Ncore 3.6 Exclusive Monitor Micro-Architecture

Release: 3.6

Rev: 0.53, October 9, 2023

Arteris® Ncore 3.6 Exclusive Monitor Micro-Architecture

Copyright © 2020 Arteris© or its affiliates. All rights reserved.

**Release Information**

|  |  |  |  |
| --- | --- | --- | --- |
| *Version* | *Editor* | *Change* | *Date* |
| 0.1 | MF/MK | Initial Document template created | 10/24/2019 |
| 0.2 | BM | Initial version | 04/25/2023 |
| 0.2 | ET | Added DII | 5/1/2023 |
| 0.21 | BM | Changed to the tables per feedback from Chengchung. Added existence parameter. Proposed a register for Timeout control | 5/1/2023 |
| 0.22 | ET | DII Updates |  |
| 0.5 | BM | Document reviewed with engineering. Made modification per feedback. Name of the parameter is changed, timeout is removed, replacement is now round robin, updated cm\_status table to remove illegal scenario |  |
| 0.51 | BM | Updates table to clarify responses |  |
| 0.52 | BM | Adds a port table and clarify the behavior in case of a DtwNullData | 8/10/2023 |
| 0.53 | BM | Clarifies the behavior when the monitor matches address and unitId but on different entrees | 10/09/2029 |
| Legend: | MK | Mohammed |  |
|  | MF | Michael Frank |  |
|  | BM | Benjamin Madon |  |
|  | ET | Eric Taylor |  |
|  | Xx | Whoever else edited this document |  |

**Confidential Proprietary Notice**

This document is CONFIDENTIAL AND PROPRIETARY to Arteris, Inc. or its applicable subsidiary or affiliate (collectively or as applicable, “Arteris” or “Arteris IP”), and any use by you is subject to the terms of the agreement between you and Arteris IP or the terms of the agreement between you and the party authorized by Arteris IP to disclose this document to you.

This document is also protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arteris IP.  **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.  You are prohibited from altering or deleting this notice from any use by you of this document.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information: (i) for the purposes of determining whether implementations infringe any third party patents;(ii) for developing technology or products which avoid any of Arteris IP's intellectual property; or (iii) as a reference for modifying existing patents or patent applications or creating any continuation, continuation in part, or extension of existing patents or patent applications; or (iv) for generating data for publication or disclosure to third parties, which compares the performance or functionality of the Arteris IP technology described in this document with any other products created by you or a third party, without obtaining Arteris IP's prior written consent.

THIS DOCUMENT IS PROVIDED “AS IS”.  ARTERIS IP PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arteris IP makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.  This document may include technical inaccuracies or typographical errors.  Arteris IP makes no representations or warranties against the risk or presence of same.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARTERIS IP BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARTERIS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be solely responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arteris IP's customers is not intended to create or refer to any partnership relationship with any other company. Arteris IP may make changes to this document at any time and without notice. If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this 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.

The Arteris IP name and corporate logo, and words marked with ® or ™ are registered trademarks or trademarks of Arteris (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arteris IP's trademark usage guidelines, available from Arteris IP upon request by emailing to contracts@arteris.com.

Copyright © 2020 Arteris Inc. or its applicable subsidiary or affiliate. All rights reserved.

**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](http://www.arteris.com/)

Table of Contents

1 Introduction 12

2 DII 13

2.1 New Parameters 13

2.2 Location of the exclusive monitor 13

3 DMI 14

3.1 New Parameters 14

3.2 Location of the exclusive monitor 14

4 DCE Exclusive monitor 15

4.1 Module interface and parameter definition 15

4.2 Behavior of the exclusive monitor used in DCE 16

4.3 Note on current behavior of the exclusive monitor block used in DCE 20

4.3.1 On the behavior of the monitor for non-exclusive store 20

4.3.2 On the number of bits to be used for the address comparison 20

4.3.3 On the number of monitors 21

5 DII implementation 22

5.1 Behavior of the monitor 22

6 DMI implementation 24

6.1 Behavior of the monitor 24

6.2 Modification in the write buffer 25

6.3 Note on mixed coherent/non-coherent domain 26

6.4 Impact on requests going to the native Axi interface 26

6.5 Note on 128 bytes exclusive 27

7 Verification 27

7.1.1 Stimulus restriction 27

8 Opens 29

Table of Figures

[Figure 1. Updated diagram for DII in NCore 3.6 13](#_Toc133843766)

