufferable/non-bufferable[9]
7	Memory Read Lock[10]	Non-Posted	DCE/DMI	normal, non-bufferable
8			DII	normal, non-cacheable, non-bufferable
9	IO-Read	Non-Posted	DII	device, non-bufferable
10	IO-Write	Non-Posted	DII	device, non-bufferable
11	Configuration Read (Type 0/1)	Non-Posted	DII	device, non-bufferable
12	Configuration Write (Type 0/1)	Non-Posted	DII	device, non-bufferable
13	Message	Posted	DII	device, non-cacheable, non-bufferable

[6] All cacheable memory accesses are bufferable

[7] Technically any normal memory access is supported

[8] Cacheable transactions are not expected

[9] Both transaction types are legal - SNPS IP uses non-bufferable transactions

[10] AXI4 does no longer support locked transactions

14	AtomicFetchAdd[11]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable
15	AtomicSwap[11]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable
16	AtomicCAS[11]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable

ARM protocol specifies certain behavior - which, for certain sequences requires some attention to avoid deadlocks, especially if traffic arriving from one PCIe device will be forwarded to a different PCIe device (bridge functionality).

Response:

- What AXI/ACE-Lite transaction encoding/attribute setting are MSI transactions using
  a. MSI Transactions are "Memory" type transactions - an MSI Write would be a "Memory Write"
  b. The bufferability attribute is generated (in default condition) based on the NS bit on the PCI link packets.

     If NS[12] = 1, ACELite = Non-Bufferable

     if NS = 0, ACELite = Bufferable

- What encoding/attribute setting distinguishes posted from non-posted memory writes

  Posted PCIE link Writes are always generated with AXI-ID = 0

  Non-Posted PCIE link Writes are always generated with AXI-ID != 0

4.11.2 ACE-Lite Transactions generated by Synopsys PCIe Controller with AMBA Bridge

The controller supports the AXI4 protocol with additional support for the ACE-Lite protocol. To enable these features, you must set AMBA_INTERFACE =3 and CC_ACELITE_ENABLE =1.

ACE-Lite Features and Limitations:

- The AXI bridge slave does not implement ACE; it ignores the value of the DOMAIN, BAR and SNOOP fields; barrier transactions are not supported
- The AXI bridge master interface generates requests that are dependent upon the NS and TH bits

[11] Our ACE-Lite implementation supports atomics but atomics must be cacheable and will not terminate properly if the addressed region is not backed up by SMC or if the region is not addressing scratch pad memory

[12] NS = non-snoop property of PCIe TLP

- The master interface always sets AxBAR and AxSNOOP to '0' so that requests are restricted to ReadNoSnoop, WriteNoSnoop, ReadOnce, and WriteUnique
- You can override the value of the DOMAIN field, except for message requests where it is always 11
- The master does not support barriers
- Cache coherency is not optional, and the master always uses the NS bit, unless you override the DOMAIN signals

TABLE 4-11: ACE LITE TRANSACTIONS - SNPS PCIE IP

Cache Coherency	Request Type	Memory Type[13] 0: Peripheral 1: Main Memory	NS[14]	TH	AxSNOOP	AxBAR	AxDomain	Shareability Domain	Memory Type
Non-coherent	Message	X	X	X	4b0000	2b00	2b11	System	Device, Non-bufferable
Non-coherent	Read or Write	0	X	X	4b0000	2b00	2b11	System	Device, Non-bufferable
Non-coherent	Read or Write	1	1	X	4b0000	2b00	2b11	System	Normal, Non-cacheable Non-bufferable
Coherent	Read	1	0	0	4b0000	2b00	2b10	Outer Shareable	Writeback, No-allocate
Coherent	Write	1	0	0	4b0000	2b00	2b10	Outer Shareable	Writeback, No-allocate
Coherent	Read	1	0	1	4b0000	2b00	2b10	Outer Shareable	Writeback, Read-allocate
Coherent	Write	1	0	1	4b0000	2b00	2b10	Outer Shareable	Writeback, Write-allocate

This Table 4-11 has been copied from SNPS PCIe IP Databook, Table 7-82 in section 7.4.2

Item	PCIe Transaction Type	Mode	Target	ACE-Lite Transaction - AxCACHE[3:0]
1	Memory Read	Non-Posted	DCE[15]	normal, cacheable, bufferable
2			DMI	normal
3			DII	normal, non-cacheable
4	Memory Write	Posted	DCE	normal, cacheable, bufferable
5			DMI	normal, non-cacheable, bufferable
6			DII	normal, non-cacheable, bufferable/non-bufferable
7	Memory Read Locked[16]	Non-Posted	DCE/DMI	normal, non-bufferable
8			DII	normal, non-cacheable, non-bufferable

[13] Set by the CFG_MEMTYPE_VALUE field of the COHERENCY_CONTROL_1_OFF register

[14] NS = non-snoop property of PCIe TLP

[15] All cacheable memory accesses are bufferable

[16] AXI4 does no longer support locked transactions

9	IO-Read	Non-Posted	DII	device, non-bufferable
10	IO-Write	Non-Posted	DII	device, non-bufferable
11	Configuration Read (Type 0/1)	Non-Posted	DII	device, non-bufferable
12	Configuration Write (Type 0/1)	Non-Posted	DII	device, non-bufferable
13	Message	Posted	DII	device, non-cacheable, non-bufferable
14	AtomicFetchAdd[17]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable
15	AtomicSwap[11]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable
16	AtomicCAS[11]	Non-Posted	DCE/DMI	normal, cacheable, non-bufferable

4.11.3 The rules for PCIe transactions (see PCIe specification, Section 2.4 Transaction Ordering)

Table 4-12 defines the ordering requirements for PCIe Transactions. the columns represent a first issued transaction and the rows represent a subsequently issued transaction. The table entry indicates the ordering relationship between the two transactions. The table entries are defined as follows:

Yes The second transaction (row) must be allowed to pass the first (column) to avoid deadlock. (When blocking occurs, the second transaction is required to pass the first transaction. Fairness must be comprehended to prevent starvation.)

Y/N There are no requirements. The second transaction may optionally pass the first transaction or be blocked by it.

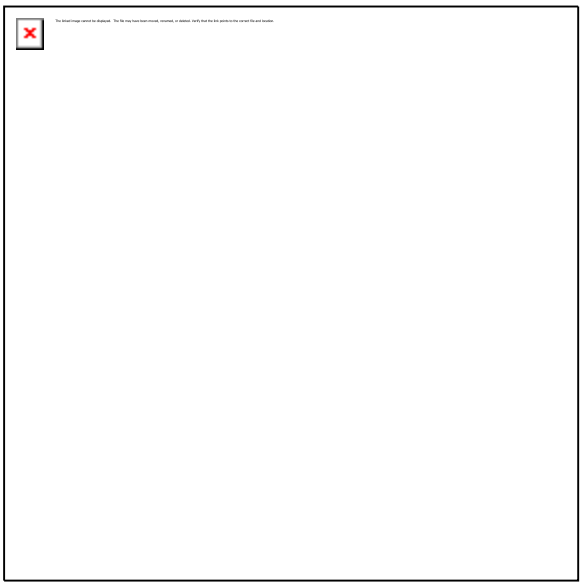
No The second transaction must not be allowed to pass the first transaction. This is required to support the producer/consumer strong ordering model.

TABLE 4-12: PCIE ORDERING RULES SUMMARY

Row Pass Column?		Posted Request (Col 2)	Non-Posted Request		Completion (Col 5)
			Read Request (Col 3)	NPR with Data (Col 4)	
Posted Request (Row A)		a) No b) Y/N	Yes	Yes Yes	a) Y/N b) Yes
Non-Posted Request	Read Request (Row B)	a) No b) Y/N	Y/N	Y/N	Y/N
	NPR with Data (Row C)	a) No b) Y/N	Y/N	Y/N	Y/N
Completion (Row D)		a) No b) Y/N	Yes	Yes	a) Y/N b) No

[17] Our ACE-Lite implementation supports atomics but atomics must be cacheable and will not terminate properly if the addressed region is not backed up by SMC or if the region is not addressing scratch pad memory

Posted Request	is a Memory Write Request or a Message Request.
Non-Posted Request (with Data)	Configuration Write Request, an I/O Write Request, or an AtomicOp Request.
Non-Posted Request	Read Request or an NPR with Data
Row A, B, C	Special cases: a) applies to all transactions unless condition b) applies b) Requests are allowed to pass when ▪ RO (relaxed ordering attribute) is set ▪ IDO (ID ordering) applies and the Requester IDs are different ▪ Both requests have been issued using different PASID
Row D	2a) Completion must not pass a Posted Request 2b) Exception: ▪ An I/O or Configuration Write Completion is permitted to pass a Posted Request ▪ Completions with IDO are permitted to pass a Posted Request if they belong to independent flows (C.ID <> R.ID) 5a) Completions with different Transaction IDs are permitted to pass each other 5b) Completions with the same Transaction ID are not allowed to pass each other



4.11.3.1 The rules for AMBA transactions (see ARM IHI 0022G, Section A6.3 Transactions and Ordering)

A transaction is a read or a write to one or more address locations. The locations are determined by **AxADDR** and any relevant qualifiers such as the Non-secure bit in **AxPROT**.

- Ordering guarantees are given only between accesses to the same Memory location or Peripheral region when the same AxID is used.
- A transaction to a Peripheral region must be entirely contained within that region.
- A transaction that spans multiple Memory locations has multiple ordering guarantees.

Transactions can be either of type Device or Normal:

Device: Device transactions can be used to access Peripheral regions or Memory locations. In Ncore 3.0 architecture, these transactions will be forwarded to DII.

Normal: Normal transactions are used to access Memory locations and are not expected to be used to access Peripheral regions.

A Normal access to a Peripheral region must complete in a protocol-compliant manner, but the result is IMPLEMENTATION DEFINED.

A write transaction can be either Non-Bufferable or Bufferable. It is possible to send an early response to Bufferable writes.

4.11.3.2 Guarantees before a completion response is received

In AMBA AXI/ACE-Lite protocol the transaction ID identifies a "flow" of associated transactions, originating at a requester agent and terminating at a target agent. Since the protocol does not identify individual transactions, ordering rules for transport, execution and responses are required to maintain consistency and to associate the request with a response.

All of the following guarantees apply to transactions from the same master, using the same ID:

- A Device write DW1 is guaranteed to arrive at the destination before Device write DW2, where DW2 is issued after DW1 and to the same Peripheral region.
- A Device read DR1 is guaranteed to arrive at the destination before Device read DR2, where DR2 is issued after DR1 and to the same Peripheral region.
- A write W1 is guaranteed to be observed by a write W2, where W2 is issued after W1 and to the same Memory location.
- A write W1 that has been observed by a read R2 is guaranteed to be observed by a read R3, where R3 is issued after R2 and to the same Memory location.

  The guarantees imply that there are ordering guarantees between Device and Normal accesses to the same Memory location.

4.11.3.3 Guarantees from a completion response

A completion response guarantees all of the following:

- A completion response to a read request guarantees that it is observable to a subsequent read or write request from any master.
- A completion response to a write request guarantees that it is observable to a subsequent read or write request from any master. This observability is a requirement of a system that is multi-copy atomic.

  Systems that contain Arm Architecture-compliant processors must be multi-copy atomic. That is, the Multi_Copy_Atomicity property must be ***True***.

The response to a Bufferable write request can be sent from an intermediate point. It does not guarantee that the write has completed at the endpoint, but it is observable to future transactions.

4.11.3.4 Response ordering guarantees

Transaction responses have all the following ordering guarantees:

- A read R1 is guaranteed to receive a response before the response to a read R2, where R2 is issued from the same master after R1 with the same ID.
- A write W1 is guaranteed to receive a response before the response to a write W2, where W2 is issued from the same master after W1 with the same ID.

4.11.4 What rules do we follow in IOAIU?

4.11.4.1 Current uArch specification

- Maintain separate AxID linked lists for read and writes, responses for same AxID will be in order within a read or write channel
- Generate Ordering bits based on AxCache[3:1] values for transactions with same AxID
  - 0: WriteNonCohFull - transaction with no older dependent transaction with same AxID (separate list for reads and writes)
  - 2: if (AxCache[3:1] > 0) (Normal) and there is an older transaction with same AxID (separate list for reads and writes)
  - 3: if (AxCache[3:1]==0) (Device) and there is an older transaction with same AxID (separate list for reads and writes)
- Add a CSR bit to run in producer consumer mode
  - In this mode IO AIU will block same AxID transactions until the STR response is sent for the dependent transaction (separate list for reads and writes)
  - When mode is not enabled then blocking is not required
- Same cache line address blocking until STR response is sent. This is done across reads and writes to avoid hazards (WAW, WAR, RAW)
- Writes will have at-least one or more reserved OTT entries and always make forward progress form IO AIU perspective.

4.11.4.2 The impact of the uArchitectural rules applied to PCIe transaction sequences

- The current implementation of Ncore does not support the concept of posted requests - order is enforced between all writes with the same AxID
  - 'non-posted' writes will not pass 'posted' writes (C2a) - **required**
  - 'non-posted' writes will not pass 'non-posted' writes (C4) - **less optimal, but legal**[18]
  - 'posted' writes will not pass other 'posted' writes (A2a) with the same ID or address - **required**
  - 'posted' writes will not be able to pass 'non-posted' writes (A4a) with the same ID - in principle this violates the requirement but:
    a. If posted writes use a different ID than NPRs, this problem goes away[19]
    b. Ncore guarantees forward progress for writes vs. reads - posted writes will make forward progress[20]
    c. Is having no posted writes a problem with performance? - **Yes**
  - relaxed ordering is implied by not using the same ID (A2b, A4b, C2a, C4) - **allowed**
- Read requests will pass all write requests, except if a hazard exists

[18] This are AXI rules - we allow to relax them for memory targets by address region (GPRAR registers)

[19] Strict request ordering based on the AxID - but performance suffers

[20] Yes, a single open entry does not look like much but, writes will always have priority over reads, so when OTT is full, a write will get in for every returning transaction (pipelining)

- read requests must not pass 'posted' write requests with the same ID and the same address[21] (B2a) - **implemented** (we respect WAR, RAW, WAW order, independent of AxID)
- read requests may pass 'posted' write requests with a different ID (B2b) - **allowed**

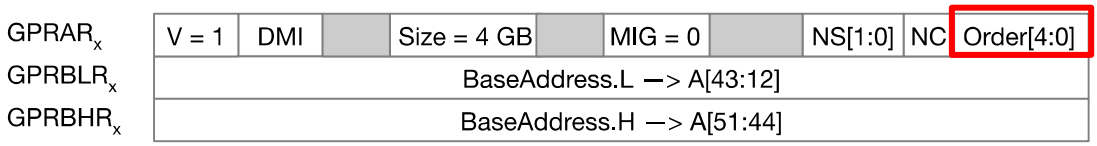
Notes:

1. This are AXI rules - we allow to relax them for memory targets by address region (GPRAR registers)
2. Strict request ordering based on the AxID - but performance suffers
3. Yes, a single open entry does not look like much but, writes will always have priority over reads, so when OTT is full, a write will get in for every returning transaction (pipelining)
4. This is not explicitly stated, but inferred from memory consistency and R-W ordering requirements - reads will always return the data written by a write that has been issued earlier.

[21] This is not explicitly stated, but inferred from memory consistency and R-W ordering requirements - reads will always return the data written by a write that has been issued earlier.

4.11.5 Performance improvement discussion

- Prefer to define ordering by address region (CMN does it that way) instead of target FUnit, each region is defined by a GPRAR



- New order control field in each GPRAR$_x$
- The control field (bits 0.. 2) within this register controls the rules when a transaction may be **issued** to the internal target (DCE/DMI or DII)[22]
- The **Policy** field determines the behavior at the target agent - DCE and DMI ignore the policy setting - and the response behavior
- Device transactions (AxCACHE[3:1] == 3b000) will enforce strict request ordering (Policy == 2b11) for all requests with the same AxID - these shall only be used for targets behind a DII.
- Memory request addressing DII targets may use stricter rules (Request order)

TABLE 4-13: CSR CONFIGURATION

Order	Function	Description
0	Reserved	Always read as zero
1	ReadID (ARID)	No ordering by AxID (free listing) for reads[23]
2	WriteID (AWID)	No ordering by AxID (free listing) for writes[15]
4:3	Policy 11 Endpoint Order 10 Relaxed Order 00 Reserved 01 Write Order	Policy determines the setting of our internal OR[1:0] field which defines the behavior of the agent at the bottom of Ncore Endpoint[24] order: ▪ All transactions to the same peripheral region are issued and completed in order ▪ Override the ordering mode of the incoming transaction to *Endpoint Order* Relaxed order[25]: ▪ Responses within a flow (same AxID) are ***ordered*** if ReadID/WriteID bits are zero, otherwise they are ***unordered*** ▪ ***Ordered*** transactions will be issued as soon as the previous transaction in the ordered sequence has been ordered at the target (StrReq has been returned) and command credits are available ▪ ***Unordered*** transactions will be issued as soon as command credits are available at the target ▪ Write: BRESP may be returned (earliest) as soon as DtwRsp has been received, following AXI ordering rules (all older responses have been sent)

[22] Read/Write ordering rules may be different for each target region - Setting ReadID/WriteID field overrides AxID ordering!

[23] Setting ReadID and WriteID (freelisting) will force the internal OR[1:0] field to 2b00 (unordered) for memory targets

[24] Read: Issue --> DtrReq, Response --> DtrReq --- Write: Issue --> DtwRsp, Response --> DtwRsp. Setting ReadID or WriteID bits is not recommended for this policy and may result in undefined behavior

[25] The equivalent of ***Strict Response*** ordering can be achieved by using ***Relaxed Order*** and setting the ReadID and WriteID bits

		Write order: ▪ This mode has been added to allow writes to follow PCI rules (A2a, C2a) at the DII Reserved: ▪ Shall be treated as Endpoint order (the same behavior as Policy = 11)

Recommended Settings

TABLE 4-14: RECOMMENDED SETTINGS

AxCACHE[3:0]	Target	Policy [1:0]	WriteID (AWID)	ReadID (ARID)	Description
4b000x	DII - Device	2bxx	0	0	Device property of transaction overrides policy setting
4b0001	DII - PCIe Device	2b01	0	0	Write ordering at DII keeps writes ordered, but allows posted writes to pass reads
4bxx1x	DII - Memory	2b11	0	0	Transaction to memory mapped device functions (GIC) - this setting guarantees completion of all outstanding writes to any target using the same AWID
4bxx1x	DII - Memory	2b10	0	0	Memory transaction (SRAM, ROM, Flash) - AxID ordered requests, Hazard check at PoS[26] or in AIU
4bxx1x	DII - Memory	2b10	0	1	Memory devices with write ordering requirements (some Flash), Hazard check in AIU
4bxx1x	single DCE/DMI	2b10	1	1	Issue of transactions free listed, Response

[26] Khaleel says, that hazard check may fail under rare circumstances - need to understand the conditions and fix if necessary

					follows Relaxed Order policy, Hazard check in AIU Best bandwidth
4bxx1x	interleaved DCE/DMI	2b10	1	1	Issue of transactions free listed, Response follows Relaxed Order policy, Hazard check in AIU Best bandwidth
4bxx1x	DCE/DMI	2b10	0	0	AXI ordering enforced - severe bandwidth penalty

Concerns about complete freewheeling

Sequential Consistency[27] cannot be guaranteed for interleaved memory transactions - a sequence of memory writes by a writer will alternately/sequentially target different memory agents (e.g. interleaving 4 DCEs at 64 bytes will result in transactions for sequential cache lines to be sent to different DCEs and/or DMI). This may be no issue for most applications.

0x0000 --> DCE_{00}, 0x0040 --> DCE_{01}, 0x0080 --> DCE_{02}, 0x00c0 --> DCE_{03}, 0x0100 --> DCE_{00} ... each DCE implements the Point-of-Coherence for part of the interleaved address space. If an observer agent's read requests do not arrive at the DCEs in the same order (due to different transport latency) the write-read order at each PoC may differ and writes may be observed in a different order than they were issued.

	DCE_{00}	DCE_{01}	DCE_{02}	DCE_{03}
t_0	W_0	W_1	W_2	W_3
t_1	R_0	R_1	W_6	R_3
t_2	R_4	R_5	R_2	R_7
t_3	W_4	W_5	R_6	W_7
t_4	R_8	R_9	W_9	W_{10}

Race condition between W_6 and R_2 has W_6 arrive earlier than R_2 - agent DCE_{02} sees a different request order and the data returned in R_3 will be different (returning the previous content of memory

[27] This is a standard problem when traffic is split to interleaved memory channels - this is a trade-off with performance impact. Applications using Producer-Consumer models need to be aware of this behavior and shall apply appropriate caution

					location 2) than expected under the assumption of a sequential write. This is a problem of having multiple PoC processing a sequential data flow originating at a single source.

Streaming write ordering

Writes arriving at the IOAIU from an external agent (e.g. PCIe complex) follow AXI protocol. Write transactions are tagged with the AWID, all transactions carrying the same ID are part of the same flow and for memory targets shall be considered sequentially ordered within that flow.

In most use cases, the agent observing a sequential stream of writes within a flow needs to observe these writes exactly in the same order they were created. This is called Ordered Write Observation and required by sequential memory consistency.

Ncore implements Ordered Write Observation by making writes visible to observers when the ordering point receives the StrRsp message; this is the idication, that the write has been completed at the POS. In CCMP StrRsp concludes the write operation.

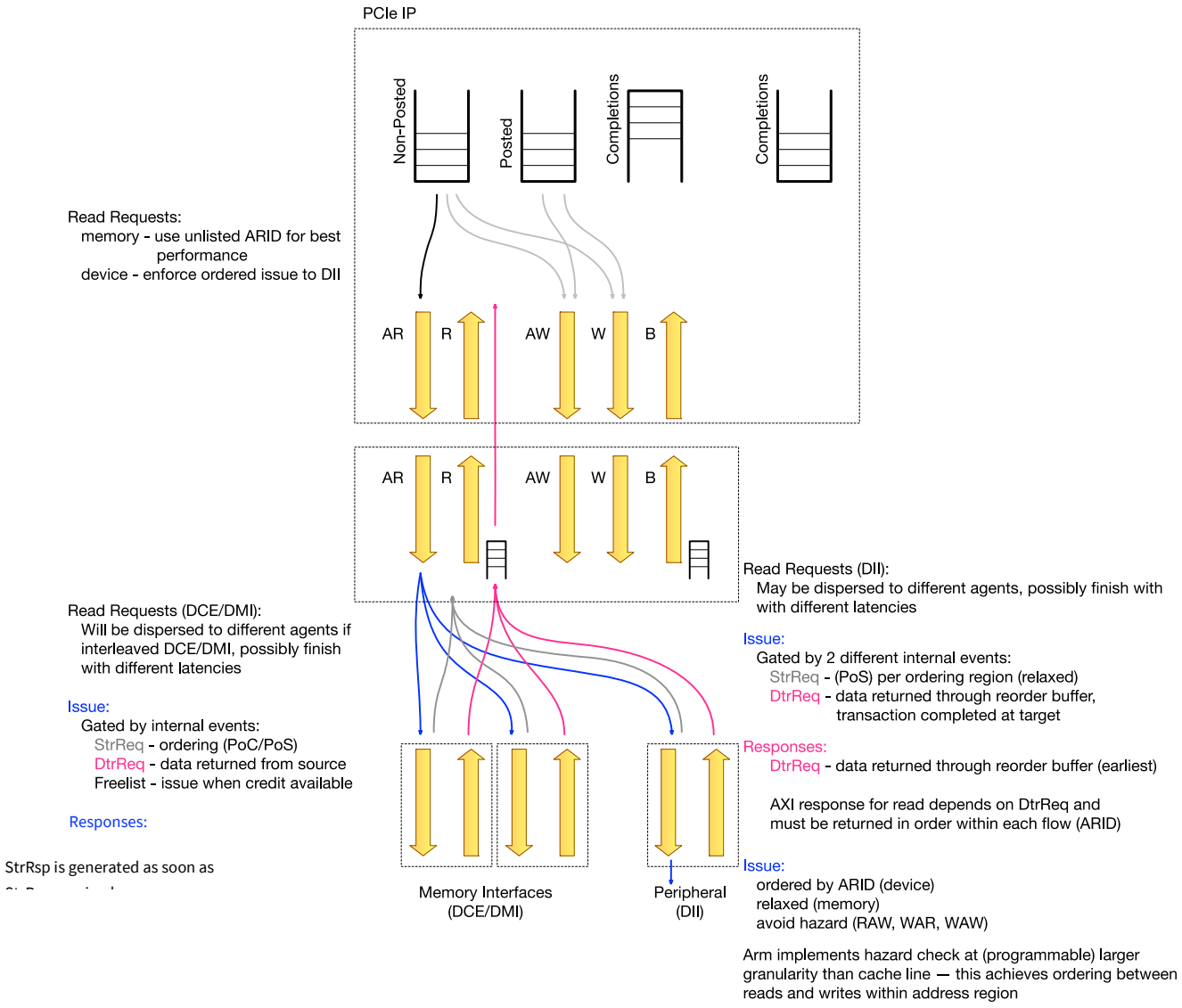
4.11.6 Transaction Progress through the system

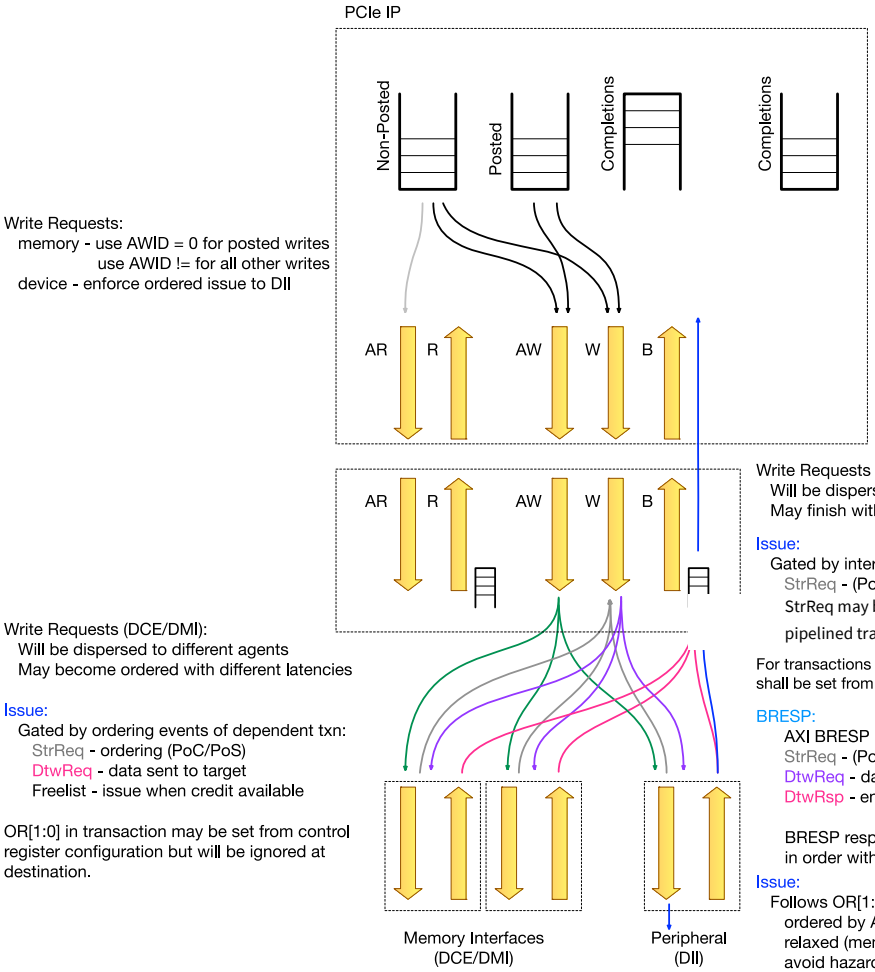
Acceptance of transactions

In AXI or ACE-Lite protocol read and write transactions may be arriving concurrently. The protocol does not infer any ordering between read and write channels in that case.

