ARM system-architectures

5. ARM – The AMBA bus

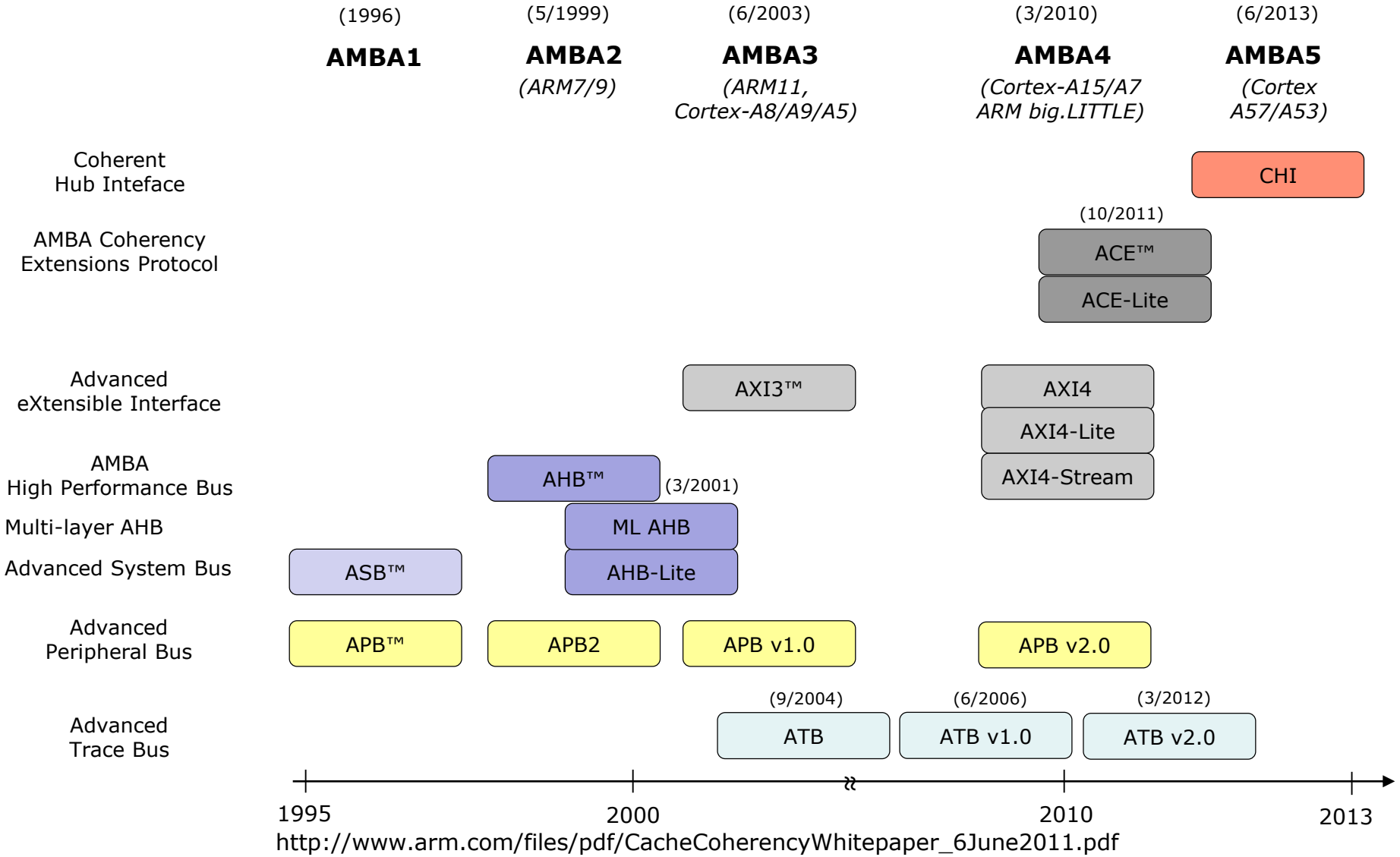
5.1 Introduction to the AMBA bus

5.1 Introduction to the AMBA bus []

- The AMBA bus (Advanced Microcontroller Bus Architecture) is used recently as the de facto standard for interconnecting functional blocks in 32-bit SOC (System-On-a-Chip) designs.
- Originally intended for microcontrollers, now it is widely used for ASICs (Application Specific integrated Circuits) and SOCs, including smartphones and tablets.
- AMBA is a royalty-free open-standard.
- AMBA, originally published about 1996, went through a number or major enhancements, designated as AMBA revisions 1 to 5 (up to date), as shown in the next Figure.

http://en.wikipedia.org/wiki/Advanced_Microcontroller_Bus_Architecture

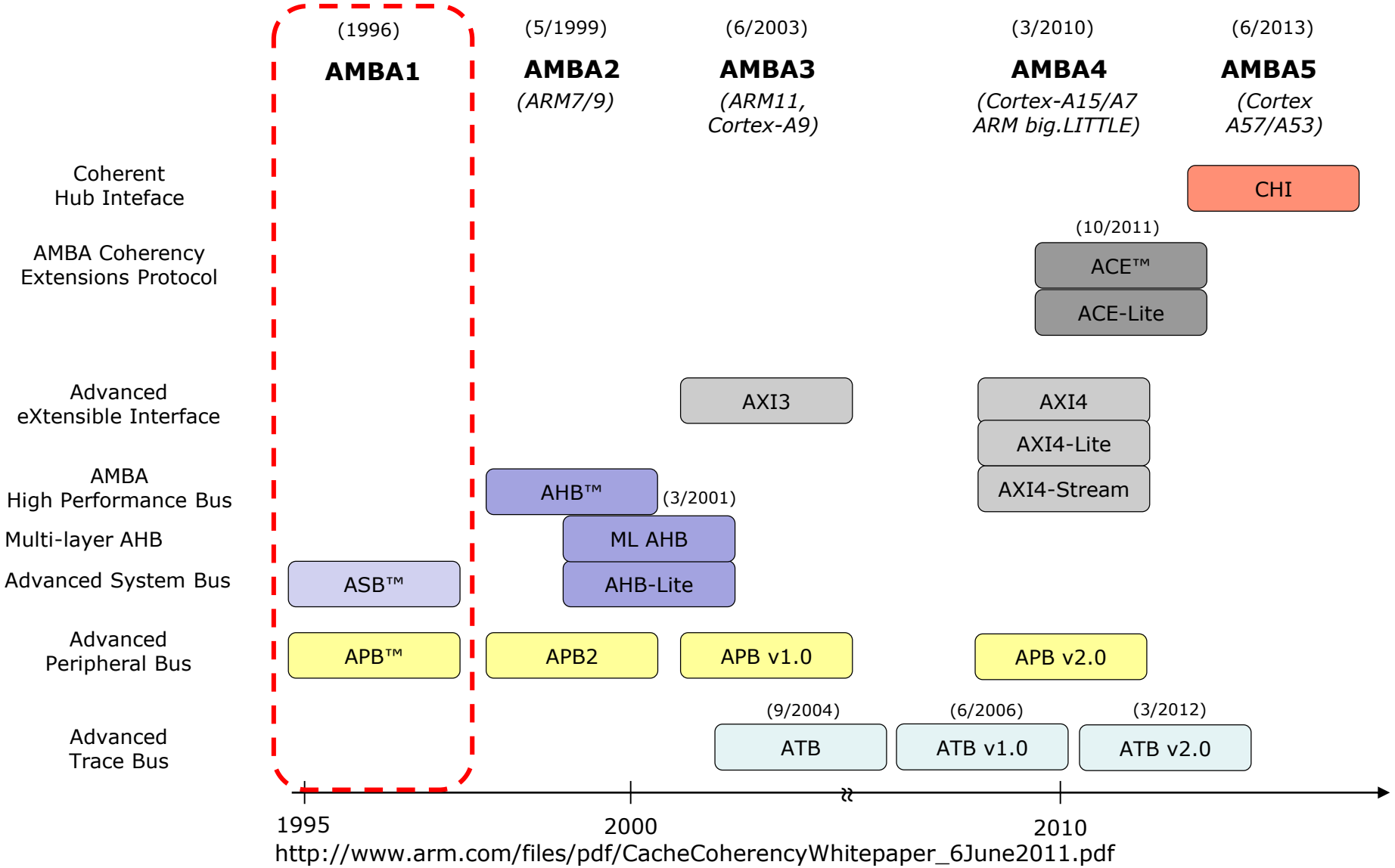
Overview of the AMBA protocol family (based on [])



http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

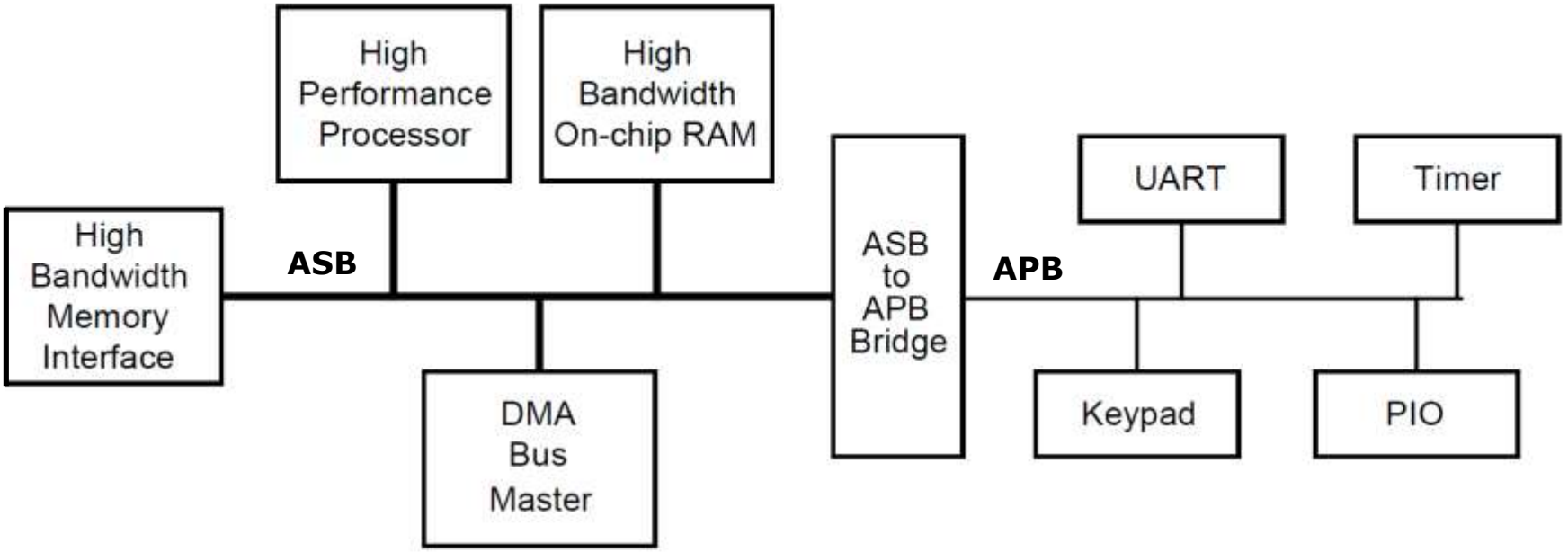
5.2 The AMBA 1 protocol family

5.2 The AMBA 1 protocol family
5.2.1 Overview (based on [])



http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

A typical AMBA1 system []



Advanced System Bus (ASB)
- High Performance
- Pipelined Operation

Advanced Peripheral Bus (APB)
- Low Power
- Latched Address and Control
- Simple Interface
- Suitable for Many Peripherals

AMBA

Advanced Microcontroller Bus Architecture

Specification

Document Number: ARM IHI 0001D

Issued: April 1997

5.2.2 The ASB bus (Advanced System Bus) []
Main features of the ASB bus

a) The ASB bus is a high performance parallel bus, more precisely a high performance bus IP utilizable for SOC designers.

b) Bus operation supports

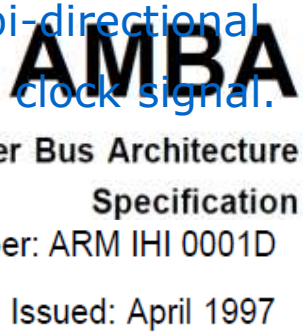
- multiple masters,
- burst transfers and
- pipelining in two stages (as bus granting and bus transfers may be performed in parallel).
- Nevertheless, there is a limitation of the ASB bus as only a single master might be active at a time.

c) The interface signals include

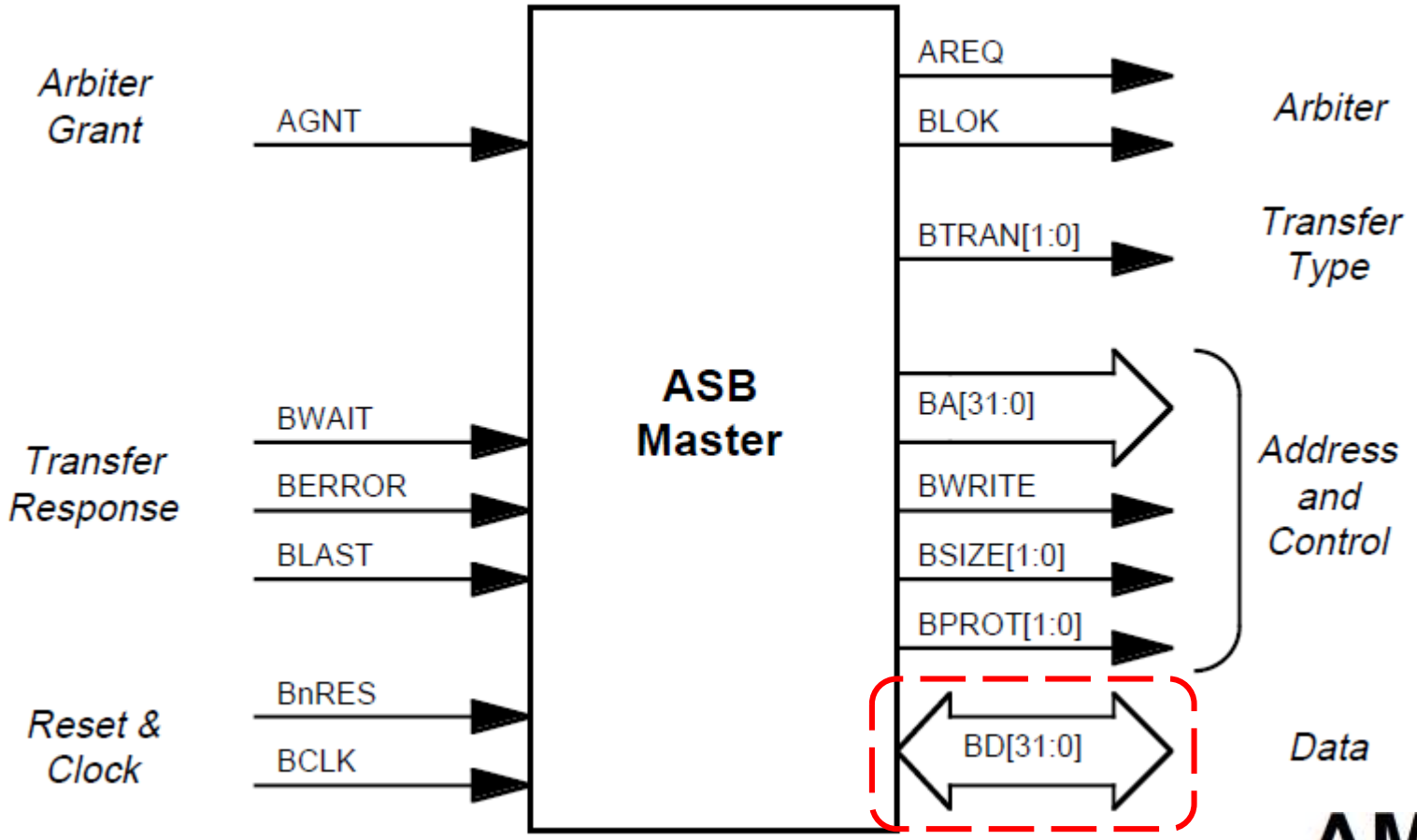
- mostly uni-directional lines, like the address, control or transfer response lines,
- but the data lines carrying write and read data between the masters and the slaves are bi-directional.

In addition the data line of the APB bus is also bi-directional.

d) The ASB protocol makes use of both edges of the clock signal.



Interface signals of ASB masters []



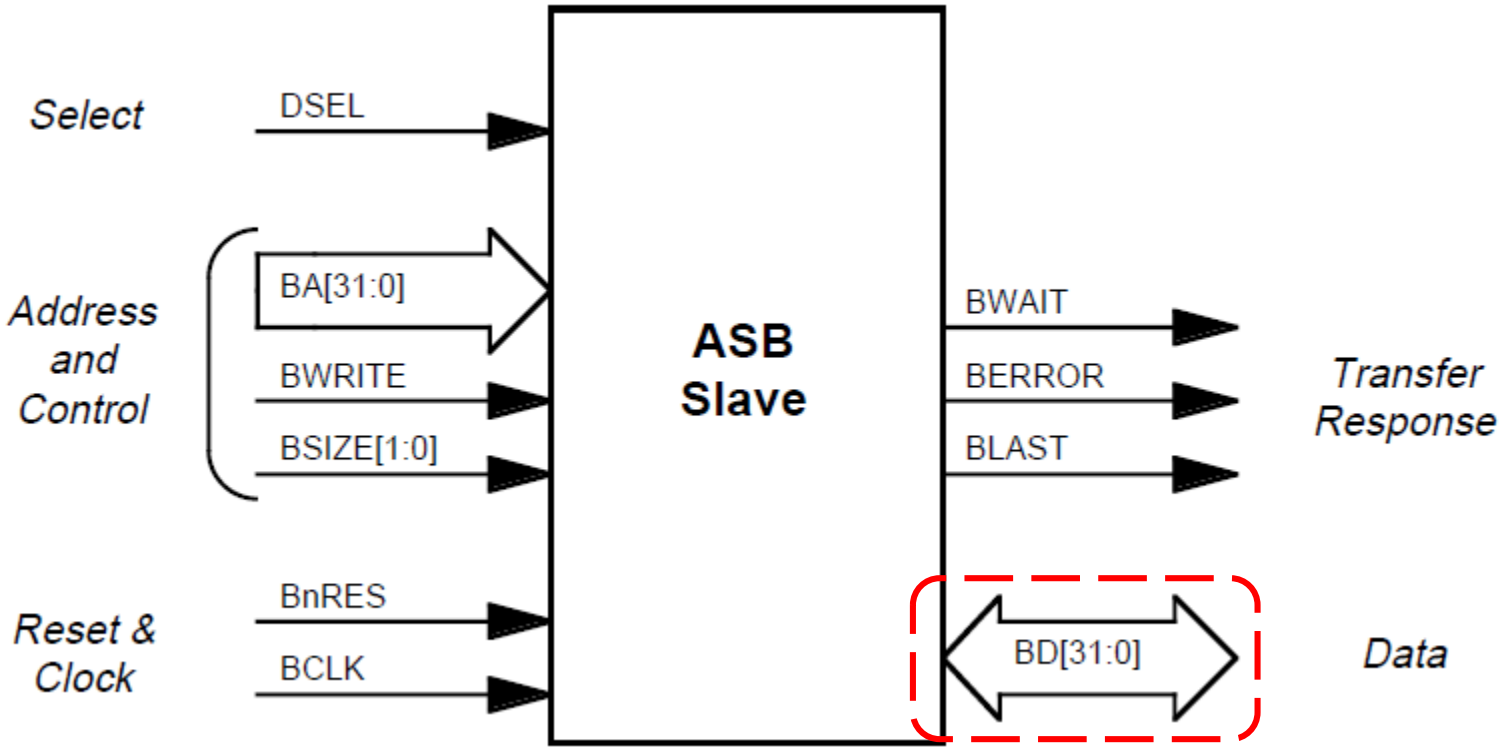
AMBA

Advanced Microcontroller Bus Architecture Specification

Document Number: ARM IHI 0001D

Issued: April 1997

Interface signals of ASB slaves []



Principle of operation of the ASB bus-1 []

- The arbiter determines which master is granted access to the bus,
- When granted, a master initiates a transfer via its port by sending the address and control information (BA[31:0] and BTRANS[1:0] over the related lines to the slaves,
- in case of a write transfer the master sends also the write data (BWDATA[31:0] to the slaves.

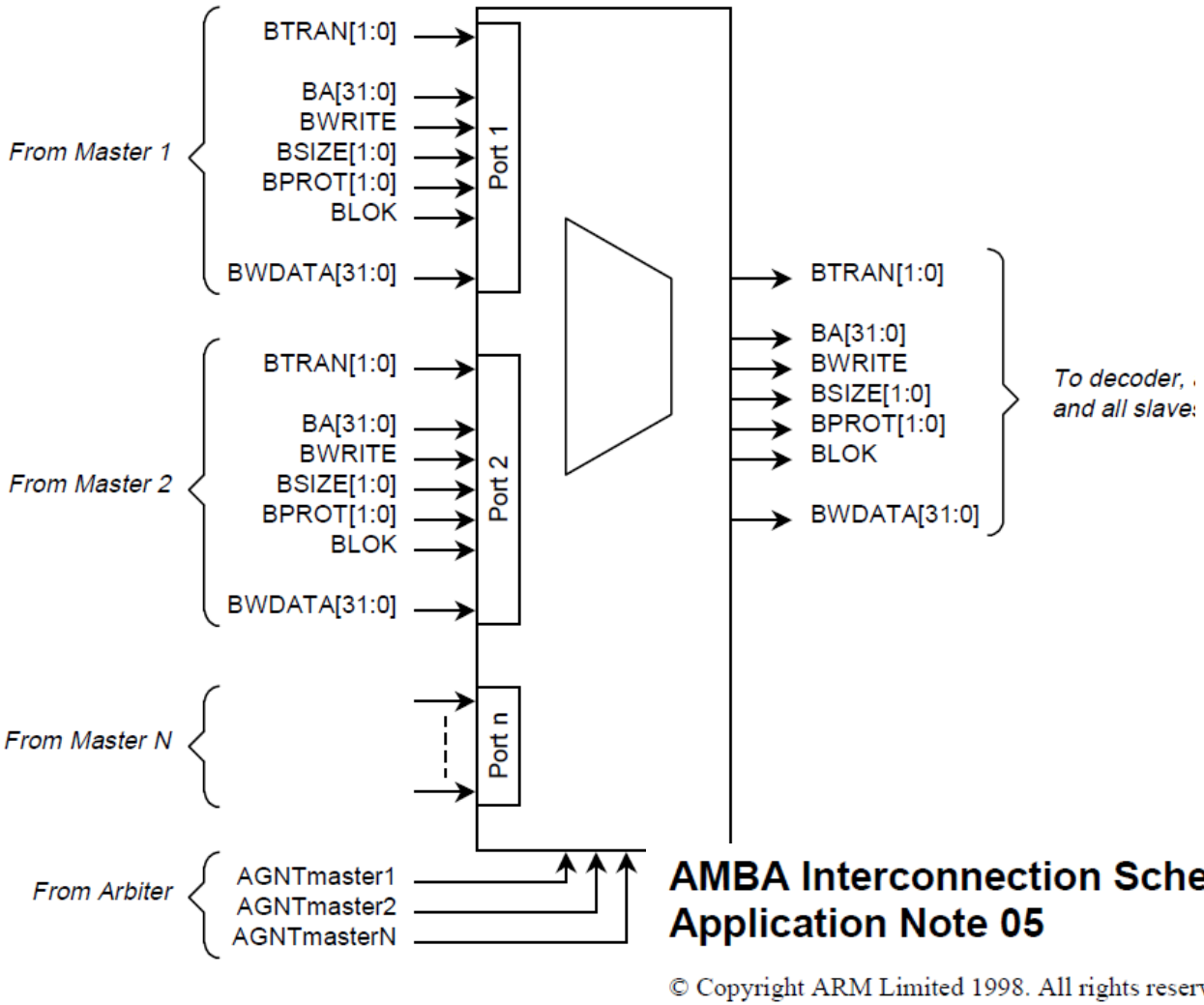


Figure: Principle of the multiplexer implementation at the ASB masters []

Principle of operation of the ASB bus-2 []

- The decoder uses the high order address lines (BADDR) to select a bus slave,
- The slave provides a transfer response back to the bus master and in case of reads the selected slave transfers the read data (BRDATA[31:0] back to the master.

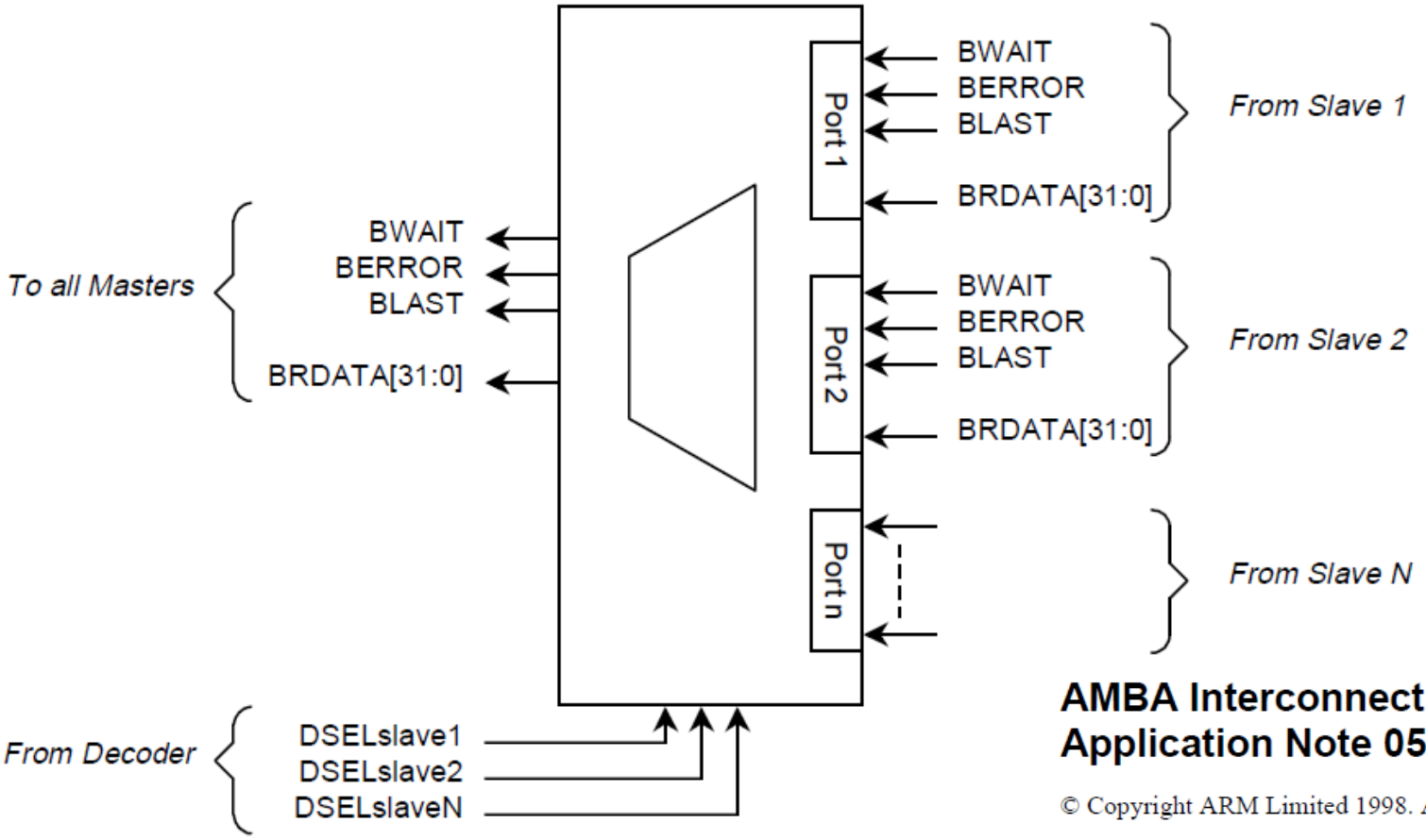
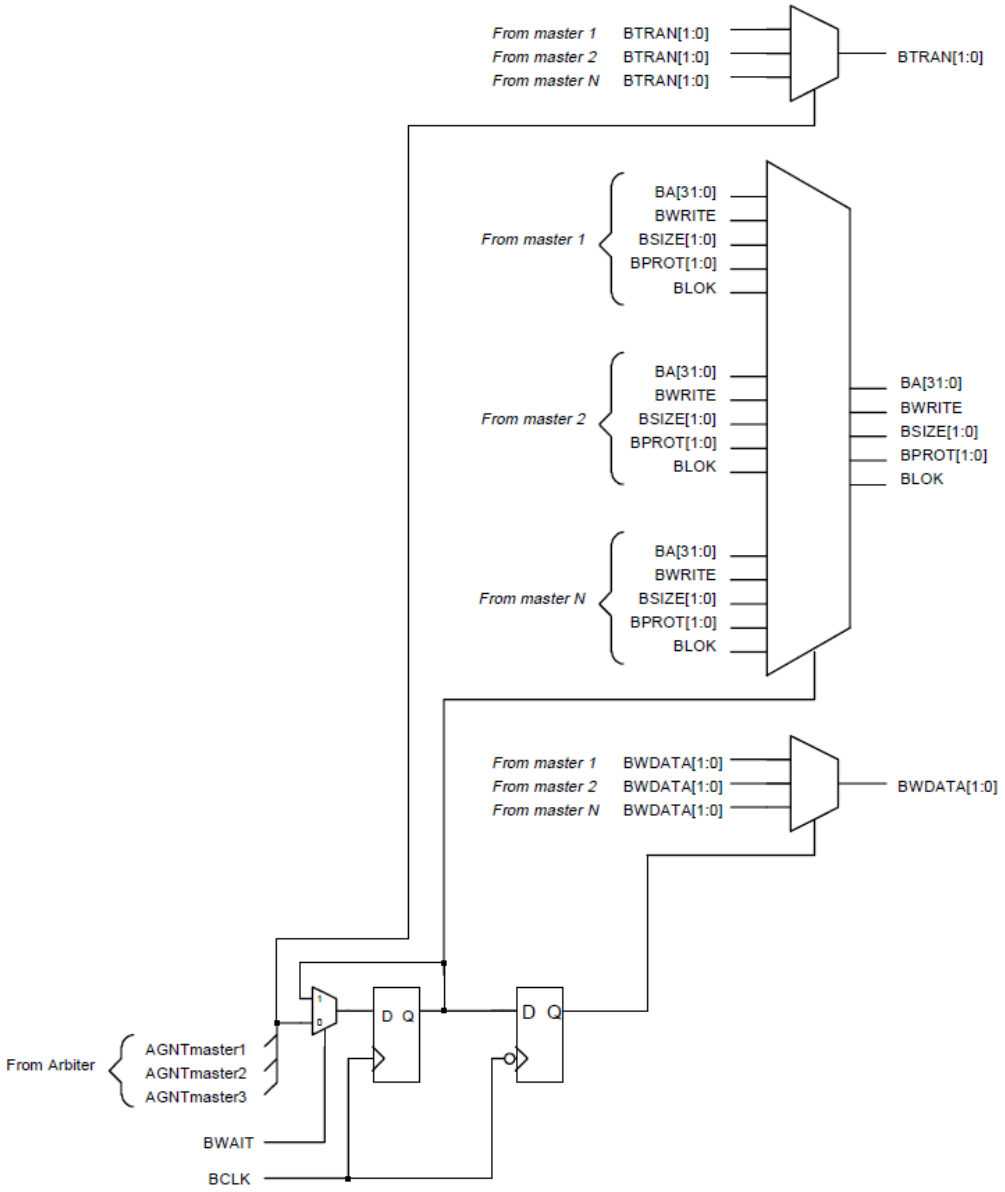


Figure: Principle of the multiplexer implementation at the ASB slaves []

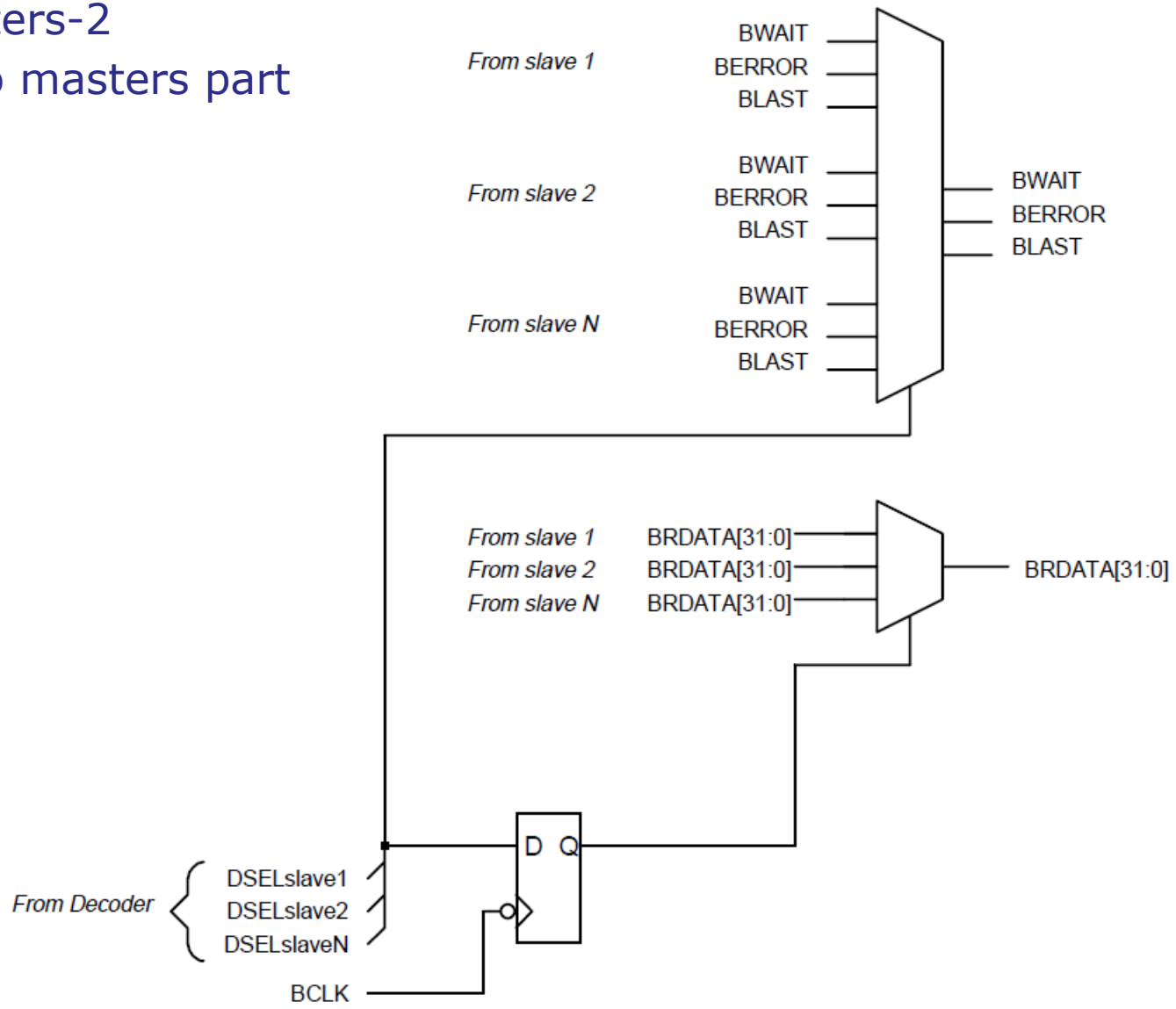
Block diagram of the AHB interface for three masters-1

The masters to slaves part



Block diagram of the AHB interface for three masters-2

The slaves to masters part



Possible transfer types on the ASB []

There are three possible transfer types on the ASB, as follows:

- Non sequential transfers

  They are used for single element data transfers or for the first transfer of a burst.
- Sequential transfers

  They are used for transfers within a burst.

  The address of a sequential transfer is related to the previous transfer.
- Address only transfers.

  They are used when no data movements is required, like address-only-transfers for idle cycles or for bus master handover cycles.

AMBA

Advanced Microcontroller Bus Architecture

Specification

Document Number: ARM IHI 0001D

Issued: April 1997

Example ASB transfer flow: Non sequential read transfer []

A non-sequential transfer occurs

- either as a single transfer
- or the start of a burst transfer.

The next Figure shows a typical non-sequential read transfer including wait states needed until read data becomes available.

AMBA

Advanced Microcontroller Bus Architecture

Specification

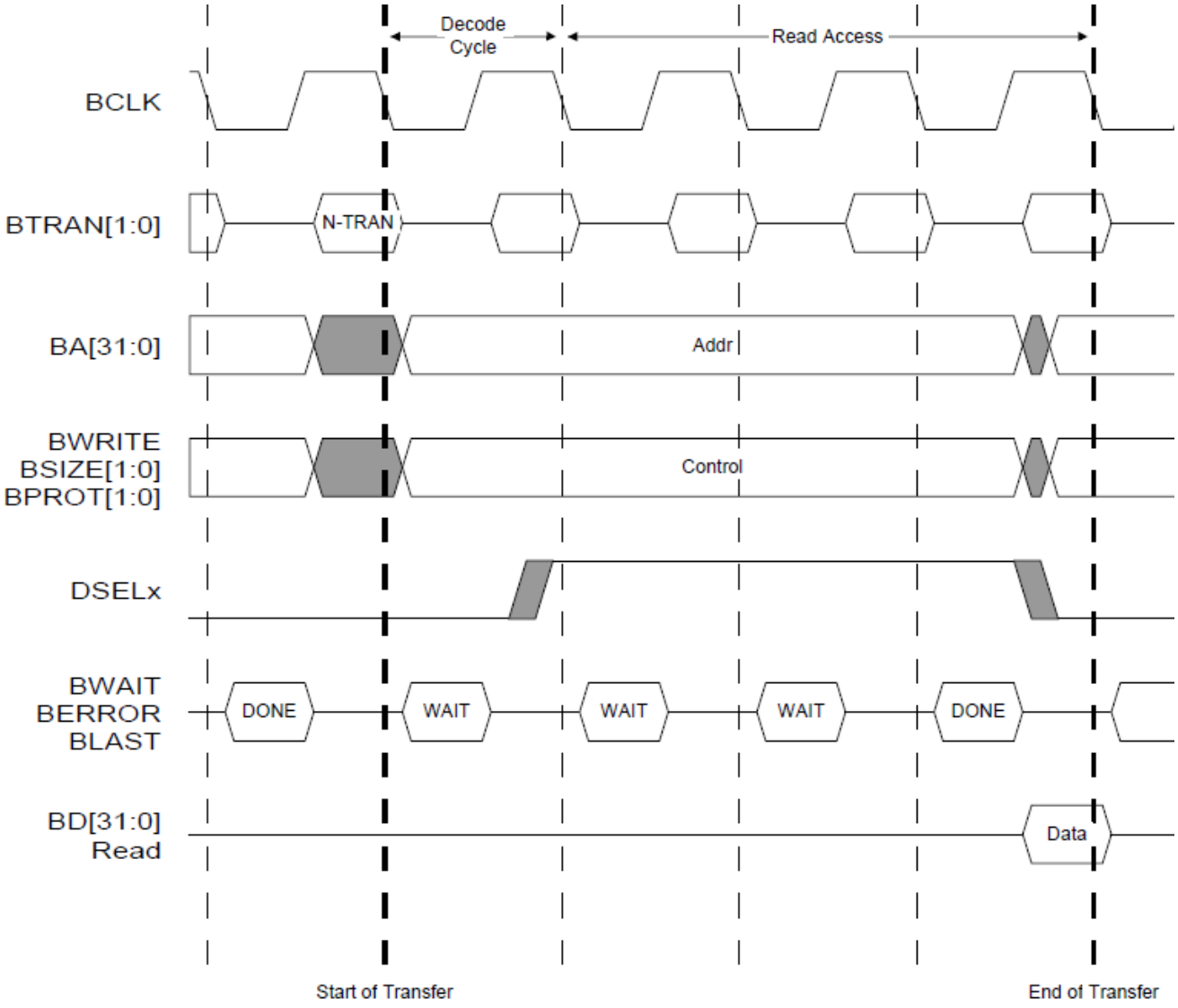
Document Number: ARM IHI 0001D

Issued: April 1997

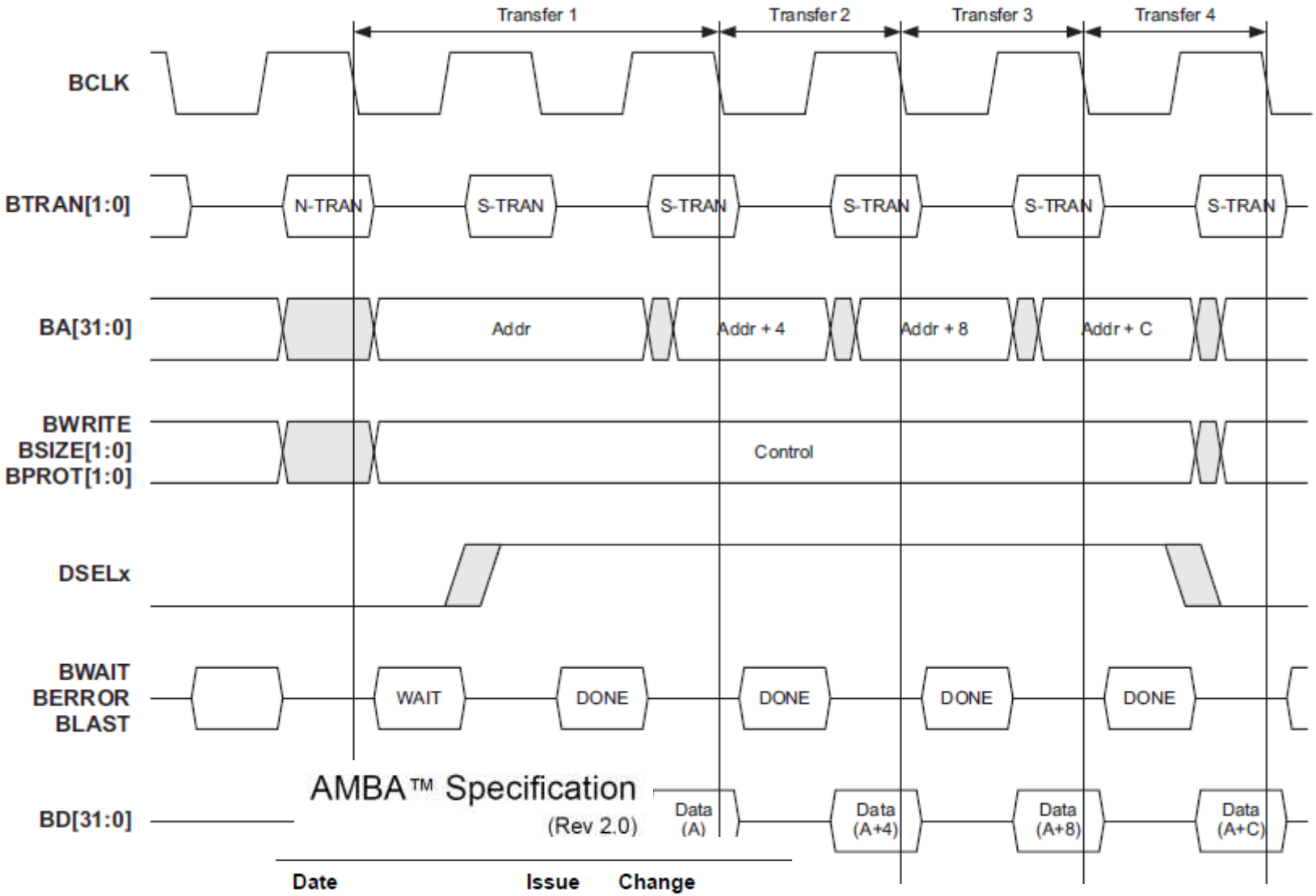
Example 1 ASB transfer flow: Non sequential read transfer []

- The transfer begins at the falling edge of the BCLK signal after the previous transfer has completed, as indicated by BWAIT signaling “DONE”.
- The type of transfer that a bus master performs is determined by the BTRAN[1:0] signals at the start of the transfer.
- The high order address lines (BA[31:0] select a bus slave, and
- the control lines identify the operation and transfer size.
- After the slave can provide the read data, it signals it by BWAIT “DONE” and
- it transfers the read data.

This completes the read access.



Example 2 ASB transfer flow: Sequential read transfer-1 []



AMBA™ Specification
(Rev 2.0)

Date	Issue	Change
13th May 1999	A	First release

Example 2 ASB transfer flow: Sequential read transfer-2 []

- For a sequential transfer (burst transfer) the address is related to that of the previous transfer.
- The control information, as indicated by the BWRITE, BPROT and BSIZE signals will be the same as the previous transfer.
- In the case of a sequential transfer the address can be calculated based on the previous address (A) and transfer size.

  E.g. for a burst of word accesses the subsequent addresses would be A, A+4, A+8 etc.
- The sequential transfer completes when the BTRAN[1:0] signal does not more indicate a continuation.

AMBA

Advanced Microcontroller Bus Architecture

Specification

Document Number: ARM IHI 0001D

Issued: April 1997

Data bus width (designated as transfer size []

The ASB protocol allows the following data bus width options:

- 8-bits (byte)
- 16-bit (halfword) and
- 32-bit (word)

They are encoded in the BSIZE[1:0] signals that are driven by the active bus master and have the same timing as the address bus [a].

By contrast, the AHB protocol allows in addition significantly wider data buses, as discussed later.

AMBA

Advanced Microcontroller Bus Architecture

Specification

Document Number: ARM IHI 0001D

Issued: April 1997

Multi-master operation []

- A simple two wire request/grant mechanism is implemented between the arbiter and each bus master.
- The arbiter ensures that only a single bus master may be active on the bus and also ensures that when no masters are requesting the bus a default master is granted.
- The specification also supports shared lock signal.

  This signal allows bus masters to indicate that the current transfer is indivisible from the subsequent transfer and will prevent other bus masters from gaining access to the bus until the locked transfer has completed.
- The arbitration protocol is defined but the prioritization is left over to the
- application.

AMBA

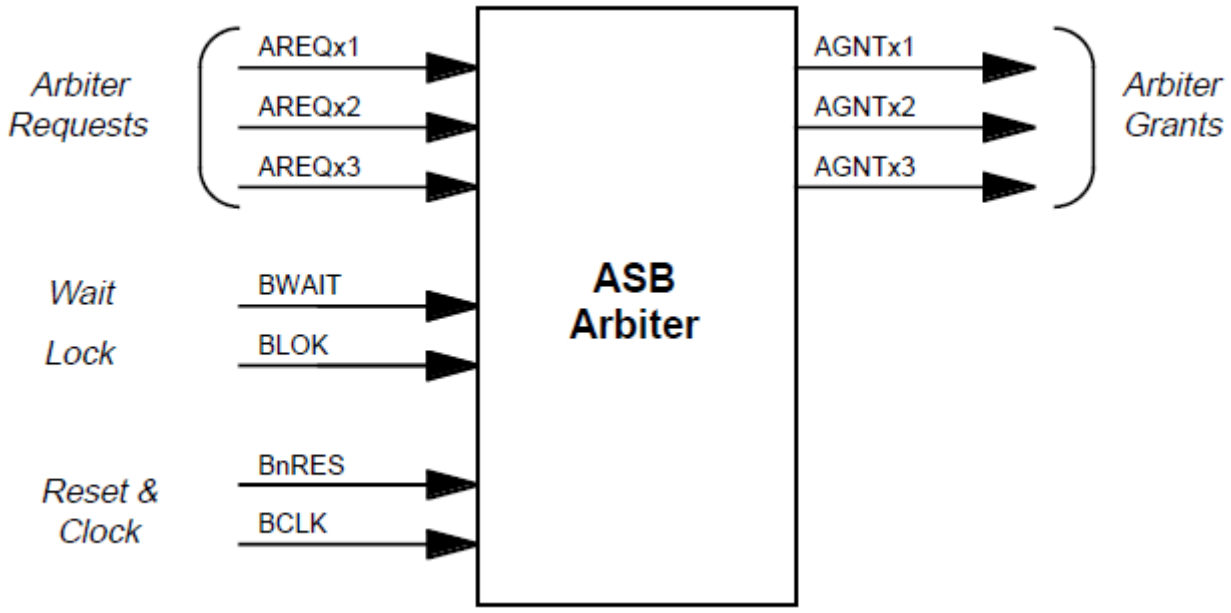
Advanced Microcontroller Bus Architecture

Specification

Document Number: ARM IHI 0001D

Issued: April 1997

Black box layout of the ASB arbiter assuming three bus masters []



AMBA

Advanced Microcontroller Bus Architecture

Specification

Document Number: ARM IHI 0001D

Issued: April 1997

Description of the operation of the arbiter (simplified)

- The ASB bus protocol supports a straightforward form of pipelined operation, such that arbitration for the next transfer is performed during the current transfer.
- The ASB bus can be re-arbitrated on every clock cycle.
- The arbiter samples all the request signals (AREQx) on the falling edge of BCLK and during the low phase of BCLK the arbiter asserts the appropriate grant signal (AGNTx) using the internal priority scheme and the value of the lock signal (BLOK).

AMBA

Advanced Microcontroller Bus Architecture

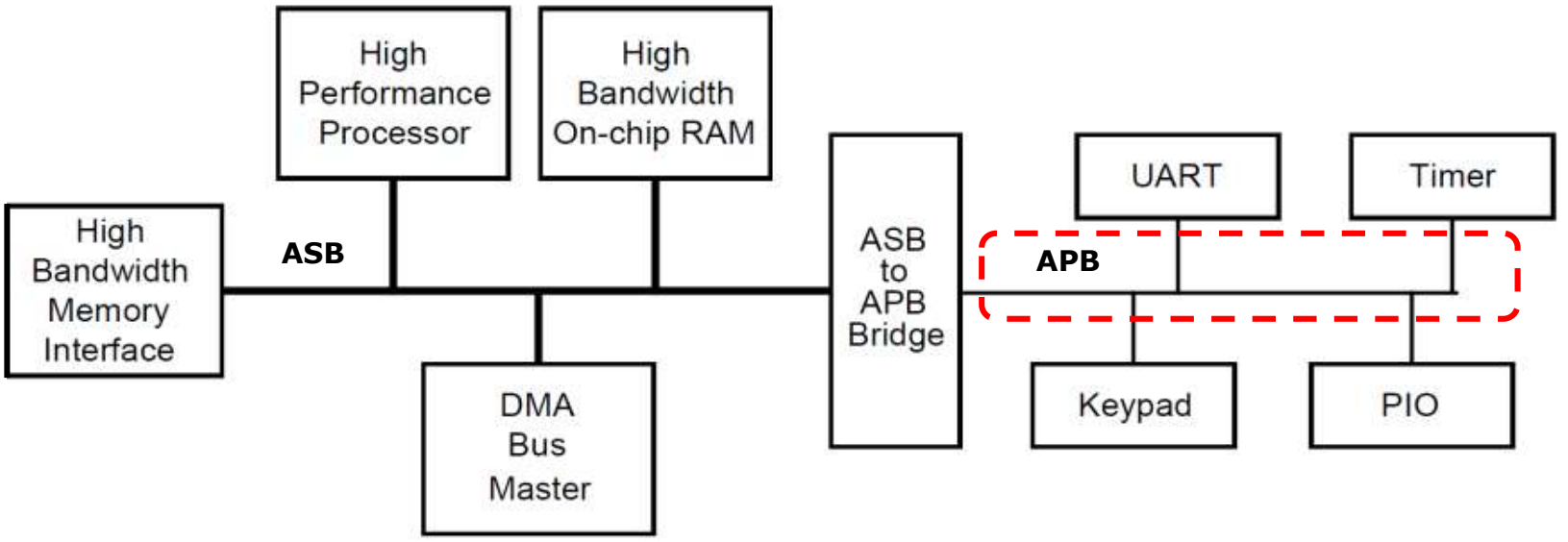
Specification

Document Number: ARM IHI 0001D

Issued: April 1997

The APB bus (Advanced Peripheral Bus-1 [])

It appears as a local secondary bus encapsulated as a single slave device, as indicated below.



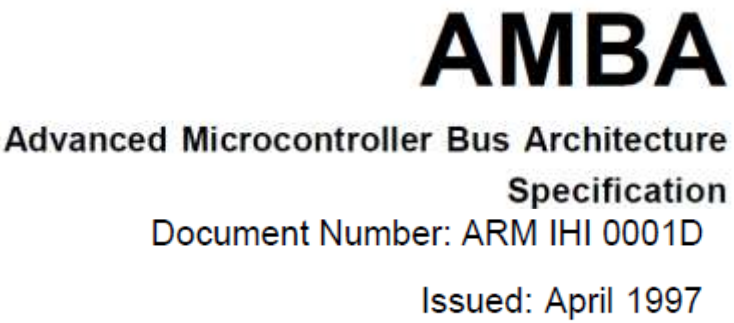
Advanced System Bus (ASB)
- High Performance
- Pipelined Operation
- Burst Transfers
- Mulitple Bus Masters

Advanced Peripheral Bus (APB)
- Low Power
- Latched Address and Control
- Simple Interface
- Suitable for Many Peripherals

Figure: A typical AMBA system []

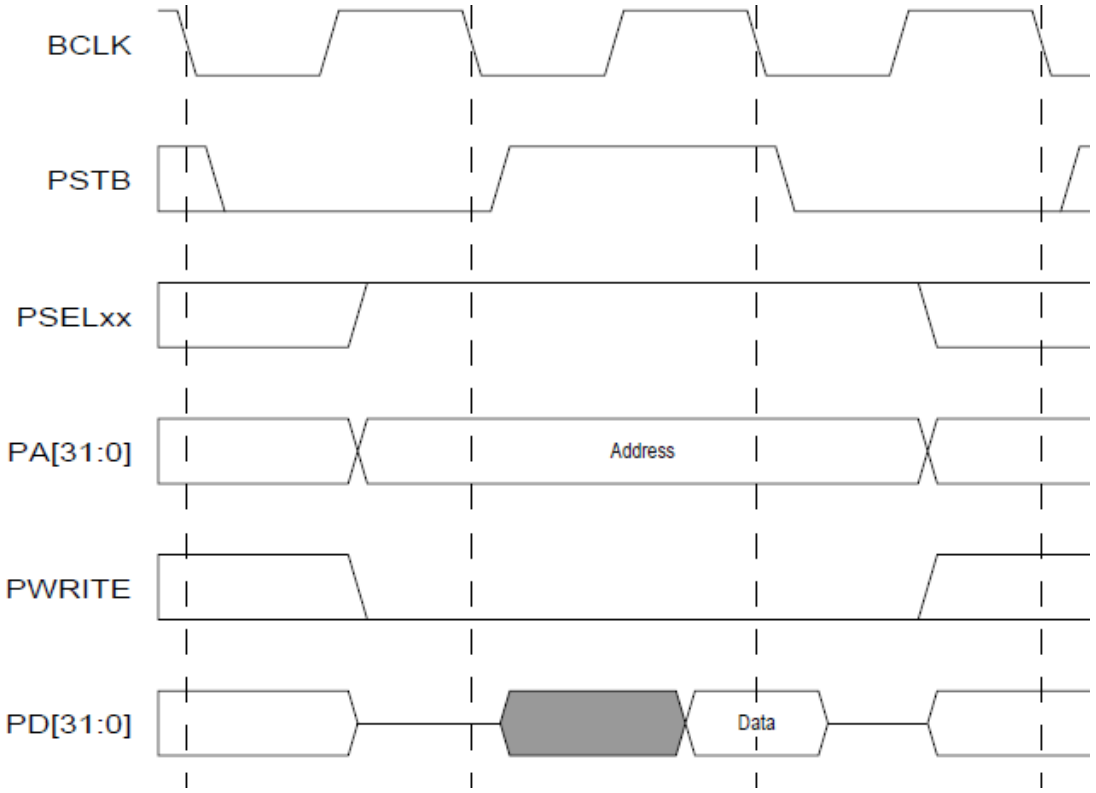
5.2.3 The APB bus (Advanced Peripheral Bus-2 []

- The APB bus is a simple, low-power extension to the system bus which builds on ASB signals directly.
- It does not support pipelining.



The APB read cycle

- The address and control signals are set up before the strobe (PSTB) and held valid after the strobe.
- Data need not be driven throughout the access but read data must be set up and valid prior the falling edge of the strobe.
- The falling edge of the strobe (PSTB) is derived from the falling edge of the system clock (BCLK).



AMBA

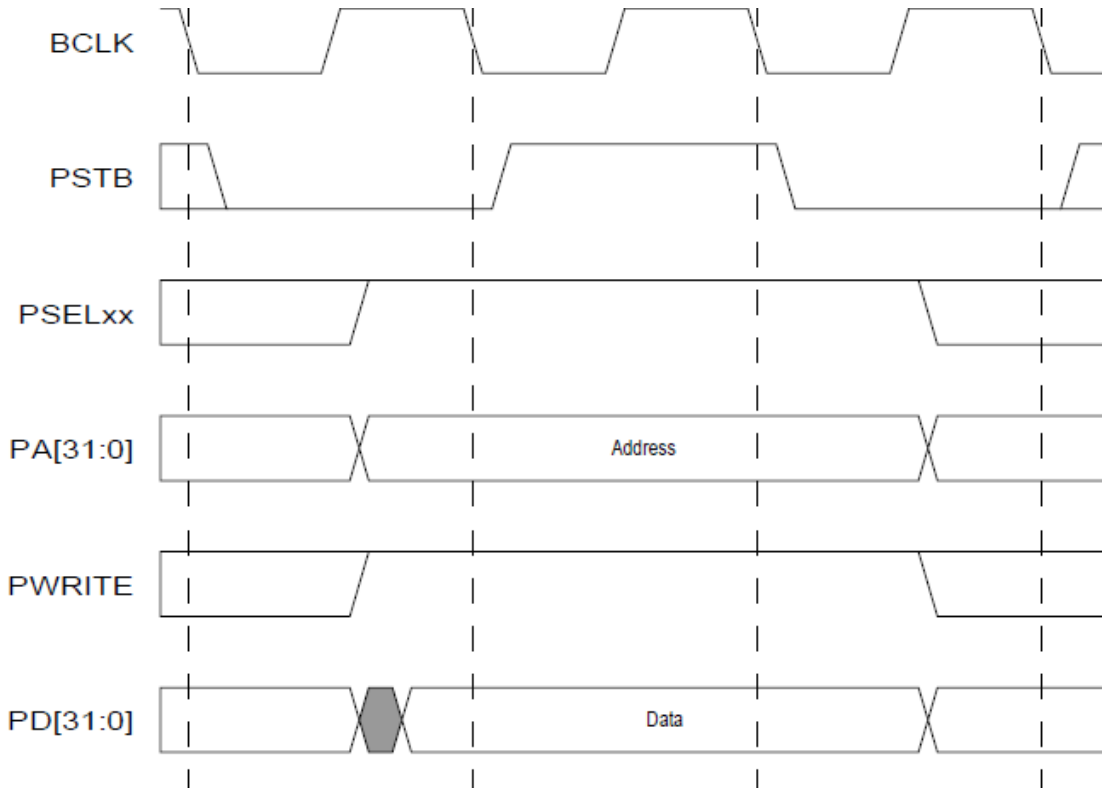
Advanced Microcontroller Bus Architecture Specification

Document Number: ARM IHI 0001D

Issued: April 1997

The APB write cycle

- The address and control signals are set up before the strobe (PSTB) and held valid after the strobe.
- The falling edge of the strobe (PSTB) is derived from the falling edge of the system clock (BCLK).



AMBA

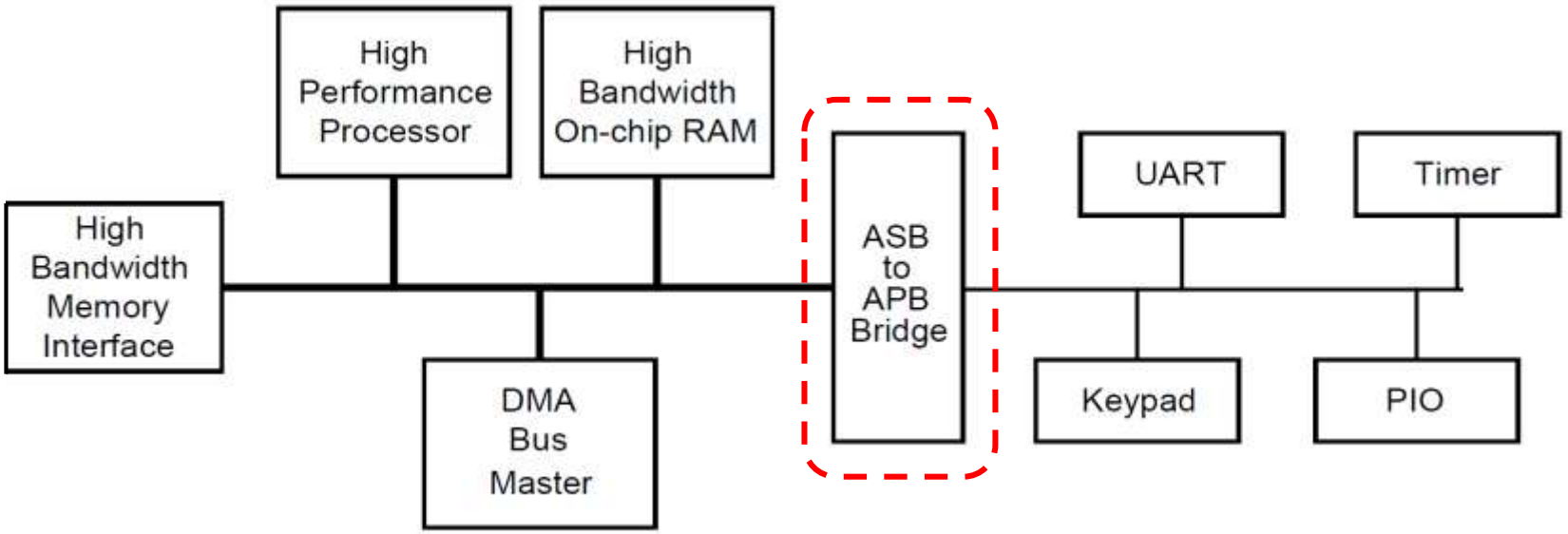
Advanced Microcontroller Bus Architecture Specification

Document Number: ARM IHI 0001D

Issued: April 1997

The APB Bridge []

The APB Bridge appears as a slave module which handles the bus handshake and control signal retiming.



AMBA

Advanced Microcontroller Bus Architecture

Specification

Document Number: ARM IHI 0001D

Issued: April 1997

Primary drawbacks of the ASB protocol []

There are two primary drawbacks of the ASB bus:

- The ASB protocol uses both edges of the clock signal that imposes increased complexity for most ASIC design and synthesis tools that are based on using only the rising edge of the clock.
- The ASB protocol includes the bi-directional data buses BD[30:0] and PA[31:0].

  We point out that bi-directional buses and their typical representation by tri-state signals is not possible under many design rules.

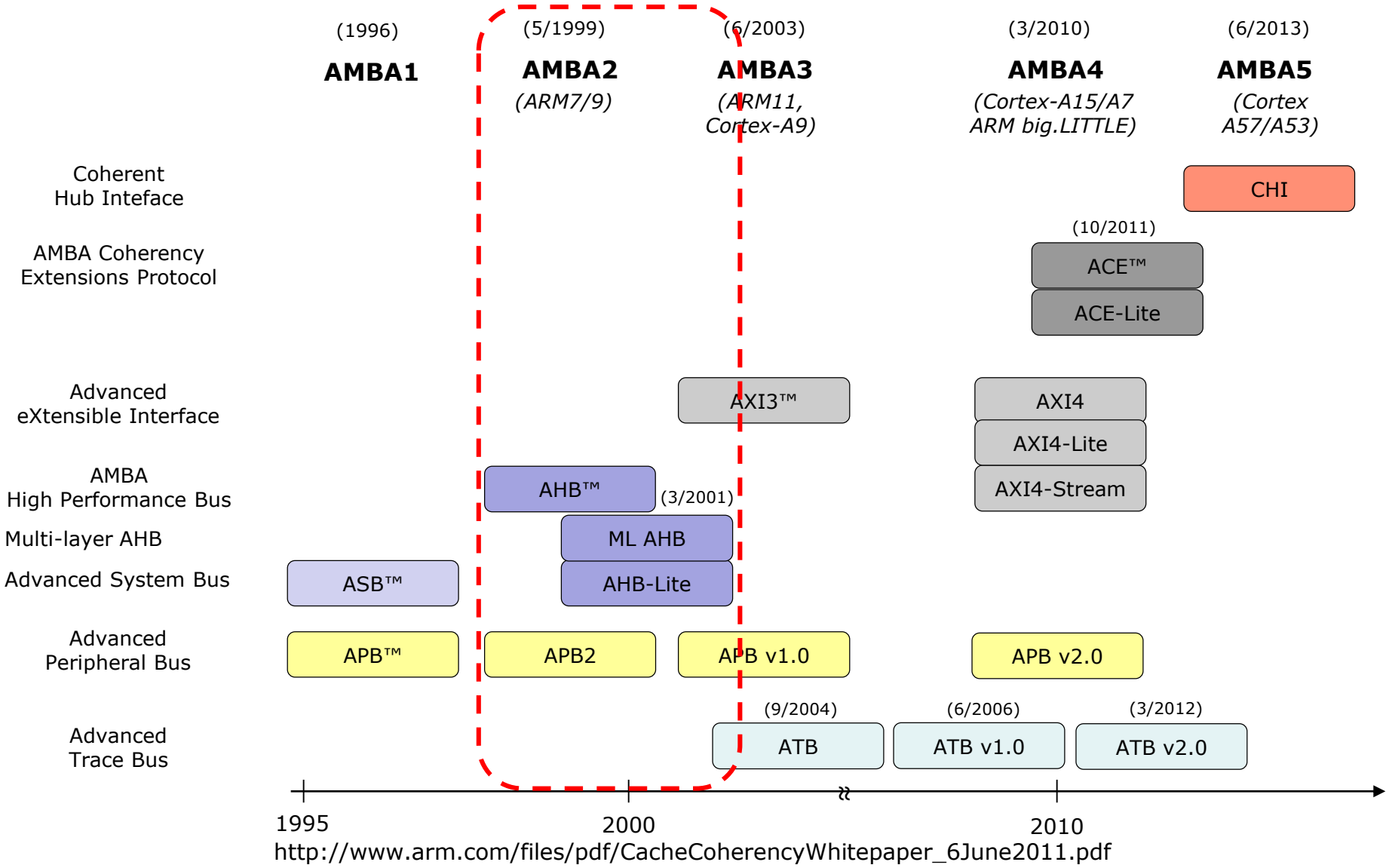
  Further on, the bus-turnaround times of tri-state buses cause usually performance penalties.

The AHB protocol amends these deficiencies.

For highest performance, typical designs based on ASB use an ARM processor with a write-back cache. A write-back cache is a cache algorithm that allows data to be written into cache without updating the system memory. Since ASB does not have any provisions for maintaining cache coherency of multiple caching bus masters only one processor can be used on ASB.

5.3 The AMBA 2 protocol family

5.3 The AMBA 2 protocol family
5.3.1 Overview (based on [])



http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Components of the AMBA 2 protocol family (1999) []

The AMBA 2 protocol family (AMBA Revision 2.0) included first

- the AHB (AMBA High Performance Bus) and
- the APB2 (Advanced Peripheral Bus Revision 2)

specification,

but was extended in 2001 by

- the Multi-layer AHB and
- the AHM-Lite

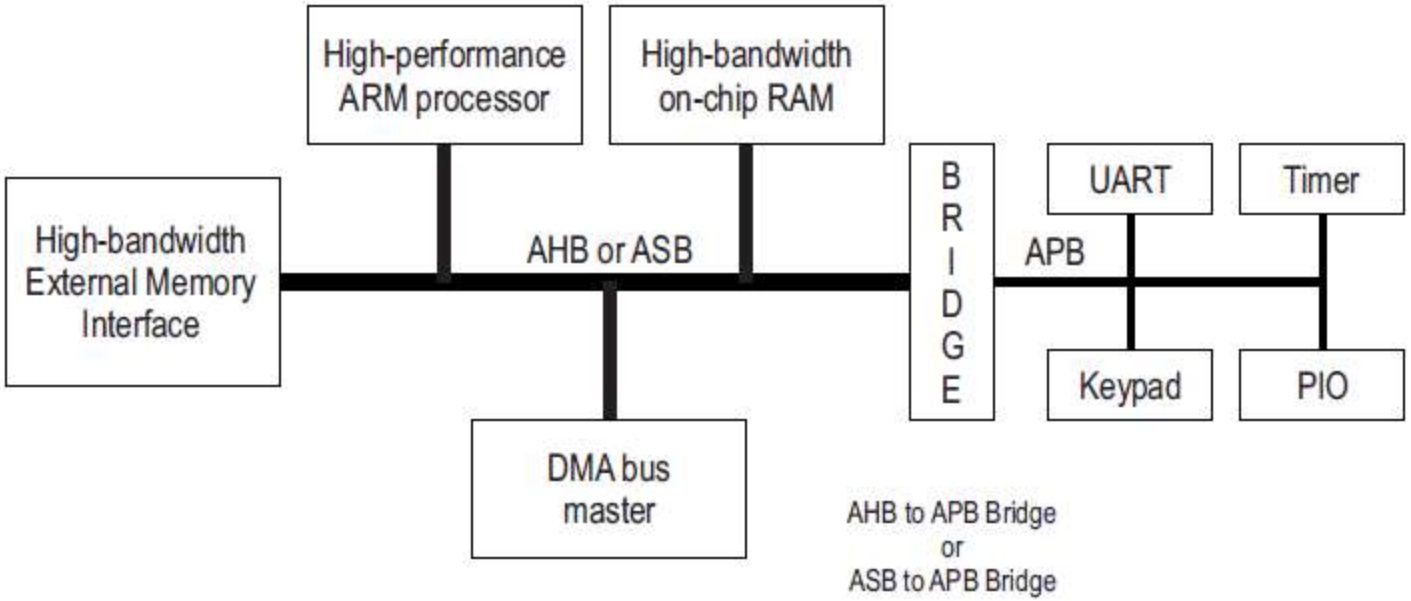
specifications, as indicated in the previous Figure.

AMBA™ Specification

(Rev 2.0)

Date	Issue	Change
13th May 1999	A	First release

A typical AMBA system with the AHB or ASB bus []



AMBA AHB

* High performance
* Pipelined operation
* Multiple bus masters
* Burst transfers
* Split transactions

AMBA ASB

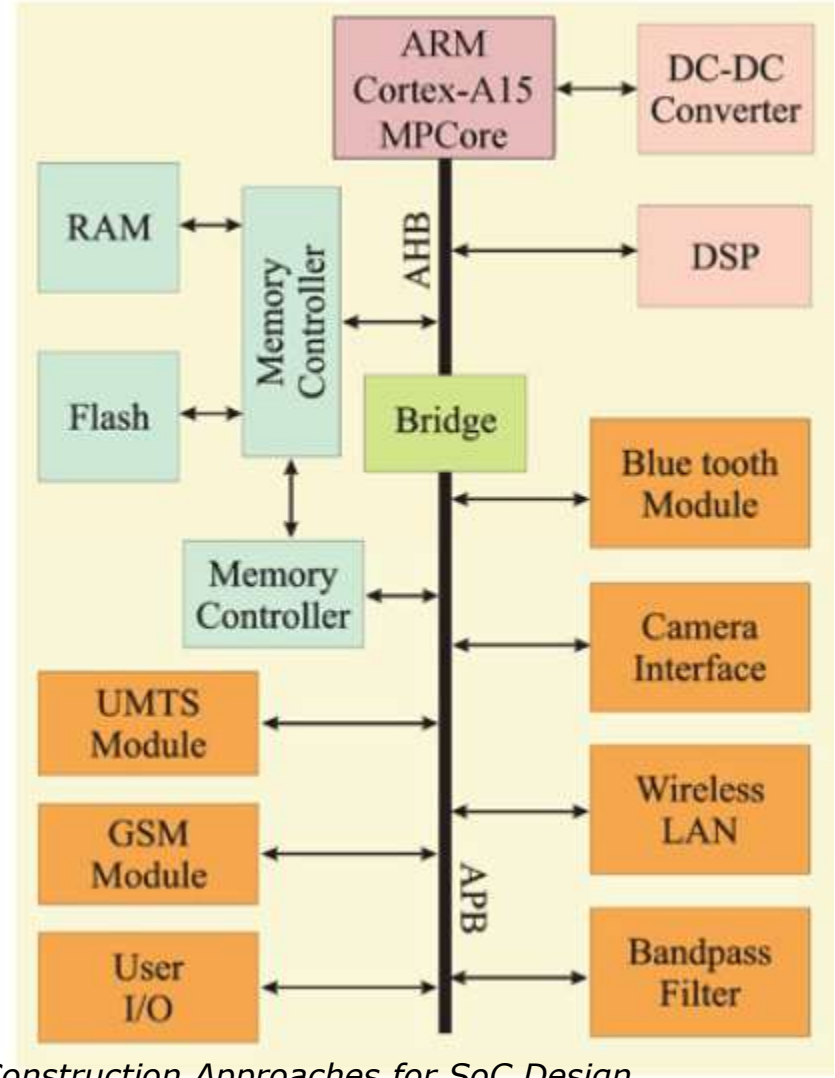
* High performance
* Pipelined operation
* Multiple bus masters

AMBA APB

* Low power
* Latched address and control
* Simple interface
* Suitable for many peripherals

Figure 1-1 A typical AMBA system

A typical AMBA system based on the AHB bus []



R. Sinha et al., *Correct-by-Construction Approaches for SoC Design*,
DOI 10.1007/978-1-4614-7864-5 2,

5.3.2 The AHB bus

Main enhancements of the AHB bus vs. the ASB bus []

a) Spit transactions.

This allows enhanced pipelining on the bus by overlapping the address and data phases of transactions from different bus masters.

b) Enhanced burst transactions.

c) Wider data bus options (termed as transfer size alternatives).

d) Using only uni-directional signals (also for data buses, in contrast to the ASB protocol).

e) Using only the rising edge of the bus clock (in contrast to the ASB protocol where both edges are used).

a) Split transactions-1

Transactions are split into two phases, into the address and the data phases, as shown below.