[Figure 2 Updated protocol control diagram for Ncore 3.6 15](#_Toc133843767)

Table of Tables

[Table 1 Monitor Updates of the DCE exclusive monitor 18](#_Toc133843759)

[Table 2 Monitor update required in DII 23](#_Toc133843760)

[Table 3 CMStatus depending on the success or failure of the exclusive transaction 23](#_Toc133843761)

[Table 4 Monitor updates required in DMI 24](#_Toc133843762)

[Table 5 CMStatus depending on the success or failure of the exclusive transaction 26](#_Toc133843763)

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.

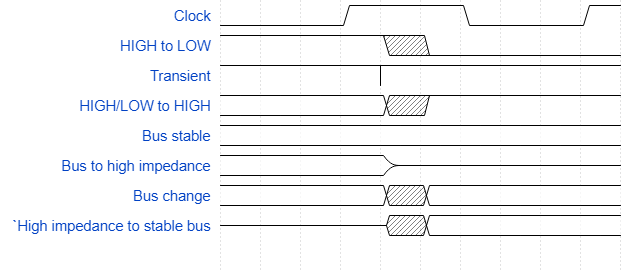
SMALL CAPITALS

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.

# Introduction

Ncore 3.4 does not have exclusive monitor for non-coherent access and simply forward the relevant information downstream on its AXI interface. Ncore 3.6 will implement exclusive access monitor at the point of serialization in DMI and DII.

The initial plan was to use the DCE monitor, but it is not suited for the requirement of the monitor in DMI as will be clear from the DCE behavior section. We will follow the same approach has the one used in symphony instead either by reusing the block with modification of by creating a new one inspired by it.

# DII

## New Parameters

The dii.tachl.cpr will take one new parameters which will be provided by Maestro :

* **nExclusiveEntries**: defines the number of monitors. Min value : 0, Max Value : 8. Default value : 0. A value of 0 means no exclusive monitor will be instantiated. It is expected by maestro to provide it to DII.

## Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Direction | Width | Description |
| valid | Input | 1 | Needs to be asserted to have a request update the exclusive monitor |
| lock\_op | Input | 1 | Needs to be mapped to the es bit of the command |
| msg\_type\_load | Input | 1 | Needs to be asserted when the request is a write |
| msg\_type\_store | Input | 1 | Needs to be asserted when the request is a store |
| address | Input | wAddr | Needs to be mapped to the addr field of the command |
| mpf2 | Input | wMpf2 | Needs to be mapped to the mpf2 field of the command |
| Initiator Id | Input | wInitiatorId | Needs to be mapped to the initiator\_id field of the command |
| security | Input | wSecurity | Needs to be mapped to the ns field of the command |
| ex\_pass | Output | 1 | Asserts when the transaction PASS (see table in [Behavior of the monitor](#_Behavior_of_the)) |
| Exmon\_event | Output | 1 | Signal to record a clear event in the monitor |

## Location of the exclusive monitor

The point of serialization inside the DII block is at the output of the skid buffer so this is where transactions will be passed through the exclusive monitor. The monitor will be put inside the Command Decode block which is responsible for driving the OTT entry information from the next command to be issued from the skid buffer.

The Command Decode block will store the pass/fail information inside the WTT/RTT entry of that transaction. The pass-fail information will then be propagated from the entry to other blocks to correctly drop write transactions and drive Concerto Message responses.

**Note: With the addition of the exclusive monitor the DII will need to guarantee ordering between read and write channels meaning a PCIe initiator cannot talk to a DII with an exclusive monitor enabled.**

The updated DII diagram is below:

Diagram

Description automatically generated

Figure . Updated diagram for DII in NCore 3.6

# DMI

## New Parameters

The dmi.tachl.cpr will take two new parameters which will be provided by Maestro :

* **nExclusiveEntries** : defines the number of monitors. Min value : 0, Max Value : 8. Default value : 0. A value of 0 means no exclusive monitor will instantiated. It is expected by maestro to provide it to DMI.

## Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Direction | Width | Description |
| valid | Input | 1 | Needs to be asserted to have a request update the exclusive monitor |
| lock\_op | Input | 1 | Needs to be mapped to the es bit of the command |
| msg\_type\_load | Input | 1 | Needs to be asserted when the request is a write |
| msg\_type\_store | Input | 1 | Needs to be asserted when the request is a store |
| address | Input | wAddr | Needs to be mapped to the addr field of the command |
| mpf2 | Input | wMpf2 | Needs to be mapped to the mpf2 field of the command |
| Initiator Id | Input | wInitiatorId | Needs to be mapped to the initiator\_id field of the command |
| security | Input | wSecurity | Needs to be mapped to the ns field of the command |
| ex\_pass | Output | 1 | Asserts when the transaction PASS (see tablein [Behavior of the monitor](#_Behavior_of_the_1)) |
| Exmon\_event | Output | 1 | Signal to record a clear event in the monitor |

## Location of the exclusive monitor

The point of serialization in DMI is located in the protocol control block, more precisely all non-coherent requests are serialized for the same address after they leave the CmdReq skid buffer. Exclusive reads leaving the skid buffer will set the exclusive monitor while write will access it to decide if it is a pass or a failure. It will use the processor id contained in the mpf2 field as well as the initiator id to uniquely identify the logical processor. The updated protocol control diagram is shown below and the exclusive monitor location is circled in red.



Figure Updated protocol control diagram for Ncore 3.6

# DCE Exclusive monitor

## Module interface and parameter definition

The module existing in DCE for Ncore 3.4 called dce\_excl\_mon.tachl (and dce\_excl\_mon\_cam.tachl) will be made a library elements called excl\_mon.tachl and excl\_mon\_cam.tachl and stored in hw-lib.

Excl\_mon is the top level and takes the following parameters which will take the same value as the top level parameter defined in DMI :

var assertOn         = u.getParam('assertOn');

var wAddr            = u.getParam('wAddr');

var wSecurity        = u.getParam('wSecurity');

var wCacheLineOffset = u.getParam('wCacheLineOffset');

var nTaggedMonitors  = u.getParam('nTaggedMonitors');

var nProcs           = u.getParam('nProcs');     // = total number of Processors

The ports are defined as follows:

u.port('input',  'clk',         1);

u.port('input',  'reset\_n',     1);

u.port('input',  'valid',             1);

u.port('input',  'lock\_op',           1);

u.port('input',  'msg\_type\_load',     1);

u.port('input',  'msg\_type\_store',    1);

u.port('input',  'address',           wAddr);

    /\* istanbul ignore else env ncore\_3p2, ncore\_3p4 \*/

    if (wSecurity > 0) {

u.port('input',  'security',          wSecurity);

    }

u.port('input',  'proc\_id\_onehot',    nProcs);

u.port('output', 'mon\_status',  1);

u.port('output', 'mon\_valid',   1);

The inputs will be connected as follows:

* Valid : connected to the valid of the skid buffer output fifo
* Lock\_op : connected to the es attribute of the output of the skid buffer output fifo
* Msg\_type\_load : set to one if a read
* Msg\_type\_store : set to one if a write
* Address, security : connected to the address, security going to the write buffer
* Proc\_id\_onehot : One hot representation of {mpf2,initiator\_id}[wnProcs-1:0], where wnProcs = clog2(nProcs). Note that the mpf2 field could be larger than strictly required and the processor id will always use the LSB bit. This will guarantee that we ignore the correct bits of the mpf2 field.

The output will be connected as follows :

* Mon\_valid and mon\_status will be “and” together and written in a new field called exPass stored in the write buffer.

## Behavior of the exclusive monitor used in DCE

The exclusive monitor in DCE is made of two distinct monitors : a basic monitors with as many monitors as there are LP, and a tag monitor which size is defined at design time by a parameter. The goal of the tag monitor is to allow multiple exclusive sequence to different address at the same time.