NCAIU

AXI or ACE-Lite transactions arriving at the NCAIU carry AxCACHE[3:0] attributes - NCAIU shall treat any transaction with AxCACHE[3:1] == 3b000 as device ordered and enforce request order - these transactions are usually sent towards physical devices and will require strong ordering.





Write Requests:
memory - use AWID = 0 for posted writes
use AWID != for all other writes
device - enforce ordered issue to DII

Write Requests (DCE/DMI):
Will be dispersed to different agents
May become ordered with different latencies

Issue:
Gated by ordering events of dependent txn:
StrReq - ordering (PoC/PoS)
DtwReq - data sent to target
Freelist - issue when credit available

OR[1:0] in transaction may be set from control register configuration but will be ignored at destination.

Write Requests (DII):
Will be dispersed to different agents
May finish with different latencies

Issue:
Gated by internal events, depends on CSR setting:
StrReq - (PoS) per ordering region (relaxed)
StrReq may be used to send the next
pipelined transaction, ordering will occurr at

For transactions with AWCACHE Properties of OR[1:0] in transaction shall be set from control register configuration

BRESP:
AXI BRESP response for write may be sent early
StrReq - (PoS) per ordering region (relaxed)
DtwReq - data sent from reorder buffer
DtwRsp - endpoint has accepted data

BRESP response follows AXI rules and must be returned in order within each flow (AWID)

Issue:
Follows OR[1:0] ordering rule in transaction
ordered by AWID (device)
relaxed (memory)
avoid hazard (RAW, WAR, WAW)

Arm implements hazard check at (programmable) larger granularity than cache line — this achieves ordering between reads and writes within address region

Further ideas:

- Support early write responses if target is DCE or DMI and the write is a buffered write to memory. Sending an early response implies that the receiving agent takes ownership of the transaction, including the data and promises to follow the ordering rules of AXI/ACE-Lite
    - write transactions are accepted - AWADDR
    - WReady is returned - interface is ready to accept data
    - WData is received
    - BReady is returned as soon as the last data beat has arrived
    - Now the responsibility for transaction ordering is on the AIU
    - No other posted write transaction may pass a transaction that has been responded to
    - No read request with the same transaction ID may pass an earlier transaction that has been responded to
    - Posted write transactions may be issued as soon as no other posted write transaction with the same ID is ahead of it (all older have at least received StrReq - they have been ordered)
    - The entry in OTT, and the buffer space holding the data, can be released as soon as DtwReq has been issued - the next agent on the way to memory has now ownership of the transaction

- Use the "ordering" bits in our protocol to propagate the ordering rule for each transaction
- Treat *WriteUnique* and *WriteNoSnp* differently
- remove hazard checks at the top of the pipeline - if transactions make it to the destination, the hazard would be managed, yet the performance loss from stalling can be avoided as long as ordering between transactions on the way to PoC/PoS can be maintained
- add programmable granularity to hazard check in DII. This will support ordering between reads and writes within a local address region - avoid side effects
- Distinguish between posted/bufferable writes and non-bufferable writes - extend the ID by one bit 'B' to mark a bufferable write --> ID = {B, AXI-ID[n:0]} and use the following rule to allow passing:

TABLE 4-15: PCIE ORDER ENFORCEMENT

$Type_{New}$	B_{New}	$Type_{OTT}$	B_{OTT}	$ID_{New} == ID_{OTT}$	Comment	Action
R	0	W	X	No	No match, different flow	pass
W	X	R	0	Don't care	Hazard Check	pass
R	X	R	0	Don't care	Hazard Check	pass
W	0	W	0	Yes	NPR - NPR, Hazard Check WAW	pass

W	0	W	1	Yes	NPR - Posted	no pass
W	1	W	0	Yes	Posted - NPR	pass
W	1	W	1	Yes	Posted - Posted	no pass

Implementing additional optimizations supported by PCIe, e.g. the relaxed ordering requirements introduced in PCIe 3 (IDO), would require visibility of PCIe Requester ID and PASID (stream ID, address space identification) as part of the arriving ACE-Lite transaction. It is assumed that this would be handled within the PCIe root complex.

4.11.7 Proposed Changes to uArchitecture of NCAIU

This chapter describes the changes required to the NCAIU architecture to support the new functionality described in the previous sections of this document.

GPRAR - support

The GPRARs for each NCAIU need to add 5 bits to control the transaction behavior for each address aperture. The configuration bits 4:0 are currently not used across all AIU architectures and their function in CAIU will not change.[28]

TABLE 4-16: GPRAR FORMAT

CAIU GPRAR$_x$: General Purpose Region Attribute Register [Offset: 0x0400]				
Bit	**Name**	**Description**	**Access**	**Reset**
0	Rsvd	Reserved	RZWI	0x0
1	ReadID	No ordering by ARID (free listing)	RW	0x0
2	WriteID	No ordering by AWID (free listing)	RW	0x0
4:3	Policy	Response policy: 11: Endpoint/Request Order - do not set ReadID/WriteID! All memory transactions are issued and completed in order (issue from DtrReq/DtwRsp), BRESP from DtwRsp 10: Relaxed Order - next dependent transaction is issued as soon as ordering at the target has been achieved (StrReq), BRESP from DtwReq 01: Write ordering at DII - keeps writes ordered, but allows posted writes to pass reads 00: Reserved, shall be treated as Endpoint/Request Order		0x0
8:5	Rsvd	Reserved	RZWI	0x0
13:9	HUI	Identifier of the home unit, depending on type **Peripheral:** Index of target DII in enumerated list **Memory:** Memory Interleave Group **HierGateway:** Gateway Interleave Group **Chip-to-Chip:** Gateway Interleave Group	RW	0x0
19:14	Rsvd	Reserved	RZWI	0x0
24:20	Size	This field represents the size of the region in a binary number with a range of 0 .. 31. The size of the region is defined by RegionSize = IGSize * $2^{(size+12)}$ Bytes	RW	0x0
28:25	Rsvd	Reserved	RZWI	0x0
30:29	HUT	This field indicates the Home Unit Type:	RW	0x0

[28] CHI and ACE CAIU will not support these settings and propagate the protocol inherited attributes - register settings will be one either ignored or required to use a specified pattern (otherwise expect undefined behavior)

		00: System Memory 01: Hierarchical Gateway Interface 10: Peripheral 11: Chip-to-Chip interface Note, the pattern has been chosen to retain compatibility with the current, single bit selector (bit 30) 0: System Memory 1: DII		
31	Valid	This bit indicates if the region is valid 0: Invalid mapping 1: Valid region mapping	RW	0x0

New Rules for the transaction processing

Transaction management

- Maintain separate linked lists for read and writes by AxID
  - transactions may be issued depending on ordering requirements using the AxID
  - AxID ordering when issueing requests towards the target agent may be disabled for an address region by GPRAR[1][29] for read accesses and GPRAR[2][30] for write accesses
    - Setting GPRAR[x] == 1 disables checking
    - The default value (after reset) will be 0 (checking enabled)
  - Responses for same AxID must be in order within a read or write channel - responses for individual flows (all transactions with the same AxID) from downstream agents may arrive out-of-order and must be reordered to meet AXI specification
- Same cache line address blocking until the StrRsp has been sent. This is done across reads and writes to avoid hazards (WAW, WAR, RAW)
  - This check shall be controlled by GPRAR[0] the "hazard" bit
  - Checking shall be disabled when hazard == 1
  - The default value (after reset) will be 0 (checking enabled)

Transaction Issue

- Each transaction will be started by sending a CmdReq. The command includes a 2-bit wide "ordering" field OR[1:0]. This field determines request processing at the target agent (DCE, DMI, DII). Valid settings for OR[1:0] are

TABLE 4-17: ORDERING FIELDS

OR[1:0]	Function	Description
2b11	Endpoint ordered	Strongly ordered with respect to the same endpoint device
2b10	Strongly ordered[31]	Strongly ordered with respect to a previous access from the same source to the same location. Writes to the same address issued by the same agent (FUnitID and AxID) must meet requirements of sequential consistency
2b01	Write order	Write ordering occurs at DII - this mode is used in conjunction with disabled ID checking. Transactions will be issued at the AIU without checking for the completion of older writes from the same AxID
2b00	Not ordered	No ordering required for this access

- Target agents with memory semantics (DCE/DMI) will ignore the OR[1:0] setting
- Read/Write requests shall be issued when they are ready-to-issue, i.e. they meet certain conditions. The conditions to issue a transaction depend on the type of transaction (Read/Write), GPRAR [4:0], the

[29] Ignore ARID when set
[30] Ignore AWID when set
[31] Current documentation: ***Request Ordered***

AxCACHE[3:1] value and an "*Event*" trigger that removes the transaction from the dependency chain (eg. StrRsp). In AXI4, all transactions with AxCACHE[3:1] == 3b000 are considered "device" transactions and must be endpoint ordered

TABLE 4-18: DEVICE TRANSACTIONS (AXCACHE[3:0] = 4B000X)

Type	Event	GPRAR[4:0]	OR[1:0]	Description
Read	DtrReq/StrReq	5bxxxx0	2b11	Endpoint ordered, transactions are issued in order[32]
Write	StrReq	5bxxxx0	2b11	

[32] Initiator will not send another request until the current transaction has received RRESP/BRESP - therefore StrReq may be used to issue the next transaction to simplify the implementation

TABLE 4-19: MEMORY READ TRANSACTIONS (AXCACHE[3:0] = 4BXX1X)

Event[33]	GPRAR[4:0]	OR[1:0]	Description
StrReq/DtrReq[34]	5b10x00	2b10	Issue respects ARID as dependency, hazard check enabled
StrReq/DtrReq[34]	5b11xx0	2b11	Endpoint ordered, transactions are issued in order
StrReq/DtrReq[34,35]	5b10x10	2b00	Issue: Ignores ARID as dependency, hazard detected, this transaction does not generate a dependency for subsequent transactions

TABLE 4-20: MEMORY WRITE TRANSACTIONS (AXCACHE[3:0] = 4BXX1X)

Event[33]	GPRAR[4:0]	OR[1:0]	Description
StrReq	5b100x0	2b10	Issue respects AWID as dependency, hazard check enabled
StrReq	5b11xx0	2b11	Endpoint ordered, transactions are issued in order - a DII agent will return DtwRsp only after receiving BRSP
StrReq[35]	5b101x0	2b00	Issue: Ignores AWID as dependency, hazard detected, this transaction does not generate a dependency for subsequent transactions
StrReq[35]	5b10101	2b01	Issue: Ignores AWID as dependency, hazard detected, this transaction does not generate a dependency for subsequent write transactions at the IOAIU but, if sent to a DII target, it will issue writes in sequential order

Ordered Write Observation Implementation

Ordered write transactions guarantee that sequential write requests arriving on one agent would become visible to any (different) observing agent in the same order they are issued. On the CHI interface this property will be enabled by hazard checking for each write - the barrier for a write must be kept until all previous writes

Response Generation to Initiator Agent

- All responses within an AXI read or a write flow (same ARID or AWID), must be generated in the order the transactions arrive at the NCAIU. This property must be guaranteed, independent of their completion order at downstream agents.

[33] Event refers to message that removes the transaction as a dependency
[34] Whatever event arrives first
[35] Ignoring AxID, transaction can progress when the transaction causing the hazard has been ordered

- The earliest time a transaction may be responded to is a function of the policy programmed into GPRAR [4:3], the arrival of messages (StrReq, DtrReq, DtwRsp) from downstream agents and the type of transaction (read/write).
- Responses may be generated when the event message arrives and no older transaction with the same AxID is outstanding (issued and waiting to complete, or completed and waiting to generate a response)

TABLE 4-21: DEVICE TRANSACTIONS (AXCACHE[3:0] = 4B000X)

Type	Event	Response	Description
Read	DtrReq	RRESP	Endpoint ordered, responses may be generated when the event message arrives and no older transaction with the same AxID is outstanding (issued or completed and waiting to generate a response)
Write	DtwRsp	BRESP	

TABLE 4-22: MEMORY TRANSACTIONS (AXCACHE[3:0] = 4BXX1X)

Type	Event	GPRAR[4:3]	Response	Description
Read	DtrRsp	3b10	RRESP	Transactions are issued and completed in order at the downstream agent, data will be returned and saved in a reorder buffer
Read	DtrRsp	3b11	RRESP	
Write	DtwReq	3b10	BRESP	Ordered response, may be sent as soon as the responsibility for the transaction, including data, has been transferred to the completer
Write	DtwRsp	3b11	BRESP	Endpoint ordering, response will only be sent after the transaction has completed at the endpoint

AXI spec "Ordered write observation" says it "can support the Producer/Consumer ordering model with improved performance". AXI4 spec does not mention PCIE spec, but we know that the PCIE spec uses a Producer/Consumer ordering model. For instance, in a system like this:

CPU <-> PCIE Controller <-> PCIE AXI Bridge <-> AXI with Device and DDR slaves

(Device IP is connected to AXI slave data port0 and APB register port,
DDR memory module is connected to AXI slave data port)

The CPU performs the following two operations,

- CPU writes data to DDR
- CPU writes Device APB register to start Device activity

Because PCIE memory writes (both prefetchable and non-prefetchable) are posted, i.e. without responses, PCIE AXI Bridge will perform the above two operations successively with the same ID but without waiting for BRESP. Before the data reaches DDR, Device may have seen the APB register write and starts reading the data, so the data may be old and invalid.

If "**Ordered write observation**" is supported, there will be no such a problem, because it requires the interface "**if two write transactions, with the same ID, are observed by all other agents in the system in the same order that the transactions are issued**", i.e. if APB register write is observed by Device, it will guarantee data write to DDR has been observable to Device.

5 Addressing and Memory Regions

Ncore 3 address map is categorized into three main spaces:

- Ncore 3 Register Space (NRS)
- General Purpose Address Space (GPAS)
- Boot Region (BR)
- Each space may contain one or more address regions.

NRS is the address space reserved for peripheral storage space, hosting Ncore unit configuration and status registers (CSR). The Organization of NRS is shown in Figure 5-1

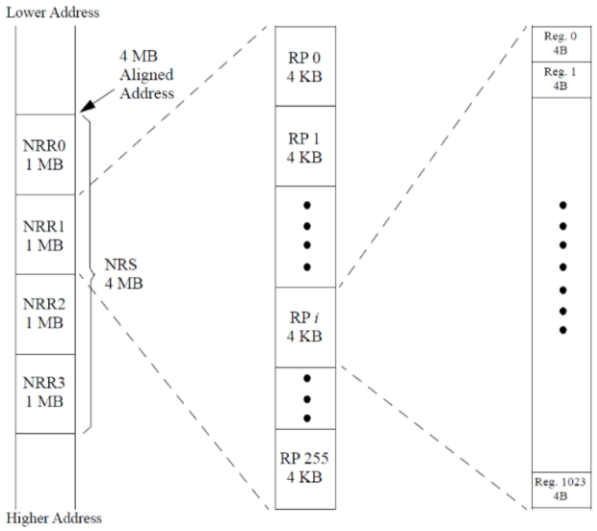


FIGURE 5-1: NRS STRUCTURE

5.1 Ncore Register Space

NRS may contain one or more Ncore register regions (NRR). Each NRR is of 1MB and is further divided into 256 register pages (RP) of 4KB.

Each Ncore 3 component is associated with an RP and may implement up to 1024, 4 Byte (32-bit) registers.

Currently, Ncore 3 system supports only one NRR. The number of valid RPs within this NRR is equal to the total number of Ncore 3 components (Function Units, FU) configured in the system.

Table 5-1 shows the assignment of RPs to different Ncore 3 components. Note that within the table:

- nCAIUs refers to the number of CAIUs in the system, and the number of other Ncore 3 components.
- nDII is always equal to total number of DIIs configured, plus one. The additional DII is the Ncore 3 system DII that will be implicitly mapped into the NRS.
- There will alway be only one DVE and one GRB.

TABLE 5-1: RP ASSIGNMENT TO NCORE COMPONENTS

Item	Ncore Component	RPNs
1	CAIU (Coherent AIU)	RPN(0) to RPN(nCAIU - 1)
2	NCAIU (Non-Coherent AIU)	RPN0nCAIU) to RPN(nCAIU + nNCAIU - 1)
3	DCE (Distributed Cohrence Engine)	RPN(nCAIU + nNCAIU) to RPN(nCAIU + nNCAIU + nDCE - 1)
4	DMI (Distributed Memory Interface)	RPN(nCAIU + nNCAIU + nDCE) to RPN(nCAIU + nNCAIU + nDCE +nDMI - 1)
5	DII (Distributed I/O Interface)	RPN(nCAIU + nNCAIU + nDCE + nDMI) to RPN(nCAIU + nNCAIU + nDCE + nDMI + nDII - 1)
6	DVE (Distributed Virtualization Engine)	RPN(nCAIU + nNCAIU + nDCE + nDMI + nDII + 1)
7	GRB	0xFF (255)

The base address of this region shall be specified at Ncore system build time and must be 4MB aligned. Figure 5-2 shows the address distribution.

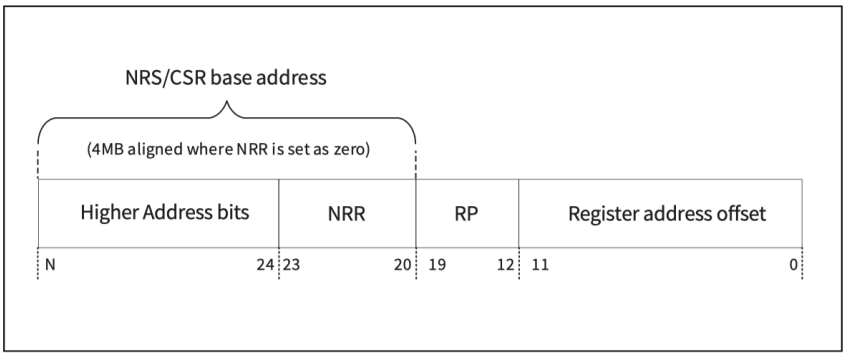


FIGURE 5-2: ADDRESS BIT MAPPING

5.1.1 Ncore CSR access security

This section describes the new CSR access method, improved from a security perspective.

5.1.1.1 Parameters

A new unit level parameter named fnCsrAccess that applies to all AIUs is introduced. It enables or disables CSR access via each AIU. At least one AIU is enabled in the system. Please refer to the parameters specification for more detail.

5.1.1.2 Detailed Description

A new NRSAR register is added. This register has a valid field that is set based on fnCsrAccess parameter. If the parameter is true then the valid field resets to 1 and if the parameter is false then the valid field resets to 0. Hardware should use this valid field to qualify a transaction hit to the CSR address space. When valid is not set this will likely result in no BAR hit error.

5.1.1.3 Ncore register space attribute register (xNRSAR)

Bits	Name	Access	Reset	Description
30:0	Reserved	RO	0x0	
31	NRS valid	R/W or RO	fnCsrAccess	Set high to enable CSR access via this AIU (This field must be RO for AIUs where the fnCsrAccess parameter is set and R/W where fnCsrAccess parameter is not set)

This secure access method is required because it is worth noting that NS bit is don't care for transactions to CSR and boot region.

5.2 General Purpose Address Space

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, minimum 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 bootup. Current Ncore system supports only 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 overlapping lower priority address space matches are ignored, and the higher priority space is selected for access. Within an address space 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 space or no matches are detected to any address space.

5.3 Interleaving

Ncore supports interleaving across snoop filters and system caches. 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 or 4 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

categorized into 3 different memory interleave groups:

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

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

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

Examples for groups have been illustrated in Figure 5-3:

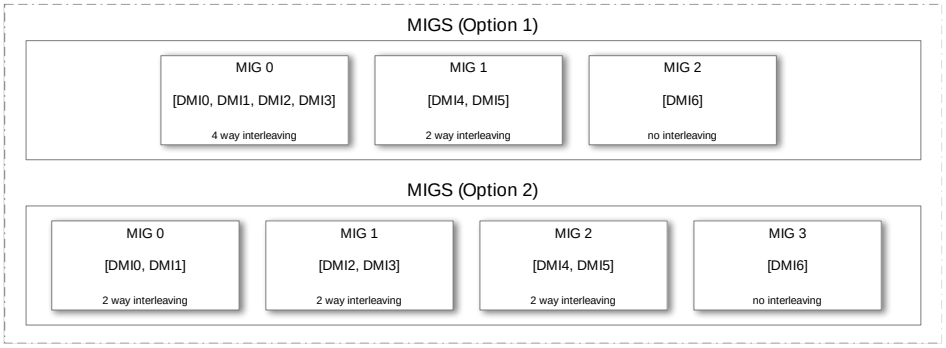


FIGURE 5-3 MIGS OPTIONS (EXAMPLES)

5.4 Instance and Set Selection

DCE & 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 cacheline. For example, if the size of the system address is 4 GB and the size of the directory cacheline is 64 Bytes, 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 cacheline comprise the directory cacheline 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

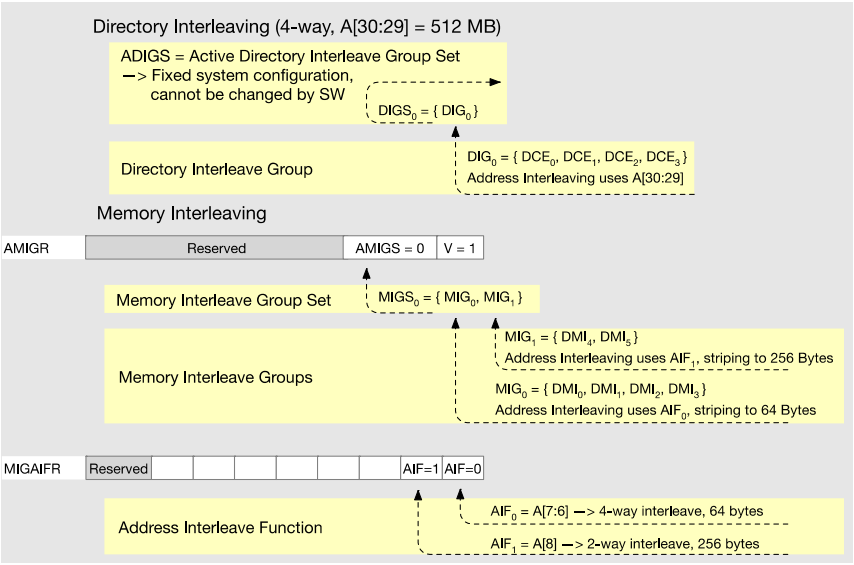


FIGURE 5-4: MEMORY AND DIRECTORY INTERLEAVING

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.

5.4.1 Directory Interleaving

The coherency protocol in Ncore is directory based, directories are located within the DCE. To improve performance up to 16 directories may be interleaved.

- Interleaving is based on one (2-way interleaving), up to four address bits (16-way interleaving)
- Directories are mapped into groups (DIG = Directory Interleave Group), where each group uses a fixed interleave factor of 2^N (N = 0 .. 4), configured in **Maestro**

- Multiple Groups may be combined to form a DIGS
- DIGS (Directory Interleave Group Set), each set may only combine DIG in a way so that no DCE appears more than once in the merged list
- Multiple DIGS may be defined in **Maestro**, one of these must be selected at system start by configuring the ADIGR in each AIU

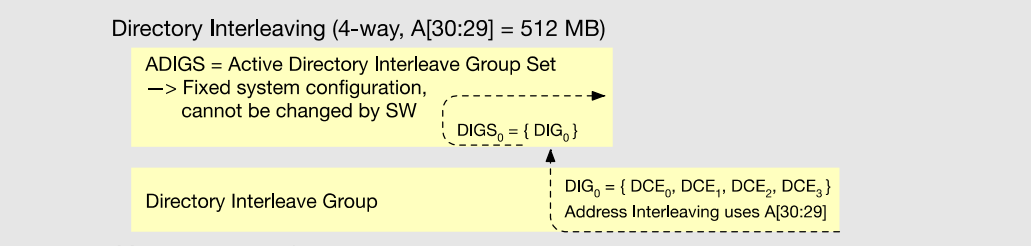


FIGURE 5-5: DIRECTORY INTERLEAVING

5.4.2 Snoop Filter, System Memory Cache and Proxy Cache

The Set selection is additionally constrained as following:

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

5.4.3 Distributed Memory Interface Selection

Ncore may be configured to use up to 16 Distributed Memory Interfaces (DMI). For performance, to balance the load and to distribute traffic evenly, DMI may be interleaved. The interleaving options are configured in **Maestro** by defining up to 8 Address Interleaving Functions.