Splitting the transfer into two phases allows overlapping the address phase of any transfer with the data phase of the previous transfer, as discussed later.

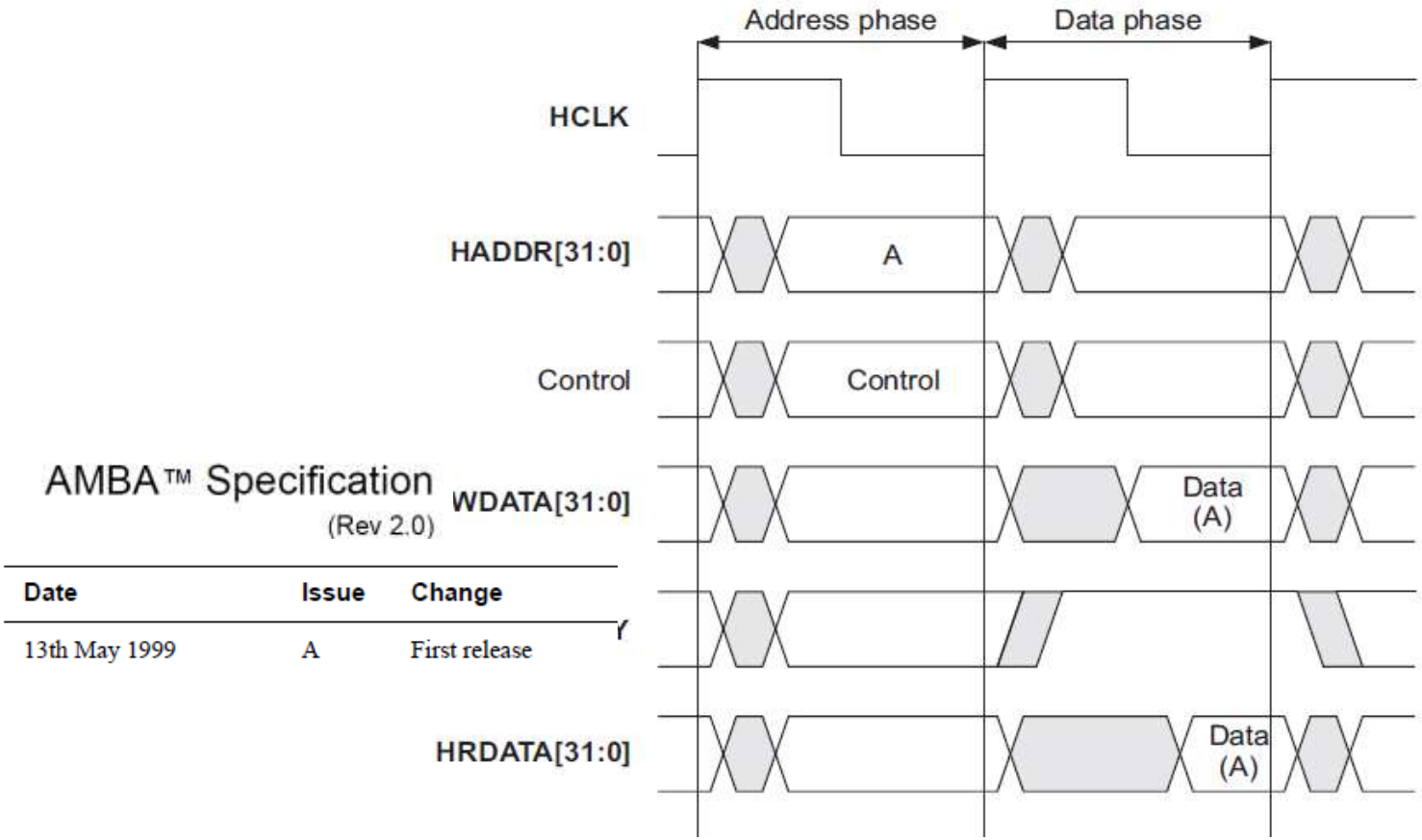


Figure: Example of a split read or write transaction without wait states []

Split transactions-2

Nevertheless, the slave may insert wait states into any transfer if additional time is needed for the completion of the requested operation, as shown in the next Figure.

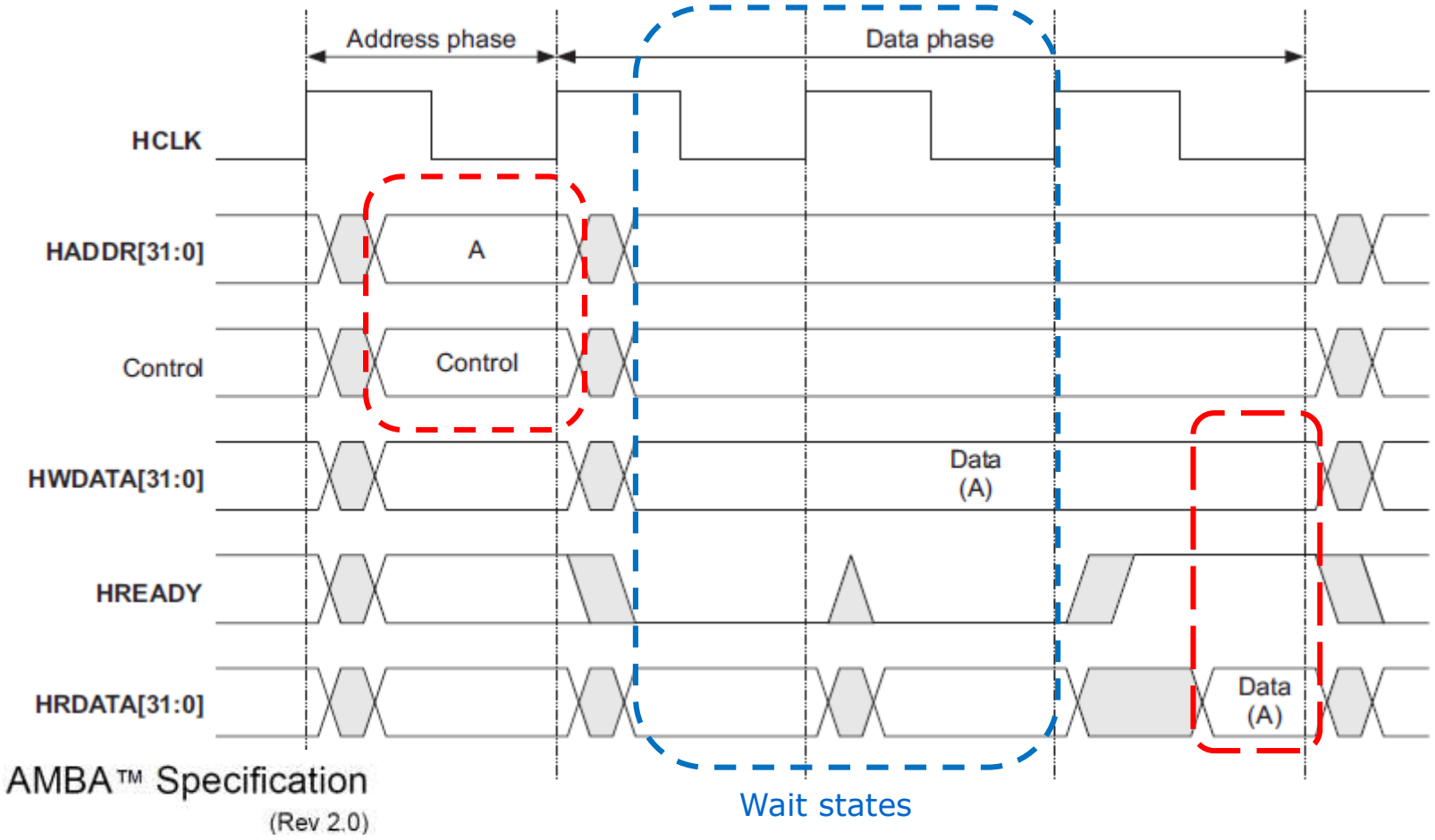
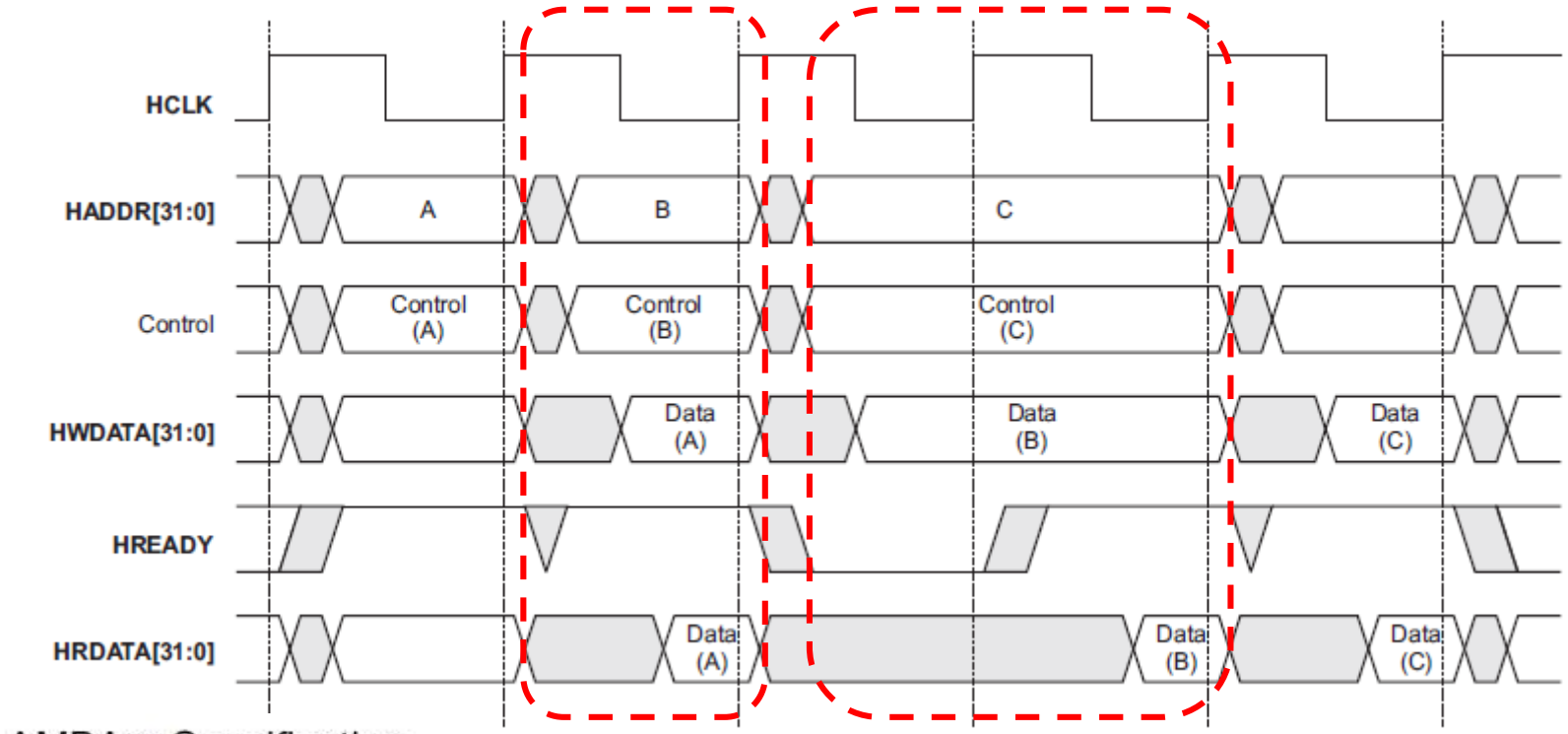


Figure: Example of a split read or write transaction with two wait states []

Date	Issue	Change
13th May 1999	A	First release

Split transactions-3

Overlapping the address and data phases of different transfers (as shown below) increases the pipeline depth of the bus operation from two to three and thus contributes for higher performance.



AMBA™ Specification (Rev 2.0)

Figure: Example of multiple (read or write) transactions with pipelining []

Date	Issue	Change
13th May 1999	A	First release

b) Enhanced burst transactions-1 []

- The AMBA AHB protocol redefines burst transfers.

  In the ASB protocol burst could be specified as a sequence of non-sequential and sequential transfers, with a specific signal (BTRAN[1:0]) identifying the end of a burst.

- The AHB protocol allows to specify bursts explicitly, as four, eight and sixteen-beat bursts or undefined-length bursts.
- Both incrementing and wrapping bursts are supported.

  In incrementing bursts the burst accesses sequential locations while incrementing the address of each transition by the transfer size (e.g. by 4 for word transfers).

  E.g. a four-beat incrementing burst of words (4-bytes) that starts e.g. at the location 0x38 will access data at the addresses 0x38, 0x3C, 0x40 and 0x44.

  In wrapping bursts, if the start address of the transfer is not aligned to the total number of bytes in the burst (transfer size x beats) then the address of the transfer in the burst will wrap when the boundary is reached.

  E.g. a four-beat wrapping burst of words (4-bytes) will wrap at 16-byte boundaries, so after a start address of 0x34 data will be accessed from the 0x38, 0x3C and 0x30 addresses.

AMBA™ Specification

(Rev 2.0)

Date	Issue	Change
13th May 1999	A	First release

Enhanced burst transactions-2

In the redesigned burst protocol three burst signals (HBURST[2:0] identify the burst type and the length, as the next Table indicates.

HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst
011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	16-beat wrapping burst
111	INCR16	16-beat incrementing burst

Table: Encoding the burst signals (HBURST[2:0] []

AMBA™ Specification

(Rev 2.0)

Date	Issue	Change
13th May 1999	A	First release

c) Wider data bus options (called as transfer size alternatives)-1 []

The ASB protocol allows data bus widths of:

- 8-bit (byte)
- 16-bit (halfword) and
- 32-bit (word)

They are encoded in the BSIZE[1:0] signals that are driven by the active bus master and have the same timing as the address bus [a].

By contrast, the AHB protocol allows in addition significantly wider data buses, as the next Table indicates.

AMBA

Advanced Microcontroller Bus Architecture

Specification

Document Number: ARM IHI 0001D

Issued: April 1997

Wider data bus options (called transfer size alternatives)-2 []

HSIZE[2]	HSIZE[1]	HSIZE[0]	Size	Description
0	0	0	8 bits	Byte
0	0	1	16 bits	Halfword
0	1	0	32 bits	Word
0	1	1	64 bits	-
1	0	0	128 bits	4-word line
1	0	1	256 bits	8-word line
1	1	0	512 bits	-
1	1	1	1024 bits	-

Table: Transfer sizes in the AHB protocol indicated by the HSIZE[2:0] signals []

From the available data bus width options practically only the 32, 64, or 128 bits wide alternatives are used.

AMBA™ Specification

(Rev 2.0)

Date	Issue	Change
13th May 1999	A	First release

d) Using only uni-directional signals-1

The AHB protocol makes use only of uni-directional data buses, as shown below.

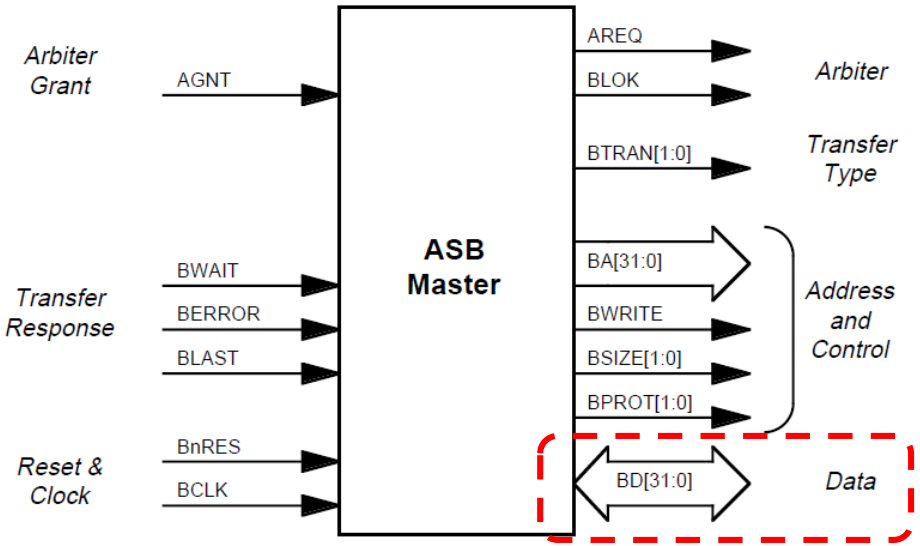
Interface signals of ASB bus masters [a]

AMBA

Advanced Microcontroller Bus Architecture Specification

Document Number: ARM IHI 0001D

Issued: April 1997

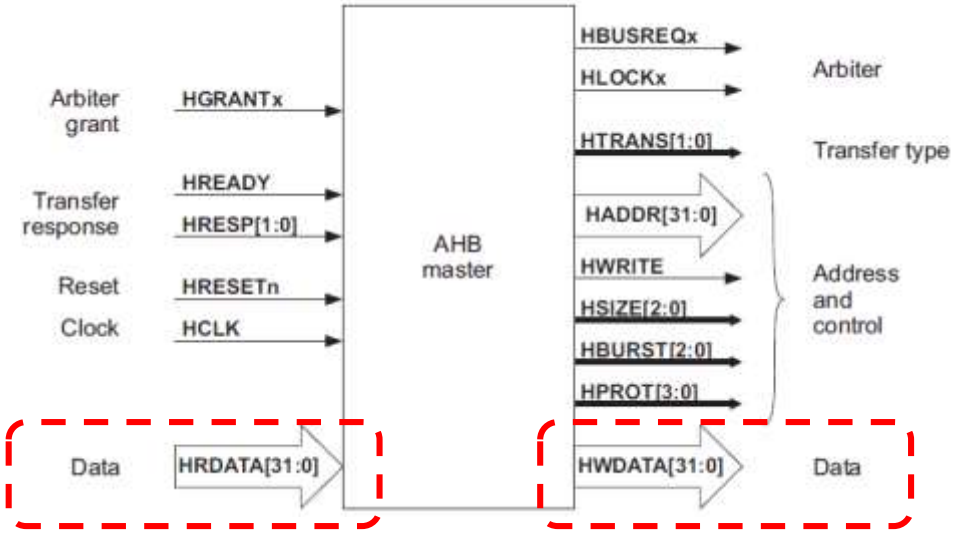


Interface signals of AHB bus masters [b]

AMBA™ Specification

(Rev 2.0)

Date	Issue	Change
13th May 1999	A	First release



d) Using only uni-directional signals-2

Benefit

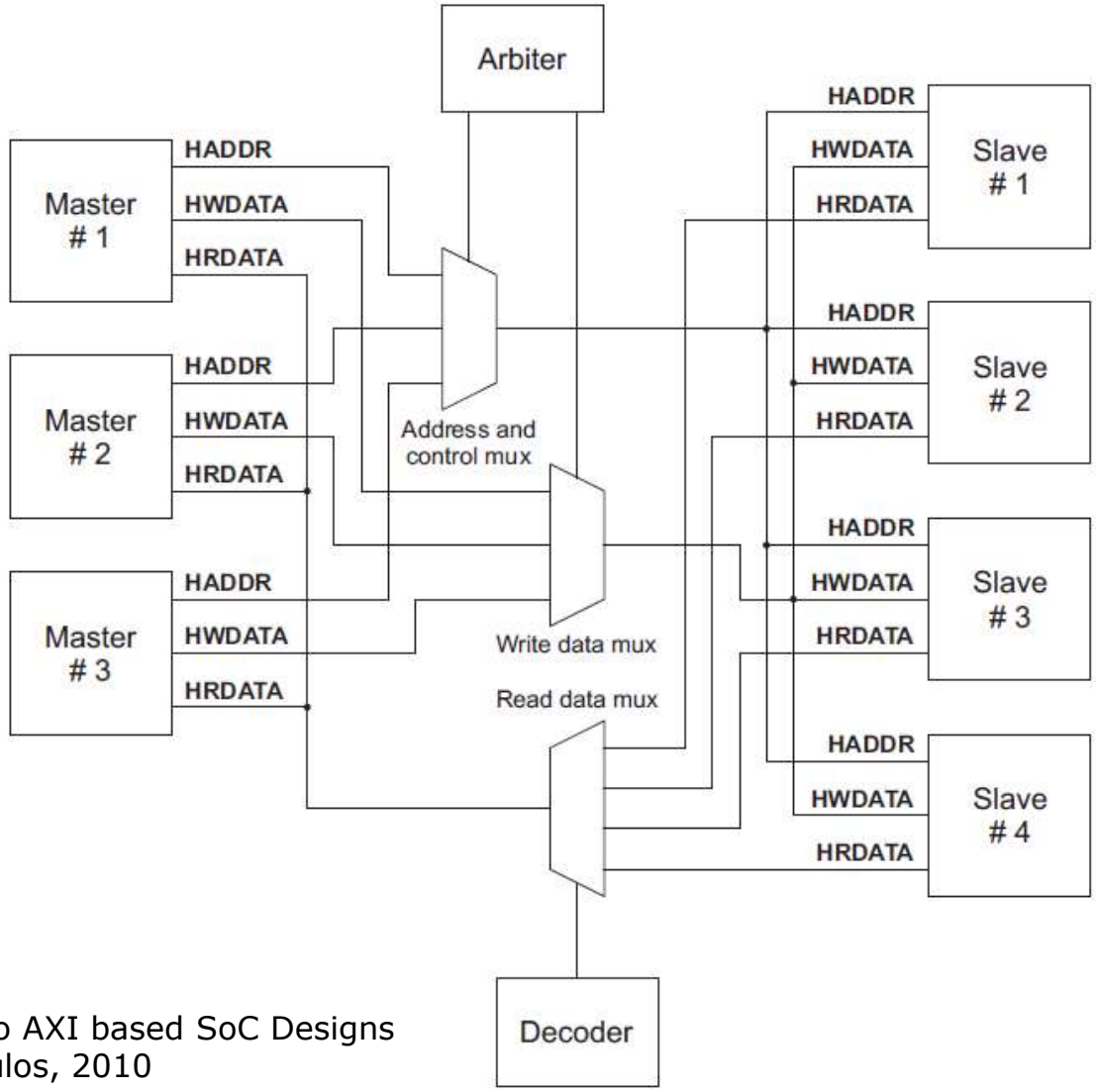
This widens the choice of available ASIC design tools.

e) Using only the rising edge of the bus clock

Benefit

This easies circuit synthesis.

Example AHB bus for three masters and four slaves-1 []



Migrating from AHB to AXI based SoC Designs
Marcus Harnisch, Doulos, 2010

Remark []

The four master ports might include each one CPU core, Direct Memory Access (DMA), DSP and USB.

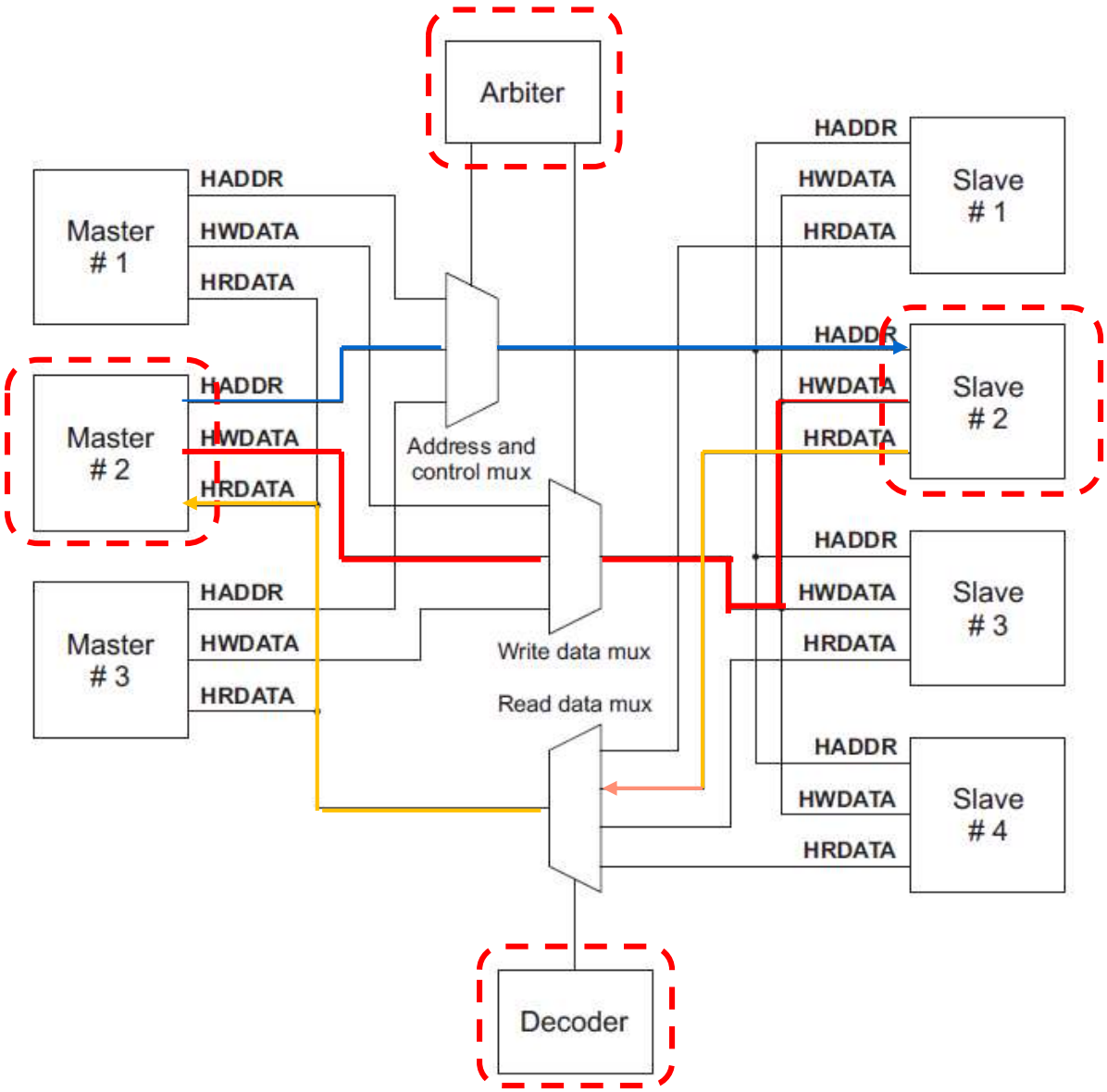
The three slaves might be on-chip RAM, off-chip SDRAM and an APB bus bridge.

http://www.design-reuse.com/articles/24123/amba-ahb-to-axi-bus-comparison.html

Example operation of the AHB bus for three masters and four slaves []

- The arbiter determines which master is granted access to the bus,
- When granted, a master initiates transfers on the bus,
- The decoder uses the high order address lines (BADDR) to select a bus slave,
- The slave provides a transfer response back to the bus master (not shown) and
- The data (read data (HRDATA) or write data (HWDATA)) is transferred between the master and the slave.

  (The transfer of write data is shown).



5.3.3 The APB2 bus (APB bus 2. Revision) []

APB2 ensures that all signal transitions are only related to the rising edge of the clock.

This modification allows APB peripherals to be integrated easily into any design flow with the following advantages in circuit design:

- performance is improved at high-frequency operation,
- the use of a single clock edge simplifies static timing analysis,
- many ASIC (Application Specific Integrated Circuit) libraries have a better selection of rising edge registers,
- cycle based simulators can be integrated more easily.

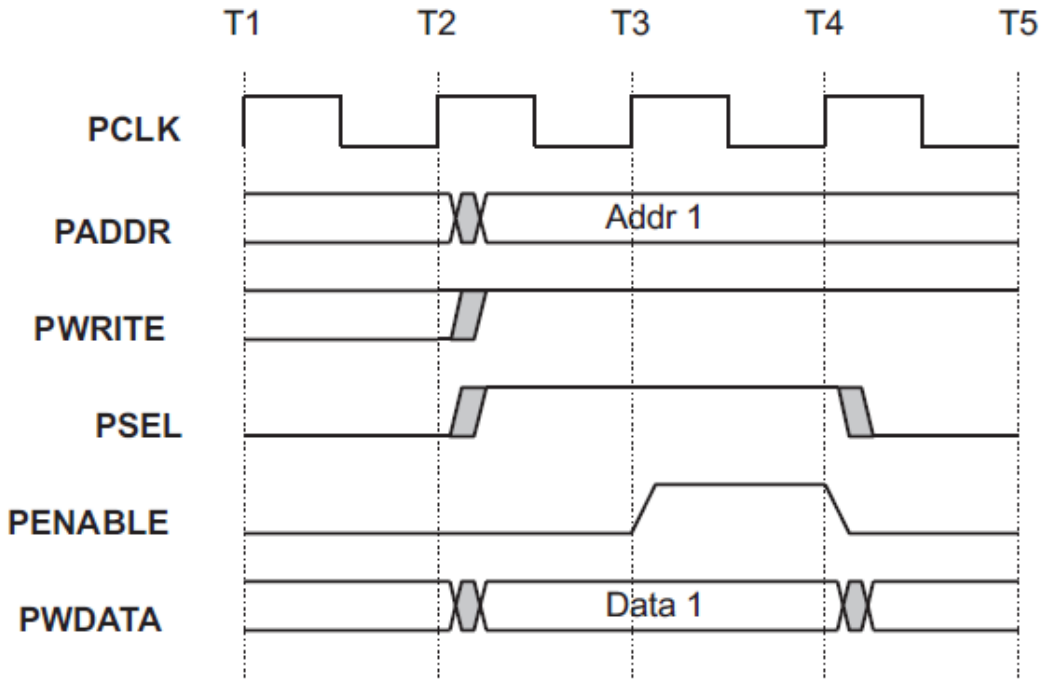
Nevertheless, the APB2 protocol further on does not support any pipelining of the address and control signals.

AMBA™ Specification

(Rev 2.0)

Date	Issue	Change
13th May 1999	A	First release

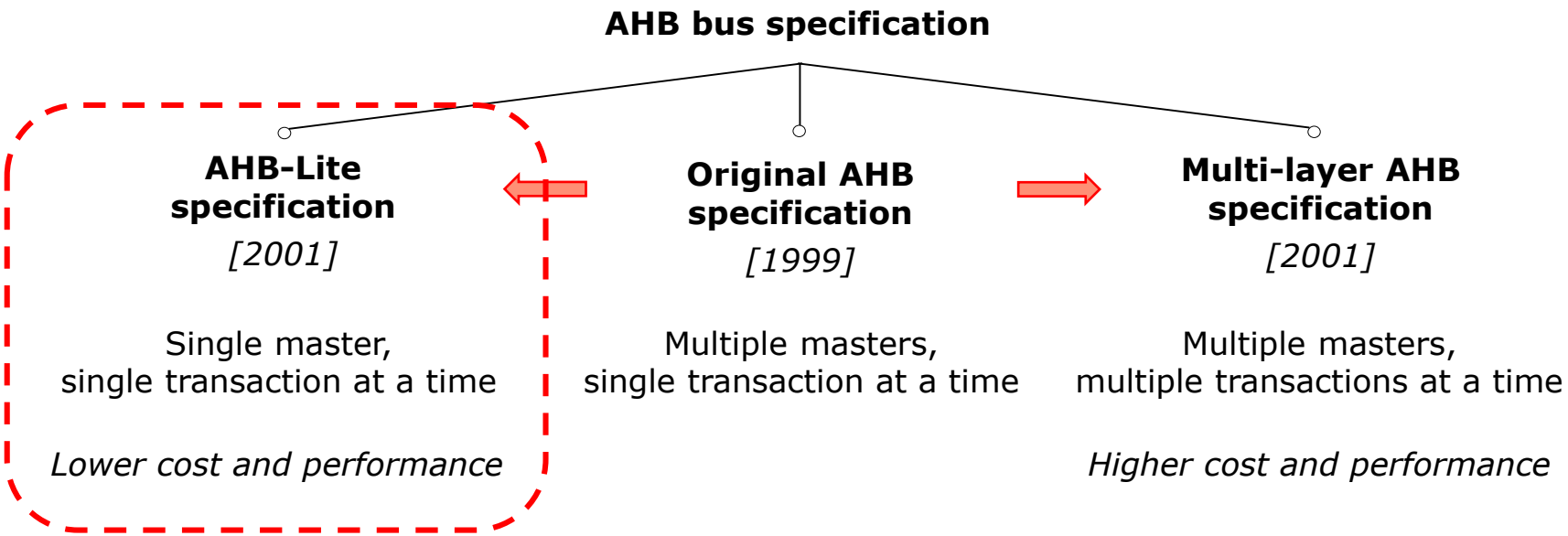
Example APB2 write transfer



- The write transfer starts with the address, write data cycle signal and select signal all changing after the rising edge of the clock.
- After the following clock edge the enable signal (PENABLE) is asserted, and this indicates, that the ENABLE cycle is taking place.
- The address, data and control signals all remain valid through the ENABLE cycle.
- The transfer completes at the end of this cycle when the PENABLE signal becomes deasserted.

5.3.4 The AHB-Lite bus extension

In 2001 ARM extended the original AHB bus is two directions [], as shown below:



http://www.design-reuse.com/news/856/arm-multi-layer-ahb-ahb-lite.html

The AHB-Lite specification

- The AHB-Lite bus was launched along with the Multi-layer AHB specification in 2001 as an extension of the AHB bus [a].

  Subsequently it was specified in a stand alone document in 2006 [b].
- The AHB-Lite bus is considered as being part of the AMBA 2 protocol family.

[a] http://www.design-reuse.com/news/856/arm-multi-layer-ahb-ahb-lite.html

[b] AMBA 3 AHB-Lite Protocol
v1.0

Specification

2001, 2006 ARM Limited.
ARM IHI 0033A

Key features of the AHB-Lite bus []

- AHB-Lite is a subset of AHB.
- It simplifies platform designs including only a single master.

Key featurures

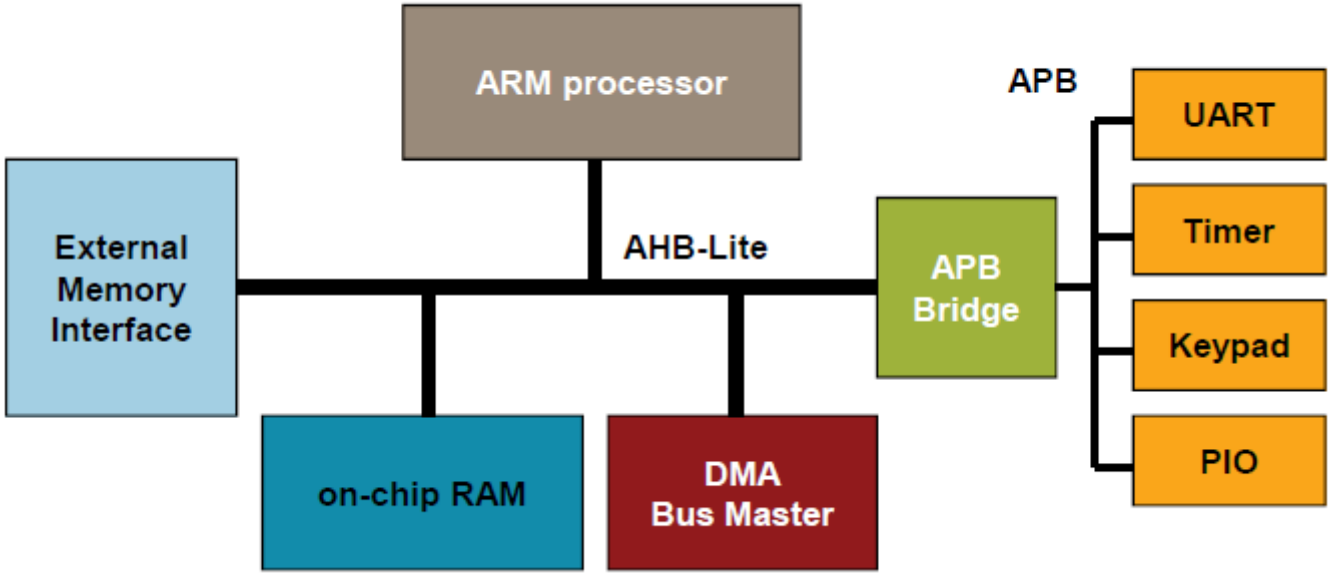
- Single Master
- Simple slaves
- Easier module design/debug
- No arbitration issues

ARM® Cortex M0 Design Start
Karthik Shivashankar
Senior Engineer, ARM R&D
Cambridge, UK

Section 2
CORTEX MO – SYSTEM DESIGN

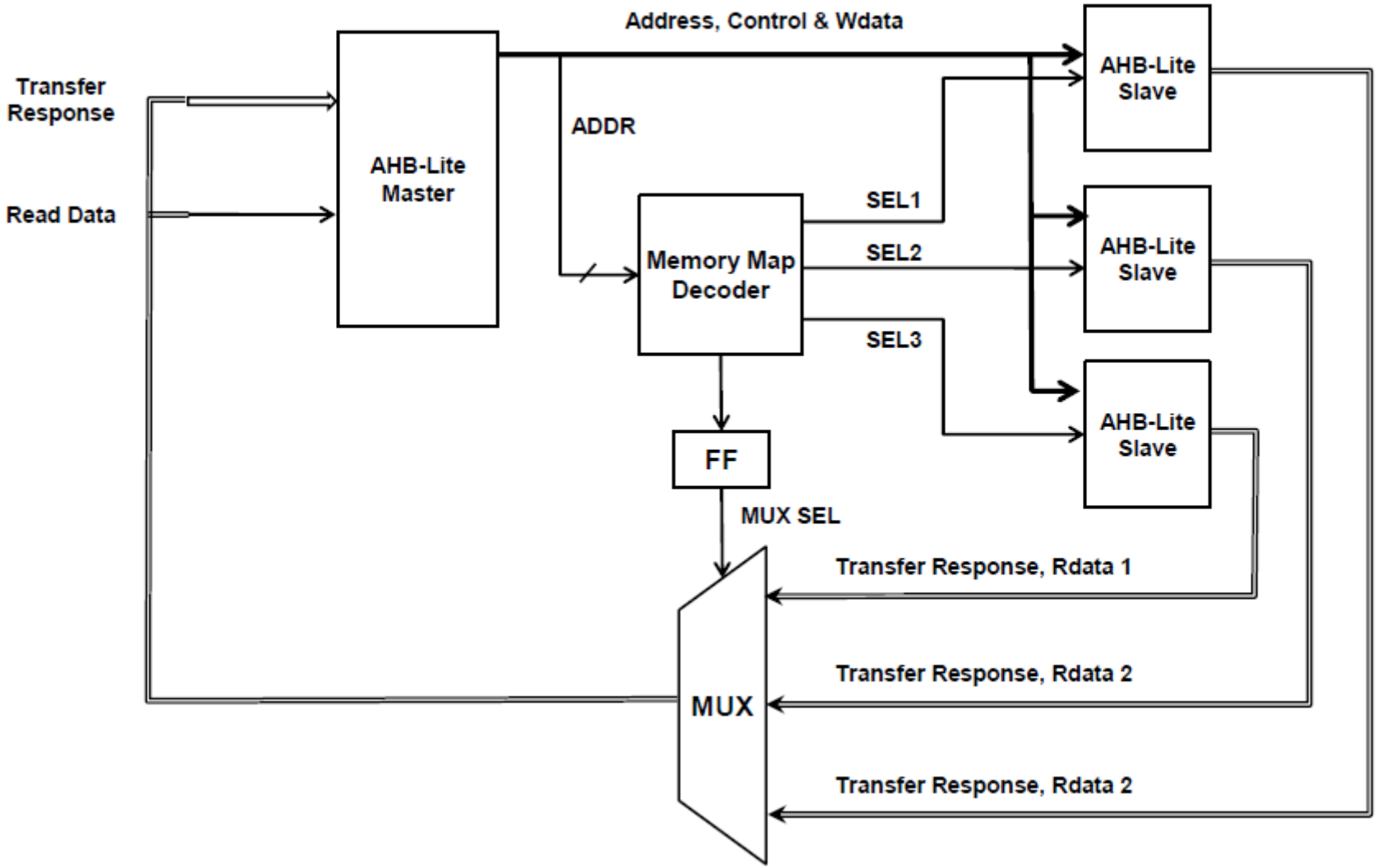
http://www.hipeac.net/system/files/cm0ds_2_0.pdf

An example AMBA system based on the AHB-Lite bus []



http://web.mit.edu/clarkds/www/Files/slides1.pdf

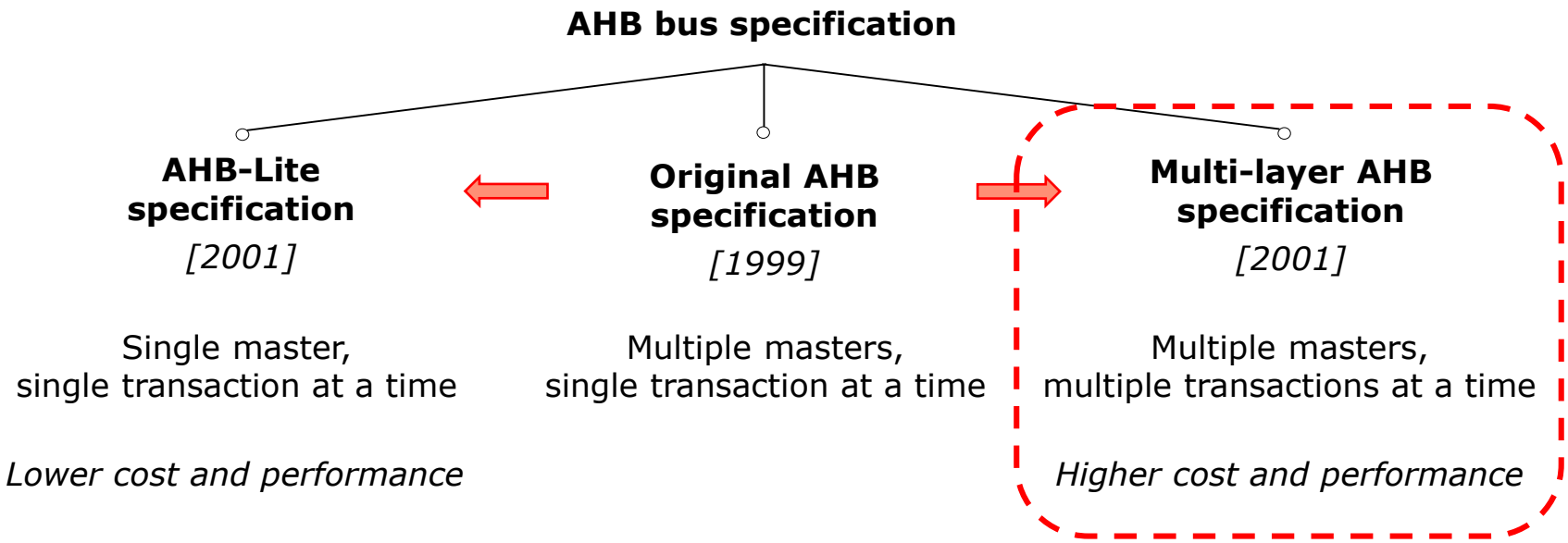
Block diagram of an example AMBA system based on the AHB-Lite bus []



http://www.hipeac.net/system/files/cm0ds_2_0.pdf

5.3.5 The Multi-layer AHB bus

In 2001 ARM extended the original AHB bus is two directions [], as shown below:



http://www.design-reuse.com/news/856/arm-multi-layer-ahb-ahb-lite.html

The AHB bus interconnect

AHB bus interconnect

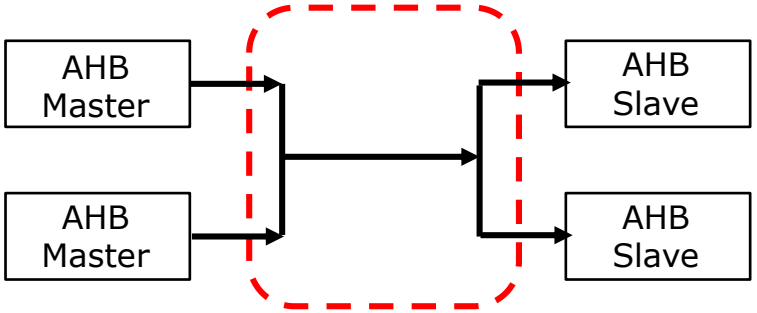
Single-layer AHB interconnect

Multiple masters, single transaction at a time

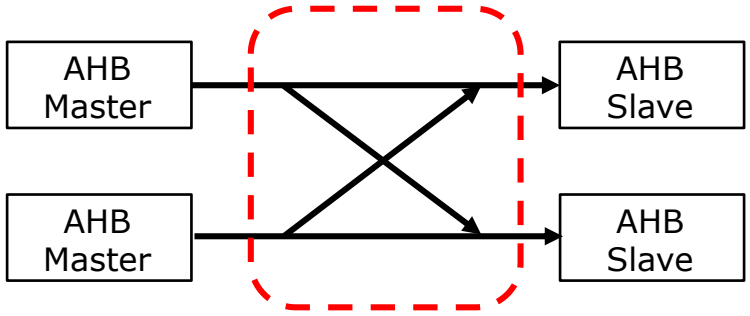
Multi-layer AHB-interconnect

Multiple masters, multiple transactions at a time

Principle of the interconnect
(Only the Master to Slave direction shown)



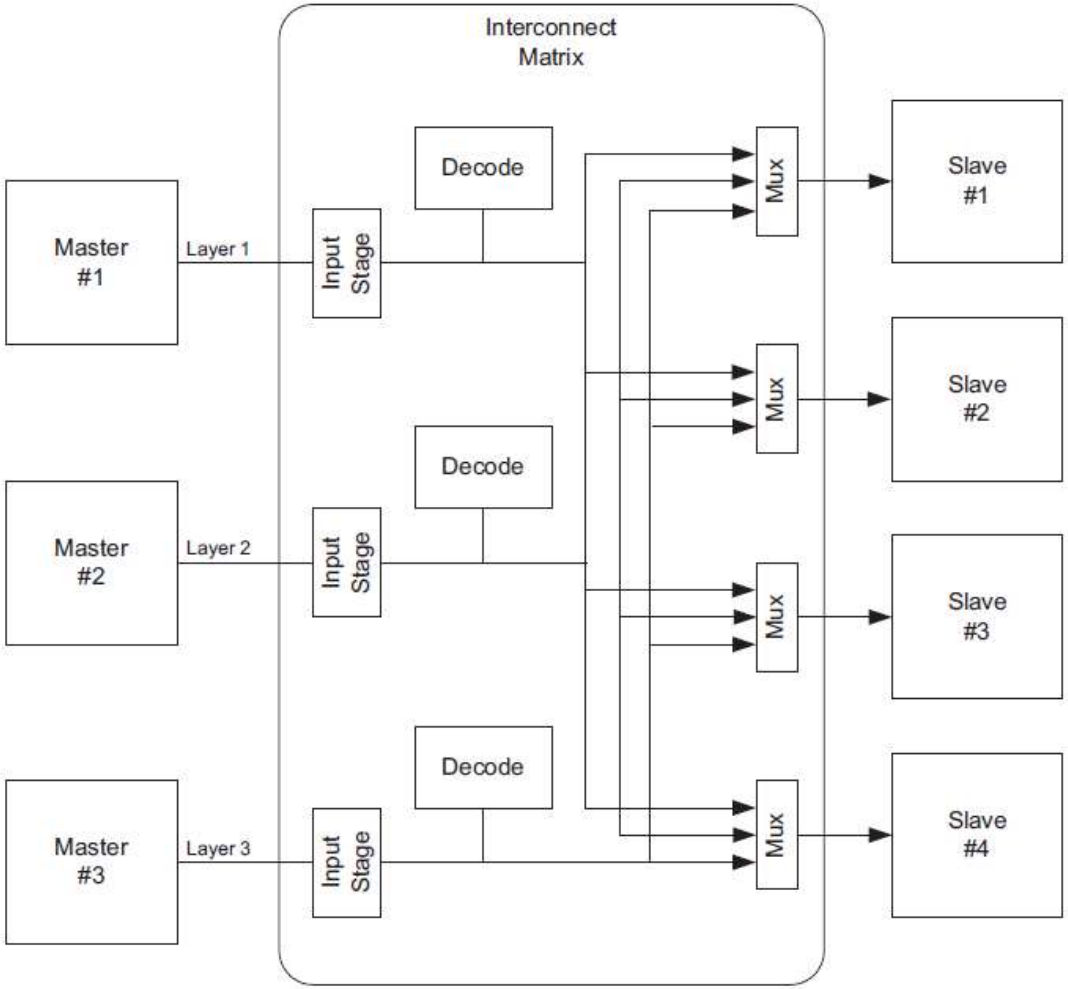
Shared bus



Crossbar

Multi-layer AHB Overview
DVI 0045A
ARM Limited. 2001

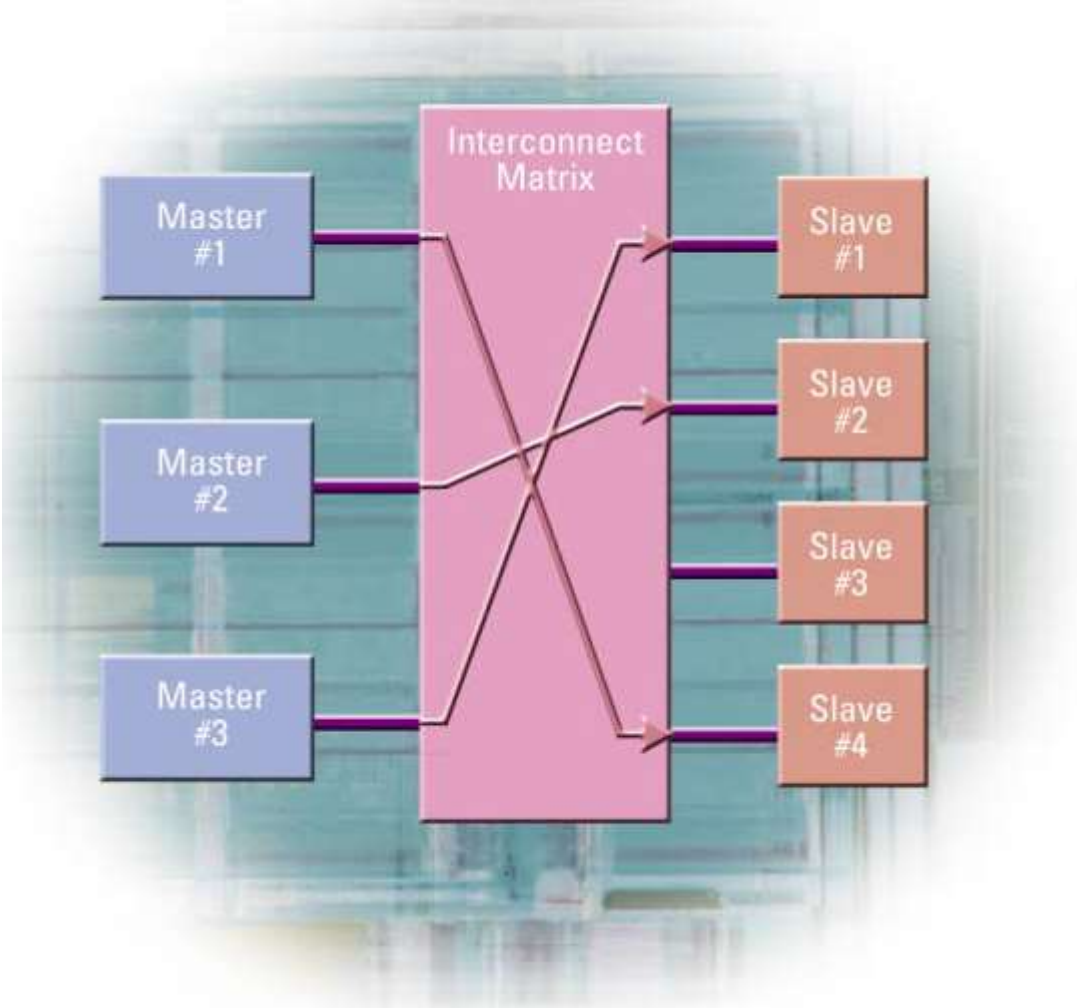
Block diagram of a three Masters/four Slaves multi-layer interconnect [] (Only the Master to Slave direction is shown)



Multi-layer AHB Overview
DVI 0045A
ARM Limited. 2001

http://pdf.datasheetarchive.com/indexerfiles/Datasheets-SL1/DSASL001562.pdf

Example operation of a three Masters/four Slaves multi-layer interconnect []
(Only the Master to Slave direction is shown)



http://www.13thmonkey.org/documentation/ARM/multilayerAHB.pdf

Main benefits of a multi-layer AHB interconnect []

- It allows multiple transactions from multiple masters to different slaves at a time, in fact, implementing a corossbar interconnect, as indicated in the next Figure.

  This results in increased bandwidth.
- Standard AHB master and slave modules can be used without modification.

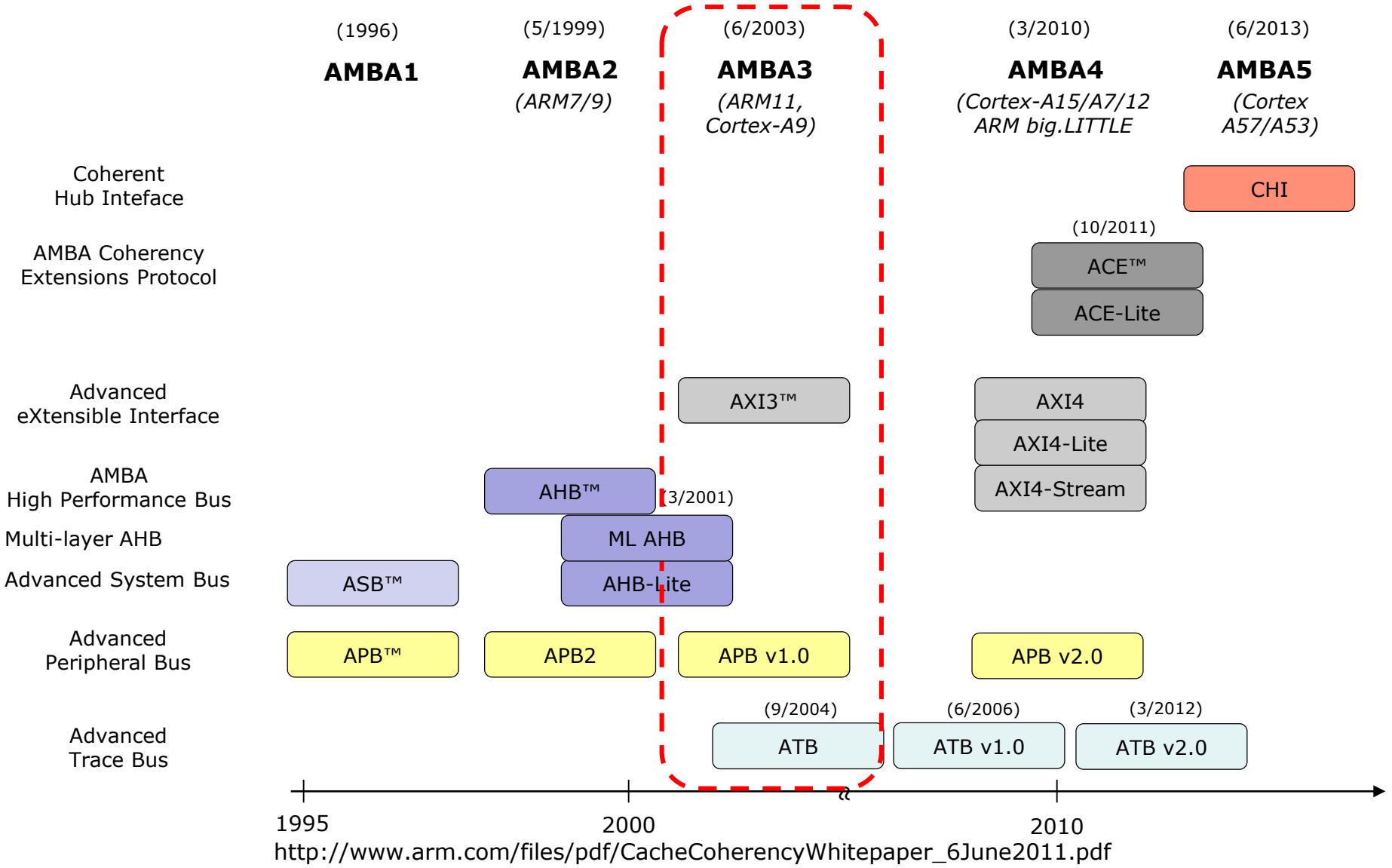
  The only hardware that has to be added to the standard AHB solution is the multiplexor block needed to connect multiple masters to the slaves.

Multi-layer AHB Overview
DVI 0045A
ARM Limited. 2001

http://pdf.datasheetarchive.com/indexerfiles/Datasheets-SL1/DSASL001562.pdf

5.4 The AMBA 3 protocol family

5.4 The AMBA 3 protocol family
5.4.1 Overview



http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Components of the AMBA3 protocol family

The AMBA 3 protocol family (Revision 3.0) includes the specifications for the buses

- AXI3 (Advanced eXtensible Interface),
- APB v1.0 (APB3, Advanced Peripheral Bus Revision 3) and
- ATB (Advanced Trace Bus),

as indicated in the previous Figure.

5.4.2 The AXI3 interface (Advanced eXtensible Interface) []

It is a complete redesign of the AHB bus.

The AXI bus specification became very complex and underwent a number of revisions, as indicated below [].

- Issue A is the original AXI specification, it was published in 6/2003.
- Issue B is the revised version (3/2004), it is now called the AXI3.
- Issue C (3/2010) adds an extended version of the protocol called AXI4 and also a new simplified protocol, the AXI4-Lite, that provides a subset of AXI4 for applications that do not require the full functionality of AXI4.
- Issue D (10/2011) integrates the definitions of AXI3 and AXI4 which were presented separately in Issue C.
- Issue E (2/2013) is a second release of the former specification without any modifications concerning the AXI4 interface.

AMBA® AXI™ and ACE™ Protocol
Specification

ARM IHI 0022E (ID022613)

Remark

The reason why the original AXI specification (Issue A) is not considered yet as the AXI3 specification is that this version foresees only four channels for the transmissions between masters and slaves instead of five as the AXI3 specification does.

Actually, in the original specification both the read and write addresses were transmitted over the same channel.

ARM presumable indicated this as a bottleneck and in the next issue (Issue B) they provided already separate channels for read and write addresses.

(The channel concept of AXI3 will be discussed in one of the next Sections).

Key innovations and enhancements of the AXI protocol are:

a) burst-based transactions,

b) the channel concept for performing reads and writes,

c) support for out-of-order transactions and

d) optional extension by signaling for low-power operation.

AMBA® AXI™ and ACE™ Protocol
Specification

ARM IHI 0022E (ID022613)

a) Burst-based transactions

In the AXI protocol all transfers are specified as burst transfers.

Allowed burst lengths

AXI3

- 1-16 transfers for all burst types
  but only the burst lengths 2, 4, 8 or 16 for wrapping bursts.
- Burst length is given by the signals
  - ARLEN[3:0], for read transfers
  - AWLEN[3:0], for write transfers

AXI4

- 1-256 transfers for incrementing bursts, else 1-16 transfers
  but only the burst lengths 2, 4, 8 or 16 for wrapping bursts.
- Burst length is given by the signals
  - ARLEN[7:0], for read transfers
  - AWLEN[7:0], for write transfers.

AMBA® AXI™ and ACE™ Protocol Specification

b) The channel concept for performing reads and writes

The channel concept incorporates a number of sub-concepts, as follows:

b1) Splitting reads and writes (actually read bursts and write bursts) into a series of transactions

b2) Providing individual channels for each phase of the transactions

b3) Providing a two-way handshake mechanism for synchronizing individual transaction phases

b4) Identifying different phases of the same transaction

b1) Splitting reads and writes (actually read bursts and write bursts) into a series of transactions []

Communication between a master and a slave is transaction-based, where each communication (such as a read or write burst) is split into a series of address, data, and response transactions, as follows.

A write burst will be set up of the following three transactions:

- A write address transaction,
- a write data transaction and
- a write response transaction, whereas

A read burst is set up of the following two transactions:

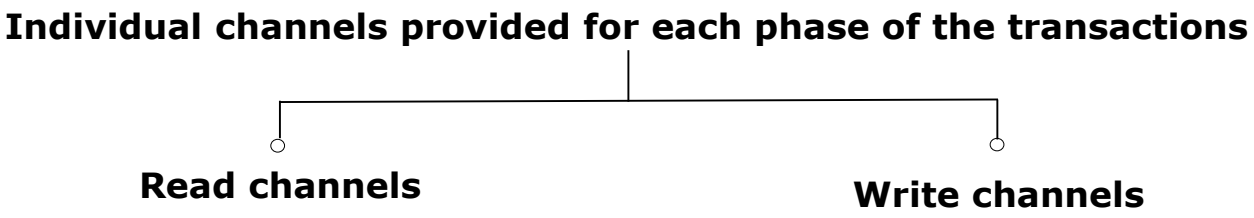
- A read address transaction and
- a read data transaction accompanied by a read response transaction.

We call these communication phases transactions since each of the communication phases is synchronized based on handshaking and is implemented as a two-way transfer of signals, as will be detailed later.

Each transaction is carried out over a dedicated channel, as detailed next.

b2) Providing individual channels for each phase of the transactions

Individual channels provided may be read or write channels, as indicated next and discussed subsequently.



Read channels-1

The AXI protocol defines the following channels for reads:

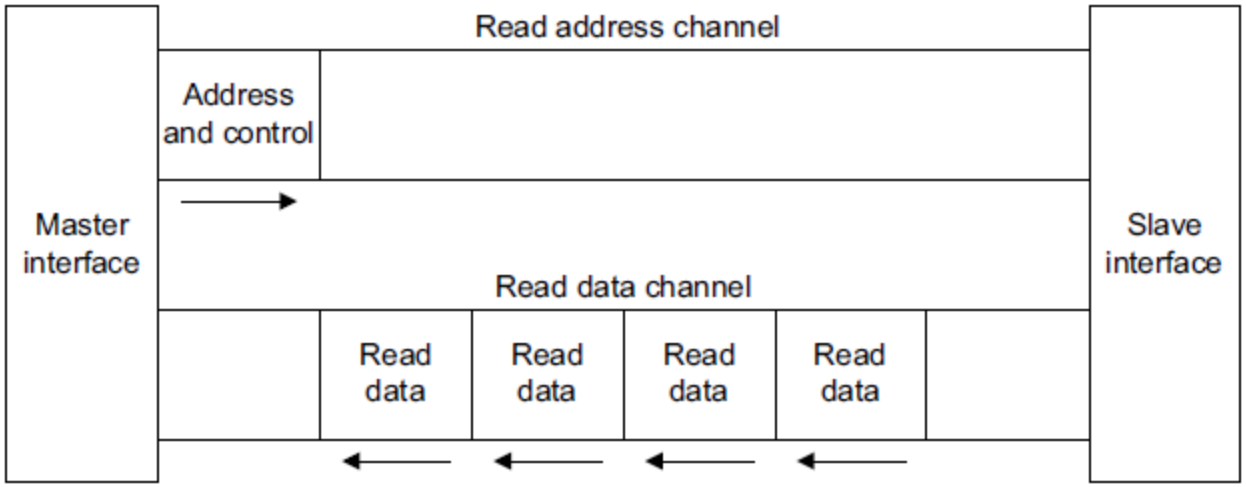


Figure: The channel architecture for reads []

Read channels-2

There are two independent channels provided for reads:

- The Read address channel

  It provides all the required address and control information needed for a read performed as a read burst.

- The Read data channel

  The read data channel carries both the read data and the read response information from the slave to the master, and includes:

  - the data bus, that can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide and
  - a read response signal indicating the completion status of the read transaction.

  It provides the read data sent during the burst transfer from the slave to the master.

Write channels-1

The AXI protocol defines the following channels for writes:

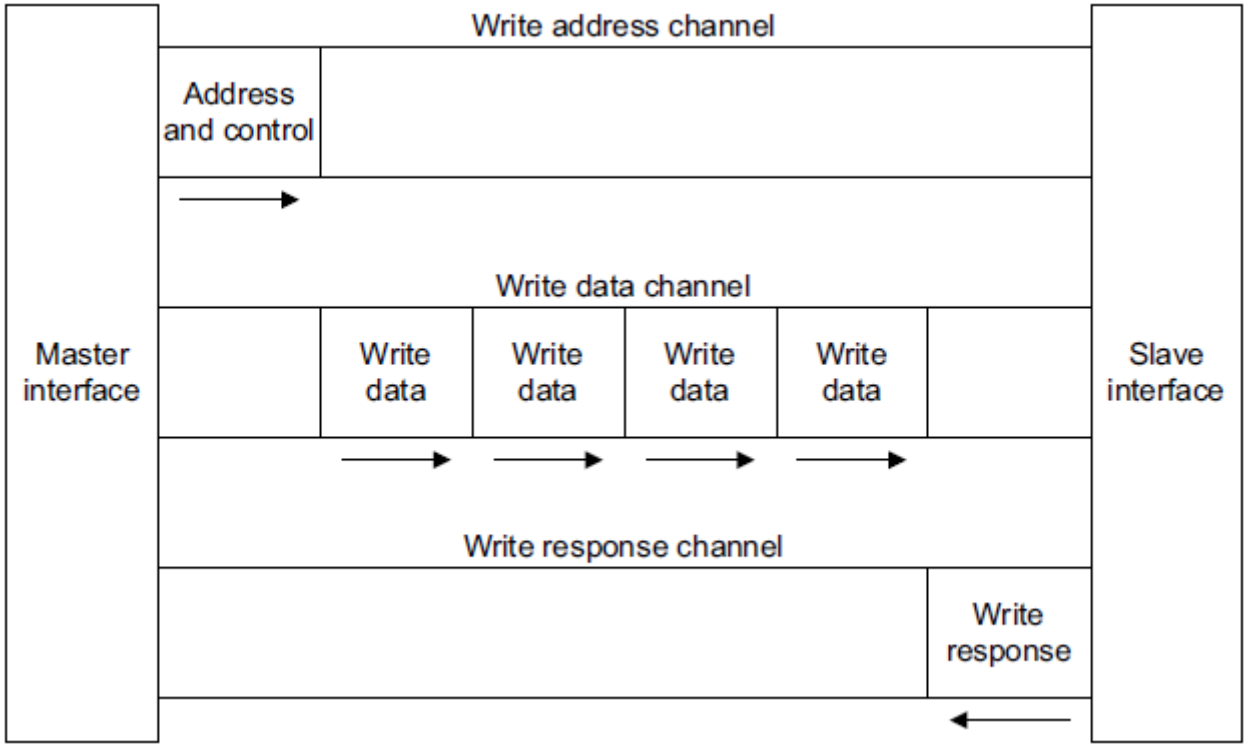


Figure: The channel architecture for writes []

Write channels-2

There are three independent channels provided for writes:

- The Write address channel

  It provides all of required address and control information needed for a write performed as a write burst.

- The Write data channel

  It provides the write data sent during the burst transfer from the master to the slave.

  Write data channel operation is always treated as buffered, so that the master can perform write transactions without slave acknowledgement of previous write transactions.

  The write data channel carries the write data from the master to the slave and includes:

  - the data bus, that can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide and
  - a byte lane strobe signal for every eight data bits, indicating which bytes of the data are valid.

- The Write response channel

  The slave uses the write response channel to signal the completion of the write transfer back to the master.

b3) Providing a two-way handshake mechanism for synchronizing individual transaction phases

Each of the five independent channels carries a set of information signals and two synchronization signals, the VALID and READY signals that implement a two-way handshake mechanism.

The VALID signal

It is generated by the information source to indicate when the information sent (address, data or control information) becomes available on the channel.

The READY signal

It is generated by the destination to indicate when it can accept the information.

The LAST signal

Both the read data channel and the write data channel also include a LAST signal to indicate the transfer of the final data item in a transaction.

See *Basic read and write transactions* on page A3-37.

Principle of handshaking by means of the VALID and READY signals []

Each of the independent channels (write address, write data, write response, read address, read data/response) uses the same straightforward handshaking mechanism to synchronize source and destination (master or slave) operation, as shown below.

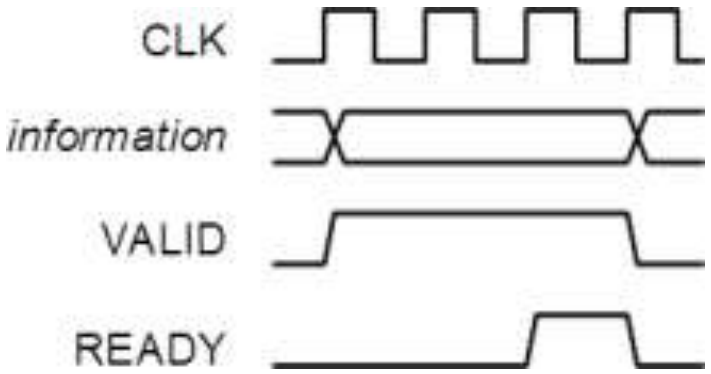


Figure: AXI bus channel handshaking mechanism []

The VALID signal indicates the validity of the information sent from the master to the slave whereas the READY signal acknowledges the receipt of the information.

This straightforward synchronization mechanism simplifies the interface design.

http://www.doulos.com/knowhow/arm/Migrating_from_AHB_to_AXI/

Providing sets of handshake and LAST signals for each channel

There is a different set of handshake and LAST signals for each of the channels, e.g. the Write address channel has the following set of handshake and LAST signals:

WVALID (Write valid): Sent by the master.

This signal indicates that valid write data and strobes are available:

1 = write data and strobes available

0 = write data and strobes not available.

WREADY (Write ready): Sent by the slave.

This signal indicates that the slave can accept the write data:

1 = slave ready

0 = slave not ready.

WLAST (Write last): Sent by the master.

This signal indicates the last transfer in a write burst.

Ordering of transactions []

Apart from trivial ordering rules there is no strict protocol-enforced timing relation between individual phases of a communication, instead every transaction identifies itself as part of a specific communication by a unique transaction ID tag, as indicated in the next Figure.

http://www.doulos.com/knowhow/arm/Migrating_from_AHB_to_AXI/

b4) Identifying different phases of the same transaction-1

- Each transaction is identified by an ID tag that allows to order related transaction phases to individual read or write bursts.
- ID tags support multi-master out-of-order transactions for increasing data throughput, as out-of-order transactions can be sorted out at the destination.

b4) Identifying different phases of the same transaction-2

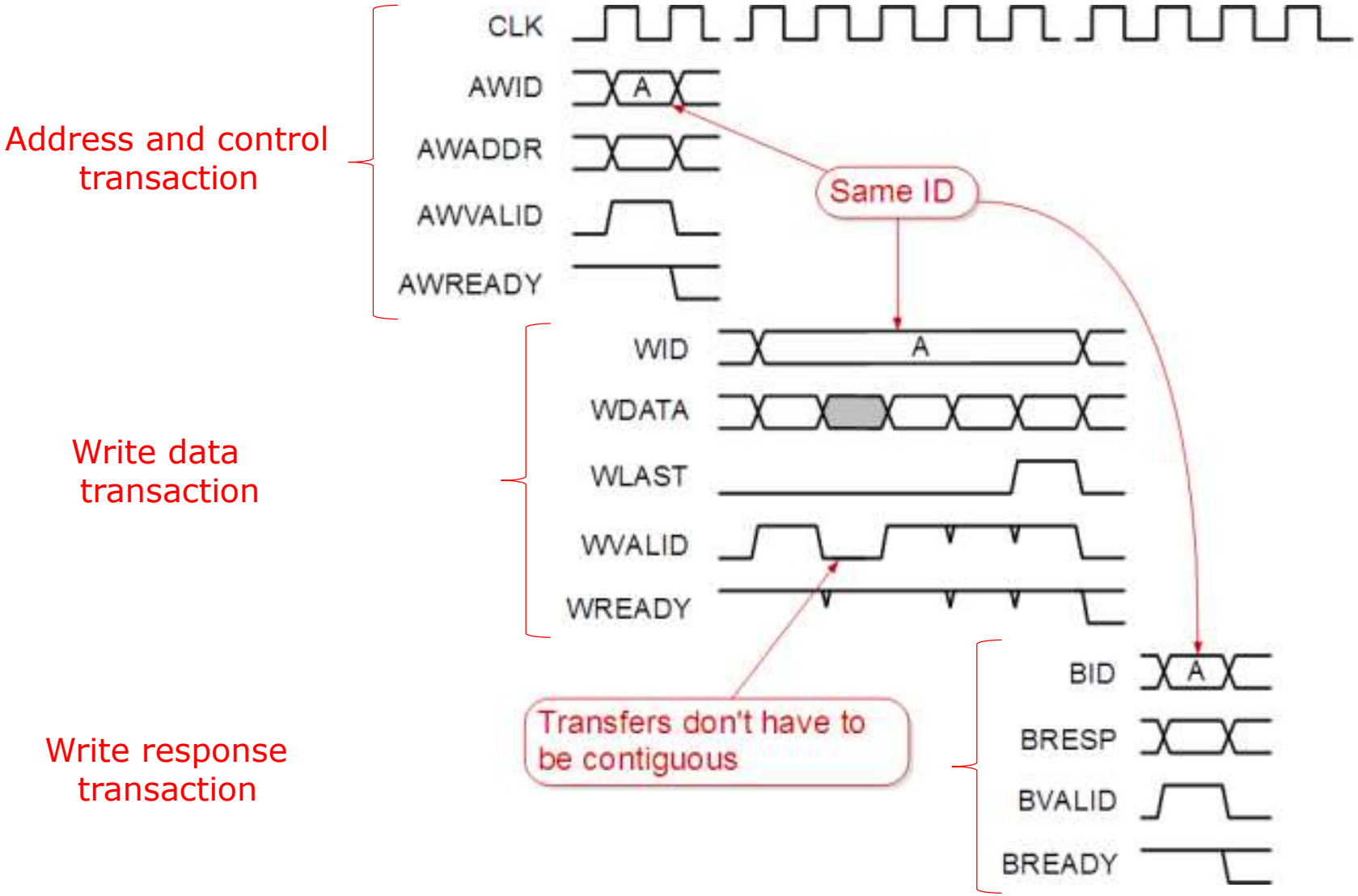
There are individual four bit long ID tags for each of the five transaction channels, as follows:

- AWID: The ID tag for the write address group of signals.
- WID: The write ID tag for a write burst.
  Along with the write data, the master transfers a WID to match the AWID of the corresponding address.
- BID: The ID tag for the write response.
- The write response (BRESP) indicates the status of the write burst performed (OK etc.).
  The slave transfers a BID to match the AWID and WID of the transaction to which it is responding.
- ARID: The ID tag for the read address group of signals.
- RID: The read ID tag for a read burst.
  The slave transfers an RID to match the ARID of the transaction to which it is responding.

b4) Identifying different phases of the same transaction-3

All transactions with a given ID tag must be ordered to an individual read or write burst (as indicated in the next Figure), but transactions with different ID tags need not be ordered.