The behavior of the block used in DCE can be summarized by the table below, where the green rows are for an exclusive store pass, the three red rows are for an exclusive fail. The presence of the basic monitor allows for every LP to have an active exclusive sequence at the same time.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of request | Basic monitor state | Tag Monitor state | Tag Monitor status (full/Not full) | Basic Monitor update | Tag Monitor update |
| Load Ex | Not set | Match | X | Do nothing | Update valid for the matched tag for this LP, invalidate any other valid for this LP on other tags. |
| No Match | Full | Set for this LP | invalidate any other valid for this LP on other tags. |
| Not full | Do nothing | Write the Tag into an empty entry and set the valid for this LP and invalidate any other valid for this LP on other tags. |
| Set | Match and valid for the tag | x | illegal | Illegal |
| Match and not valid for the tag | x | Clr for this LP | Update valid for the matched tag for this LP, invalidate any other valid for this LP on other tags. |
| No Match | Full | Do nothing | Do Nothing |
| Not full | Clr for this LP | Write the Tag into an empty entry and set the valid for this LP. |
| Store Ex | Not Set | Match but not valid | X | Clear for this LP | Set the valid for this LP |
| Match and valid | Set for this LP  And clear every other LP | Clear all valid for this tag, and for this LP |
| No Match | Full | Set for this LP | Clear all valid for this LP. |
| Not full | Clear all | Write the Tag into an empty entry and set the valid for this LP and invalidate any other valid for this LP on other tags. |
| Set | No Match | x | Clear all other LP | Clear all valid for this LP |
| Match and not valid for this LP | x | Clear this LP | Clear all valid for this LP |

Table Monitor Updates of the DCE exclusive monitor

Description from DCE micro-architecture :

*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 three 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. Furthermore, the valid bit for every other processor are cleared in the basic 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.*

## Note on current behavior of the exclusive monitor block used in DCE

### On the behavior of the monitor for non-exclusive store

My current understanding is that in case of a coherent exclusive sequence, a store to a location for which an exclusive sequence was started will update the monitor on the agent side (it will receive a snoop to invalidate the cacheline), consequently it will fail in the initiator despite passing in DCE. This is the reason why the address check on the POC monitor is optional, if exclusive access are not so frequent that two agent will require them at the same time, a basic monitor will work just fine the majority of the time.

Such mechanisms do not exist for the non-coherent address space.

Also, note that POC monitor specifically mentions exclusive stores clear other LP monitors while the system monitor do not. Specifically, the AXI spec does not say anything about the kind of write which can update the location for which an exclusive sequence is pending: “ *The exclusive access monitor records the address and* ***ARID*** *value of any exclusive read operation. Then it monitors that location until either a write occurs to that location or until another exclusive read with the same* ***ARID*** *value resets the monitor to a different address.” Version H.c of the AMBA AXi specifications.*

For all these reasons, I believe that the grey rows from “Table 1 Monitor Updates” must be implemented in DMI. This also renders the basic monitor useless (since any write would clear it).

### On the number of bits to be used for the address comparison

The AXI specification suggests that the entire address must be used while the CHI specification specifically says that any number of bits can bit used for snoopable address space in the PoC. But it also says for the system monitor “The minimum number of bytes to be monitored during an exclusive operation is defined by the transaction size. The System monitor can monitor a larger number of bytes, up to 64 bytes, which is the maximum size of an Exclusive access.” Which seems to suggest that 64 bytes is the maximum granularity for the monitor.

The AXI specification has a similar sentence replacing 64 bytes by 128 bytes. While it seems that DCE could use any number of buts, it is using 64 bytes granularity, and I think the same thing must be done in DMI.

### On the number of monitors

When there are less monitors than the number of processors there could be starvation issues. How do we make sure that the same logical processor does not acquire the monitor over and over again? Should a more advanced mechanism for the monitor allocation be used?

Reading the AXI specification, it feels like every LP should have its own monitor. It is not explicitly mentioned though. The CHI specification spends most of the time talking of the snoopable address space, and while it says that it can use any number of tagged monitors it also specifically says that “The monitor must support the parallel monitoring of all exclusive-capable Logical Processors in the system.” But this only applies to snoopable address.

# DII implementation

## Behavior of the monitor

There will be an entry for each exclusive transaction flow. Each entry will consist of an “ID” that must match constructed from the combination of the mpf2 field and the initiator ID. In this case the mpf2 field will contain the AXID which will be the same ID used for the native layer transaction.

The DII will only see non-coherent exclusives.