- Interleaving is implemented by an Address Interleave Function (AIF)
- An AIF is based on 0 (no interleaving) to 4 (16-way) address bits and must be defined in **Maestro** (hard-coded). Each predefined AIF is assigned a unique 3-bit AIF ID.
- At least one AIF_m with a matching interleave factor N_m must exist for each defined MIG.
- DMI are mapped into Memory Interleave Groups (MIG). Each MIG must enumerate N members, where N is the interleaving factor. MIG are defined in **Maestro**.
- Ncore supports up to 8 MIG – an active AIF may be set for each MIG by setting the associated field in the MIGAIFR
- Multiple MIG may be combined to form a Memory Interleave Group Set (MIGS), each set may only combine MIG in a way so that no DCE appears more than once in the merged list
- Multiple MIGS may be defined in **Maestro**, one of them must be selected at system start by configuring the AMIGRs in each DCE and AIU

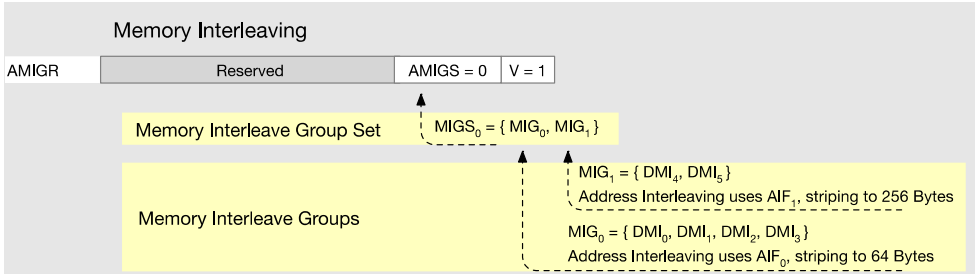


FIGURE 5-6: MEMORY INTERLEAVING GROUP REGISTER

5.4.4 Distributed Coherency Engine Selection

DCE selection is additionally constrained as follows:

- The number of instances may be any integer within 1 to 8
- If the number of instances equals a power-of-two integer, the number of primary selection bits must equal the binary logarithm ($\log_2$) of the number of instances
- If the number of instances does not equal a power-of-two integer, the number of primary selection bits must equal an integer greater than the binary logarithm of the number of instances
- The primary selection bits must be chosen from the address bits common to all snoop filters and above the cacheline offset
- If the number of ports is not equal to a power-of-two integer, the distribution of snoop filter cacheline addresses is non-uniform. For example, if the DCE is configured with three instances, the minimum number of primary selection bits equals two. In this configuration, 50% of cacheline addresses maps to one instance, and 25% maps to each of the other two instances. The non-uniformity may be smoothed by increasing the number of primary selection bits. By specifying four primary selection bits, 37.5% of cacheline addresses map to one instance, and 31.25% of cacheline addresses maps to each of the other two instances. By making the distribution of cacheline addresses more uniform, hot-spots are less likely to develop at an DCE instance.

5.4.5 Address Space Compaction

Through N-way interleaving, each memory region will be spread to N DMI modules – each memory request will arrive at DMI_x with an address relative to the base of the region and needs to be relocated to remove gaps in the physical memory address space:

- Each DMI provides a configurable memory remapping function to support address space compaction
- The number of descriptors to describe compactable blocks is defined at configuration time in **Maestro**
- Each descriptor has 3 programmable registers
    - ATER - Address Translation Enable Register

- RFAR - Relocation From Address Register
- RTAR - Reolocation Target Register

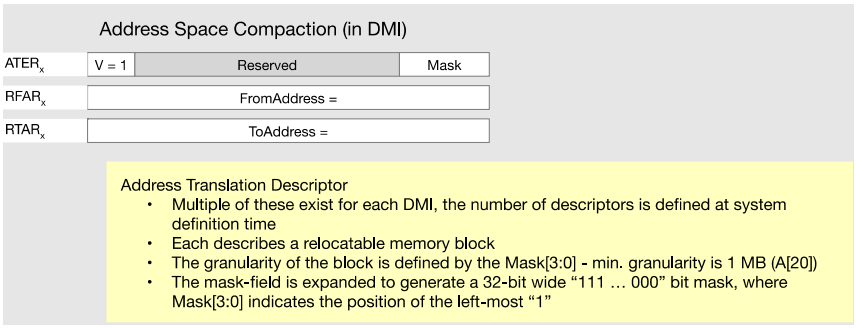


FIGURE 5-7: ADDRESS SPACE COMPACTION

The incoming physical memory address will be masked off by a 32-bit mask and compared to the RFAR (relocation "from" address register) in all valid ATR (Address Translation Descriptor)

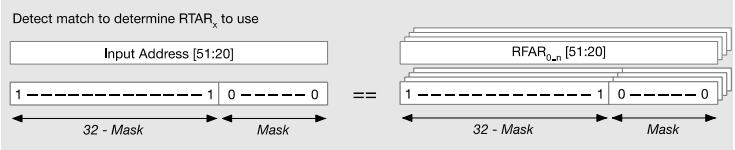


FIGURE 5-8: MATCHING INPUT ADDRESS TO ATR DESCRIPTOR

If a descriptor matches, the masked value of the RTAR (relocation target address register) will replace the previously masked MSBs in the original address

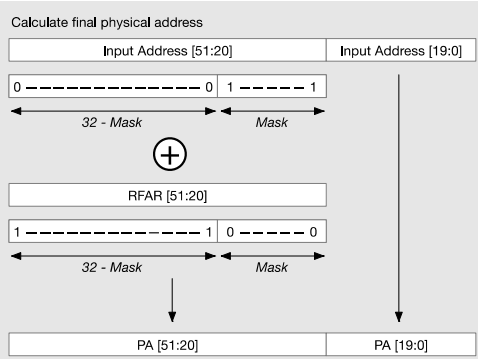


FIGURE 5-9: RELOCATION TO NEW BASE ADDRESS

5.4.6 Definition of a Memory Address Map

As previously described, General Purpose Address Space is partitioned into regions. Regions are defined by region descriptors in GPRegister space.

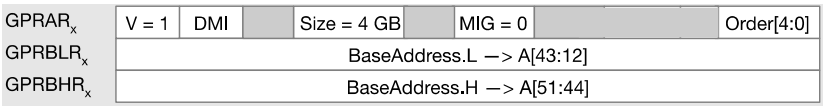


FIGURE 5-10: LAYOUT OF GP ADDRESS SPACE DESCRIPTOR

Agent interfaces for Ncore 3 implement address space decoding, transactions will be directed to a target for completion based on the aperture within the global address space.

Each aperture is described by a set of 3 General Purpose Region Definition Registers, two registers are required to define the base address of a region (GPRBLR = AddrLow, GPRBHRx = AddrHigh). The base address of any aperture is aligned to 4096 bytes.

TABLE 5-2: APERTURE BASE REGISTERS

Bit	Name	Description	Access	Reset
31:0	AddrLow	Low order bits 43:12 of a region's base address	R/W	0x00000000

Bit	Name	Description	Access	Reset
7:0	AddrHigh	High order bits 51:44 of a region's base address	R/W	0x00
31:8	Rsvd	Reserved	RO	0x000000

The third register, GPRARx, defines the region attributes. The number of apertures is configurable by the user, a special (default) boot region exists and uses the same register format. A similar set of registers exists in DCE.

TABLE 5-3: APERTURE ATTRIBUTE REGISTER

Bit	Name	Description	Access	Reset
0	Rsvd	Reserved This bit had been defined as the ***Hazard*** flag in the PCIe document and was no longer required	RO	0x0

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th><th>Access</th><th>Reset</th></tr>
<tr><td>1</td><td>ReadID (ARID)</td><td>No ordering by AxID (free listing) for reads[36]</td><td>RW</td><td>0x0</td></tr>
<tr><td>2</td><td>WriteID (AWID)</td><td>No ordering by AxID (free listing) for writes[15]</td><td>RW</td><td>0x0</td></tr>
<tr><td>4:3</td><td>Policy[1:0][37]
<table><tr><td>11</td><td>Endpoint Order</td></tr><tr><td>10</td><td>Relaxed Order</td></tr><tr><td>00</td><td>Reserved</td></tr><tr><td>01</td><td>Write Order</td></tr></table></td><td>Policy determines the setting of the internal OR[1:0] field which defines the behavior of the agent at the bottom of Ncore
Endpoint[38] order:
▪ All transactions to the same peripheral region are issued and completed in order
▪ Override the ordering mode of the incoming transaction to Endpoint Order
Relaxed order[39]:
▪ Responses within a flow (same AxID) are ordered if ReadID/WriteID bits are zero, otherwise they are unordered
▪ Ordered transactions will be issued as soon as the previous transaction in the ordered sequence has been ordered at the target (StrReq has been returned) and command credits are available
▪ Unordered transactions will be issued as soon as command credits are available at the target
Write order:
▪ This mode has been added to allow writes to follow PCI rules (A2a, C2a) at the DII
Reserved:
▪ Shall be treated as Endpoint order (the same behavior as Policy = 11)</td><td>RW</td><td>0x0</td></tr>
<tr><td>5</td><td>NC[40]</td><td>Non-Coherent - this bit applies only for AXI transactions, ignore for CHI/ACE/ACE-Lite and use transaction property NC:</td><td>RW</td><td>0</td></tr>
</table>

[36] Setting ReadID and WriteID (freelisting) will force the internal OR[1:0] field to 2b00 (unordered) for memory targets

[37] ReadID/WriteID/Policy: These fields are not applicable to CHI or ACE transactions and shall be reserved in CAIU

[38] Read: Issue --> DtrReq, Response --> DtrReq --- Write: Issue --> DtwRsp, Response --> DtwRsp. Setting ReadID or WriteID bits is not recommended for this policy and may result in undefined behavior

[39] The equivalent of ***Strict Response*** ordering can be achieved by using ***Relaxed Order*** and setting the ReadID and WriteID bits

[40] Each aperture shall be configurable to either use of coherent or non-coherent transactions Coherency does not apply to AUI target = DII and the value of the bit shall be ignored for this setting

Bit	Name	Description	Access	Reset
		0: Use coherent mode as indicated by incoming transaction 1: Enforce non-coherent transaction		
7:6	NSX[1:0]	This field describes the 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 later use)	RW	0x0
8	Rsvd	Reserved	RO	0x0
13:9	HUI	Home Unit Identifier used to access this memory regions - this field uses the index of the target HU within the enumerated set of DMI or DII	RW	0x0
19:14	Rsvd	Reserved	RO	0x0
24:20	Size	This field describes the size of the aperture in address space. The size of a region is calculated as: size_of(IG) * $2^{(Size+12)}$ with IG = Interleave group	RW	
28:25	Rsvd	Reserved	RO	0x0
29	Rsvd	Reserved - this bit has been reserved to be used as an extension to HUT when hierarchical gateways or chip-to-chip interfaces will be implemented	RO	0x0
30	HUT	This field describes the Home Unit Type 0: System Memory 1: I/O (Peripheral Memory/Devices)	RW	0x0
31	Valid	This bit indicates if an aperture descriptor is valid 0: Invalid Mapping - this descriptor is not describing a valid region 1: Valid region descriptor	RW	0x0

ReadID/WriteID/Policy: These fields are not applicable to CHI or ACE transactions and shall be reserved in CAIU

NC - bit: Each aperture shall be configurable to either use of coherent or non-coherent transactions Coherency does not apply to AUI target = DII and the value of the bit shall be ignored for this setting

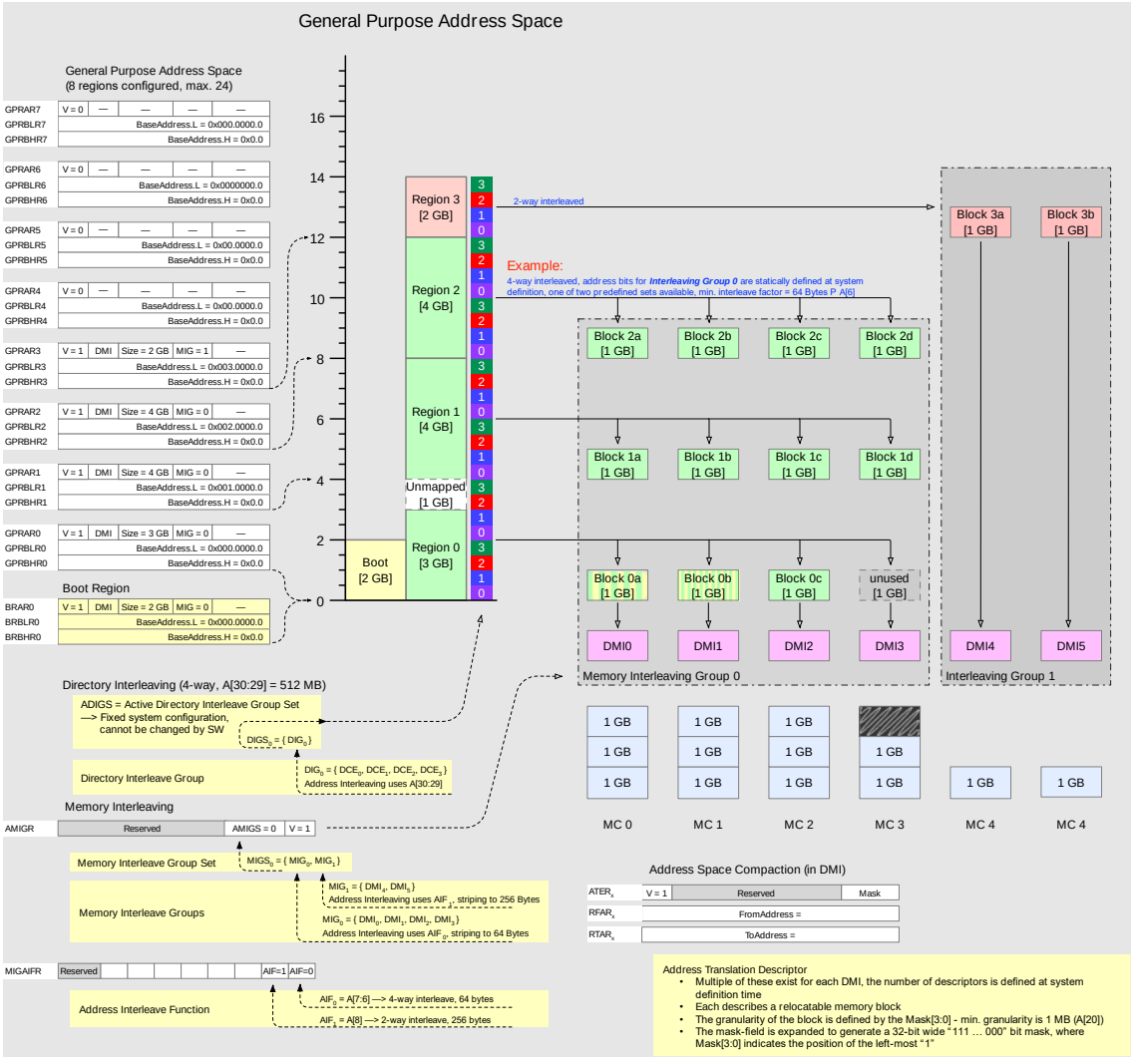


FIGURE 5-11: EXAMPLE ADDRESS MAP

6 System messages

This section defines a new service, system messages. The purpose of system messages is to convey operating state information or state transitions for agents or asynchronous events within the endpoints of an Ncore domain. In the future they will be used to implement new functionality in the general class of system related activities.

This service will be implemented by a new message class, **System**. System messages will be sent from a source agent to one or more destination agents. Every system message must receive a response to acknowledge receipt by the destination agent - depending on the protocol implemented by a specific message pair, receipt of response may initiate additional activity (for example a state transition at the source agent).

6.1 SYSreq and SYSrsp Messages

SysReq message is an implementation of the new system message class. The purpose of these messages is to propagate system related information between agents or functional units within the system.

SysReq messages are asynchronous, do not use credit management as there shall be only one outstanding transaction per target, it shall be acknowledged by the receiver returning the SysRsp message. The SysRsp response implements status and error reporting in the CMStatus:

Command error[41] :

- command was not properly formatted
- No support for this operation
- Wrong unit - this unit is not supporting the OpCode or unknown OpCode
- Unit is busy and cannot perform the requested operation at this time
- Ok - no error occurred, command completed as expected

SysReq OpCode:

- SysReq command uses CMType = 0x7B[42]
- SysRsp response uses CMType = 0xFB[43]

---

[41] Some or all of these will not be supported in the initial release Ncore 3.2

[42] This opcode is available - between 0x7A (STRreq) and 0x7C (RBRreq)

[43] One of the few free opcodes within the response block (0xF0 - 0xFF)

TABLE 6-1: SYSREQ COMMAND LAYOUT

Name of Field	Width Param	Param Values	Description	Direction	Source
CMH:			**CCMP Message Header**		
TargetId	wTargetId	6 - 12	Target Identifier	In/Out	
InitiatorId	wInitiatorId	6 - 12	Initiator Identifier	In/Out	
CMType	wCMType	8	Type of Message - SysReq Opcode	In	
MessageId	wMessageId	6 - 12	MessageId	In/Out	
HProtection	wHProt	Derived	Protection for the Concerto Message Header fields	In/Out	
CMHE:			**CCMP Message Header Extension**		
TTier	wTTier	0 - 4	Traffic Tier of the message for use by the transport fabric	In/Out	
Steering	wSteering	0 - 4	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message	In/Out	
Priority	wPriority	0 - 4	Priority of the message to be used by the transport fabric	In/Out	
QL	wQL	0 - 4	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronicity property, etc. to be applied for this message	In/Out	
CMB:			**CCMP Message Body**		
RMessageID	wRMessageID	6 - 12	MessageID for the command associated with this response	In/Out	
CMStatus	wCmStatus	8	Status - using standard command status layout	In/Out	
SysReqOp	wSysReqOp	4	As of now - only 16 different operations for this command class	In/Out	
NDProt	wSysReqProt	0 - 10	Protection bits for SysReq Message	In/Out	

TABLE6-2: SYSRSP RESPONSE LAYOUT

Name of Field	**Width Param**	**Param Values**	**Description**	**Direction**	**Source**
CMH:			**CCMP Message Header**		
TargetId	wTargetId	6 - 12	Target Identifier	In/Out	
InitiatorId	wInitiatorId	6 - 12	Initiator Identifier	In/Out	
CMType	wCMType	8	Type of Message - SysReq Opcode	In	
MessageId	wMessageId	6 - 12	MessageId	In/Out	
HProtection	wHProt	Derived	Protection for the Concerto Message Header fields	In/Out	
CMHE:			**CCMP Message Header Extension**	**In/Out**	
TTier	wTTier	0 - 4	Traffic Tier of the message for use by the transport fabric	In/Out	
Steering	wSteering	0 - 4	Steering. This field is used to indicate steering of the message the transport fabric should apply to this message	In/Out	
Priority	wPriority	0 - 4	Priority of the message to be used by the transport fabric	In/Out	
QL	wQL	0 - 4	QoS Label. This field is used by the transport fabric for QoS functions such as bandwidth provisioning, isochronic- ity property, etc. to be applied for this message	In/Out	
CMB:			**CCMP Message Body**		
RMessageID	wRMessageID	6 - 12	MessageID for the command associated with this response	In/Out	
CMStatus	wCmStatus	8	Status - using standard command status layout	In	
NDProt	wSysRspProt	0 - 10	Protection bits for SysRsp Message	In/Out	

6.1.1 SysReq Commands

Being not credited imposes restrictions on the use of SysReq messages. It requires that a functional unit must always be able to process a single SysReq command and generate a meaningful response using SysRsp. In the first implementation in these message operations will be supported:

TABLE 6-3: SYSREQ COMMAND OPERATION PAYLOAD

Command	OpCode	Payload	Function
SysReq.NOP	0x00	TimeStamp	No function, beyond carrying the timestamp in both directions. This transaction may be used to measure the roundtrip latency between two agents. If an agent does not implement a timestamp counter, the value used in the response shall be -1 (all ones)
SysReq.Attach	0x01	TimeStamp SysCoReq	Attach the requester to the coherency domain. This command is used by a cache coherent initiator (CAIU) to attach itself to the coherency engine (DCE). The command needs to be sent to all interleaved DCE. The OK response signals successful attachment, only after that the agent will be allowed to send coherent requests and participate in snoop protocol
SysReq.Detach	0x02	TimeStamp SysCoReq	Detach the requester from the coherency domain. This command removes the agent from the set of snooped coherent agents managed by DCE
SysReq.Event	0x03	TimeStamp EventReq	System Event - these events are generated within various agents[44] in the system and convey asynchronous events between event producers and event-consumers . These agents are CPUs, monitors for exclusive transactions, SMMU, Accelerators or other agents. Agents may be producer, consumers or both.

6.1.2 Command CMStatus

The CMStatus[7:0] field is shown only as input in the direction column, implying that the field is not transmitted along with the rest of the *SysReq* message at the initiator, but is an input-only field at the target of the message.

The CMStatus[ ] field is used to report the result of a processing step of the operation, including detection of errors, to an Ncore unit. However, *SysReq* being the start of the operation, there are no results to report.

If any type of error is detected in the native request transaction, the initiator AIU must not issue a corresponding *SysReq* message into the Ncore system to commence the processing of the native operation. The operation must be locally terminated and error reported back to the native agent.

A *SysReq* message may encounter a transport error along its path to the target. This error must be reported to the target via the CMStatus[ ] field constructed at the transport interface of the target Ncore unit.

TABLE 6-4: SYSCMD AND SYSRSP COMMAND STATUS FIELD ENCODING

Status Type	CMStatus [7:6]	CMStatus [5:0]

[44] In Ncore 3 only DCE (exclusive monitors) and AIU (EventInReq) will issue Event messages

<table>
<tr><td rowspan="3">Success
(No error)</td><td rowspan="3">2b00</td><td colspan="2">SysRsp:
System Response carries back the status of the command execution at the target agent</td></tr>
<tr><td>[5:3] = 3b000</td><td>[2:0]:
xx0: No Operation performed - this may not be an error!
001: Completion - Execution state machine is still running
011: Final - Execution state machine went to IDLE after sending this response
101/111: reserved (TBD)</td></tr>
<tr><td colspan="2">SysReq:
{Reserved} - there should be no error on the incoming command, except transport problems</td></tr>
<tr><td rowspan="2">Error</td><td rowspan="2">2b01</td><td colspan="2">SysRsp:
System Response carries back the status of the command execution at the target agent</td></tr>
<tr><td>[5:3] = 3b000</td><td>[2:0]:
xx0: No Operation performed - unit signals error
001: Completion - with unknown error (conditions TBD)
011 - 110: Format Error - reserved
111: Unit is busy and cannot perform operation at this time</td></tr>
</table>

Transport related error status uses the same representation as CmdRsp.

6.2 SysCoReq Implementation

The SysCo protocol is a 4 state handshake protocol implemented by ARM processor DSU. The purpose of the protocol is to attach and detach the agent to the coherent domain.

See ARM document *dsu_trm_100453_0401_03_en.pdf,* section A4.9 Cluster powerdown for a detailed description of the Initiator Agent handshake protocol.

- On power up
  - Adds a request agent (DSU-CAIU) to the coherency domain
  - After de-asserting RESET:
    - SYSCOACK is inactive
    - Agent will respond to snoop and DVM requests but will not yet make cacheable requests
      - Agent will try to CONNECT by asserting SYSCOREQ signal
    - Interconnect is ready to receive cacheable requests
      - Interconnect shall assert SYSCOACK
- On power down
  - Remove a request agent (DSU-CAIU) from the coherency domain
  - At this time SYSCOREQ and SYSCOACK are asserted
  - Flushing caches and quiescing traffic:
    - Agent halting execution (WFI/WFE/HALT)
    - Agent de-asserts SYSCOREQ
    - Agent shall continue to respond to snoop and DVM requests but may no longer make cacheable requests
    - The interconnect has no longer active transactions for this agent and the system directory's snoop filters have not any cache lines recorded
    - De-assert SYSCOACK
    - Shut down the agent (assert RESET, power-down)

6.2.1 System Architecture

In support of SysCo (Attach/Detach) each coherent agent (CAIU/ProxyCache[45]) shall implement a SysCo-engine.

In support of SysEvent each agent with event sources or receivers shall implement a SysEvent-engine.

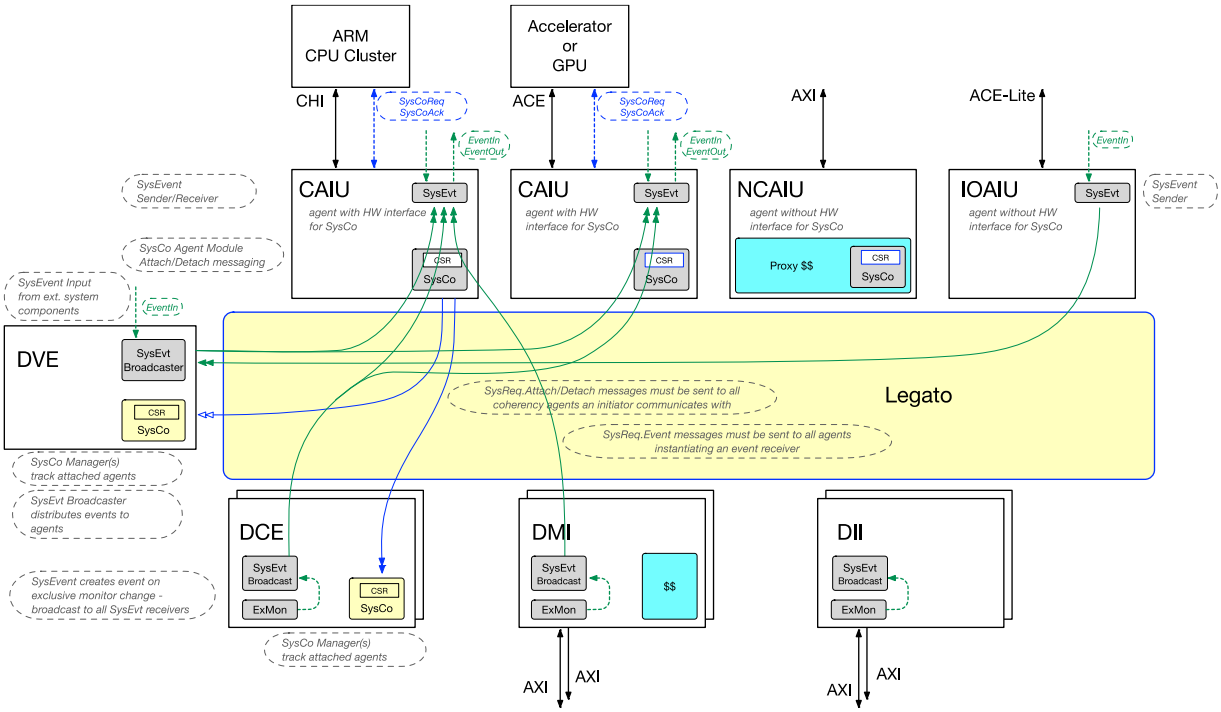


FIGURE 6-1: SYSCO TOP LEVEL ARCHITECTURE[46]

After reset, an agent will not be part of a coherency domain, it may only send non-coherent transactions and will not respond to snoop requests.