Example: Identification the three phases of an AXI write burst []



http://www.doulos.com/knowhow/arm/Migrating_from_AHB_to_AXI/

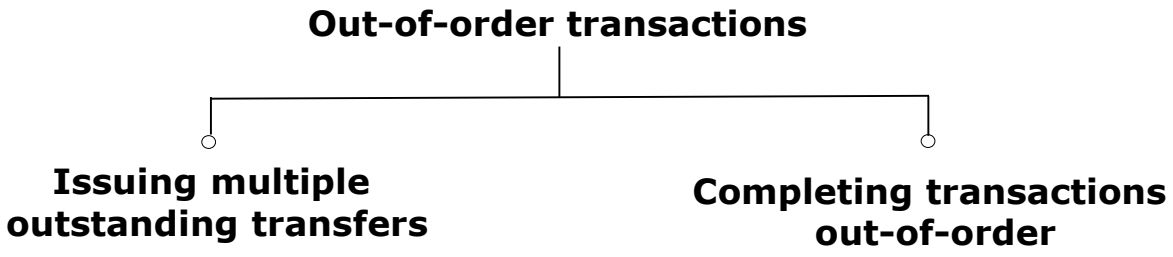
c) Support of out-of-order transactions

The AXI protocol (AXI3 and its revisions, such as the AXI4) allows out-of-order transactions to provide higher performance compared with the AHB protocol.

Out-of-order transactions are supported when the bus protocol allows

- issuing multiple outstanding transfers and
- completing transactions out-of-order,

as indicated below.



c1) Support for issuing multiple outstanding transfers

The ability to issue multiple outstanding transfers means that masters can initiate transactions (by issuing new transaction addresses) without waiting for earlier transactions to complete.

This feature can improve system performance because it enables parallel processing of transactions.

c2) Completing transfers out-of-order

The ability to complete transfers out-of-order means that transfers to faster memory regions can complete without waiting for earlier transactions to slower memory regions.

This feature can also improve system performance because it reduces transaction latency.

Note

There is no requirement for slaves and masters to use these advanced features.

Simple masters and slaves may process one transaction at a time in the order they are issued.

Implementing out-of-order transfers in the AXI3 protocol

Out-of-order transfers promise higher performance but the price for it is a more complex implementation resulting in higher cost and higher power consumption.

The implementation of out-of-order transfers is based on the ID-signals, briefly described before and a complex set of ordering rules, we do not want to discuss here, but refer to the related ARM documents (ARM IHI 0022B to ARM IHI 0022E).

Remarks to the ordering rules of transactions

As stated before, there is a complex set of ordering rules, we do not want to discuss here, nevertheless, to give a glimpse into these rules we give next an excerpt from them.

- The basic rules governing the ordering of transactions are as follows:
  - Transactions from different masters have no ordering restrictions.

    They can complete in any order.
  - Transactions from the same master, but with different ID values, have no ordering restrictions.

    They can complete in any order.
  - Transactions from the same master with the same ID values, have ordering restrictions.

    For these ordering restrictions we refer to the related ARM documents (ARM IHI 0022B to ARM IHI 0022E).

d) Optional extension by signaling for low-power operation []

It is an optional extension to the data transfer protocol that targets two different classes of peripherals:

- Peripherals that require a power-down sequence, and that can have their clocks turned off only after they enter a low-power state.

  These peripherals require an indication from a system clock controller to determine when to initiate the power-down sequence.

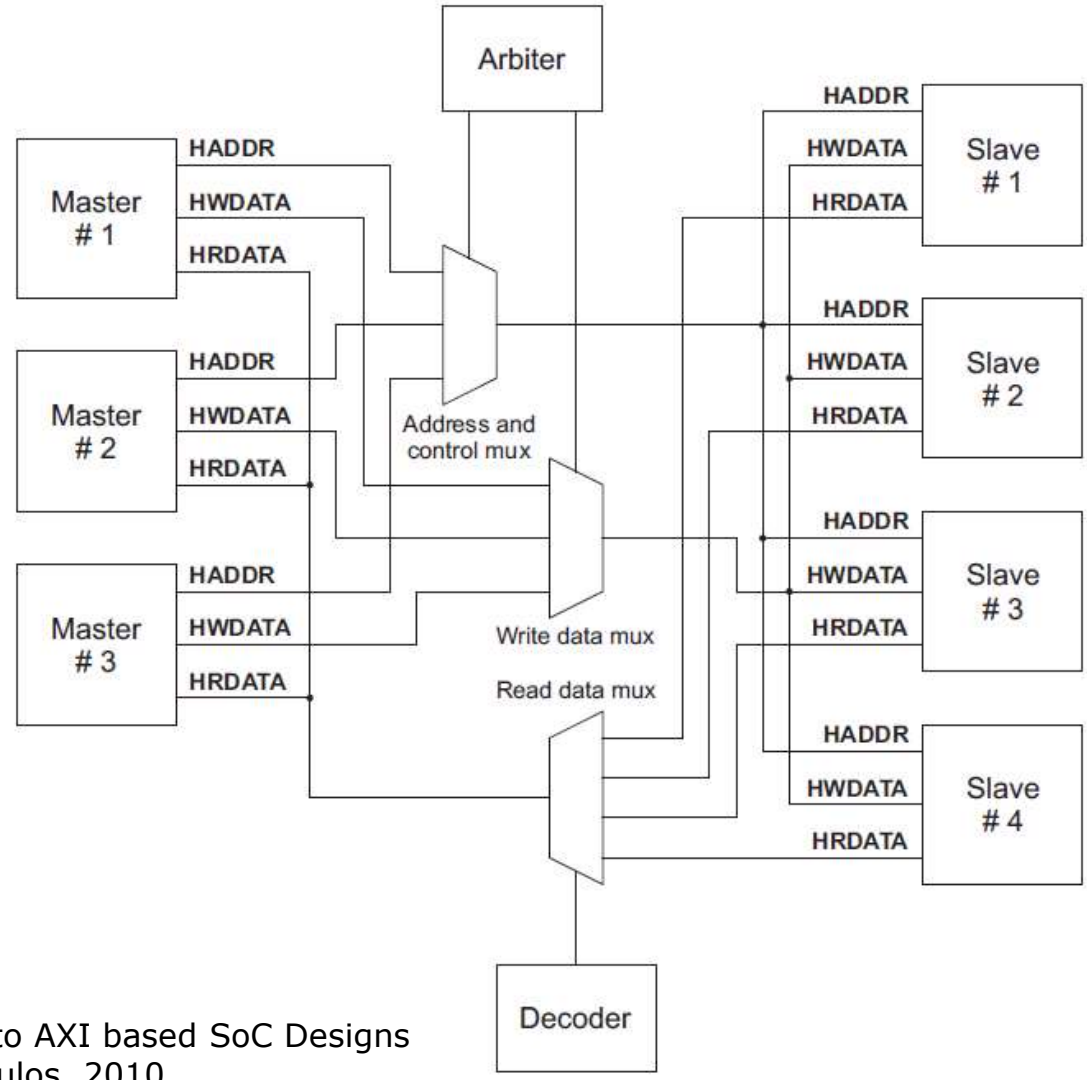
- Peripherals that have no power-down sequence, but can independently indicate when it is acceptable to turn off their clocks.

Interconnecting AXI masters and slaves-1 []

The AHB protocol is underlying a traditional bus architecture with the features:

- arbitration between multiple bus masters and
- multiplexing the signals of the masters and the slaves,

  as the next Figure shows.
- In addition, the AHB protocol allows for overlapping the address and data phases of transactions of different masters to two-stage pipeline bus operation (beyond the overlapped (third stage) bus granting operation).



Migrating from AHB to AXI based SoC Designs
Marcus Harnisch, Doulos, 2010

Interconnecting AXI masters and slaves-2 []

By contrast, the AXI3 protocol and its further revisions assume that masters and slaves are connected together in a more flexible way by some sort of an interconnect, as shown below.

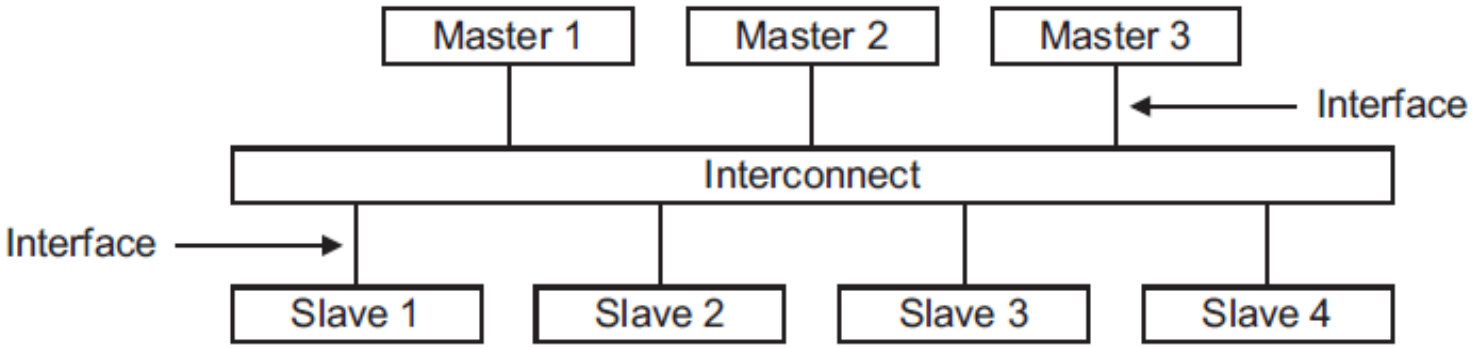


Figure: Assumed interconnect between masters and slaves in the AXI3 protocol []

Interconnecting AXI masters and slaves-3 []

The AXI protocol is based on five independent channels for each of the different transaction types, such as address and control transactions, read data transactions etc., as discussed before.

This fact gives the freedom to choose different interconnect types for different transaction types to optimize performance and cost depending on the expected data volume on a specific channel.

The basic alternatives for implementing AXI interconnects are

- a shared bus or
- a crossbar implemented as multilayer busses.

Shared buses have the limitation of allowing only a single transaction from a granted source to a specified destination at a time, whereas

crossbar switches allow multiple transactions from multiple sources to different destinations at a time, as shown in the next Figures.

Interconnecting AXI masters and slaves-4 - the master to slave directions []

Interconnecting AXI masters and slaves

Shared bus implementation (Single layer bus)

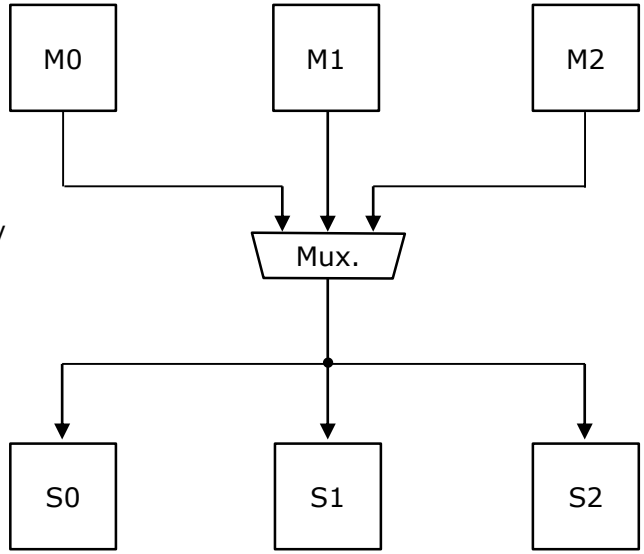
Up to a single transaction can be carried out at a time.

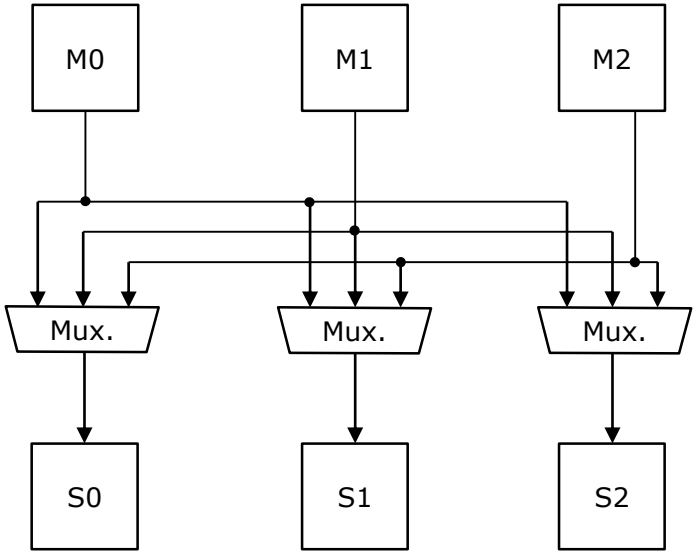
Crossbar implementation (Multi-layer bus)

Multiple transactions can be carried out at a time, nevertheless to different destinations (slaves/masters).

Example for a master to slave direction

E.g. address/ control or write data transaction





Interconnecting AXI masters and slaves-5 – the slave to master direction []

Interconnecting AXI masters and slaves

Shared bus implementation (Single layer bus)

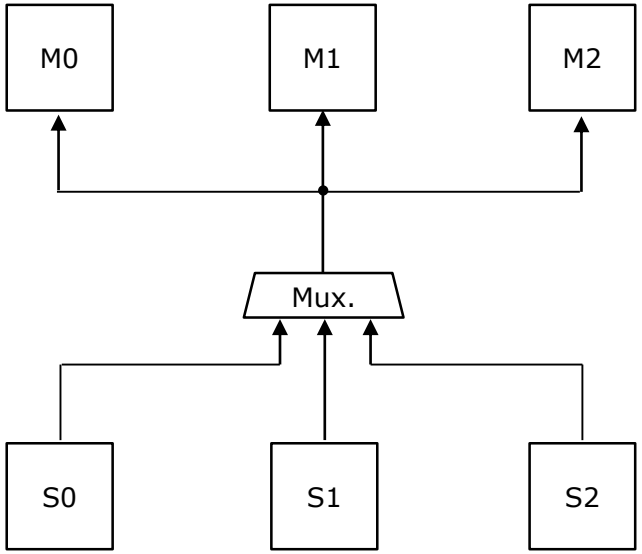
Up to a single transaction can be carried out at a time.

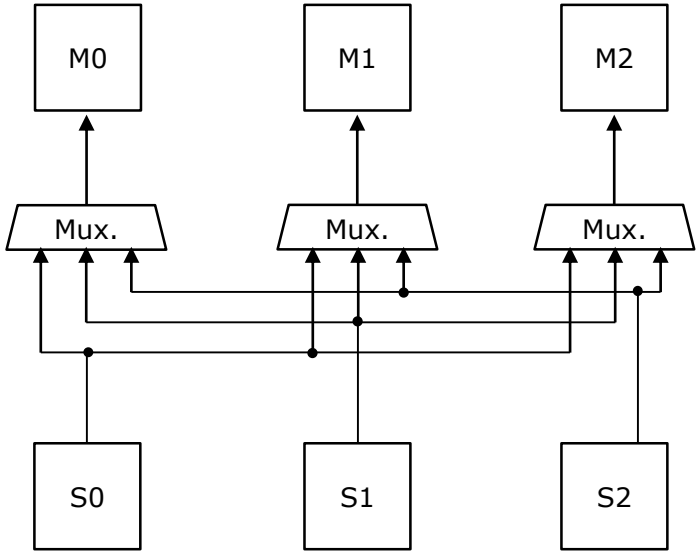
Crossbar implementation (Multi-layer bus)

Multiple transactions can be carried out at a time, nevertheless to different destinations (slaves/masters).

Example for a slave to master direction

E.g. read data transaction





Interconnecting AXI masters and slaves-6 []

- Crossbar switches provide multiple transfers at a time but impose much higher complexity and implementation cost than shared buses.
- On the other hand different transaction channels in an AXI interconnect carry different data volumes.

  E.g. read or write data channels will transfer more than a single data item in a transaction, whereas for example, read or write response channels or address channels transmit only a single data item per transaction.
- Given the fact that the AXI specification defines five independent transaction channels, obviously, in an AXI implementation it is possible to choose different interconnect types (shared bus or crossbar) for different transaction channels, depending on the expected data volume to optimize cost vs. performance. .
- Based on the above considerations, read and write data channels can be expected to be routed via crossbar switches whereas address and response channels via shared buses.

http://www.doulos.com/knowhow/arm/Migrating_from_AHB_to_AXI/

Remarks to the system layout []

In an actual system layout all components of an AXI system need to agree on certain parameters, such as *write buffer capability*, *read data reordering depth* and many others.

http://www.doulos.com/knowhow/arm/Migrating_from_AHB_to_AXI/

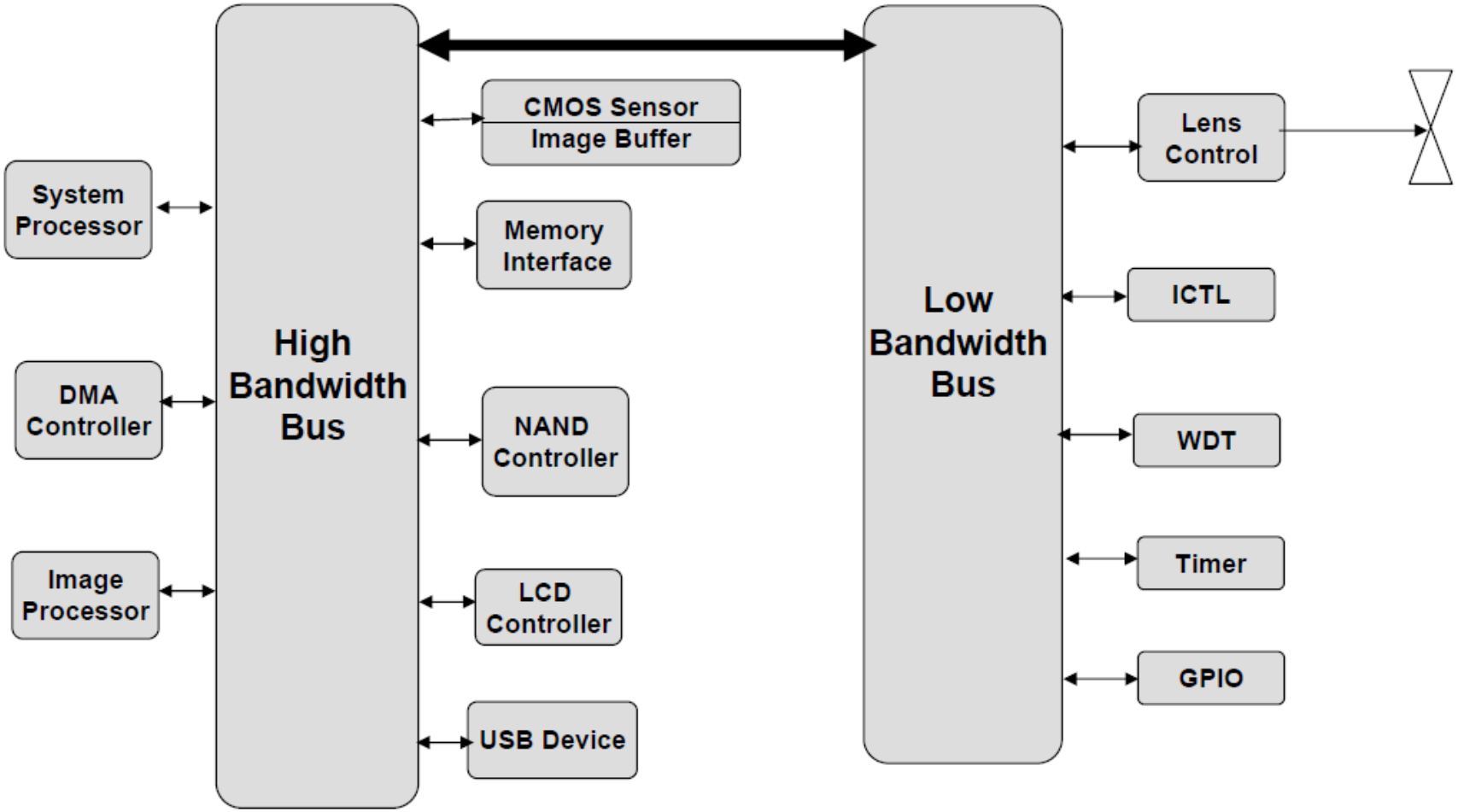
Throughput comparison AHB vs. AXI []

- For assessing the throughput of the AHB and the AXI bus it is appropriate to compare the main features of these buses, as follows:
- The AHB bus is a single-channel shared bus, whereas the AXI bus is a multi-channel read/write optimized bus.
- In case of the single-layer AHB bus all bus masters or requesting bus ports may use the same single-channel shared bus.
- In case of a multi-layer AHB bus each bus master or requesting port may use a different interconnect layer unless they request the same destinations.
- For the AXI bus each bus master or requesting bus may use one of the five channels (Read address channel, Read data channel, Write address channel, Write data channel, and Write response channel).

  Nevertheless, it is implementation dependent whether individual channels are built up as shared buses or as crossbars (multi-layer interconnects).

http://www.design-reuse.com/articles/24123/amba-ahb-to-axi-bus-comparison.html

Case example for comparing the bandwidth provided by the AHB and AXI buses – Digital camera []



http://rtcgroup.com/arm/2007/presentations/179%20-%20Comparative%20Analysis%20of%20AMBA%202.0%20and%20AMBA%203%20AXI%20Protocol-Based%20Subsystems.pdf

System requirements: Digital camera []

Requirements for a high-end compact consumer digital camera

- 10 mega pixel image sensor.
- 352 * 288 LCD display.
- Video record capability : 640 * 480 mpeg2 @ 30 frames per second.
- Raw image data output.
- USB interface.

What does this mean for the bandwidth requirements for our system ?

- 10 Mp image sensor produces 240 Mb of data per raw image.
- Video recording bandwidth requirements
    - To sample @ 30 fps requires a bandwidth of 30*240Mb/s = 7.2 Gb/s (uncompressed stream from the image buffer)
    - Compressed stream to the storage media requires bandwidth of 15Mb/s
- LCD display requires a bandwidth of 4 Mb/s for video data.

http://rtcgroup.com/arm/2007/presentations/179%20-%20Comparative%20Analysis%20of%20AMBA%202.0%20and%20AMBA%203%20AXI%20Protocol-Based%20Subsystems.pdf

AHB implementation []

- Results show a 4 master 8 slave DW_ahb bus will synthesize to 200MHz @ 90nm.
- For a data width of 32, peak system bandwidth is 32 * 200 = 6.4 Gb/s.
- Since AHB is a shared address and data bus architecture, other system traffic must be considered.

http://rtcgroup.com/arm/2007/presentations/179%20-%20Comparative%20Analysis%20of%20AMBA%202.0%20and%20AMBA%203%20AXI%20Protocol-Based%20Subsystems.pdf

AXI3 implementation []

- Results show a 4 master 8 slave DW_axi will synthesize to 400 Mhz @ 90nm.
- For a data width of 32, peak data bandwidth of 400 * 32 = 12.8 Gb/s on a single master/slave link.
- For all master slave links in read and write directions we get a system bandwidth of 2 (read and write channels) * 4 (number of masters) * 12.8 = 102.4 Gb/s.
- Separate address busses means there is no control overhead to affect bandwidth.
- Multiple address multiple data architecture reduces arbitration latencies.
- Register slicing can be used to ease I/O delay requirements while adding only 1 cycle of latency.

http://rtcgroup.com/arm/2007/presentations/179%20-%20Comparative%20Analysis%20of%20AMBA%202.0%20and%20AMBA%203%20AXI%20Protocol-Based%20Subsystems.pdf

5.4.3 The ATB bus (Advanced Trace Bus)

- The ATB bus was first described as part of the CoreSight on-chip debug and trace tool for AMBA 3 based SOCs, termed as the AMBA 3 ATB protocol in 2004 [a].

  Subsequently it was specified in a stand alone document in 2006 [] and designated as the AMBA 3 ATB protocol v1.0.

- This version is considered as being part of the AMBA 3 protocol family.
- It allows on-chip debugging and trace analysis for AMBA-based SoCs.
- Each IP in the SoC that has trace capabilities is connected to the ATB.
- Then master interfaces write trace data on the ATB bus, while slave interfaces receive trace data from the ATB [c].

Here we do not want to go into any details of the ATB bus.

[a] http://common-codebase.googlecode.com/svn/trunk/others/Cortex_M0_M3/CoreSight_Architecture_Specification.pdf

[b] AMBA 3 ATB Protocol
v1.0
Specification

2006 ARM Limited.
ARM IHI 0032A
AMBA™ 3 ATB Protocol
v1.0
Specification

[c] R. Sinha et al., *Correct-by-Construction Approaches for SoC Design*,
DOI 10.1007/978-1-4614-7864-5 2, © Springer Science+Business Media New York 2014

5.4.5 The APB3 bus

- It was published as part of the AMBA 3 protocol family in 2003.
- The APB3 bus can interface with the AMBA AHB-Lite and AXI 3 interfaces.
- There are only minor changes vs. the APB2 bus including the introduction of
  - a ready signal (PREADY) to extend an APB transfer, and
  - an error signal (PSLVERR) to indicate a failure of the transfer.

http://web.eecs.umich.edu/~prabal/teaching/eecs373-f11/readings/ARM_AMBA3_APB.pdf

ACP (Accelerator Coherency Port) [], []

Only in AMBA3?
ACE replaces it??

The ACP port is a standard (64 or 128-bit wide) AXI slave port provided for non-cached AXI master peripherals, such as DMA Engines or Cryptographic Engines.
It is optional in the ARM11 MPCore and mandatory in subsequent Cortex processors (except low-cost oriented processors, such as the Cotex-A7MPCore).
The AXI 64 slave port allows a device, such as an external DMA, direct access to coherent data held in the processor's caches or in the memory, so device drivers that use ACP do no need to perform cache cleaning or flushing to ensure cache coherency.

2.4 Accelerator Coherency Port

The *Accelerator Coherency Port* (ACP) is an optional AXI 64-bit slave port that can be connected to non-cached AXI master peripherals, such as a DMA engine or cryptographic engine.

This AMBA 3 AXI compatible slave interface on the SCU provides an interconnect point for a range of system masters that for overall system performance, power consumption or reasons of software simplification, are better interfaced directly with the Cortex-A9 MPCore processor. *ACP interface clocking* on page 2-21 describes ACP timing.

The following sections describe the ACP:

- *ACP requests*
- *ACP interface clocking* on page 2-21
- *ACP limitations* on page 2-21.

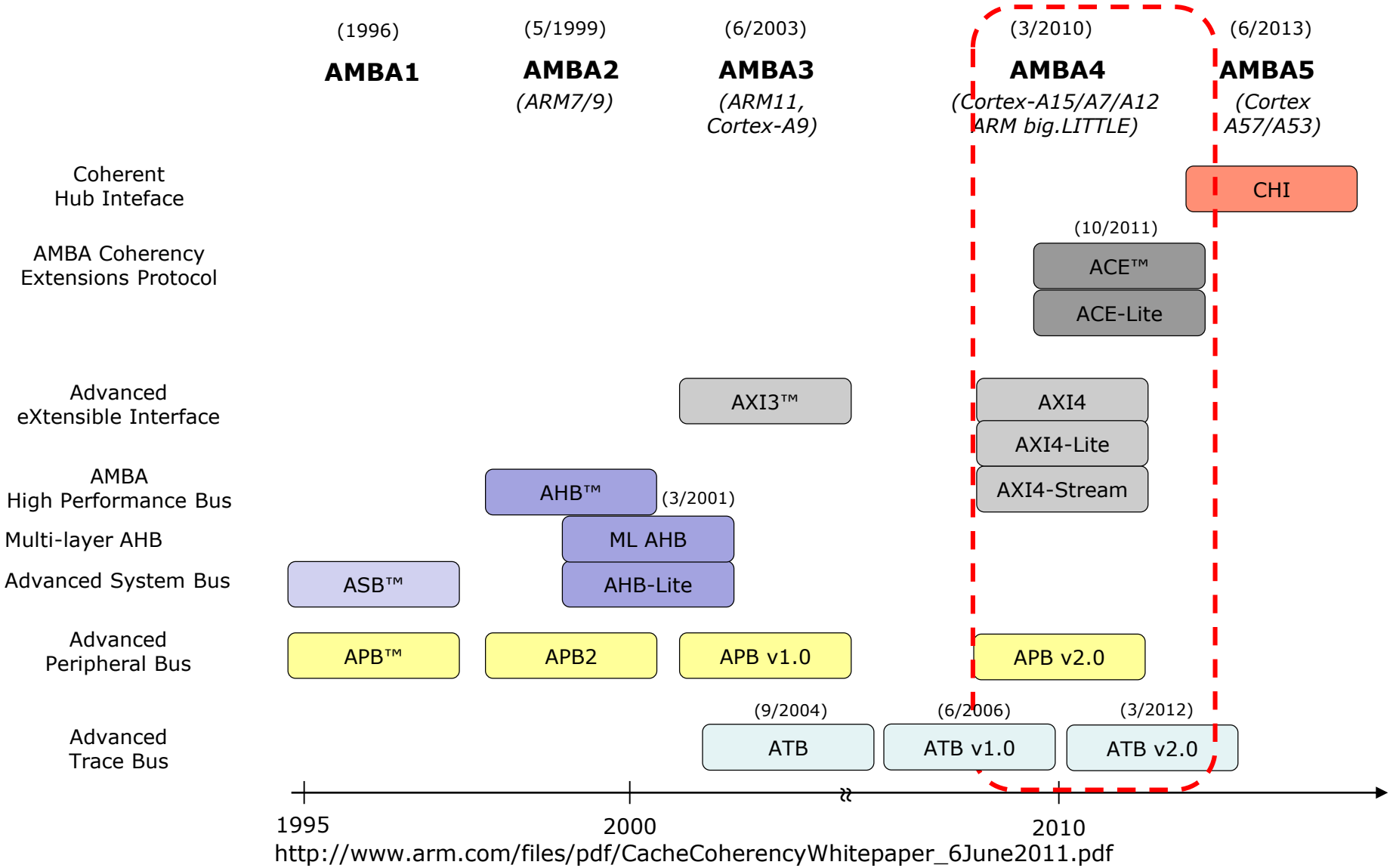
Implementing DMA on ARM SMP Systems Application Note 228 8/2009 ARM

ACP is an implementation of an AMBA 3 AXI slave interface. It supports memory coherent
accesses to the Cortex-A15 MPCore memory system, but cannot receive coherent requests,
barriers or distributed virtual memory messages.

A15 MPCore trm

5.5 The AMBA 4 protocol family

5.5 The AMBA 4 protocol family
5.5.1 Overview (based on [])



http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Components of the AMBA4 protocol family

The AMBA 4 protocol family (Revision 4.0) incorporates the following specifications:

- AXI4 (Advanced eXtensible Interface),
- AXI4-Lite,
- AXI4-Stream v1.0,
- APB v2.0 (Advanced Peripheral Bus Revision 4),
- ATB v1.1 (Advanced Trace Bus),
- and
- ACE (AXI Coherency Extensions),
- ACE-Lite,

as indicated in the previous Figure.

5.5.2 The AXI4, AXI4-Lite and AXI4-Stream interfaces

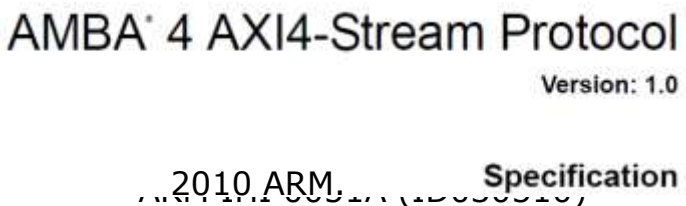
5.5.2.1 Overview

The AXI4 and AXI4-Lite interfaces were published in the first release of the AMBA AXI protocol Version 2.0 (Issue C) (2010) [a], whereas the AXI-Stream specification was issued separately (2010) [b].

[a]

ARM IHI 0022C 2010

[b]



Main features of the AXI4, AXI4-Lite and AXI4-Stream interfaces []

All three alternatives share the same principles, signal names and handshake rules, but differ in key features, as indicated below.

	AXI4	AXI4-Lite	AXI4-Stream
Dedicated for	high-performance and memory mapped systems	register-style interfaces (area efficient implementation)	non-address based IP (PCIe, Filters, etc.)
Burst (data beta)	up to 256	1	Unlimited
Data width	32 to 1024 bits	32 or 64 bits	any number of bytes
Applications (examples)	Embedded, memory	Small footprint control logic	DSP, video, communication

Table: Overview of the main features of AXI4, AXI4-Lite and AXI4-Stream

http://www.em.avnet.com/en-us/design/trainingandevents/Documents/X-Tech%202012%20Presentations/XTECH_B_AXI4_Technical_Seminar.pdf

5.5.2.2 The AXI4 interface (Advanced eXtensible Interface)

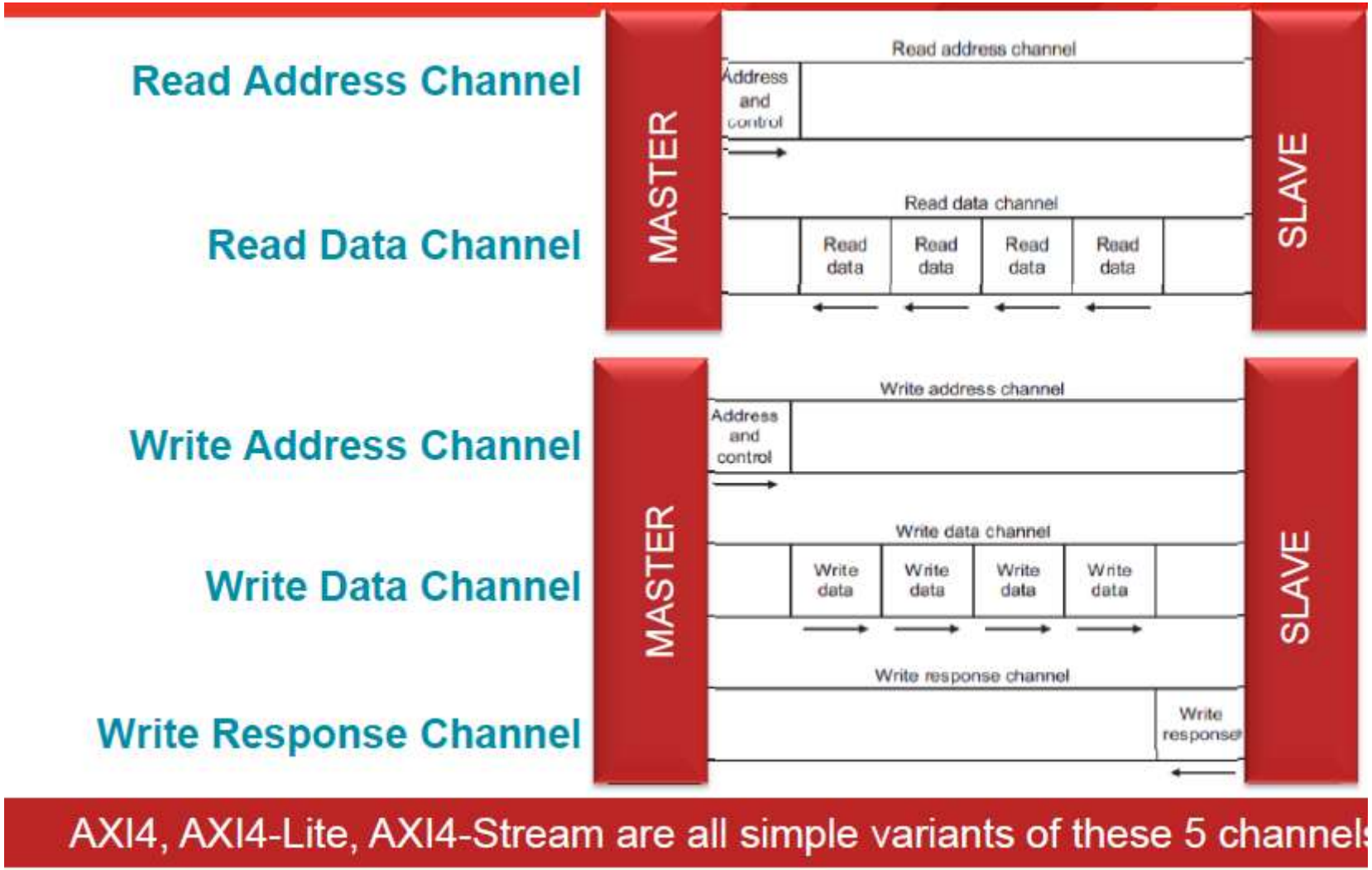
Main updates to AXI3 include []:

- support for burst lengths of up to 256 beats for incrementing bursts
- Quality of Service (QoS) signaling
- updated write response requirements
- additional information on ordering requirements
- optional user signaling
- removal of locked transactions
- removal of write interleaving.

Here we do not go into details of the updates but refer to the given reference.

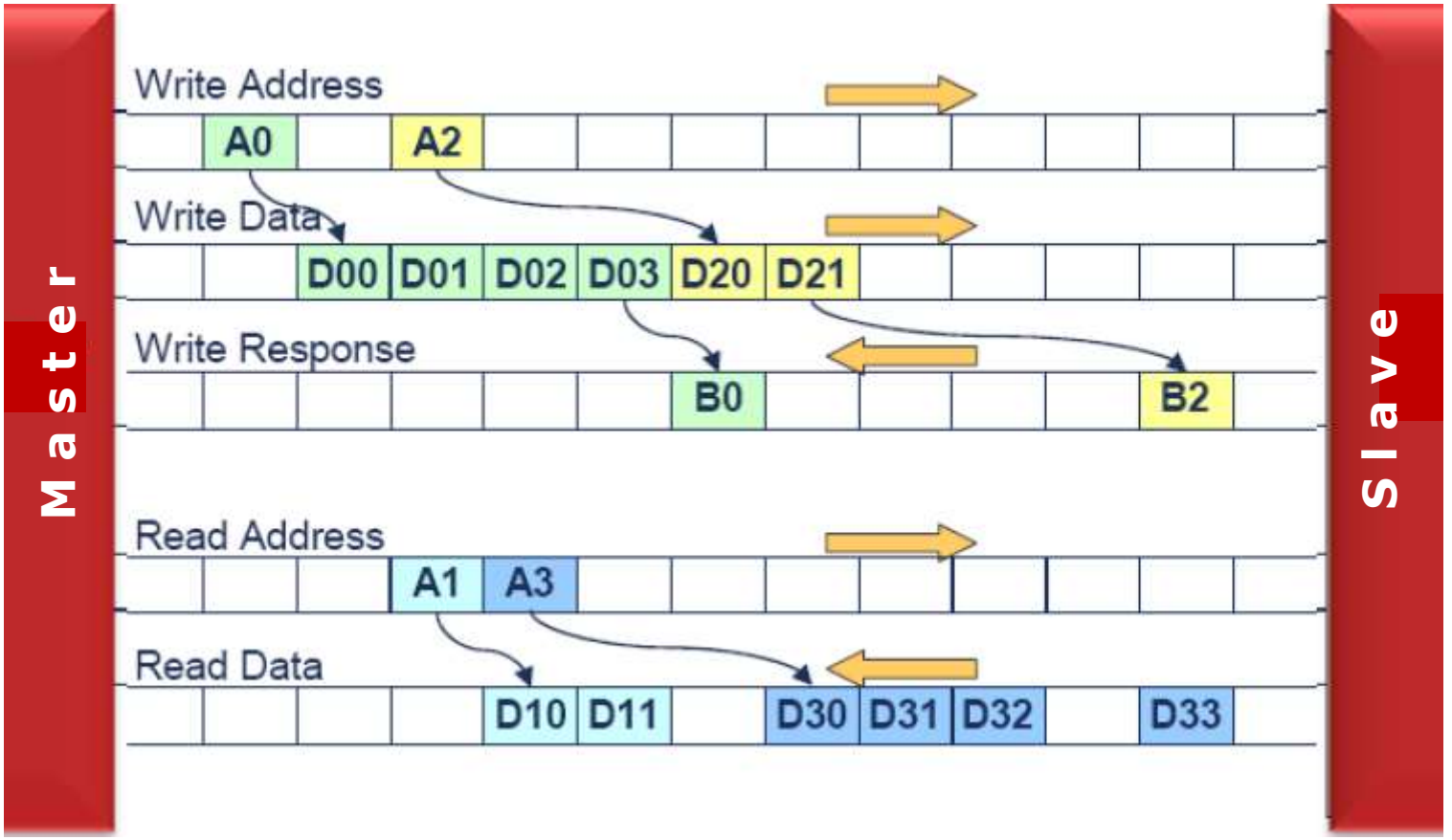
Nevertheless, we recap key features of the AXI4 interface subsequently, as a reference for pointing out main differences to AXI4-Lite and AXI4-Streams.

Basic AXI4 signaling: 5 point-to-point channels []



http://www.em.avnet.com/en-us/design/trainingandevents/Documents/X-Tech%202012%20Presentations/XTECH_B_AXI4_Technical_Seminar.pdf

Example for AXI4 transactions []



http://www.em.avnet.com/en-us/design/trainingandevents/Documents/X-Tech%202012%20Presentations/XTECH_B_AXI4_Technical_Seminar.pdf

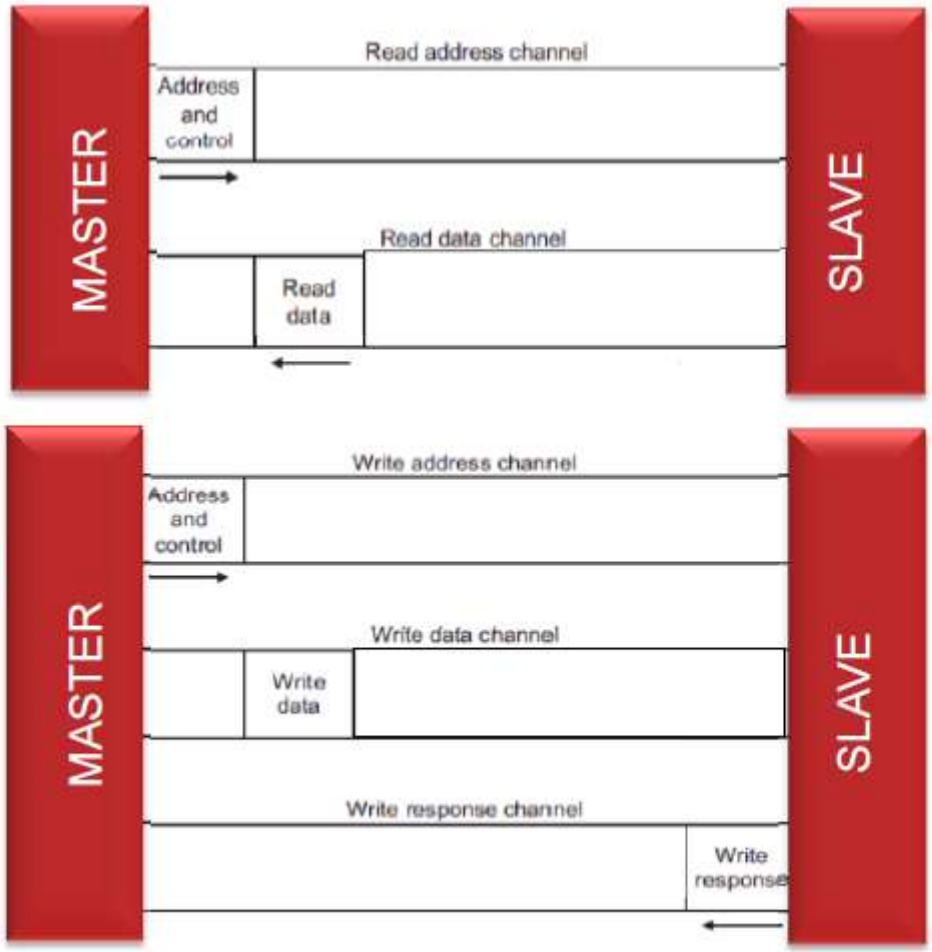
Key features of the AXI4 interface []

- **Single address multiple data**
  - Burst up to 256 data beats
- **Data Width parameterizable**
  - 32, 64, 128, 256, 512, 1024 bits

http://www.em.avnet.com/en-us/design/trainingandevents/Documents/X-Tech%202012%20Presentations/XTECH_B_AXI4_Technical_Seminar.pdf

5.5.2.3 The AXI 4-Lite interface
Key features []

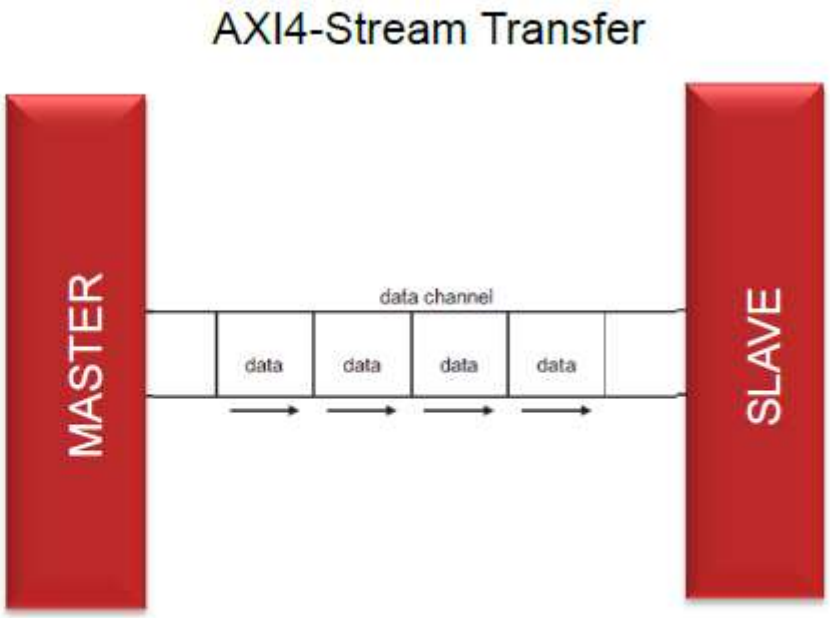
- No burst
- Data width 32
- Very small footprint
- Bridging to AXI4 handled automatically by AXI_Interconnect (if needed)



http://www.em.avnet.com/en-us/design/trainingandevents/Documents/X-Tech%202012%20Presentations/XTECH_B_AXI4_Technical_Seminar.pdf

5.5.2.4 The AXI 4-Stream interface
Key features []

- **No address channel, no read and write, always just master to slave**
  - Effectively an AXI4 “write data” channel
- **Unlimited burst length**
  - AXI4 max 256
  - AXI4-Lite does not burst
- **Virtually same signaling as AXI4 Data Channels**
  - Protocol allows merging, packing, width conversion
  - Supports sparse, continuous, aligned, unaligned streams



http://www.em.avnet.com/en-us/design/trainingandevents/Documents/X-Tech%202012%20Presentations/XTECH_B_AXI4_Technical_Seminar.pdf

5.5.3 The APB bus v2.0 (APB4 bus)

This is the second update of the APB bus, as indicated below:

Issue date	Release
9/2003	First release of APB v1.0
8/2004	Second release of APB v1.0
4/2010	First release of APB v2.0

Table: Releases of the APB bus []

Both updates include only minor differences to the previous releases, as documented in [].

We do not go here into details but refer to the cited publication.

5.5.4 The ATB bus v1.1 (ATB4 bus)

This is the first update of the original ATB bus, as indicated below:

Issue date	Release
6/2006	First release of ATB version 1.0
3/2012	Second release of ATB version 1.1

Table: Releases of the ATB bus []

The Second release of the ATB bus includes only minor differences to the original release, as documented in [].

Here, we do not go into details but refer to the cited publication.

AMBA™ 4 ATB Protocol Specification
ATBv1.0 and ATBv1.1

IHI 0032B (ID040412) ARM 2012

5.5 The ACE protocol

First defined in the First release of AMBA AXI and ACE Protocol Specification (Issue D) in 2011.
Revised in the Second release of AMBA AXI and ACE Protocol Specification (Issue E) in 2012.
ACE expands the coherency model provided by the MPCore technology for 2 to 4 cores to multiple CPU core clusters, e.g. to two CPU core clusters each with 4 cores.
ACE is not limited to coherency between identical CPU core clusters but it can support coherency also for dissimilar CPU clusters and also I/O coherency for accelerators.
The Cortex-A15 MPCore processor was the first ARM processor to support AMBA 4 ACE.

http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

All shared transactions are controlled by the ACE coherent interconnect. ARM has developed the CCI-400 Cache Coherent Interconnect to support coherency for up to two CPU clusters and three additional ACE-Lite I/O coherent masters, as indicated in the next Figure.

Cortex-A15 Inter-Cluster Coherency, big.LITTLE coherency and I/O Coherency

The Cortex-A15 MPCore processor was the first ARM processor core to support AMBA 4 ACE. ACE enables expansion of the SoC beyond the 4-core cluster of the ARM MPCore technology. Accesses to shared memory are broadcast on the ACE interface enabling coherency between multiple Cortex-A15 clusters, enabling scaling to 8 cores and beyond. But ACE is not limited to coherency between identical CPU clusters, it can also support full coherency between dissimilar CPUs and I/O coherency for accelerators. For example it supports coherency between a Cortex-A15 cluster and a Cortex-A7 cluster (which also supports the ACE coherency protocol) in a big.LITTLE system.

All shared transactions are controlled by the ACE coherent interconnect. ARM has developed the CCI-400 Cache Coherent Interconnect product to support up to two clusters of CPUs and three additional ACE-Lite I/O coherent masters. In this example diagram below we show the Mali-T604 GPU, which is

http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Key features of the ACE protocol []

The ACE protocol provides a framework for maintaining system level coherency while leaving the freedom for system designers to determine

- the ranges of memory that are coherent,
- the memory system components for implementing the coherency extensions and also
- the software models used for the communication between system components.

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022d/index.html

Implementation of the ACE protocol

The ACE protocol is implemented by extending the non-coherent AXI interface by three snoop channels, as indicated in the next Figure.

About the ACE protocol

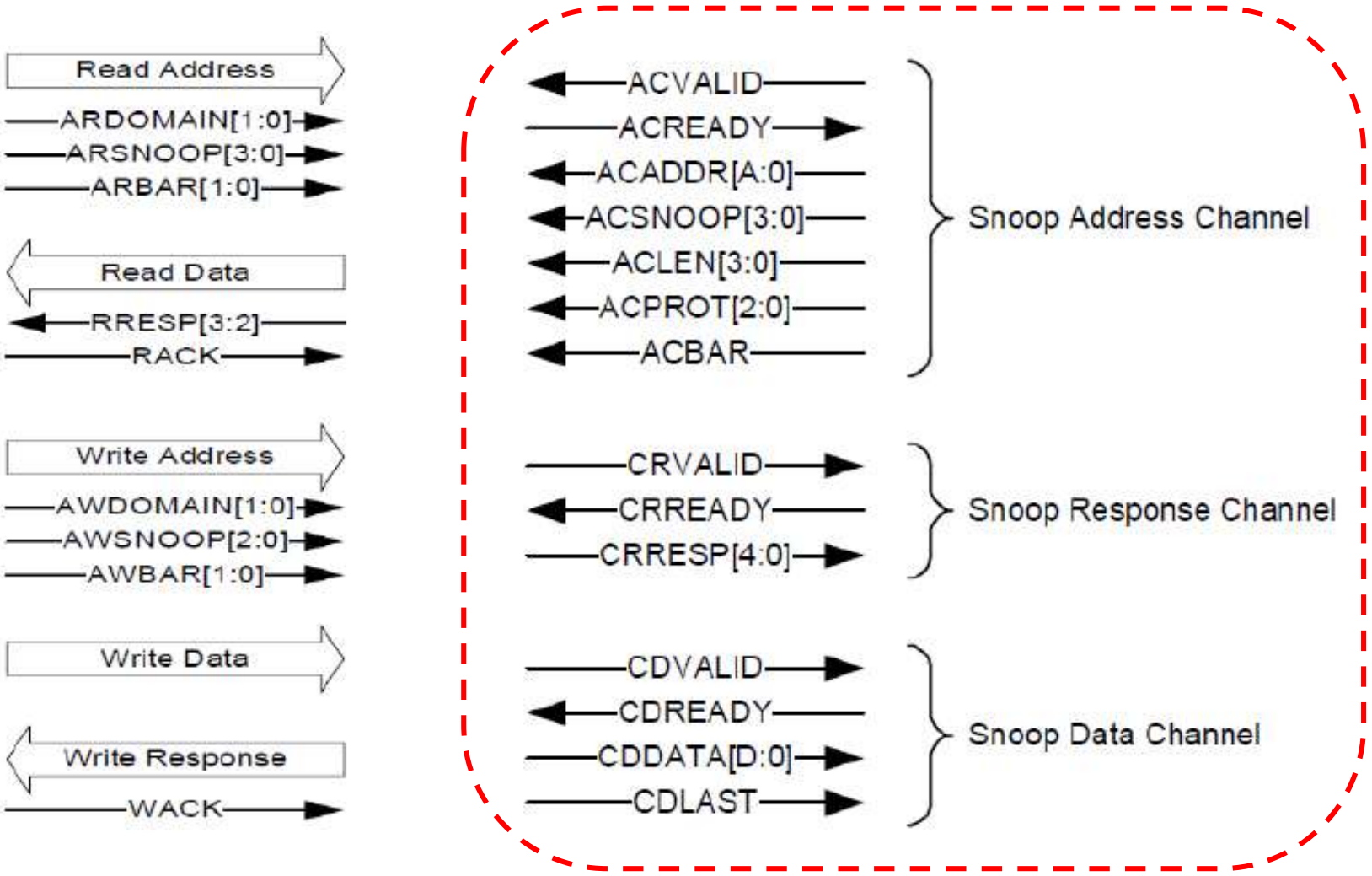
The ACE protocol extends the AXI4 protocol and provides support for hardware-coherent caches. The ACE
protocol is realized using:

- A five state cache model to define the state of any cache line in the coherent system. The cache line state
determines what actions are required during access to that cache line.
- Additional signaling on the existing AXI4 channels that enables new transactions and information to be
conveyed to locations that require hardware coherency support.
- Additional channels that enable communication with a cached master when another master is accessing an
address location that might be shared.

The ACE protocol also provides:

- Barrier transactions that guarantee transaction ordering within a system, see *Barriers* on page C1-148
- *Distributed Virtual Memory* (DVM) functionality to manage virtual memory, see *Distributed Virtual Memory*
on page C1-149.

Extending the AXI interface by three snoop channels in the ACE interface []



http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

Additional channels defined by ACE

Three new channels are supported, these are the snoop address channel, the snoop data channel, and the snoop response channel.

The snoop address (AC) channel is an input to a cached master that provides the address and associated control information for snoop transactions.

The snoop response (CR) channel is an output channel from a cached master that provides a response to a snoop transaction. Every snoop transaction has a single response associated with it. The snoop response indicates if an associated data transfer on the CD channel is expected.

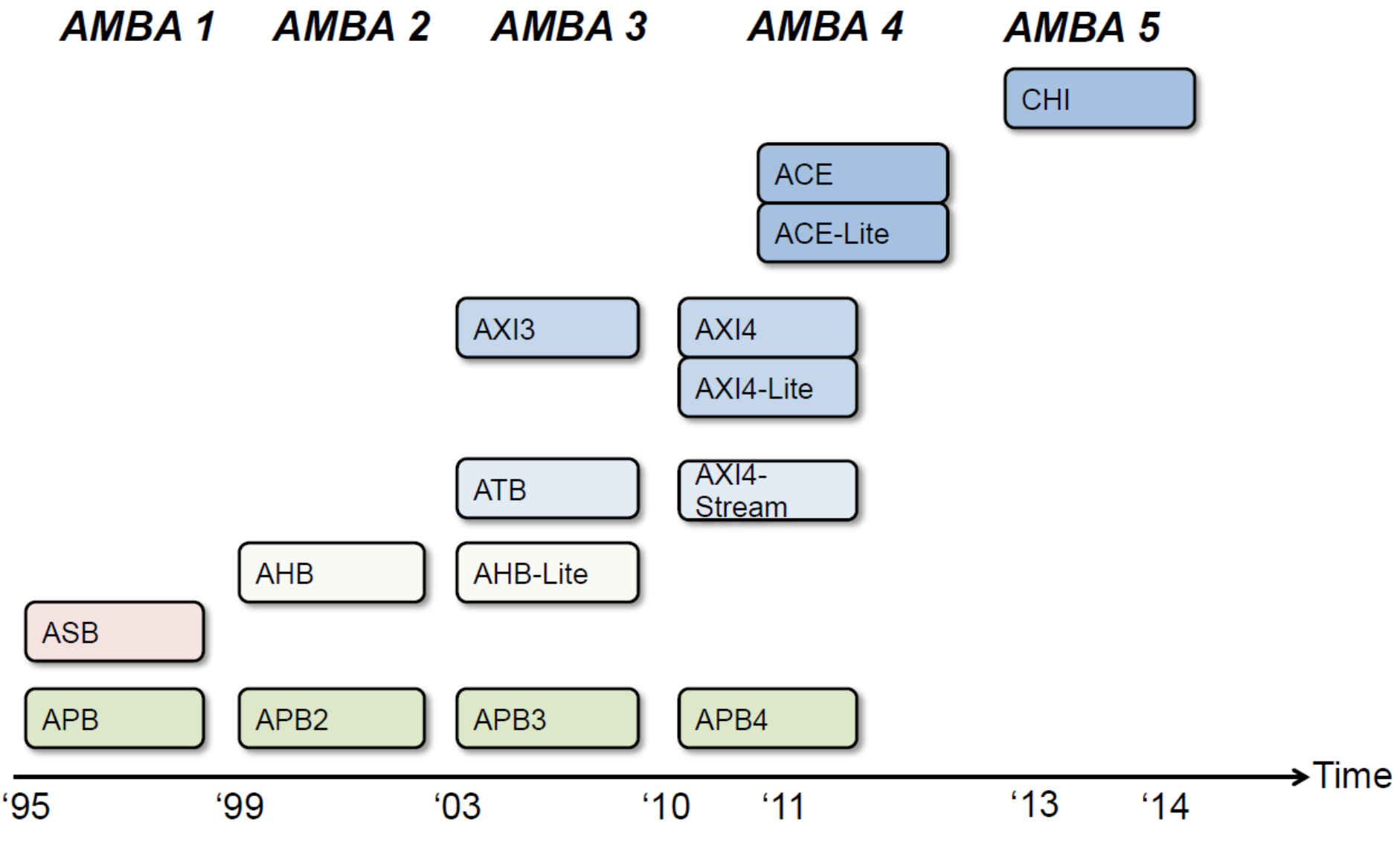
The snoop data (CD) channel is an optional output channel that passes snoop data out from a master. Typically, this occurs for a read or clean snoop transaction when the master being snooped has a copy of the data available to return.

About snoop filtering

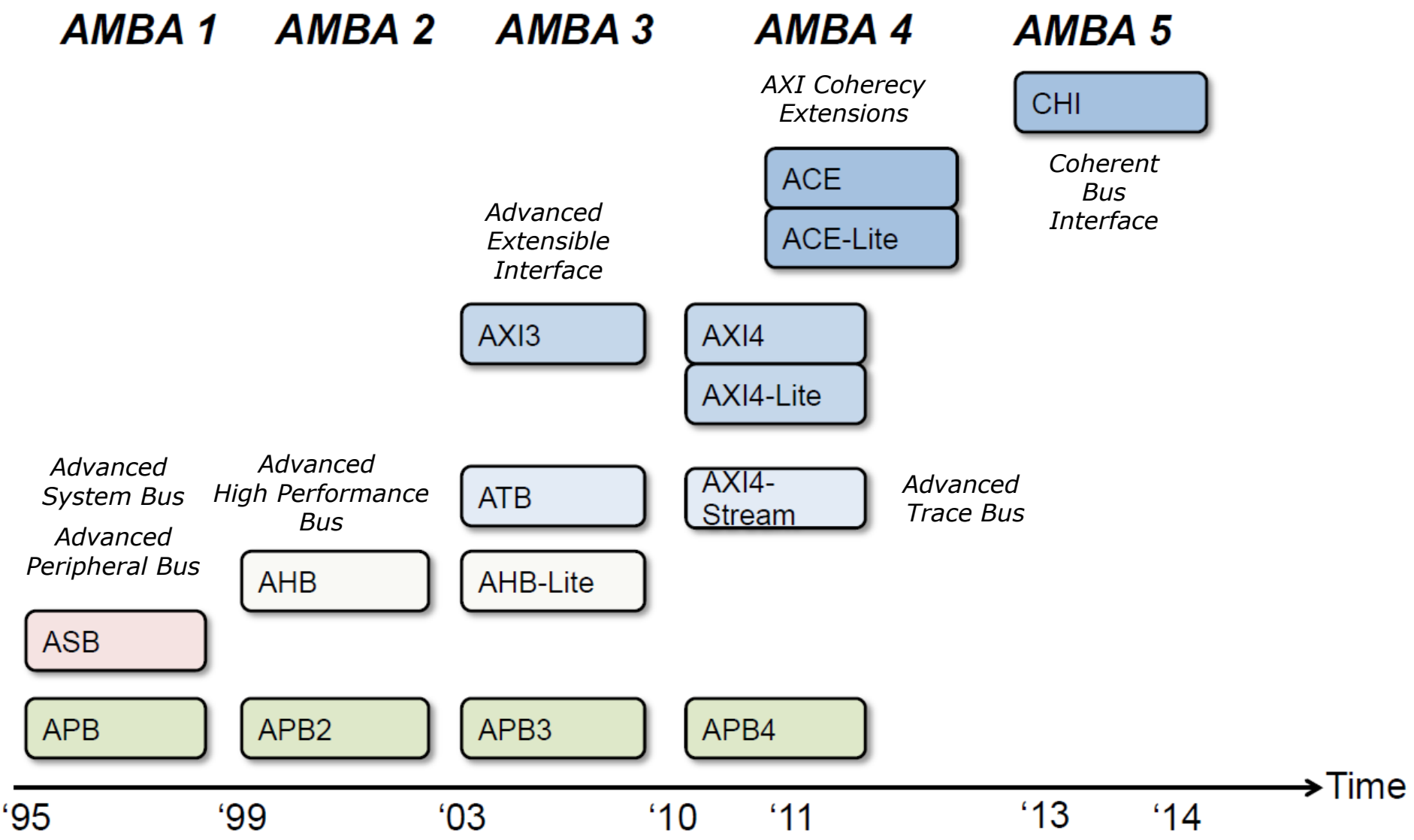
Snoop filtering tracks the cache lines that are allocated in a master’s cache. To support an external snoop filter, a cached master must be able to broadcast which cache lines are allocated and which are evicted.
Support for an external snoop filter is optional within the ACE protocol. A master component must state in its data sheet if it provides support. See Chapter C10 *Optional External Snoop Filtering* for the mechanism the ACE protocol supports for the construction of an external snoop filter.
For a master component that does not support an external snoop filter, the cache line states permitted after a transaction has completed are less strict.

2.3.3 Introducing the AMBA 4 ACE interface

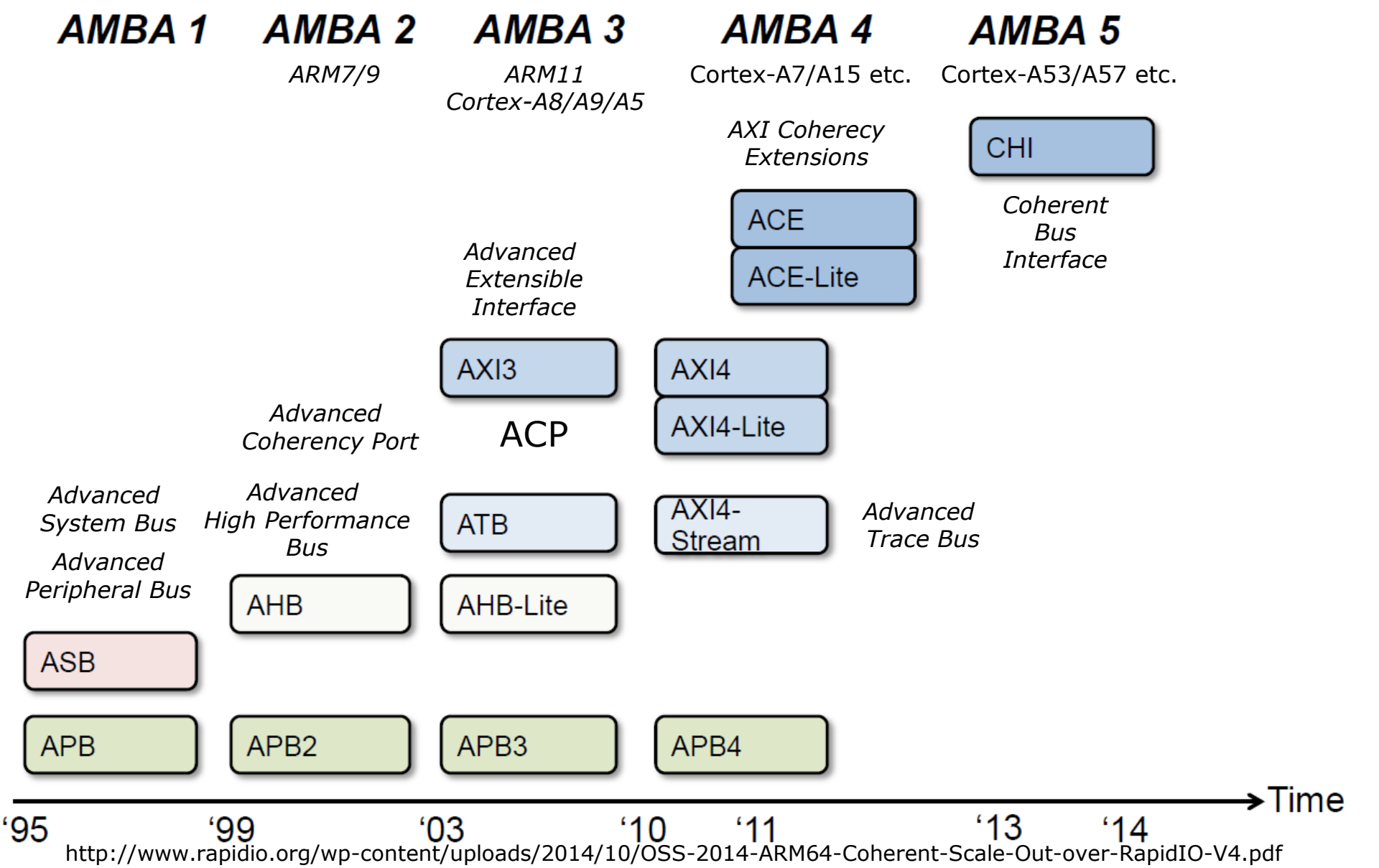
RapidIO Evolution of AMBA specifications



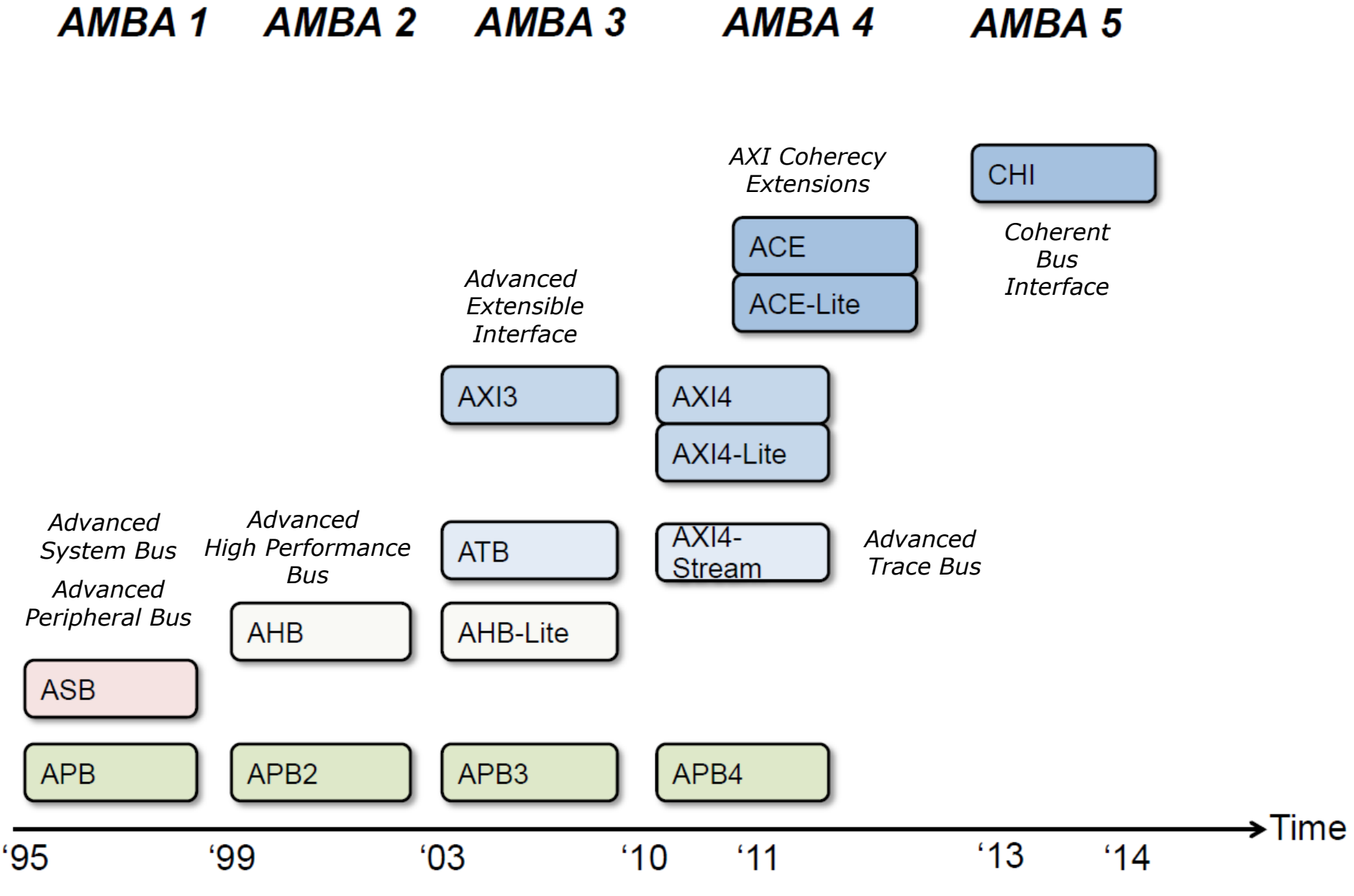
RapidIO Evolution of AMBA specifications



Evolution of the AMBA specifications (based on [])



http://www.rapidio.org/wp-content/uploads/2014/10/OSS-2014-ARM64-Coherent-Scale-Out-over-RapidIO-V4.pdf


AMBA 1
AMBA 2
AMBA 3
AMBA 4
AMBA 5
AXI Coherecy Extensions
CHI
ACE
ACE-Lite
Coherent Bus Interface
Advanced Extensible Interface
AXI3
AXI4
AXI4-Lite
Advanced System Bus
Advanced High Performance Bus
ATB
AXI4-Stream
Advanced Trace Bus
Advanced Peripheral Bus
AHB
AHB-Lite
ASB
APB
APB2
APB3
APB4
Time
'95
'99
'03
'10
'11
'13
'14

AMBA 1 **AMBA 2** **AMBA 3** **AMBA 4** **AMBA 5**

AXI Coherecy Extensions

Coherent Bus Interface

Advanced Extensible Interface

Advanced System Bus

Advanced Peripheral Bus

Advanced High Performance Bus

Advanced Trace Bus

The ACP (accelerator Coherency Port) in A5 MPCore and A9 MPCore
will be replaced by ACE in A7 MPCcore
and subsequent proc.s

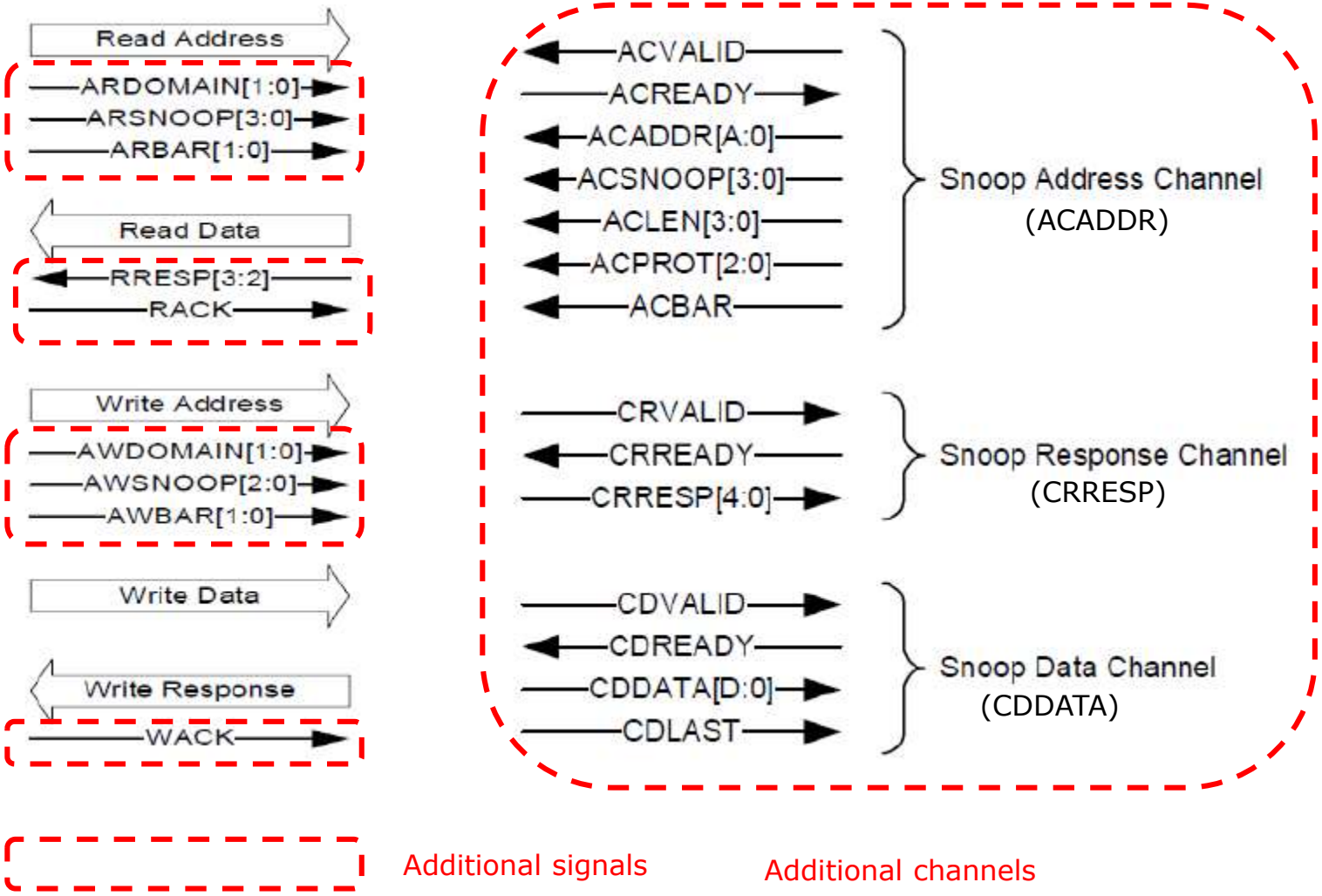
A9 MPCore: An optional *Accelerator Coherency Port* (ACP) suitable for coherent memory transfers
A5 MPCore: an *Acceleration Coherency Port* (ACP), an optional AXI 64-bit slave port that can be connected to a noncached peripheral such as a DMA engine.