To support exclusives the DII needs two changes:

1. To resolve RAR and WAW hazards the AxID allocation is not based of address. This enforces that all transactions to the same address will be ordered within the read and write channel. This change is for all DII configurations.
2. For DII configurations with an exclusive monitor the DII adds RAW and WAR hazard checking against transactions with no ordering causing a head of line block at the exit of the skid buffer. This is because if transactions are outstanding with no ordering the DII will not enforce any cross table address hazards.
3. The DII will force all exclusive transactions to be at least RO ordering, and will keep EO if it is EO

**Note: the table below is the same as DMI.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of request | Match  ns + Addr | Match AxId, FunitId | Tag Monitor status (full/Not full) | Tag Monitor update | Monitor response |
| Load Ex | Match | Match | X | - If on the same entry: Do nothing  - If on different entries: Replace the Tag on the ID that matched | PASS |
| No Match | Full | Replace an existing entry. Selection is round robin. | PASS |
| Not Full | Set a new entry | PASS |
| No Match | Match | X | Replace the Tag | PASS |
| No Match | Full | Replace an existing entry. Selection is round robin. | PASS |
| Not Full | Set a new entry | PASS |
| Load | x | x | x | Do nothing | x |
| Store | Match | x | x | Clear all the entries with matching address | x |
| No Match | x | x | Do nothing | x |
| Store Ex | Match | Match | X | Clear all the entries with matching address | PASS |
| No Match | Do Nothing | FAIL |
| No Match | x | x | Do nothing | FAIL |
| x | Do nothing | FAIL |

Table Monitor update required in DII

For load exclusive, the ID match has priority over address match. Which means that if an entry matches for address but not for ID and another one matches for ID but not for address the expected behavior is to replace the tag in the entry which matched ID.

If there is a fail for an exclusive store then that command will be dropped and the cmstatus will correspond to the following table. The DII will be responsible for dropping the command, data, and properly completing the protocol.

When the exclusive is a fail the DtwRsp will always be an early response and will contain a cm\_status indicating the success/failure of the exclusive operation. When it is a pass, the DtwRsp is a late response, meaning that the transaction must wait for the response from the native interface. The exclusive monitor status will be stored in the context table and will be used to generate the late response per the table below (See CCMP spec 4.5.3.1 CMStatus, and error architecture specification):

|  |  |  |  |
| --- | --- | --- | --- |
| Status | Native response | CMStatus[7:6] | CMStatus[5:0] |
| Pass | OKAY | 2’b00 | 6’b000001 |
| Pass | SLVERR | 2’b10 | 6’b000011 |
| Pass | DECERR | 2’b10 | 6’b000100 |
| Fail | OKAY | 2’b00 | 6’b000000 |

Table CMStatus depending on the success or failure of the exclusive transaction

# DMI implementation

## Behavior of the monitor

I suggest that we use a monitor structure closer to the one used in symphony where each monitor entry is used for one and only one exclusive sequence. Consequently, it will store the address, the ns bit, the AxId contained in the mpf2 field and the funitID. Assuming we do not support 128 byte exclusive transaction, I propose that we use 64 byte aligned address such that we do not need to store the size of the transaction. This way narrower access will always match wider access to the cost of more failure of the exclusive sequence, which I think is acceptable.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type of request | Match  ns + Addr | Match AxId, FunitId | Tag Monitor status (full/Not full) | Tag Monitor update | Monitor response |
| Load Ex | Match | Match | X | - If on the same entry : Do nothing  - If on different entries : Replace the Tag on the ID that matched | PASS |
| No Match | Full | Replace an existing entry. Selection is round robin. | PASS |
| Not Full | Set a new entry | PASS |
| No Match | Match | X | Replace the Tag | PASS |
| No Match | Full | Replace an existing entry. Selection is random. | PASS |
| Not Full | Set a new entry | PASS |
| Load | x | x | x | Do nothing | x |
| Store | Match | x | x | Clear all the entries with matching address | x |
| No Match | x | x | Do nothing | x |
| Store Ex | Match | Match | X | Clear all the entries with matching address | PASS |
| No Match | Do Nothing | FAIL |
| No Match | x | x | Do nothing | FAIL |
| x | Do nothing | FAIL |

Table Monitor updates required in DMI

For load exclusive, the ID match has priority over address match. Which means that if an entry matches for address but not for ID and another one matches for ID but not for address the expected behavior is to replace the tag in the entry which matched ID.

In the tables above a match can only occur if the entry is valid. The response indicated assumes that the native interface response is OKAY and not an error response (these cases will be described in 6.3).

I would recommend having the size set to the number of agents which are expected to use exclusives at the same time to limit the number replacements. It should be well communicated to the customer.