Ncore will use SysReq messaging to notify all coherency managers (DCE) when an agent requires to be attached or detached from the coherency protocol.

6.2.2 Attach/Detach Protocol

The initial state of any CAIU is detached - any CAIU that connects to a coherency agent with SYSCOREQ/SYSCOACK (SysCo) support must implement the SysCoReq interface and the SysCo message protocol engine.

45 Any agent participating in the coherency protocol that does not support SysCoReq/Ack needs to implement a Sysco-engine using CSR protocol to attach/detach to the coherency domain

46 This high level architecture shows optional exclusive monitors and SysEvent-engines in DMI/DII, these are not part of Ncore 3.2

Any other coherent initiator agent (CAIU connecting agents that do not support SysCo, NCAIU with ProxyCache) must implement the SysCo message protocol engine. The engine must provide a CSR interface to initiate the coherency attach/detach process by SW (write to register).

A coherency agent must not send any coherency transactions (Snoop/DVMSnoop) to a functional unit, unless that unit has been attached to this coherency agent by using the SysCo protocol.

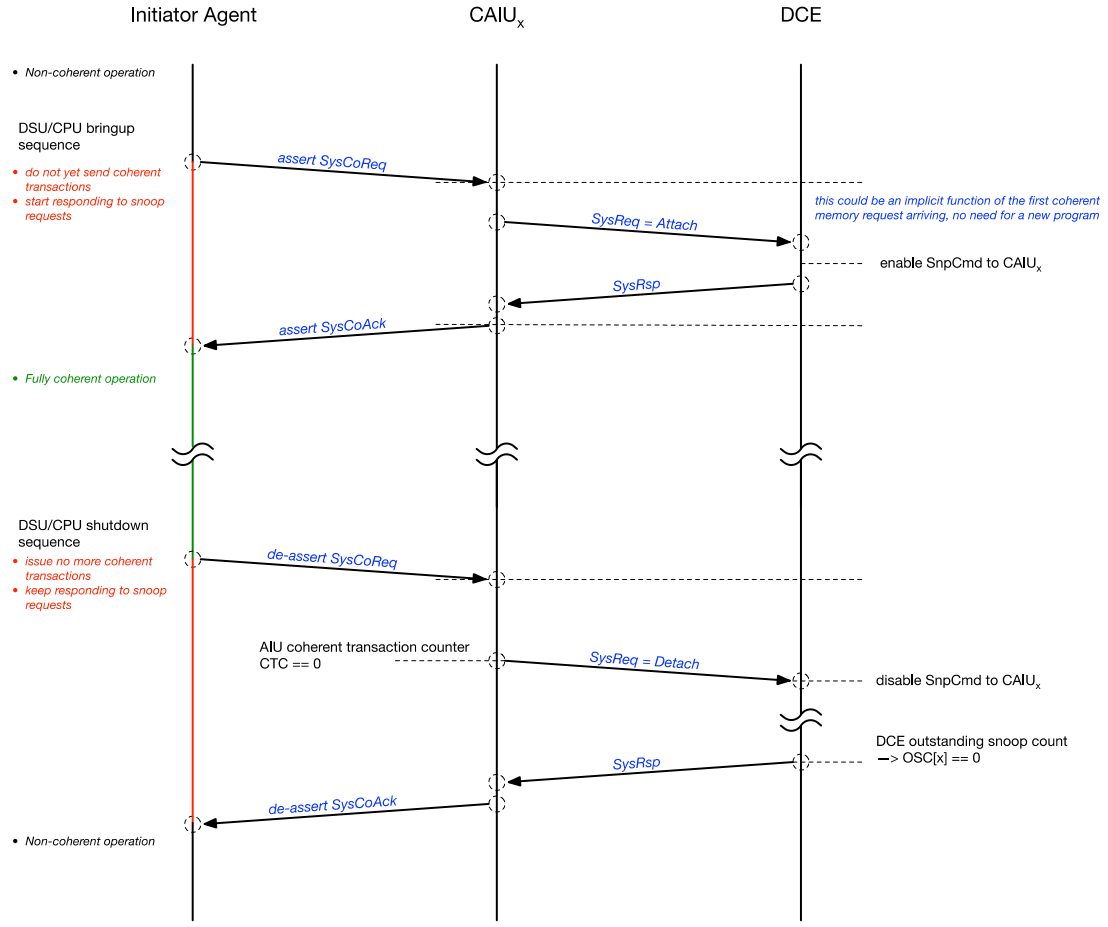


FIGURE 6-2: SYSCO FLOW

Figure 6-2 shows the details of the SysCo protocol and the associated exchange of messages. For clarity, only messaging with a single DCE is shown in the flow chart - in a real system a CAIU needs to attach to all DCE and DVE covering the coherency domain by sending messages to each one. The CAIU may start using coherent transactions only after all SysRsp have been received.

While in *detached* state, CAIU must not send coherent transactions, it is permitted (but not required) to respond to snoop requests.

CAIU must keep track of outstanding requests and keep a running count of outstanding coherent transactions. When a transition from attached to *detached* state has been requested, either by HW SysCo protocol or by a CSR write request, it must stop making coherent transaction into the system

(ARM DSU will no longer make coherent request after SysCoReq has been deasserted) and wait until all outstanding coherent transactions have been retired, before making a SysReq.Detach to DCEs.[47]

A coherency agent (DCE/DVE) shall no longer snoop an agent after receiving SysReq.Detach from it[48]

A coherency agent must keep track of outstanding requests and keep a running count of outstanding coherent transactions to an agent, SysReq.Detach shall only be acknowledged by a coherency agent after all outstanding coherent transactions for a snooped agent have been completed.[49]

6.2.3 SysCo Engine at Initiator Agent

The SysCo engine manages the attach/detach process for each coherent initiator agent.

For agent interfaces with HW protocol support 2 signals (SysCoReq = please attach me, SysCoAck = you are part of the coherency domain) will be provided. These signals may be asynchronous to the internal clock domain and shall be properly synchronized using proper CDC implementation.

For agent interfaces without HW protocol support, the CSR interface may be used to request attachment or detachment from the coherency system.

Internally, each SysCo Engine shall implement (also see Figure 6-3):

- CSR with 4 bits
  - C: Connecting, read-only
    - On read returns '1' when the agent is waiting to connect (FSM.state == ***Connect***)
  - Req: Activate, read-write
    - Write to '1' to trigger FSM.state transition from ***Idle*** to ***Connect***
    - Write to '0' to trigger FSM.state transition from ***Connect*** to ***Detach***
    - Cleared by reset
  - A: Attached, read-only
    - On read returns the current state of the interface (0 - detached, 1 - attached)
  - Err: Error, read, WZ (write-to-zero)
    - read as '1' when a protocol error or time-out has been signaled
    - read as '0' after reset - or after no protocol error has been detected in a transaction
    - write to zero - clears the bit
- SysCoFSM with 6 states:
  - ***Idle*** – after reset and when disconnected - *SysCoAck* shall be de-asserted in this state
    - On SysCoReq assertion or CSR.Req set, transition to ***Connect***

47 This requirement guarantees termination of concurrent traffic at the source and, assuming that software initiated a proper shutdown of local caches, also maintains consistent memory state

48 This guarantees termination of snoop traffic initiated by other agents

49 Only after all coherent traffic has completed, the agent may be transitioning to detached state and acknowledge the transion to the DSU

- ***Connect*** – request to connect by sending a *SysReq.Attach* message to all coherency agents this agent communicates with and wait for acknowledgement by protocol, a time-out counter shall be implemented to report error on no response.
  - Transition to ***Attached*** when *SysRsp.Ok* received
  - Transition to ***Attach-Error*** when *SysRsp.Error* or time-out
- ***Attach-Error*** - transient state, sets CSR.Err and transitions to ***Detach***, reach a safe state
- ***Attached***– connected, normal operating state - *SysCoAck* shall be asserted in this state
  - On *SysCoReq* de-assertion or *CSR.Req* cleared, transition to ***Detach***
- ***Detach*** – requesting to disconnect by sending a *SysReq.Detach* message to all coherency agents this agent communicates with and wait for acknowledgement by protocol, a time-out counter shall be implemented to report error on no response.
  - Transition to ***Idle*** when *SysRsp.Ok* received
  - Transition to ***Detach-Error*** when *SysRsp.Error* or time-out
- ***Detach-Error*** - transient state, sets CSR.Err and transitions to ***Idle***, reach a safe state

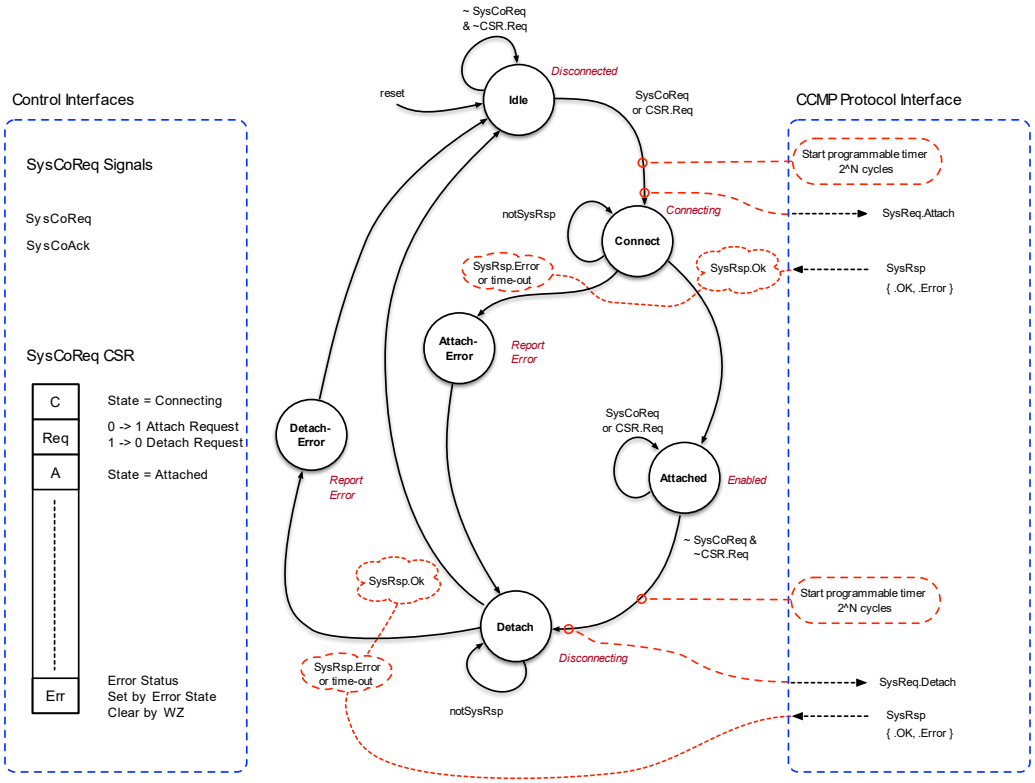


FIGURE 6-3: SYSCO STATE MACHINE

6.2.4 SysCo Engine at Coherency Agent

The SysCo engine manages the attach/detach process for each coherency agent. Coherency agents are functional units which issue Data Snoop or DVM Snoop requests. These agents need to know if message target is active and will respond to a request. Snooping an inactive agent would break the protocol and, even though it could be detected by using the currently implemented time-out mechanism, the performance impact of an unreliable transaction is not acceptable to customers.

The coherency agent's SysCo engine manages the participation of initiator agents in the snoop protocol by preventing snoops being sent to detached agents.

Agents that want to participate in the coherency protocol must register by sending a SysReq.Attach command to any coherency agent (DCE/DVM) they expect to be communicating with.

Agents that transition to sleep or any other inactive state and can no longer participate in the coherency protocol by responding to snoops, must detach from the coherency domain by sending a SysReq.Detach command to all coherency agents within said domain.

The coherency agent will track all outstanding transactions to agents and will respond by acknowledging the detach request as soon as no more active transactions exist for that agent (all outstanding snoop responses have been received).

Internally, each coherency agent's SysCo engine shall implement:

- At least one CSR with (at least) one bit T[i] to tracked each agent's state, where i is the unique ordinal index of the initiator agent in the ordered set of initiators as assigned by Maestro
    - T[i] == 0 indicates $agent_i$ is not actively participating in the coherency protocol and no transaction with that agent is allowed.
    - T[i] == 1 indicates the $agent_i$ is active and participating in the coherency protocol, snoop or DVM transactions may be issued.
    - For DCE:
        - The coherency protocol allows cache lines to be silently evicted from any agent's cache. As a consequence, the tracking snoop filters implemented in DCE may not correctly reflect the agent's cache state and falsely indicate presence of a line.
        - It is required that agents that are no longer meant to participate in the coherency protocol go through a well defined sequence of steps:
            i. Clean and invalidate the contents of the device's cache
            ii. Initiate transition out of the coherency domain, by following the HW handshake provided for Arm DSU or by SW procol initiating the transition using the provided CSR interface
            iii. Entering sleep or power down state
        - After steps (i) and (ii), it can be safely assumed that the agent does no longer hold valid data and any presence in the snoop filter is a result of silent evictions.
        - If a presence is indicated by a snoop filter and the agent is in detached state (T[i] == 0), no snoop to that agent shall be sent, the snoop filter entry for that

agent shall be updated to remove the cache line and a spurious snoop filter event shall be generated to the performance counters
    - The set of CSR holding the T[i] bits shall provide read-only access to retrieve the current state (attached or detached)
    - An engineering register within CSR space shall be provided - writes to this register shall modify the vector T[i]
    - For DVE:
        - No DVMSnoop shall be sent to a detached agent

- SysCoMgr to receive SysReq messages from initator agents:
    - On receipt of SysReq.Attach:
        - Map the SourceID of the arriving SysReq to the associated T[i] and set the bit
        - Issue a SysRsp.Ok if the CMStatus of SysReq is OK
        - Issue a SysRSP.Error if
            i. CMStatus or SysReq indicates an error
            ii. target does not exist within the T-vector
            iii. target is already set - we should not see duplicate attempts to set/clear a bit, this would be considered a protocol violation!
    - On receipt of SysReq.Detach:
        - Map the SourceID of the arriving SysReq to the associated T[i] and clear the bit
        - Issue a SysRsp.Ok if the CMStatus of SysReq is OK
        - Issue a SysRSP.Error if
            i. CMStatus or SysReq indicates an error
            ii. Target does not exist within the T-vector
            iii. Target is already clear - we should not see duplicate attempts to set/clear a bit, this would be considered a protocol violation!

SysReq messages are not credited - special care must be taken to avoid loss of messages and guarantee proper accounting through SysRsp. Initiating agents shall generate no more than one outstanding message per each coherency agent at a time. This limits the total number of transactions arriving at a coherency agent.

It is an implementation choice to use a queue on the receiver side and execute SysReq only when it has the resources to respond with SysRsp.

6.3 Event Propagation

Events are asynchronous messages and will be used by agents to notify other agents in a system that an action shall be taken. Other than interrupts, events do not change the flow of program execution. Agents may be waiting on events, entering a lower-power state while doing so.

In the ARM ecosystem events may be used to quickly synchronize execution between agents by explicitly waiting for (WFE instruction) and sending events (SEI instruction). Other sources of events may be defined, for example, exclusive monitors changing state or SMMU service requests completed.

Events generated at a source within an agent will enter Ncore through the *EventInputInterface* and will be forwarded to DVE for distribution. DVE will broadcast event messages to all active[50] agent interfaces (CAIU) with an Event Receiver in a system. The Event Receiver's *EventOutputInterface* will deliver events to the connected target (e.g. a processor cluster (DSU) or accelerator).

Event interfaces use a 4-phase handshake protocol implemented with 2 signals: EventReq, EventAck.

Events are not cummulative, multiple events arriving at an agent interface will only generate a single event handshake on the interface. Event messages arriving while a handshake is actively taking place will be registered and will initiate another handshake after the previous has completed.

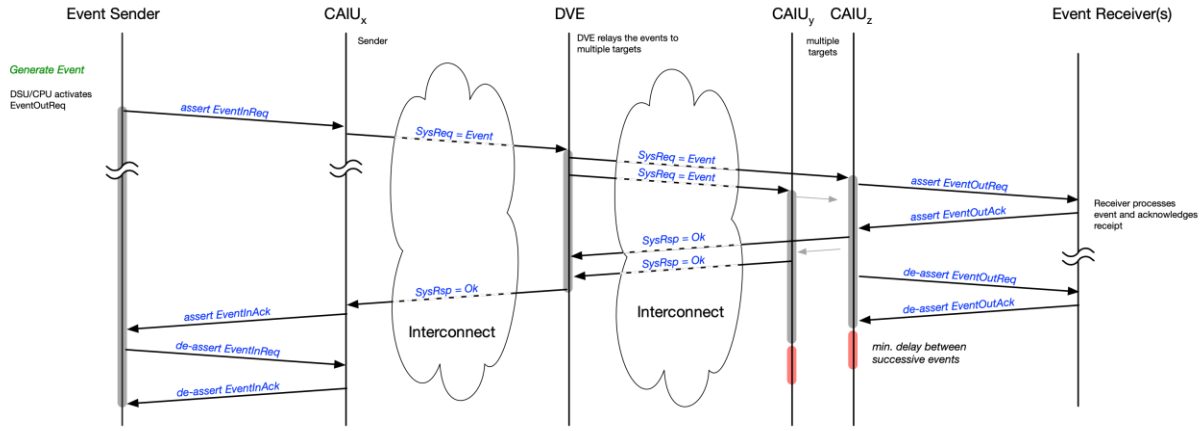


FIGURE 6-4: EVENT MESSAGING

Requirements:

- Each AIU shall support a bi-directional event interface
- The event interface may receive events from the attached agent following the REQ/ACK-protocol

[50] An interface is considered active if it has registered as participating in coherency traffic by SysCoReq protocol - note, at this point this may be restrictive and we may consider introducing a separate activation protocol in the future

- The event interface shall forward events, generated internally or received from another agent, to the attached agent
- A received/generated event will be sent to DVE
- Events will be transported within Ncore using SysMsg transactions
- Event messages will be broadcast to all active AIU by DVE
- Events may originate within Ncore (when an exclusive monitor is cleared) - these events will be broadcast by the agent hosting the exclusive monitor
- Events entering Ncore at any AIU shall be forwarded to all AIUs

6.3.1 Event Messaging Architecture

Please refer to Figure 6-1 for the system overview.

The event architecture knows different agents:

Source = any system function that generates events, either as a consequence of state changes or by program execution (e.g. instructions SEV (send event))

- Processors executing SEV
- Page table activity, misses/errors/service requests in SMMU
- Accelerators, finishing tasks or computations
- Global exclusive monitors (DCE/DMI/DII) changing state

Emitter - anything that generates event signals connecting to event receivers. Event signals use a 4-phase handshake protocol (request-pending-acknowledge-complete) based on Req/Ack to simplify clock domain crossing:

- DSU - may provide an interface to issue events to the AIU it is connecting to
- AIUs - shall provide an interface to send events to an attached DSU or accelerator

TABLE 6-5: EVENT MESSAGING INTERFACES - SOURCES AND DESTINATIONS

Component	Type	Signal	Present	Function
C-AIU CHI	Source Emitter	EventInReq, EventInAck EventOutReq, EventOutAck	yes yes	CPU cluster sending events to system CPU cluster receiving events from NoC
C-AIU ACE	Source Emitter	EventInReq, EventInAck EventOutReq, EventOutAck	yes yes	CPU/accelerator cluster sending events to system CPU cluster receiving events from NoC
NCAIU	Source Emitter	EventInReq, EventInAck EventOutReq, EventOutAck	yes/opt yes/opt	peripheral I/F - with SMMU
DCE	Source	--	yes	No external interface exposed, source of events is internal
DMI	--	--	no	no internal source of events
DII	Source		not in 3.4/3.6	possible use to receive events from peripherals connected downstream

Note:
1. Optional - allow configurability within Maestro to populate individual peripheral interfaces. NCAIU with ACE-Lite E, connecting to an SMMU shall implement the interface because SMMU is capable of emitting events

Sender - Block within an agent interface that implements event inputs (EventInReq, EventInAck). A sender will convert the arriving event (assertion of EventInReq) into SysReq.Event messages, broadcast those to all known (active?) receivers and acknowledge the event using the above defined 4-phase

handshake. The sender will collect the responses from all receivers and shall report an error on timeout (not receiving Ok responses to all sent messages) or if any response indicates an error.

Receiver - Block within an agent interface that implements event outputs (EventOutReq, EventOutAck). A receiver will convert arriving SysReq.Event messages into event signals, asserting EventOurReq. The receiver shall respond back to the initiator by returning SysRsp.Ok.

Each agent participating in the Event protocol shall implement a CSR to provide:

- Timeout value (implementation dependent)
- Status Bit(s) to indicate Error or detailed Error Status (implementation dependent)
- Enable Bit to enable send protocol - Even when disabled, the sender shall acknowledge event requests on the 2-wire interface
- Enable Bit to enable receiving - Even when disabled, the Receiver shall properly terminate all received SysReq.Event messages by responding with SysRsp.Ok

A first (protocol) timeout value shall be provided by a CSR, the timeout shall be programmable to provide at least 4096 clock cycles of timeout - it may be sufficient to only allow powers of two - this timeout will be used to detect fatal protocol or initialization errors.

A second (handshake) timeout value shall be provided by a CSR, the timeout shall be programmable to provide a smaller range (64 to 256 clock cycles) - this timeout will be used to detect a non-responding target of events.

6.3.2 Event Sender

The sender state-machine will be idle after reset. When the EventInReq is asserted by the source, the state machine will enter the Send state and start sending SysReq.Event messages to all receivers in the system. Maestro shall provide a vector, listing all receivers.

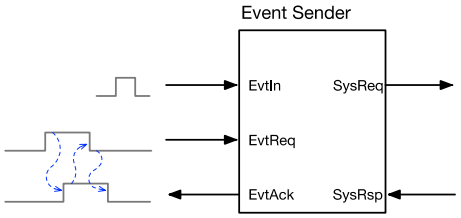


FIGURE 6-5: EVENT SENDER - INTERFACES

Event messages, like all SysReq transactions, are not credited and a sender must not send more than one transaction to each target agent.

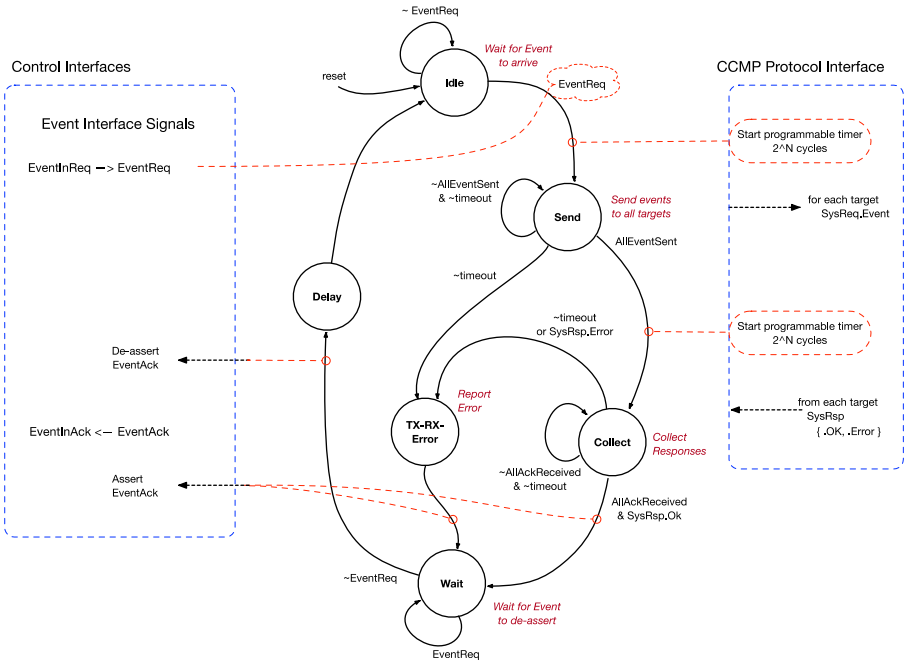


FIGURE 6-6: EVENT SENDER FSM

After sending out all messages, the sender shall provide means to verify that all messages receive responses - as events will be handled one-at-a-time and no more than one message will be sent to each agent, counting the number of responses is sufficient.

An error is considered:

- not all outbound transactions receive a response within the timeout period

- one or more SysRsp return an error status - the status shall reflect accumulated error from all received responses (most severe error within CMSTATUS -- List to be defined)

6.3.3 Event Receiver

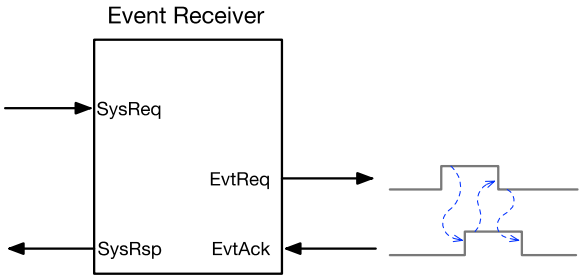


FIGURE 6-7: EVENT RECEIVER INTERFACE

Whenever a SysReq.Event message arrives, it will be recorded within the input queue. The queue shall provide one dedicated storage location for each source of events. Possible sources of events are: CAIU, NCAIU, DCE, DMI, DII etc.

Events are indistinguishable from each other and may be aggregated - all arriving messages within a certain time period, for example while the interface is occupied with a previous event, may be combined into a single event.

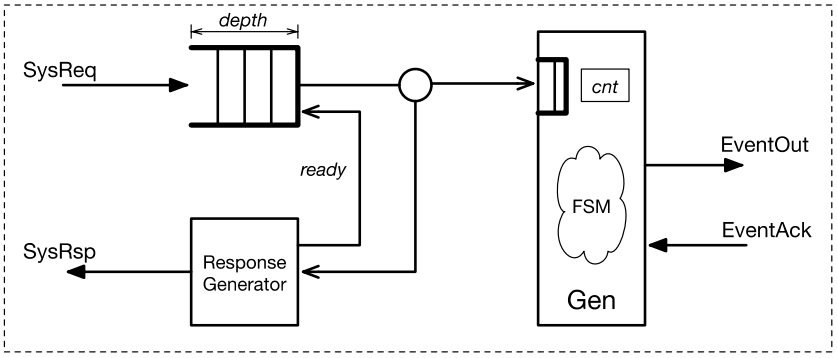


FIGURE 6-8: EVENT RECEIVER ARCHITECTURE

The output of the queue feeds into the event generator and the response generator - every arriving event must be responded to.

