

# Heterogeneous Multi-Processor Coherent Interconnect

Kai Chirca, Matthew Pierson

**Processors, Texas Instruments Inc, Dallas TX**

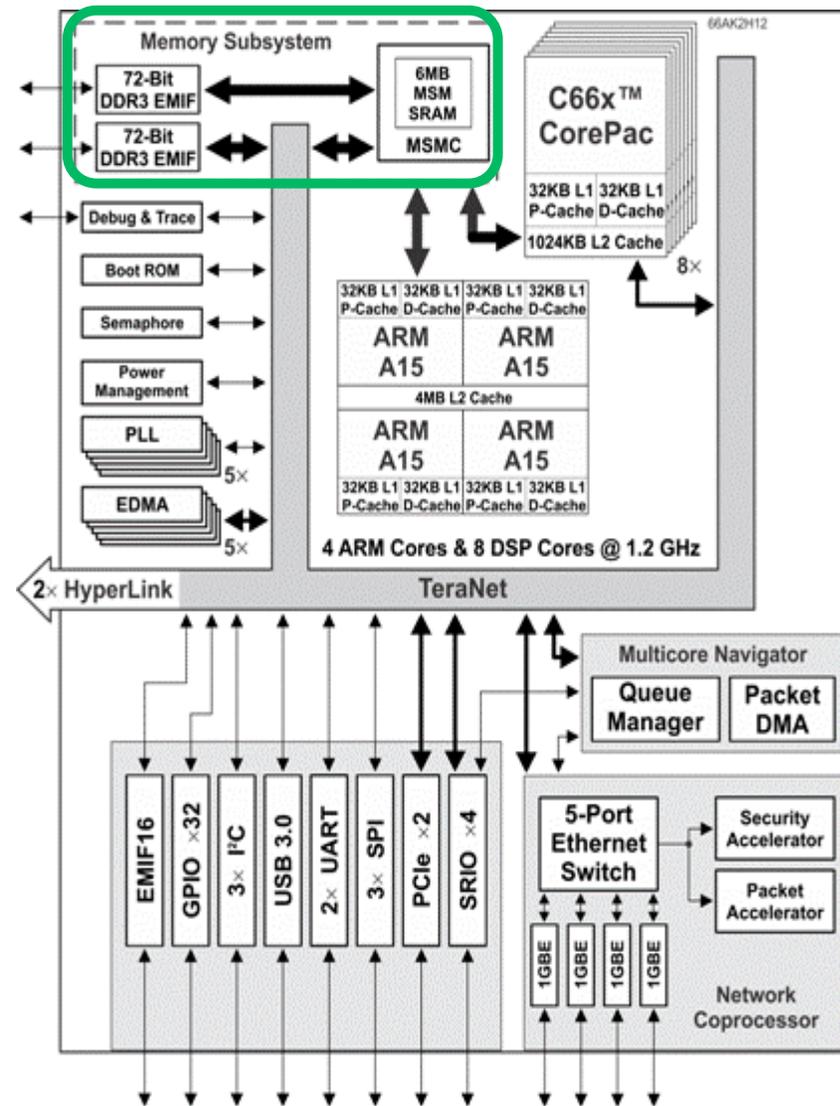
# Agenda

- ❑ TI KeyStonell Architecture and MSMC (Multicore Shared Memory Controller)
- ❑ Crossbar Architecture
  - ❑ Topology and Scalability
- ❑ Heterogeneous Processor Connection
- ❑ Bandwidth Management of Shared Resources
- ❑ IO Coherence
- ❑ Memory Protection and Address Translation (MPAX)
- ❑ MSMC Scaling – Multi-MSMC Topology