2.3.3 Introducing the AMBA 4 ACE interface

ARM extended AMBA 3 AXI to AMBA 4 ACE (AMBA with Coherency Extension) by 3 further channels and a number of additional signals

in order to implement system wide coherency, as the next Figure indicates.

Extension of the AMBA 3 (AXI) interface with snoop channels and additional signals to have the AMBA 4 (ACE) interface []



http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

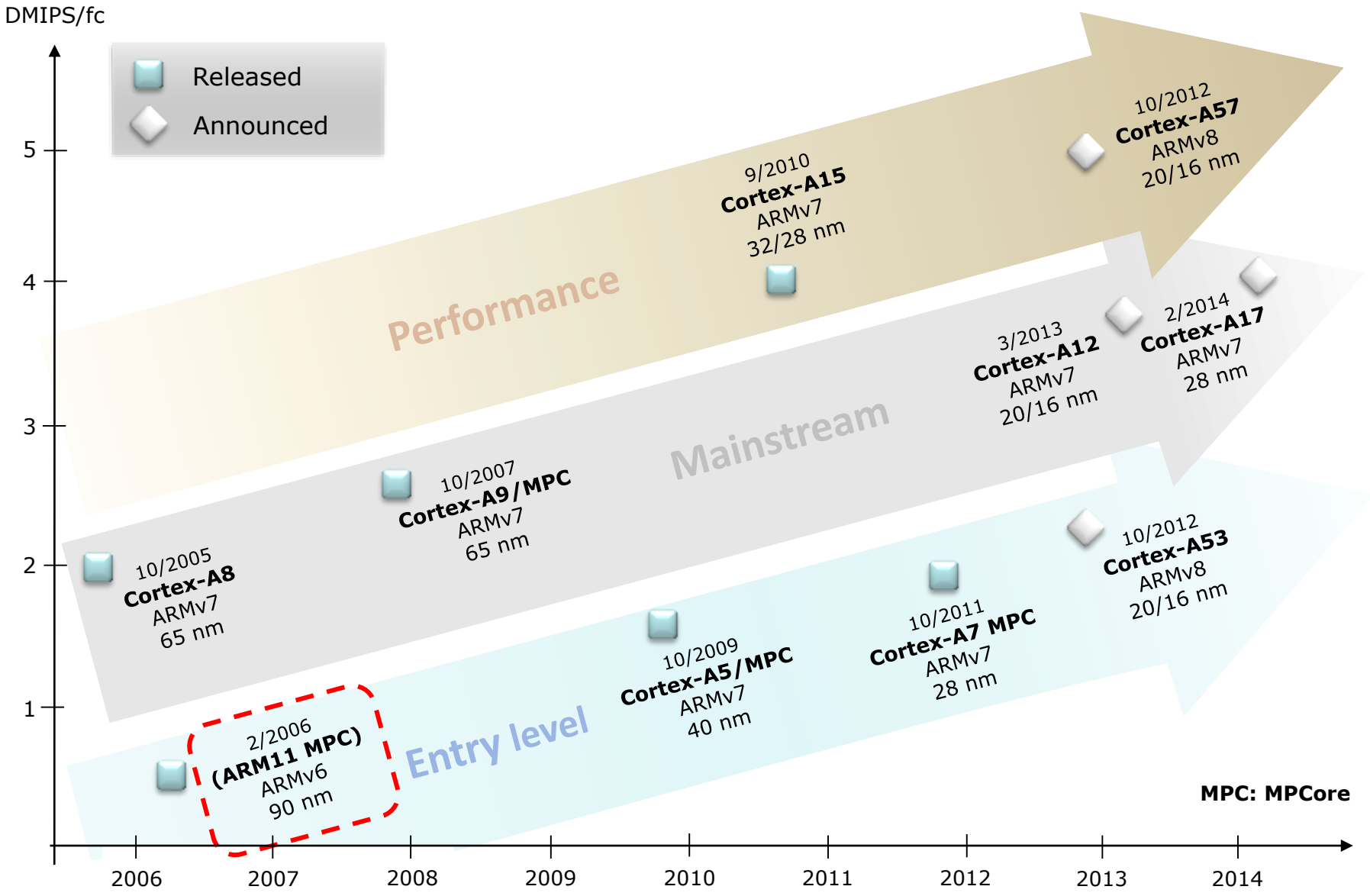
Use of the additional snoop channels []

- The ACADDR channel is a snoop address input to the master.
- The CRRESP channel is used by the master to signal the response to snoops to the interconnect.
- The CDDATA channel is output from the master to transfer snoop data to the originating master and/or external memory.

http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

2.2.2 ARM’s 1. generation cache coherency management

2.2.2 ARM's 1. generation cache coherency management (based on [])



Note

Cache coherency, as discussed subsequently for multicore processors relates to the coherency between the L1 data caches and the higher levels of the memory system.

By contrast, instruction caches are usually read only caches so they will be managed by a much simpler mechanism (read invalidate) to maintain cache coherency.

ARM implemented their 1. generation cache coherency management scheme originally in their first multicore processor, the ARM11 MPCore.

Introduction to ARM's 1. generation cache coherency management

ARM revealed their concept for managing cache coherency for multicore processor (termed by ARM as multiprocessors) in two patents [a], [b] and a conference talk [] in 7/2003.

AMD's concept was based on a centralized cache coherency management unit, called the Snoop Control Unit (SCU), as shown in the Figure on the right side.

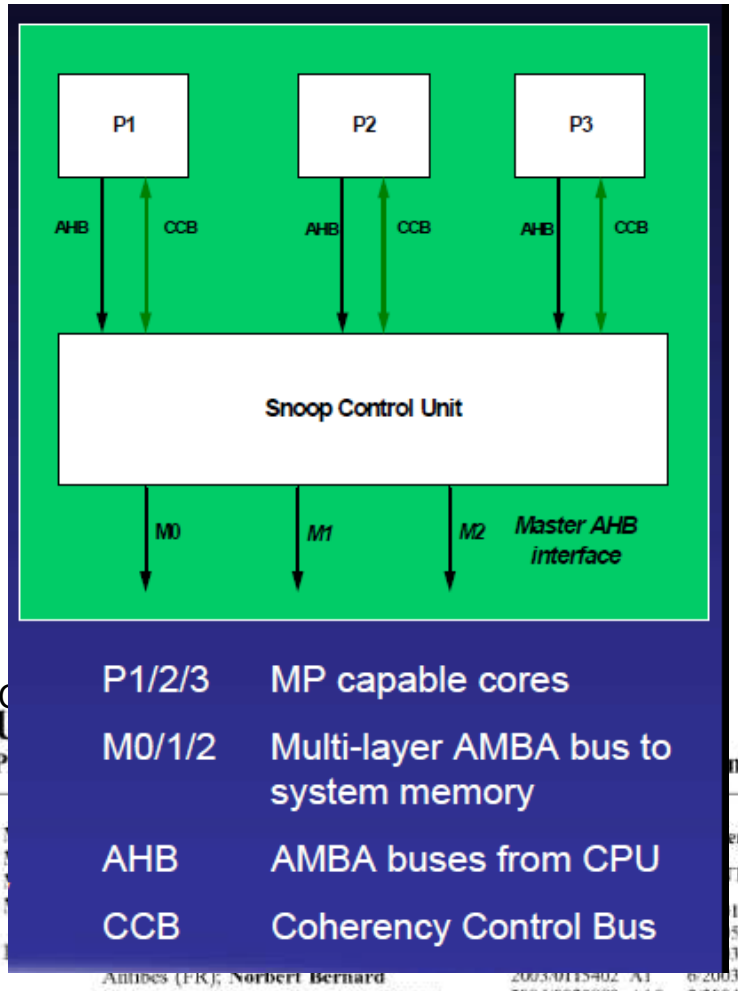
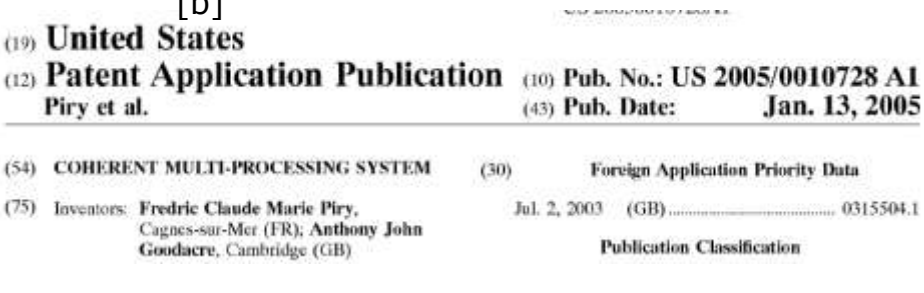


Figure: ARM's concept for managing cache coherence in a multicore system [c]

[c] http://www.mpsoc-forum.org/previous/2003/slides/MPSoC

[b]


(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2005/0010728 A1**

Piry et al. (43) **Pub. Date: Jan. 13, 2005**

(54) **COHERENT MULTI-PROCESSING SYSTEM**

(75) Inventors: **Fredric Claude Marie Piry**, Cagnes-sur-Mer (FR); **Anthony John Goodacre**, Cambridge (GB)

(30) **Foreign Application Priority Data**

Jul. 2, 2003 (GB) 0315504.1

Publication Classification

(12)

(54)

(75)

Antibes (FR); **Norbert Bernard**

Eugene Lataille, Antibes (FR); **Stuart David Biles**, Cambridge (GB)

US 7,

nt:

ences Cited

T DOCUMEN

1 Koenen

5 Gaither et al.

3 Akrishnan et

2003/0115402 A1 6/2003 Dahlgren et a

2004/0039880 A1* 2/2004 Pentkovski et

2004/0068595 A1* 4/2004 Dieffenderfer

2004/0073623 A1* 4/2004 Benkual et al

Principle of ARM's 1. generation cache coherency management []

- To achieve cache coherency AMD developed a specific scheme unlike the usual snooping or directory based approaches.
- In this scheme the cores send read/write requests to a central coherency control unit (the SCU) via the AHB bus and augment these requests with relevant cache state information sent over a dedicated bus, called the CCB Bus (Coherency Control Bus), as shown below.

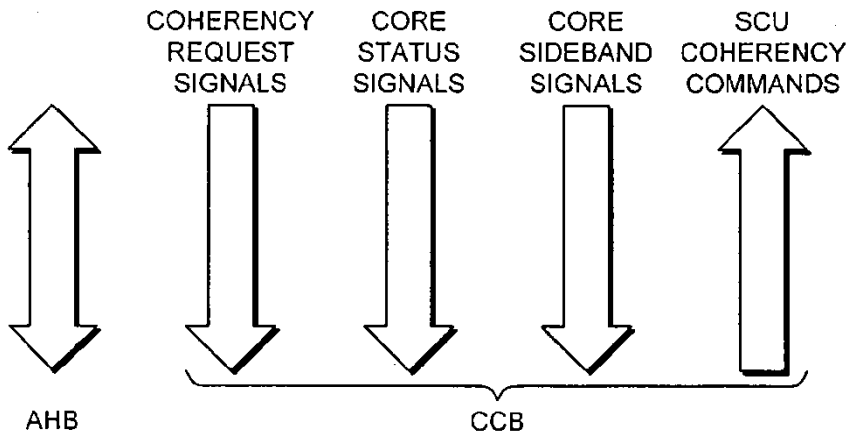


Figure: Signal groups of the CCB bus []

Note that in usual implementations the coherency control unit observes the read/write requests of the cores (and external I/O channels) and if needed sends snoop requests to the cache controllers to be informed about the state of the referenced cache line.

[]: előző oldal [b]

(cont.)

- The additional information sent over the CCB bus to the SCU specify e.g. whether or not data requested are held in the caches, what the status of the referenced cache line is, etc.
- Based on the cache coherency model chosen and by taking into account the additional information delivered by the CCB signals, the SCU decides on the required actions needed to maintain cache coherency for read and write requests of the cores and sends the appropriate coherency commands to the cache controllers via the CCB bus.

Here we note that both the patent description and its first implementation in the ARM11 MPCore make use of the MESI protocol.

Remark

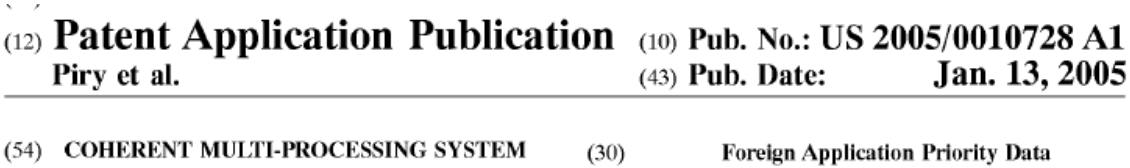
To outline the signals carried over the CCB bus, subsequently we cite an excerpt from the patent description [], with minor modifications to increase readability.

"Coherency request signals are characterizing the nature of a memory access being requested such that the coherency implications associated with that memory access request can be handled by the snoop control unit.

As an example, line fill read requests for the cache memory associated with a coherent multi-processing core may be augmented to indicate whether they are a simple line fill request or a line fill and invalidate request whereby the snoop control unit should invalidate other copies of the data value concerned which are held elsewhere.

In a similar way, different types of Write request may be distinguished between by the coherency request signals on the CCB in a manner which can then be acted upon by the snoop control unit.

The core status signals pass coherency related information from the core to the snoop control unit such as, for example, signals indicating whether or not a particular core is operating in a coherent multi-processing mode, is ready to receive a coherency command from the snoop control unit, and does or does not have a data value which is being requested from it by the snoop control unit."



(12) **Patent Application Publication** (10) **Pub. No.: US 2005/0010728 A1**
Piry et al. (43) **Pub. Date: Jan. 13, 2005**

(54) **COHERENT MULTI-PROCESSING SYSTEM** (30) **Foreign Application Priority Data**

The core sideband signals passed from the core to the snoop control unit via the CCB include signals indicating that the data being sent by the core is current valid data and can be sampled, that the data being sent is "dirty" and needs to be written back to its main stored location, and elsewhere as appropriate, that the data concerned is within an eviction Write buffer and is no longer present within the cache memory of the core concerned, and other signals as may be required.

The snoop control unit coherency commands passed from the snoop control unit to the processor core include command specifying operations relating to coherency management which are required to be performed by the processor core under instruction of the snoop control unit. As an example, a forced change in the status value associated with a data value being held within a cache memory of a processor core may be instructed such as to change that status from modified or exclusive status to invalid or shared in accordance with the applied coherency protocol.

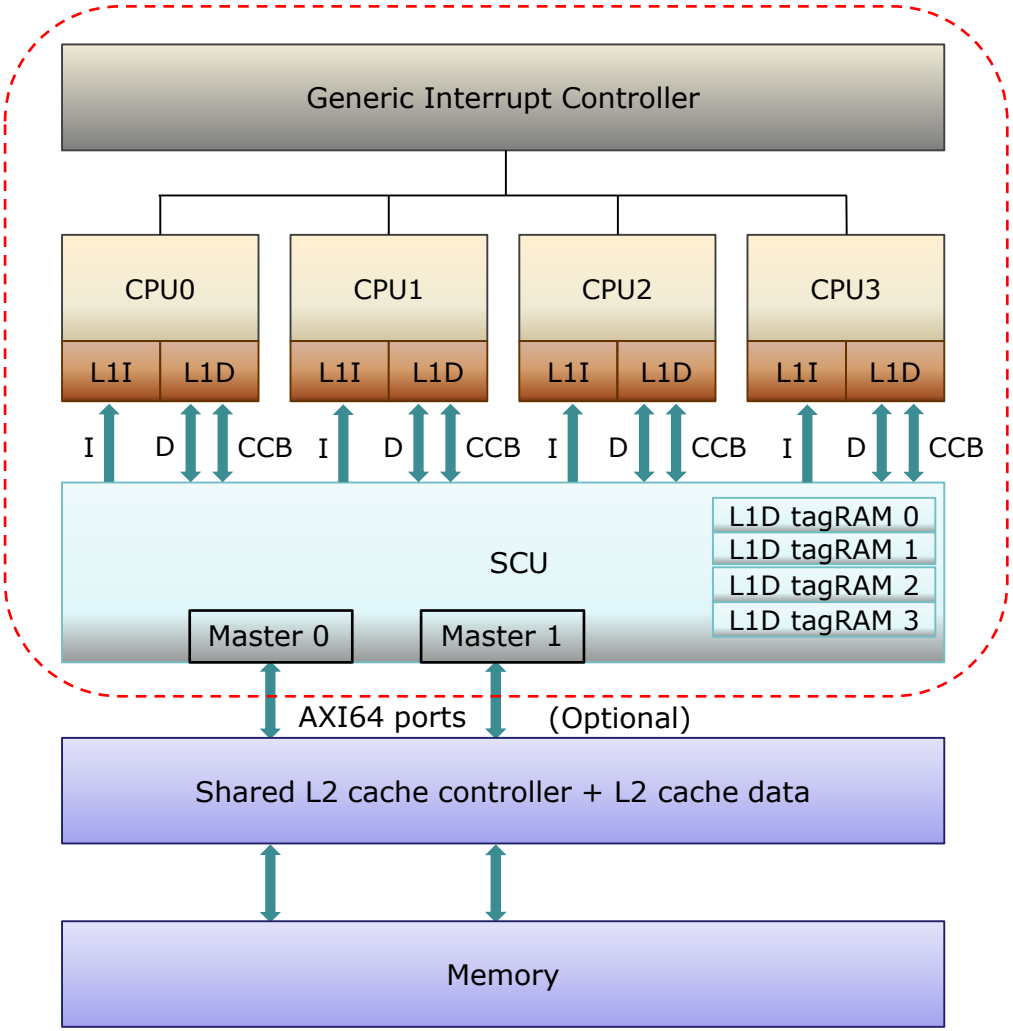
Other commands may instruct the processor core to provide a copy of a current data value to the snoop control unit such that this may be forwarded to another processor core to service a memory read request, from that processor core. Other commands include, for example, a clean command."

Implementation of ARM's 1. gen. cache coherency management concept in the ARM11 MPCore processor

The cache management technique described in the patents [], [] was implemented first in the ARM11 MPCore processor (2005) that may include up to quad cores.

Block diagram and key features of the implemented coherency management technique are shown in the next Figure.

Block diagram of the ARM11 MPCore processor-1

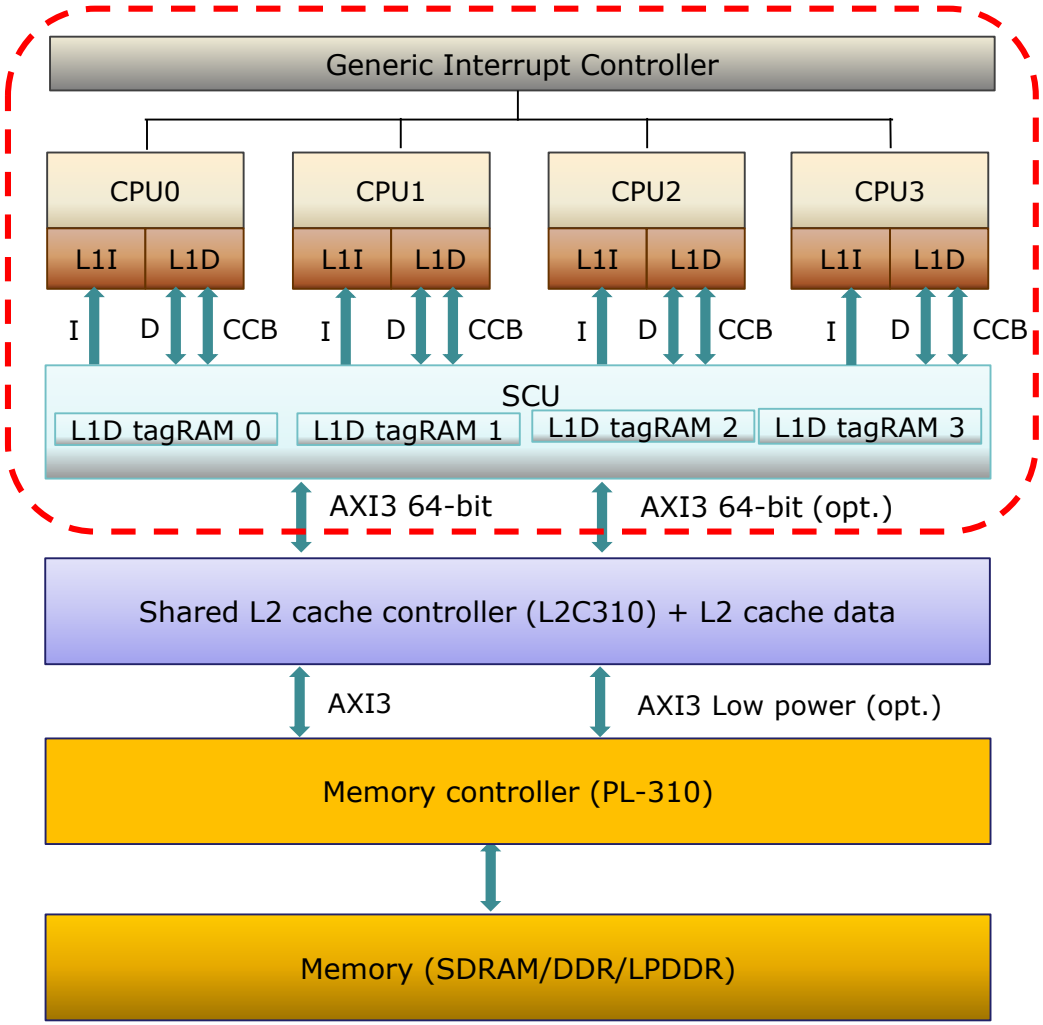


ARM11 MPcore

CCB: Coherency Control Bus

- Enhanced MESI
- SCU holds copies of each L1D directory to reduce snoop traffic between L1D caches and the L2
- Direct cache to cache transfers supported

Block diagram of the ARM11 MPCore processor-1

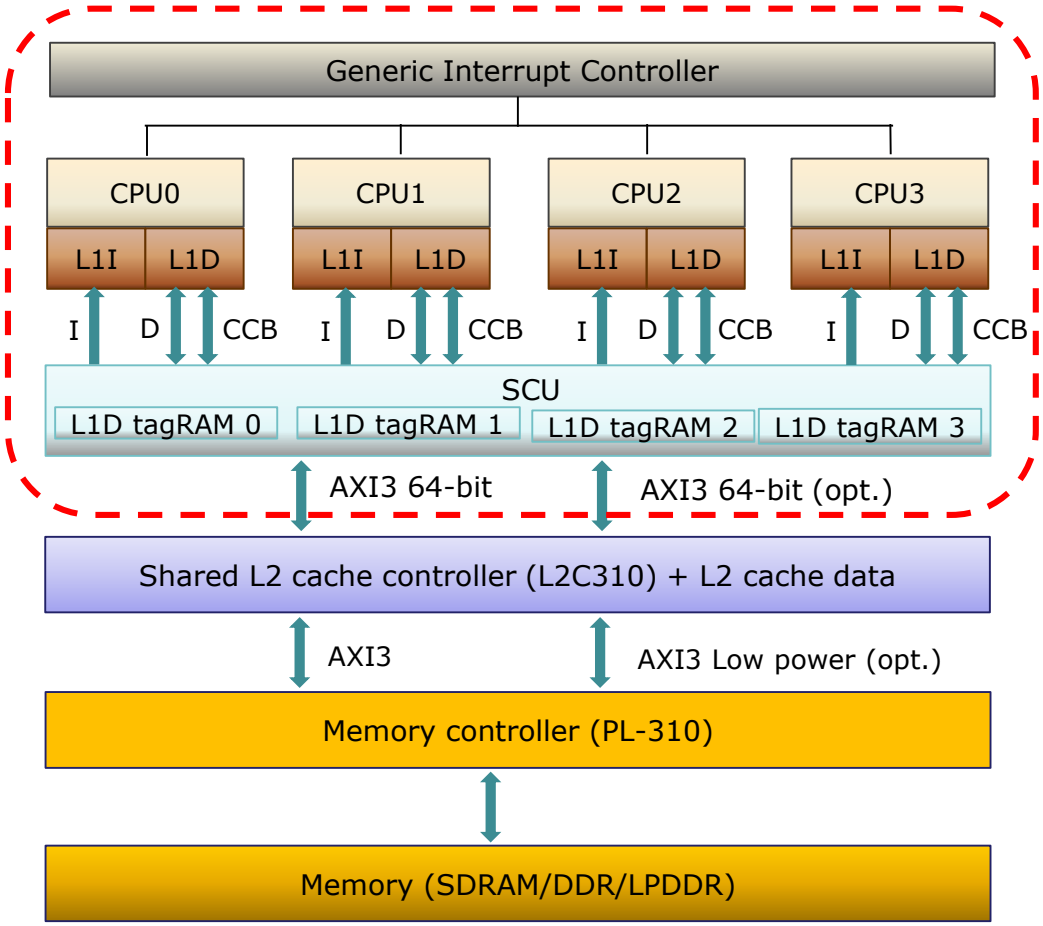


ARM11 MPcore

CCB: Coherency Control Bus

- Enhanced MESI
- SCU holds copies of each L1D directory to reduce snoop traffic between L1D caches and the L2
- Direct cache to cache transfers supported

Block diagram of the ARM11 MPCore processor-1

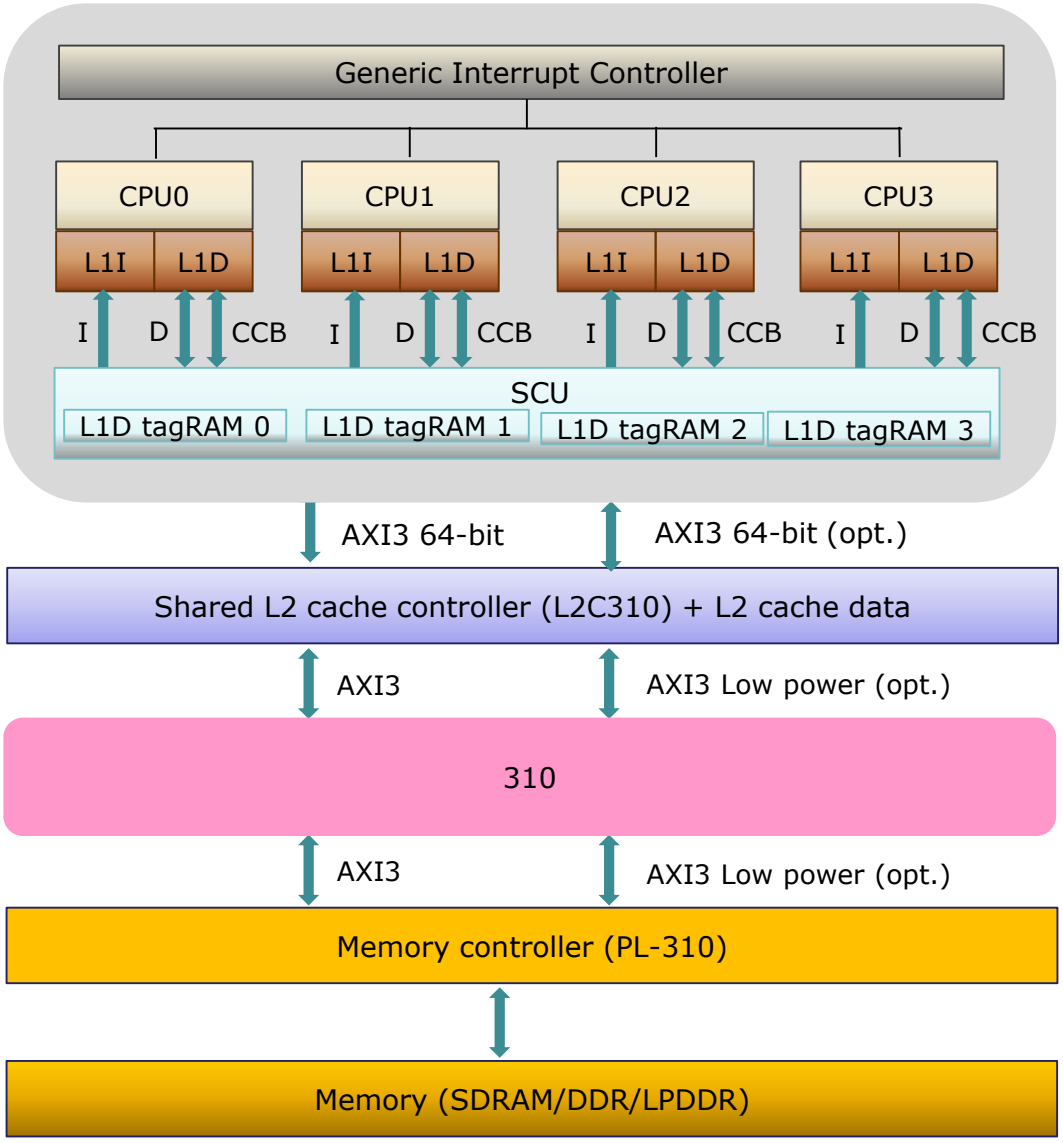


ARM11 MPcore

CCB: Coherency Control Bus

- Enhanced MESI
- SCU holds copies of each L1D directory to reduce snoop traffic between L1D caches and the L2
- Direct cache to cache transfers supported

Block diagram of the Cortex-A9 MPCore processor-1

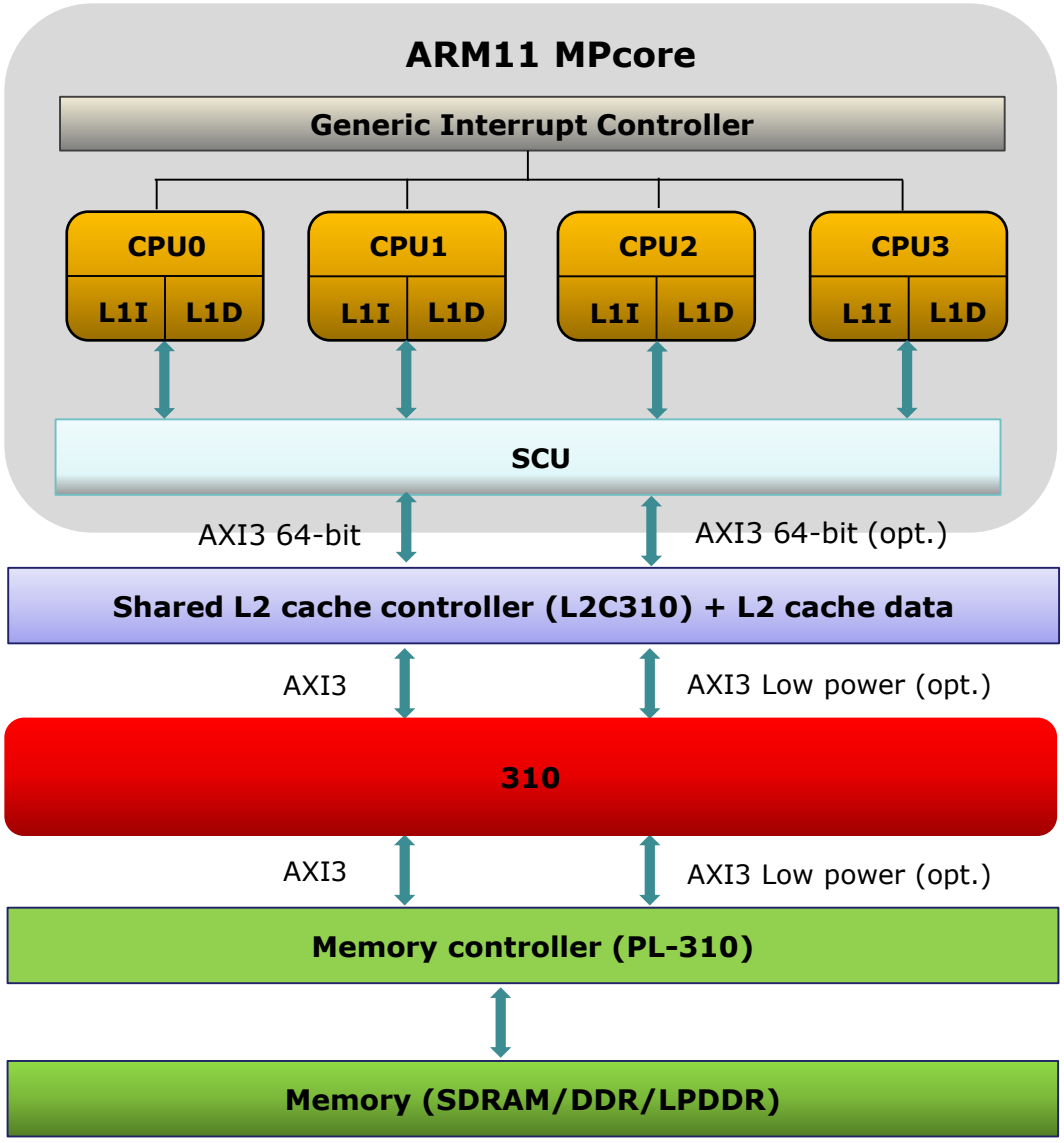


ARM11 MPcore

CCB: Coherency Control Bus

- Enhanced MESI
- SCU holds copies of each L1D directory to reduce snoop traffic between L1D caches and the L2
- Direct cache to cache transfers supported

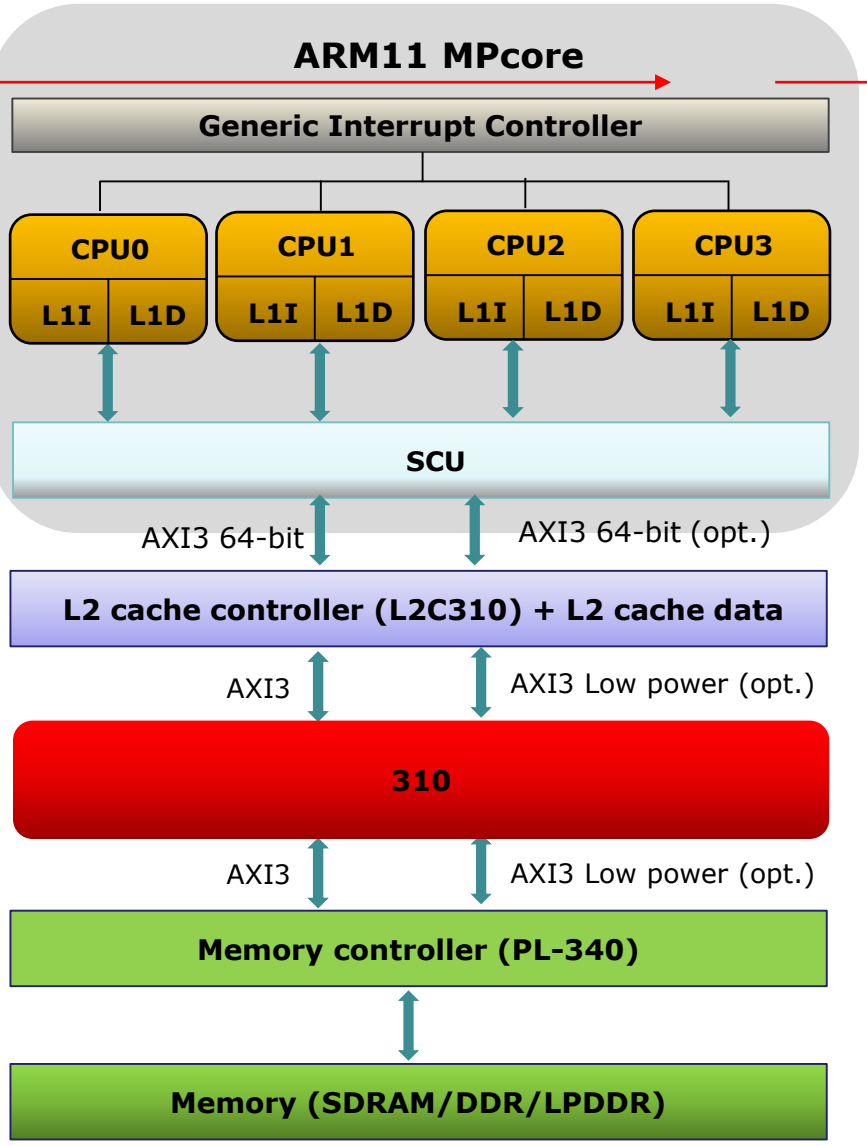
Block diagram of the Cortex-A9 MPCore processor-1



CCB: Coherency Control Bus

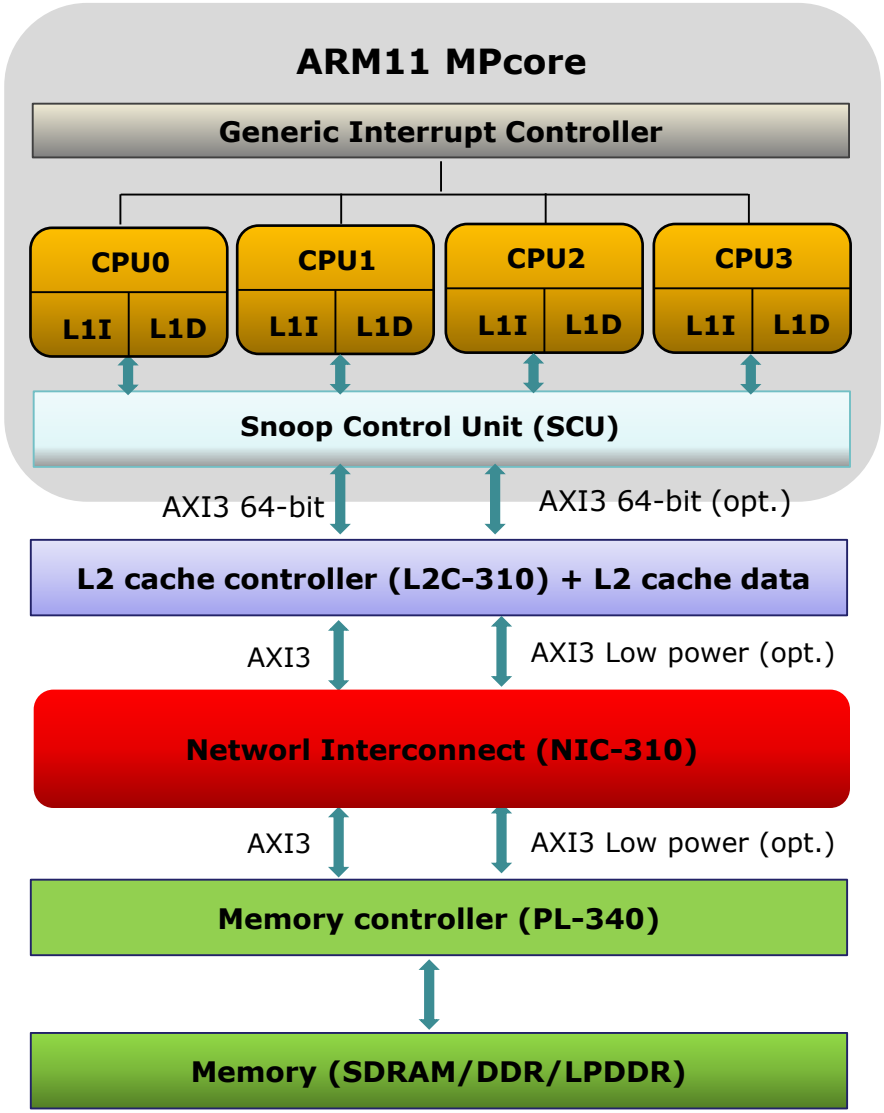
- Enhanced MESI
- SCU holds copies of each L1D directory to reduce snoop traffic between L1D caches and the L2
- Direct cache to cache transfers supported

Block diagram of the Cortex-A9 MPCore processor-1

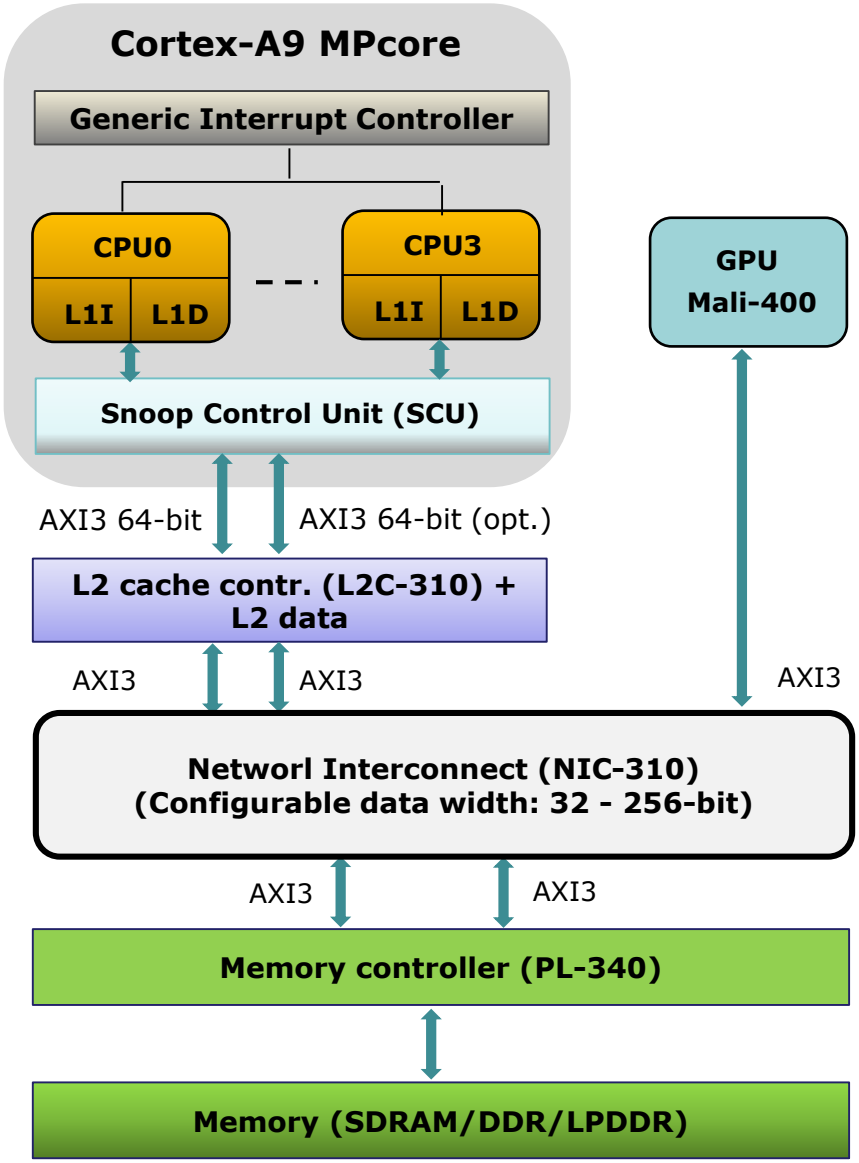


CCB: Coherency Control Bus

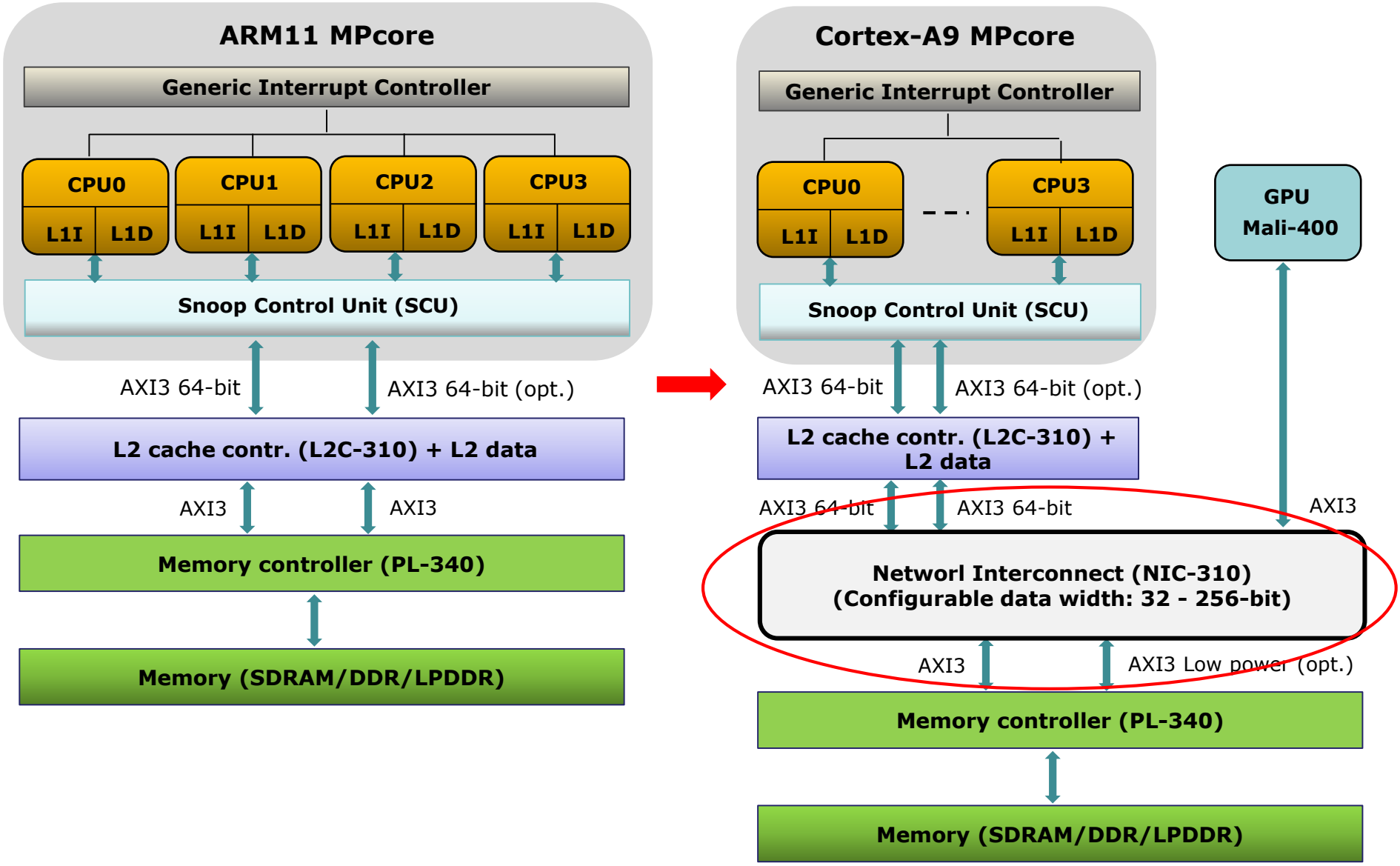
- Enhanced MESI
- SCU holds copies of each L1D directory to reduce snoop traffic between L1D caches and the L2
- Direct cache to cache transfers supported

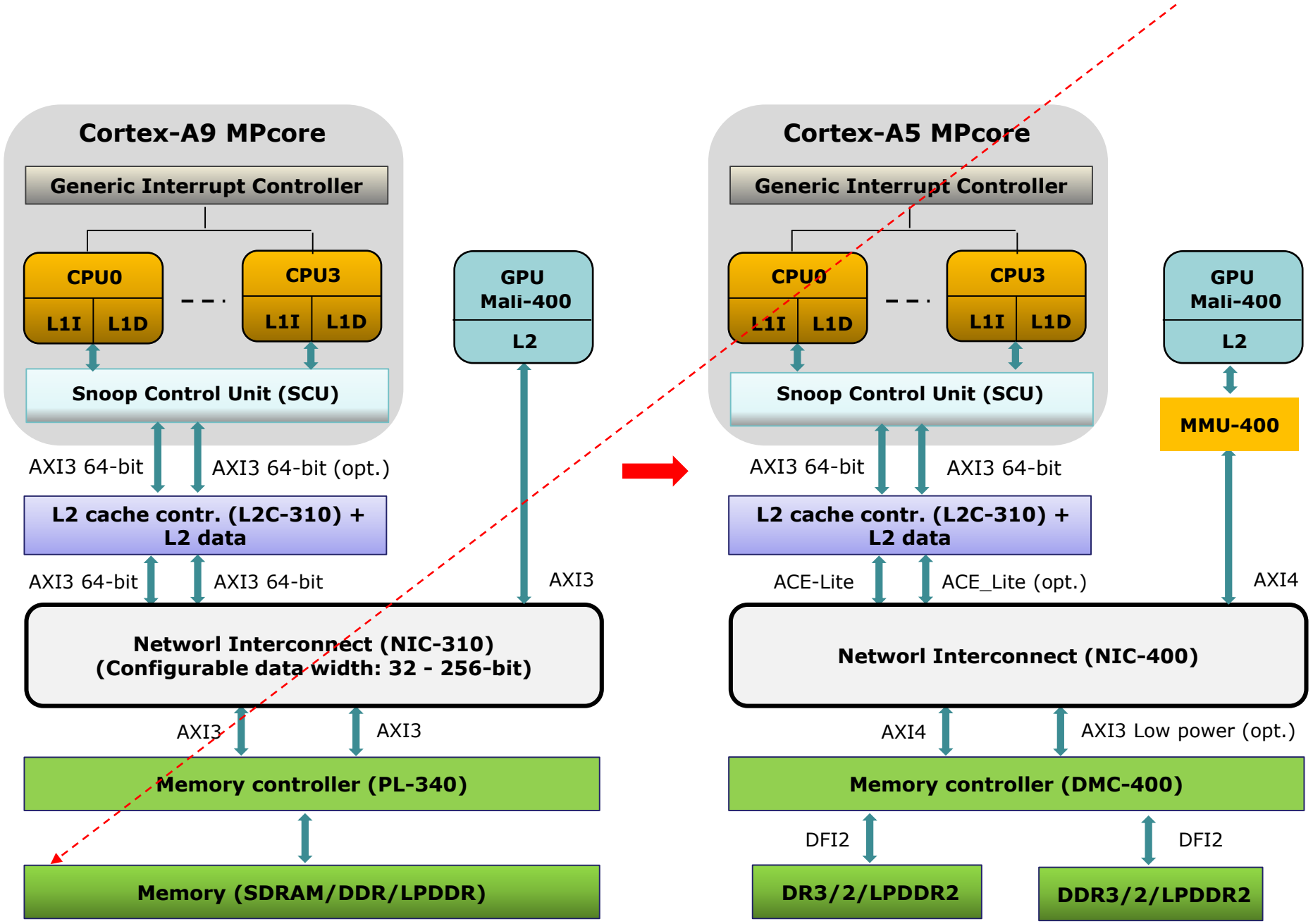

ARM11 MPcore
Generic Interrupt Controller
CPU0
L1I
L1D
CPU1
L1I
L1D
CPU2
L1I
L1D
CPU3
L1I
L1D
Snoop Control Unit (SCU)
AXI3 64-bit
AXI3 64-bit (opt.)
L2 cache controller (L2C-310) + L2 cache data
AXI3
AXI3 Low power (opt.)
Networl Interconnect (NIC-310)
AXI3
AXI3 Low power (opt.)
Memory controller (PL-340)
Memory (SDRAM/DDR/LPDDR)

Introduction of a network interconnect in the Cortex-A9 MPCore

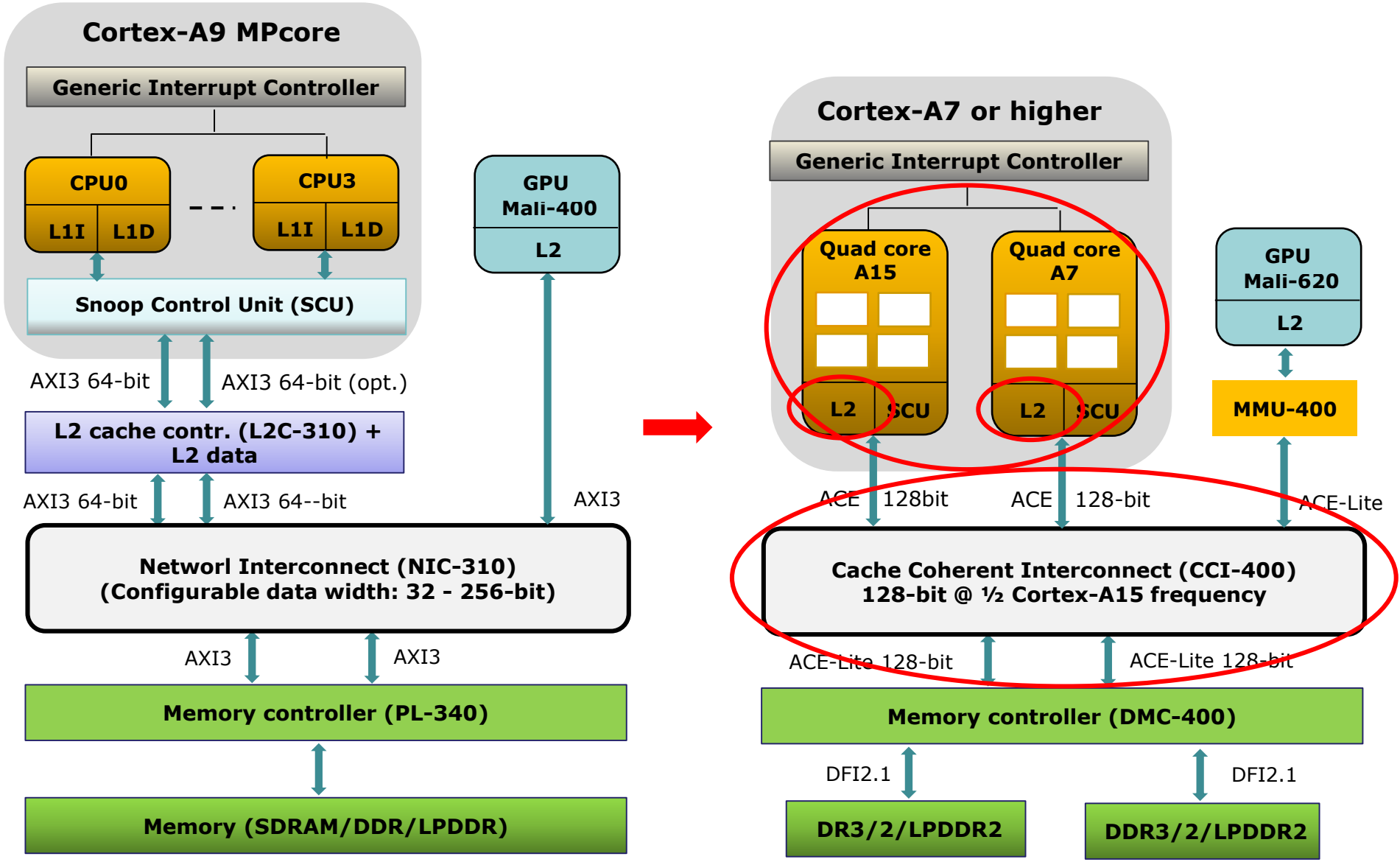


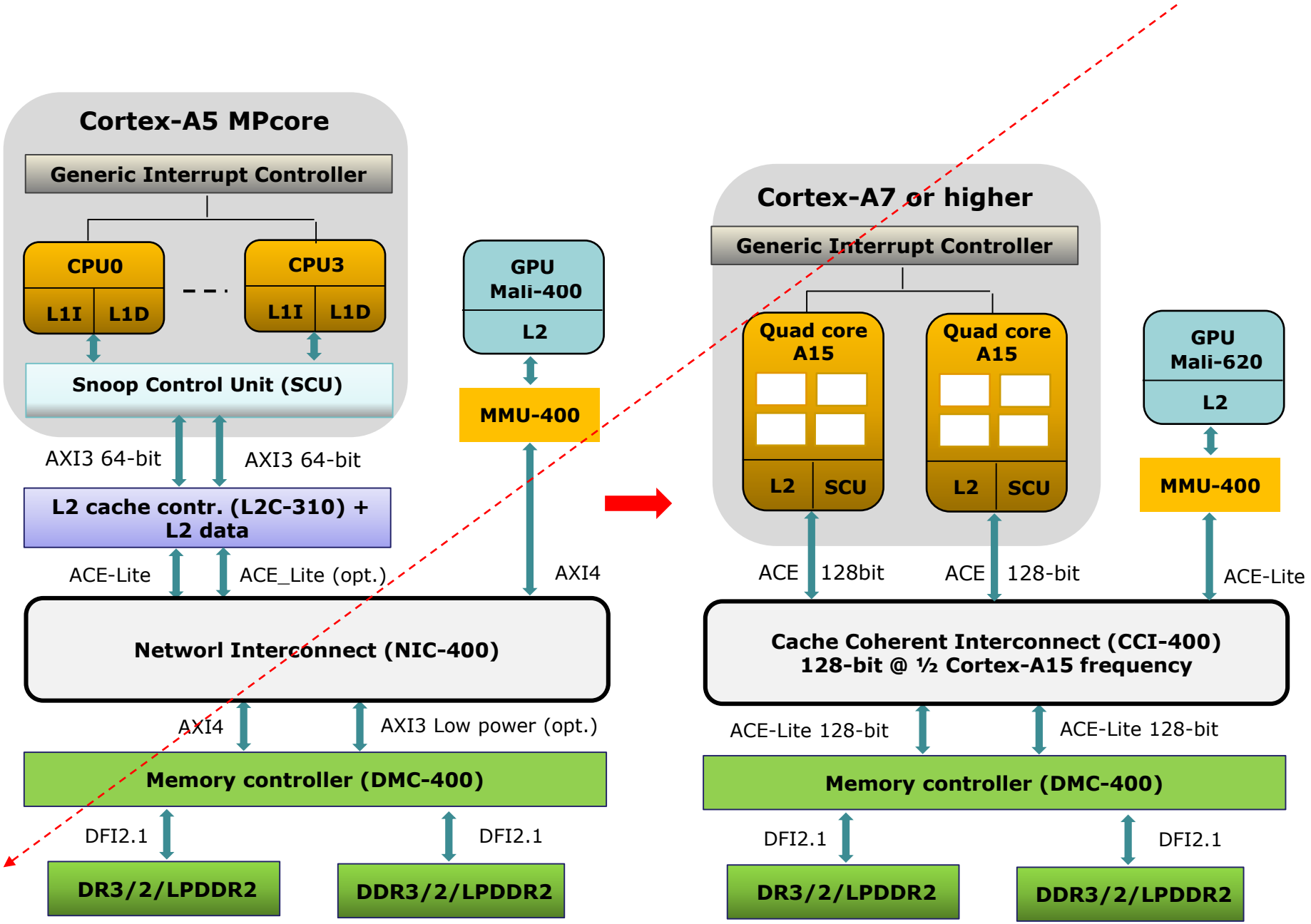
Introduction of a network interconnect along with the Cortex-A9 MPCore



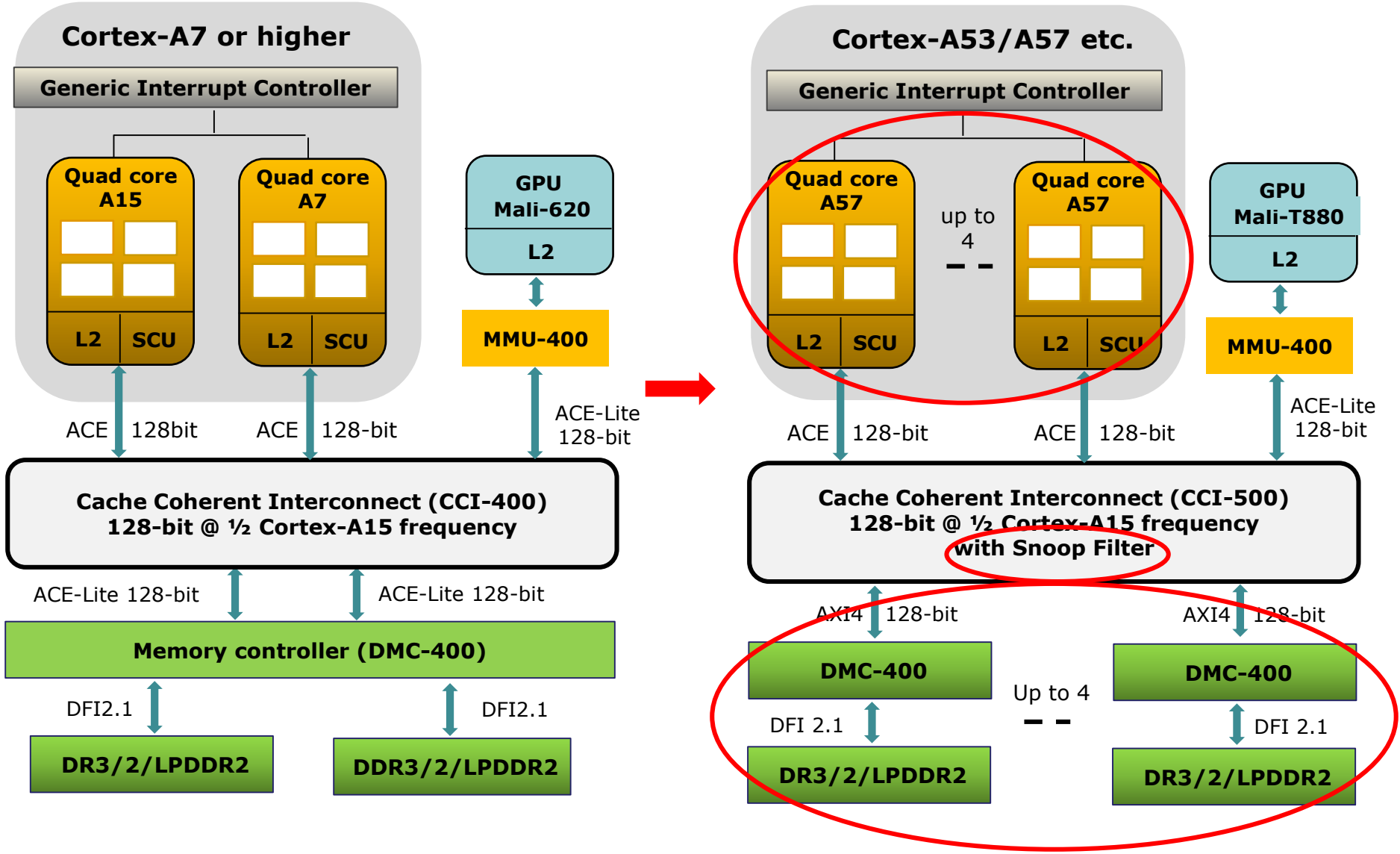

Cortex-A9 MPcore
Generic Interrupt Controller
CPU0
L1I
L1D
CPU3
L1I
L1D
Snoop Control Unit (SCU)
GPU
Mali-400
L2
AXI3 64-bit
AXI3 64-bit (opt.)
L2 cache contr. (L2C-310) +
L2 data
AXI3 64-bit
AXI3 64-bit
AXI3
Networl Interconnect (NIC-310)
(Configurable data width: 32 - 256-bit)
AXI3
AXI3
Memory controller (PL-340)
Memory (SDRAM/DDR/LPDDR)
Cortex-A5 MPcore
Generic Interrupt Controller
CPU0
L1I
L1D
CPU3
L1I
L1D
Snoop Control Unit (SCU)
GPU
Mali-400
L2
MMU-400
AXI3 64-bit
AXI3 64-bit
L2 cache contr. (L2C-310) +
L2 data
ACE-Lite
ACE_Lite (opt.)
AXI4
Networl Interconnect (NIC-400)
AXI4
AXI3 Low power (opt.)
Memory controller (DMC-400)
DFI2
DFI2
DR3/2/LPDDR2
DDR3/2/LPDDR2

Intro of integrated L2, two core clusters and Cache Coherent Interconnect

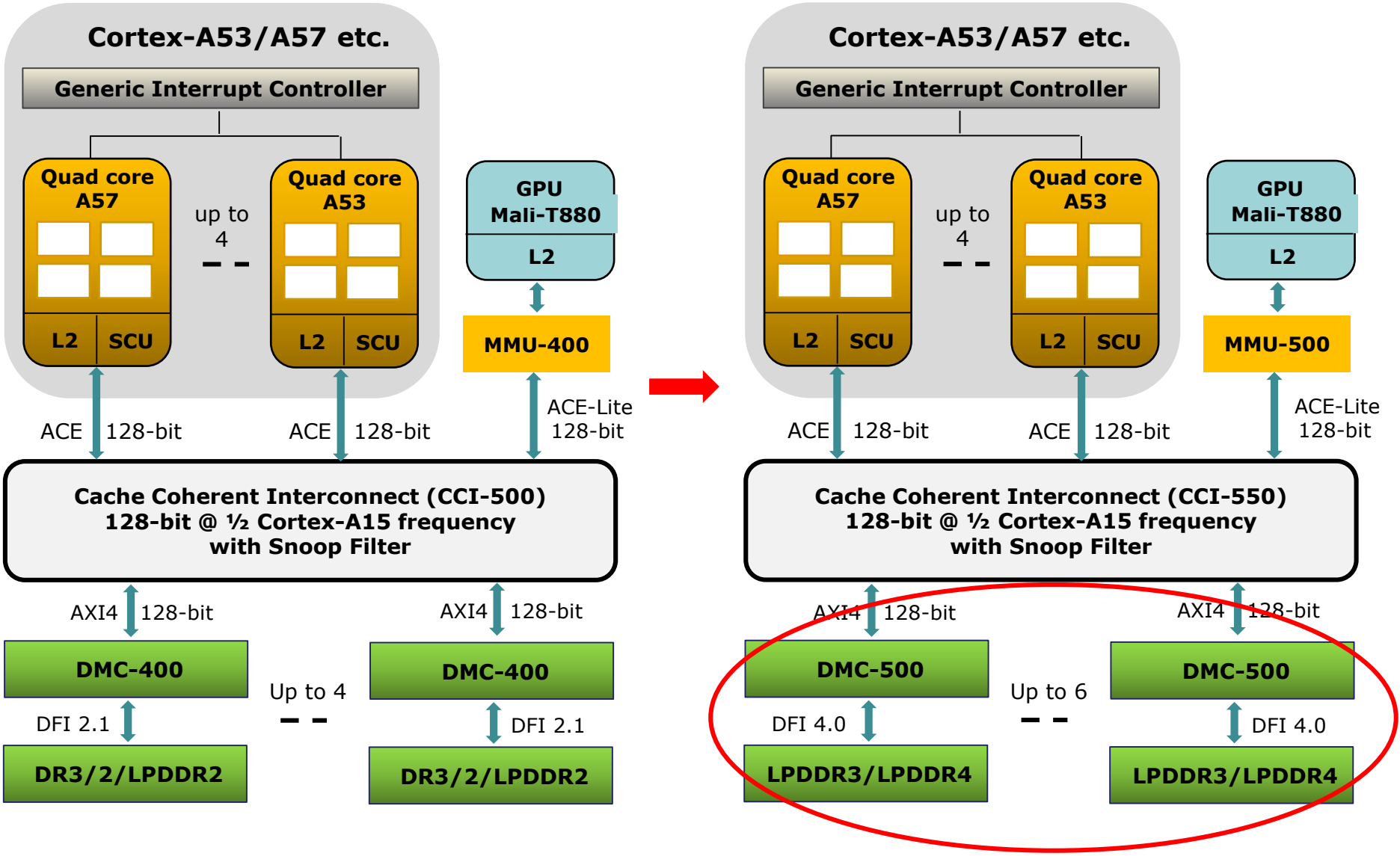



Cortex-A5 MPcore
Generic Interrupt Controller
CPU0
L1I
L1D
CPU3
L1I
L1D
Snoop Control Unit (SCU)
AXI3 64-bit
AXI3 64-bit
L2 cache contr. (L2C-310) + L2 data
ACE-Lite
ACE_Lite (opt.)
GPU Mali-400
L2
MMU-400
AXI4
Networl Interconnect (NIC-400)
AXI4
AXI3 Low power (opt.)
Memory controller (DMC-400)
DFI2.1
DFI2.1
DR3/2/LPDDR2
DDR3/2/LPDDR2
Cortex-A7 or higher
Generic Interrupt Controller
Quad core A15
L2
SCU
Quad core A15
L2
SCU
ACE
128bit
ACE
128-bit
GPU Mali-620
L2
MMU-400
ACE-Lite
Cache Coherent Interconnect (CCI-400)
128-bit @ ½ Cortex-A15 frequency
ACE-Lite 128-bit
ACE-Lite 128-bit
Memory controller (DMC-400)
DFI2.1
DFI2.1
DR3/2/LPDDR2
DDR3/2/LPDDR2

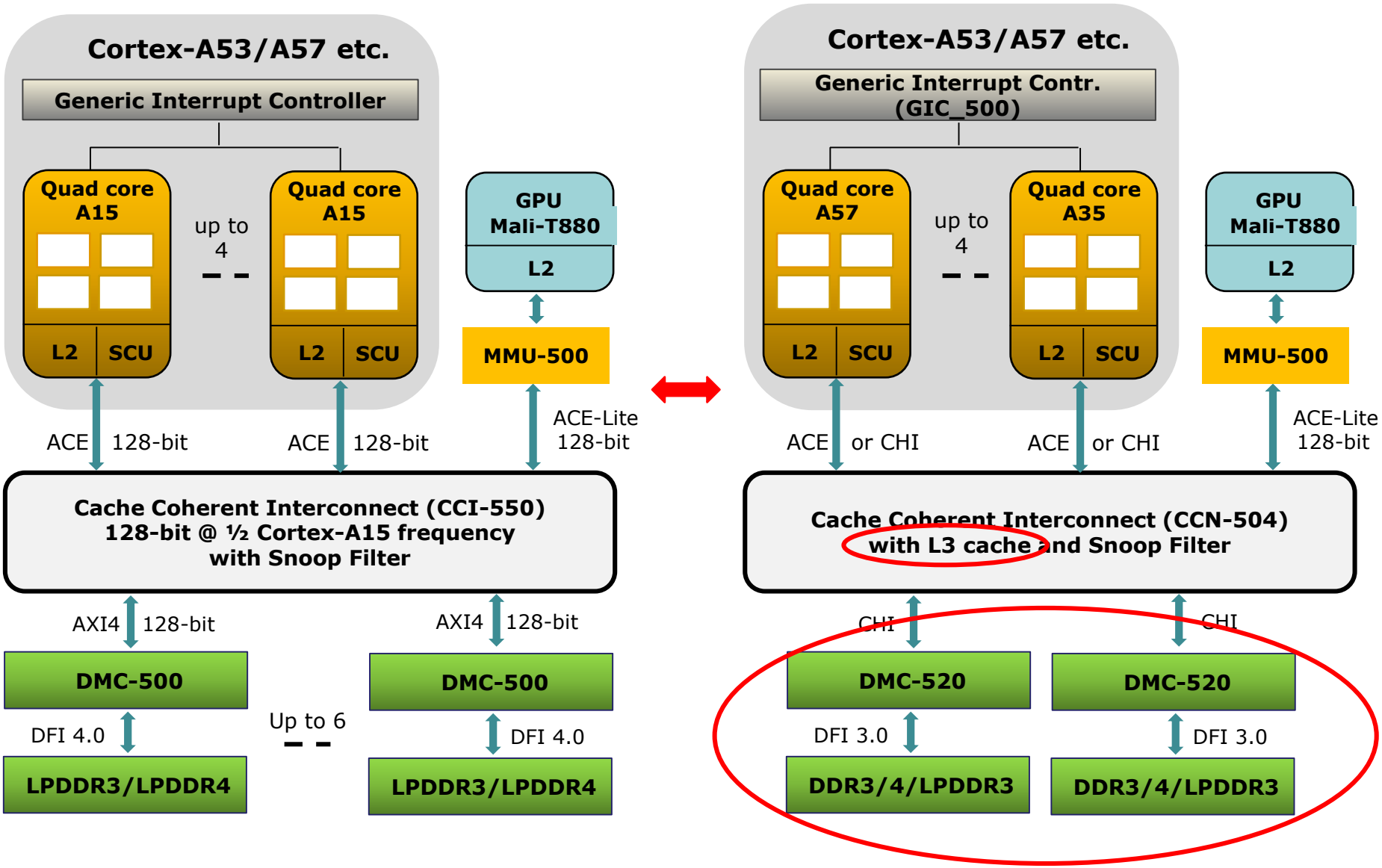
Intro of up to 4 core clusters, a Snoop Filter and up to 4 mem. channels



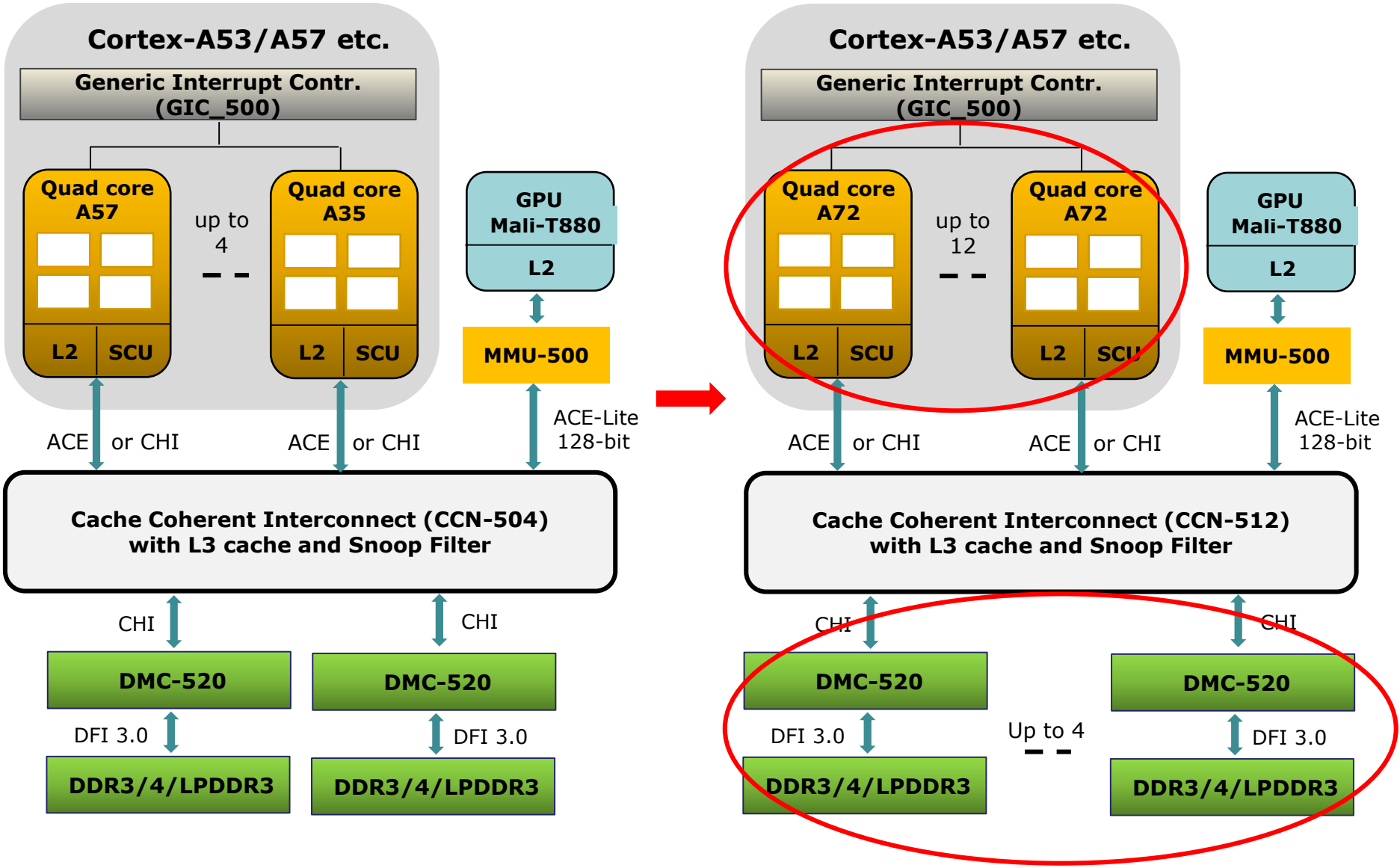
Introduction of up to six memory channels



Introduction of an L3 cache in server platforms but only 2 mem. channels



Introduction of up to 12 core clusters and up to 4 memory channels



Block diagram of the ARM11 MPCore processor-2

In the ARM11 MPCore a centralized unit, dubbed the Snoop Control Unit (SCU) maintains cache coherency.