If the arriving message does not indicate an error status, the response (order of severity) shall be:

- Ok - if the agent is enabled to receive events (least severe error)

- Busy - if the agent is disabled
- Error - The event generator did not receive EventAck within the timeout period (note, the timeout period for the event handshake may be hard-coded to a significantly smaller value than the protocol timeout)
- Error - The received command message indicated an error (most severe error)

The Event Receiver State Machine, shown in Figure 6-9, receives arriving event messages and converts them into the 4-phase handshake protocol. Even though the messaging protocol allows fast bursts of event messages to arrive - multiple event messages may trigger only a single event sequence through the state machine.

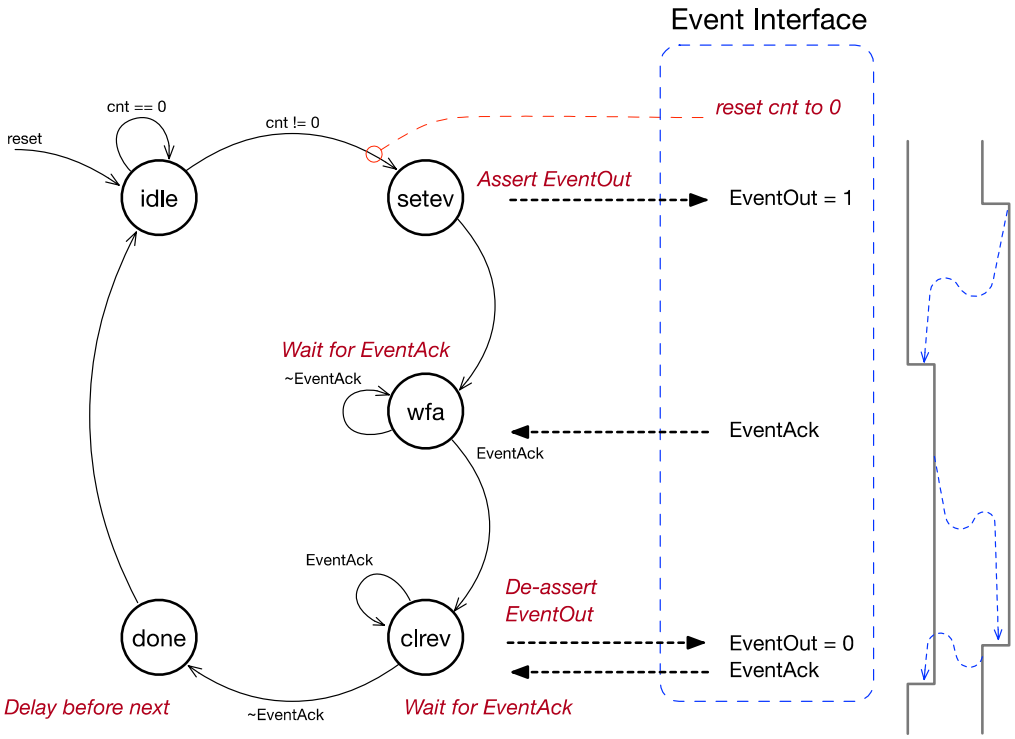


FIGURE 6-9: EVENT RECEIVER STATE MACHINE

7 Quality of service (QoS)

The purpose of QoS is to provide certain service guarantees, determined by bandwidth or latency requirements. This chapter describes Ncore 3 QoS-related features. The current implementation is based on message priority. The QoS value on native interface command is mapped to QoS bucket (priority) mapping function. The mapping is provided through predetermined design time parameters. The mapping from QoS to Priority must be identical in each Ncore 3 unit.

The transport layer (CDTI/CSTI) doesn't have any QoS related features. NCore 3 offers priority arbitration in Concerto units. This bucket value is used for arbitration at all points in the system where requests accumulate in a buffer or transaction table. Indeed, the QoS arbitration happens in AIUs, DCEs and DMIs.

The general behavior to elect a transaction in each unit respects the following criterion:

- The transaction has no unsatisfied ordering dependencies.
- There is no other candidate transaction that is under starvation prevention.
- An appropriate credit is available.
- The transaction has the highest QoS value among the candidate transactions.
- The transaction is the oldest among candidate transactions with the same QoS value.

Ncore also 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.

7.1 Fields related with QoS

CCMP uses several distinct labels for expressing the types of QoS services requested for the message. These are presented in the message header using the following fields:

Priority: This label refers to the relative preference level of the message at an arbitration station.

The 4-bit QoS value for an incoming native transaction is mapped to one of 8 QoS buckets (3-bit value priority field) by Maestro mapping. QoS value mapping is performed at native interface: AIUs/DMIs/DIIs.

TABLE 7-1: SYSTEM PARAMETERS FOR QOS CONFIGURATION

Parameter Name	Description	Type	Default	Min	Max	Enum	Notes
qosEnabled	Enable QoS support	Boolean	FALSE				
qosMap	QoS value to priority map	String					From Native QoS value (4-bit) to Priority (3-bit)
qosEventThreshold	QoS event threshold	integer	64	32	1024		

The system parameters for QoS are described in the parameters specification. Table 7-1 is here for illustration.**QoS:** In addition, **QoS** field is used to send QoS value received at native initiator to native target. The QoS field received with native transaction is copied into the QoS field of all messages issued in Ncore to process the transaction as well as all the related transactions generated on native interfaces. This is done by:

- Copying the QoS field within the transaction into the QoS field of the CMD request message generated by the AIU.
- Copying the QoS field of the message into QoS fields of all native transactions begotten by the message at its target unit.

Qos Label (QL): This label may indicate a combination of other QoS attributes. It is currently unused and tied to 0.

Caution: smi_pri (Priority inside of the network) has opposite order (decreasing order) with AxQoS (increasing order). That is, in the network smaller priority values are higher QoS. So, by default QoS 15 will map to pri 0.

7.2 Starvation Avoidance

Starvation mechanism can be configured in the design with parameters and the behavior can be controlled through CSR. It is implemented as two common state machines with three bits per transaction table entry. It has two phases:

7.3 Starvation Detection:

A global counter is used to count commands sent. Once the counter reaches a programmable threshold, an overflow bit is set in all valid entries which have not sent out the initial command and have their protocol dependencies (Address, AXI ID etc.) resolved. This indicates the start of the period where the entry must make forward progress. The Overflow bit will be cleared if the entry sends a command. If the global counter reaches the threshold again while the overflow bit is still set, then that entry is starving because it hasn't made progress for at least one complete period. If there is one or more starved entry, then the design enters starvation mode.

7.3.1 Starvation Handling:

In this mode only requests that are marked as starved can arbitrate and send commands out. Arbitration scheme used here is implementation defined, it can be age order, find first, round robin or priority based. Note that the starved transactions must be sent out with the same original QoS and priority. The global counter will be paused in this mode. If a starved request can't make forward progress due to non-availability of credits, then the design should simply wait for the credit to become available. Once all the starved requests have been serviced the normal processing resumes and the global counter resumes counting.

Software via CSR will have the option to disable the starvation logic and change the starvation threshold.

7.4 NCore 3 Unit Functionalities

Ncore units AIUs, DCEs & DMIs participate in QoS and implement following:

7.4.1 IOAIU

Transactions (initial commands) are sent out based on QoS values in age order after all other (AXID / address / read and write) dependencies are resolved.

All outgoing associated messages will have the mapped priority value on it plus the initial QoS value.

Proxy cache evictions take the value of the transaction that caused the eviction.

7.4.2 CHI_AIU

Transactions (initial commands) are sent out based on QoS values in age order after all other (LPID / address / read and write) dependencies are resolved.

All outgoing associated messages will have the mapped priority value on it plus the native QoS value.

7.4.3 DCE

Incoming commands are stored in a skid buffer and QoS in age order is used to pick commands from the buffer.

All outgoing associated messages will have the calculated priority value on it plus the native QoS value.

Recall transaction uses fixed QoS value that can be specified at design time or via CSR.

7.4.4 DVE

Incoming commands are stored in a skid buffer and age order is used to pick commands from the buffer. Unlike in other Concerto units, priority based QoS is not used in DVE skid buffer.

DVE propagates QoS and priority values from the initial command. DVE propagates the priority and QoS value of the command that caused the generated snoops. DVE propagates the priority value for other generated requests and responses.

7.4.5 DMI

Incoming commands are stored in one of the skid buffers and QoS in age order is used to pick commands from the buffer. The buffers in this case are:

- MRD skid buffer
- RBID skid buffer
- NC read write command skid buffer (does same address checks)

All outgoing associated messages will have the calculated priority value on it plus the native QoS value. QoS value is then forwarded on native interface.

Cache evictions take the value of the transaction that caused the eviction.

7.4.5.1.1 DMI Threshold support

The purpose of these changes is to work around AXI head of line blocking issue. Expected top-level view of DMI and DMC connectivity is shown in Figure 7-1. The user of NCore is expected to develop the below buffer and mux/demux.

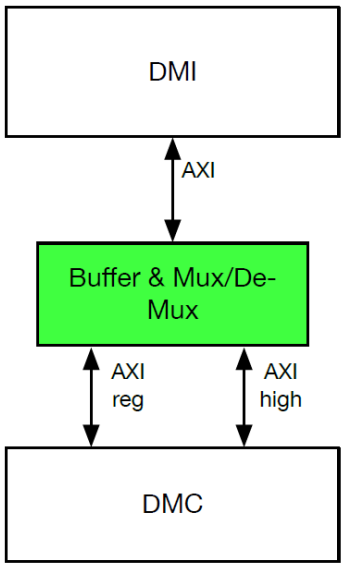


FIGURE 7-1 DMI AND DMC CONNECTIVITY

The DMC used has 2 AXI ports – one for regular traffic "AXI reg" and another for high priority or real time traffic "AXI high". The buffer & mux/de-mux block contains buffer that is larger than the DMC's AXI reg port buffer. And the module is responsible for routing the high-priority or real time traffic to DMC's AXI, and other traffics will be routed to DMC's AXI reg port. The buffer being larger than the buffer DMC's AXI reg port buffer guarantees that high priority traffic does not see head of line blocking.

7.4.5.1.2 Parameters

Three new DMI level parameters is introduced:

TABLE 7-2: DMI PARAMETERS FOR QOS CONFIGURATION

Parameter Name	Description	Type	Default	Min	Max
DmiQoSThVal	DMI QoS threshold value. Traffic with QoS equal to or above this value are considered as high priority hard real time traffic	Int	8	1	15
nDmiWttQoSRsv	WTT entries in DMI reserved for high priority real time traffic	Int	8	1	32
nDmiRttQoSRsv	RTT entries in DMI reserved for high priority real time traffic	Int	8	1	32

NOTE nDmiRtt/WttQoSRsv Architecture max value for architecture is 64

Table 7-2 is here for illustration. Please refer to the parameters specification for the description of DMI parameters for QoS.

7.4.5.1.3 *Changes in DMI*

Needed changes in DMI are shown in Figure 7-2. This change involves implementing a feedback loop from DMI egress AXI port i.e., both RTT/WTT to the SMI ingress skid buffers. RTT and WTT must reserve certain number of entries for high priority real time traffic. The number of reserved entries in RTT and WTT are defined by nDmiRttQoSRsv and nDmiWttQoSRsv parameters respectively. QoS threshold value for high priority real time traffic is defined by parameter DmiQoSThVal. These three values can be changed at a boot time via a CSR.

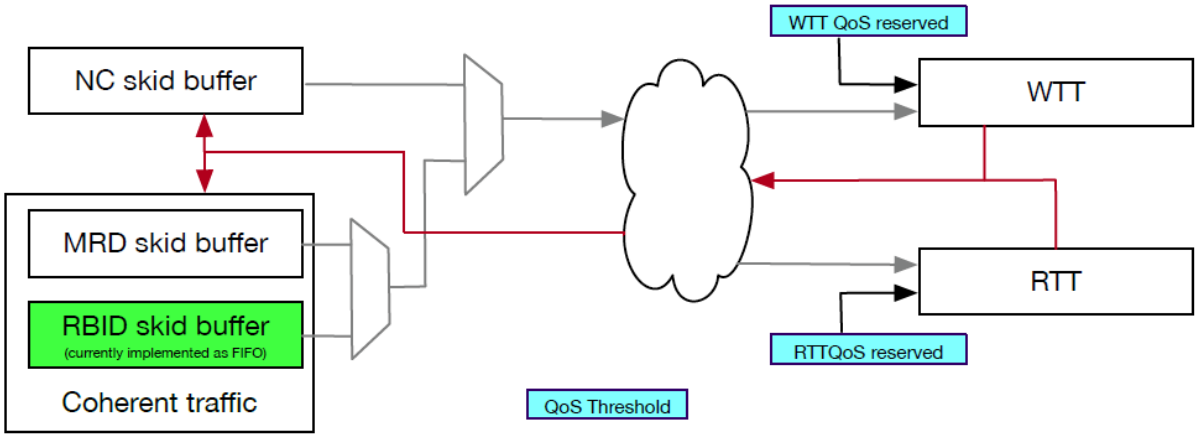


FIGURE 7-2 DMI FEEDBACK LOOP

During runtime when WTT/RTT have only reserved entries left they individually give feedback to the skid buffers to send only high-priority real time traffic. As this traffic is provided individually the skid buffer is responsible to resend individually for either read or write traffic i.e., at any given time only one or both RTT/WTT can push back regular traffic and the skid buffer should handle them separately.

Once regular entries free up in RTT/WTT then the feedback is removed, and regular traffic can start making forward progress. The feedback path is shown with read arrows. Note that the current implementation of coherent writes in DMI is in FIFO order this will need to be changed to priority-based skid buffer.

The user or customer is expected to size the WTT/RTT and reserved entries with some margins to account for pipeline delays. The micro-arch is expected to provide DMI pipeline delay between Skid buffer and the WTT/RTT.

The skid buffer reservation logic should ignore the reserved entries feedback and push starved entries through.

8 Power management

Ncore 3 clock and power are defined in terms of regions, domains, and sub domains.

Regions are defined as follows:

- A power region represents a group of elements that run off a power supply that is driven by one source.
- A clock region represents a group of elements that are clocked by a single source. Clock regions are considered Asynchronous to each other.

Domains are defined as follows:

- A power domain represents a group of elements whose power can be turned on and off.
- A clock domain represents a group of elements whose clock can be turned on and off. A Clock domain may be defined at build time as synchronous or asynchronous with other clock domains within the clock region.

Sub Domains are defined as follows:

- There are no power sub domains.
- A clock sub domain is a group of elements in a clock domain whose clocks can be turned on and off dynamically as a part of logic function such as clock divider. These are not supported in the current version of Ncore i.e., it supports only one clock subdomain per clock domain, and it cannot be divided down.

The parent child relationship of regions, domains and sub domains is shown in Figure 8-1. A Clock region is associated to a power region. A clock domain is associated to a power domain within the same parent child relationship.

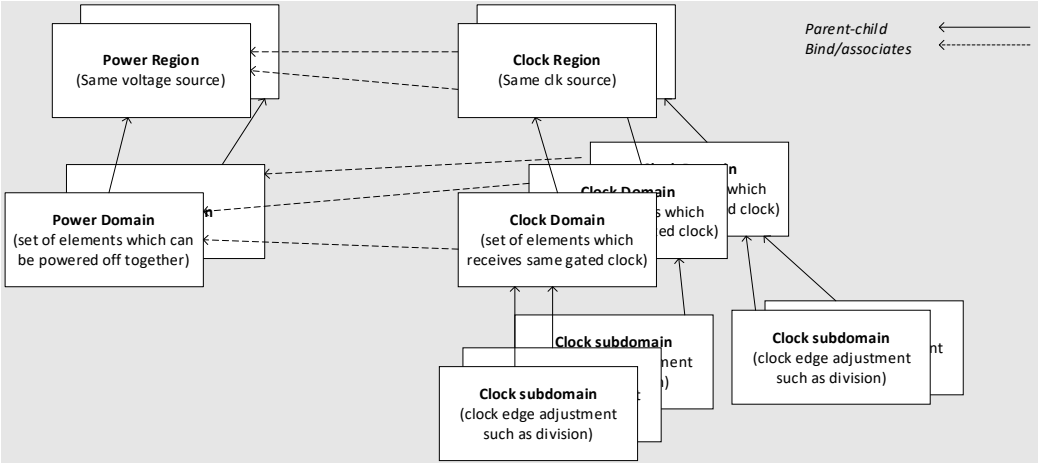


FIGURE 8-1 POWER AND CLOCK, PARENT CHILD RELATIONSHIP

8.1 Synchronization

Synchronizing logic is inserted at every asynchronous crossing between two clock regions. As clock domains can be defined as synchronous or asynchronous, synchronizing logic is only inserted if two clock domains are defined as asynchronous.

Synchronizing logic may include any one or combination of the following:

- Multiple flopping of a single bit signal
- Request Acknowledgment handshake protocol
- Synchronizing FIFOs using gray counters

8.2 Power Gating

Ncore does not support power gating. It supports only a single Power domain. It does not provide a unified power format (UPF) file that describes level shifters and clamping cells. A user may support multiple power domains by defining UPF file by themselves.

8.3 Clock Gating

Ncore supports two levels of clock gating.

First level of clock gating is done at the flop level. These clock gates are expected to be inserted by the synthesis tool. It is recommended to gate all collection of flops which can be grouped into three or more. RTL is expected to provide enable pins on these flops to achieve this. First level of clock gating is configurable as the user may decide not to enable the option during synthesis

Second level of clock gating is at per Ncore unit basis. This is a single clock gate instantiated within each Ncore unit, It gates the whole Ncore unit except external interfaces and synchronizers. It is enabled when the Ncore unit is idle and does not have any transactions pending within itself. Second level clock gating is configurable via build time parameters as specified in the parameter spec.

8.4 Q channel

The top-level system view with Q channel support is shown in Figure 8-2. Note that the view is only for a single clock domain implementation. In the case of multi clock domain implementation there will be one PMA master / Q channel interface per domain.

Q channel in Ncore can be used to provide additional clock gating capability where the clock gate exists in users' logic and is not implemented by Ncore. When this capability is used, two synchronous domains must be created, one that is always on and another that can be dynamically clock gated with the help of a Q channel. This is required to make sure that the CSR network connected to Ncore is always on and does not cause CSR transactions to be lost. Q channel clock gating requires Software mechanism as described in Q channel clock gating sequence section.

Q cannel in Ncore does not support auto-wakeup, i.e., in powered down state QACTIVE will not be asserted to initiate a wakeup request by power management unit (PMU). Note that Ncore does not implement a PMU, this is expected to be user logic. Ncore only implements PMA master and PMA slave.

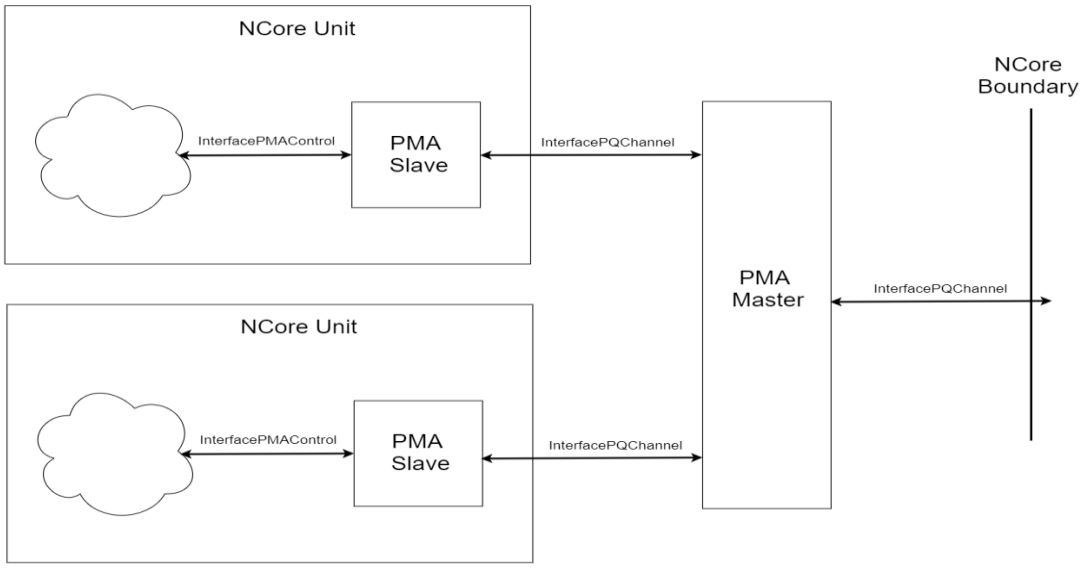


FIGURE 8-2 Q CHANNEL PMA TOP-LEVEL VIEW

9 Error handling

This chapter goes over Error handling in Ncore 3. Effort has been made to reduce the number of changes from previous implementation. In a later version of Ncore 3 more changes and improvements will be introduced. The document first describes different registers and encodings and then goes into details of how these errors are handled.

9.1 Error registers

Refer to CSR CPRs in in the hw-ncr repository under cpr/csr for register addresses.

9.1.1 Correctable Error Registers

9.1.1.1 Correctable Error Control Register (xCECR)

Register Description: This register controls the detection and interrupt signaling of Correctable Errors in the unit.

Register Fields:

Bits	Name	Access	Reset	Description
0	ErrDetEn	RW	0b0	Correctable Error Detection Enable. When set to 1, this bit enables detection and logging of correctable errors.
1	ErrIntEn	RW	0b0	Correctable Error Interrupt Enable. When set to 1, this bit enables the assertion of Correctable Error Interrupt signal when a new Correctable Error is detected and ErrCount value equals to ErrThreshold.
3:2	Reserved	-	-	-
11:4	ErrThreshold	RW	0x00	Correctable Error Threshold. Number of corrected errors for which logging, and interrupt (if ErrIntEn is set to 1) is suppressed.
31:12	Reserved	-	-	-

9.1.1.2 Correctable Error Status Register (xCESR)

Register Description: This register logs information of Correctable Errors in the unit. If a new Correctable Error attempts to log this register the same cycle xCESAR is configured, this register will contain information from xCESAR.

Register Fields:

Bits	Name	Access	Reset	Description
0	ErrVld	W1C	0b0	Correctable Error Valid. This bit is set when a Correctable Error is detected and ErrCount equals to ErrThreshold. xCESR, xCELR0 and xCELR1 contain logged information about this Correctable Error. Write 1 to the field resets it to 0b0.
1	ErrCountOverflow	RO	0b0	Correctable Error Count Overflow. This bit is set to 1 when ErrVld is asserted, and a new Correctable Error

				is detected. Once the field is set, it will keep asserted until being reset. The field is reset to 0b0 when ErrVld is reset.
9:2	ErrCount	RO	0x00	Correctable Error Count. This field increments when a Correctable Error is detected. Once the field reaches ErrThreshold, the value will be frozen from that time until being reset. The field is reset to 0x00 when ErrVld is reset. If write 1 to ErrVld and a new Correctable Error is detected at the same cycle, the field will not increment for the error.
11:10	Reserved	-	-	-
15:12	ErrType	RO	0x00	Correctable Error Type. When ErrVld is set to 1, the field indicates the type of error that has been detected and logged.
31:16	ErrInfo	RO	0x0000	Correctable Error Information. When ErrVld is set to 1, the field contains additional unit-specific and error-type-specific information about the error. For example, information that could help identify the site of the error.

9.1.1.3 Correctable Error Location Register (xCELR0)

Register Description: This register logs location information of Correctable Errors in the unit. The information is unit-specific (see Error Type and Information Summary). The fields are made RW accessible to utilize this register as the alias register for error information injection purpose.

Register Fields:

Bits	Name	Access	Reset	Description
19:0	ErrEntry	RW	0x00000	Error Entry. When ErrVld is set to 1, the field contains location information of the error. If the error is from a memory array, it indicates the Entry or Set (if from a cache) number where the error occurs. Otherwise, it is the [19:0] bits of the error address.
25:20	ErrWay	RW	0x00	Error Way. When ErrVld is set to 1, the field contains location information of the error. If the error is from a cache, it indicates the Way number where the error occurs. Otherwise, it is the [25:20] bits of the error address.
31:26	ErrWord	RW	0x00	Error Word. When ErrVld is set to 1, the field contains location information of the error. If the error is from a cache, it indicates the beat number where the error occurs. Otherwise, it is the [31:26] bis of the error address.

9.1.1.4 Correctable Error Location Register (xCELR1)

Register Description: This register logs extra address information that cannot fit in xCELR0. The fields are made RW accessible to utilize this register as the alias register for error information injection purpose.

Register Fields:

Bits	Name	Access	Reset	Description
19:0	ErrAddr	RW	0x000	Error Address Higher Bits. Contains higher bit of error address if xELR0 cannot fit the whole error address.
31:20	Reserved	-	-	-

9.1.1.5 Correctable Error Status Alias Register (xCESAR)

Register Description: This register is used to inject error status into xCESR. If a new Correctable Error attempts to log the same cycle this register is configured, the information in this register will be logged in the xCESR.

Register Fields:

Bits	Name	Access	Reset	Description
0	ErrVld	RW	0b0	Alias bit for setting ErrVld in xCESR.
1	ErrCountOverflow	RW	0b0	Alias bit for setting ErrCountOverflow in xCESR.
9:2	ErrCount	RW	0x00	Alias field for setting ErrCount in xCESR.
11:10	Reserved	-	-	-
15:12	ErrType	RW	0x00	Alias field for setting ErrType in xCESR.
31:16	ErrInfo	RW	0x0000	Alias field for setting ErrInfo in xCESR.

9.1.1.6 Correctable Resiliency Threshold Register (xCRTR)

Register Description: This register contains the resiliency correctable error threshold as an indication to the functional safety controller (FSC). Exists only if the unit enables resiliency.

Register Fields:

Bits	Name	Access	Reset	Description
7:0	ResThreshold	RW	0x01	Resiliency Correctable Error Threshold. See FSC documents for more information.
31:8	Reserved	-	-	-

9.1.2 Uncorrectable Error Registers

9.1.2.1 Uncorrectable Error Interrupt Register (xUEDR)

Register Description: This register enables the detection of Uncorrectable Errors.