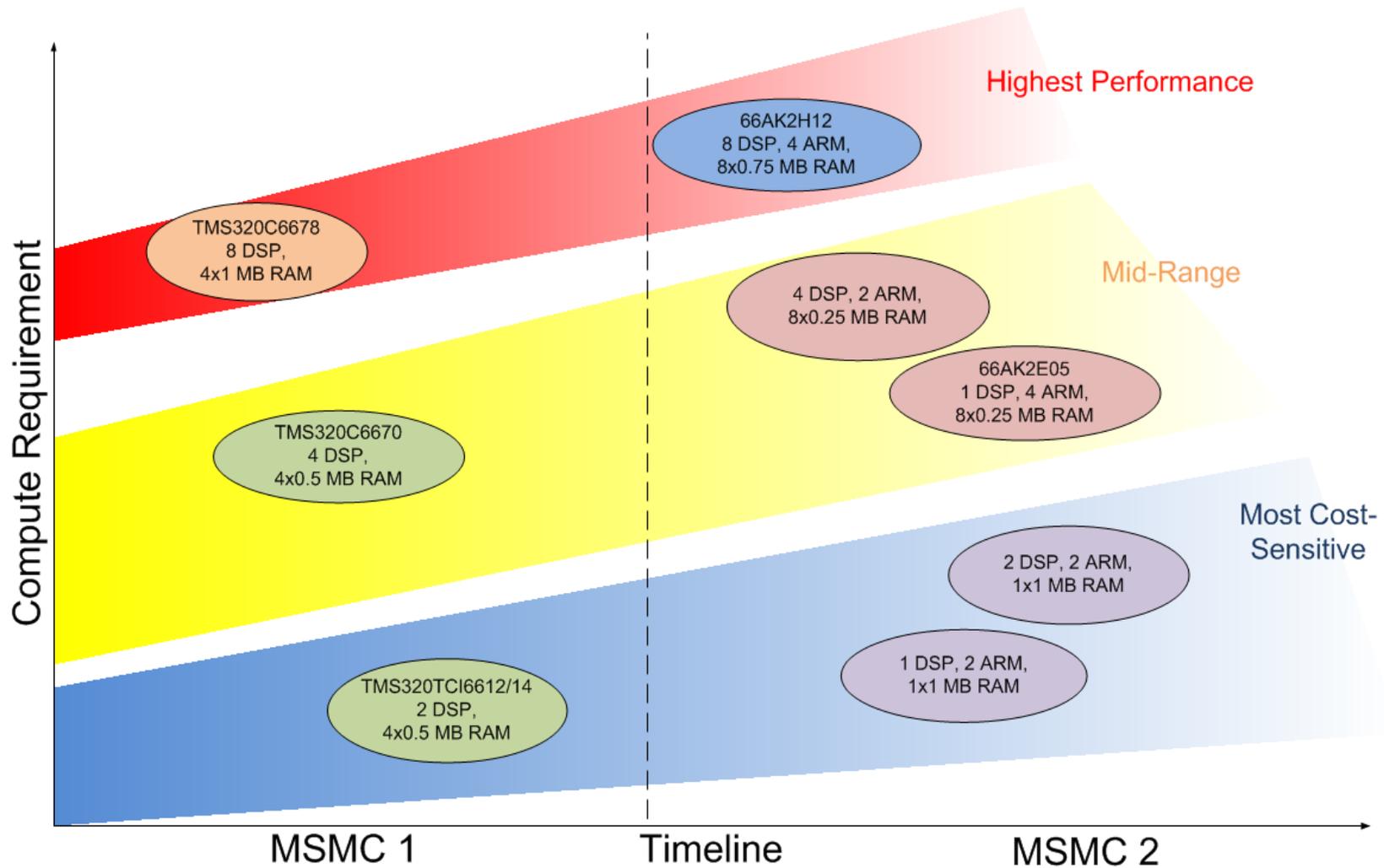


# TI KeyStone II Multicore ARM + DSP

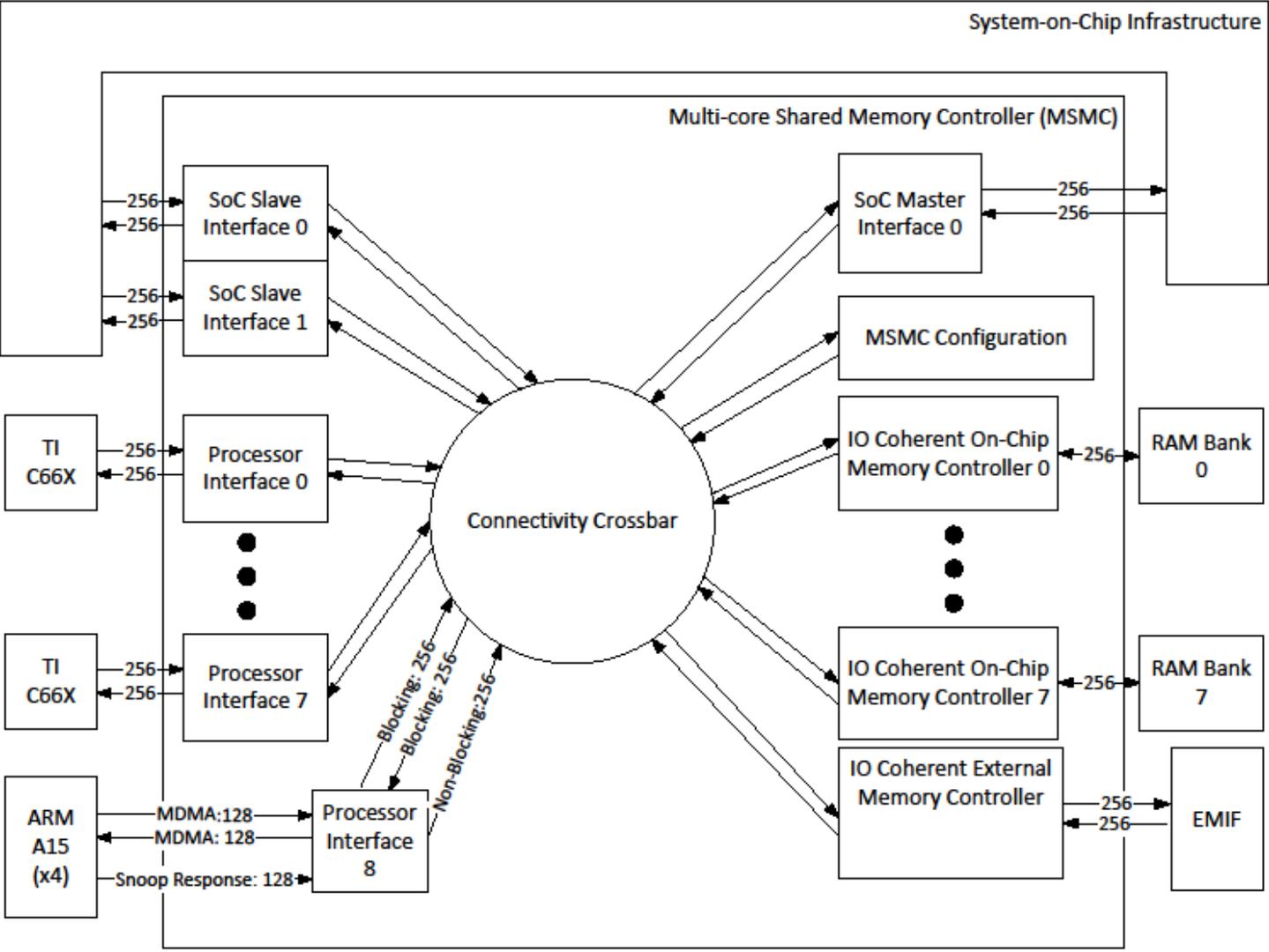
- Deliver the highest performance at the lowest power levels and costs
  - Up to 5.6GHz of ARM
  - Up to 9.6GHz of DSP processing
- Optimal for embedded applications like
  - Cloud computing
  - Media processing
  - High-performance computing
  - Transcoding, security, gaming, analytics, and virtual desktop infrastructure.
- KeyStone II architecture is the first TI platform to offer
  - Standalone quad ARM® Cortex™-A15
  - Combined ARM Cortex-A15 and TMS320C66x high performance DSP solutions, for embedded infrastructure applications.
- MSMC (multicore shared Memory Controller) is the heterogeneous processor compute cluster platform