Here we point out that subsequently we are concerned only with the L1D caches, as the read only L1I caches need a more simple management.

The SCU receives read/write requests from the cores via the Instruction and Data buses (I and D buses), as seen in the related block diagram.

In addition, the cores augment their read/write requests by sending relevant information to the SCU about the cache line requested via the CCB bus, specifying e.g. whether or not the data requested is held in the cache, what the status of the referenced cache line is, etc., as described before.

It is then the task of the SCU to manage the read/write requests such that cache coherency between the L1D and L2 caches and also the memory remains maintained according to a chosen cache coherency protocol.

Underlying principle of operation of the SCU []

- At boot time, each core can be configured to take part in the coherency domain, in which case the SCU will maintain coherency between them.
- The SCU effectively monitors the traffic between the L1D caches and the next level of the memory hierarchy and takes actions according to the chosen MESI cache coherency protocol, discussed in Section 1.x.
- Nevertheless, the implementation of the MESI protocol covers only the L1D – next memory hierarchy traffic and does not deal with the I/O traffic.

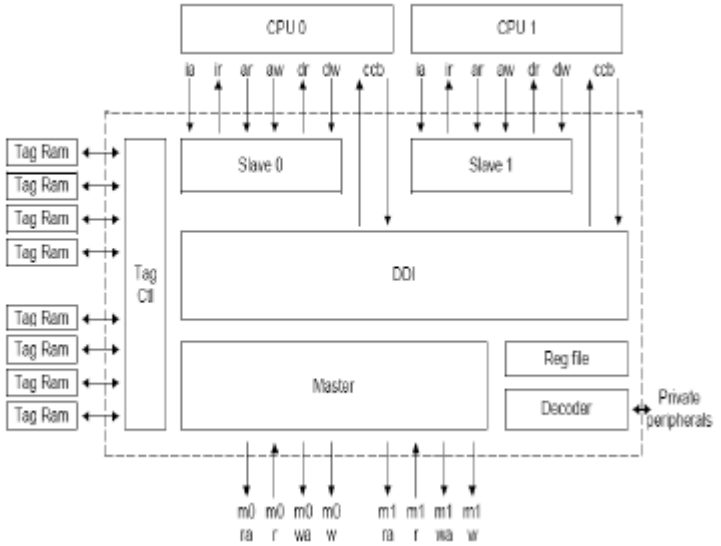
  As a consequence, cache coherency for I/O traffic (i.e. DMA transfers), need to be assured by software, as it will be discussed later in this Section.
- On the other hand, the MESI protocol became implemented with three additions in order to increase its efficiency, as described next.

Implementing DMA on ARM SMP Systems
Application Note 228 8/2009 ARM

Implementing cache coherency in the ARM11 MPCore []

Increased power efficiency and increased scalability by avoiding system accesses:

- **Processor-local coherence "Directory"**
  - Checks if data is in cache without interrupting CPU
  - Filters **Snoop** to only CPU that are sharing data
  - Allows independent tasks to run at full single thread performance resulting in linear scalability
  - Keeps power lower since directory is local
- **Direct Data Intervention (cache-2-cache transfer)**
  - Copy **clean** data from one CPUs cache to another
  - Removing need for main memory accesses reducing associated power
- **Migratory Line**
  - Move **dirty** data between CPUs and skips shared state.
  - Avoids power and latency associated with write back
- **Write allocation cache**
  - With adaptive back-off for temporally inappropriate allocation such as during a memset()



ARM MPCore Snoop Control Unit

- Clocked at CPU frequency for lower latency lookups and filtered access to CPU
- Keeps data within processor permitting lower power consumption than if time-sliced on a uniprocessor
- Power aware allowing per CPU logic and cache shutdown for advance power management.

http://www.iet-cambridge.org.uk/arc/seminar07/slides/JohnGoodacre.pdf

ARM's key extensions to the MESI protocol

Already in their patent applications [], [] AMD made three key extensions to the MESI protocol, as described next:

a) Direct Data Intervention (DDI)

b) Duplicating tag RAMs

c) Migratory line.

These extensions were implemented in the ARM11 MPCore and AMD's subsequent multicore processors, as follows.

US 7,162,590 B2

US 2005/0010728 A1

a) Direct Data Intervention (DDI) [c]

- Operating systems often let migrate tasks from one core to another.

  In this case the migrated task needs to access data that is stored in the L1 cache of another core.

- Without using a snooping mechanism (as ARM's cache management technique avoids using snooping in case of core requests) migrating cores becomes a complicated and long process.

  First the original core needs to invalidate and clean the relevant cache lines out to the next level of the memory architecture.

  Subsequently, once the data is available from the next level of the memory architecture (e.g. from the L2 or main memory), the data has to be loaded into the new core's data cache, as indicated in the right side of the next Figure).

- DDI eliminates this problem as with DDI the SCU will receive the cache line from the owner cache immediately and will forward it to the requesting core without accessing the next level of the memory hierarchy.

[c] Implementing DMA on ARM SMP Systems
Application Note 228 8/2009 ARM

Accessing cached data during core migration []

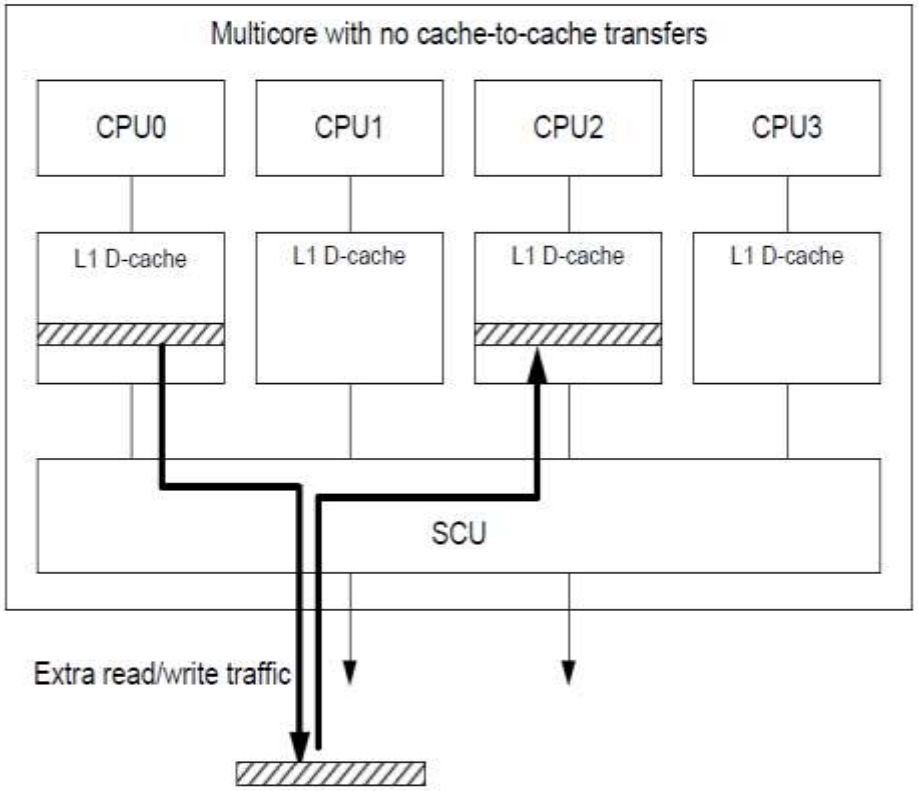


Figure: Accessing cached data during core migration without cache-to-cache transfer

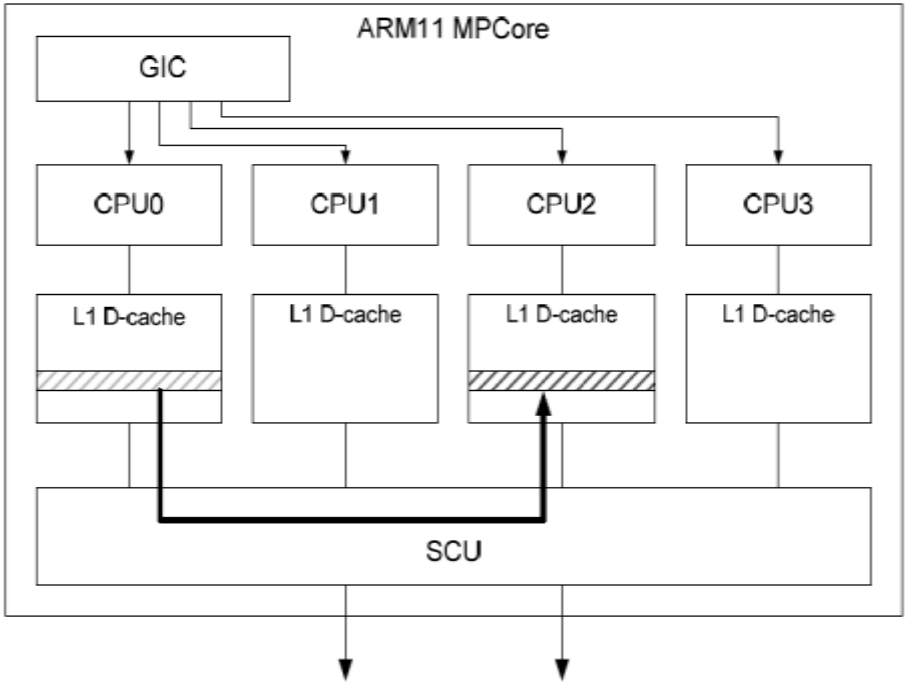


Figure: Accessing cached data during core migration with cache-to-cache transfer

b) Duplicating tag RAMs []

- With duplicating tag RAMs the SCU keeps a copy of the tag RAMs of all L1 data caches.
- The duplicates of L1 tag RAMs are used by the SCU to check for data availability before sending coherency commands to the relevant cores.

  With the possibility of checking availability of requested cache lines in all caches belonging to the cores coherency commands need to be sent only to the cores that must update their data cache.
- In addition, this feature enables also to detect if a cache line requested by a core is available in another core's data cache before looking for it in the next level of the memory hierarchy.

Maintaining I/O coherency in the ARM11 MPCore processor

Interpretation of I/O coherency []

- I/O coherency relates typically to DMA coherency.
- DMA (Direct Memory Access) allows additional bus masters to read or write system memory without core intervention, i.e. while a DMA transfer is in progress the core can continue executing code.

  DMA transfers are organized as block transfers.

  When the DMA transfer is completed, the DMA controller will signal it to the core by an interrupt request.
- DMA channels transfer blocks of data to or from devices, examples are DMA transfers used for network packet routing or video streaming.

c) Migratory lines []

- The migratory lines feature enables moving dirty data from one core to another without writing to L2 and reading the data back in from external memory.
- This avoids power and latency associated with write backs.

trm arm11 MPCore

The need for maintaining I/O coherency

Maintaining I/O coherency for DMA reads

- A write-back cache typically holds more recent data than the system memory.
- Modified data in a cache line will be marked as "dirty" and must obviously be written back to main memory before reading the memory e.g. by a DMA agent.

The process of writing back "dirty" data to the main memory is often called cache cleaning or flushing.

In other words the consistency of DMA reads presumes a previous cleaning or flushing the related cache lines.

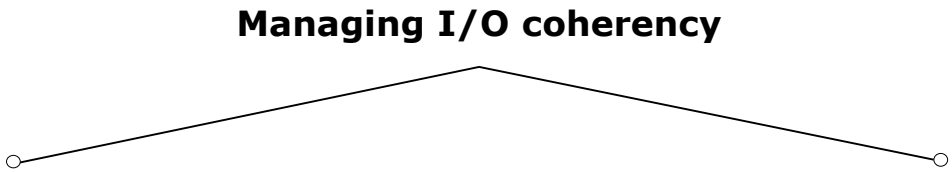
Maintaining I/O coherency for DMA writes

- If a core in a multicore processor or a processor in a multiprocessor has a local copy of data but an external agent (DMA) will write to main memory, the cache contents become out-of-date (termed also as "stale".
- Then to avoid reading of stale data from the caches after the external agent has written new data into the memory, i.e. to avoid data corruption, before writing new data into the memory, stale data must be removed i.e. invalidated from the caches.

In other words the consistency of DMA writes presumes a previous invalidation of the related cache lines.

http://community.arm.com/groups/processors/blog/2013/12/03/extended-system-coherency--part-1--cache-coherency-fundamentals

Options for managing I/O coherency []



Software managed I/O coherency

Maintaining I/O coherency is responsibility of software, typically of the OS.

The OS must ensure that cache lines are cleaned before an outgoing DMA transfer is started, and invalidated before a memory range affected by an incoming DMA transfer is accessed.
This causes some overhead to DMA operations, since such operations are performed usually by loops directed by the OS.

Used in ARM11 MPCore

Hardware managed I/O coherency

Hardware overtakes the responsibility for maintaining I/O coherency

Accesses to shared DMA memory regions are routed to the cache controller which will clean the relevant cache lines for DMA reads or invalidate them for DMA writes.

By providing the ACP port in ARM's subsequent multicore processors first in the Cortex-A9 MPCore (10/2007)

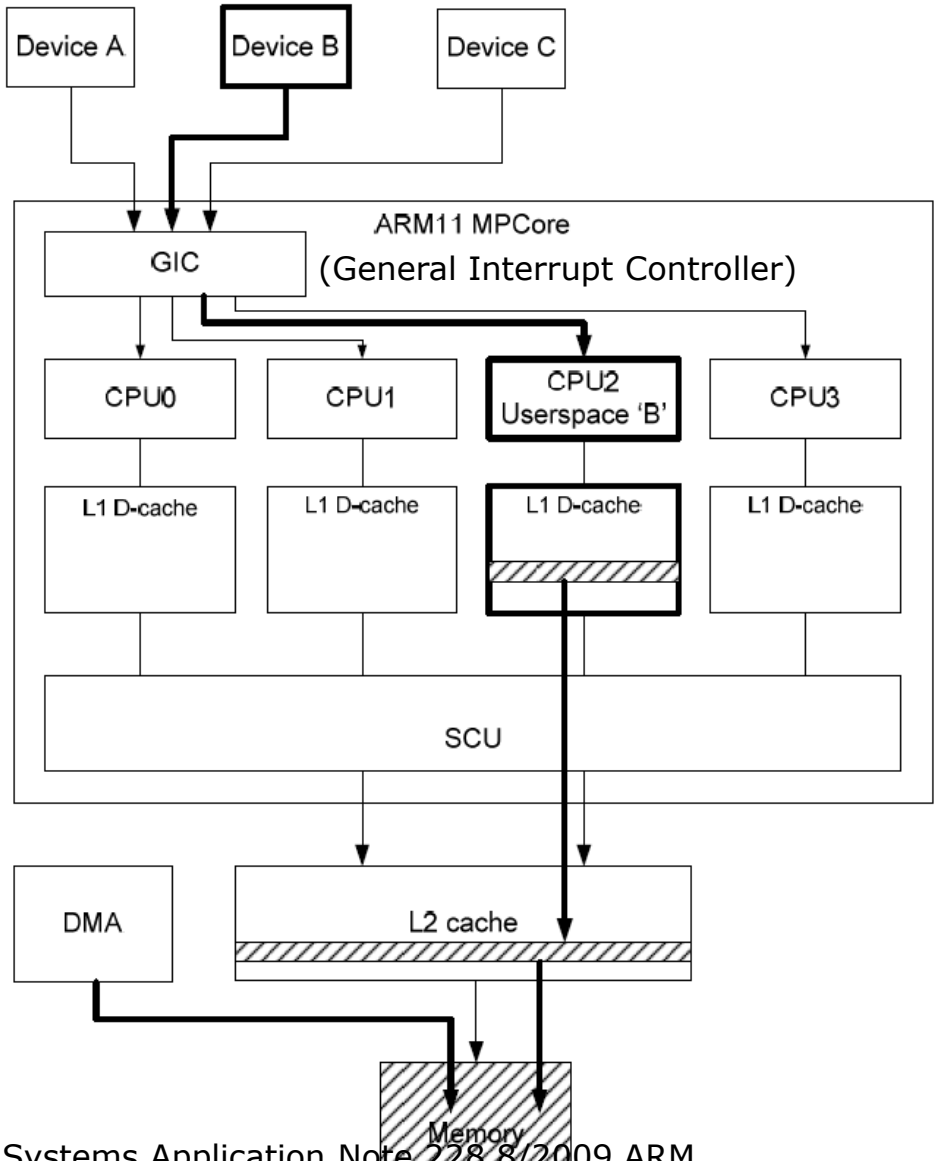
(To be discussed in the next Section).

Example: Maintaining software managed I/O coherency for writes []

- Before executing a DMA write operation, the device associated with the DMA write request (Device B) issues an interrupt request to the CPU that is associated with the DMA operation (CPU2).
- It is then the task of the interrupt handler (provided by the OS or OS patch) to care for the appropriate cache maintenance.
- Maintaining cache coherency requires invalidating all cache lines associated with the memory domain to be written into (Userspace 'B') before a DMA write operation.
- It is achieved as follows.

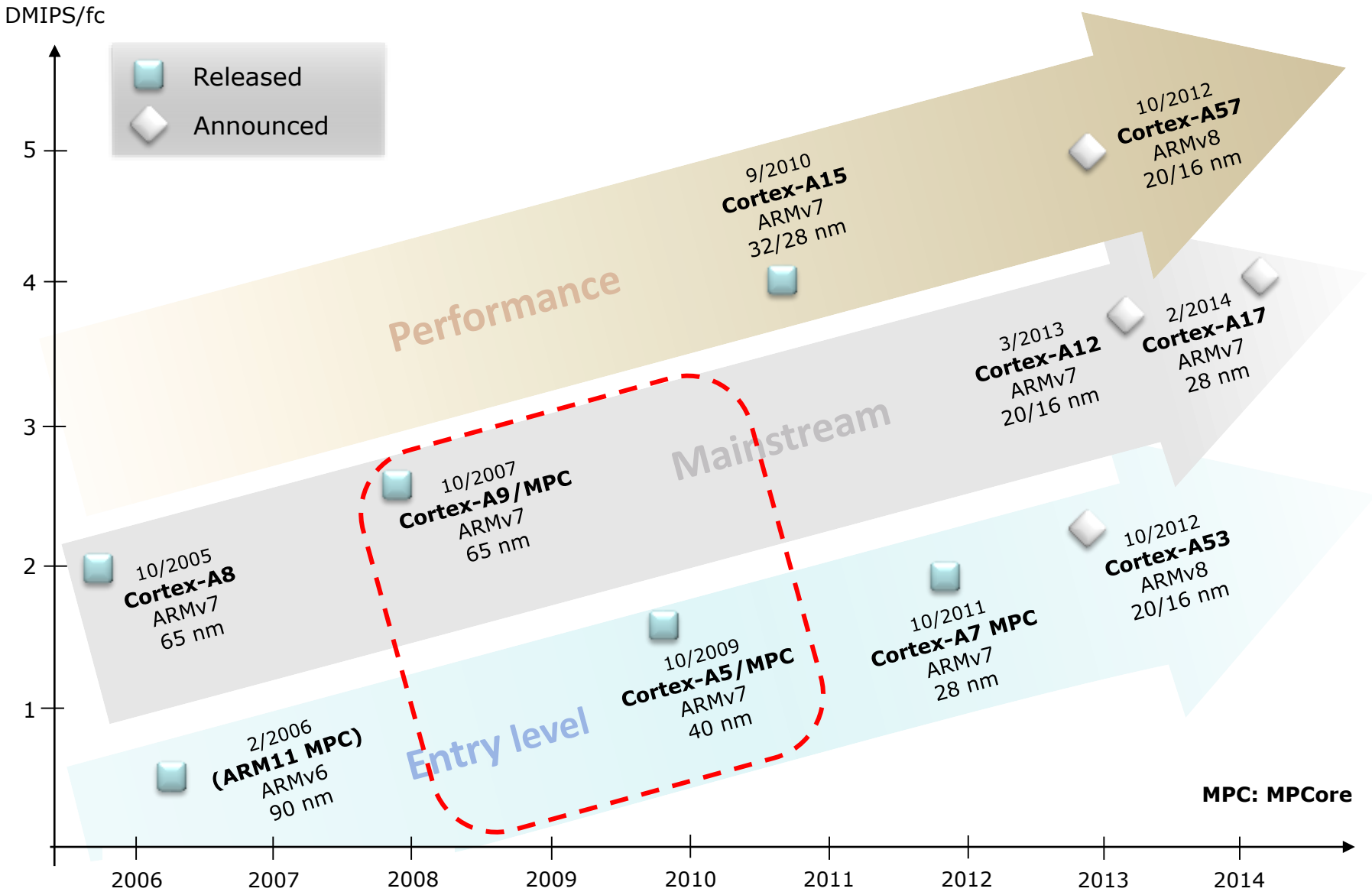
  First a word will be written into every cache line belonging to CPU2.

  In this way old data being in caches of other cores become invalidated.
- Subsequently, also the cache lines of L1D of CPU2 will be invalidated.
- Finally, the DMA write transfer can be performed.



2.2.3 ARM’s 2. generation cache coherency management

2.2.3 ARM's 2. generation cache coherency management (based on [])

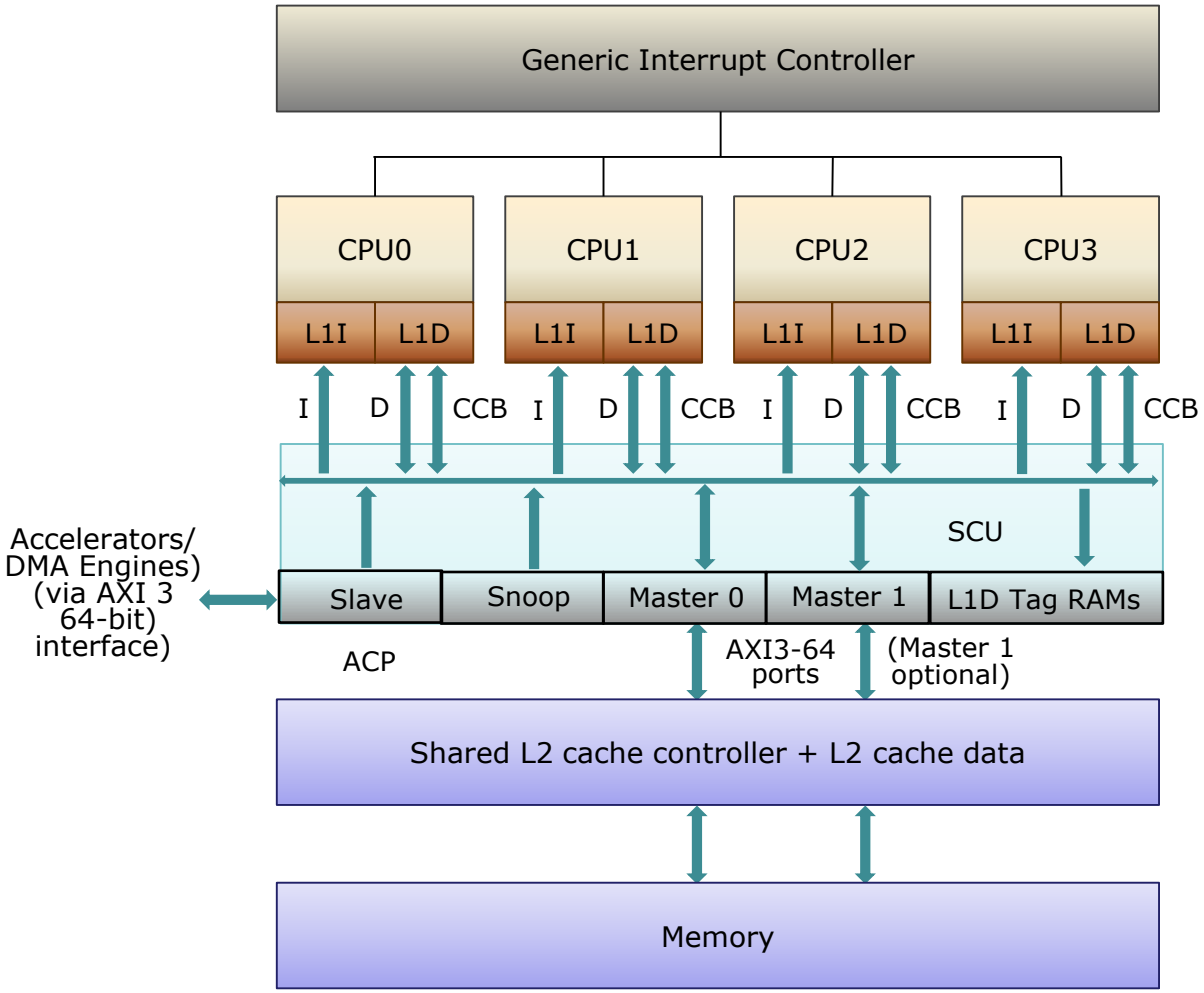


Key features of ARM's 2. generation cache coherency management

- It supports the extension of MPCore based system architectures with accelerators (and also DMA Engines) such that
  - accelerators (e.g. Crypto Engines) and also DMA Engines gain access to the CPU cache hierarchy for increasing system performance and reducing overall power consumption whereby
  - AMBA 3 AXI technology is used for compatibility with attaching standard peripherals and accelerators.
- This simplifies software and reduces cache flush overheads.
- The principle of the extension of system architecture is to provide a slave port, designated as the ACP (Accelerator Coherency Port) with an AMBA3 AXI-64 interface in the 2. generation cache coherency management for accelerators and DMA channels, as indicated in the next Figure.
- ARM's second generation cache coherency management is implemented in the Cortex-A9 MPCore and Cortex-A5 MPCore systems.

http://www.mpsoc-forum.org/previous/2008/slides/8-6%20Goodacre.pdf

Block diagram of the Cortex-A9 MPCore (4/2008)



CCB: Coherency Control Bus

- Enhanced MESI
- SCU holds copies of each L1D directory to reduce snoop traffic between L1D and L2
- Cache to cache transfers supported

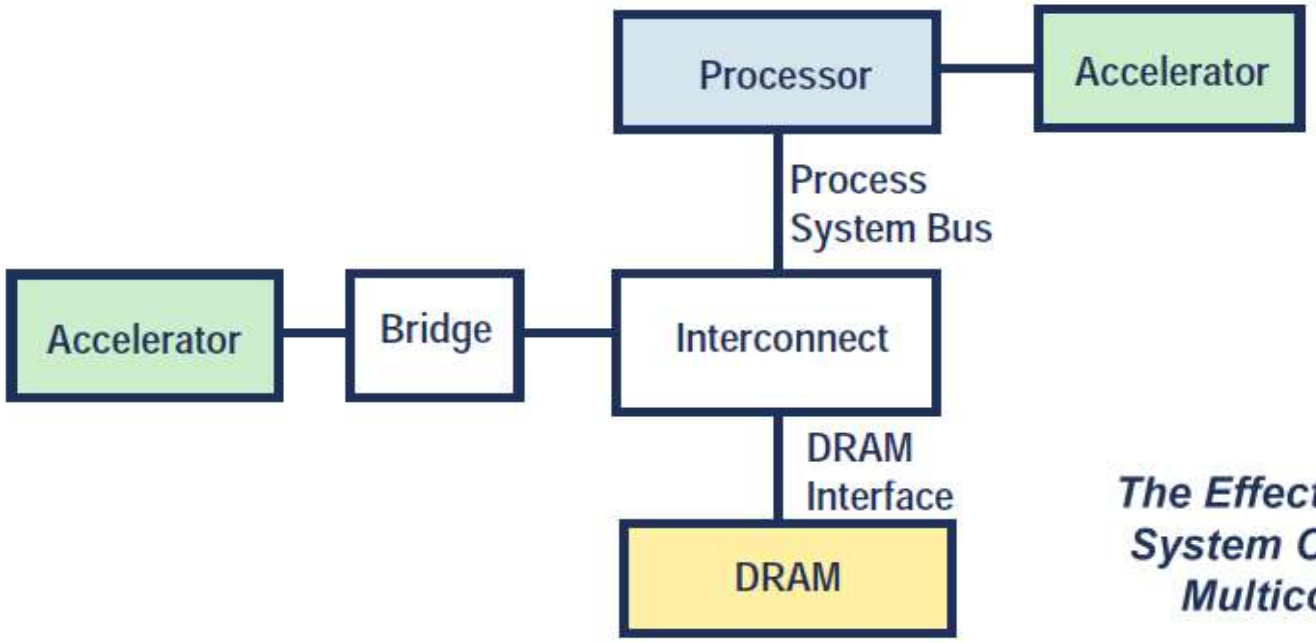
Support of attaching accelerators to the system architecture effectively via a dedicated, cache coherent port

Alternative ways to attach accelerators to SoC designs []

Basically, there are three alternatives to attach accelerators to a system architecture, as follows:

- Attaching accelerators over a system interconnect,
- attaching accelerators tightly coupled to the processor, and
- attaching accelerators on die

as indicated on the next Figure.



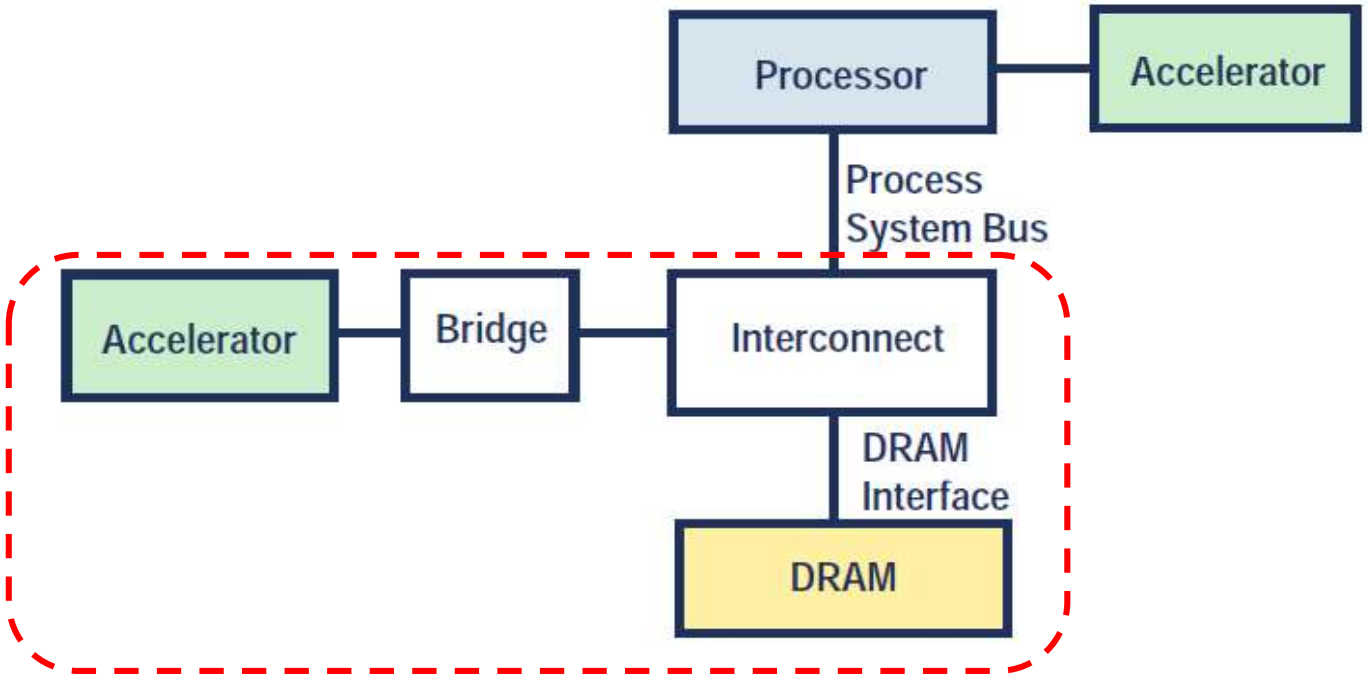
The Effect and Technique of System Coherence in ARM Multicore Technology

John Goodacre
Senior Program Manager
ARM Processor Division

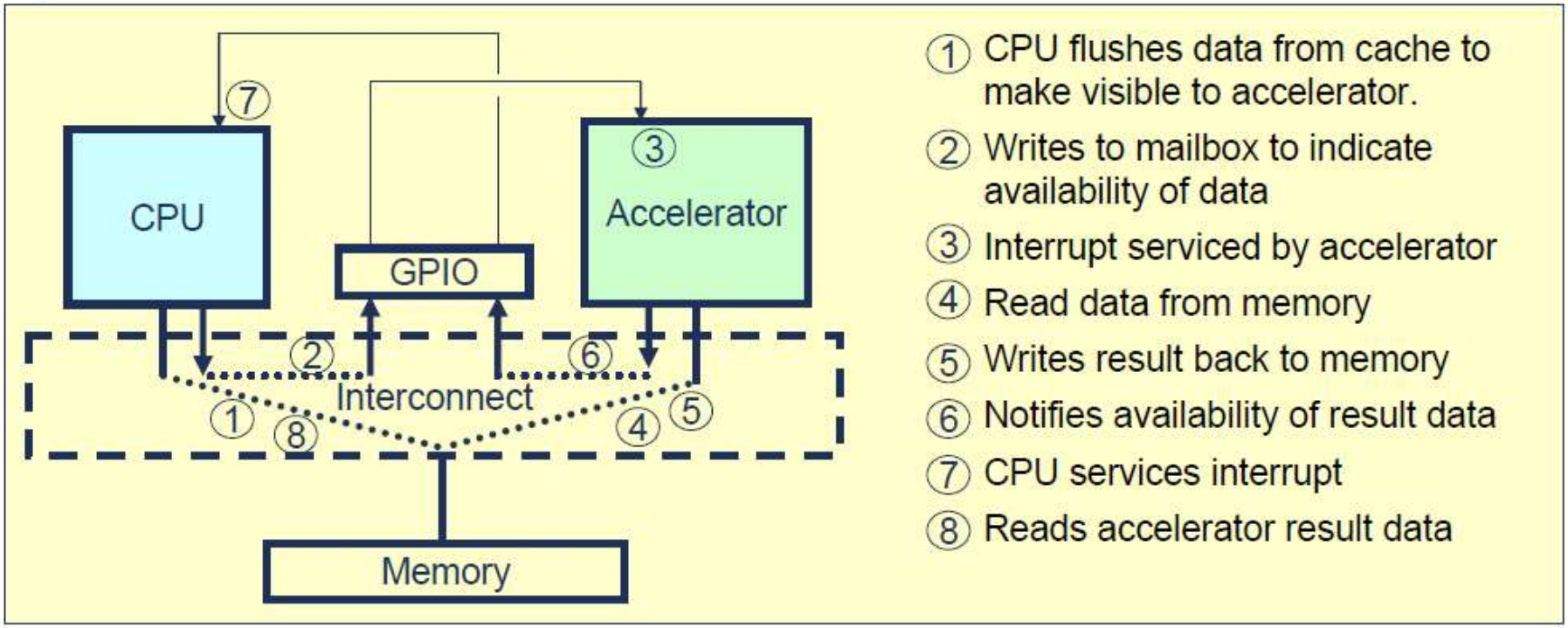
2008

Figure: Alternative ways to attach accelerators to a system architecture []

a) Principle of attaching an accelerator to the system via an interconnect-1 []



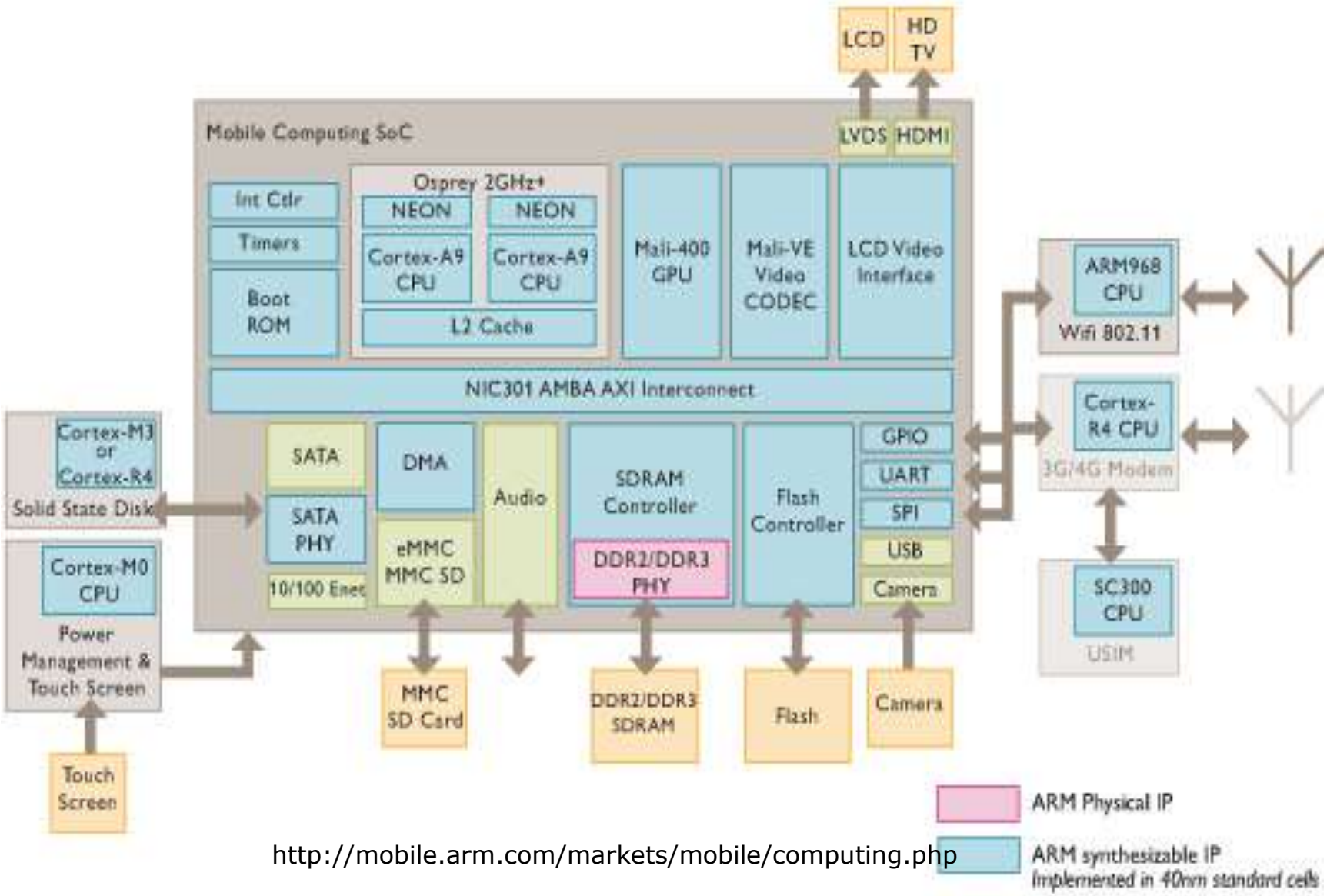
Principle of attaching an accelerator to the system via an interconnect-2 []



Impediments of the traditional way of attaching accelerators to the system architecture

- Inefficient usage of CPU caches
- significant performance and power implications of data movements
- high signaling latencies due to mailbox and interrupt latencies.

System example 1: Attaching an accelerator (Mali-400 GPU) and DMA to the system via an interconnect (NIC-301) []

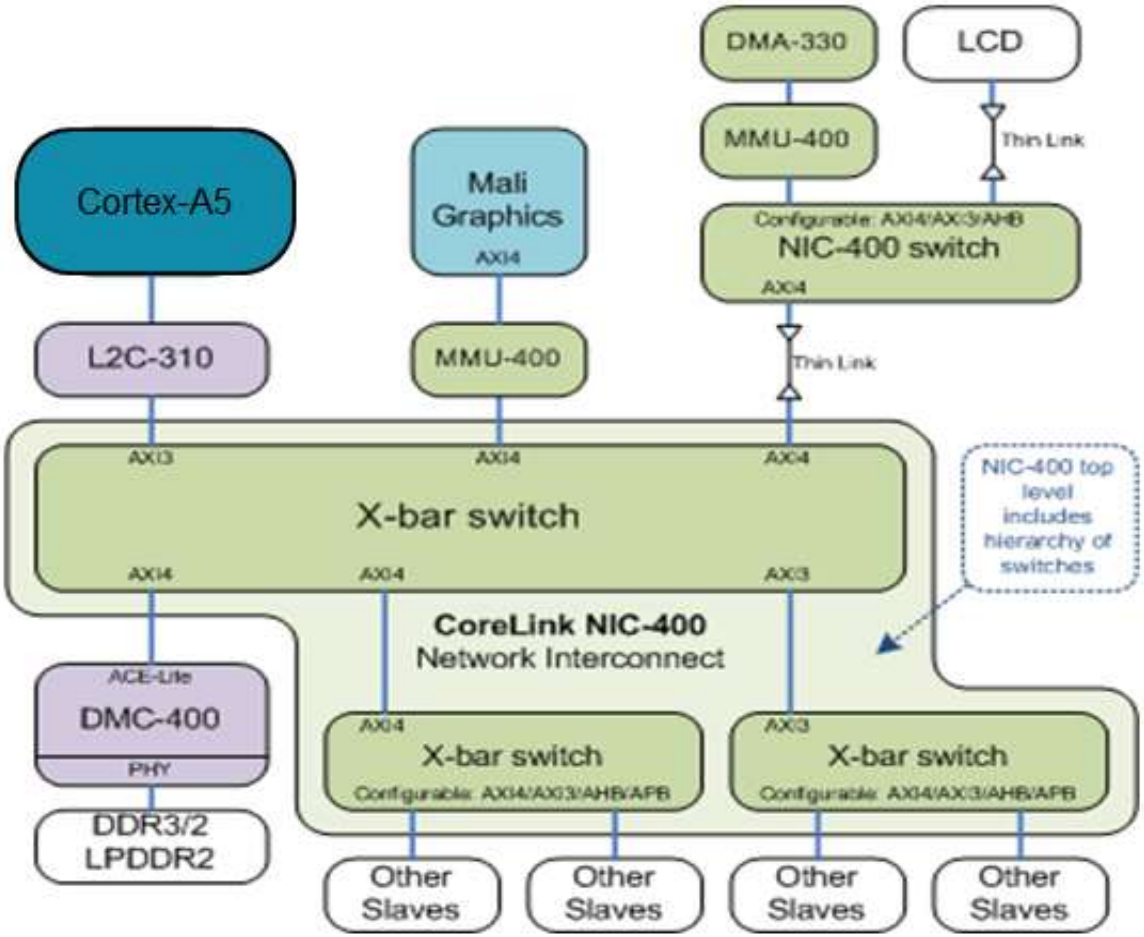


http://mobile.arm.com/markets/mobile/computing.php

Note

In this case cache coherency needs to be maintained by software.

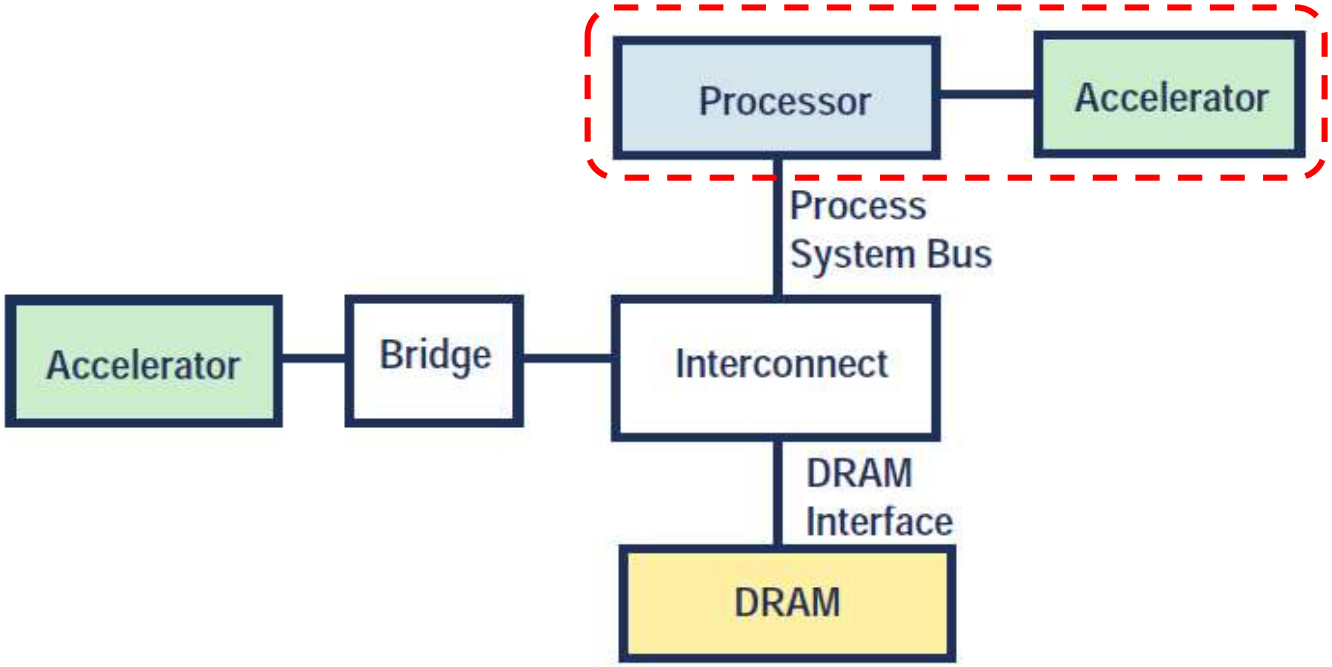
System example 2: Attaching an accelerator (Mali GPU) and DMA to the system via an interconnect (NIC-400 X-bar switch) []



Note http://www.arm.com/products/processors/cortex-a/cortex-a5.php

In this case cache coherency needs to be maintained by software as well.

b) Principle of attaching an accelerator directly to the processor []

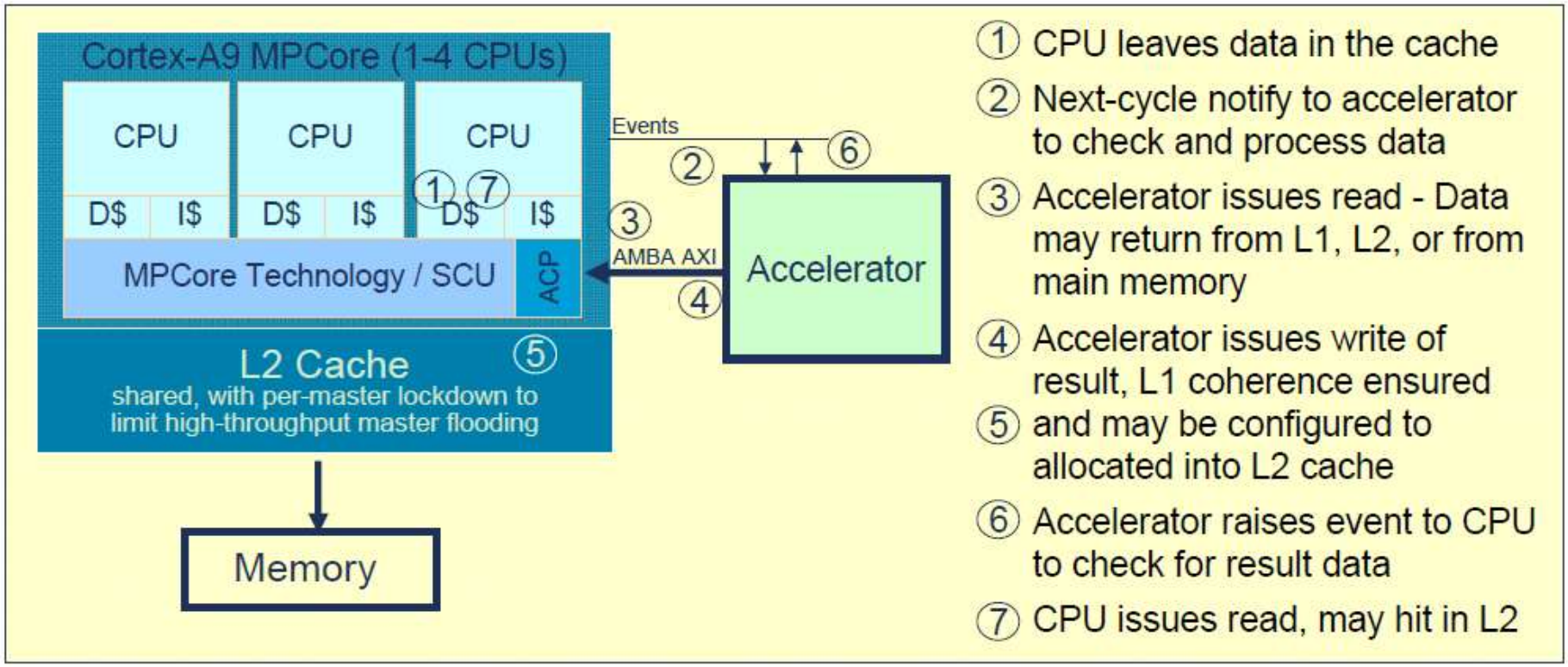


The Effect and Technique of System Coherence in ARM Multicore Technology

John Goodacre
Senior Program Manager
ARM Processor Division

2008

c) Attaching an accelerator directly to the processor via the ACP port []



The Effect and Technique of System Coherence in ARM Multicore Technology

John Goodacre
Senior Program Manager
ARM Processor Division

2008

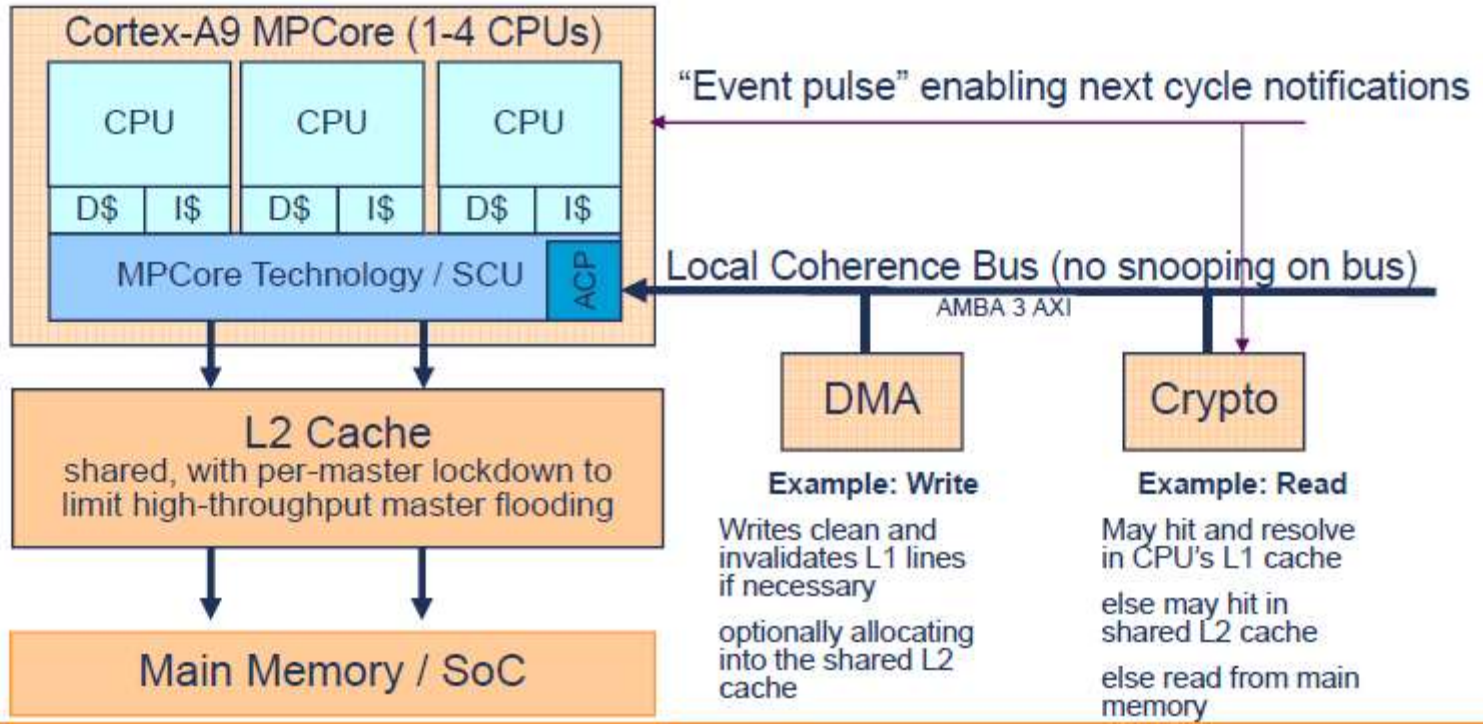
The ACP (Accelerator Coherency Port) [], []

- The ACP port is a standard AMBA 3 AXI (AXI3) slave port provided for non-cached peripherals, such as DMA Engines or Cryptographic Engines.

  It is an interconnect point for a range of peripherals that for overall system performance, power consumption or software simplification are better interfaced directly with the processor.
- The ACP port allows a device direct access to coherent data held in the processor's caches or in the memory, so device drivers that use ACP do no need to perform cache maintenance, i.e. cache cleaning or flushing to ensure cache coherency.
- It is 64-bit wide in ARM's second generation cache coherency implementations (in ARM Cortex-A9/A5 MPCore processors), but 64 or 128-bit wide in subsequent 3. generation implementations.
- The ACP port is optional in the ARM Cortex-A9 MPCore and mandatory in subsequent Cortex-A processors (except low-cost oriented processors, such as the Cortex-A7MPCore).

A9 MPCore trm

System example for using the ACP port []



http://www.iet-cambridge.org.uk/arc/seminar07/slides/JohnGoodacre.pdf

Comparing the traditional way of attaching accelerators with attaching accelerators via the ACP port []

- Example: CRC engine for TCP packet forwarding on 64 byte packet
  - Using typical system latency, ignoring common processing overhead
  - Assumed writes are fully buffered

Algorithm Stage	Approximate Cycle Counts	
Design style	Traditional shared memory with mailbox communication	ACP attached accelerator with synchronous event
Packet received and processed by CPU	0	0
Flush cache to make data visible to accelerator	20	0
Accelerator notified of data availability	> 4 [write to mailbox GPIO]	1 [Send Event]
Accelerator Reads data from cache line	Typical - 120 [read data from off-chip]	10 [read from L1/L2]
Accelerator Write data (assuming buffered)	8	
Processor reads processes cache line	Typical - 120	
Total latency overhead	**~272 cycles**	

The Effect and Technique of System Coherence in ARM Multicore Technology

John Goodacre
Senior Program Manager
ARM Processor Division

2008

Benefits of attaching accelerators via the ACP port []

- About 25 % reduction in memory transactions due to reduction of cache flushes,
- software no longer needs to be concerned with cache flushes, which can be particularly troublesome on a multicore processor.

The Effect and Technique of System Coherence in ARM Multicore Technology

John Goodacre
Senior Program Manager
ARM Processor Division

2008

Remark

- In 10/2006 AMD announced their Fusion project to integrate the CPU and GPU (i.e. an accelerator) on the same die.
- Nevertheless, at the same time AMD did not reveal any concept how to solve the data communication overhead between the CPU and the GPU.
- This missing concept was then announced by the Heterogeneous System Architecture (HSA) concept in 6/2011 (at AMD's Fusion Developer Summit).

  Originally, HSA was termed as the Fusion System Architecture (FSA) but became renamed to HSA in 1/2012 to give the concept a chance to became an open standard.

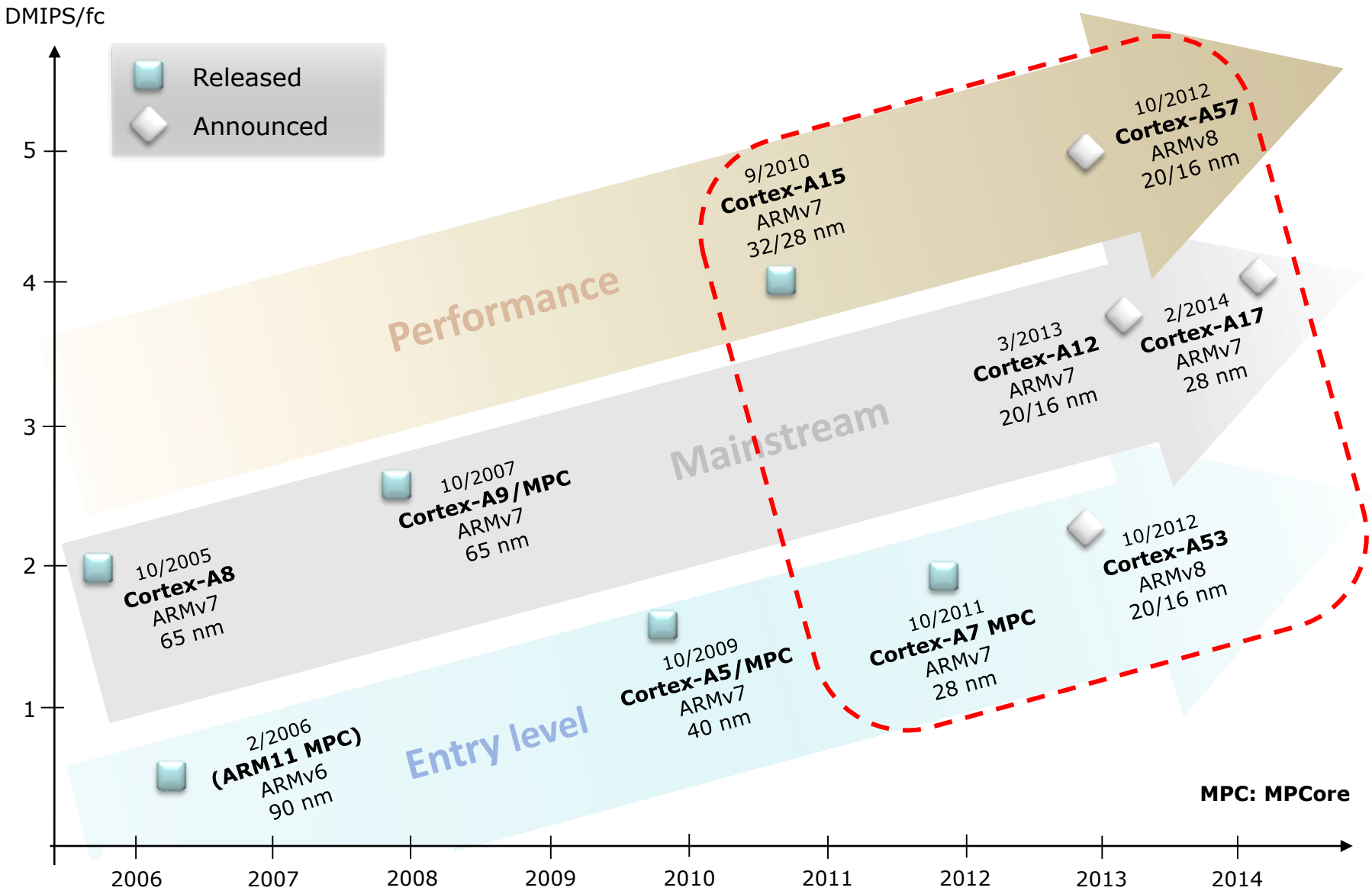
  Actually, HSA is an optimized platform architecture for OpenCL providing a unified coherent address space allowing heterogeneous cores working together seamlessly in coherent memory.

Remark to the implementation cache coherency management in the Cortex-A5 MPCore (5/2010)

It is basically the same implementation as in the Cortex-A9 MPCore, nevertheless the L1D cache coherency protocol was changed from enhanced MESI to enhanced MOESI [].

2.2.4 ARM’s 3. generation cache coherency management

2.2.4 ARM's 4. generation cache coherency management (based on [])



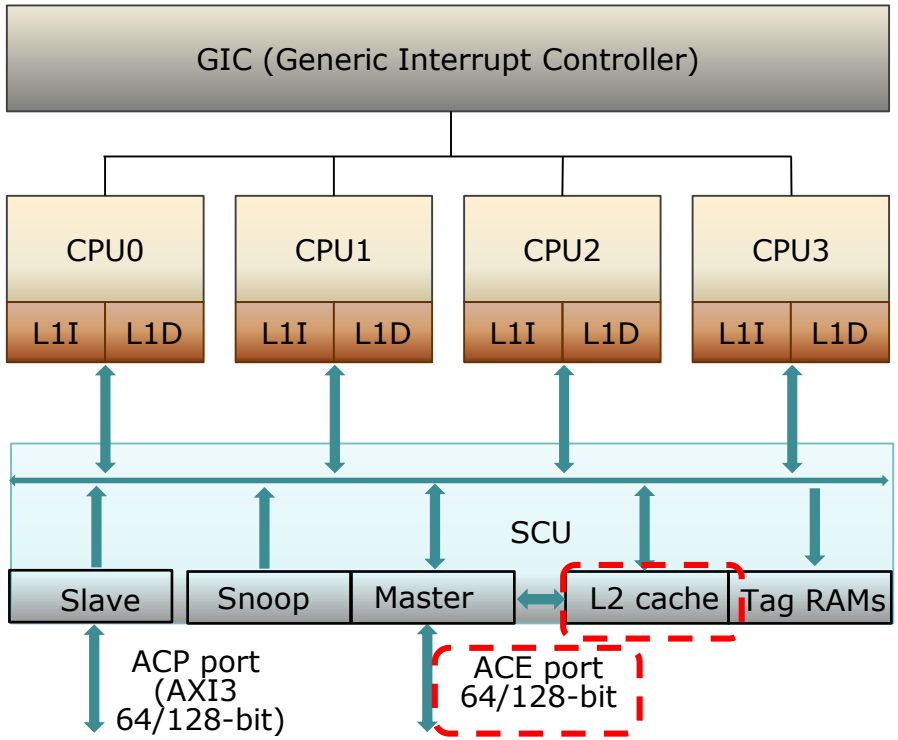
Major innovations of AMD's 3. generation cache coherency management

a) Complete redesign of the previous implementation.
   The new implementation is based on an integrated L2 cache with SCU functionality.

b) Enhanced master port carrying external snoop requests as well,

as shown in the next Figure for the Cortex-A15 processor.

Block diagram of the Cortex-A15 MPCore (4/2011)



Interconnection not specified

- L1D: Enhanced MESI
- L1/L2: MOESI
- SCU holds copies of each L1D directory to reduce snoop traffic between L1D and L2
- Cache to cache transfers

a) Complete redesign of the previous implementation []

- ARM's second generation cache coherent management technique is based on ARM's patents [], [] that introduced among others a CCB bus, duplicated L1D tag RAMs and an appropriate hardware logic (SCU).
- In contrast, their third generation cache coherency management technique is completely redesigned such that the processor includes an integrated L2 cache with SCU functionality and can handle also external snoop requests that arrive on the ACE master interface.

Principle of operation (simplified) [], []

- The L2 cache services L1I and L1D cache misses coming from each core or arising from memory accesses coming from the ACP port.

  It also handles external snoop requests arriving on the master interface (AMBA 4 ACE in the Cortex-A15/A7/A12 processors or AMBA 4 ACE or AMBA 5 CHI in the Cortex-A57/A53 processors).

  Note that in the previous generation cache management technique one or optionally two masters were attached to the processor via the AXI 3 interface did not carries external snoop requests in contrast to the AMBA 4 ACE or AMBA 5 CHI interface that include snoop channels and additional signals as well.

- The L2 cache system includes duplicate copies of L1D data cache tag RAMs from each core for handling snoop requests.

  The duplicate tags are used to filter snoop requests so that the processor can function effectively even for a high volume of snoop requests.

- When an internal L2 snoop request or an external snoop request hits in the duplicate tags a snoop request is made to the appropriate L1D cache.

  The core prioritizes external requests from the SCU over internal request coming from the L2 cache.

  The L1D caches include coherency logic to handle snoop requests.

A15 trm

A7 trm

b) Enhanced master port carrying external snoop requests

With the third generation cache coherency technology the master port became a single

- ACE 64/128 bit master port for the Cortex-A15/A7/A12 processors and
- ACE or CHI master interface for the Cortex-A57/53 processors,

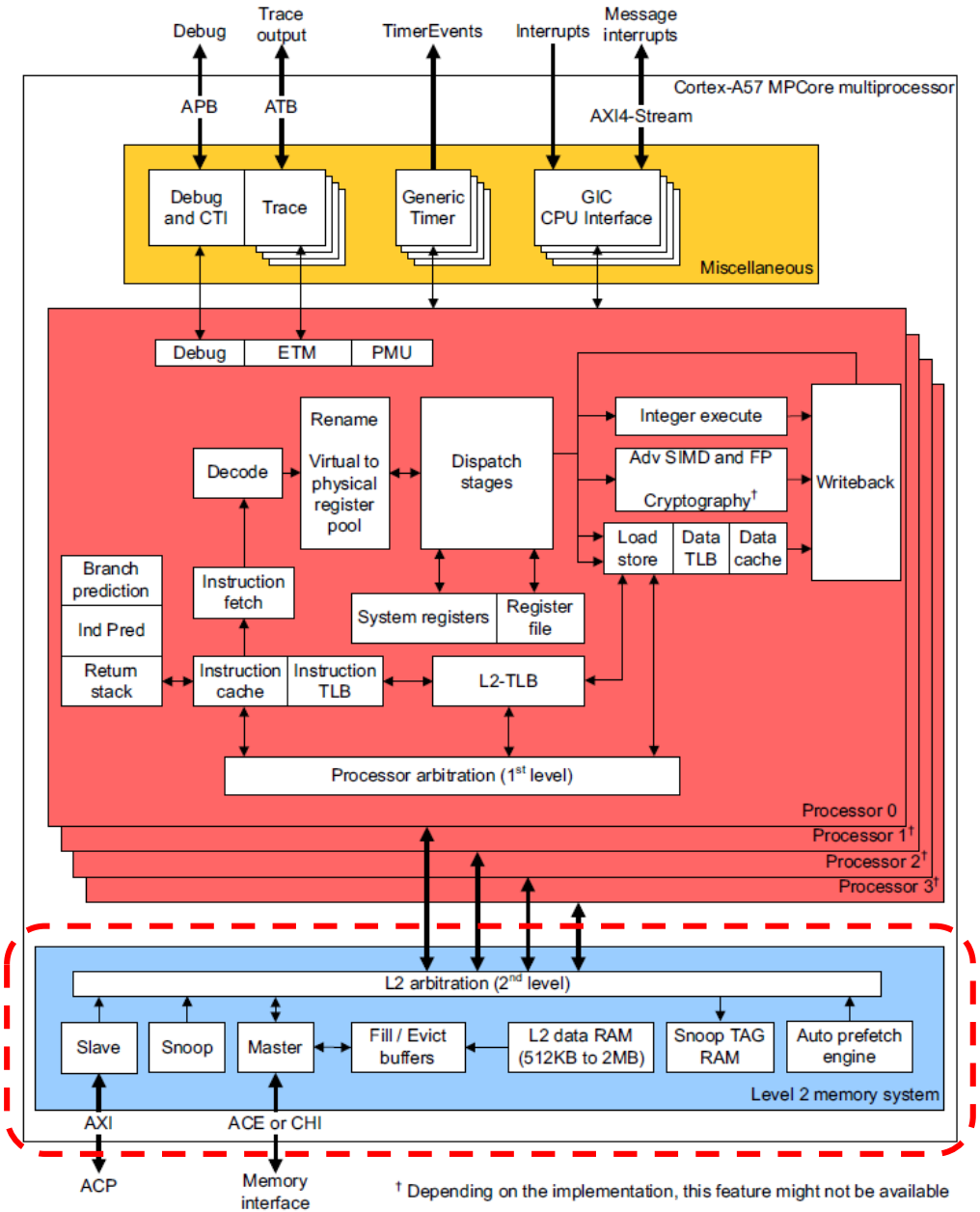
instead of one standard and one optional AMBA 3 AXI-64 ports provided by the previous generation cache coherency technique.

Note that the key difference concerning the port provision is that the ACE or CHI buses carry snoop requests from external sources whereas the AXI 3 bus didn't.

This a major enhancement since it provides the possibility to build up cache coherent systems of multiple heterogeneous processors including processors core clusters.

Consequently, the new design has to handle both internal and external snoop requests.

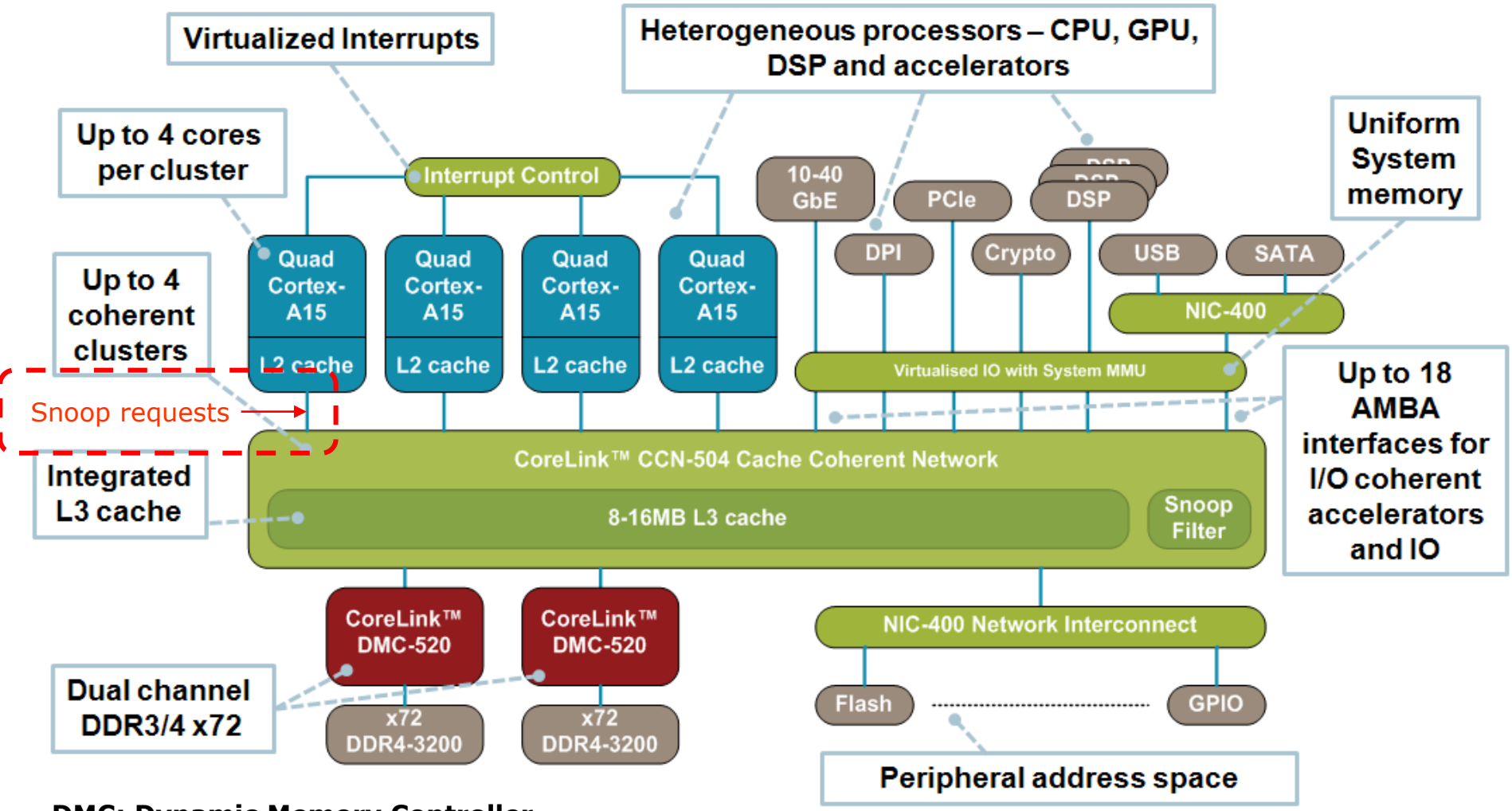
Case example 1:
ARM's Cortex-A57 processor []



SCU

http://infocenter.arm.com/help/topic/com.arm.doc.ddi0488c/DDI0488C_cortex_a57_mpcore_r1p0_trm.pdf

Case example 2: Cache coherent system based on a Cache Coherent Network controller []



DMC: Dynamic Memory Controller

http://www.arm.com/products/system-ip/interconnect/corelink-ccn-504-cache-coherent-network.php

Remark

Beginning with the Cortex-A15/A7 processors ARM changed their previous naming scheme while designating single core processors by Cortex-Ax and multi-core processors by Cortex-Ax MPCore.