Register Fields:

Bits	Name	Access	Reset	Description
0	ProtErrDetEn	RW	0b0	Protocol Error Detection Enable. When set to 1, this bit enables the detection and logging of Protocol type Uncorrectable Error.
1	TransErrDetEn	RW	0b0	Transport Error Detection Enable. When set to 1, this bit enables the detection and logging of Transport type Uncorrectable Error.
2	MemErrDetEn	RW	0b0	Memory Error Detection Enable. When set to 1, this bit enables the detection and logging of Memory type Uncorrectable Error. This field exists only when there are protected memory arrays in the unit.
3	DecErrDetEn	RW	0b0	Decode Error Detection Enable. When set to 1, this bit enables the detection and logging of Access type Uncorrectable Error.
4	TimeOutErrDetEn	RW	0b0	Time Out Error Detection Enable. When set to 1, this bit enables the detection and logging of Time Out Error.
31:5	Reserved	-	-	-

9.1.2.2 Uncorrectable Error Interrupt Register (xUEIR)

Register Description: This register enables the interrupt signaling of Uncorrectable Errors.

Register Fields:

Bits	Name	Access	Reset	Description
0	ProtErrIntEn	RW	0b0	Protocol Error Interrupt Enable. When set to 1, this bit enables the assertion of Protocol type Uncorrectable Error Interrupt signal.
1	TransErrIntEn	RW	0b0	Transport Error Interrupt Enable. When set to 1, this bit enables the assertion of Transport type Uncorrectable Error Interrupt signal.
2	MemErrIntEn	RW	0b0	Memory Error Interrupt Enable. When set to 1, this bit enables the assertion of Memory type Uncorrectable Error Interrupt signal. This field exists only when there are protected memory arrays in the unit.
3	DecErrIntEn	RW	0b0	Decode Error Interrupt Enable. When set to 1, this bit enables the assertion of Access type Uncorrectable Error Interrupt signal.
4	TimeOutErrIntEn	RW	0b0	Time Out Error Interrupt Enable. When set to 1, this bit enables the assertion of Time Out Error Interrupt signal.
31:5	Reserved	-	-	-

9.1.2.3 Uncorrectable Error Status Register (xUESR)

Register Description: This register logs information about Uncorrectable errors in the unit. If a new Uncorrectable Error attempts to log this register the same cycle xUESAR is configured, this register will contain information from xUESAR.

Register Fields:

Bits	Name	Access	Reset	Description
0	ErrVld	W1C	0b0	Uncorrectable Error Valid. This bit is set when an Uncorrectable Error is detected. xUESR, xUELR0 and xUELR1 contain logged information about this Uncorrectable Error. Write 1 to the field resets it to 0b0. If write 1 to the field and a new Uncorrectable Error is detected at the same cycle, the field is cleared.
3:1	Reserved	-	--	-
7:4	ErrType	RO	0x00	Uncorrectable Error Type. When ErrVld is set to 1, the field indicates the type of error that has been detected and logged.
15:8	Reserved	-	-	-
31:16	ErrInfo	RO	0x0000	Uncorrectable Error Information. When ErrVld is set to 1, the field contains additional unit-specific and error-type-specific information about the error. For example, information that could help identify the site of the error.

9.1.2.4 Uncorrectable Error Location Register (xUELR0)

Register Description: This register logs location information of Uncorrectable Errors in the unit. The information is unit-specific (see Error Type and Information Summary). The fields are made RW accessible to utilize this register as the alias register for error information injection purpose.

Register Fields: see Correctable Error Location Register (xCELR0)

9.1.2.5 Uncorrectable Error Location Register (xUELR1)

Register Description: This register logs extra address information that cannot fit in xUELR0. The fields are made RW accessible to utilize this register as the alias register for error information injection purpose.

Register Fields: see Correctable Error Location Register (xCELR1)

9.1.2.6 Uncorrectable Error Status Alias Register (xUESAR)

Register Description: This register is used to inject error status into xUESR. If a new Uncorrectable Error attempts to log the same cycle this register is configured, the information in this register will be logged in the xUESR.

Register Fields:

Bits	Name	Access	Reset	Description
0	ErrVld	RW	0b0	Alias bit for setting ErrVld in xUESR.
3:1	Reserved	-	-	-
7:4	ErrType	RW	0x00	Alias field for setting ErrType in xUESR.
15:8	Reserved	-	-	-
31:16	ErrInfo	RW	0x0000	Alias field for setting ErrInfo in xUESR.

9.2 Error Type and Information Summary

9.2.1.1 Correctable ErrorType and Info Table

ErrType Code	Error Type and ErrInfo Code	Error Location			
		Addr	Word	Way	Entry
0x0	Data Correctable Error • [2:0] – Storage Type ○ 3'b000 – OTT-Data ○ 3'b001 – Read Buffer ○ 3'b010 – Write Buffer ○ 3'b011 – Snoop Filter ○ 3'b100 – Trace Buffer • [7:3] – Reserved • [15:8] – Snoop Filter ID (Reserved for non-snoop filter)	MBZ*/ MBZ	MBZ/ N/A	MBZ/ Way	Entry /Set
0x1	Cache Correctable Error • [0] – Array Type ○ 1'b0 - Tag Array ○ 1'b1 - Data Array	MBZ	Word	Way	Set
0x2 – 0xF	Reserved	N/A	N/A	N/A	N/A

*MBZ: Must Be Zero

9.2.1.2 Uncorrectable ErrorType and Info Table

ErrType Code	Error Type and ErrInfo Code	Error Location			
		Addr	Word	Way	Entry
0x0	Data Correctable Error • [2:0] – Storage Type ○ 3'b000 – OTT-Data ○ 3'b001 – Read Buffer ○ 3'b010 – Write Buffer ○ 3'b011 – Snoop Filter ○ 3'b100 – Trace Buffer • [7:3] – Reserved • [15:8] – Snoop Filter ID (Reserved for non-snoop filter)	MBZ*/ MBZ	MBZ/ N/A	MBZ/ Way	Entry /Set
0x1	Cache Correctable Error • [0] – Array Type ○ 1'b0 - Tag Array	MBZ	Word	Way	Set

	○ 1'b1 - Data Array				
0x2	Native Interface Write Response Error • [1:0] - Response • [2] - Security Attribute • [3]- Eviction	Transaction Address			
0x3	Native Interface Read Response Error • [1:0] - Response • [2] - Security Attribute • [3]- Fill	Transaction Address			
0x4	Native Interface Snoop Response Error • [1:0] – Response (Reserved for Snoop filters) • [2] - Security Attribute	Transaction Address			
0x5-0x6	Reserved	N/A	N/A	N/A	N/A
0x7	Decode Error • [3:0] – Type ○ 4'b0000: No address hit ○ 4'b0001: Multiple address hit ○ 4'b0010: Illegal CSR access format ○ 4'b0011: Illegal DII access type ○ 4'b0100: unconnected DMI unit access ○ 4'b0101: unconnected DII unit access ○ 4'b0110: unconnected DCE unit access ○ 4'b0111: No credits configured for access ○ 4'b1000: Illegal security access ○ 4'b10001 to 4'b1111: Reserved • [4] – Command Type (Reserved for CHI-AIU & DCE) ○ 1'b0: Read ○ 1'b1: Write • [5] – Reserved • [15:6] – Transaction ID/AXID (Reserved for DCE)	Transaction Address			
0x8	Transport Error • [0] – Type ○ 1'b0: Wrong Target Id • [5:1] – Reserved • [15:6] – Source ID (FUnit_Id)	-			
0x9	Timeout Error • [1:0] – Reserved • [2] - Security Attribute • [15:3] - Reserved	Transaction Address			
0xA	Sys Event Error • [0] – Type ○ 1'b0: Time out error on message (protocol timeout) ○ 1'b1: Time out error on interface	-			

	• [15:1] – Reserved				
0xB	SysCo Error • [0] – Type ○ 1'b0: Time out error on message (protocol timeout) • [15:1] – Reserved	-			
0xC – 0xF	Reserved	N/A	N/A	N/A	N/A

9.3 Native Interface Error Response Logging and interrupt

Uncorrectable Error is logged when an error response is detected at the native interfaces if ProtErrDetEn = 1.

- DMI/DII logs all AXI Bresp/Rresp errors
- CHIAIU logs all CHI data/non-data response errors
- IOAIU logs all ACE CRresp errors (CRresp[1] is logged into ErrInfo Response field)

An Uncorrectable interrupt is raised if ProtErrIntEn = 1.

CCMP must be completed with error CmStatus for DtwReq and DtrReq, if the first beat of data contains error else the CmStatus does not indicate an error. If following beats of data contain error, it is indicated by setting the appropriate DBad of the data beat. Native interface to CmStatus mapping is shown inTable 9-1.

TABLE 9-1: NATIVE INTERFACE TO CMSTATUS MAPPING

Native interface	Native interface error	CmStatus	Description
AXI (DMI/DII)	SLVERR	0b10000011	Data error in CmStatus. As per ARM spec this error is reported when the slave cannot provide data due to various reasons
	DECERR	0b10000100	Address error in CmStatus. As per ARM Spec this error is reported when the slave cannot be found by the network
CHI (CAIU)	Data Error	0b10000011	Data error in CmStatus. As per ARM spec this error is reported when data is corrupted. The error may be reported on datflit or rspflit.
	Non-Data Error	0b10000100	Address error in CmStatus. As per ARM Spec this error is reported when the response cannot be completed due to various reasons. The error may be reported on datflit or rspflit.
ACE (CAIU)	On CR channel bit 1	0b10000100	Address error in CmStatus. As per ARM spec, the agent cannot complete the snoop request. (Today RTL issues Target Signaled error for DVM snoops, this needs to be updated in later versions to address error) Concerto expects following: If the originating snoop was an invalidating type, then the agent is expected to go to invalid state else stay in the same state. Snoop filer is updated accordingly.

In the cases where the data is provided at the native interface the error is propagated on either DtwReq or DtrReq. When data is not provided then the error is propagated on SnpRsp or DtwRsp.

In Ncore System, data can be rotated per DW (Double Word), the beat data error information from native interface is carried in each DW, and after rotation, certain error type can overwrite the others. For example, in AXI, if the first beat sent on SMI contains double word for both SLVERR and DECERR, Address Error is injected into CMStatus.

CmStatus to native interface mapping is shown inTable 9-2. In the case of DtwReq and DtrReq the first beat CmStatus may not indicate error, but consecutive beats may indicate DBad in this case the consecutive beats are treated as Data Error.

TABLE 9-2 CMSTATUS TO NATIVE INTERFACE MAPPING

Native interface	CmStatus	Native interface error	Description
AXI/ACE-Lite/Ace-Lite E/ACE (NCAIU/CAIU)	0b10000011	SLVERR	Data error is best represented by SLVERR
	0b10000100	DECERR	Address error where the slave cannot be found is best represented by DECERR. In the case of DVMs (CmpRsp) this must be decoded to SLVERR
CHI (CAIU)	0b10000011	Data Error	Data error, this includes SLVERR (reported by DMI/DII) is best mapped to Data Error on CHI as Ncore does not have enough information to decode the SLVERR and it may be reported due to bad data
	0b10000100	Non-Data Error	Address error, this includes DECERR (reported by DMI/DII) is best mapped to Non-Data Error on CHI as this error is reported when the final target cannot be found.

9.4 Transport Error Logging and interrupt

Ncore 3 units support only Wrong Target ID error. Uncorrectable Error is logged when seeing message targetID (exclude Port_ID) mismatches unit's FUnit_ID if TransErrDetEn = 1. The message is dropped. An Uncorrectable interrupt is raised if TransErrIntEn = 1.

9.5 Resiliency Related Error Logging and interrupt

SMI Protection Error

- If both resiliency and SMI protection are enabled, it is FSC's responsibility to log and interrupt SMI protection error, which includes: Concerto Header Error, Concerto Message Error and Concerto Data Error.
- Upon an SMI protection Uncorrectable Error, the message is dropped by the unit and error is logged and interrupt by FSC.
- Upon a SMI Protection Correctable Error, the error is corrected by the unit and the error counter is incremented. If Error Count reaches ResThreshold and a new error is detected, the error is logged, and an interrupt is asserted by FSC.

Please see the resiliency section for FSC error handling.

In addition, following uncorrectable errors are reported to the fault checker in resiliency mode and will trigger mission fault

- Uncorrectable memory errors which include both parity and ECC
- Wrong target ID error

9.6 Address Map Decode Error Logging and interrupt

AIU and DCE detects and logs the following Uncorrectable Errors upon address decoding if DecDetErrEn = 1, If multiple of these errors are reported for the same transaction, then reporting order of priority is from top to bottom as below:

- No Address hit
  - If the target cannot be found in the address map
- Multiple Address hit
  - If multiple targets are found in the address map
- Illegal Security access
  - If Non-Secure transaction access secure region as configured in the address map
- Illegal CSR access format
  - If the target is decoded as 32-bit CSR DII, and the following requirements are not met
    - 4 Byte
    - Size-aligned
    - EndPoint order
- Illegal DII access type
  - Coherent access on DII
- Unconnected DMI unit access
  - If the target DMI as decoded by the address map is not connected to the AIU
- Unconnected DII unit access
  - If the target DII as decoded by the address map is not connected to the AIU
- Unconnected DCE unit access
  - If the target DCE as decoded by the address map is not connected to the AIU
- No credits configured for access
  - If the target as decoded by the address map does not have any credits configured

An Uncorrectable interrupt is raised if DecErrIntEn = 1.

In addition, an AIU needs to complete the protocol and return error in the response.

- ACE/ACE-LITE/AXI: DECERR
- CHI: non-data error

A DCE needs to drop the recall transaction, i.e., pretend it finished successfully. In the case of ‘no credits configured for access’ error (Mrd Credits), DCE must issue an address error with in the cm status field of the StrReq.

9.7 CCMP Completion

AIU does not expect DtrReq if StrReq contains error CmStatus for a read request, and it needs to manufacture response to the processor.

When IOAIU detects an OTT data buffer uncorrectable error

- DtwReq is returned with CMStatus = 8'b10000011 (Data Error) and poison bit set. If error is not detected on first beat, then only poison bit is set (writes)
- Data that is to be stored in cache the poison bit is set
- If the data is to be sent on to the native interface i.e., R Channel, RResp is returned with SLVERR (Reads)

When IOAIU is configurated with cache:

- For a tag Uncorrectable Error during cache lookup due to native interface request, IOAIU treats the request as a cache miss and furthermore does not allocate to the cache. In this case uncorrectable error is logged in CSrs and reported via interrupt.
- For a tag Uncorrectable Error during SNPReq lookup, IOAIU issues SNPRsp with CmStatus = 8'b10000100 (Address Error).
- For a data Uncorrectable Error upon access cache data RAM, if first beat data is poisoned and DtrReq/DtwReq is returned with CmStatus = 8'b10000011 (Data Error) and poison bit set, if a data beat after the first beat is poisoned then only poison bit is set and CmStatus may not indicate error. In the case where data is to be provided on the native interface i.e. R Channel, RResp is returned with SLVERR on the appropriate data beat.

When IOAIU processes a multi-line transaction

- For read transaction each individual Cache line is for error reporting purposes is treated separately as if it was a single cache line transaction. Requirements are defined in the section above
- For write transaction the final response on the native interface must be the accumulation of all individual cache line write responses. Here Address error (DECERR) has higher precedence than (SLVERR).

When DCE gets a snoop filter look up error

- DCE issues STRReq with CMStatus = 8'b10000100 (Address Error).

When DCE is configured with zero DMI credits

- DCE issues STRReq with CMStatus = 8'b10000100 (Address Error).

When DCE gets any snoop response error

- If any snooped AIU is providing the data, the protocol is completed by the DtrReq and the error is not propagated by DCE via StrReq. (Exception for read Stash below)
- If no snooped AIU is providing the data (DtrReq) then DCE issues StrReq with propagated error CmStatus from the last received SnpRsp with error. (Exception for read Stash below)
- In the case of Read Stash where the non-target reports SnpRsp error DCE does not propagate it and issues MrdReq to complete the request
- In the case of Read Stash where the target reports SnpRsp error, DCE propogates the error on StrReq and aborts the transaction.
- In the case of Write stash and Write Unique SnpRsp errors are not propagated onto to StrReq.
- If the original snoop was an invalidating snoop type (i.e the snooped agent is required to go to invalid state) then the snoop filter is updated to invalidate that agent, for all other snoop types, snoop filter status is not updated for that agent. (In general, it is expected that the agent that responded with an error went to an invalid state, but Ncore takes a more conservative approach due to ambiguity in ARM specs and only invalidates the snoop filter for cases where the snoop type required an invalidation)
  Invalidating snoop type are: SnpInvDtw, SnpInvDtr, SnpInv, SnpInvStsh, SnpUnqStsh, SnpStshUnq.

When DMI is configurated with cache:

- For a tag Uncorrectable Error during SMC lookup due to CmdReq/ MrdReq/ DtwMrgMrdReq messages, DMI issues a control propagation DtrReq message that indicates an address corruption error. In addition, the DMI manufactures appropriately sized and poisoned data payload for the Dtrreq with CmStatus = 8'b10000100 (Address Error). Since the entire data payload is poisoned, the actual content of the payload is not important. For a tag error during SMC lookup due to a prefetch message, DMI drops the message. DMI completes the CCMP Prefetch transaction by sending STRreq without an error signal.
- For a data Uncorrectable Error upon access SMC data RAM, data is poisoned and DTRReq is returned with CMStatus = 8'b10000011 (Data Error)

When DVE gets a Snoop response error

- DVE issues CmpRsp with CMStatus = 8'b10000100 (Address Error).

CMStatus error is propagated only on following responses

- SnpRsp coming into DCE, only if no other related SnpRsp issued a DTR then this is propagated on to StrReq else it is not propagated on to StrReq
- DtwRsp coming into AIUs, this is propagated on to the native interface
- MrdRsp coming into DCE, this is propagated on to StrReq
- CmpRsp coming into AIUs, this is propagated on to the native interface

CMStatus error is propagated only on following requests

- StrReq going out of DCE
- DtrReq going out of AIU, DMI, and DII

In the case of coherent Atomic transactions, where there are two parts the coherent part (clean operation with DCE) and the non-coherent part (atomic transaction with DMI) following applies

- If an error is reported back to the initiating AIU during the first coherent part, then the error is propagated on the Native interface and the second part is not carried out. Note that the AIU is expected to issue proper response and deallocate its transaction entry
- If an error is reported back to the initiating AIU during the second non-coherent part, then the error is propagated on the Native interface. Note that the AIU is expected to issue proper response and deallocate its transaction entry

In the case of ACE and IOAIU with proxy cache configurations where WriteBacks, WriteClean, WriteEvict, Evict and cache eviction can have two parts i.e. non-coherent write to DMI and coherent update to DCE. If an error is reported back during the non-coherent transaction part, then the coherent update parts needs to be completed. In the case of ACE the error needs to be propagated on the native interface.

9.8 Data Poisoning

When seeing any DBad on DtwReq, the receiver should not log error and just issue normal CMStatus in the response. However, the data is marked as poisoned if filling into a cache and the write strobes for the whole beat are de-asserted if writing to downstream.

Note that there can be cases where the CmStatus of the DtrReq and DtwReq may not report the error but DBad may be set. This happens in cases where the error was detected after processing the first beat of data. If error is reported in the CmStatus then the whole Cache line is considered to be poisoned.

When seeing DBad, the corresponding DataBenable is not taken into account. It is a conservative behaviour and it may be improved in future version by not poisoning the data when it is DataBenable is not asserted on the marked data.

9.9 Timeout Error

Timeout counters are implemented by AIUs, DCEs and DMIs; latter two implement it specifically to handle timeout due to recall or evictions transactions respectively. Every other transaction is covered by AIUs.

Timeout Detection:

One global counter is designed to count cycles. Once the counter reaches a programmable threshold, the timeout overflow bit in all valid entries is set. This indicates the start of the period where the entry must make progress. The timeout overflow bit will be cleared if the entry deallocates. If the global counter reaches the threshold again while the timeout overflow bit is still set, then that entry has timed out. The event is logged in an error register to allow an interrupt to be raised if enabled.

Timeout Handling:

The status is logged in the CSR software will now have the option to either disable timeout and continue or treat this as a fatal error. The timeout counter will stop running, until software restarts it via CSR.

9.10 Sys event and SysCo Errors

Sys event sender can report a timeout error if it does not get a SysRsp message for the SysReq it sent out. This is also referred to as protocol timeout.

Sys event receiver can report a timeout error if it does not get an ack back on the event pins.

Sys Co handler can report a timeout error if it does not get a SysRsp message for the SysReq it sent out.

9.11 Unsupported Message Type

If a Ncore 3 unit received an unsupported Message (for example DMI gets a snoop command), the message will not be processed and eventually trigger a timeout error.

9.12 Error Reporting in CMStatus

If multiple errors are detected in a message, Ncore 3 Unit will report control errors (eg. Address Error) over data error in CMStatus.

10 Functional Safety

Ncore 3 resiliency implementation for all Ncore units is outlined here. The top-level system view is shown in Figure 10-1:Resiliency Top-level system view.

At system level Ncore 3 implements the following functional safety features:

- Unit duplication with 1 to 4 cycle delay
- Native interface protection via place holder
- Transport protection on SMI interface, this can be either parity of ECC (SECDED).
- SRAM structure protection in Ncore units, this can be either parity or SECDED for address and data.
- Configuration registers protection

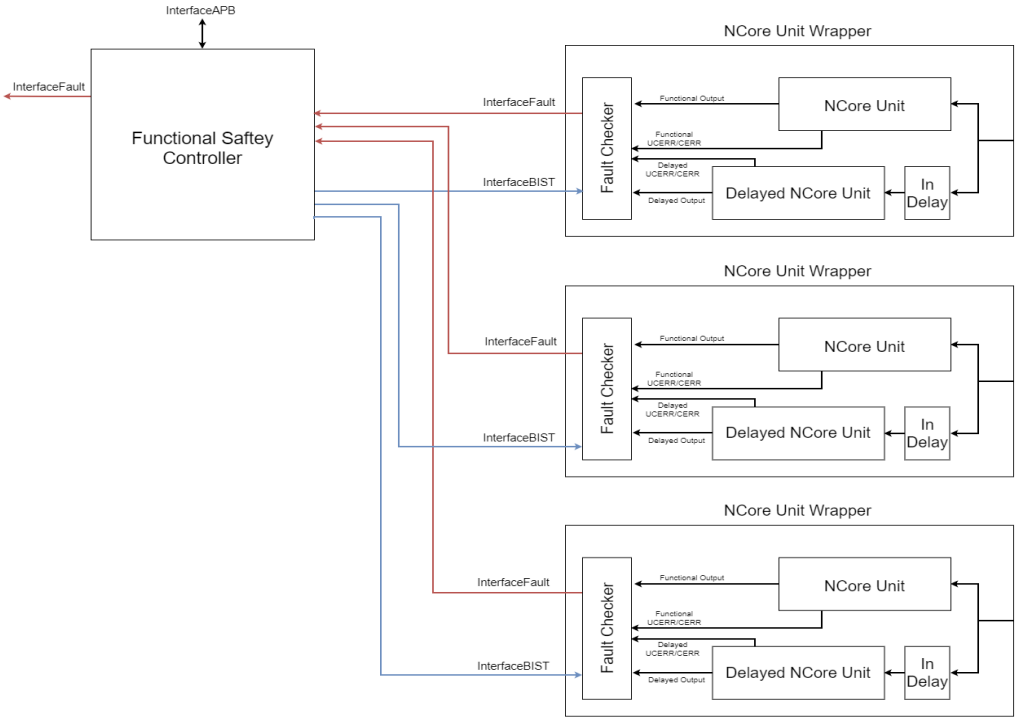


FIGURE 10-1:RESILIENCY TOP-LEVEL SYSTEM VIEW

Any system configuration with ASIL level functional safety must instantiate a wrapper for each functional module. This wrapper shall include at least one active functional unit and a fault checker unit (with BIST support). Ncore 3, ASIL protection requires a duplicated functional unit, the fault checker performs a cycle-by cycle comparison of the active and the checker unit.

10.1Functional SafetyFunctional Safety Controller

The functional Safety Controller has two operations.

- Performs BIST Functionality over the Fault Checkers
- Accumulates the InterfaceFault interfaces into a single master InterfaceFault

FSC supports the following features:

- Accumulates faults and reports them upstream and in registers.

- Performs BIST of Resiliency reporting logic of connected blocks.

- Configurable through an APB Interface.

- Can connect to blocks in multiple power and clock domains.

- All internal registers shall be protected by parity

See FSC Micro Architecture specification for more details

10.2 Fault Checker

The fault checker takes in four interfaces.

1. The functional outputs.
2. The delayed outputs.
3. The functional CERR/UCERR signals.
4. The delayed CERR/UCERR signals.

and then drives an InterfaceFault to the safety controller. See fault checker Micro Architecture specification for more information on how this block accumulates these signals and drives InterfaceFault.

10.3 Input Delay

An input delay unit is a simple unit which delays all inputs to the block by a parameterizable number of cycles. This is used to drive the delayed unit. See checker delay Micro Architecture for more detail.

10.4 Correctable Error Threshold

Each block will have a register controlling its correctable error threshold specifically for reporting to the FSC. (Each block may also have a register for a correctable error interrupt threshold). This value is driven to the Fault Checker. In addition, this value must be part of the interfaces that are checked by the fault checker. The fault checker uses this value to count correctable errors before it asserts a cerr_over_thresh signal to the FSC.

10.5 Memories

A Ncore 3 unit might have external memories. If this is the case it will instantiate it inside the top wrapper and not within the unit itself. The signaling to the memories will be included in the fault checker inputs like all other signals. The signaling from the memories will be delayed and driven to the delayed unit like all other inputs.

10.6 SRAM protection

SRAMs shall be protected with parity or SECDEC in Ncore 3 units. In addition to the data bits, the address bits shall also be protected thanks to the following sequence of operations.

For Write:

- The address is not written into the SRAM.
- The ECC is calculated over a concatenation of address bits and data bits, with address bits being placed as the upper bits.
- The ECC and the data are written into the SRAM.

For Read:

- A concatenation of the expected address bits and the data bits from SRAM, together with the ECC bits from SRAM, is fed into the ECC logic for error detection and correction.
- If the ECC-corrected address bits do not match the expected address bits, then an uncorrectable error is detected.