# TI KeyStoneII Devices and MSMC RoadMap

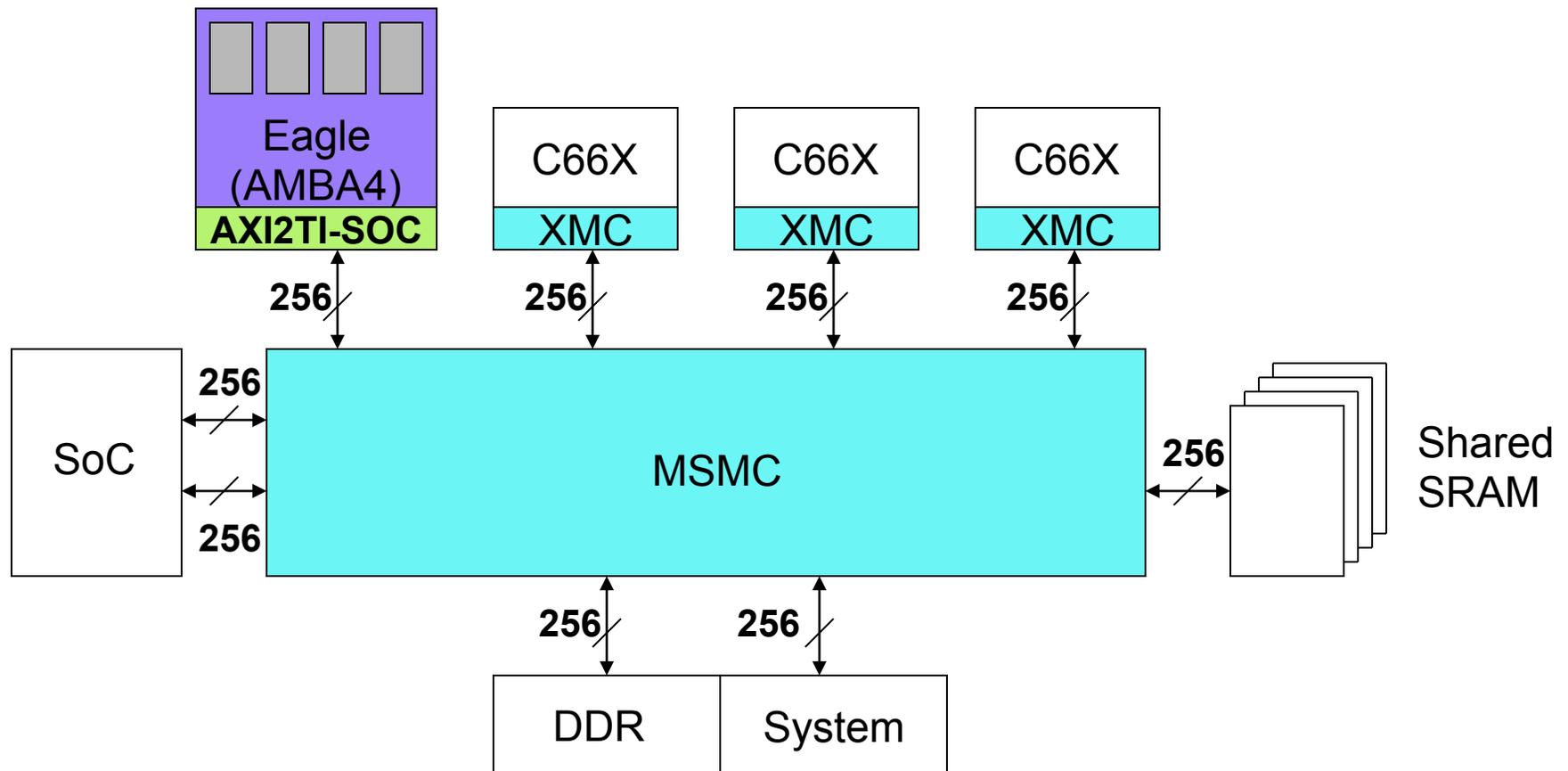


# MSMC Crossbar Topology

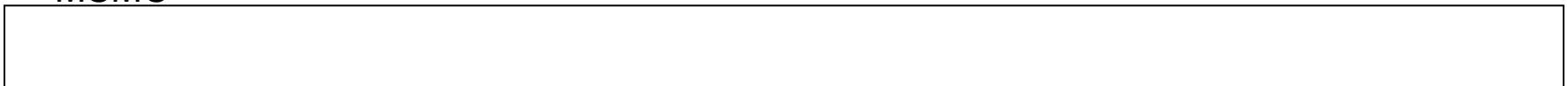


# Heterogeneous Processor Connection