In their new naming scheme ARM designates all of their Cortex-A processors by Cortex-Ax without attaching the MPCore tag to the name string.

2.3 Maintaining system wide cache coherency in ARM's multiprocessor platforms

2.3.1 Introduction of system-wide cache coherency in ARM's platforms

2.3.1 Introduction of system-wide cache coherency in ARM's platforms

Increased need for system wide cache coherency in new applications []

- OpenCL provides access to the vast processing power of Mali™-T604
- Applications include:
  - Video editing and effects
  - Camera & image processing (e.g. smile detection)
  - Image recognition (e.g. automotive lane detection)
  - Gesture recognition systems
  - Game engines (physics engines, particle physics)
  - Photorealistic ray tracing
- And Artificial Inte

Building High Performance, Power Efficient Cortex and Mali systems with ARM CoreLink

Robert Kaye

Key components of system wide cache coherency, as introduced by ARM

The introduction of system wide cache coherency has a number of prerequisites that will be discussed in the next Sections, as follows:

2.3.2 Defining a system wide cache coherency model

2.3.3 Introducing the AMBA 4 ACE interface

2.3.4 Supporting two kinds of coherency; full coherency and I/O coherency

2.3.5 Introducing the concept of domains

2.3.6 Providing support for system wide page tables

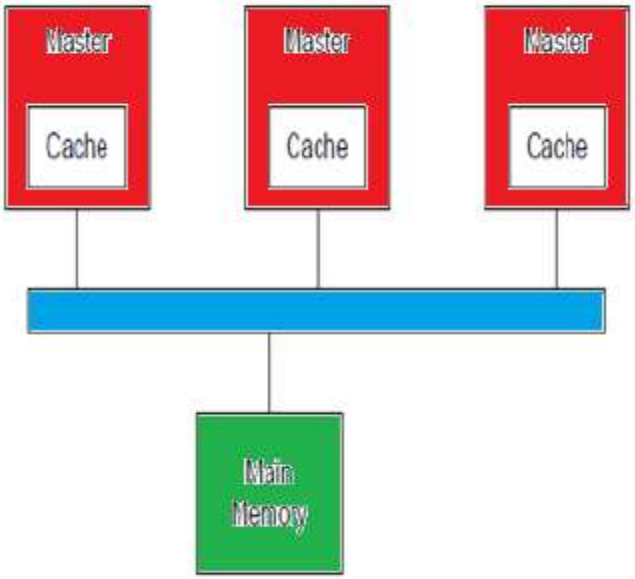
2.3.7 Providing cache coherent interconnects

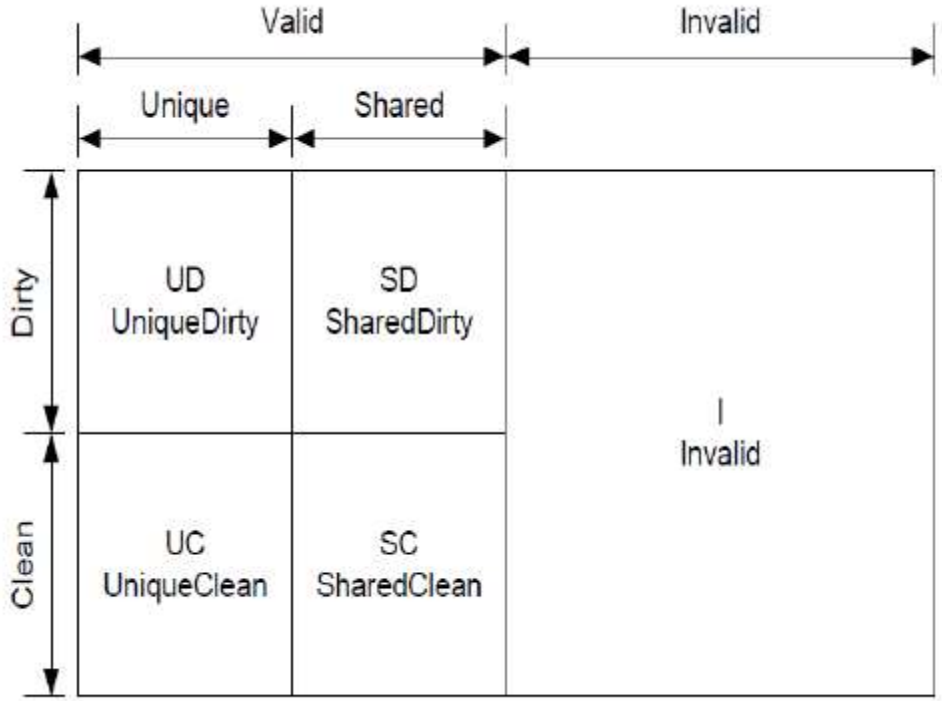
2.3.2 Defining a system-wide cache coherency model

2.3.2 Defining a system-wide cache coherency model

Assumed system-wide coherency model []

- Multiple masters with local caches
- Supporting 5 states, termed as AMBA 4 ACE Cache Coherency States, as follows:





http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

AMBA 4 ACE Cache Coherency States []

- Each cache line is either Valid or Invalid meaning it either contains cached data or not.
- Each Valid cache line can be in one of four states subject to two properties:

  a) a line is either Unique or Shared, depending on whether it has non-shared data or potentially shared data and

  b) a line is either Clean or Dirty, basically depending upon whether memory keeps the latest, most up-to-date copy of the data and the cache line is merely a copy of the memory, or it is Dirty meaning that the cache line holds the, latest up to date copy of data and it must be written back to memory later.

- There is however one exception to the above interpretation that is when multiple cache line share the line and the line is Dirty.

  In this case, all caches must contain the latest data value at all times, but only one may be in the SharedDirty state, the others being held in the SharedClean state.

- The SharedDirty state is thus used to indicate which cache has responsibility for writing the data back to memory.
- The SharedClean state is more accurately described as data is shared but there is no need to write it back to memory.

http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Relationship between ACE states and MOESI states []

- The ACE states can be mapped directly onto the MOESI cache coherency states.

  Nevertheless, ACE is designed to support components that may use different cache state models, including MESI, MOESI, MEI etc..

  Thus some components may not support all ACE transactions, e.g. the ARM Cortex-A15 internally makes use of the MESI states for providing cache coherency for the L1 data caches, meaning that the cache cannot be in the SharedDirty (Owned) state.

- To emphasize that ACE is not restricted to the MOESI cache state model, ACE does not use the familiar MOESI terminology.

http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

ACE cache line states and their alternative MOESI naming []

ARM ACE	MOESI		ACE meaning
UniqueDirty	M	Modified	Not shared, dirty, must be written back
SharedDirty	O	Owned	Shared, dirty, must be written back to memory
UniqueClean	E	Exclusive	Not shared, clean
SharedClean	S	Shared	Shared, no need to write back, may be clean or dirty
Invalid	I	Invalid	Invalid

http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Implementing different system wide and processor wide cache coherency protocols

As long as ARM makes use of the MOESI protocol to provide system wide cache coherency they prefer to employ the MESI protocol for providing processor wide cache coherency for most of their multicore processor.

This can easily be implemented as follows.

The cache lines incorporate state information for the 5-state MOESI protocol but for maintaining multicore coherency only four states will used such that the SharedDirty and SharedClean states are both considered as SharedDirty states meaning that they need to be written back to the memory.

2.3.4 Supporting two types of coherency, full coherency and I/O coherency

2.3.4 Supporting two types of coherency, full coherency and I/O coherency

Types of coherency []

Full coherency
(Two-way coherency)

Provided by the ACE interface.

The ACE interface is designed to provide full hardware coherency between CPU clusters that include caches.

With full coherency, any shared access to memory can 'snoop' into the other cluster's caches to see if the data is already there; if not, it is fetched from higher level of the memory system (L3 cache, if present or external main memory (DDR).

I/O coherency
One-way coherency)

Provided by the ACE-Lite interface.

The ACE-Lite interface is designed to provide hardware coherency for system masters that do not have caches of their own or have caches but do not cache sharable data.

Examples: DMA engines, network interfaces or GPUs.

http://community.arm.com/groups/processors/blog/2013/12/03/extended-system-coherency--part-1--cache-coherency-fundamentals

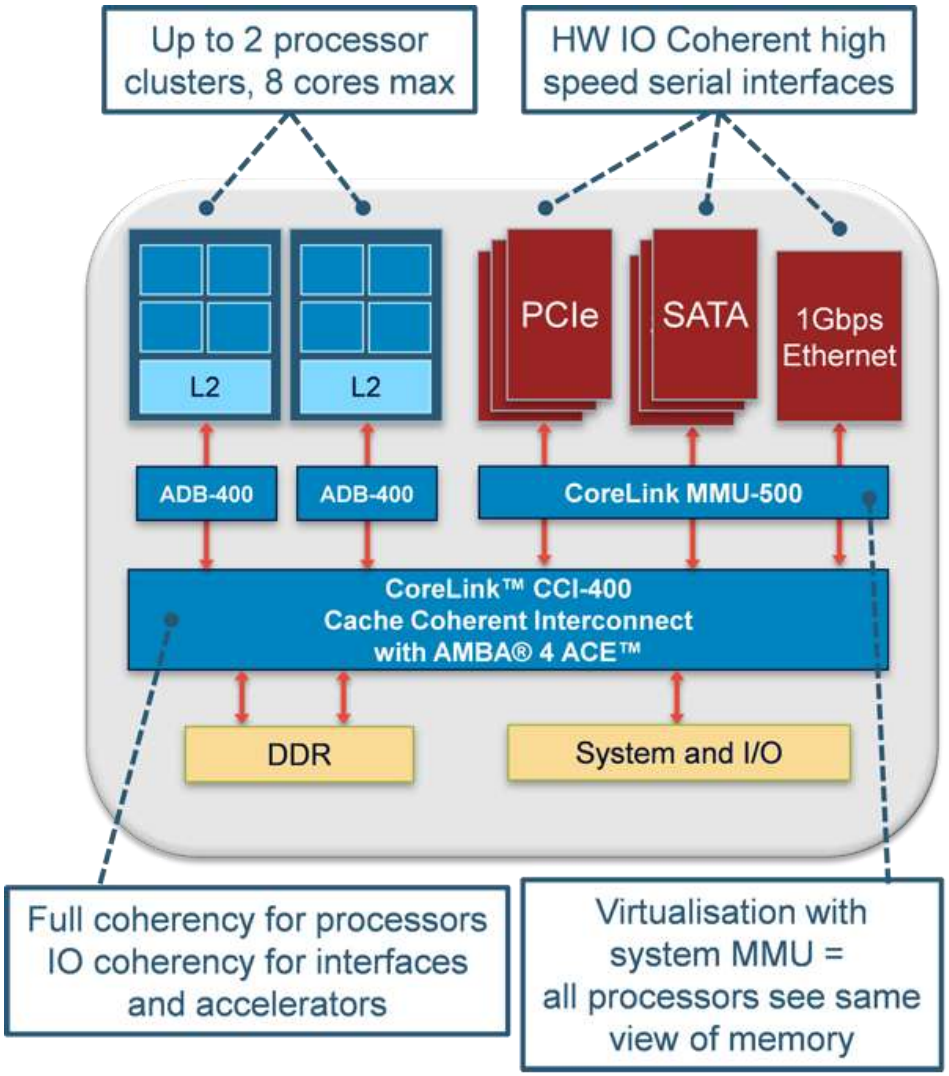
Remark

The AMBA 4 ACE---Lite interface has the additional signals on the existing channels but not the new snoop channels.

ACE---Lite masters can snoop ACE–compliant masters, but cannot themselves be snooped.

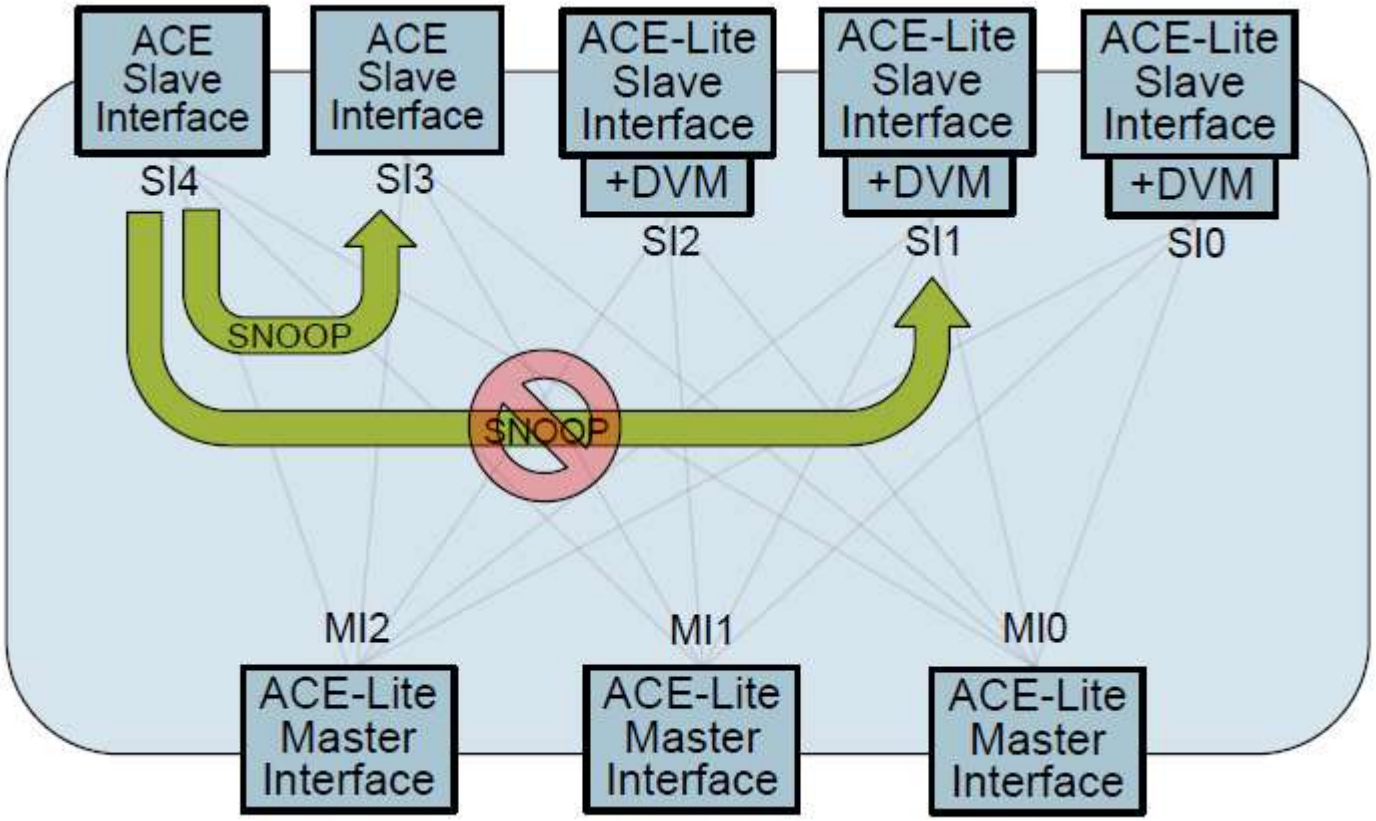
http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Example 1: Full coherency for processors, I/O coherency for interfaces and accelerators []



Example 2: Snooping transactions in case of full coherency []

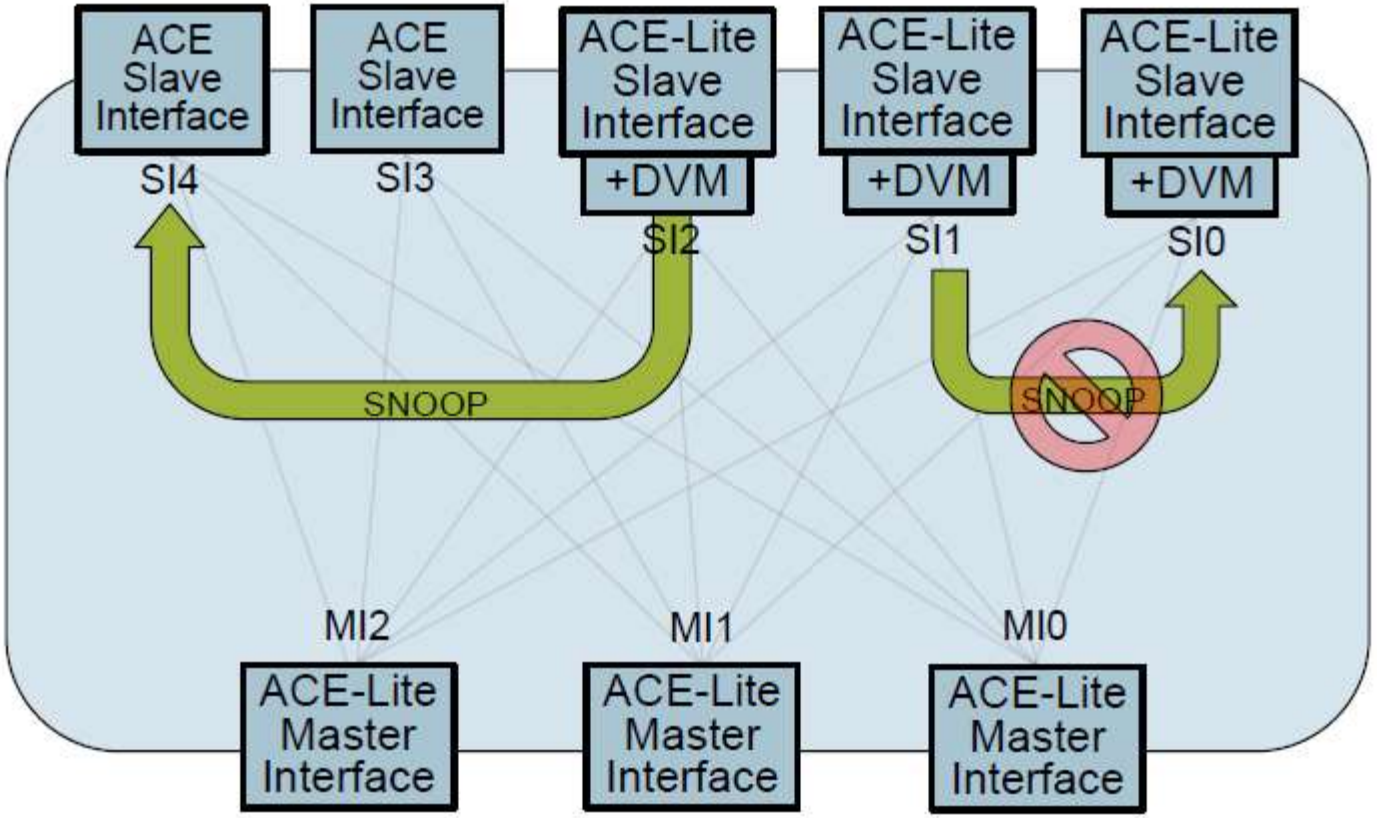
- **Transactions from ACE masters can snoop other ACE masters' caches**
- **Transactions from ACE masters cannot snoop ACE-Lite masters' caches**



http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

Example 3: Snooping transactions in case of I/O coherency []

- **Transactions from ACE-Lite masters can snoop other ACE masters' caches**
- **Transactions from ACE-Lite masters cannot snoop ACE-Lite masters' caches**



http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

2.3.5 Introducing the concept of domains

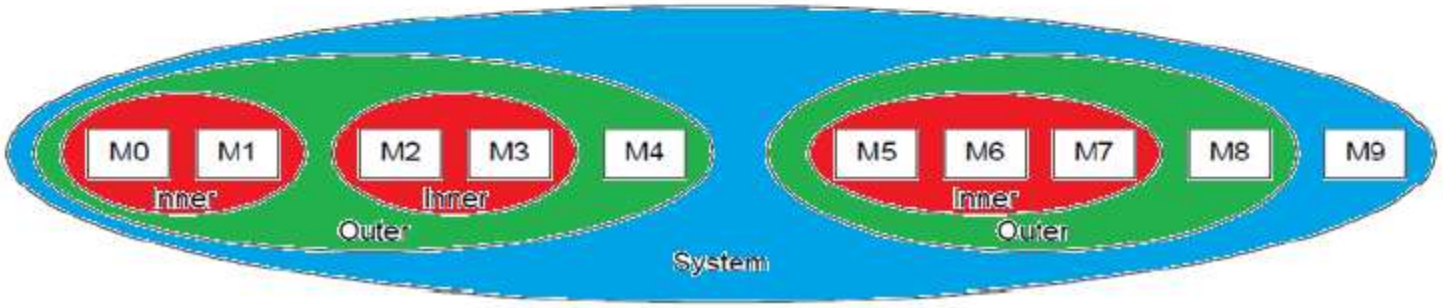
2.3.5 Introducing the concept of domains

In order to increase the efficiency of their system wide cache coherency solution ARM introduced also coherency domains along with their AMBA 4 ACE interface.

This allows course-grain filtering of snoops and directs cache maintenance and DVM requests in a system with partitioned memory.

Domain types

Name	Description
Non-shareable	Contains just a single master.
Inner shareable	The Inner domain can include additional masters.
Outer shareable	The Outer domain contains at least all masters in the Inner domain and may also include additional masters.
System shareable	The system domain includes all masters in the system.



http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

Note

- The inner domain shares both code and data
  (i.e. is running the same operating system, whereas
- The outer domain shares data but not code.

http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Domain control pins []

There are three domain control pins in the Cortex-A7/A15 or the Cortex-A57/A53 processors that direct how the snoop and maintenance requests to the system will be handled by the SCU, as follows in case of the Cortex-A7 processor:

The BROADCASTINNER pin

It controls issuing coherent transactions targeting the Inner Shareable domain on the coherent interconnect.

When asserted, the processor is considered to be part of an Inner Shareable domain that extends beyond the processor and Inner shareable snoop and maintenance operations are broadcast to other masters in this domain on the ACE or CHI interface.

When BROADCASTINNER is asserted, BROADCASTOUTER must also be asserted.

When BROADCASTINNER is deasserted, the processor does not issue DVM requests on the ACE AR channel or CHI TXREQ channel.

The BROADCASTOUTER pin

BROADCASTOUTER controls issuing coherent transactions targeting the Outer shareability domain on the coherent interconnect.

When asserted, the processor is considered to be part of the Outer Shareable domain and Outer shareable snoop and maintenance operations in this domain are broadcast externally on the ACE or CHI interface, else not.

The BROADCASTCACHEMAINT pin

It controls issuing L3 cache maintenance transactions, such as CleanShared and CleanInvalid, on the coherent interconnect.

When set to 1 this indicates to the processor that there are external downstream caches (L3 cache) and maintenance operations are broadcast externally, else not.

Permitted combinations of the domain control signals and supported configurations in the Cortex A7 processors []

Signal	Feature						
	AXI3 mode[a]	ACE non-coherent[b]		ACE outer coherent		ACE inner coherent	
		No L3 Cache	With L3 Cache	No L3 Cache	With L3 Cache	No L3 Cache	With L3 Cache
BROADCASTCACHEMAINT	0	0	1	0	1	0	1
BROADCASTOUTER	0	0	0	1	1	1	1
BROADCASTINNER	0	0	0	0	0	1	1

a. SYSBARDISABLE must be set to 1 in AXI3 mode.

b. ACE non-coherent is compatible with connecting to an ACE-Lite interconnect.

Note

Similar interpretations are true for the Cortex-A-15/A57/A53 processors.

A7 trm

2.3.6 Providing support for system-wide page tables

2.3.6 Providing support for system-wide page tables

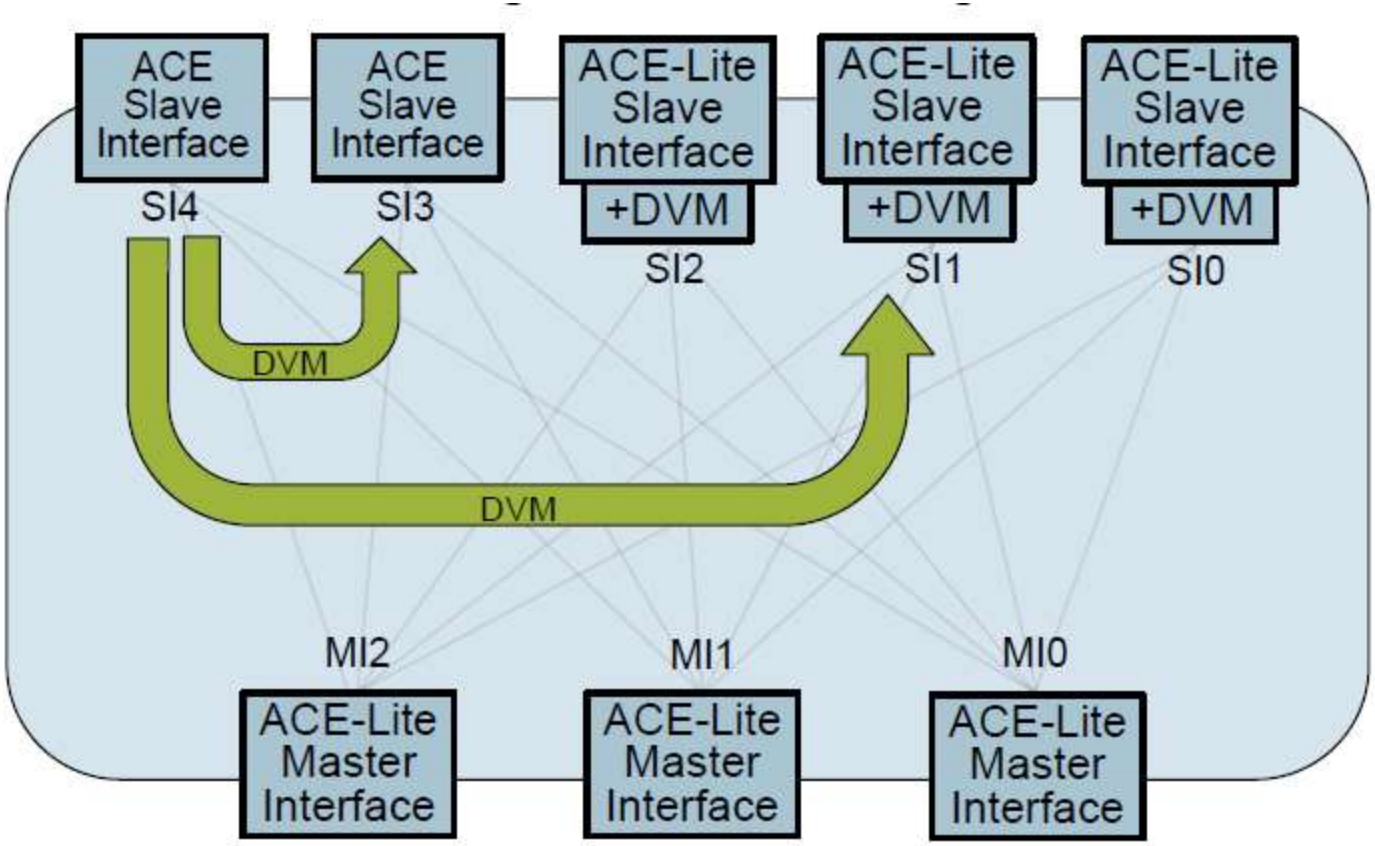
This precondition is termed also as Distributed Virtual Memory (DVM) support [].

- Multi-cluster CPU systems share a single set of MMU page tables in the memory.
- A TLB (Translation Look-Aside Buffer) is a cache of MMU page tables being in the memory.
- When one master updates page tables it needs to invalidate TLBs that may contain a stale copy of the MMU page table entry.
- DVM support of AMBA 4 (ACE) does facilitate this by providing broadcast invalidation messages.
- DVM messages are sent on the Read channel using the ARSNOOP signaling.

  A system MMU may take use of the TLB invalidation messages to ensure its entries are up-to-date.

Example for DVM messages []

- **ACE masters can accept DVM messages, per the ACE protocol**
- **CCI-400 DVM extensions allow ACE-Lite masters to receive DVMs messages**
- **ACE-Lite masters cannot generate DVM messages**



http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

Remarks on the address translation in ARM's Cortex-Ax processors []

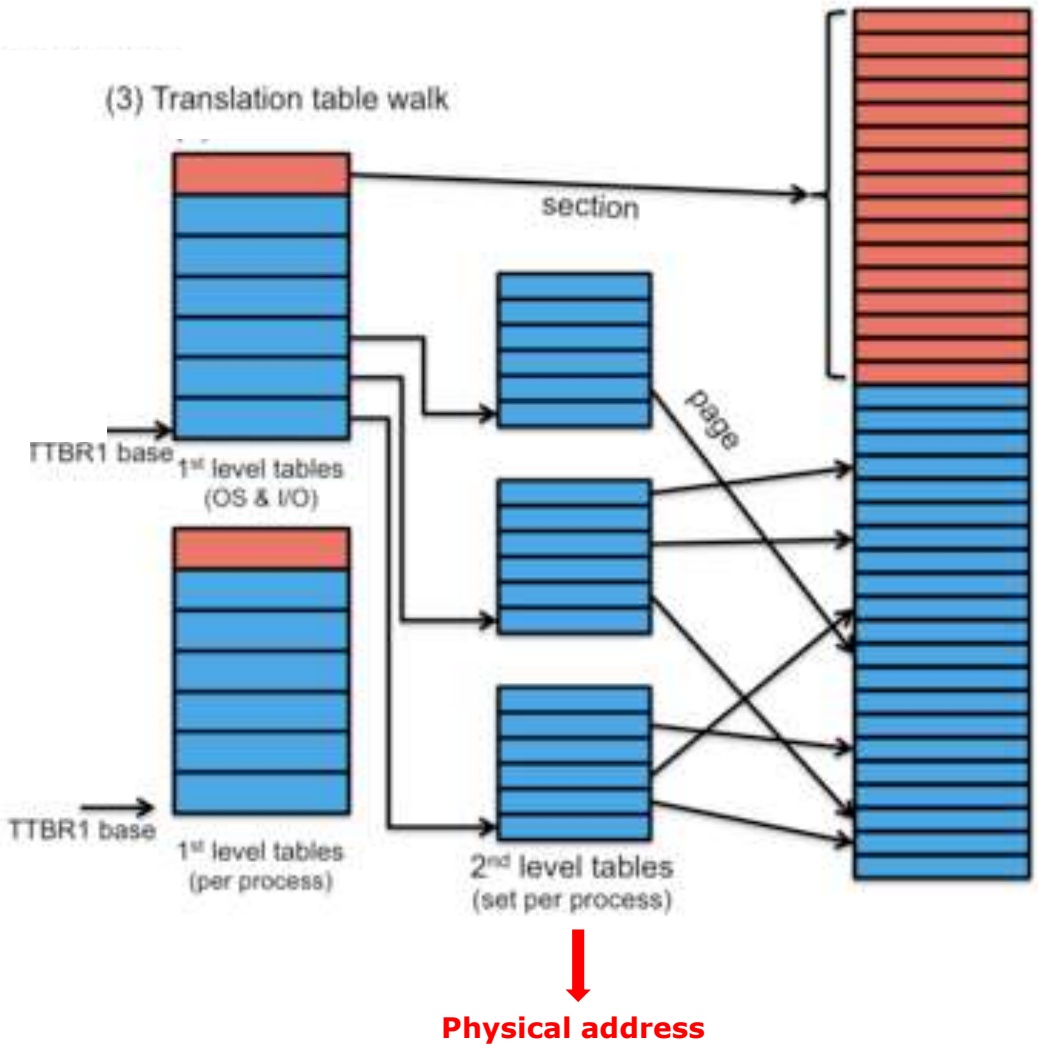
In order to give a glimpse into the address translation process we briefly summarize main steps of the address translation process as performed by the Cortex-A15 processor, nevertheless strongly simplified by neglecting issues including access permissions, protection etc.

Principle of the address translation

- The ARM Cortex-A series processors implement a virtual-to-physical address translation that is based on two-level page tables kept in the main memory.
- The address translation process performed by means of the page tables is usually termed as table walk.
- Here we do not want to go into details of the page tables or of the table walk process, rather we will only illustrate this in the next Figure.

(We note that a good overview of virtual to physical address translation is given e.g. in []).

Principle of virtual to physical address translation based on two-level page tables (based on [])



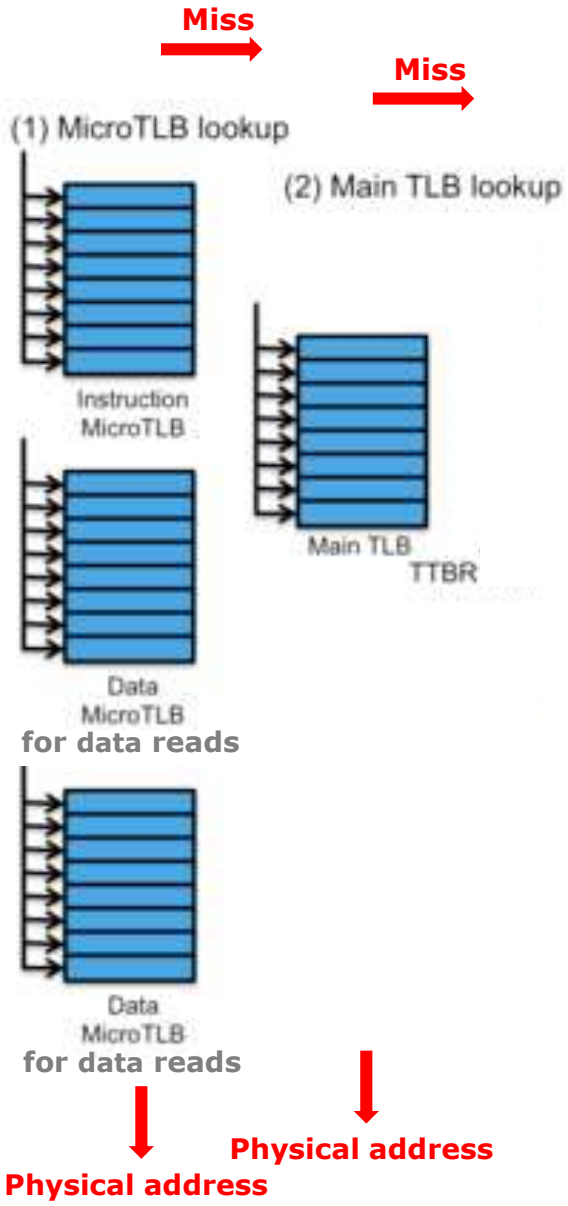
http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html

Use of TLBs to speed up the address translation process

Traditionally, the address translation process is speeded up by caching the latest translations.

The ARM Cortex-A series processors (and a few previous lines, such as the ARM9xx or ARM11xx lines) implement caching of the latest address translations by means of a two level TLB (Translation Look-aside Buffer) cache structure, consisting of Level 1 and Level 2 TLBs, as indicated in the next Figure.

Example of two-level TLB caching (actually that of the A15) (based on [])

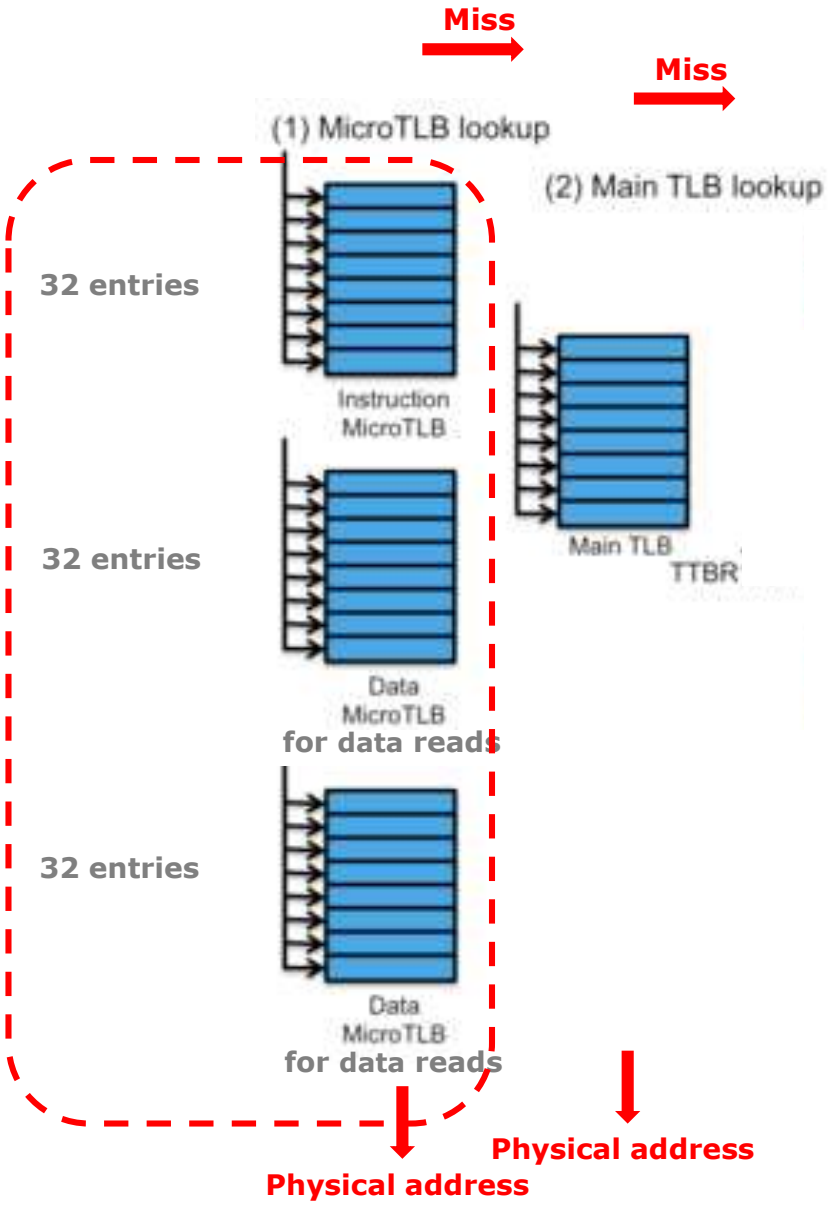


First level TLBs-1

- The first level of caching is performed by the L1 TLBs (dubbed also as Micro TLBs).
- Both the instruction and the data caches have their own Micro TLBs.
- Micro TLBs operate fully associative and perform a look up in a single cycle.

  As an example the Cortex-A15 has a 32 entry TLB for the L1 instruction cache and two separate L1 TLBs with 32-32 entries for data reads and writes, as indicated in the next Figure.

Example of a two-level TLB system (actually that of the A15) (based on [])



First level TLBs-2

- A hit in an L1 TLB returns the physical address to the associated L1 (instruction or data cache) for comparison.
- Misses in the L1 TLBs are handled by a unified L2 TLB.

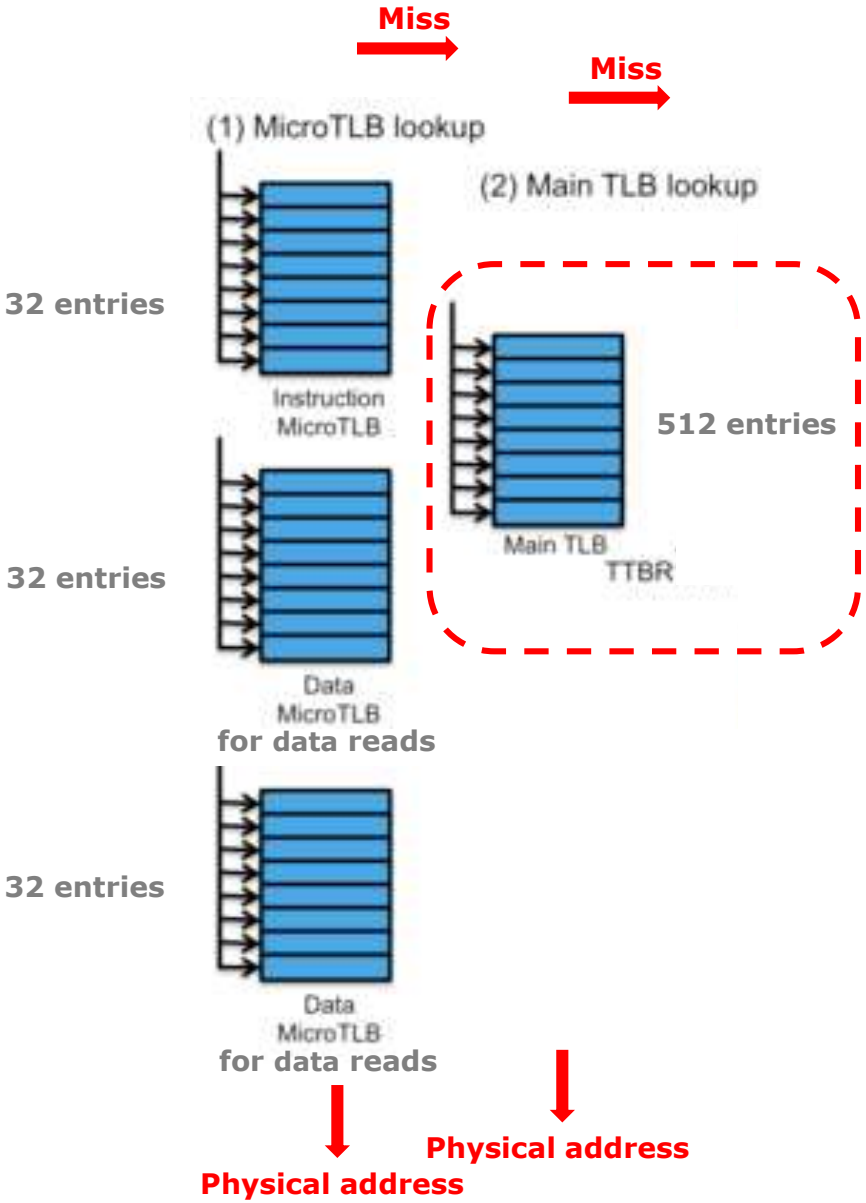
  (Unified means that the same TLB is used for both instructions and data).

http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html

The second level TLB-1

- The second level TLB is a 512 entry 4-way set associative cache in the Cortex-A15 (called the main TLB), as shown next.

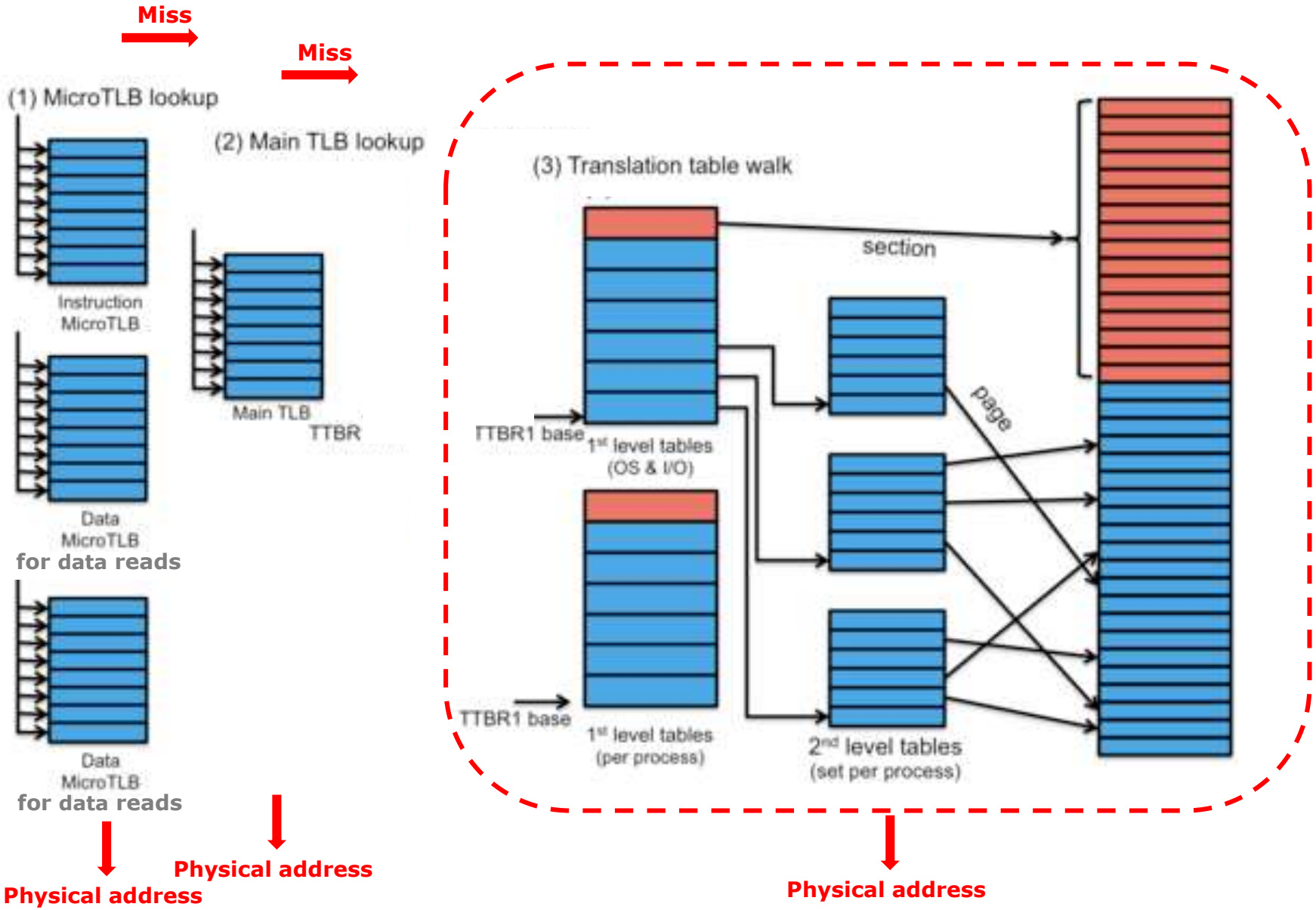
Example of a two-level TLB system (actually that of the A15) (based on [])



The second level TLB-2

- Accesses to the L2 TLB take a number of cycles.
- A hit in the L2 TLB returns the physical address to the associated L1 cache for comparison.
- A miss in the L2 TLB invokes a hardware translation table walk, as indicated in the next Figure.

Principle of the virtual to physical address translation in the A15 (based on [])



http://www.cs.rutgers.edu/~pxk/416/notes/09a-paging.html

The hardware table-walk

- It retrieves the address translation information from the translation tables in the physical memory, as show in the Figure, but this process is not detailed here.
- Once retrieved the translation information is placed into the associated TLBs, possible by overwriting the existing values.
- The entry to be overwritten is chosen typically in a round robin fashion.

Note

In an operating system, like Linux each process has its own set of tables, and the OS switches to the right table whenever there is a process (context) switch by rewriting the Translation Table Base Register (TTBR) [].

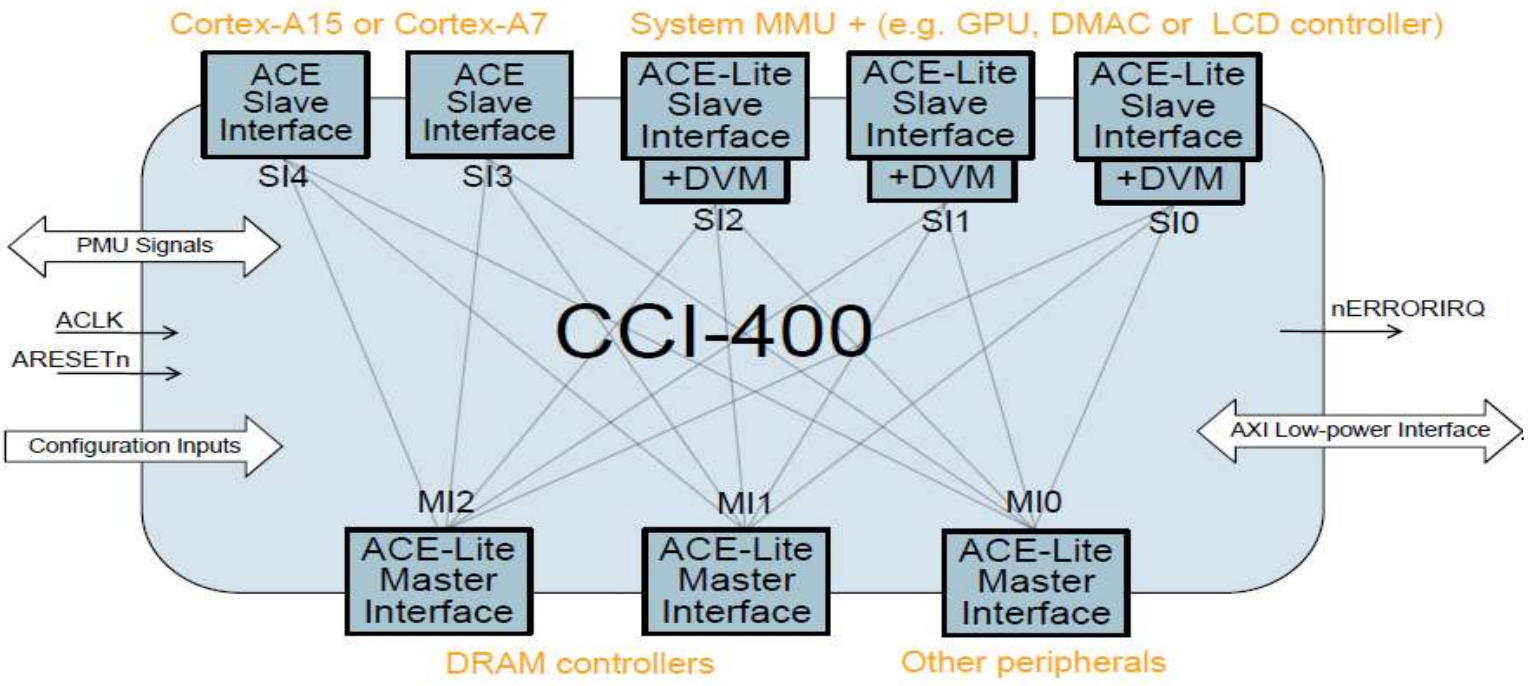
http://www.slideshare.net/prabindh/enabling-two-level-translation-tables-in-armv7-mmu

2.3.7 Providing cache coherent interconnects

2.3.7.1 Overview of ARM’s cache coherent interconnects

The role of an interconnect in the system architecture

An interconnect provides the needed interconnections between the major system components, such as cores, accelerators, memory, I/O etc, like in the Figure below [].



Interface ports receiving data requests e.g. from processor cores, the GPU, DMAs or the LCD, are called Slaves whereas those initiating data requests e.g. to the memory or other peripherals are designated as Masters, as indicated in the Figure above.

http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

Interconnect topologies of processor platforms

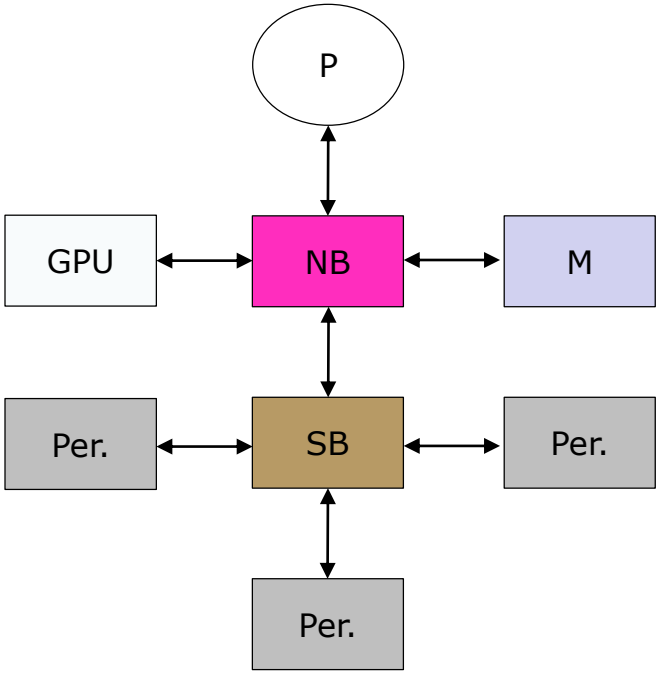
Interconnect topologies of processor platforms from ARM

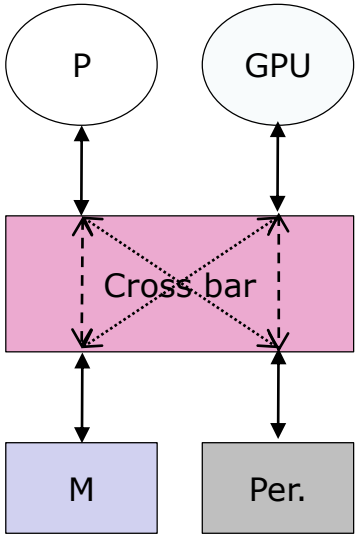
Chipset based interconnect topology
[before 2004]

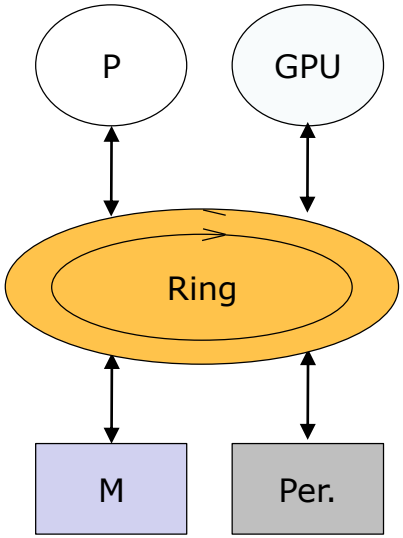
Crossbar-based interconnect topology
[from 2004 on]

Ringbus-based interconnect topology
[from 2012 on]

Examples

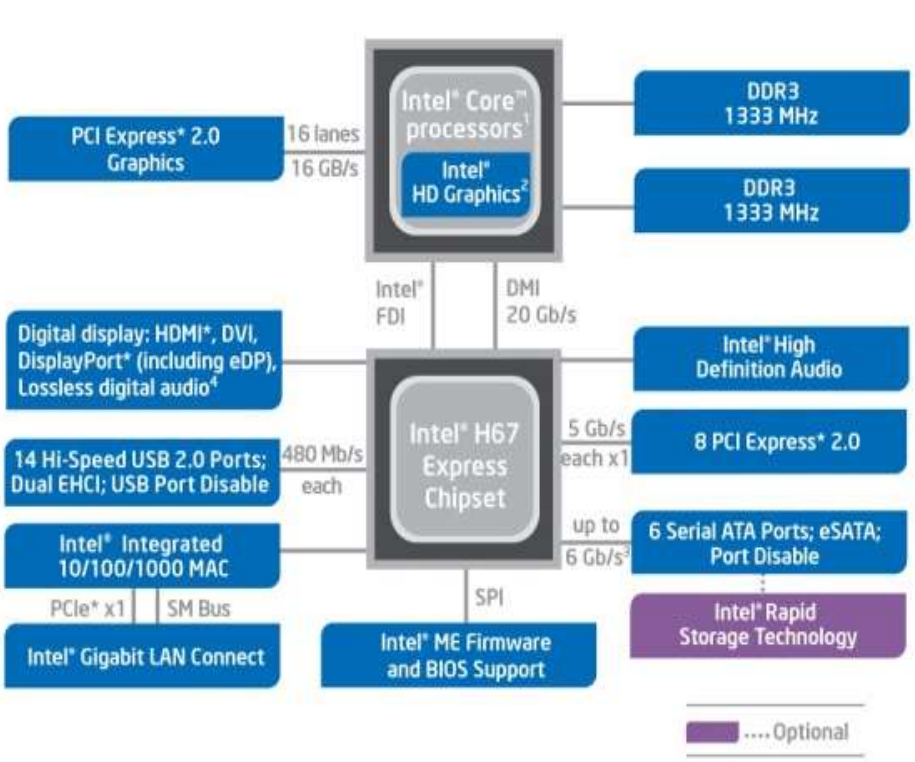


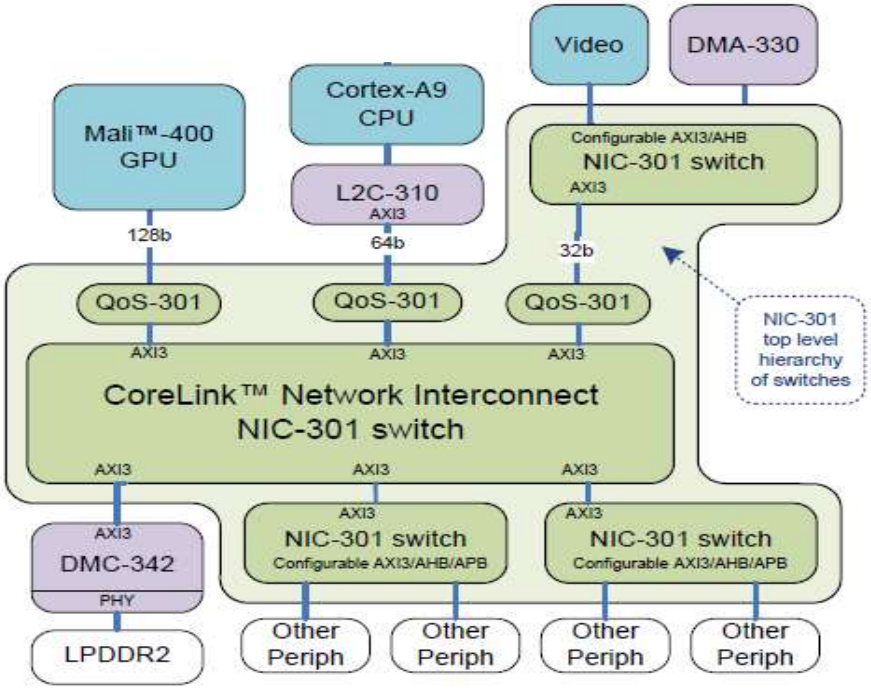




4.3 Example for a Sandy Bridge based desktop platform with the H67 chipset (1)

Contrasting chipset and interconnect based platform topologies [], [] 102]





On the left side there is a double star like chipset based platform topology (actually a Sandy Bridge based desktop platform with the H67 chipset from Intel, whereas on the right a crossbar interconnect based platform topology (actually an NIC-301 inerconnect with a Cortex-A9 processor from ARM.

2.3.7.1 Overview of ARM's cache coherent interconnects

Overview of ARM's recent on-die interconnect solutions

ARM's recent on-die interconnect solutions

ARM's non-cache coherent interconnects

Cache coherency is maintained by software.

Induces higher coherency traffic.

Is less effective in terms of performance and power consumption.

Examples

NIC-301 (~ 2006)
NIC-400 (2011)

ARM's cache coherent interconnect

Cache coherency is maintained by hardware.

Induces less coherency traffic.

Is more effective in terms of performance and power consumption

CCI-400 (2010)
CCN-504 (2012)
CCN-508 (2013)

NIC: Network Interconnect

CCI: Cache Coherent Interconnect
CCN: Cache Coherent Network

2.3.7.1 Overview of ARM's recent on-die interconnect solutions

ARM's recent on-die interconnect solutions

ARM's non-cache coherent interconnects

Cache coherency is maintained by software.

Induces higher coherency traffic.

Is less effective in terms of performance and power consumption.

ARM's cache coherent interconnect

Cache coherency is maintained by hardware.

Induces less coherency traffic.

Is more effective in terms of performance and power consumption

ARM's cache coherent interconnects for mobiles

They are crossbar based

Examples

CCI-400 (2010)
CCI-500 (2014)
CCI-550 (2015)

ARM's cache coherent interconnects for enterprise computing

They are ring bus based

CCN-502 (2014)
CCN-504 (2012)
CCN-508 (2013)
CCN-512 (2014)

PL-300 (2004)
NIC-301 (2006)
NIC-400 (2011)

NIC: Network Interconnect

CCI: Cache Coherent Interconnect
CCN: Cache Coherent Network

2.3.7.1 ARM-s non cache coherent interconnects

ARM’s non-cache coherent interconnects

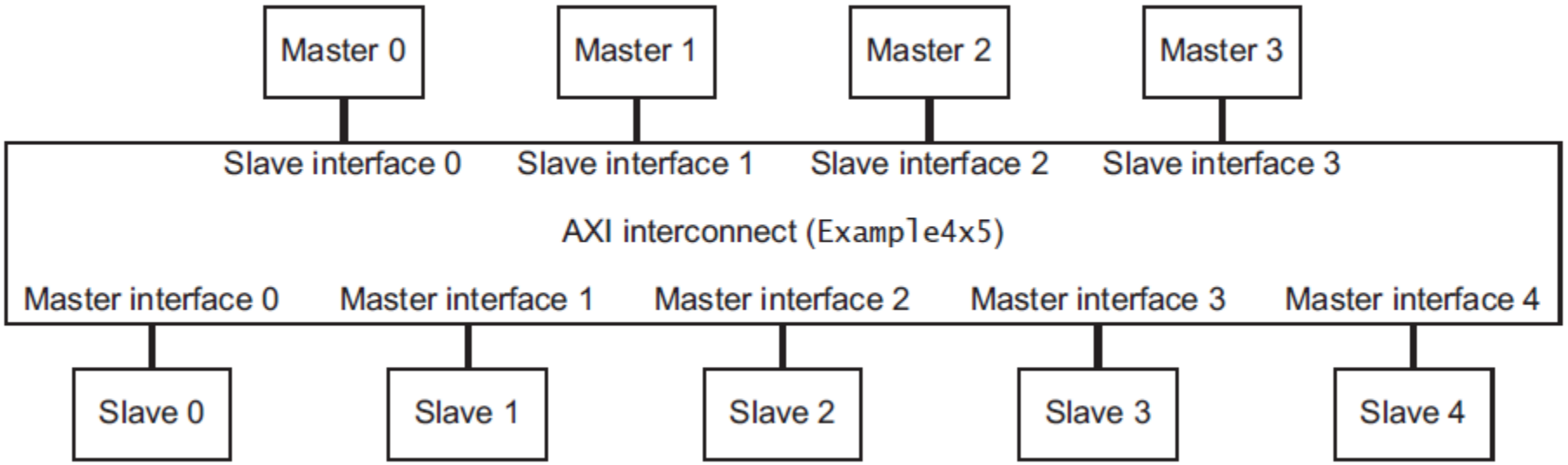
- They are crossbar-based.
- The underlying bus system does not include signals required to maintain cache coherency.
- These are ARM's first interconnects, as follows:
  - *PL-300 (2004)*
  - *NIC-301 (2006)*
  - *NIC-400 (2011)*

Main features of ARM's non cache coherent interconnects

Main features	PL-300	NIC-301	NIC-400
Date of introduction	06/2004	05/2006	05/2011
Supported processor models (Cortex-Ax MPCore)	ARM11	A8/A9/A5	A15/A7
No. of slave ports	Configurable	Configurable (1-128)	Configurable (1-64)
Type of slave ports	AXI3	AXI3/AHB-Lite	AXI3/AXI4/AHB-Lite
Width of slave ports	32/64-bit	32/64/128/256-bit	32/64/128/256-bit
No. of master ports	Configurable	Configurable (1-64)	Configurable (1-64)
Type of master ports	AX3	AXI3/AHB-Lite/APB2/3	AXI3/AXI4/AHB-Lite/ APB2/3/4
Width of master ports	32/64-bit	32/64/128/256-bit (APB only 32-bit)	32/64/128/256-bit (APB only 32-bit)
Integrated snoop filter	No	No	No
Interconnect topology	Swithes	Switches	Switches
Fitting memory controllers	PL-340	PL-341/DMC-340/1/2	DMC-400

https://www.arm.com/products/system-ip/interconnect/corelink-cci-550-cache-coherent-interconnect.php

High level block diagram of ARM's first (AXI3-based) PL-300 interconnect []



PrimeCell® AXI Configurable Interconnect (PL300) Revision: r0p1
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0354b/DDI0354.pdf

Internal structure of a NIC-400 Network Interconnect []

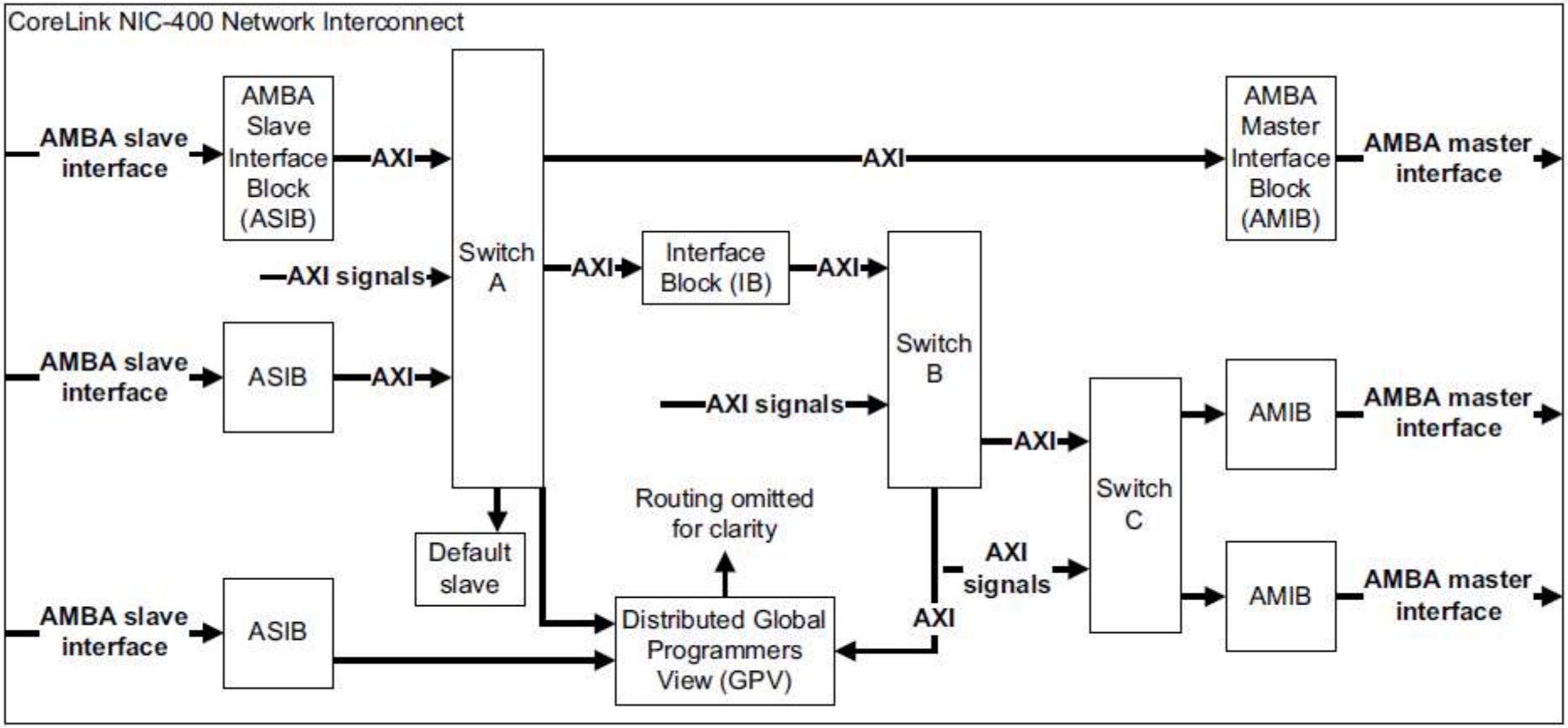
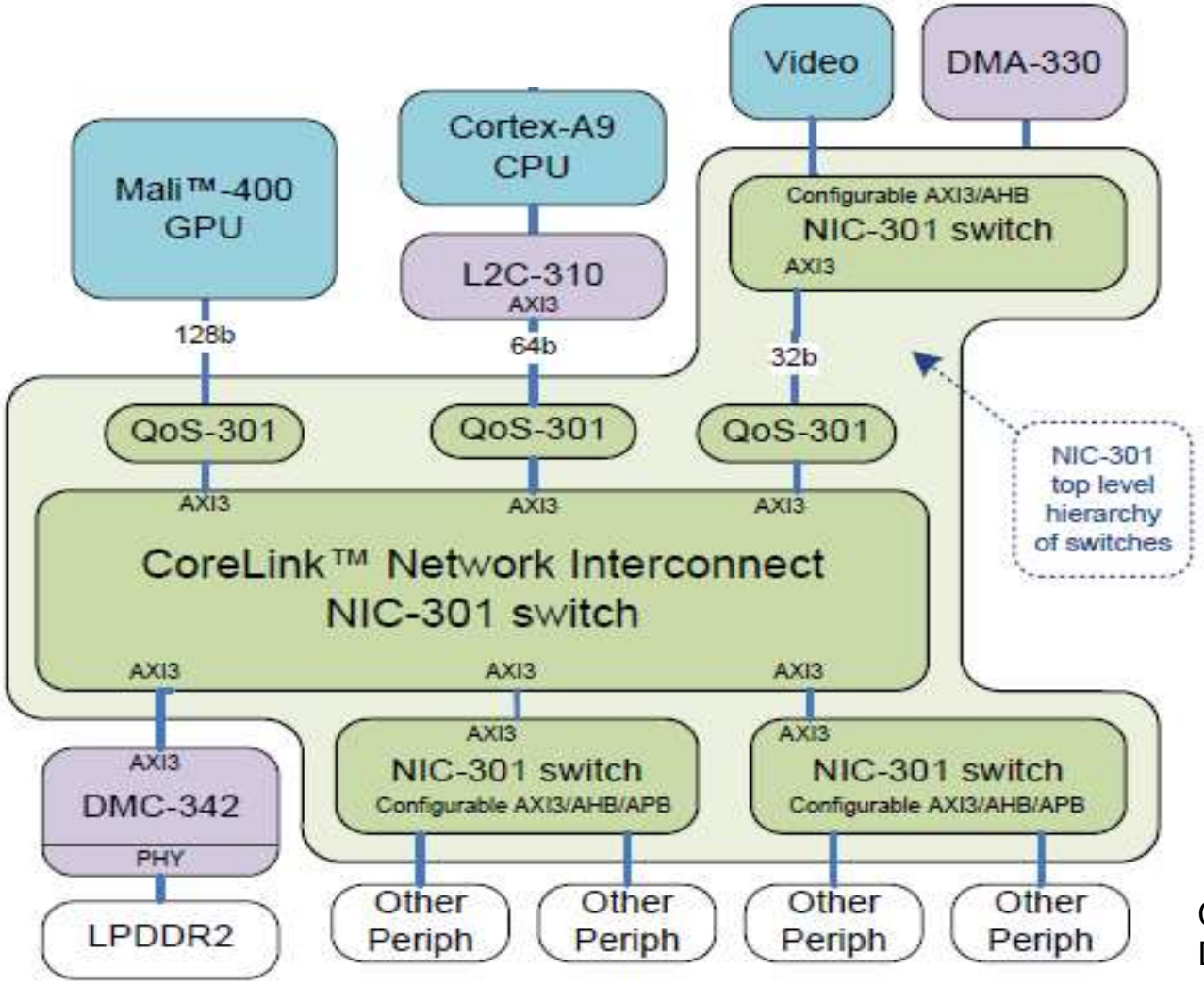