10.7 Transport protection

The Legato transport layer features parity and ECC checks both on headers and data payload. CDTI network (see section 3.4) enables data and header protection end to end for SMI messages between Ncore 3 units. CDTI may comprise several individual networks, each shall allow to enable parity or ECC protection. CSTI shall also enable parity or ECC protection. In addition, the transport layer is protected by a timeout for each transaction sequence. The loss of a message is expected to trigger a timeout detection at the protocol layer. Other checks are implemented to verify the validity of the message at reception: the Target ID of the message shall correspond to the ID of the unit receiving it. It augments the coverage of errors captured for each transmission.

10.8 CSR register protection

Configuration and status registers shall be protected by parity or duplication.

11 Power and Clock

Ncore clock and power are defined in terms of regions, domains, and sub domains.

Regions are defined as follows:

- A power region represents a group of elements that run off a power supply that is driven by one source.
- A clock region represents a group of elements that are clocked by a single source. Clock regions are considered Asynchronous to each other.

Domains are defined as follows:

- A power domain represents a group of elements whose power can be turned on and off.
- A clock domain represents a group of elements whose clock can be turned on and off. A Clock domain may be defined at build time as synchronous or asynchronous with other clock domains within the clock region.

Sub Domains are defined as follows:

- There are no power sub domains.
- A clock sub domain is a group of elements in a clock domain whose clocks can be turned on and off dynamically as a part of logic function such as clock divider. These are not supported in the current version of Ncore 3 i.e., it supports only one clock subdomain per clock domain, and it cannot be divided down.

The parent child relationship of regions, domains and sub domains is shown in Figure 11-1. A Clock region is associated to a power region. A clock domain is associated to a power domain within the same parent child relationship.

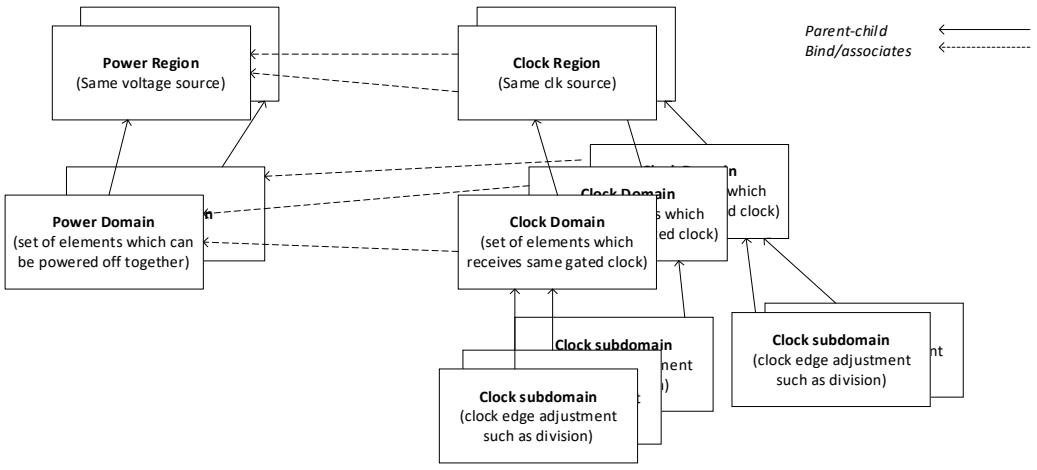


FIGURE 11-1 POWER AND CLOCK, PARENT CHILD RELATIONSHIP

11.1 Synchronization

Synchronizing logic is inserted at every asynchronous crossing between two clock regions. As clock domains can be defined as synchronous or asynchronous, synchronizing logic is only inserted if two clock domains are defined as asynchronous.

Synchronizing logic may include any one or combination of the following:

- Multiple flopping of a single bit signal
- Request Acknowledgment handshake protocol
- Synchronizing FIFOs using gray counters

11.2 Power Gating

Ncore 3 does not support power gating. It supports only a single Power domain. It does not provide a unified power format (UPF) file that describes level shifters and clamping cells. A user may support multiple power domains by defining UPF file by themselves.

11.3Clock Gating

Ncore 3 supports two levels of clock gating.

First level of clock gating is done at the flop level. These clock gates are expected to be inserted by the synthesis tool. It is recommended to gate all collection of flops which can be grouped into three or more. RTL is expected to provide enable pins on these flops to achieve this. First level of clock gating is configurable as the user may decide not to enable the option during synthesis

Second level of clock gating is at per Ncore unit basis. This is a single clock gate instantiated within each Ncore unit, It gates the whole Ncore unit except external interfaces and synchronizers. It is enabled when the Ncore unit is idle and does not have any transactions pending within itself. Second level clock gating is configurable via build time parameters as specified in the parameters specification.

11.4Q channel

The top-level system view with Q channel support is shown in Figure 11-2. Note that the view is only for a single clock domain implementation. In the case of multi clock domain implementation there will be one PMA master / Q channel interface per domain.

Q channel in Ncore can be used to provide additional clock gating capability where the clock gate exists in users' logic and is not implemented by Ncore. When this capability is used, two synchronous domains must be created, one that is always on and another that can be dynamically clock gated with the help of a Q channel. This is required to make sure that the CSR network connected to Ncore is always on and does not cause CSR transactions to be lost. Q channel clock gating requires Software mechanism as described in Q channel clock gating sequence section.

Q cannel in Ncore does not support auto-wakeup, i.e., in powered down state QACTIVE will not be asserted to initiate a wakeup request by power management unit (PMU). Note that Ncore does not implement a PMU, this is expected to be user logic. Ncore only implements PMA master and PMA slave.

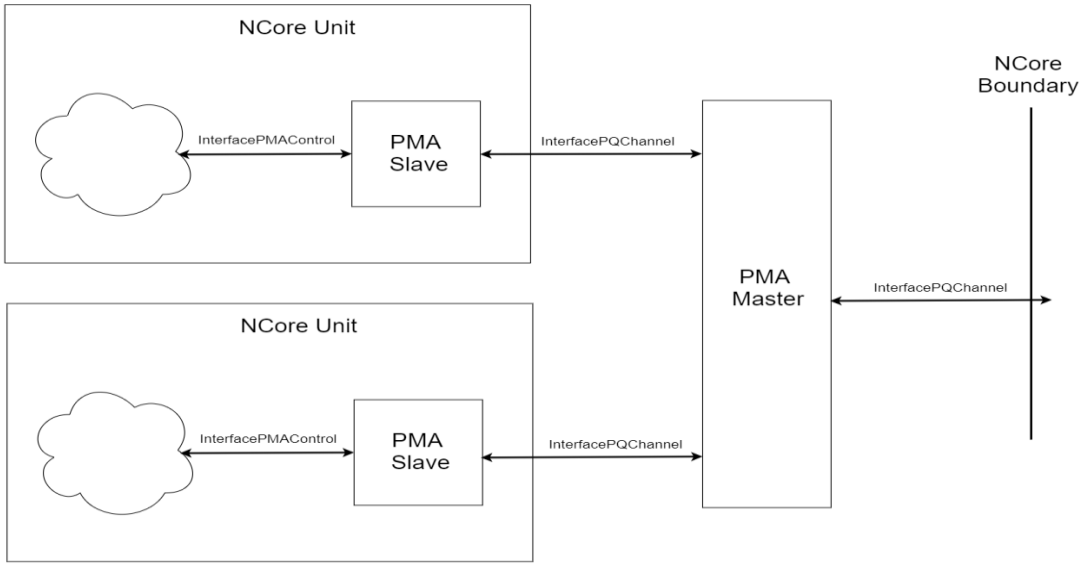


FIGURE 11-2 Q CHANNEL PMA TOP-LEVEL VIEW

12 Security

12.1 Non-secure bit of transactions

NCore 3 supports ARM trustzone model. In this model, the security attribute represents the NS bit, or non-secure bit. When the NS bit equals zero, the agent is performing a secure memory access, and when the NS bit equals one, the agent is performing a non-secure access. On ACE and ACE-lite agents, the NS bit is signaled on bit[1] of the AxPROT signal, the value of the NS bit is copied into the Security bit.

To support the Trustzone model in general, the security attribute must be stored in address buffers and tag filter entries, and any address comparisons must include comparison of the security attribute as well. To avoid security model violations, any address comparison where the security attribute mismatches is mismatch.

There is an exception, NS bit is not checked for transactions targeting CSR and boot regions. Nevertheless, an additional access security check (see section 5.1) is ensuring that only trusted agents are allowed to access CSR. CSR access must be disabled for untrusted agents while at least one agent must be enabled in the system.

The NS bit in a CHI transaction or AxPROT[1] in an ACE, ACE-Lite, and AXI transaction when set to 0 indicate that the transaction is secure. The Ncore 3 platform takes specific actions to ensure the security semantics are satisfied within the platform.

What various units in Ncore 3 need to do for handling the NS bit is listed below when Trust- Zone is enabled. When TrustZone is disabled, the NS bit is ignored by Ncore 3.

When TrustZone is enabled in the Ncore 3 system, secure and non-secure accesses can never access the same data via hardware. Any aliasing is expected to be resolved by software.

TABLE 12-1: TRUSTZONE IMPLEMENTATION IN EACH UNIT

Component	
AIU	CMDs and UPDs: • The CHI AIUs pass on the NS bit on the CHI interface to NS bit in ConcertoCMDreq messages. • The ACE, ACE-Lite, ACE-Lite-E, and AXI AIUs pass on the AxPROT[1] bit to the NS bit in the Concerto CMDreqmessages. SNPs: • The CHI AIUs pass on the NS bit in the SNPreq message to the NS bit on the CHI snoop interface. • The ACE AIU passes the NS bit in the SNPreq to AxPROT[1] bit on the ACE snoop interface.
DCE	• NS bit is treated as the most significant address bit. It is also stored as the most significant tag bit in the directory. • DCEs treat the NS bit as the most significant address bit for directory look-up. A tag match requires the NS bit match the most significant tag bit. • DCE copies the NS bit from the CMDreq to the corresponding SNPreqs, if any
DMI	Coherent accesses: • NS bit arrives via MRDs and RBR requests. (This came to DCE with CMDreq.) • NS bit is treated as the most significant address bit. • NS bit is stored in the tag arrays of the SMC as the most significant address bit. • Successful tag match in SMC requires that the NS bit matches the most significant bit of the address tag. • NS bit value is sent on AxPROT[1] bit on AXI when a CMDreq is directly forwarded to the AXI. • Cast out from SMC: Most significant address bit in the address tag of the cache line is sent on AxPROT[1] bit on AXI. The address itself on AXI is the tag address with the most significant tag bit set to 0 (or stripped off, as needed). Non-coherent accesses: • NS bit arrives via CMDreqs. • Non-coherent data can be stored in SMC. • The NS bit is stored in tag arrays of the SMC as the most significant address bit. • Successful tag match in the SMC requires: – NS == 0, and most significant bit of the tag can be 0 or 1, or ▪ Note: Coherency aliasing – meaning accessing the same data coherently and non-coherently can be a software error. However, under aliasing, 2 tags can match when NS == 0, one with most significant bit 0 and 1, which were perhaps stored under coherent accesses. Under this condition, SMC chooses the cacheline with most significant address bit matching the NS bit (i.e. == 0) – NS == 1, and most significant bit of the tag matches the NS bit value (i.e. == 1).
DII	• NS bit arrives via CMDreqs. • NS bit is forwarded as AxPROT[1] onAXI.
Proxy Cache	• Coherent accesses: AxPROT[1] bit must match the most significant bit of the tag (NS bit is stored as the most significant addressbit). • Snoops: NS bit in SNPreq must match the most significant bit of the tag (NS bit is stored as the most significant addressbit).

	• Cast out from SMC: Most significant address bit in the address tag of the cache line is sent on AxPROT[1] bit on AXI. The address itself on AXI is the tag address with the most significant tag bit set to 0 (or stripped off, as needed). • Non-coherent accesses: Same behavior as SMC.

12.2 Non secure attribute of memory regions

As discussed in section 5.4.6 Definition of a Memory Address Map, Agent interfaces for Ncore 3 implement address space decoding, transactions will be directed to a target for completion based on the aperture within the global address space.

Each aperture is described by a set of 3 General Purpose Region Definition Registers, two registers are required to define the base address of a region. The third register, GPRARx, defines the region attributes.

Each aperture may have a security property encoded in the NSX[1:0] field. NSX[0] is the non-secure attribute of the memory region while NSX[1] has been reserved for later use in Arm v9.0 to support REALM and ROOT security state.

When the region is secure (NSX[0] is 0):

Only secure transactions (with the NS bit on the agent interface set to zero) shall be accepted. All non-secure transactions shall be terminated with an error response, the illegal access shall be reported by the error handling system within Ncore.

When the region is unsecure NSX[0] is 1: All transactions will be accepted by the AIU.

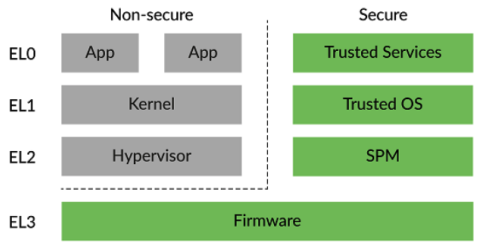
The NSX[0] bit shall be initialized to "high" (= 1'b1) for the GPRS defining the address window mapping the SysDII and the BootRegion. This allows any transaction to be forwarded to these agents. An alternate implementation may chose to ignore the NS bit for these address spaces.

12.3 Background: New Security states

TrustZone was introduced in Armv6 and provides the following two Security states:

- Secure state
- Non-secure state

The following diagram shows the two Security states in AArch64 with the software components that are typically found in each Security state:

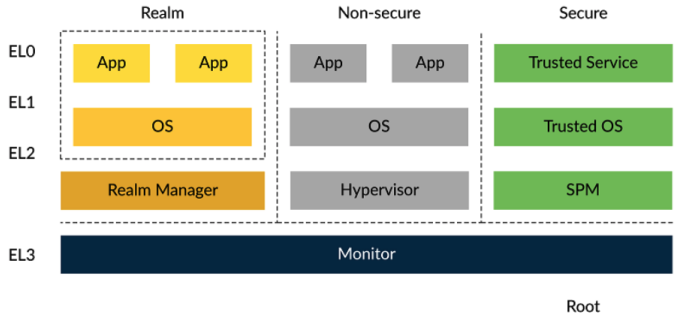


The architecture isolates software running in Secure state from software running in Non-secure state. This isolation enables a software architecture in which trusted code runs in Secure state and is protected from code in Non-secure state.

Realm Mode Extensions is part of Armv9, RME builds on the Arm TrustZone technology, it extends the above model, and provides the following four Security states:

- Secure state
- Non-secure state
- Realm state
- Root state

We have reserved one bit in the NSX[1:0] field to support this feature in the future without having to move other register bits around. The following diagram shows the Security states in an RME-enabled PE, and how these Security states map to Exception levels:



13 Trace and Debug

This section describes the trace and debug architecture for Ncore. Tracing can help localizing problems and errors in a NoC which may result from functional and performance anomalies. It requires taking a snapshot of traffic in a live system with temporal order to understand what is going on. A system in general will need to implement following functionality:

- Label: Identify and select transactions based on a set criterion like address, target etc.

- Order: Identify temporal ordering of transactions, this requires a time stamp.
- Capture: Take a snapshot of the labeled transaction at a feasible location.
- Aggregate and access: Accumulate all captured transactions at a central location and provide SW access.
- Analysis: Software is required to access the captured information and analyze it.

This specification is specified with restrictions to minimize Maestro software changes and not to add any new wires to the network, this is more of an add on to current Ncore system which does have some limitations.

13.1 Register offset

Debug and trace registers described in this document are located at 0x900 offset within each Ncore unit.

13.2 Detailed Description

A top-level overview of the system with trace and debug capability is shown in Figure 13-1. Existing data network is used to transport trace information. The Trace trigger block is present in all the AIUs. The capture block is present in all AIUs, DMIs and DIIs. DCE and DVE do not participate in Trace capture. Note that DCE does not connect to the data network. Trace information from other units can be used to post process and build DCE and DVE trace. The trace trigger and capture blocks together fill in the functionality of label, order, and capture. Trace accumulate block is centrally located in DVE; it can be accessed via CSRs. It fills in the functionality of aggregate and access, note that it also participates in ordering transactions. The Analysis functionality falls within the software domain and is not specified in this document for now.

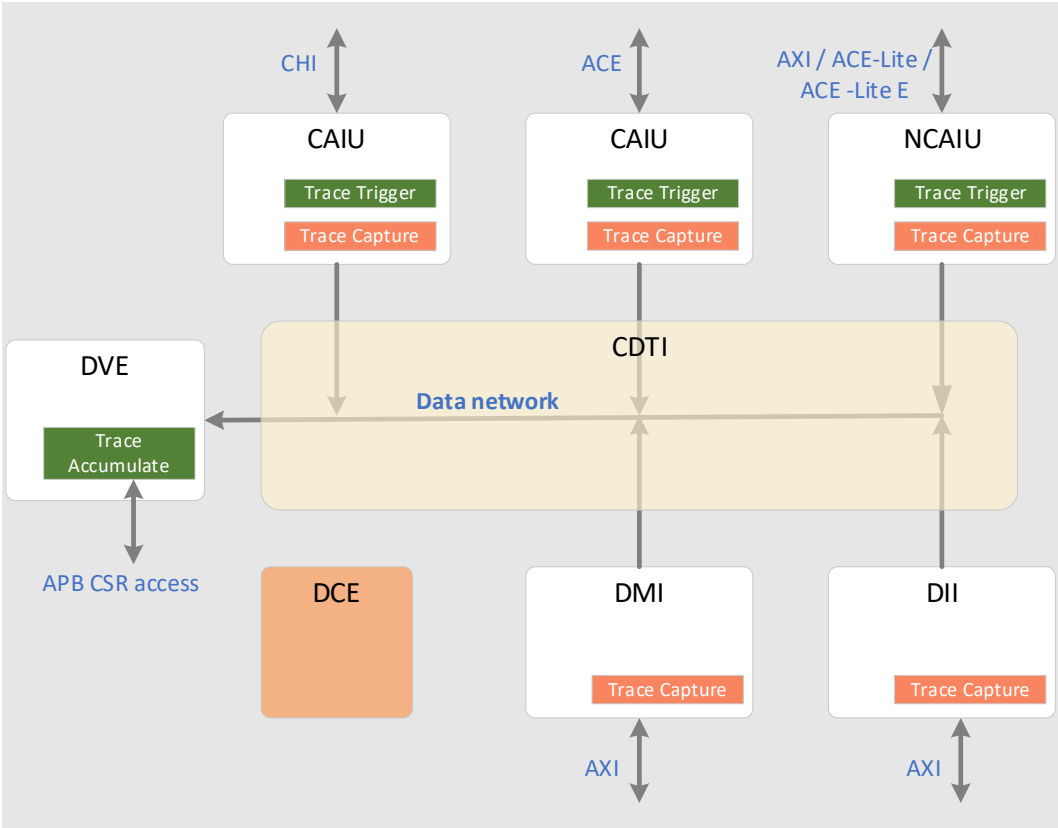


FIGURE 13-1 TRACE AND DEBUG OVERVIEW

13.2.1 Trace Trigger

The trace trigger block is responsible for identifying transactions to be captured and marking them. Ncore protocol has a “Trace Me” field in its messages. Ncore units shall propagate this trace signaling appropriately in all derived messages originating from a transaction that is marked to be traced (note that SysReq and SysRsp are not traced and will have their trace me field tied to zero). Furthermore, the trace signal shall be propagated on interfaces that support trace signaling based on the specific requirements of the interface (today this includes CHI-B and ACE-Lite E/ ACE5-Lite).

There are two classifications of marking a transaction to be traced.

1. Master initiated trace, this applies to CHI and ACE-Lite E (or 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.

13.2.1.1 Master Initiated Trace

CHI and ACE-Lite E (ACE5-Lite) interfaces have trace signaling capability, the master can identify transactions that are to be traced by appropriate interface signaling. Ncroe shall honor this signaling and meet the interface requirements. In the case of ACE5-Lite the more conservative approach must be

taken where all responses on the native interface must propagate the trace signal, furthermore in the case of write channel the trace signaling on W cannel is ignored and trace signaling on AW is taken into account.

13.2.1.2 Ncore Initiated Trace

All AIUs in the Ncore system including those that support trace signaling shall 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 number of CSR sets can be present as specified at build time by the parameter nTraceRegisters. Different trace options are described below, in a single set of register all the options below can be enabled together, they are checked with an AND option i.e. that is 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. Upto 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/ACE-Lite E. 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. The target type is specified using HUT and HUI. In the case of DII, HUI indicates DII NunitID and in the case of DMI, HUI indicates MIG number. One target type can be specified per register set.

13.2.1.3 Trace Control Register (xTCTRLR)

Bits	Name	Access	Reset	Description
0	Trace Signal	R/W	0x1	Trace using the native interface trace signaling (only present if the Native interface supports trace else it is reserved)
1	Address Range	R/W	0x0	Trace if there is a match on the address range specified
2	Op code	R/W	0x0	Trace if there is a match on the op codes specified. (Does not apply to AXI NCAIU, must be reserved in the case)
3	Memory attribute	R/W	0x0	Trace if there is a match on the memory attribute specified
4	User bits	R/W	0x0	Trace if there is a match on the user bits specified (Reserved if user bits do not exist in the configuration)
5	Target type	R/W	0x0	Trace if there is a match on the target type
6	HUT	R/W	0x0	Specifies target type DMI or DII
11:7	HUI	R/W	0x0	Specifies the HUI (NunitID for DII and MIG number of DMI)
18:12	Reserved	RO	0x0	

23:19	Size	R/W	0x0	Address range size, 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
31:24	Memory Attribute	R/W	0x0	Specifies the memory attribute: **CHI:** bits 31:28 are MemAttr field. bits 27: 24 are reserved. **AXI/ACE/ACE-Lite/ACE-Lite E :** Bits 31:28 are AxCache field. Bits 27:26 are for AR and AW respectively, if both are set match is done on both AR and AW. Bits 25:24 are reserved

13.2.1.4 Trace Base Address Low Register (xTBALR)

Bits	Name	Access	Reset	Description
31:0	Base address LO	R/W	0x0	Lower order bits 43:12 of the base address of the region

13.2.1.5 Trace Base Address High Register (xTBAHR)

Bits	Name	Access	Reset	Description
7:0	Base address Hi	R/W	0x0	Higher order bits 51:44 of the base address of the region
31:8	Reserved			

13.2.1.6 Trace Op Code Register (xTOPCR0)

Bits	Name	Access	Reset	Description
14:0	Op code 1	R/W	0x0	Op code number 1 (mapping defined based on Native interface, MSB unused bits are RO reserved)
15	Valid 1	R /W	0x0	Valid bit for op code 1
30:16	Op code 2	R/W	0x0	Op code number 2 (mapping defined based on Native interface, MSB unused bits are RO reserved)
31	Valid 2	R /W	0x0	Valid bit for op code 2

13.2.1.7 Trace Op Code Register (xTOPCR1)

Bits	Name	Access	Reset	Description
14:0	Op code 3	R/W	0x0	Op code number 3 (mapping defined based on Native interface, MSB unused bits are RO reserved)
15	Valid 3	R /W	0x0	Valid bit for op code 3
30:16	Op code 4	R/W	0x0	Op code number 4 (mapping defined based on Native interface, MSB unused bits are RO reserved)
31	Valid 4	R /W	0x0	Valid bit for op code 4

13.2.1.8 Trace User Bits Register (xTUBR)

Bits	Name	Access	Reset	Description
31:0	User bits	R/W	0x0	Configured number of user bits rest of the field is reserved

13.2.1.9 Trace User Bits Mask Register (xTUBMR)

Bits	Name	Access	Reset	Description
31:0	User bits mask	R/W	0x0	User bits mask register, set 1 for bits that need to be used for comparison.

13.2.2 Trace Capture

All AIUs, DMIs and DIIs shall support trace capturing capability. The block snoops SMI interface and captures messages that have the TraceMe field set. It captures non-data request, response and data messages (this includes all NDP header fields that are of non-zero width and NDP payload); actual data within the data message is not captured. Which SMI ports are to be snooped and captured is configurable via CSR settings. Multiple SMI ports can be snooped and captured simultaneously. The capture block has a capture buffer that is sized based on the parameter nUnitTraceBufSize, this parameter specifies the number of 64-byte entries in the buffer. The captured data packed into DTWs, once enough data is captured to build a single DTW, it is sent out on the SMI data network to the DVE in the system. If enough data is not accumulated to build a single DTW then a DTW with at-least a single concerto message and padding must be sent out once a timeout is reached (the timeout value is implementation defined example 12bit counter). If the capture buffer is full then no more messages are captured until the buffer drains to accommodate at least one more DTW.

The DTW data format example is shown in Figure 13-2. The example is for a 128-bit bus width. The packet always starts with a 32-bit free running counter time stamp, followed by the captured concerto message. As shown multiple full concerto messages can be packed in a single a DTW. The DTW may end with padding of 0s if a full message cannot be packed into the DTW.

The capture block will give out two events. These events indicate SMI messages that were dropped and captured in a clock cycle respectively. As there can be upto 8 SMI messages that are dropped or captured, these event maybe 3 bits wide each.

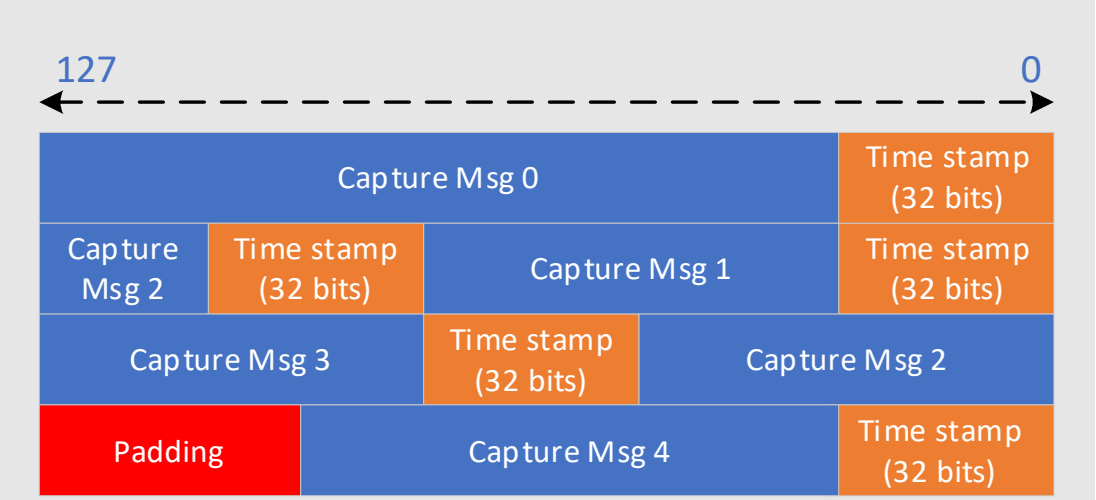


FIGURE 13-2 TRACE DTW DATA BUILD FORMAT

The DtwRsp cm_status field is used to synchronize the Ncore unit local time stamp counter with the DVE time stamp counter. This filed is as described inTable 13-1.