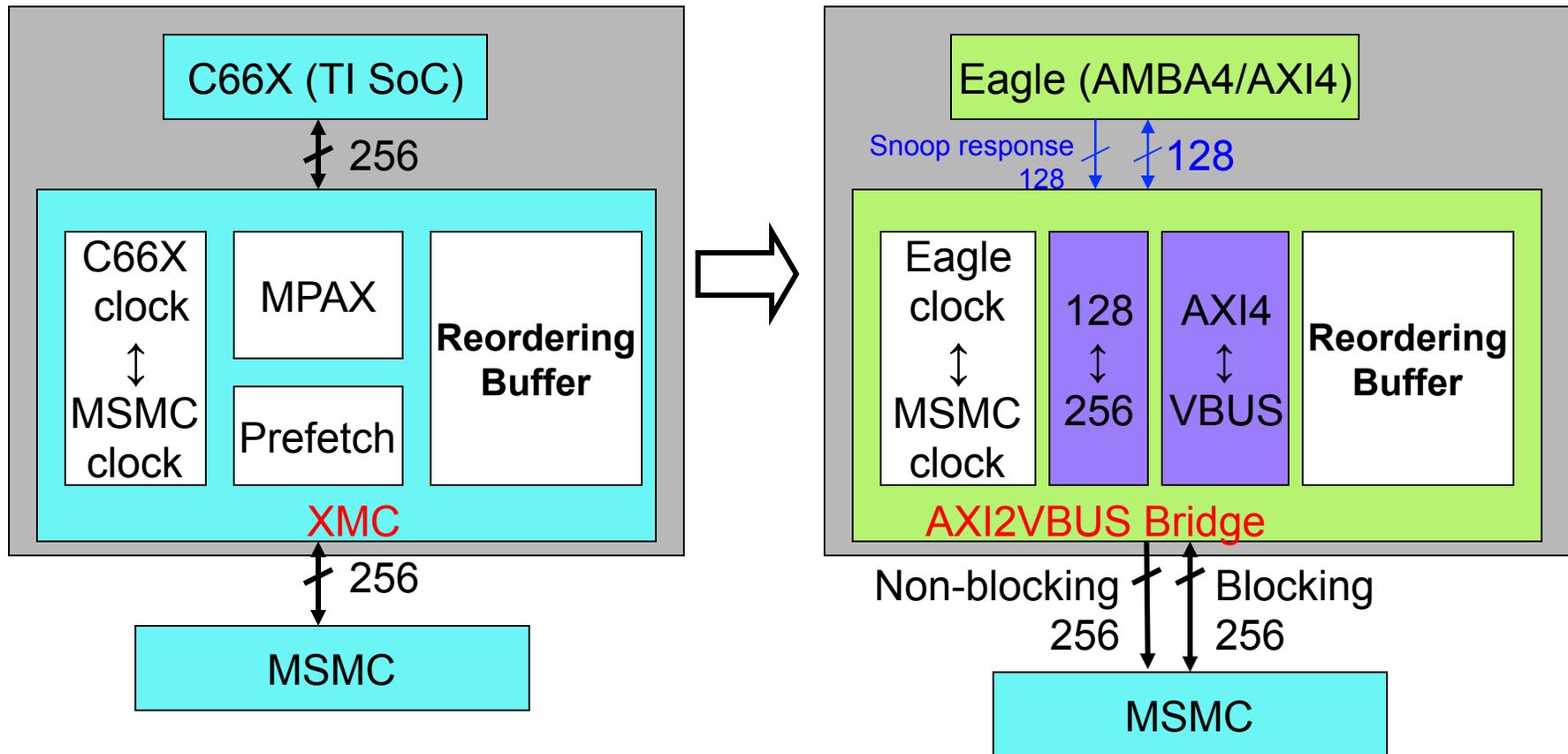
# DSP Only Configuration



- C66X → Eagle
- XMC → AXI2TI-SoC bridge
- AXI2TI-SoC bridge supports flexible clock/width/protocol conversion between Eagle and MSMC