Figure 1-1 CoreLink NIC-400 Network Interconnect top-level block diagram

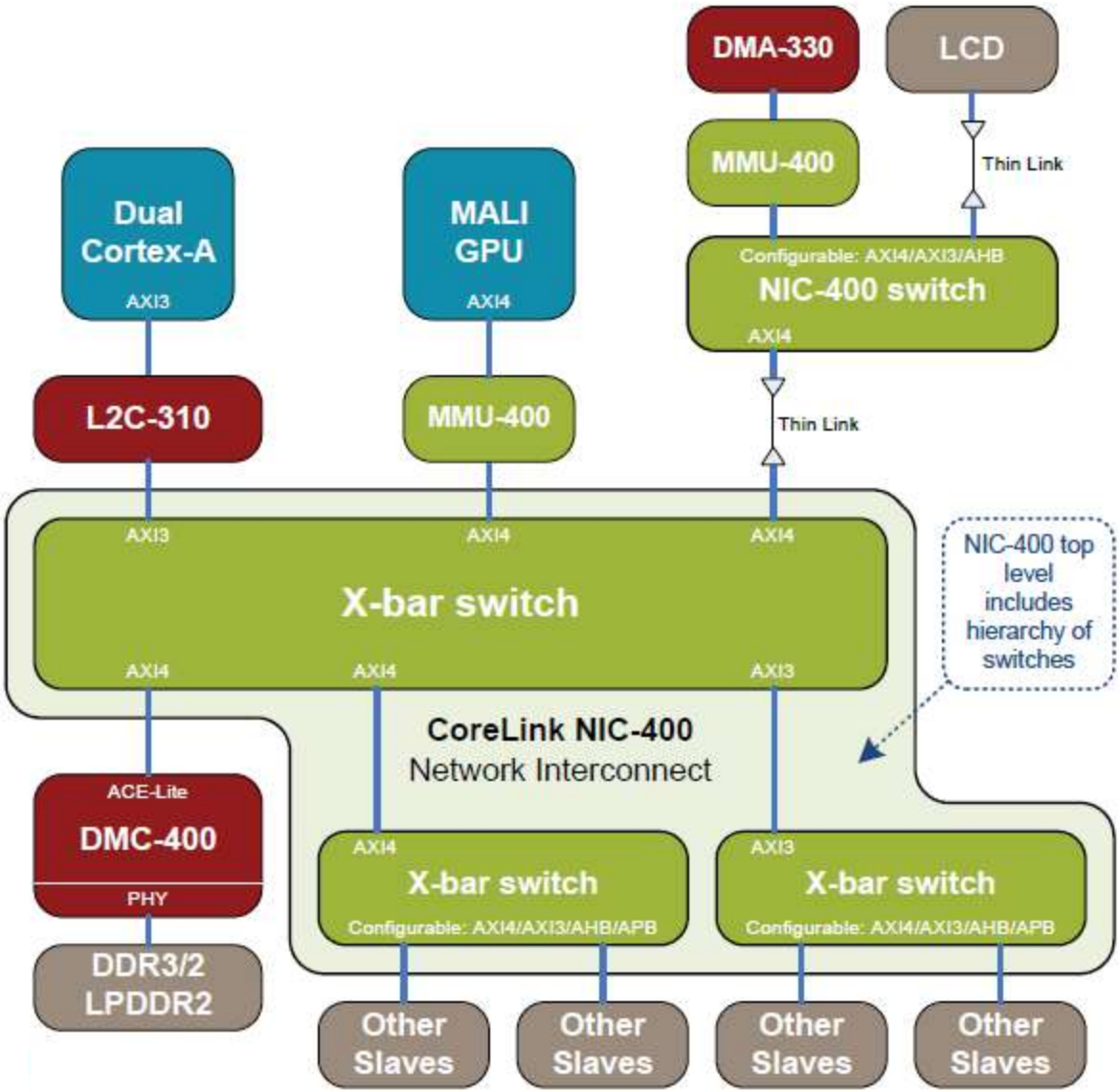
Example 1: NIC-301 based platform with a Cortex-A9 processor []



QoS: Quality of Service
DMC: Dynamic Memory Controller

http://www.arm.com/files/pdf/AT_-_Building_High_Performance_Power_Efficient_Cortex_and_Mali_systems_with_ARM_CoreLink.pdf

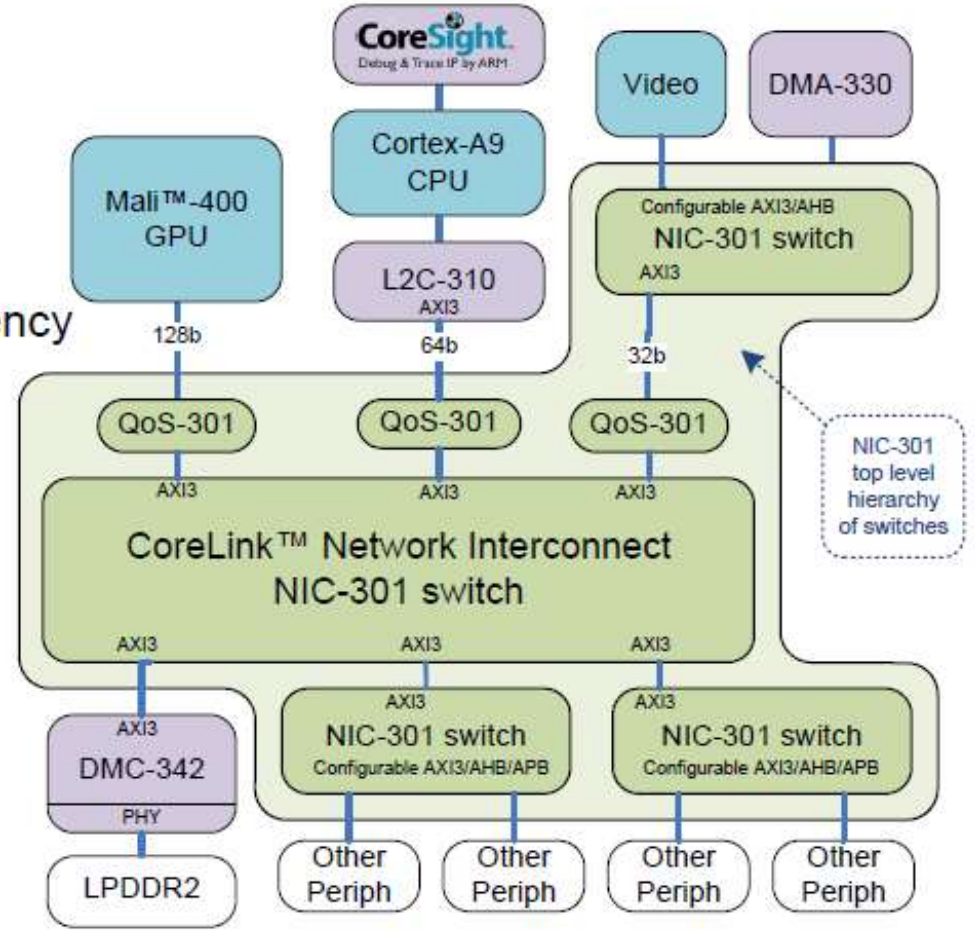
Example 2: NIC-400 based platform with a Cortex-A7 processor []



L2C: L2 cache controller
DMA: DMA cotroller
MMU: Memory Management Unit
DMC: Dynamic Memory Controller

Example 1: NIC-301 based platform with a Cortex-A9 processor []

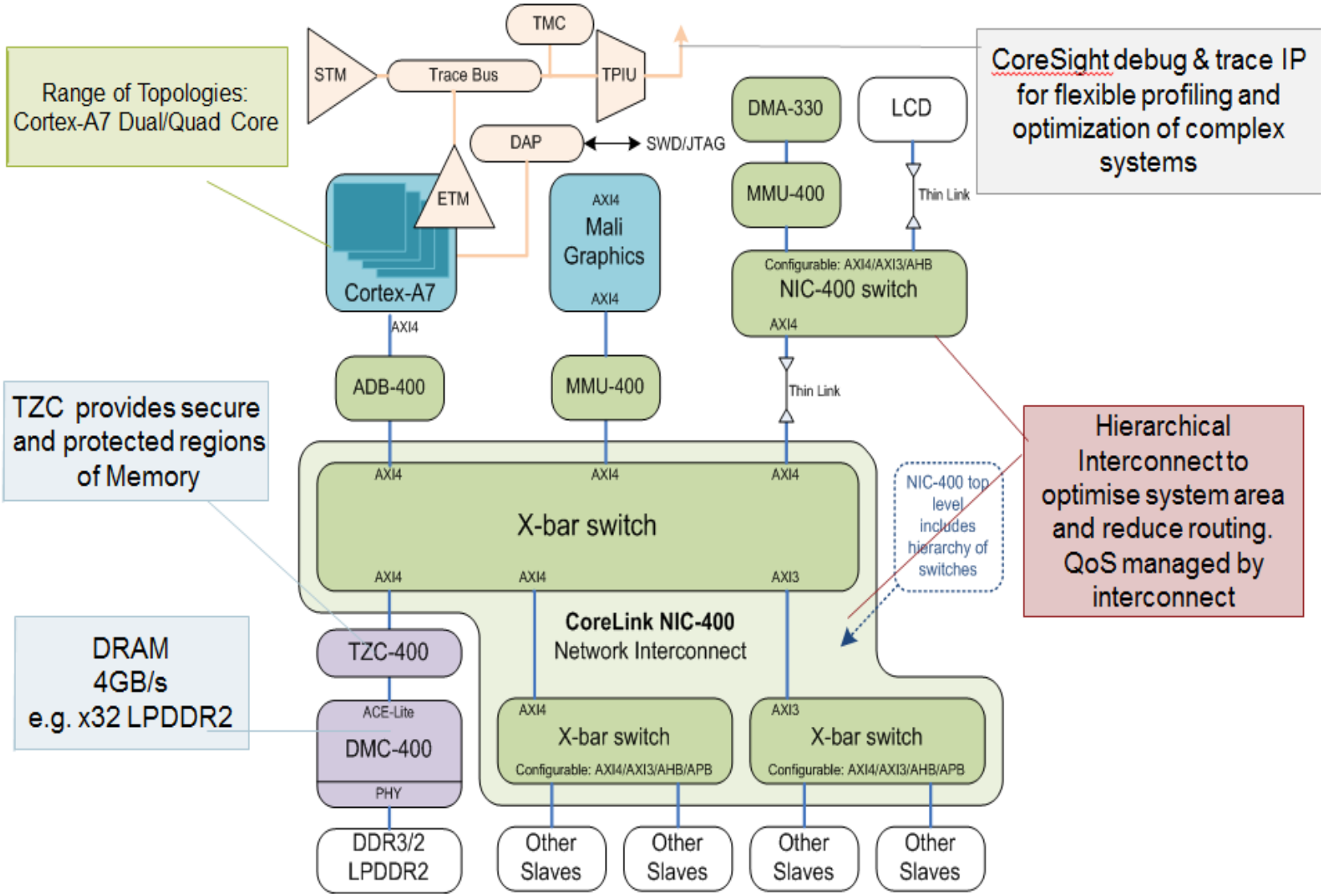
- NIC-301 Network Interconnect
  - Hierarchical design
  - Advanced Quality of Service (QoS) for performance and latency
- Level 2 Cache Controller
  - Increase CPU performance
  - Reduce external memory accesses
- Dynamic Memory Controllers
  - LPDDR2, DDR2
  - LPDDR, DDR, NVM
- Programmable DMA Controller
  - Off load the CPU
  - Multi-channel



CoreLink™
Intelligent System IP by ARM

http://www.arm.com/files/pdf/AT_-_Building_High_Performance_Power_Efficient_Cortex_and_Mali_systems_with_ARM_CoreLink.pdf

Example 2: Cortex-A7 MP/NIC-400 based entry-level smartphone platform []



http://www.arm.com/products/processors/cortex-a/cortex-a7.php

NIC-400 configurable interconnect

- High performance **and** rest of SoC interconnect
- AXI4, AXI3, AHB and APB4

Thin Links to reduce routing

- Between switches

End-to-end Quality of Service with virtual networks and regulation

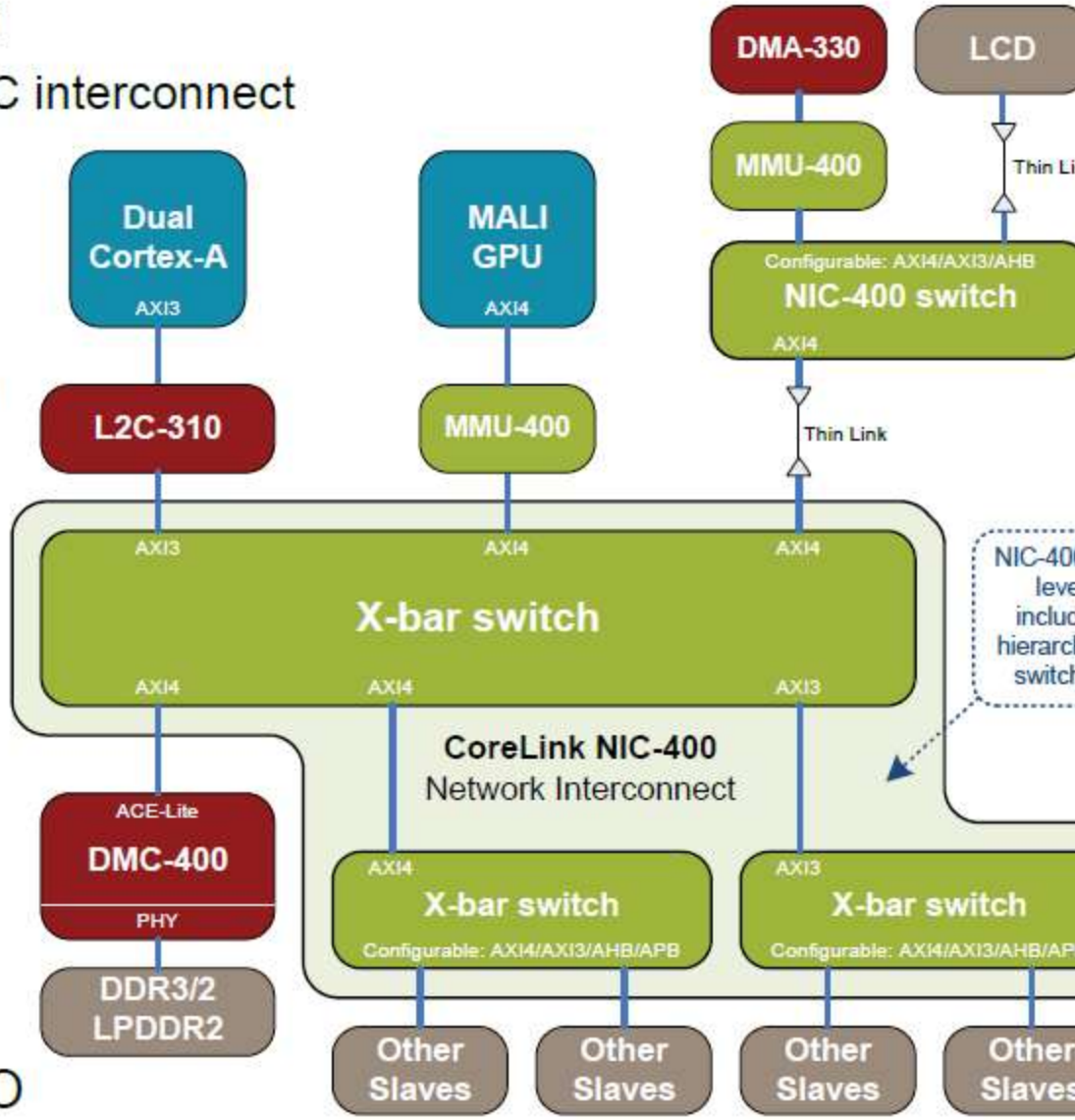
- Separates low latency, real-time and high bandwidth masters
- Prevents head-of-line and cross-stream blocking

DMC-400 high efficiency, multi-channel memory controller

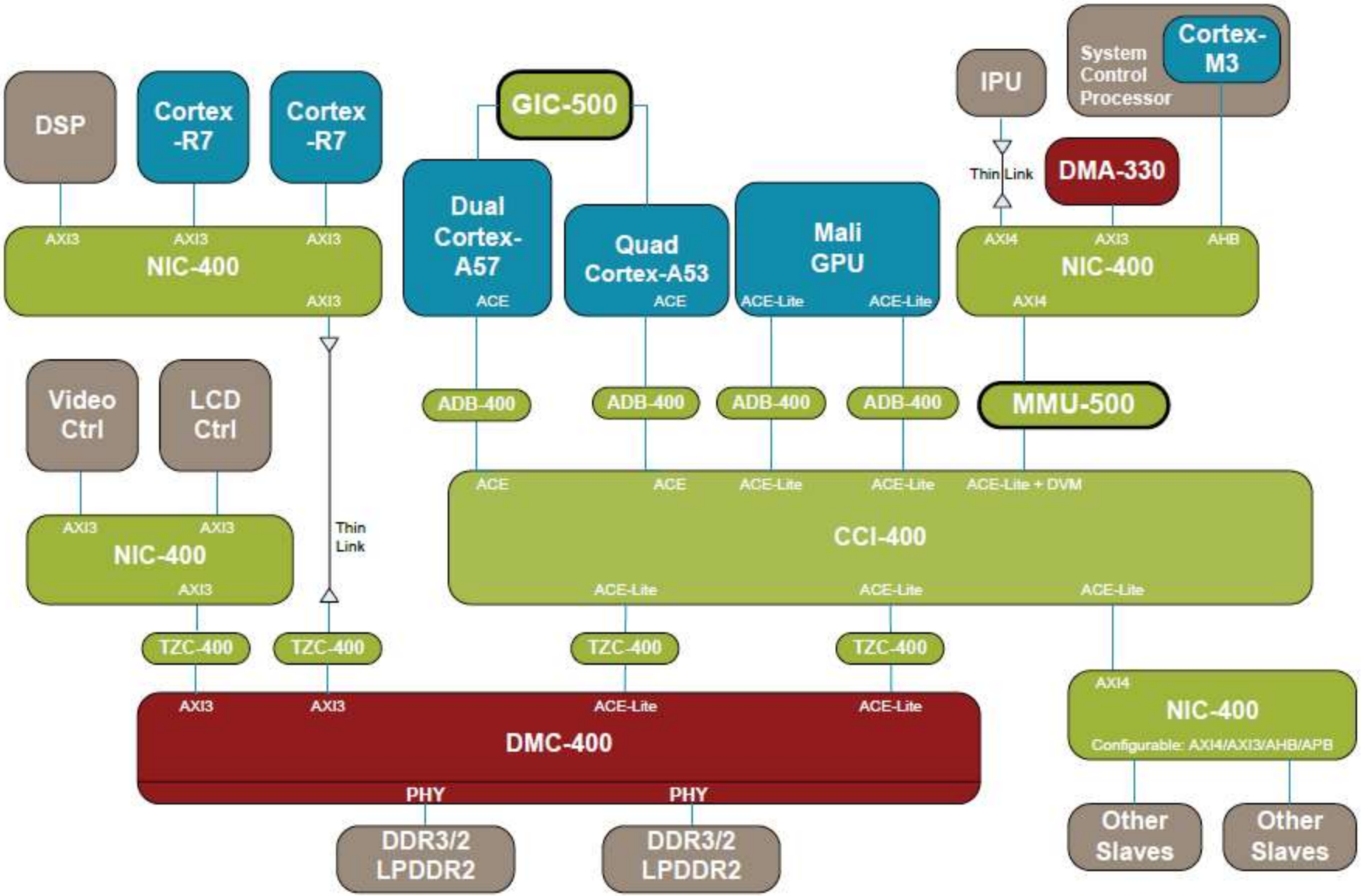
- AXI3 and AXI4 supported
- Unified QoS mechanisms

MMU-400 assists hypervisor for I/O

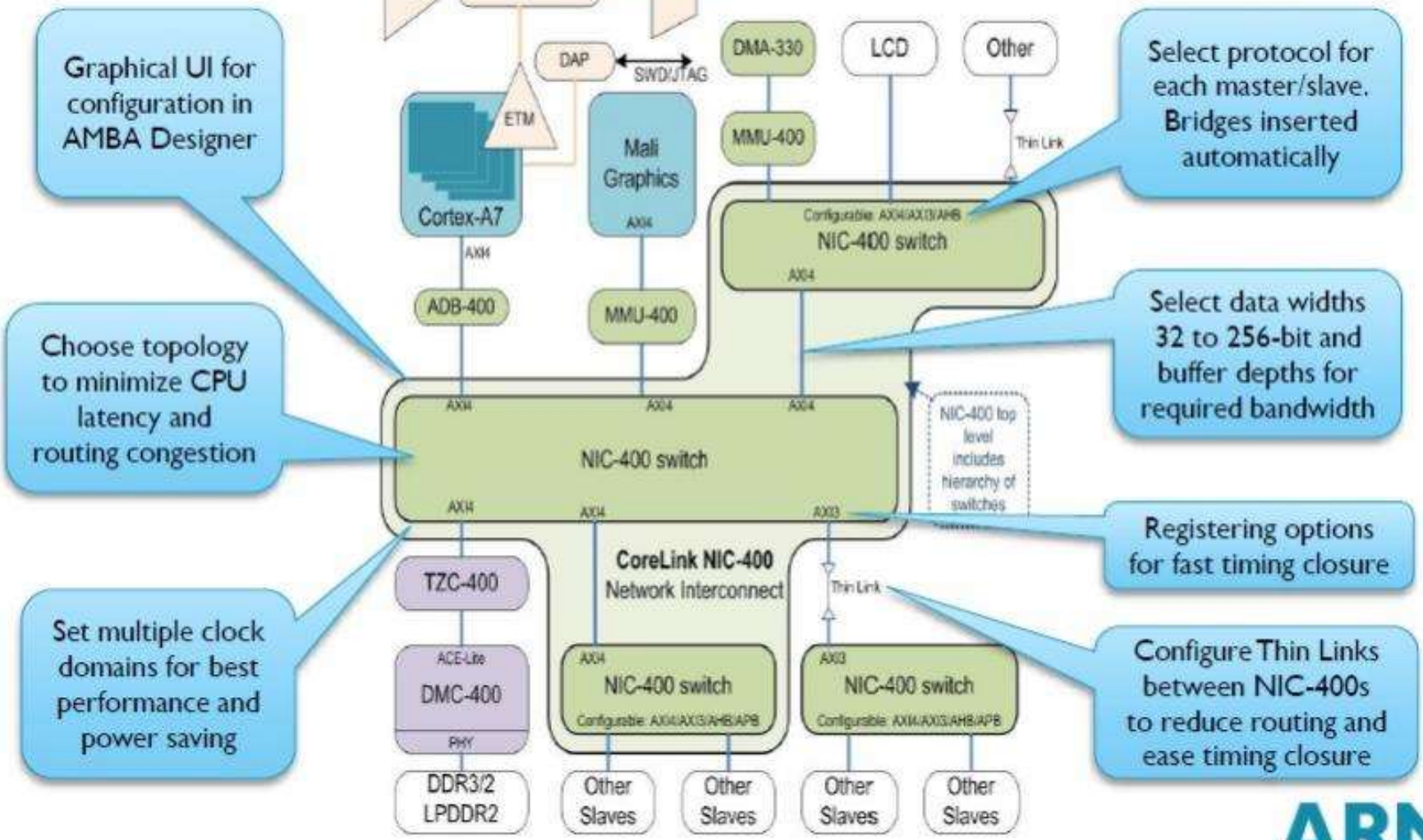
- Stage 2 address translation



ARM CoreLink 400 & 500 Series System IP 12/2012



ARM CoreLink 400 & 500 Series System IP 12/2012


Graphical UI for configuration in AMBA Designer
Choose topology to minimize CPU latency and routing congestion
Set multiple clock domains for best performance and power saving
Select protocol for each master/slave. Bridges inserted automatically
Select data widths 32 to 256-bit and buffer depths for required bandwidth
Registering options for fast timing closure
Configure Thin Links between NIC-400s to reduce routing and ease timing closure
DAP
SWD/JTAG
DMA-330
LCD
Other
ETM
Mali Graphics
MMU-400
Thin Link
Cortex-A7
Configurable AXI4/AXI3/AHB
NIC-400 switch
AXI4
ADB-400
MMU-400
NIC-400 top level includes hierarchy of switches
CoreLink NIC-400
Network Interconnect
TZC-400
ACE-Lite
DMC-400
PHY
DDR3/2 LPDDR2
AXI3
Configurable AXI4/AXI3/AHB/APB
Other Slaves
ARM

2.3.7.1 ARM-s non cache coherent interconnects used for mobiles

Overview of ARM's cache coherent interconnects

ARM's cache coherent interconnects

ARM's cache coherent interconnects used for mobiles

No integrated L3 cache

Does not support Cortex-A50 line processors

2 ACE-128 bit master ports for CPU clusters

The interconnect fabric is implemented as a crossbar

Examples

CCI-400 (2010)
CCI-500 (2014)
CCI-550 (2015)

(See Section 2.3.7.2)

ARM's cache coherent interconnects used for enterprise computing

Integrated L3 cache

Supports Cortex-A50 line processors

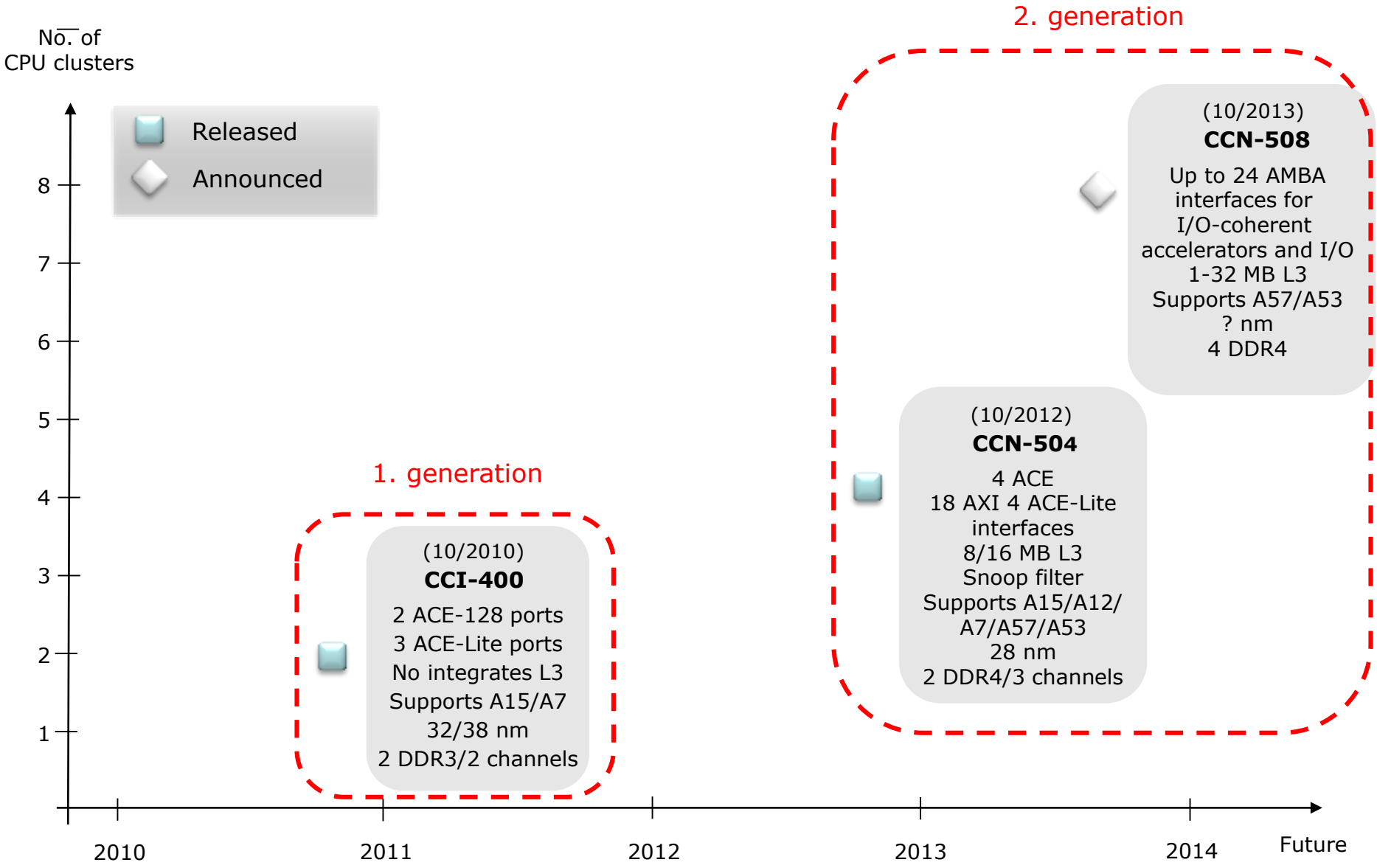
4/8/12 ACE-128/CHI or 8 CHI master ports for CPU clusters

The interconnect fabric is implemented as a ring bus, termed internally as Dickens

CCN-502 (2014)
CCN-504 (2012)
CCN-508 (2013)
CCN-512 (2014)

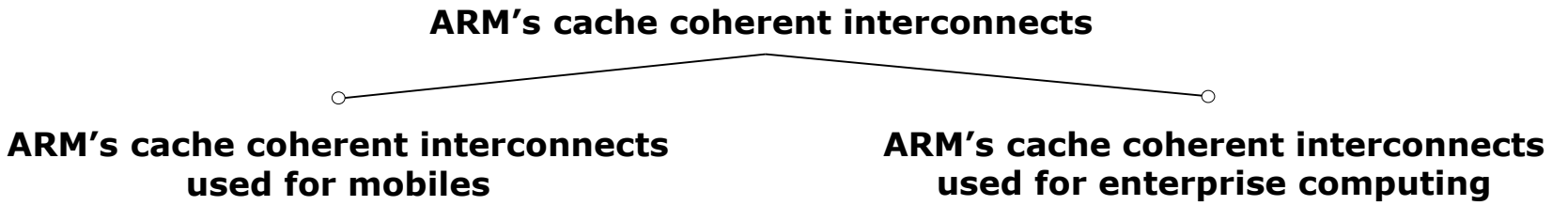
(See Section 2.3.7.3)

Overview of ARM's cache coherent interconnects



2.3.7.2 ARM’s cache coherent interconnects for mobiles

2.3.7.2 ARM’s cache coherent interconnects for mobiles



2.3.7.2 ARM's cache coherent interconnects for mobiles

- Until writing of this slides ARM announced three cache coherent interconnects for mobiles, as follows:
  - *CCI-400 (2010)*
  - *CCI-500 (2014)*
  - *CCI-550 (2015)*

- The CCI-400 is the first ARM interconnect to support cache coherency while
- using the AMBA 4 ACE interconnect bus.
  (AMBA Coherency Extensions).
- The CCI-400 Cache Coherent Interconnect is part of the CoreLink 400 System, as indicated in the next Table.

CoreLink 400 System components (based on [])

Name	Product	Headline features
NIC-400	Network interconnect	Non-coherent hierarchical interconnect
CCI-400	Cache Coherent Interconnect	Dual clusters of Cortex – A15/A17/A12/A7 2 128-bit ACE-Lite master ports 3 128-bit ACE-lite slave ports
DMC-400	Dynamic Memory Controller	2 channel (2x72-bit) LPDDR2/DDR3 memory controller
MMU-400	System Memory Management	Up to 40 bit virtual addresses ARMv7 virtualizations extensions compliant
GIC-400	Generic Interrupt Controller	Share interrupts across clusters, ARMv7 virtualization extensions compliant
ADB-400	AMBA Domain Bridge	It can optionally be used between components to integrate multiple power domains or clock domains for implementing DVFS
TZC-400	TrustZone Address Space Controller	Prevents illegal access to protected memory regions

CoreLink 500 System components for mobiles (based on [])

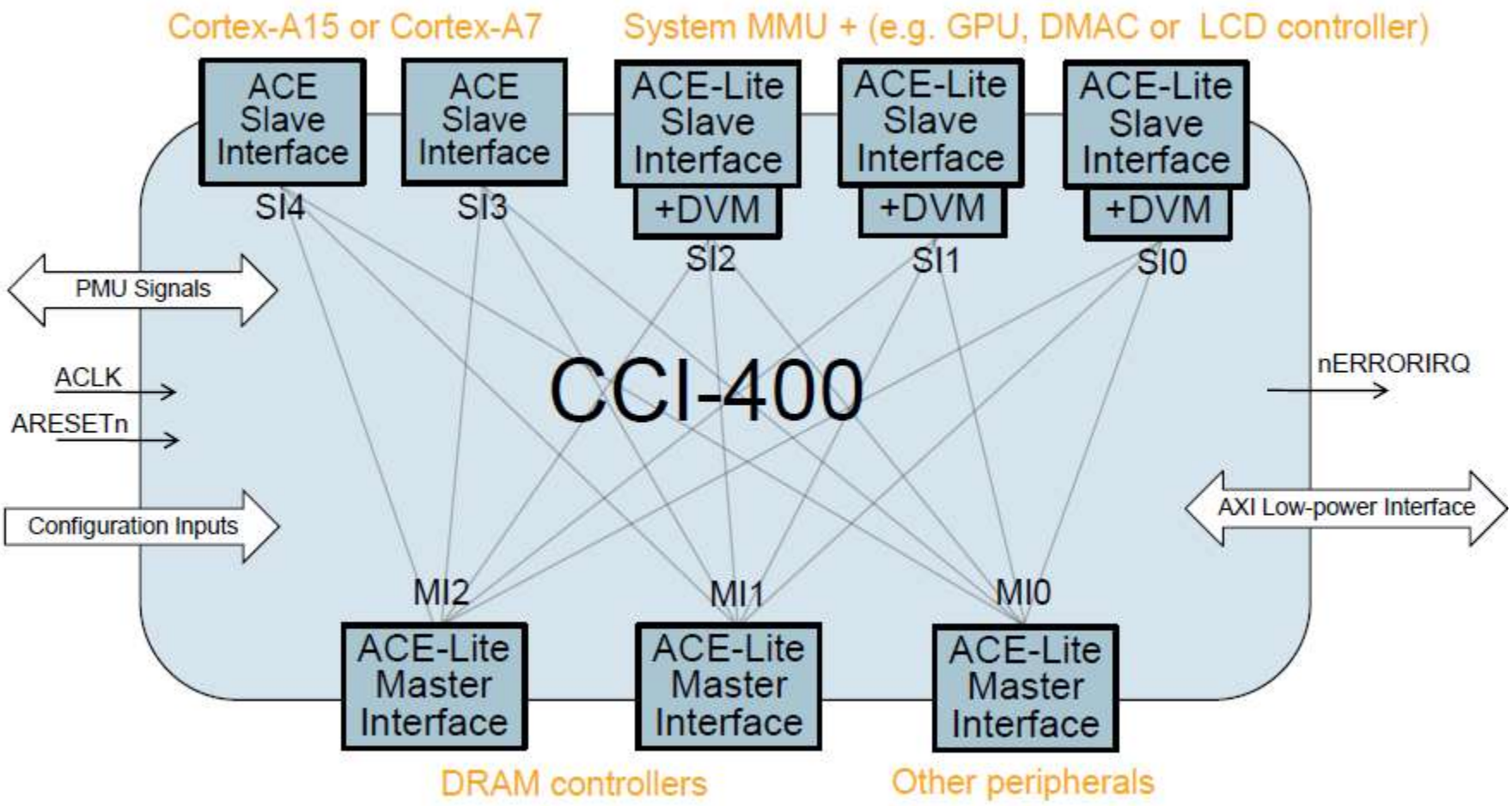
Name	Product	Headline features
CCI-5xx	Cache Coherent Interconnect with snoop filter	There are cache coherent interconnects with snoop filters to reduce snoop traffic
DMC-500	Dynamic Memory Controller	2 channel (2x72-bit) LPDDR2/DDR3 memory controller
MMU-500	System Memory Management	Up to 48 bit virtual addresses Adds ARMv8 virtualization support but supports also A15/a7 page table formats
GIC-500	Generic Interrupt Controller	Share interrupts across clusters, ARMv8 virtualization extensions compliant

Main features of ARM's cache coherent interconnects used for mobiles []

Main features	CCI-400	CCI-500	CCI-550
Date of introduction	10/2010	11/2014	10/2015
Supported processor models (Cortex-Ax MPCore)	A15/A7/A17/ A57/A53	A15/A7/A17 /A57/A53/A72	A57/A53 and next proc.
No. of fully coherent ACE slave ports for CPU clusters (of 4 cores)	2	1-4	1-6
No of I/O-coherent ACE-Lite slave ports	1-3	0-6 (max 7 slave ports)	0-6 (max 7 slave ports)
No. of ACE-Lite master ports for memory channels	1-2 ACE-Lite DMC-500 (LPDDR4/3)	1-4 AXI-4 DMC-500 (LPDDR4/3	1-6 AXI4 DMC-500 (LPDDR4/3)
No. of I/O-coherent master ports for accelerators and I/O	1 ACE-Lite	1-2 AXI4	1-3 (max 7 master ports)
Data bus width	128-bit	128-bit	128-bit
Integrated L3 cache	No	No	No
Integrated snoop filter	No, broadcast snoop coherency	Yes, there is a directory of caches content, to reduce snoop traffic	
Interconnect topology	Switches	Switches	Switches

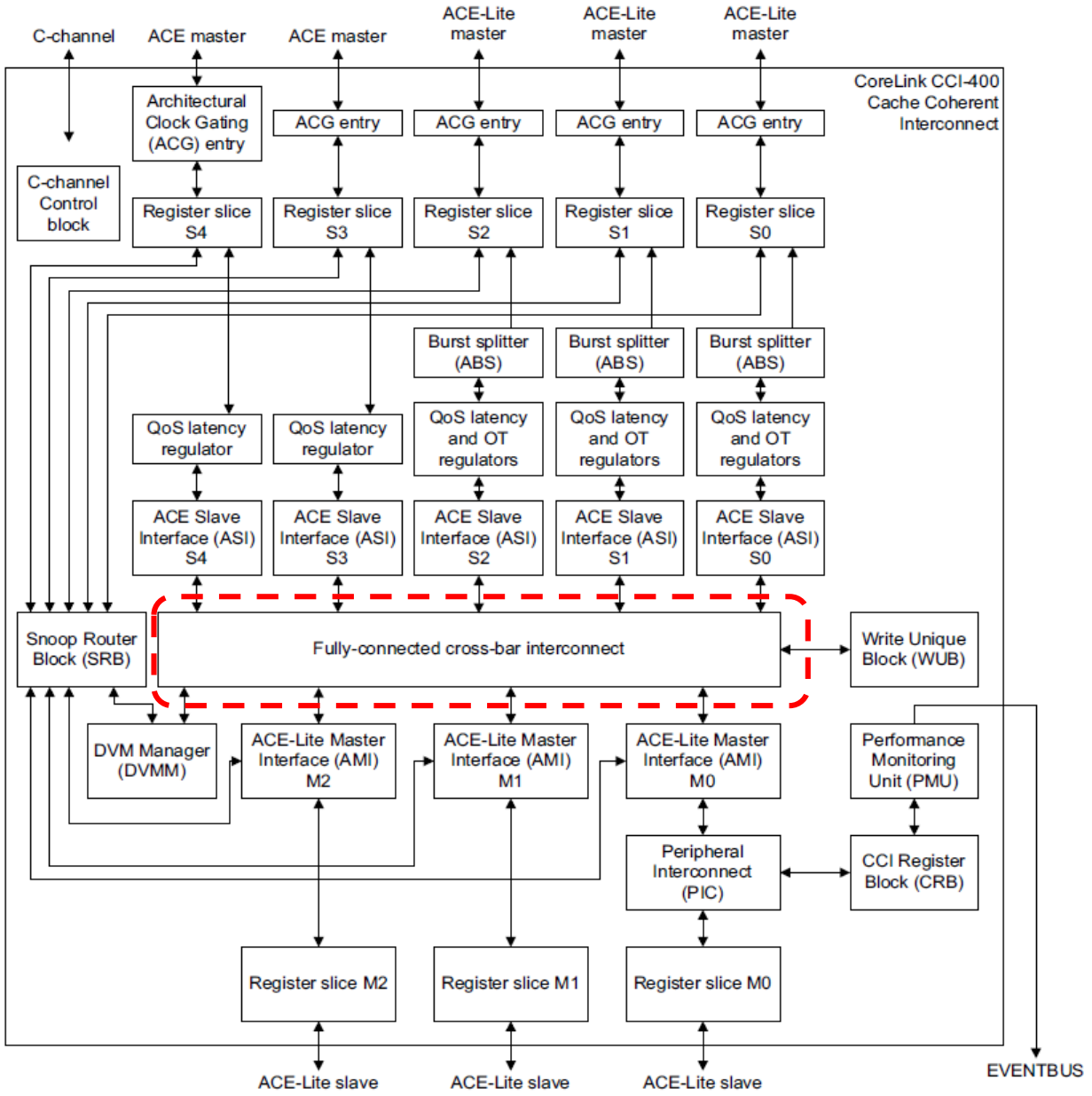
https://www.arm.com/products/system-ip/interconnect/corelink-cci-550-cache-coherent-interconnect.php

Block diagram of the CCI-400 []



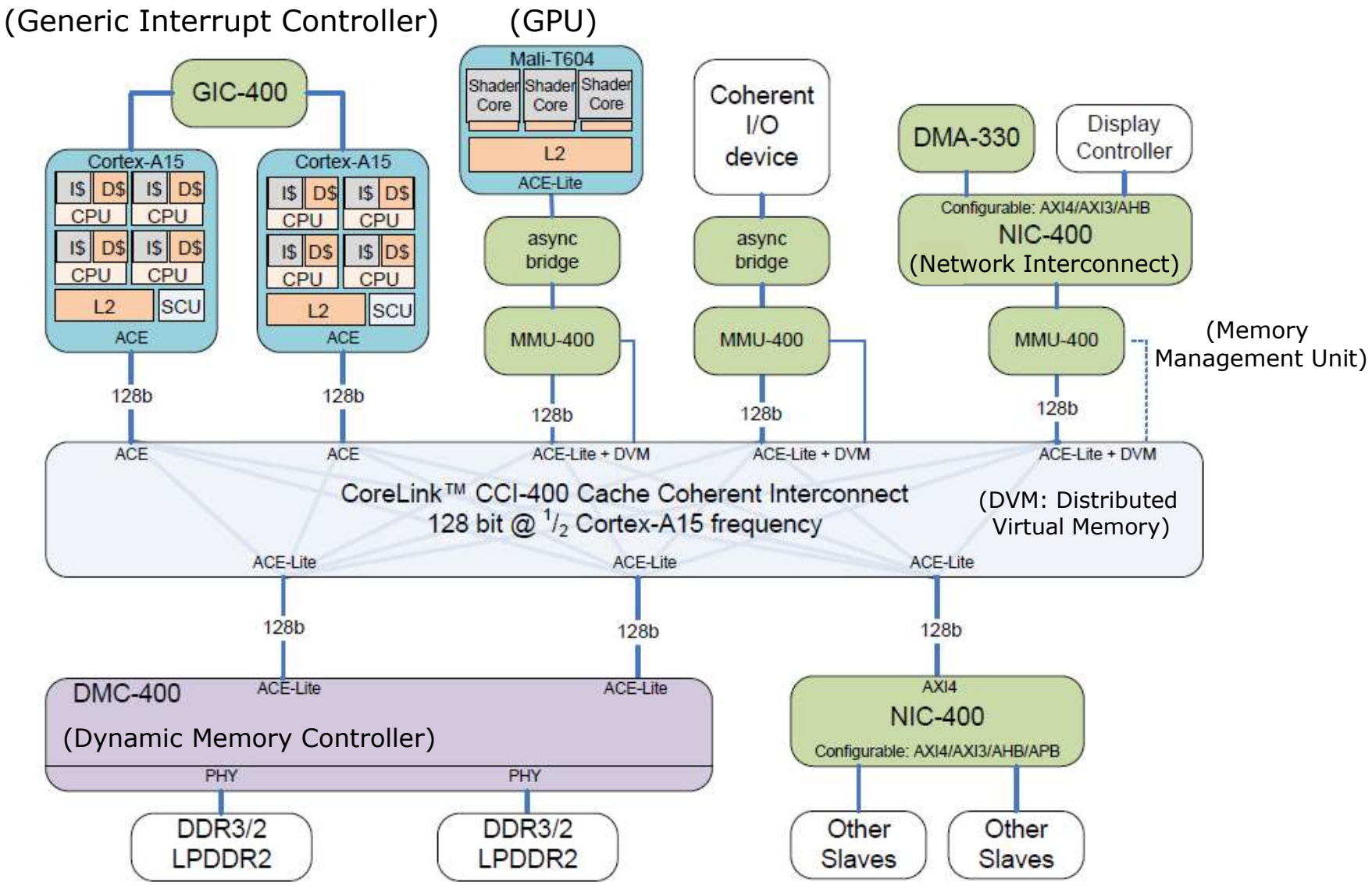
http://www.ece.cmu.edu/~ece742/lib/exe/fetch.php?media=arm_multicore_and_system_coherence_-_cmu.pdf

Internal architecture of the CCI-400 Cache Coherent Interconnect []

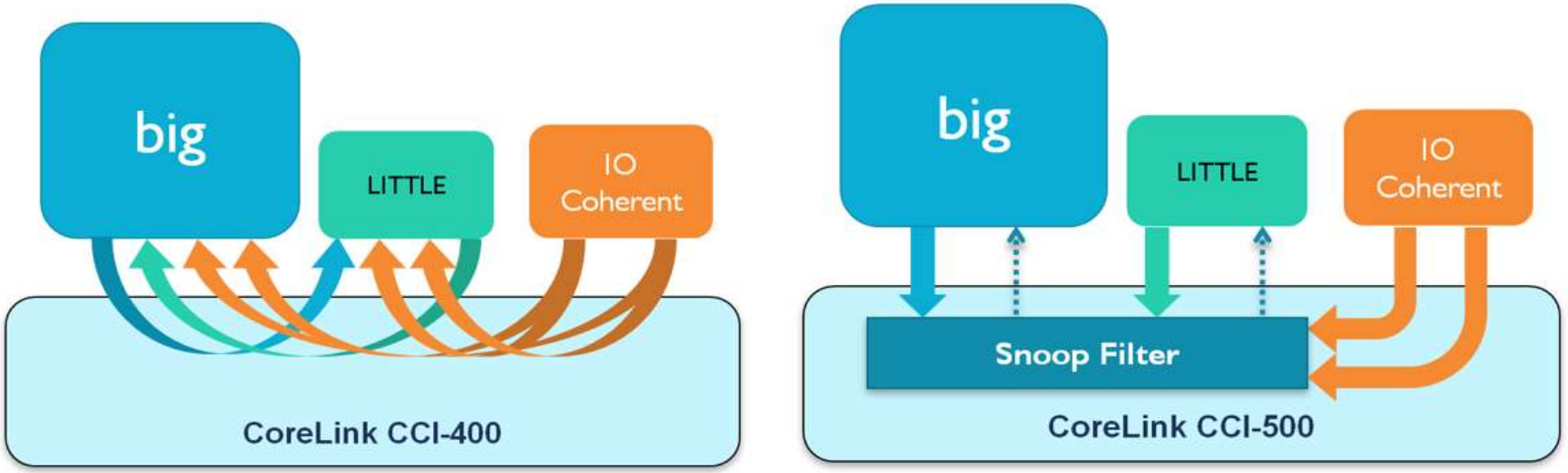


http://infocenter.arm.com/help/topic/com.arm.doc.ddi0470c/DDI0470C_cci400_r0p2_trm.pdf

Example 1: Cache coherent SOC based on the CCI-400 interconnect []



http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf


big
LITTLE
IO Coherent
CoreLink CCI-400
big
LITTLE
IO Coherent
Snoop Filter
CoreLink CCI-500

Hardware Coherency and Snoops

The simplest implementation of cache coherency is to broadcast a snoop to all processor caches to locate shared data on-demand. When a cache receives a snoop request, it performs a tag array lookup to determine whether it has the data, and sends a reply accordingly.
For example in the image above we can see arrows showing snoops between big and LITTLE processor clusters, and from IO interfaces into both processor clusters. These snoops are required for accessing any shared data to ensure their caches are hardware cache coherent. In other words, to ensure that all processors and IO see the same consistent view of memory.

For most workloads the majority of lookups performed as a result of snoop requests will miss, that is they fail to find copies of the requested data in cache. This means that many snoop-induced lookups may be an unnecessary use of bandwidth and energy. Of course we have removed the much higher cost of software cache maintenance, but maybe we can optimize this further?

Introducing the Snoop Filter

This is where a snoop filter comes in. By integrating a snoop filter into the interconnect we can maintain a directory of processor cache contents and remove the need to broadcast snoops.

The principle of the snoop filter is as follows:
A tag for all cached shared memory is stored in a directory in the interconnect (snoop filter)
All shared accesses will look up in this snoop filter which has two possible responses:

HIT –> data is on-chip, a vector is provided to point to the cluster with the data
MISS –> go fetch from external memory

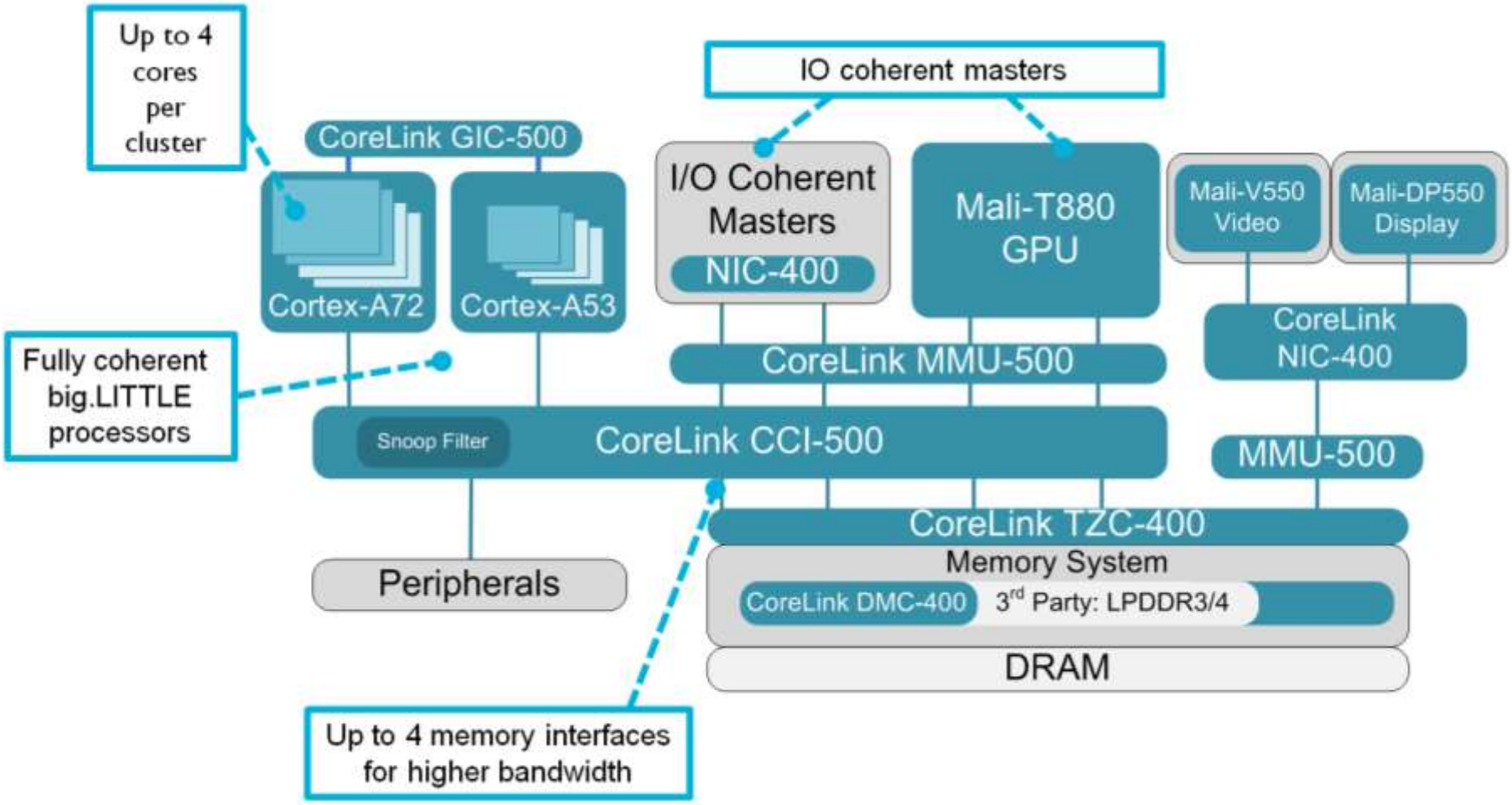
https://community.arm.com/groups/processors/blog/2015/02/03/extended-system-coherency--part-3--corelink-cci-500

Introduction of snoop filters in the CCI-500

Snoop filter Supported The multiprocessor device provides support for an external snoop filter in an interconnect. It
indicates when clean lines are evicted from the processor by sending Evict transactions on the ACE
write channel. However there are some cases where incorrect software can prevent an Evict transaction from being sent, therefore you must ensure that any external snoop filter is built to
handle a capacity overflow that sends a back-invalidation to the processor if it runs out of storage.

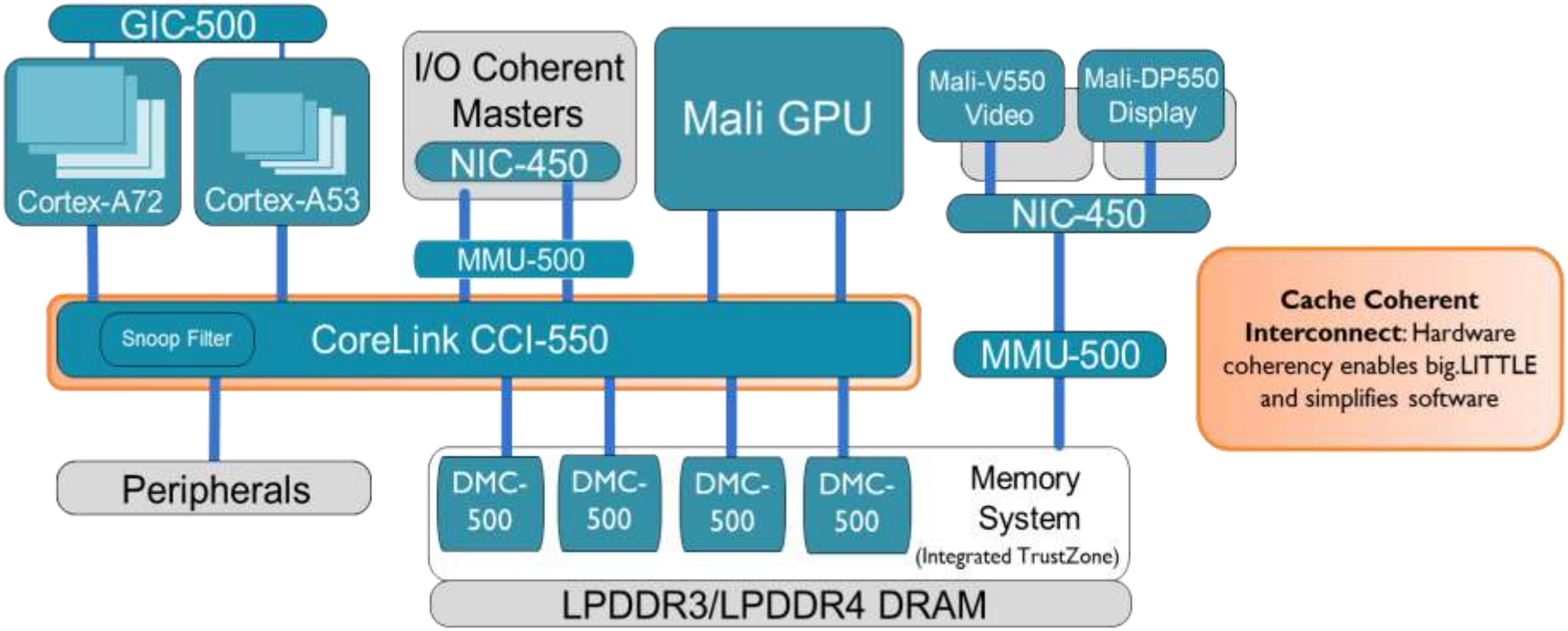
a7 trm

Example 1: Cache coherent SOC based on the CCI-500 interconnect []



http://www.arm.com/products/system-ip/interconnect/corelink-cci-500.php

Example 1: Cache coherent SOC based on the CCI-550 interconnect []

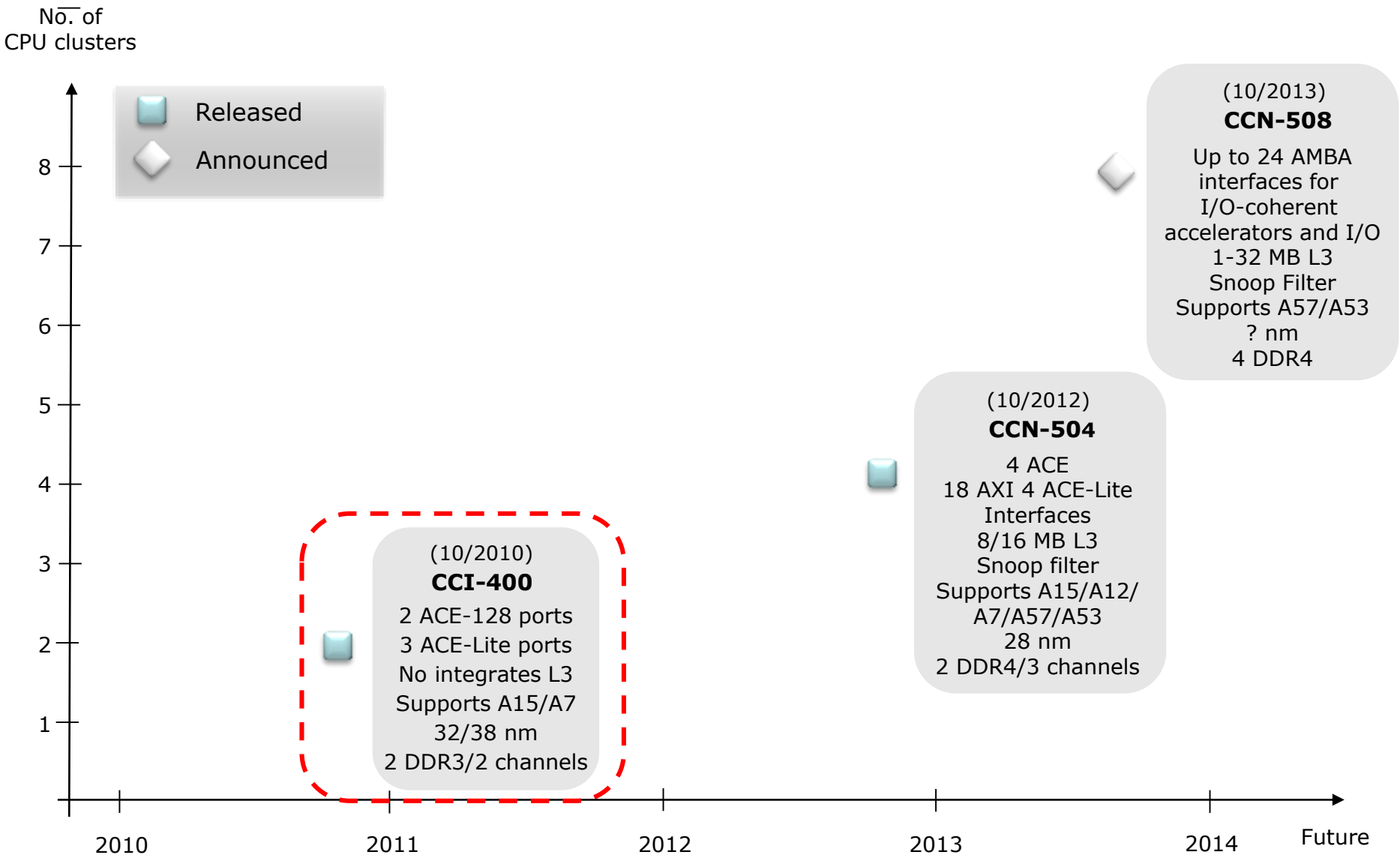


Use of ARM's interconnect IP by major SOC providers

Samsung Exynos 8890: SCI (Samsung Coherent Interconnect)
7420 CCI-400

AMD A1100 SLS Fabric Interconnect from Silver Lighning Systems
60 Gbps switching fabric is available as a PCI Express expansion card or a standalone ASIC for custom server applications.

ARM's 1. generation cache coherent interconnects

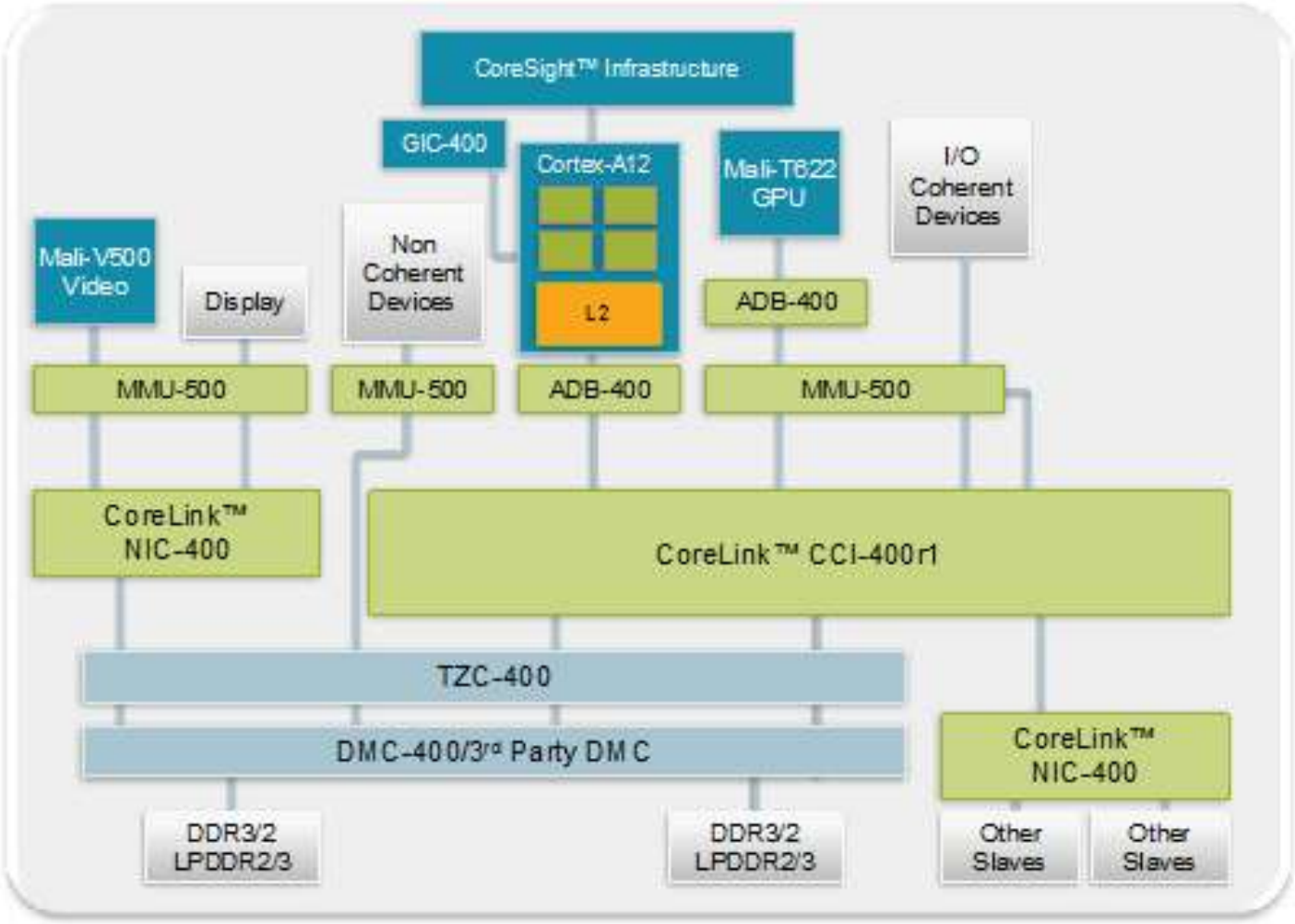


Main features of the CCI-400 Cache Coherent Interconnect []

Main features	CCI-400
Number of master ports for CPU clusters	2 (128-bit ACE) for up to 2x4 cores
No. of 128-bit ACE-Lite Master ports	2
No. of ACE-Lite Slave ports	3 for GPU, accelerators and I/O interfaces
Supported processors (Cortex-Ax)	A15/A7/A12/A57/A53
Integrated L3 cache	No
Integrated snoop filter	No, only external
Address space	40 bit Physical (1TB), supports ARMv7-A & ARMv8-A
Support of memory channels	2x DDR2/3 approximately 25GB/s sustained bandwidth at 533MHz for dual channel memory
Quality of Service	Integrated bandwidth and latency regulators, QoS Virtual Networks
Interconnect topology	Crossbar
Technology	32/28 nm

Extended System Coherency - Part 2 - Implementation, big.LITTLE, GPU Compute and Enterprise
Közzétette neilparris ARM Processors alatt a a következő napon: 2014.02.17. 11:41:28

Example 2: Cache coherent SOC with Cortex-A12 and MaliT622 based on the CCI-400 interconnect []



http://www.dailytech.com/ARM+Reveals+Cortex+A12+With+Malli+T600+GPU+OpenGL+ES+30+for+MidRange+Mobile/article31677c.htm 6/2013

Further technologies supported by the CCI-400

- big.LITTLE technology
- Barriers
- Virtualization
- End-to-end QoS (Quality of Service)
  (Allows traffic and latency regulations, removes traffic blocking, as indicated in the next Figure).