When the monitor is full, an entry will be selected for replacement. It will be selected round robin starting by entry 0. Every time an entry is replaced the entry number to be selected will increase by 1. When it reaches the last entry, it will reset to entry 0.

## Modification in the write buffer

As mentioned in the previous paragraph a new field exPass will be stored in the write buffer. At a later time, the corresponding DtwReq will arrive in DMI and it will be stored in the non-coherent write buffer. When it is selected for processing, the drop condition will look at this field. If it is set it will activate and the write will be dropped otherwise it will continue processing as before. When not an exclusive transaction the exPass field will always be set.

Current drop condition:

assign write\_prot\_drop = (write\_prot\_cm\_type == WR\_NDATA) |

\js if (useCmc) {

                         ((write\_prot\_cm\_type == WR\_CLEAN) & ~write\_prot\_ac);

\js } else {

                         (write\_prot\_cm\_type == WR\_CLEAN);

\js }

New drop condition :

assign write\_prot\_drop = (write\_prot\_cm\_type == WR\_NDATA) |

\js if (useCmc) {

                         ((write\_prot\_cm\_type == WR\_CLEAN) & ~write\_prot\_ac);

\js } else {

                         (write\_prot\_cm\_type == WR\_CLEAN)

\js } | ~exPass ;

## Protocol response update

First off, in the absence of an exclusive monitor, the behavior is identical to Ncore 3.4.

If an exclusive monitor is present, when the exclusive is a fail the DtwRsp will always be an early response and will contain a cm\_status indicating the success/failure of the exclusive operation. When it is a pass, the DtwRsp is a late response, meaning that the transaction must wait for the response from the native interface. The exPass field will be stored in the context table and will be used to generate the late response per the table below (See CCMP spec 4.5.3.1 CMStatus, and error architecture specification):

|  |  |  |  |
| --- | --- | --- | --- |
| Status | Native response | CMStatus[7:6] | CMStatus[5:0] |
| Pass | OKAY | 2’b00 | 6’b000001 |
| Pass | SLVERR | 2’b10 | 6’b000011 |
| Pass | DECERR | 2’b10 | 6’b000100 |
| Fail | NA | 2’b00 | 6’b000000 |

Table CMStatus depending on the success or failure of the exclusive transaction

Additionally, in the unlikely event an exclusive cmd write receives DtwNullData, the DtwRsp will be early with a CMStatus containing an OKAY response (6’b0), and the monitor will be cleared.

## Note on mixed coherent/non-coherent domain

The location of the monitor is such that a coherent write to the same location as an ongoing exclusive sequence will not invalidate the sequence. It is expected that if exclusive are used on mixed domain, software is in charge of making sure that only coherent or non-coherent request are issued to the location during an exclusive access.

## Impact on requests going to the native Axi interface

In the absence of an exclusive monitor in the configuration, the behavior is unchanged compared to Ncore 3.4. Otherwise, exclusive transactions going out on DMI native interface will not a use specific ID any longer. Instead, it will use the same ID as non-exclusive read and write requests. Ordering will rely on using the same axId for same address writes**. It becomes critical that a write following an exclusive store uses the same Id to guarantee ordering at the receiver.**

Read and write ordering inside DMI are guaranteed by checking hazard in the table i.e a dependent read will be blocked until the write finishes and a dependent write will be blocked until the read finishes. This mechanism happens both in the protocol control block between the write buffer and the read buffer as well as on the resource control block with pending reads and write on the native interface.

## Note on 128 bytes exclusive

The maximum size of exclusive transaction on AXI/ACE is defined as 128 bytes in the protocol specification. Since DMI can't handle transaction larger than 64 bytes, there are two possibilities:

* They are not supported, and the exclusive load will fail in the IOAIU. (OKAY response)
* We could set the monitor such that they monitor 128 bytes at a time. IOAIU would send two exclusive load and two exclusive write to the DMI hitting the same monitor. We would need an extra mechanism to make sure that nothing interrupts the sequence in the middle.

I propose that we do not support support 128 byte exclusive transaction in which case it will be the ioaiu responsibility to issue on OKAY response to the initiator and to convert the read into a normal read.

# Verification

## Stimulus restriction

For a CHI request the restriction are as follows :

Text

Description automatically generated with medium confidence

For an AXI request the restrictions are as follows :

Timeline

Description automatically generated with low confidence

# Opens

* Needs more work on Verification paragraph.
* 128 byte exclusives