# Eagle/MSMC Bridge



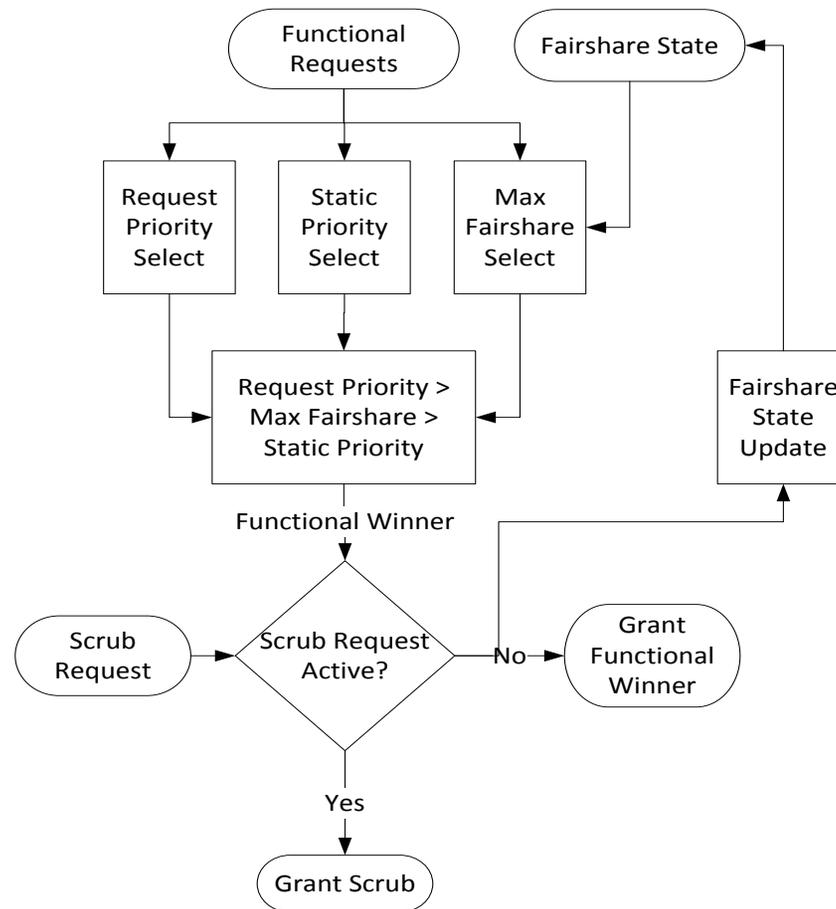
- AXI2TI-SoC bridge w/ Eagle performance optimization
- Provides the identical TI SoC interface as XMC on MSMC side
- Provides asynchronous/synchronous clock conversion on the fly



# Bandwidth Management

# Arbitration

MSMC provides a multi-level, user programmable independent arbiter for each endpoint for when masters attempt to access the same endpoint in a given cycle.



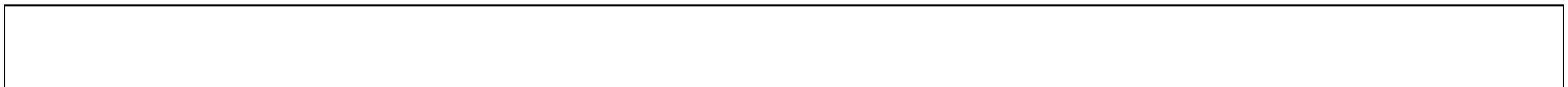
# Arbitration Layer One: Request Priority

## Transaction Encoded Priority

Each request sent to MSMC carries a 3-bit encoded priority level used by the first level of the arbiter. MSMC's first level of arbitration will select all requests with the highest priority.

## Starvation Bound

The user can program each masters starvation limit via memory mapped registers. Every cycle that an active request fails arbitration its starvation count decrements. When that counter reaches zero, MSMC elevates that request's priority to the highest available level. Once that request wins arbitration, that requestor's counter is reset to the starvation limit.

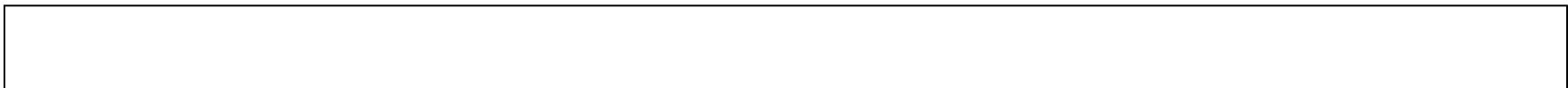


# Arbitration Layer Two: Fairshare

Once all of the highest priority requestors are selected, MSMC selects winner(s) based on the internally maintained Fairshare state. Each endpoint arbiter maintains this state information for all connected requestors.

1. The arbiter selects the competing master with the highest Fairshare value.
2. Once the arbiter selects a winner, MSMC updates the Fairshare state for all masters competing at layer two.
  - Winner: Fairshare state decrements by  $N-1$ , where  $N$  is the number of competing masters
  - Losers: Fairshare state increments by 1, biasing the next arbitration further in their direction
  - Fairshare counts only updated when masters at the same priority level compete
  - Fairshare credit total among all masters for a given endpoint always 0