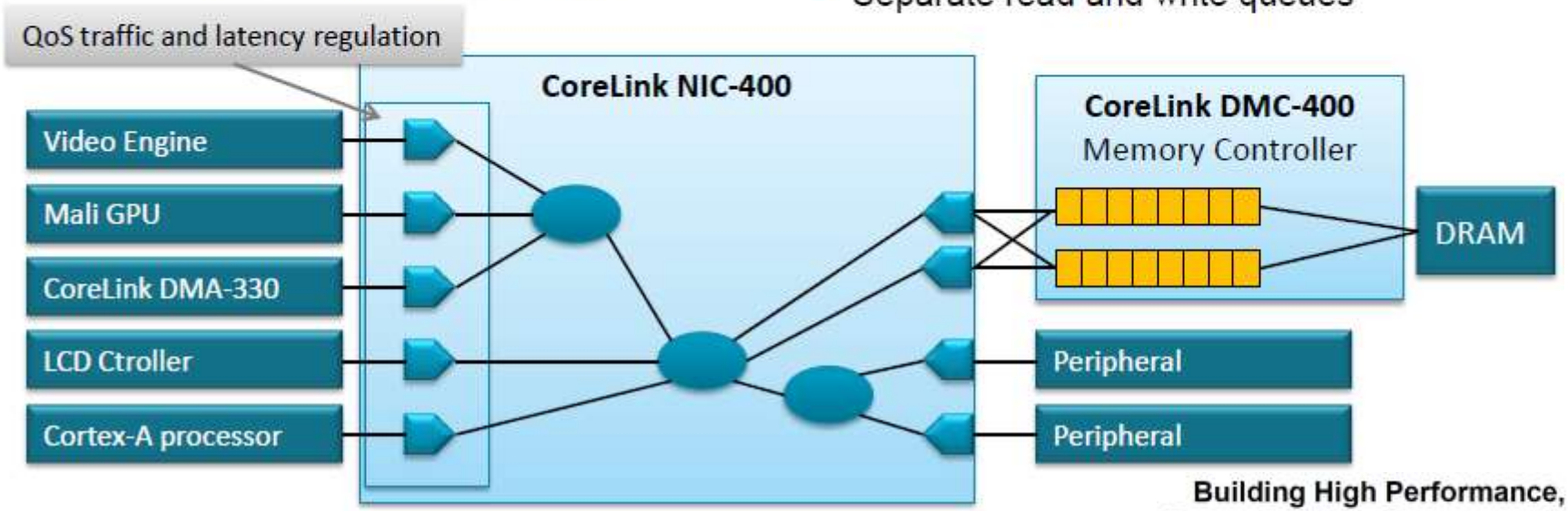
End-to-end QoS (Quality of Service) []

Interconnect

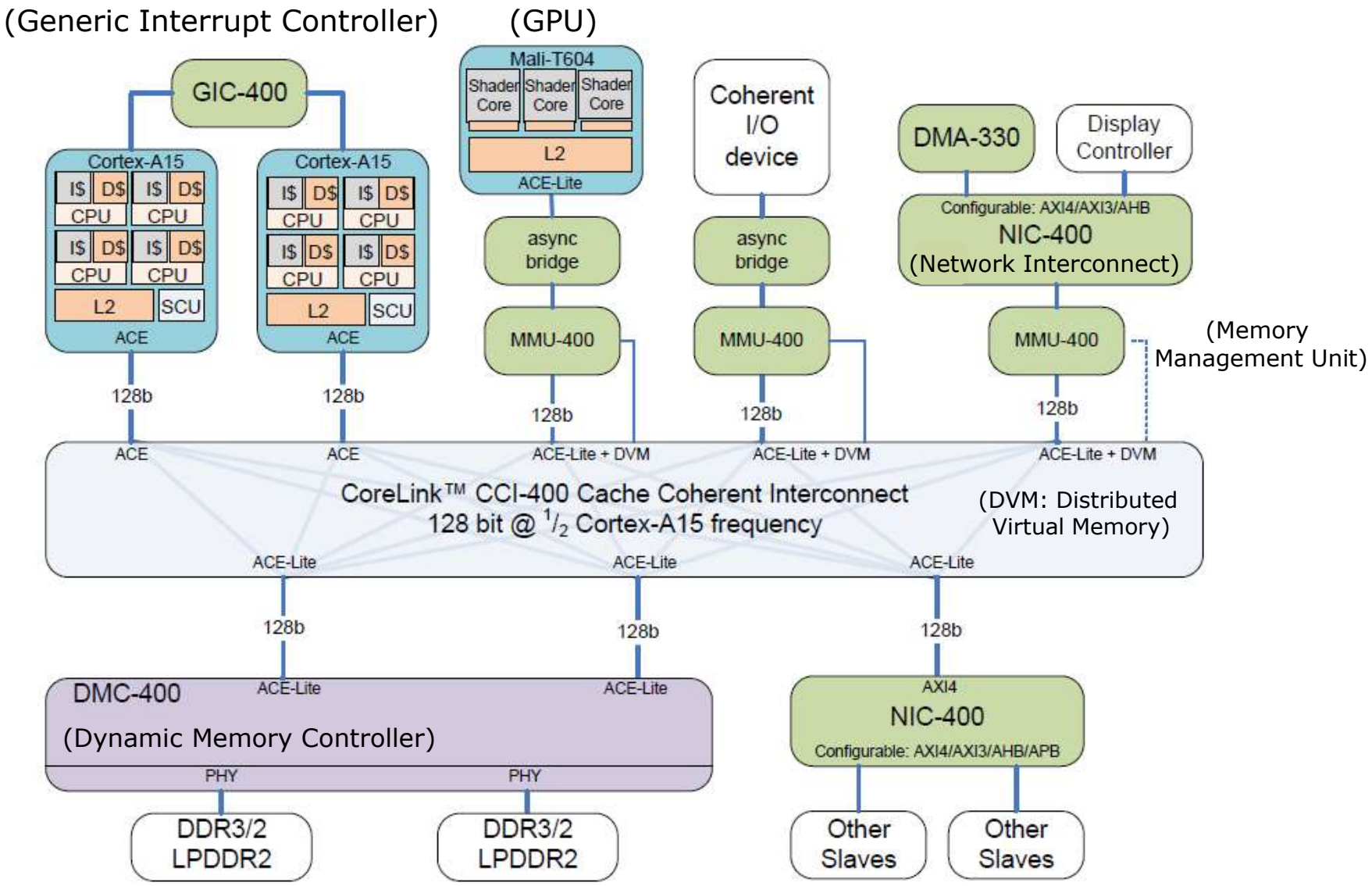
- Traffic regulation on entry
  - Maximum bandwidth limits
  - Outstanding transaction management
- Dynamic priority
  - Uses QoS value in NIC-301, NIC-400
  - Changes priority to meet target latency
- Virtual Networks
  - Remove blocking through system

Memory Controller

- Scheduler aims for high memory efficiency whilst meeting QoS requirements
- Support for latency regulation and arbitration with QoS value
- Timeout mechanism for streaming and real-time traffic
- Separate read and write queues

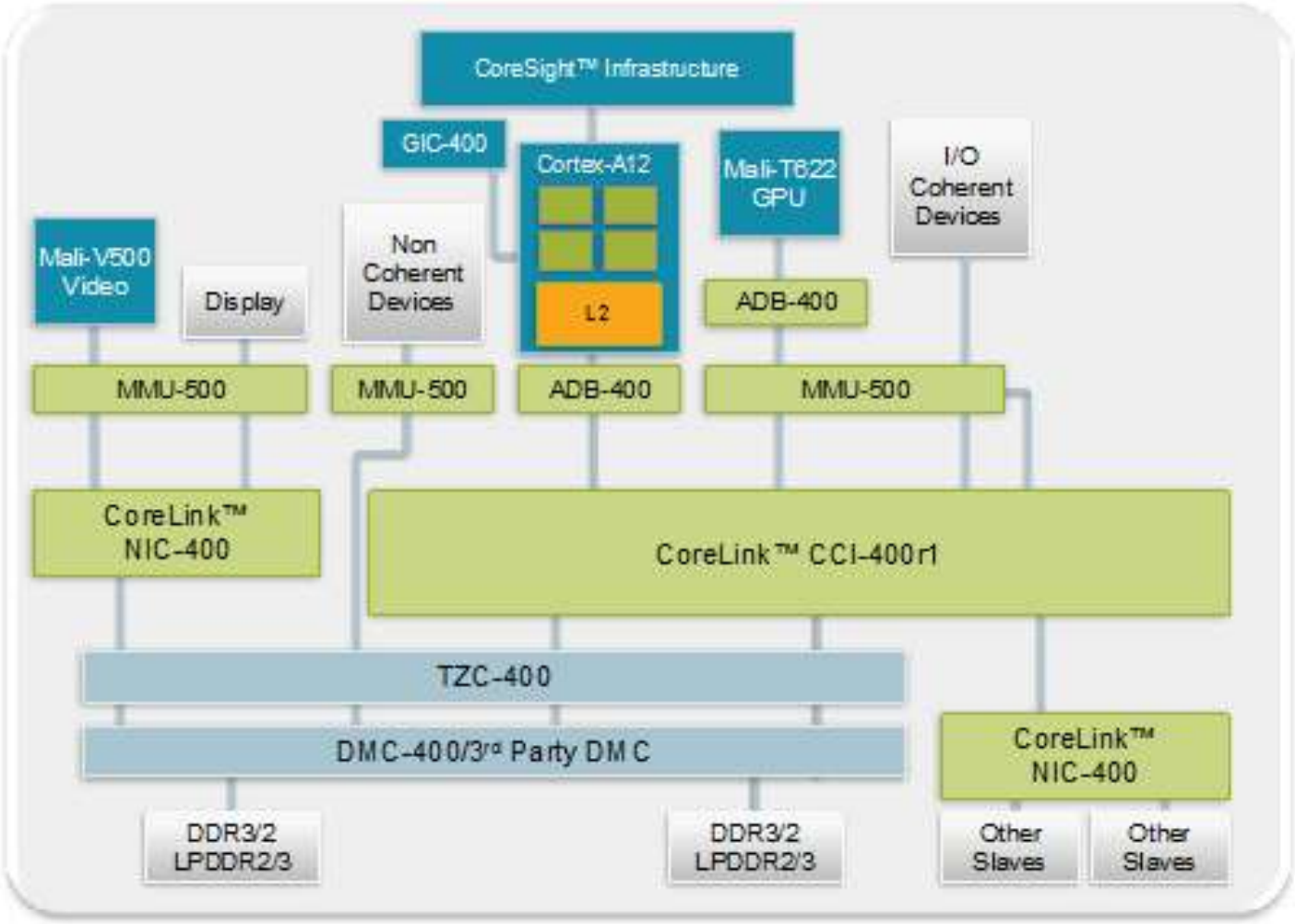


Example 1: Cache coherent SOC based on the CCI-400 interconnect []



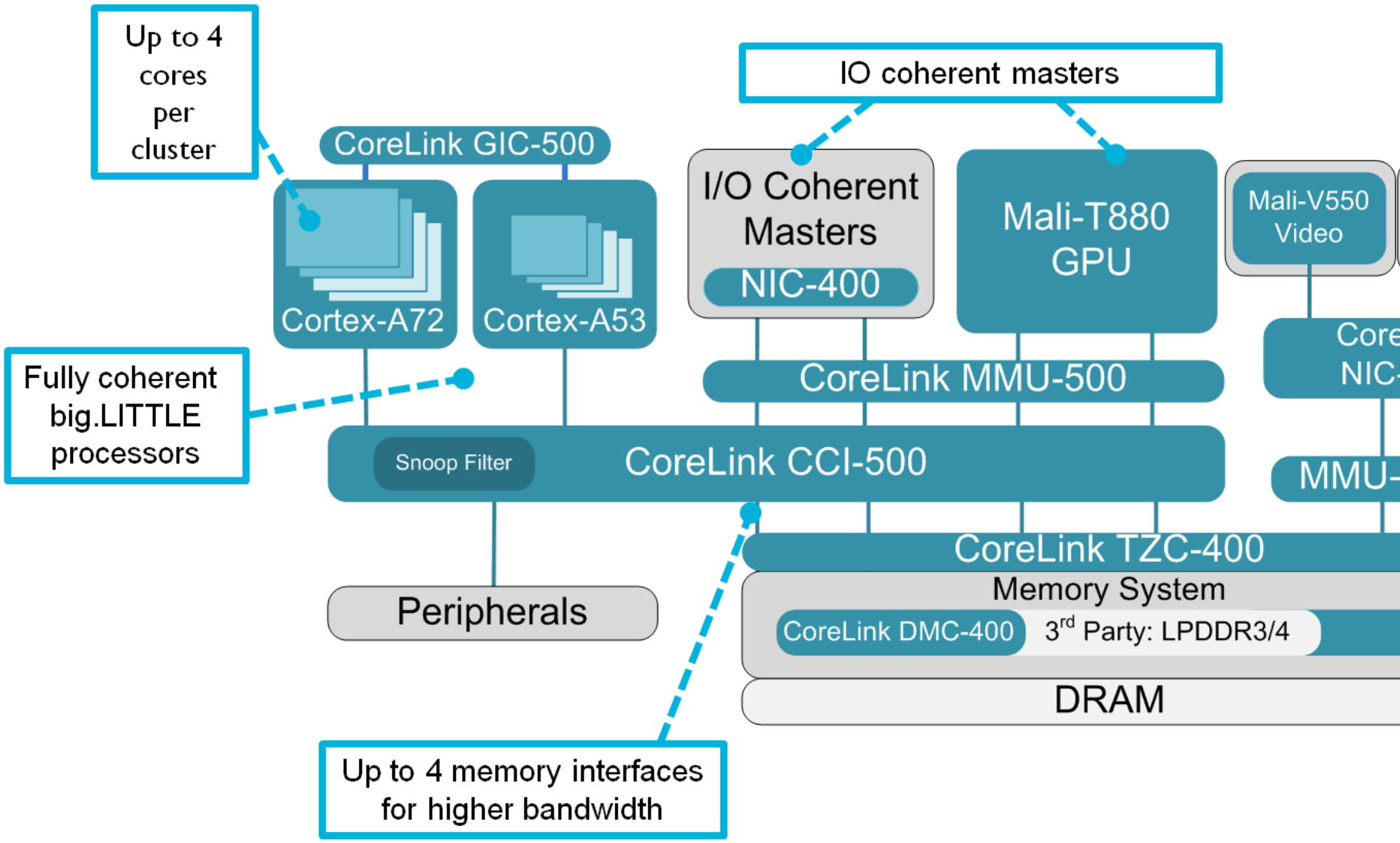
http://www.arm.com/files/pdf/CacheCoherencyWhitepaper_6June2011.pdf

Example 2: Cache coherent SOC with Cortex-A12 and MaliT622 based on the CCI-400 interconnect []

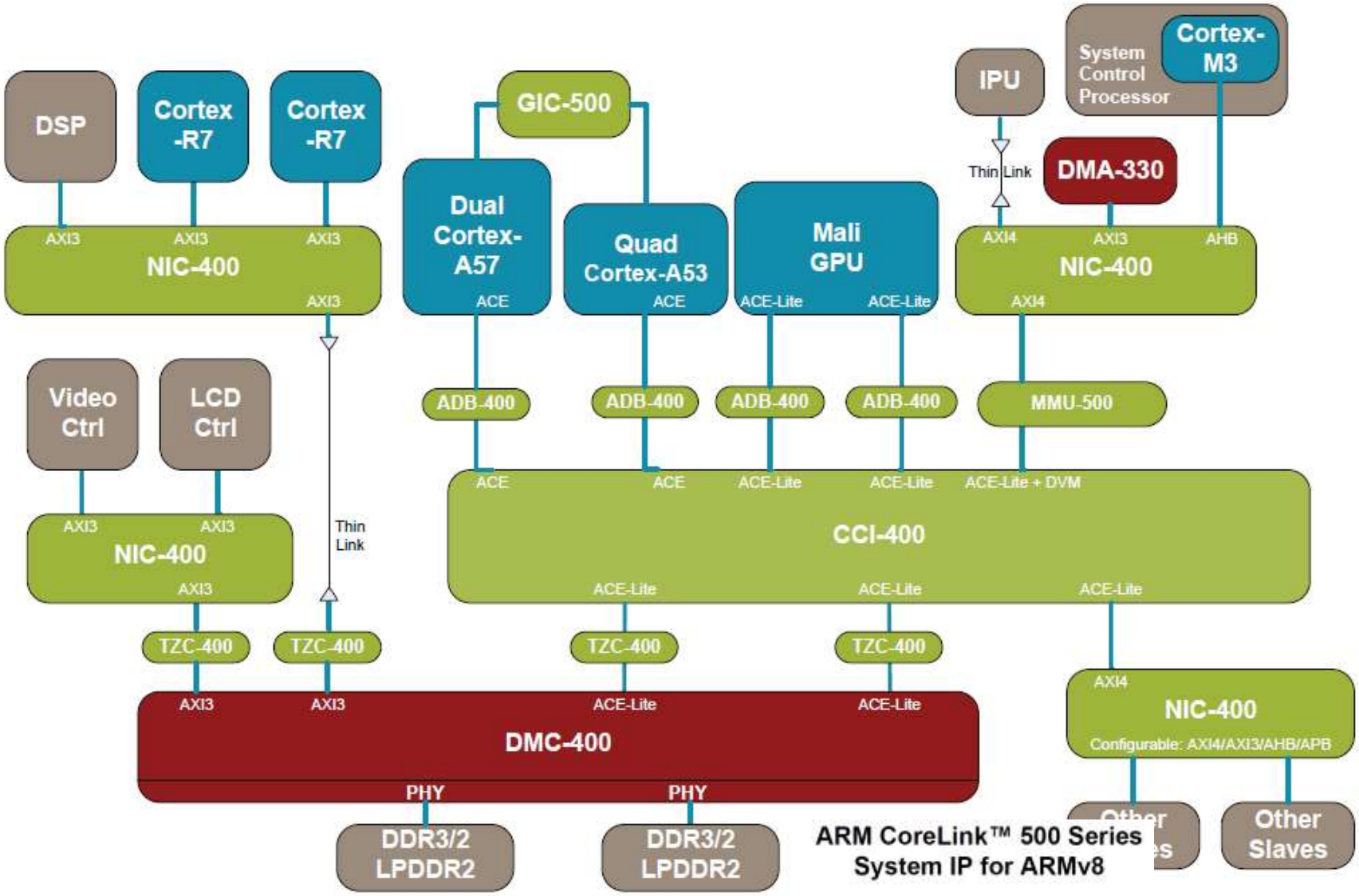


http://www.dailytech.com/ARM+Reveals+Cortex+A12+With+Malli+T600+GPU+OpenGL+ES+30+for+MidRange+Mobile/article31677c.htm 6/2013

CoreLink™ CCI-500



Example 3: High-End mobile platform based on the CCI-400 interconnect with dual Cortex-A57 and quad Cortex-A53 clusters and Mali GPU []



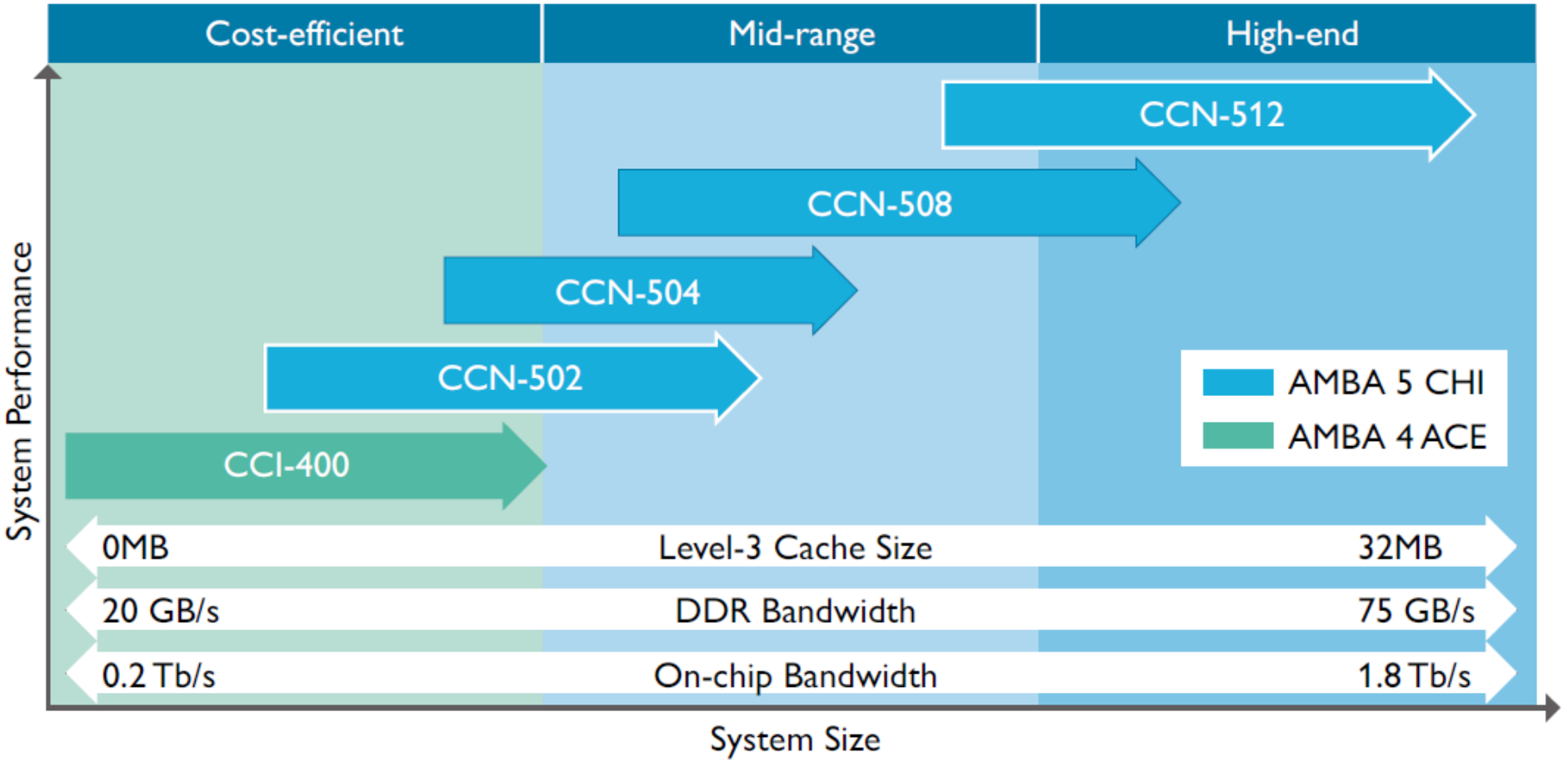
2.3.7.3 ARM’s cache coherent interconnects for enterprise computing

2.3.7.3 ARM's cache coherent interconnects for enterprise computing

Recently, there are four implementations:

- the CCN-502 (Core Coherent Network–502) (2014)
- the CCN-504 (Core Coherent Network–504) (2012)
- the CCN-508 (Core Coherent Network-508) (2013) and
- the CCN-512 (Core Coherent Network–504) (2014)

Key parameters of ARM's cache coherent interconnects for enterprise computing (simplified) []



ARM CoreLink Cache Coherent Network (CCN) Family

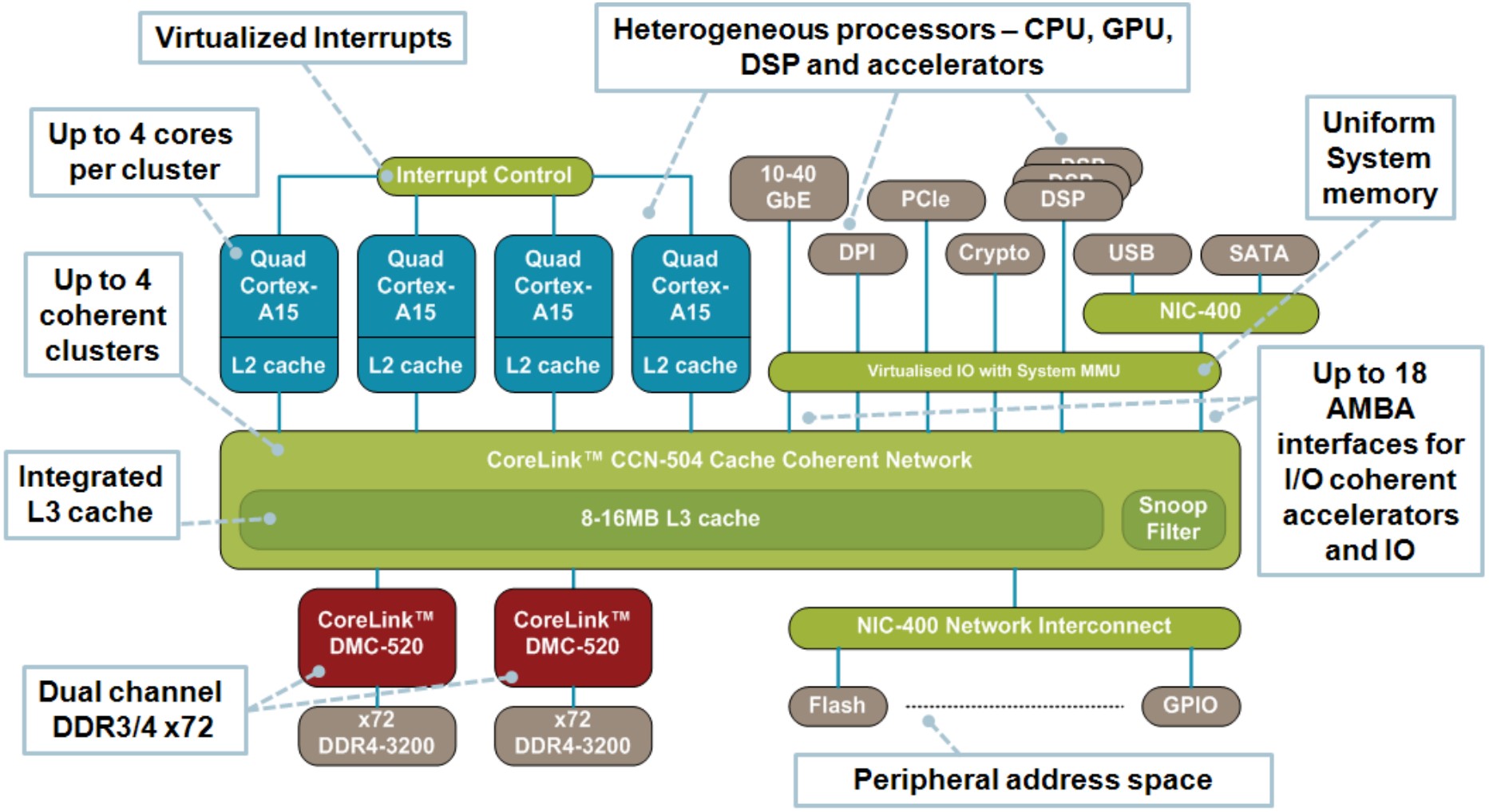
Main features of the CCN-5xx interconnects for enterprise computing

Main features	CCN-502	CCN-504	CCN-508	CCN-512
Date of introduction	12/2014	10/2012	10/2013	10/2014
Supported processors (Cortex-Ax)	A57/A53	A15/A57/A53	A57/A53 and next proc.	A57/A53 and next proc.
No. of fully coherent slave ports for CPU clusters (of up to 4 cores)	4 (CHI)	4 (AXI4/CHI)	8 (CHI)	12 (CHI)
No. of I/O-coherent slave ports for accelerators and I/O	9 ACE-Lite/AXI4	18 ACE-Lite/AXI4 /AXI3	24 ACE-Lite/AXI4	24 ACE-Lite/AXI4
Integrated L3 cache	0-8 MB	1-16 MB	1-32 MB	1-32 MB
Integrated snoop filter	Yes	Yes	Yes	Yes
Support of memory controllers (up to)	4x DMC-520 (DDR4/3 up to DDR4-3200)	2x DMC-520 (DDR4/3 up to DDR4-3200)	4x DMC-520 (DDR4/3 up to DDR4-3200)	4x DMC-520 (DDR4/3 up to DDR4-3200)
DDR bandwidth up to	102.4 GB/s	51.2 GB/s	102.4 GB/s	102.4 GB/s
Interconnect topology	Ring	Ring (Dickens)	Ring	Ring
Sustained interconnect bandwidth	0.8 Tbps	1 Tbps	1.6 Tbps	1.8 Tbps
Technology	n.a.	28 nm	n.a.	n.a.

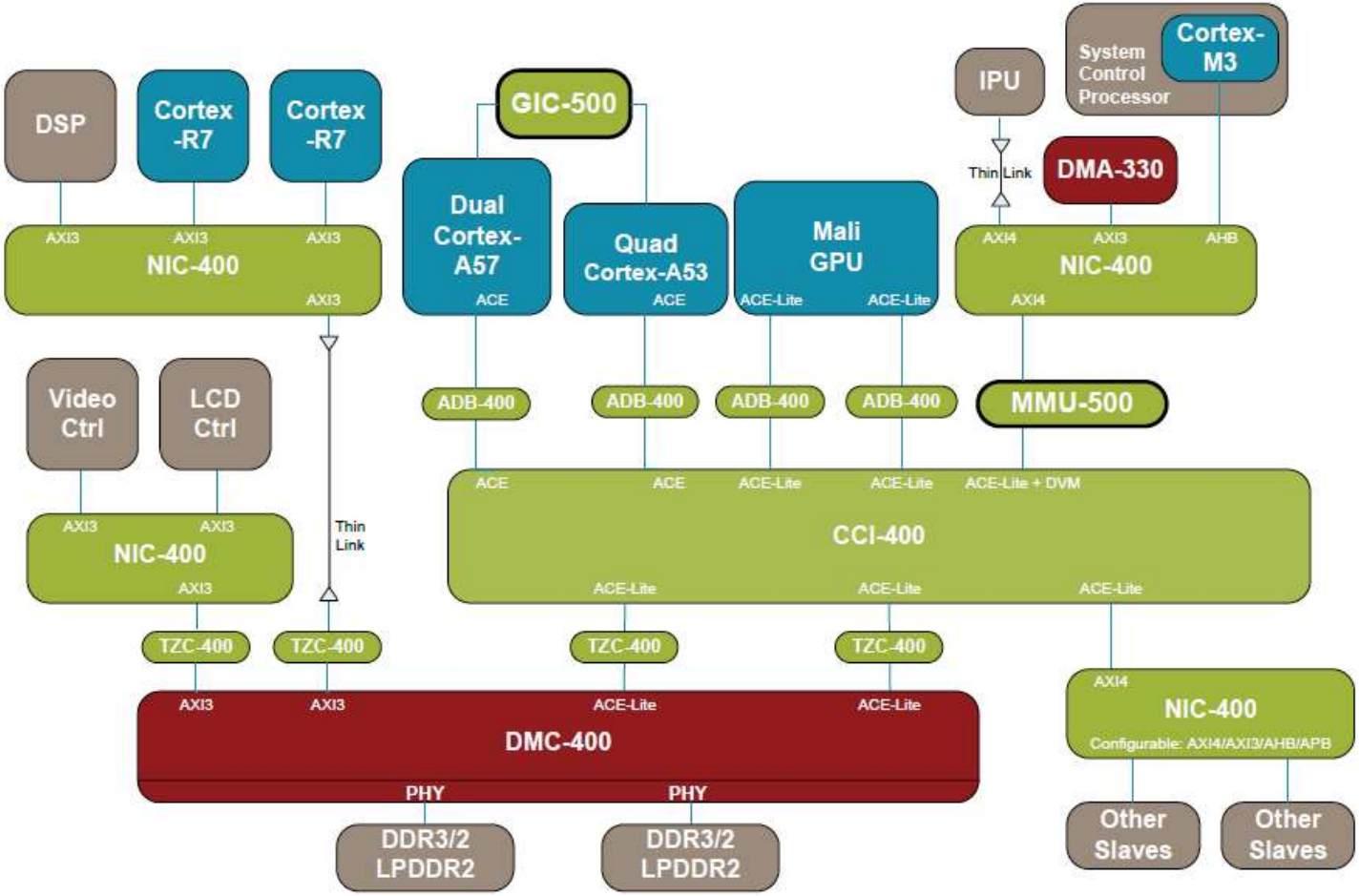
CoreLink 500 System components (based on [])

Name	Product	Key features
CCN-5xx	Cache Coherent Network	Support of 4/8/12 core clusters of up to four cores mainly for ARM v8 processors
DMC-520	Dynamic Memory Controller	2 channel (2x40/72-bit) DDR3/DDR4/LPDDR3 memory controller 128-bit system interface DFI 3.1 memory interface Memory transfer rate up to DDR4-3200
MMU-500	Memory Management Unit	Up to 48-bit virtual addresses
GIC-500	Generic Interrupt Controller	Share interrupts across clusters

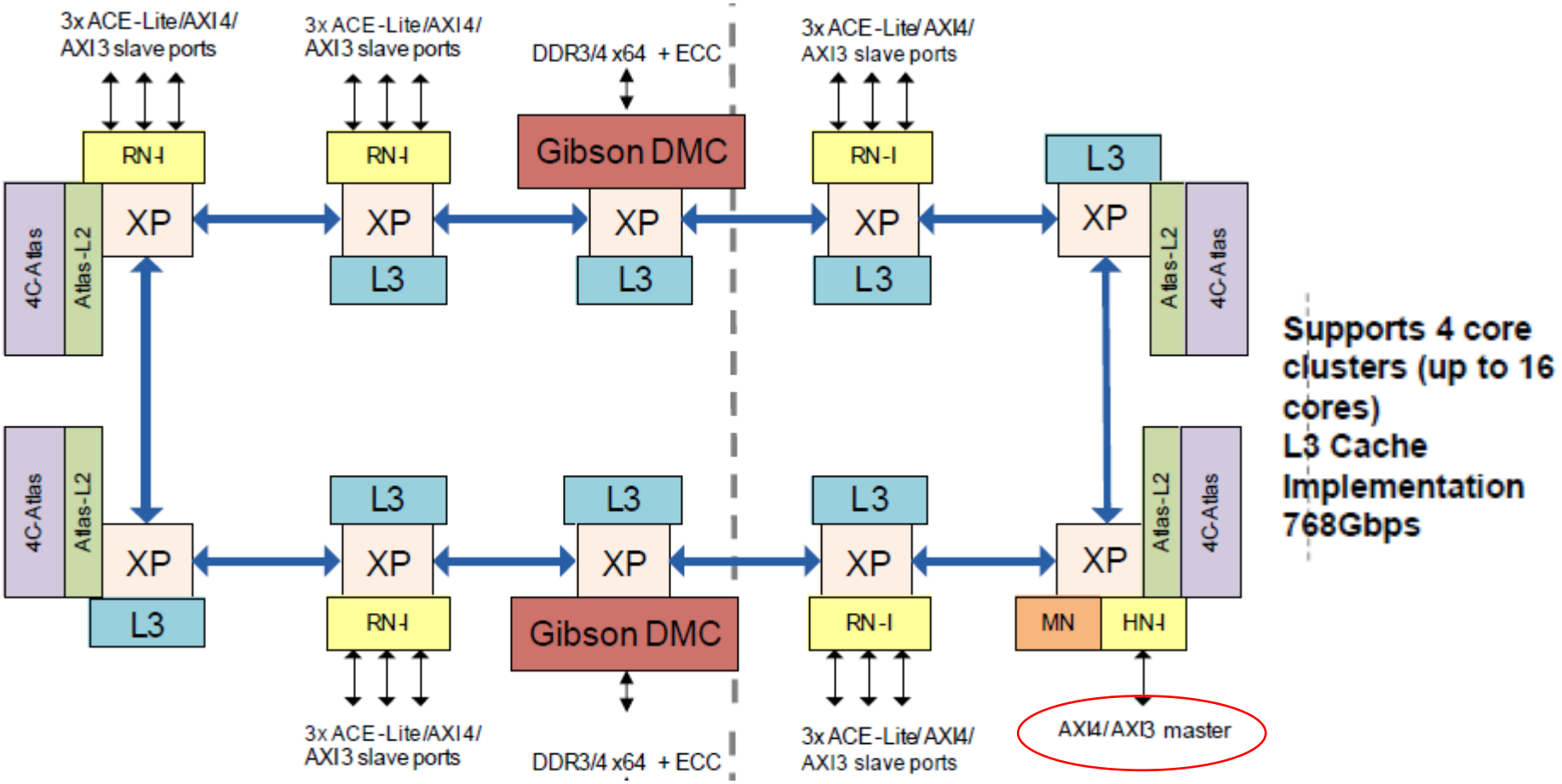
Example 1: SOC based on the cache coherent CCN-504 interconnect []



http://www.arm.com/products/system-ip/interconnect/corelink-ccn-504-cache-coherent-network.php


DSP
Cortex -R7
Cortex -R7
GIC-500
IPU
System Control Processor
Cortex-M3
Thin Link
DMA-330
Dual Cortex-A57
Quad Cortex-A53
Mali GPU
AXI3
AXI3
AXI3
NIC-400
AXI3
ACE
ACE
ACE-Lite
ACE-Lite
AXI4
AXI3
AHB
NIC-400
AXI4
Video Ctrl
LCD Ctrl
ADB-400
ADB-400
ADB-400
ADB-400
MMU-500
ACE
ACE
ACE-Lite
ACE-Lite
ACE-Lite + DVM
AXI3
AXI3
NIC-400
AXI3
Thin Link
CCI-400
ACE-Lite
ACE-Lite
ACE-Lite
TZC-400
TZC-400
TZC-400
TZC-400
AXI4
AXI3
AXI3
ACE-Lite
ACE-Lite
DMC-400
NIC-400
Configurable: AXI4/AXI3/AHB/APB
PHY
PHY
DDR3/2 LPDDR2
DDR3/2 LPDDR2
Other Slaves
Other Slaves

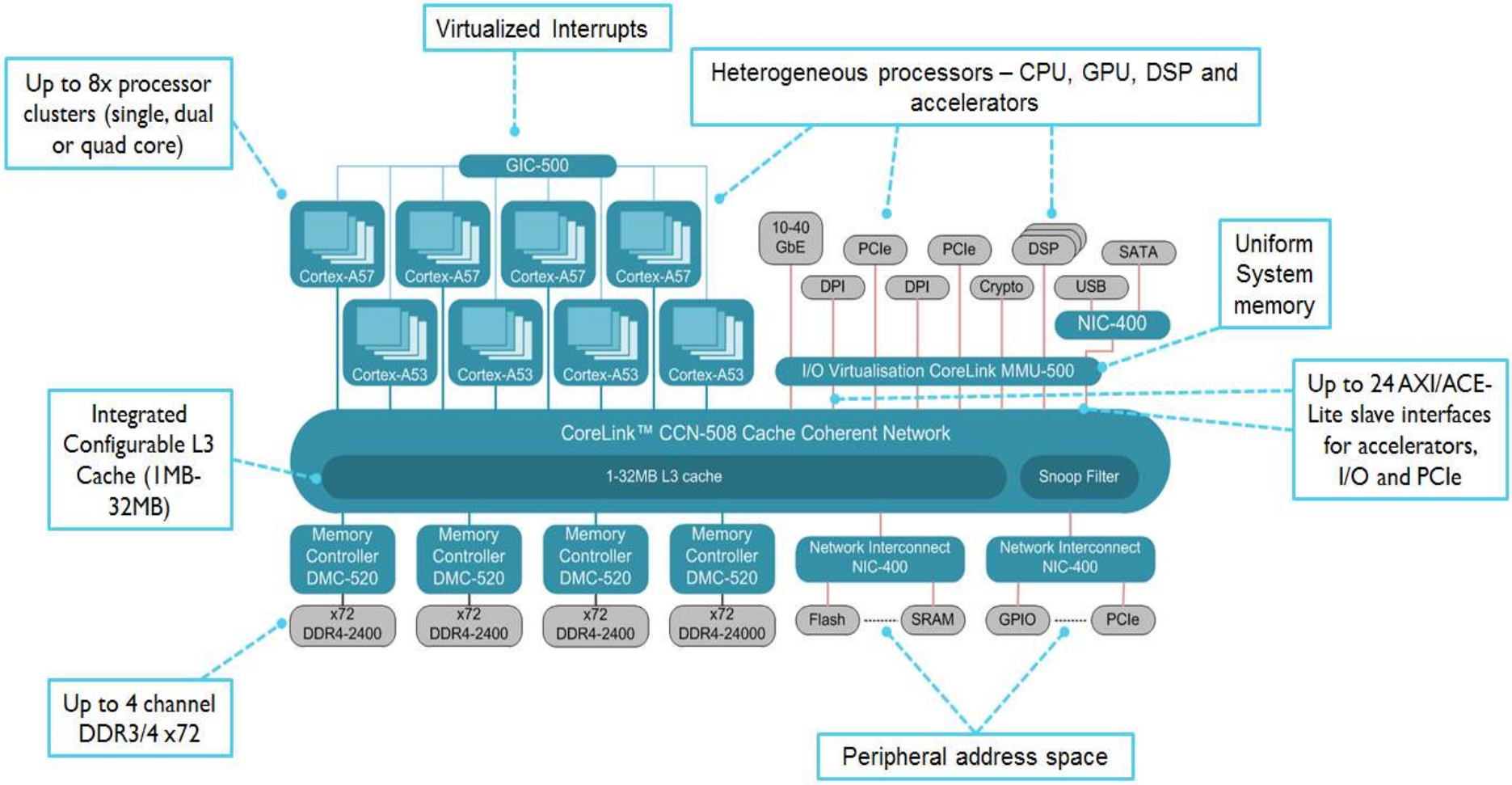
The ring interconnect fabric of the CCN-504 (dubbed Dickens) []



Remark: The Figure indicates only 15 ACE-Lite slave ports and 1 master port whereas ARM's specifications show 18 ACE-Lite slave ports and 2 master ports.

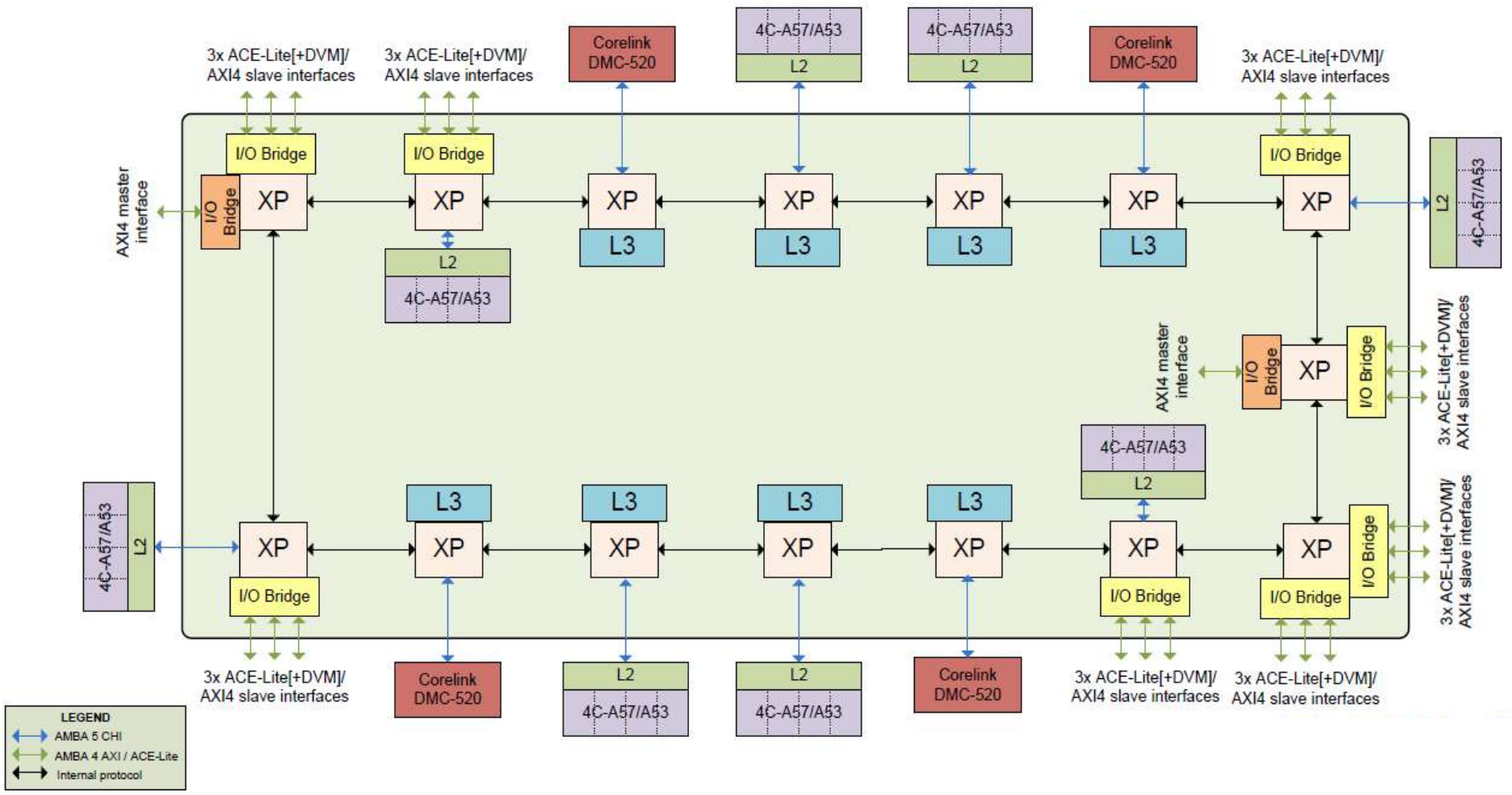
http://www.freescale.com/files/training/doc/dwf/DWF13_APF_NET_T0795.pdf

Example 2: SOC based on the cache coherent CCN-508 interconnect []



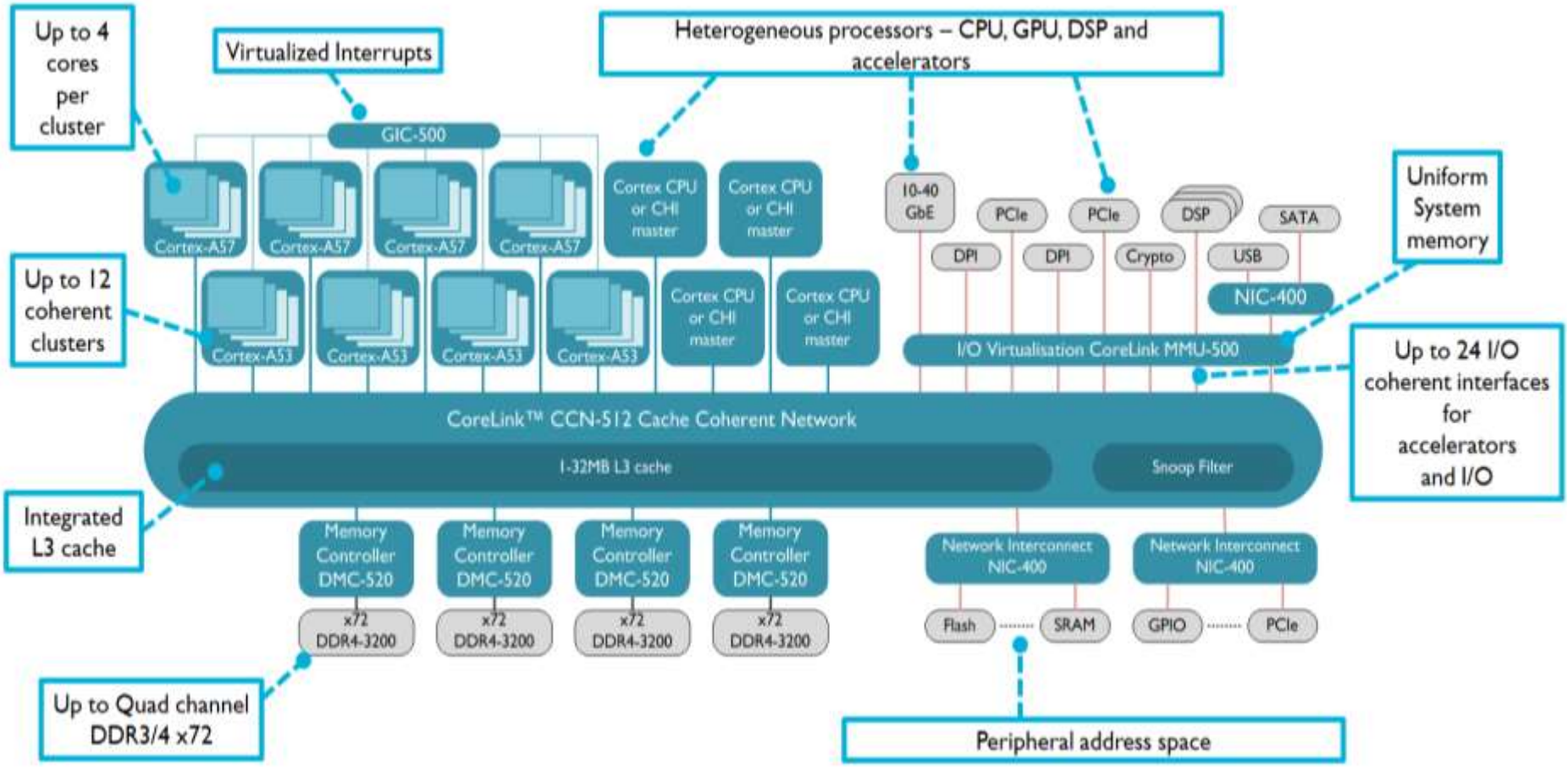
https://www.arm.com/products/system-ip/interconnect/corelink-ccn-508.php

The ring interconnect fabric of the CCN-508 []



http://www.hotchips.org/wp-content/uploads/hc_archives/hc26/HC26-11-day1-epub/HC26.11-4-ARM-Servers-epub/HC26.11.420-Network-Flippo-ARM_LSI%20HC2014%20v0.12.pdf

Example 2: SOC based on the cache coherent CCN-512 interconnect []



https://www.arm.com/products/system-ip/interconnect/corelink-ccn-512.php

Use of ARM's interconnect IP by major SOC providers

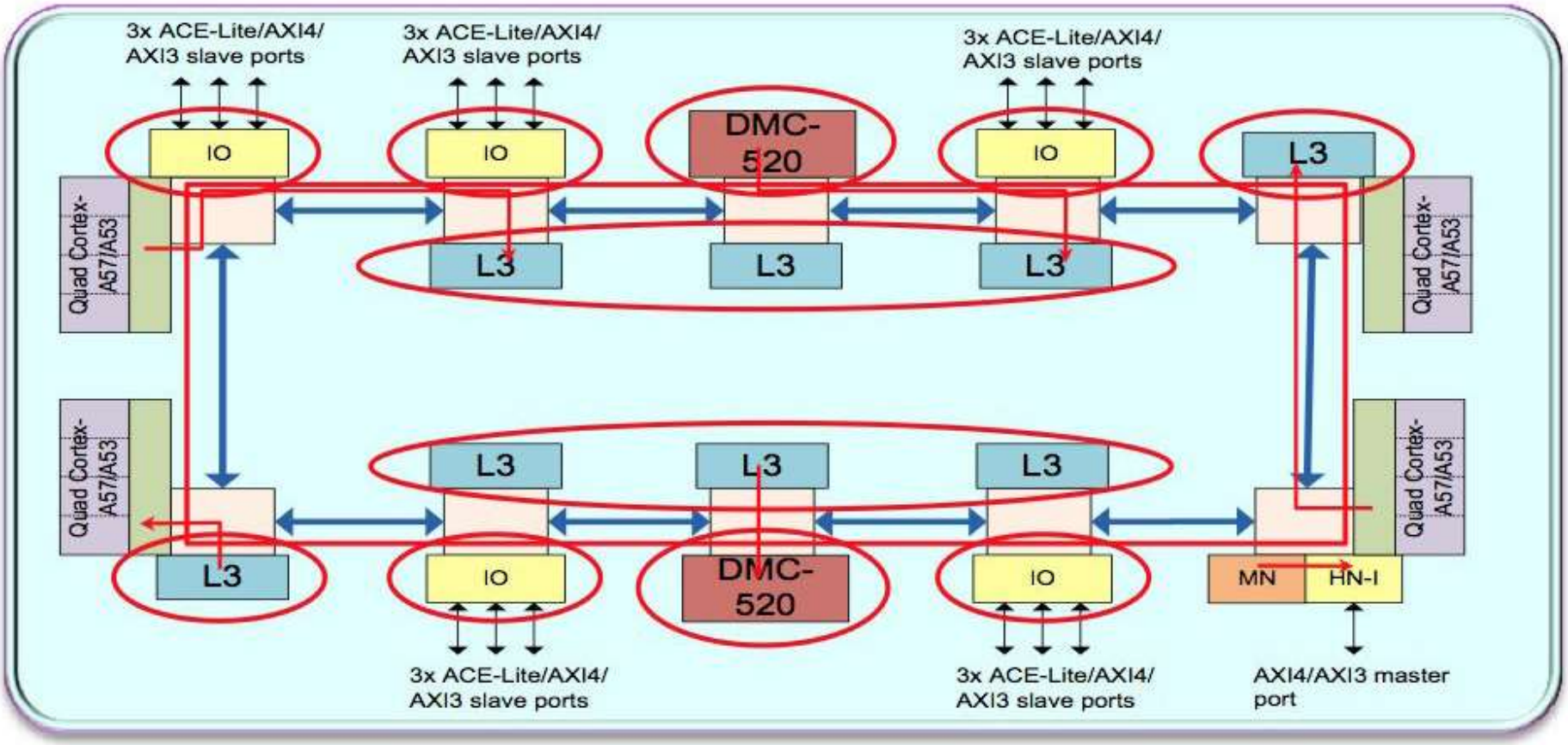
Samsung Exynos 8890: SCI (Samsung Coherent Interconnect)
7420 CCI-400

AMD A1100 SLS Fabric Interconnect from Silver Lighning Systems
60 Gbps switching fabric is available as a PCI Express expansion card or a standalone ASIC for custom server applications.

2.1 The CCN-504 2. generation cache coherent interconnect

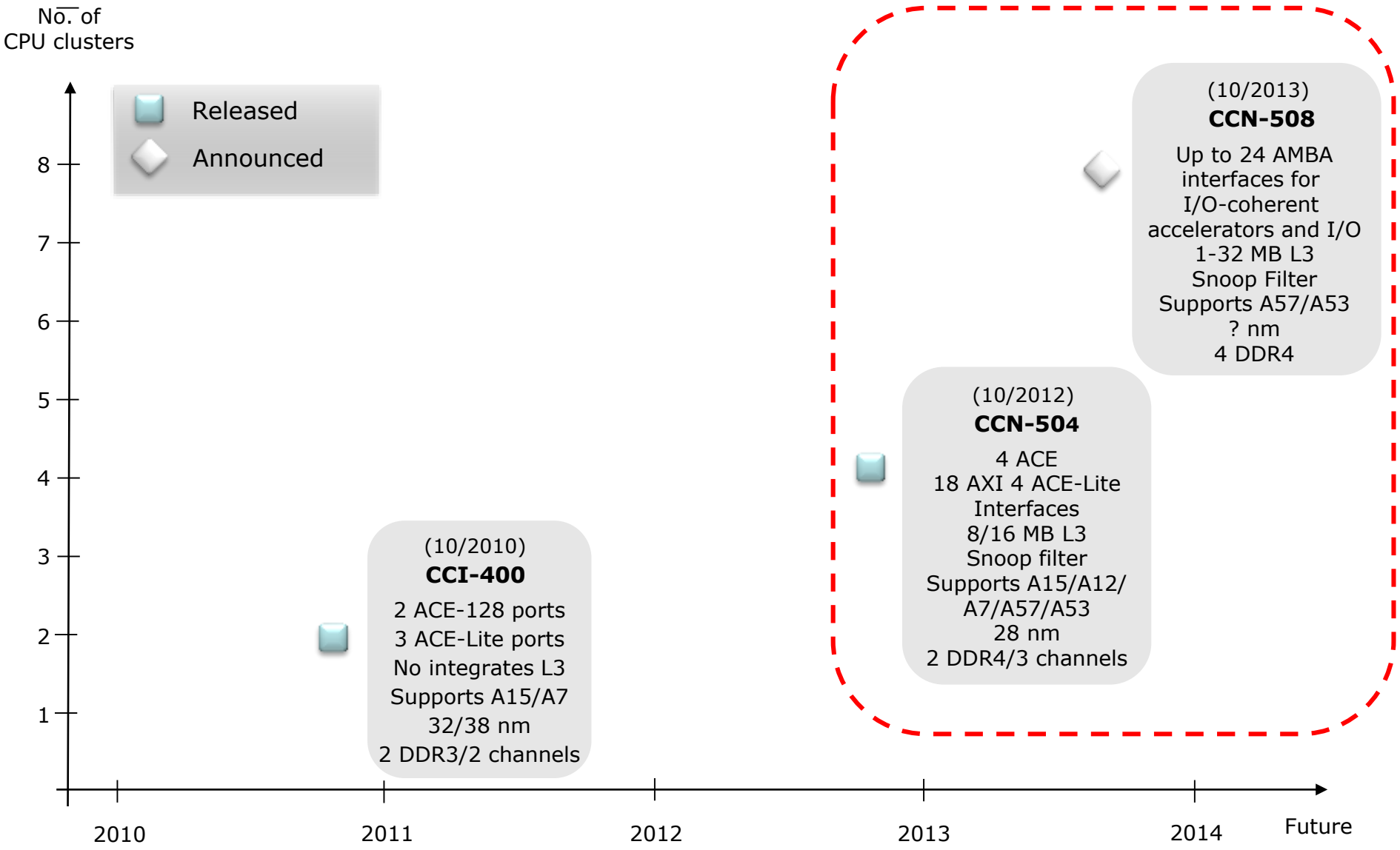
- Became available in 10/2012.
- Key feature: It allows the integration of up to 4 coherent CPU clusters for up to 16 cores.
- It is part of the CoreLink 500 System, as indicated in the next Table.

The ring interconnect fabric of the CCN-504 (dubbed Dickens) []



http://www.theregister.co.uk/2014/05/06/arm_corelink_ccn_5xx_on_chip_interconnect_microarchitecture/?page=2

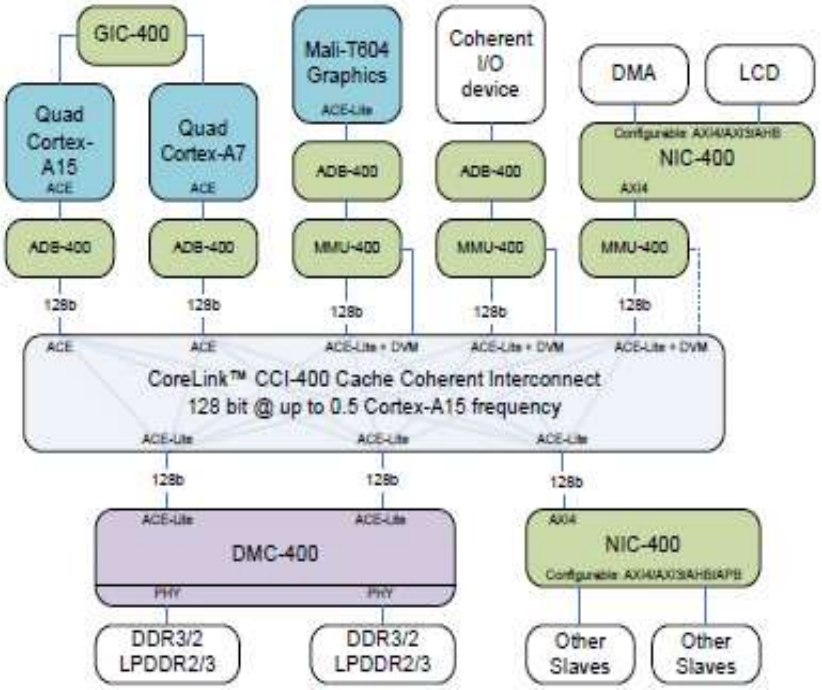
Main features of ARM's 2. generation cache coherent interconnects

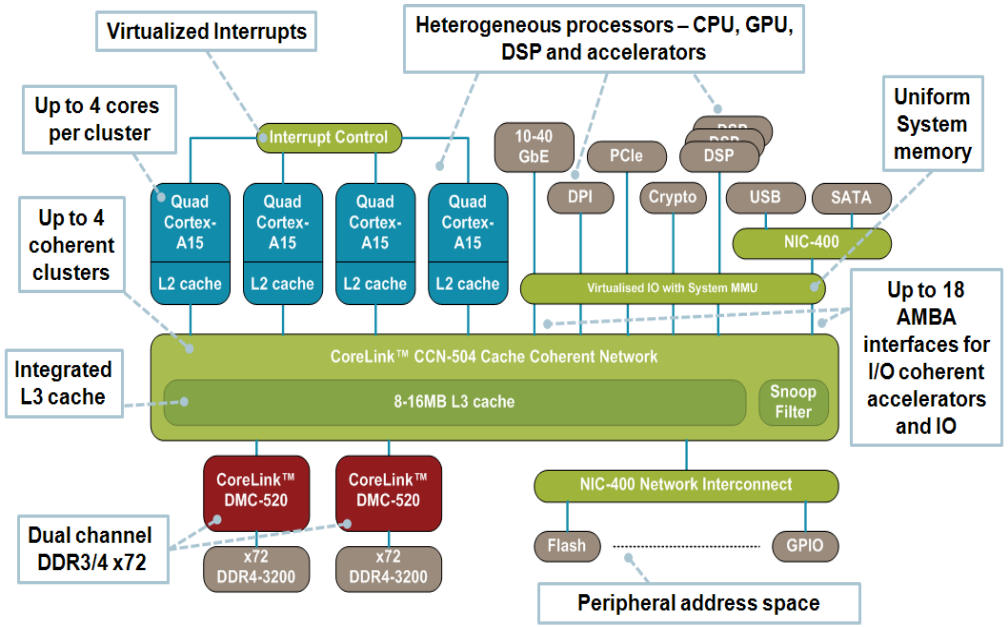


Main features of the CCN-504 Cache Coherent Network

Main features	CCN-504	CCN-508
Number of master ports for CPU clusters	4 (128-bit ACE or CHI)	8 (CHI)
AMBA interfaces for I/O coherent accelerators and I/O (ACE-Lite AXI3/4)	18	24
Supported processors (Cortex-Ax)	A15/A7/A12/A57/A53	A57/A53
Integrated L3 cache	8-16 MB	1-32 MB
Integrated snoop filter	Yes	Yes
Support of memory channels	2x DDR3/4	4x DDR3/4
Interconnect topology	Ring (Dickens)	Ring
Technology	28 nm	n.a.

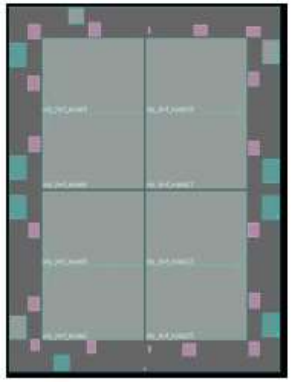
Comparing the CCI-400 and the CCN-504 cache coherent interconnects []



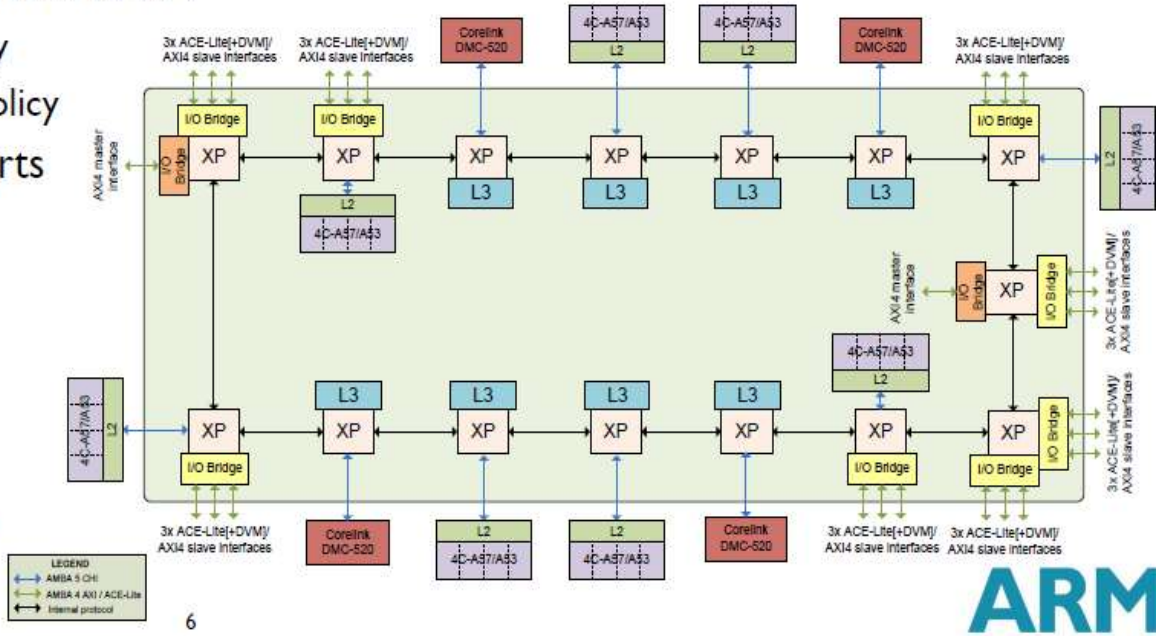


	CCI-400	CCN-504
Coherent CPU Clusters	2(Up to 8 CPUs)	4(Up to 16 CPUs)
ACE-Lite bus slave sockets	2(Fixed)	Up to 16(Configurable)
ACE-Lite bus master sockets	3(Fixed)	3(Fixed)
Level-3 Cache	N/A	Integrated(Configurable)

CoreLink CCN-508 System Architecture



- 8x processor clusters (32-CPU system), 4 25GB/s memory-controller ports
- Transport Layer – fully distributed, implementation-aware design
  - Core-frequency, single-cycle transport - five bi-directional rings supporting AMBA 5 CHI virtual channels
  - 4x128-bit data rings - 2.9Tbps peak, 1.8Tbps sustained throughput
- L3 + PoC/PoS + Snoop-filter: 8x partitions, 1M-32M total capacity
  - Ordering/coherency point(s) for accesses to DRAM
  - Snoop-filtering for coherency scalability
  - Adaptive exclusion/inclusion caching policy
- 8x I/O-master bridges – 24 ACE-Lite ports
  - I/O coherent w/ SMMU support
  - 320+Gbps usable bandwidth / bridge
  - Uses L3 for flexible I/O caching
- Integrated system-clocking support
  - Source-synchronous async bridges
  - Flexible clock-ratios, timing-mitigation

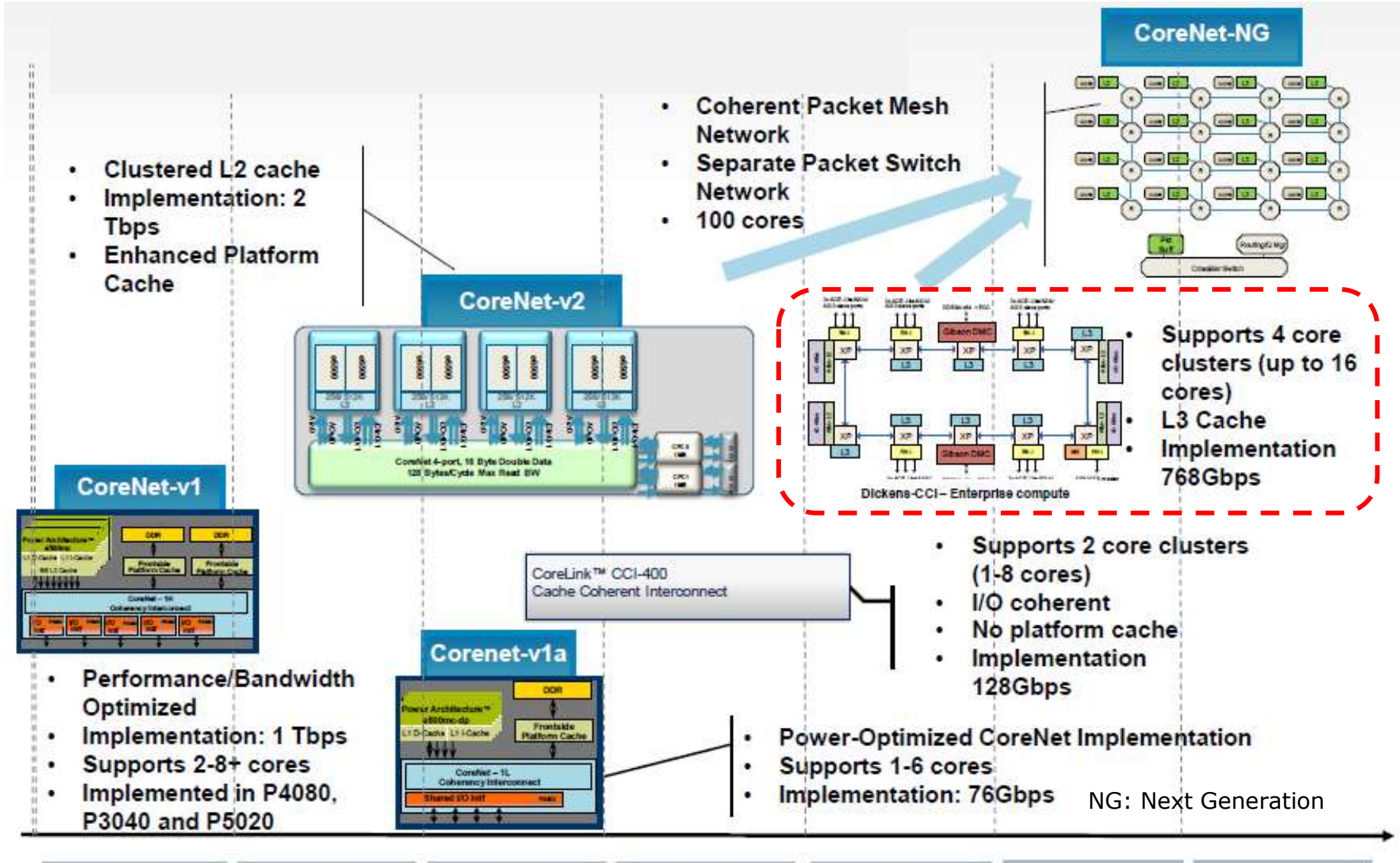


AVAGO TECHNOLOGIES

ARM

http://www.hotchips.org/wp-content/uploads/hc_archives/hc26/HC26-11-day1-epub/HC26.11-4-ARM-Servers-epub/HC26.11.420-Network-Flippo-ARM_LSI%20HC2014%20v0.12.pdf

Coherent interconnect roadmap of Freescale []



http://www.freescale.com/files/training/doc/dwf/DWF13_APF_NET_T0795.pdf Apr. 2013

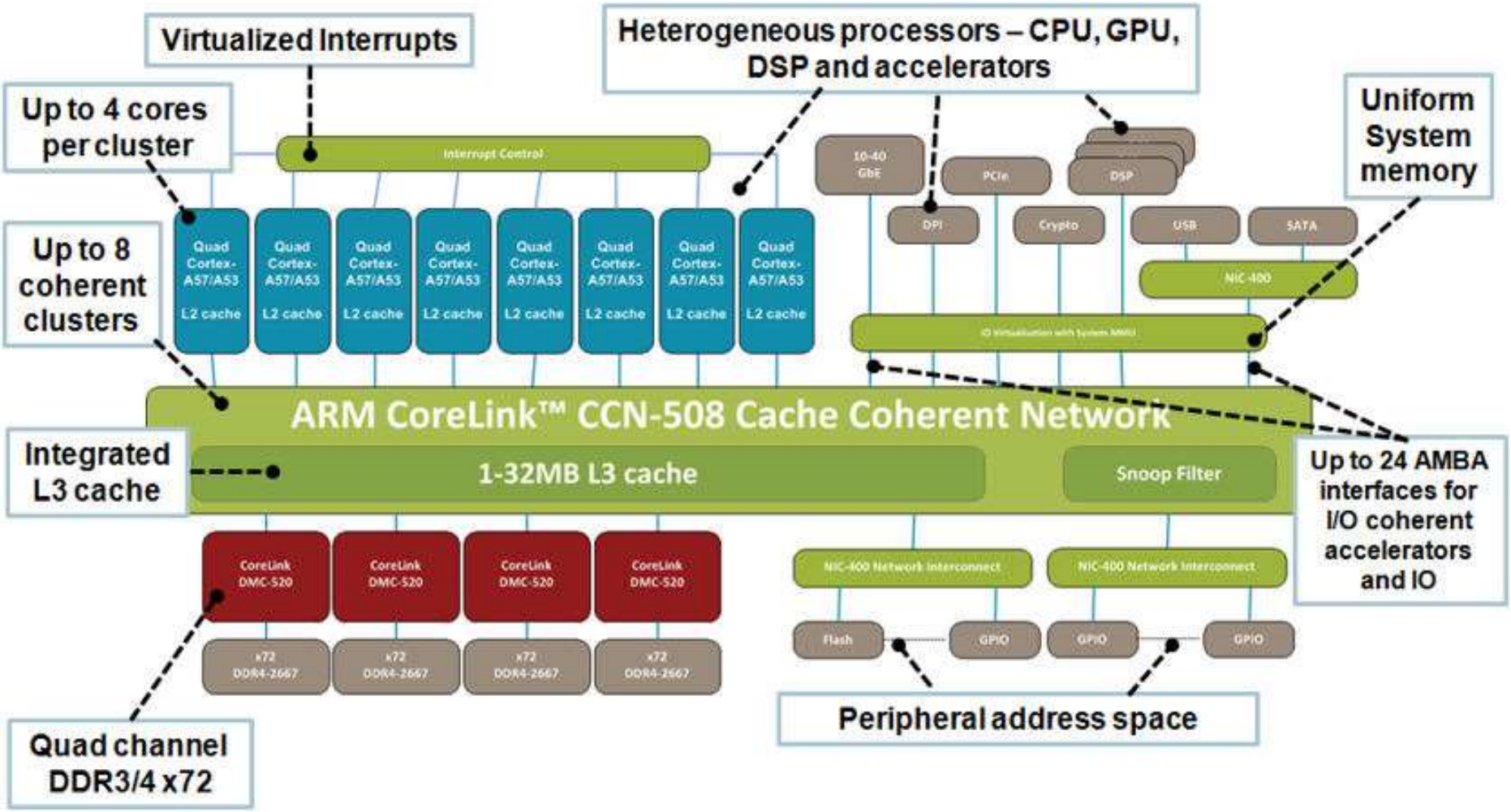
2.1 The CCN-508 2. generation cache coherent interconnect

- Announced in 10/2013
- Key feature: It allows the integration of up to 8 coherent CPU clusters for up to 32 cores.

Main features of the CCN-5xx Cache Coherent Networks

Main features	CCN_502	CCN-504	CCN-508	CCN-512
Date of introduction	12/2014	10/12 Availability in 2013	10/2013	10/2014
Number of master ports for CPU clusters	4 (CHI)	4 (CHI)	8 (CHI)	12 (CHI)
AMBA interfaces for I/O coherent accelerators and I/O (ACE-Lite AXI3/4)	9	18	24	24
Supported processors (Cortex-Ax)	A57/A53	A15/A57/A53	A57/A53	A57/A53
Integrated L3 cache	0-8 MB	1-16 MB	1-32 MB	1-32 MB
Integrated snoop filter	Yes	Yes	Yes	Yes
Support of memory channels up to	4x DDR4/3	2x DDR4/3	4x DDR4/3	4 x DDR4/3
Interconnect topology	Ring?	Ring (Dickens)	Ring	Ring
Sustained interconnect bandwidth	0.8 Tbps	1 Tbps	1.28 Tbps	1.8 Tbps
Technology	n.a.	28 nm	n.a.	n.a.

Example: Cache coherent SOC based on the CCN-508 interconnect []

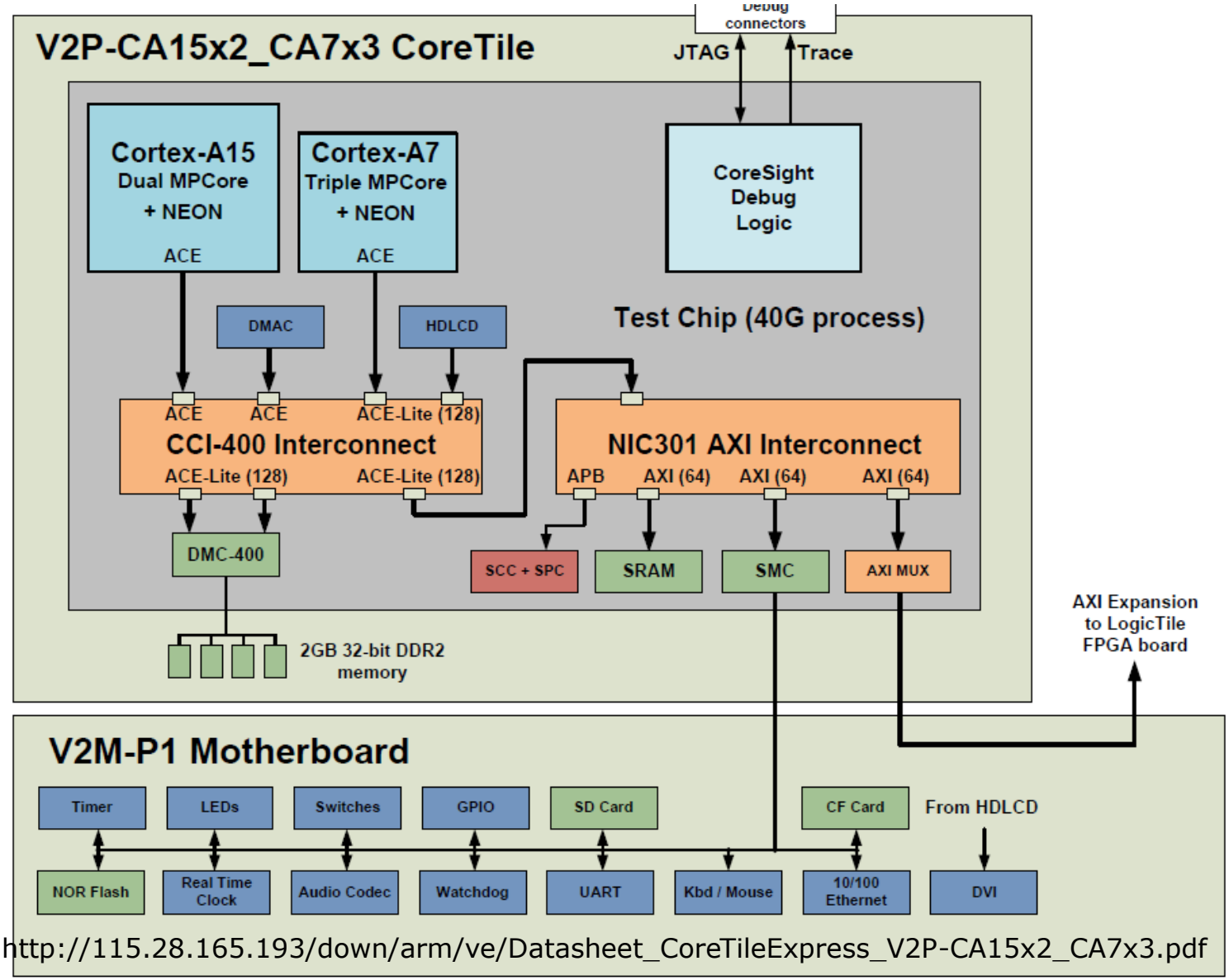


http://www.arm.com/products/system-ip/interconnect/corelink-ccn-508.php

Intermixing cache coherent and non-coherent subsystems

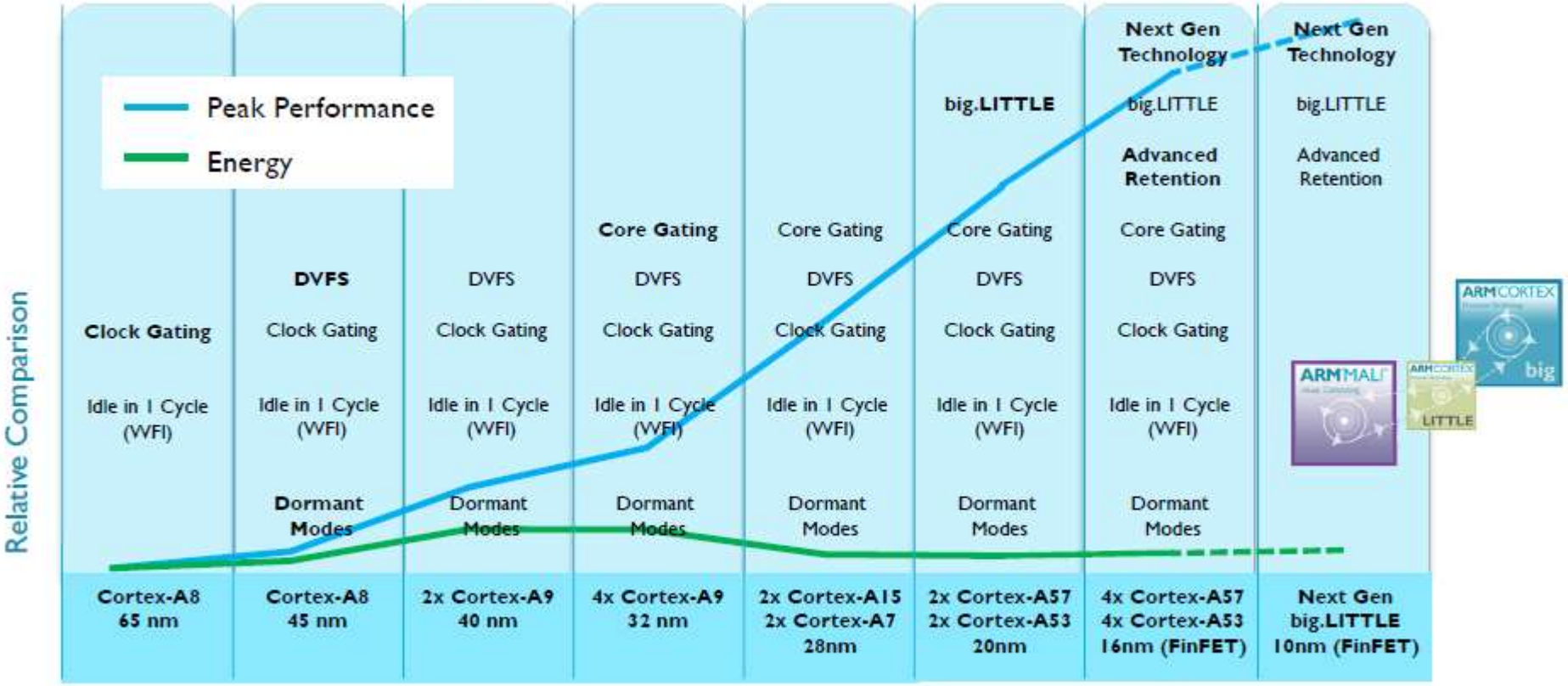
It is feasible to implement cache coherent subsystems (based on the cache coherent interconnects e.g. CCI-400 or CCN-504/CCN-508) and non cache coherent subsystems (based on the non cache coherent interconnects NIC-301/NIC-400) to implement a SOC, as the next Figure shows.

SOC consisting of a cache coherent and a non cache coherent subsystem []



http://115.28.165.193/down/arm/ve/Datasheet_CoreTileExpress_V2P-CA15x2_CA7x3.pdf

Advanced Technologies Deliver Best Power and Performance



ARM