TABLE 13-1 DTWRSP CM_STATUS DESCRIPTION

Field	Bits	Name	Description
cm_status	7	Signe	Positive (0) or negative (1)
	6:0	Value	Value by which the counter needs to be adjusted, this maps to counter bits [10:4] of the local counter, bits [3:0] are zero.

The general concept is to have a feedback loop with a small correction and programable increment, as shown in Figure 13-3 master timeout stamp counter (MTSP) is compared with the unit timeout stamp counter (UTSC). The resultant error is multiplied with a gain and the UTSC increment value is updated accordingly.

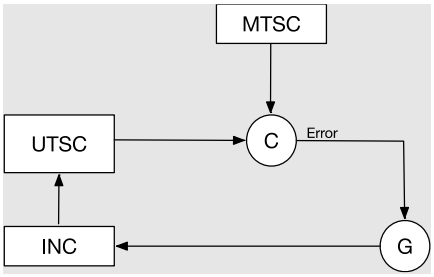


FIGURE 13-3 COUNTER SYNC FEEDBACK LOOP

C in the comparator of MTSC (counter in DVE) and UTSC (the local counter); comparison happens in DVE and error is sent back in DtwDbgRsp. G is the gain factor which is multiplied with the error and is used to correct INC (increment value of UTSC)

In this implementation INC is a register of 12 bits (4 bits of integer and 8 bits of fraction). This value initially represents the clock frequency difference between DVE and the Ncore unit, it gets updated based on the feedback error. Example values are shown in Table 13-2. The error is represented by 8 bits in singed magnitude format where bit 8 is the sign and lower 7 bits is magnitude within DtwDbgRsp. The 7 magnitude bits are multiplied with the 4-bit gain from CSRs to give an 11-bit number. MSB 8-bits of this 11-bit number are treated as fractional value that is either added or subtracted to the INC value based on the sign of the correction.

TABLE 13-2 INCREMENT VALUE EXAMPLES

Clock ratio with DVE	Value in INC register
2	{4'b0010, 8'b0000_0000}
1	{4'b0001, 8'b0000_0000}
0.5	{4'b0000, 8'b1000_0000}
0.25	{4'b0000, 8'b0100_0000}
0.33	{4'b0000, 8'b1100_0000}

13.2.2.1 Capture Control Register (xCCTRLR)

Bits	Name	Access	Scope	Reset	Description
0	Ndn0 SMI TX	R/W	All	0x0	Ndn0 SMI Tx snoop and capture enable
1	Ndn0 SMI RX	R/W	All	0x0	Ndn0 SMI Rx snoop and capture enable
2	Ndn1 SMI TX	R/W	All	0x0	Ndn1 SMI Tx snoop and capture enable
3	Ndn1 SMI RX	R/W	All	0x0	Ndn1 SMI Rx snoop and capture enable
4	Ndn2 SMI TX	R/W	All	0x0	Ndn2 SMI Tx snoop and capture enable
5	Ndn2 SMI RX	R/W	All	0x0	Ndn2 SMI Rx snoop and capture enable
6	Dn0 SMI TX	R/W	All	0x0	Dn0 SMI Tx snoop and capture enable
7	Dn0 SMI RX	R/W	All	0x0	Dn0 SMI Rx snoop and capture enable
15:8	Reserved	RO	All	0x0	
19:16	gain value	R/W	All	0x2	4-bit gain value
31:20	Inc value	R/W	All	0x100	Time stamp counter increment value: top 4 bits are integer and lower 8 bits are fractional

13.2.3 Trace Accumulate

This block is present only in DVE and the main functionality is to accumulate incoming trace DTWs from different Ncore capture units. The capture buffer is sized based on the parameter nMainTraceBufSize. Each entry in the buffer is organized as shown inFigure 13-4 . It is a 72-byte entry with 64-byte payload (DtwReq data received from the capture block).

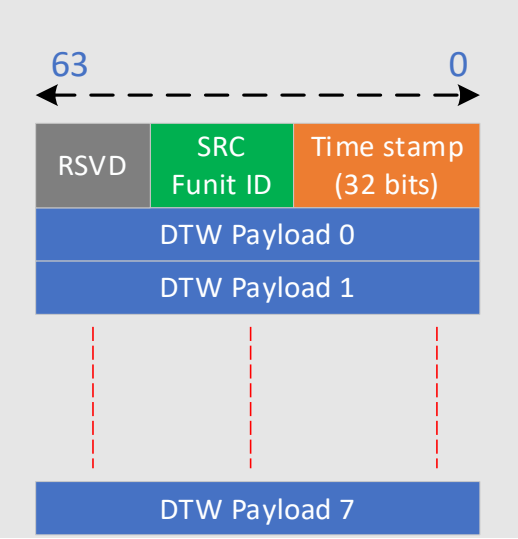


FIGURE 13-4 TRACE FORMAT

The time stamp here is the local free running counter at DVE. This counter is used to synchronize other Ncore unit counters. This is done by taking the first-time stamp from the received DtwReq and comparing it with bits [31:4] of the local counter, if the received time stamp value is bigger, then the cm status bit 7 of the DtwRsp is set to 1 else set to 0. Bits [10:4] of the local counter are placed into the cm status bits [6:0]. If the counter values match, then the cm status bits [6:0] are set to zeros.

The trace buffer shall be accessible via CSRs.

The accumulate block will give out an event that indicates DTW messages that are dropped per clock cycle.

13.2.3.1 Trace Accumulate Status & Control Register (xTASCR)

Bits	Name	Access	Reset	Description
0	Buffer empty	RO	0x1	Set when all the entries within the buffer are read and the buffer is empty
1	Buffer full	RO	0x0	Set when the buffer is full.
2	Circular buffer	R/W	0x0	When set, local buffer acts as a circular buffer, older messages are dropped once the buffer is full. When not set new messages are dropped when the buffer is full
3	Buffer clear	WSC	0X0	Write 1 to clear the complete buffer
4	Buffer read	WSC	0x0	Write one to trigger a read
31:5	Reserved	RO	0x0	

13.2.3.2 Trace Accumulate data header Register (xTADHR)

Bits	Name	Access	Reset	Description
7:0	Funit ID	RO	0x0	F unit ID of the trace capture Ncroe unit
30:8	Reserved	RO	0x0	

31	Valid	RSC	0x0	This is set once read data is valid, self clears on read

13.2.3.3 Trace Accumulate data Time stamp Register (xTADTSR)

Bits	Name	Access	Reset	Description
31:0	Time stamp	RO	0x0	Time stamp value from DVE free running counter

13.2.3.4 Trace Accumulate data (0-15) Register (xTAD0R)

These are 16 registers, where register 0 maps to data [31:0] and so on so forth.

Bits	Name	Access	Reset	Description
31:0	Capture data	RO	0x0	Capture data

13.3Debug and BIST disable pin

Resiliency requirements require that Ncore IP provide a pin that can disable enablement of BIST and debug & trace feature via CSRs. The user of the Ncore IP may decide to use this pin as an Effuse, pin on the chip or route it to a secure register.

This pin “<prefix>_en _debug_bist” must be present only when resiliency is enabled via the resiliency enable parameter.

Internal to Ncore the pin must be routed to all CAIUs, NACIUs, DMIs, DIIs and FSC. Note that this pin is not propagated to DCE.

13.4Debug APB port

To enable debug of a hung Ncore 3 system a slave APB port must be added to the CSR network that can access all the Ncore CSRs. At top level this port signals must be “<prefix>_debug_apb_<rest of the signal name>”.

Following APB port restrictions apply

- Fixed data bus width 32 bits
- Fixed address bus width of 20 bits
- Fixed access size of 4 bytes
- All access are 4 byte aligned.

This port is expected to be used for debug only, if same register is accessed concurrently via this debug APB port and the internal Ncore CSR accesses then the effect on the CSR is undefined. Ncore 3 does not guarantee any ordering between the two accesses.

14 Performance Monitoring

This chapter describes the performance counter architecture for Ncore.

A performance counter unit must be implemented in each Ncore unit i.e. AIUs, DCEs, DMIs, DIIs and DVE. This document goes over the architecture details and parameters of this unit and different events that are reported. The performance counter unit is not an optional unit and must always be present.

14.1 Parameters

Two unit level parameters are introduced: nPerfCounters and nLatencyCounters. They indicate for each type of counter the number to implement per Ncore 3 unit. See the parameter specification for more details on these parameters.

14.2 Detailed Description

The performance counter unit can track and report upto 31 events. The Ncore unit instantiating the performance unit specifies the events. These events maybe single bit or multi bit events and are specified in 14.4 section. The performance counter unit can be configured to implement either 4 or 8 counters.

Each counter is implemented as a 64 bit counter that is capable of counting upto 2 events at a time with following counter modes.

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.
5. **32-bit count:** In this mode the 64 bit counter acts as two 32 bit counters for two separate events

The 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
4. Use as 32 bit counter for second event

Both count registers freeze when an overflow is detected at either of the two registers, with the exception when they are configured to work in tandem as a 64 bit counter.

14.3 Register Definition

Different registers supported by the performance counter unit are described below. A set of registers exist per counter configuration i.e. each Ncore unit can either have 4 or 8 set of registers. Note that xMCNTCR and xLCNTCR is only one per Ncore unit.

14.3.1 Counter Value Register (xCNTVR)

Bits	Name	Access	Reset	Description
31:0	Count value	RO	0x0	Count value lower bits 31:0, freeze at overflow.

14.3.2 Counter Saturation Register (xCNTSR)

Bits	Name	Access	Reset	Description
31:0	Count saturation value	RO	0x0	Can be configured as follows (freeze at overflow): 1. Capture upper count value of 63:31 2. Use as IIR filter bits 3. Use as max/saturation value for accumulation events 4. Use as 32-bit counter, for count event second

14.3.3 Counter Control Register (xCNTCR)

Bits	Name	Access	Reset	Description
0	Count Enable	R/W	0x0	Write one to enable counting, this may be set by the “Master Count Enable” signal from DVE or can be set or reset by “Local Count Enable”
1	Count Clear	WSC	0x0	Write one to clear the counter (both CNTVR and CNTSR)
2	Interrupt Enable	R/W	0x0	Write one to enable rollover or overflow interrupt (gets ORed with correctable error interrupt)
3	Rollover/Overflow status	RO	0x0	One indicates overflow, sticky bit clears by clearing the counter. (When either xCNTVR or xCNTSR overflow, with exceptions as specified above)
6:4	Counter control	R/W	0x0	000 – Normal count 001 – AND count 010 – XOR count 011 – Instantaneous count (used for multi bit events like Active OTT entries) 100 – use as 32-bit counter (count event first)

				100 to 111 – reserved
9:7	SSR count	R/W	0x0	000 – Clear CNTSR (value of CNTSR is always zero) 001 – Capture upper 63:31 count in CNTSR 010 – use CNTSR as IIR filter (currently used only for active TT count) 011 – use CNTSR as max/saturation value (currently used only for CHI AIU interleave count) 100 – use as 32-bit counter (count event second) 101 to 111 reserved
12:10	Filter select	R/W	0x0	Low pass filter coefficients (IIR filter) 000 – 0 001 – ½ 010 – ¼ 011 – 1/8 100 – 1/16 101 – 1/32 110 – 1/64 111 – 1/128
15:13	Minimum stall period	R/W	0x0	Value is 2 ^ (minimum stall period) clock cycles valid range is 0 to 7
16:21	Count event second	R/W	0x0	Select the second count event (must be different from Count event first), value of zero is not valid, if zero no second event is selected.
22:23	Reserved			
24:29	Count event first	R/W	0x0	Select the first count event, value of zero is not valid, if zero no event is selected.
31:30	Reserved			

14.3.4 BW Counter Filter Register (xBCNTFR)

Bits	Name	Access	Reset	Description
19:0	Filter value	R/W	0x0	Value of user bits or Funit ID (as selected by filter select) to be filtered on. Must be LSB aligned
29:20	Reserved			
30	Filter select	R/W	0x0	Set to filter BW counting on Funit ID, else filter on user bits. This is valid only if Filter enable is set
31	Filter enable	R/W	0x0	Set to enable BW counting filtering else disabled

14.4 Performance events

14.4.1 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)
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 OTT entries
21		Reserved	
22	3	Captured SMI packets	Number of SMI packets Captured
23	3	Dropped SMI packets	Number of SMI packets dropped
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 evet counter	Counts number of events triggered by the Noc
28		Reserved	
29		Reserved	
30	1	Div 16 counter	Divide by 16 free running counter
31	1	Number of QoS starvations	Number of times QoS starvations occurred

14.4.2 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
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		Reserved	
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 OTT entries
21		Reserved	
22	3	Captured SMI packets	Number of SMI packets Captured
23	3	Dropped SMI packets	Number of SMI packets dropped
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 evet counter	Counts number of events triggered by the Noc
28		Reserved	
29		Reserved	
30	1	Div 16 counter	Divide by 16 free running counter
31	1	Number of QoS starvations	Number of times QoS starvations occurred

14.4.3 Proxy Performance events

Event #	Width	Name	Description
1	1	Cache read hit	
2	1	Cache write hit	
3	1	Cache snoop hit	
4	1	Cache eviction	
5	1	Cache no ways to allocate	
6	1	Cache fill stall	
7	1	Cache read stall	
8	1	Cache write stall	
9	1	Cache replay	
10	1	Cache read miss	

11	1	Cache write miss	
12	1	Cache snoop miss	
13		Reserved	
14		Reserved	
15		Reserved	
16		Reserved	
17		Reserved	
18		Reserved	
19		Reserved	
20		Reserved	
21		Reserved	
22		Reserved	
23		Reserved	
24		Reserved	
25		Reserved	
26		Reserved	
27		Reserved	
28		Reserved	
29		Reserved	
30		Reserved	
31		Reserved	

14.4.4 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
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	Count all DtwReqs that come into DMI
18	1	DtrReq event	Count all DtrReqs that are issued by 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
23	4	Dropped SMI packets	Number of SMI packets dropped
24	1	Address Collisions	Count number of address collisions on incoming transactions
25	1	Number of Merge events	Count number of DtwMergeMrds
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 counter
31	1	Number of QoS starvations	Number of times QoS starvations occurred

14.4.5 SMC Performance events

Event #	Width	Name	Description
1	1	Cache read hit	
2	1	Cache write hit	
3	1	Cache CMO hit	Operations like cleanInvlidate etc
4	1	Cache eviction	
5	1	Cache no ways to allocate	
6	1	Cache fill stall	
7	1	Cache read stall	
8	1	Cache write stall	
9	1	Cache replay	
10	1	Cache read miss	
11	1	Cache write miss	
12	1	Cache CMO miss	
13		Reserved	
14		Reserved	
15		Reserved	
16		Reserved	
17		Reserved	
18		Reserved	
19		Reserved	
20		Reserved	
21		Reserved	
22		Reserved	
23		Reserved	
24		Reserved	
25		Reserved	
26		Reserved	
27		Reserved	
28		Reserved	
29		Reserved	

30		Reserved	
31		Reserved	

14.4.6 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	Count all DtwReqs that come into DII
18	1	DtrReq event	Count all DtrReqs that are issued by DII
19		Reserved	
20	8	Active WTT entries	Number of active WTT entries
21	8	Active RTT entries	Number of active RTT entries
22	3	Captured SMI packets	Number of SMI packets Captured
23	3	Dropped SMI packets	Number of SMI packets dropped
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 counter
31		Reserved	

14.4.6.1 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
30	1	Div 16 counter	Divide by 16 free running counter
31	1	Number of QoS Starvations	Number of times QoS starvations occurred

14.4.7 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 STT entries
21		Reserved	
22	1	Captured DtwDbgReq packets	Number of DtwDbg packets captured
23	1	Dropped DtwDbgReq packets	Number of DtwDbg packets dropped
24		Reserved	
25		Reserved	
26		Reserved	
27		Reserved	
28		Reserved	
29		Reserved	
30	1	Div 16 counter	Divide by 16 free running counter
31		Reserved	

14.5Master Trigger

This acts as a single trigger to start/stop all performance counters in Ncore at approximately the same time. This is achieved by adding a Master count enable bit in DVE. When the bit is set it must be propagated to all Ncore units via a single wire. This signal is expected to be active high asynchronous signal, which is synchronized by the receiving Ncore unit. The Ncore unit will detect rising edge to start the counters and falling edge to stop the counters.

It is expected that SW will set this control register bit then read it to verify that it is set, wait from some x amount of time, and then reset it.

14.6Bandwidth Counters

Ncore provides Bandwidth counters in AIUs, DMIs and DIIs. These counters count in 64Byte data accuracy and at 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 they count following

- Read data bandwidth, this refers to data being read into the master/cache

- Write data bandwidth, this refers to data being written out of the master/cache, this may include evicted data.
- Snoop data bandwidth, this refers to data being moved out of the master/cache due to snoops

In DMIs/DIIs the count following

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

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

The above information can be used to calculate the effective BW. The counter must be disabled before reading them to get 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.

14.7 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 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 of 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 separate block as shown inFigure 14-1. In AIUs when the selected type of transaction is allocated to the transaction table (OTT) it is also allocated to the latency counter table, whereas in the case of DMI/DII it is allocated when the transaction is ready to be issued on the Native interface. The ID represents the transaction table (OTT/WTT/RTT) ID and counter is a 9-bit counter. The counter is started as soon as the entry is allocated. If all entries within the latency counter table are taken, then any new qualified transaction just does not participate in the latency histogram.

The counter increments every 2, 4, 8 or 16 clock cycle based on CSR configuration. If the counter hits its max value, then it stays saturated until deallocation and the saturated value is used for latency histogram binning.

In the case of an AIU an entry in latency counter table is deallocated at approximately the same time the corresponding entry is deallocated from the transaction table. In the case of DMIs/DIIs an entry in latency counter table is deallocated at approximately the same time corresponding response is received from the native interface. The deallocated entry counter value is then subtracted by the configured 8-bit offset, if the offset corrected value is negative then it is saturated downwards to zero. The quantizer takes the offset corrected value and increment appropriate histogram bin.

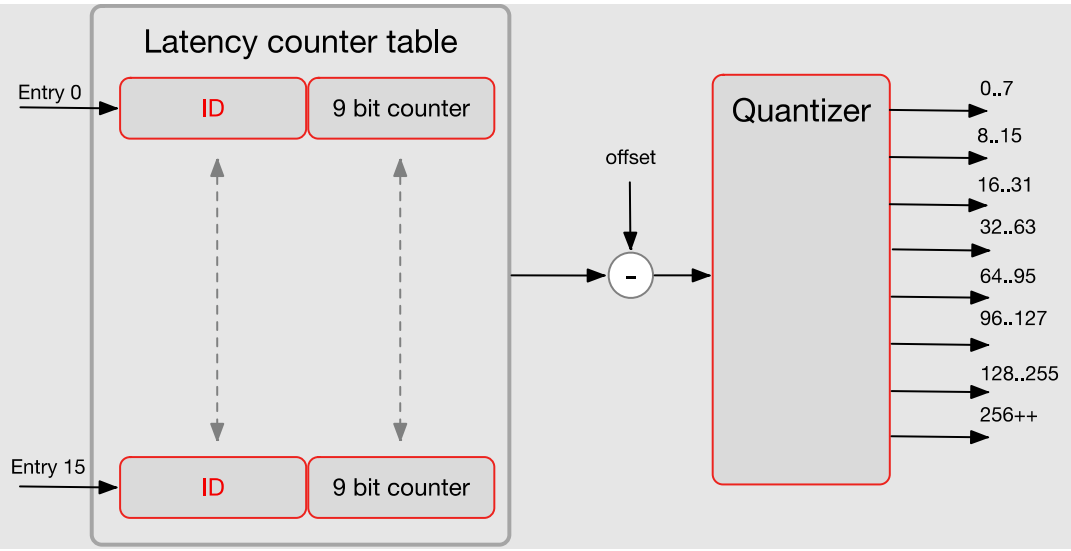


FIGURE 14-1 LATENCY HISTOGRAM BLOCK

To create a single histogram 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 pre scale** to desired pre scale 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 the 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:

- ***xCNTVR0***: Bin 0
- ***xCNTSR0***: Bin 1
- ***xCNTVR1***: Bin 2
- ***xCNTSR1***: Bin 3
- ***xCNTVR2***: Bin 4
- ***xCNTSR2***: Bin 5
- ***xCNTVR3***: Bin 6
- ***xCNTSR3***: Bin 7

15 System Software Guidelines

The following sections discuss the various requirements for system software.

15.1 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) in the case that one or more 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 needs to be followed.

15.1.1 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 non-coherent request. Boot-up sequence to be followed are:

- 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
- 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.

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.

15.1.2 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.

- 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 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 0xFF and address offset 0xFF8. This register will identify number of different Ncore components which include AIUs, DCEs, DMIs. DIIs and DVEs. RPs to Ncore component mapping is shown in NRS section
- Step through all Ncore component xIDR registers (register address offset 0x0) and xINFO registers (register address offset 0xFFC) starting form RPN 0.

Note:

1. registers access is restricted to address aligned, 8 byte non-coherent device access. Any other access format will return an error.
2. access to an undefined RPN will return an error.
3. Reserved registers within a valid RPN are treated as read as zero and write ignore.

15.1.3 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 following 2 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 31 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, The rules for PCIe transactions (see PCIe specification, Section 2.4 Transaction Ordering).
- If present, configure address translation registers <specify register name> in DMI/DII.

15.1.4 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

15.1.5 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

15.1.6 SMC configuration

Following steps must be followed to configure and enable SMC

- Read register DMIUSMCISR to verify both tag and data RAM are initialized
- Configure register DMIUSMCAPR to specify desired cache allocation policy

- 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 setup 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

15.1.7 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 is to be specified in number of cachelines and must be an integer multiple of 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

15.1.8 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.

15.2 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 compo- nents.
    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 bring back 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.

16 PPA

This section provides the Ncore 3 power, performance, area goals.

16.1 Performance goals

Overall, Ncore 3 aims to sustain the maximum theoretical memory bandwidth with hardware coherency without preventing the agents to use their full cache and without limiting their message throughput.

The theoretical aggregate (simultaneous read plus write) bandwidth for a 256-bit wide ACE or CHI interface is 32 Bytes x 1,6Ghz x 2 = 102,4 GB/s.
Ncore 3.4 shall sustain the max theoretical bandwidth for reads: 32 Bytes x 1,6Ghz = 51,2GB/s per AIU.
Ncore 3.4 shall sustain the max theoretical bandwidth for writes: 32 Bytes x 1,6Ghz = 51,2GB/s per AIU.
It does not sustain the aggregate bandwidth max theoretical bandwidth for a 256-bit wide interface.

Ncore 3.4 shall sustain these goals while scaling the number of agents participating. The round trip latency is becoming higher with larger system or when using slower memory devices. To sustain the bandwidth the Ncore 3 units CAUI, IOAIU and DMI shall sutain a large number of inflight transactions:

- CAIU and IOAIU supports up to 128 oustanding transactions
- DMI supports up to 128 oustanding reads and 64 oustanding writes

Scenari for performance verification tests include memory stream tests. The simulation results shall match the results obtained high level modelling running the corresponding benchmarks. The memory bandwidth is the bottleneck, the SMI links or Concerto units do not limit the memory bandwidth.

Other scenari stress the theoretical bandwidth:

- Agent throughput shall not be limited. It shall saturate its interface both for write data and read data channels
- Cache utilization of agents shall not be limited by Ncore 3 Snoop Filter size, organization and policy

Non coherent traffic performance shall also be optimized. Ncore 3.4 shall provide equivalent performance as FlexNoC for an AXI4 interface.

16.2 Timing goals

The product shall support a system clock frequency up to 1.6 GHz with a SVT, LVT and ULVT mix of transistor gate lengths and worst PVT/ Parasitic corner in 7nm technology node.

The CHI AIU shall support the option for an asynchronous interface.

17 Glossary

The following terms have a specific meaning in this document.

Term		Definition
Arteris	-	A NoC Company
AIU	-	Agent Interface Unit
iAIU	-	initiator AIU - request is started here
sAIU	-	snooped AIU - a transaction may result in one or more AIUs being snooped as part of the coherence protocol
aTag	-	Allocation Tag – associated with a memory location, stored in memory
NCore3	-	A coherent NoC provided by Arteris with AMBA interfaces and built-in caches.
BR	-	Boot Region
CAIU	-	Coherent Agent Interface Unit
CDTI	-	Control and Data Transport Interconnect
CMD	-	Command Message
CSTI	-	Control and Status Transport Interconnect
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
GPAS	-	General Purpose Address Space
MRD	-	Memory Read Message
NCAIU	-	Non-coherent Agent Interface Unit
NRS	-	Ncore Register Space
RB	-	Request Buffer
pTag	-	Physical Address Tag or Physical Tag , a memory location is accessed, the Requester uses both the address of the location and the tag value that it believes is associated with the location
PoC	-	The PoC (Point of [system] Coherency) is the point at which all blocks (for example, CPUs, DSPs, or DMA engines) which can access memory, are guaranteed for a particular address to see the same copy of a memory location. Typically, this will be the main external system memory.
PRD	-	Product Requirement Document
SMC	-	System Memory Cache
SNP	-	Snoop Message
SF	-	Snoop Filter
STR	-	State Reply Message
TSR	-	Technical Safety Requirement
UPD	-	Update Message
VC	-	Virtual Channel

18 Document References

18.1 Supported standards and specifications

- Arm CHI A/B/C
- ArmAXI-4
- Arm v8.2A, v8.3A, v8.4A, v8.5

18.2 Relevant documents and specifications

- ARM document *dsu_trm_100453_0401_03_en*
- Ncore 3.4 FSC Micro-Architecture Specification
- Ncore 3.4 Architecture Parameter Specification
- Arteris Symphony Concerto-C Rev. 0.95.1

19 Notes

Notes .....

20 Opens

Questions/Feedback/Need to discuss:……