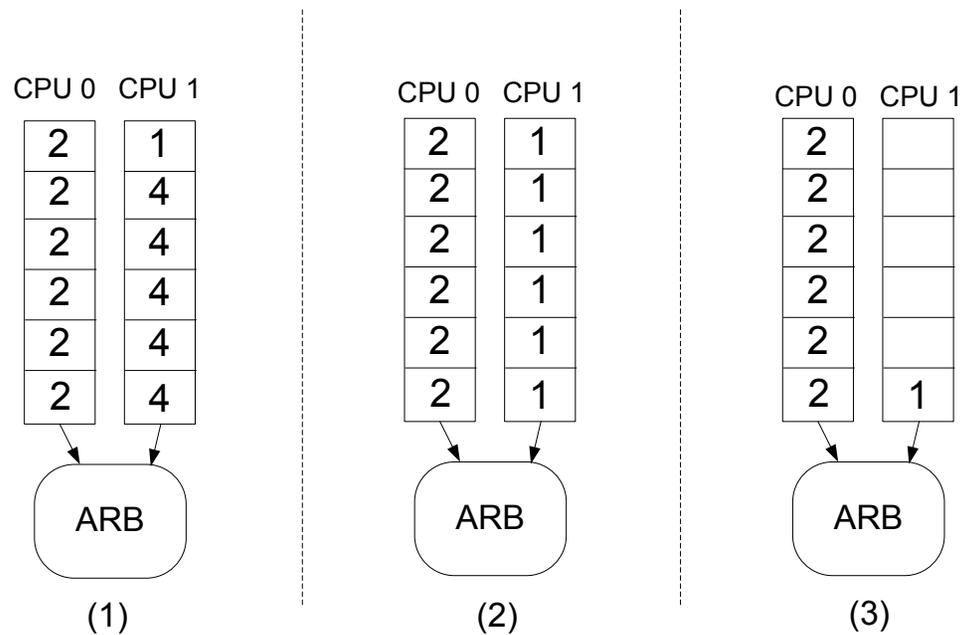
Cycle	Competing Masters	Current States	Winner	Next States	Credit Total
1	A, B, C, D	0,0,0,0	A	-3, +1, +1, +1	0
2	B, C, D	-3, +1, +1, +1	B	-3, -1, +2, +2	0
3	A, C, D	-3, -1, +2, +2	C	-2, -1, 0, +3	0
4	A, B, D	-2, -1, 0, +3	D	-1, 0, 0, +1	0



# Eliminating Head-of-Line Blocking

Multiple MSMC features to avoid head-of-line blocking/priority inversion:

1. Priority Escalation – Subsequent higher priority transactions ‘deputize’ earlier transactions with higher priority until they are out of the way.



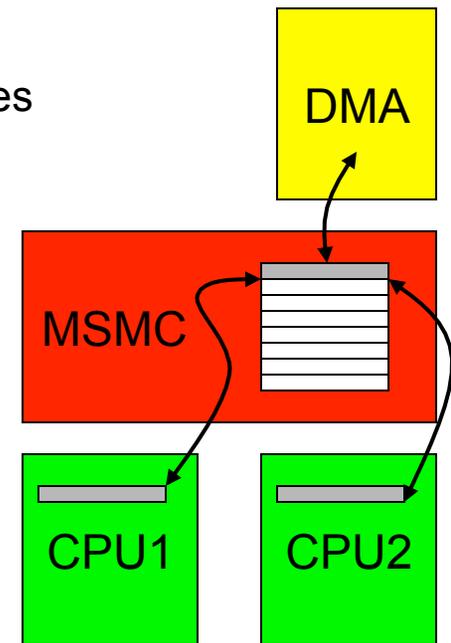
2. Prefetch Queueing and Squashing – Always move prefetches off of the interface into side queues and eliminate old prefetches when queue fills.



# IO-Coherence Support

# Cache Coherence Problem

- Cache systems allow  $n$  copies of same memory location stored around the chip
  - Modifications made locally with no broadcast
  - Modified lines stored indefinitely before write-back to main memory
  - Typically, the entire cache block is written back to main memory even if only a part of the block was modified
- Issues
  - Inconsistent memory commit observation across different cores
  - Unknown latency until other cores see updates
  - Cached copies not updated when third party (DMA) updates main memory
  - Any sharing of cache blocks can cause write commit corruption, even in the absence of true sharing
- Legacy solution is software coherence
  - Manual Cache Maintenance Operations
  - Software Inter-processor Synchronization
  - **Time consuming and error prone**



# MSMC IO-Coherence Examples

MSMC maintains IO-Coherence for all shared accesses to MSMC SRAM and the external memory directly connected to MSMC

- One cache coherent ARM A15 cluster
- Two SoC infrastructure connections

The following slides show some example coherent interactions.

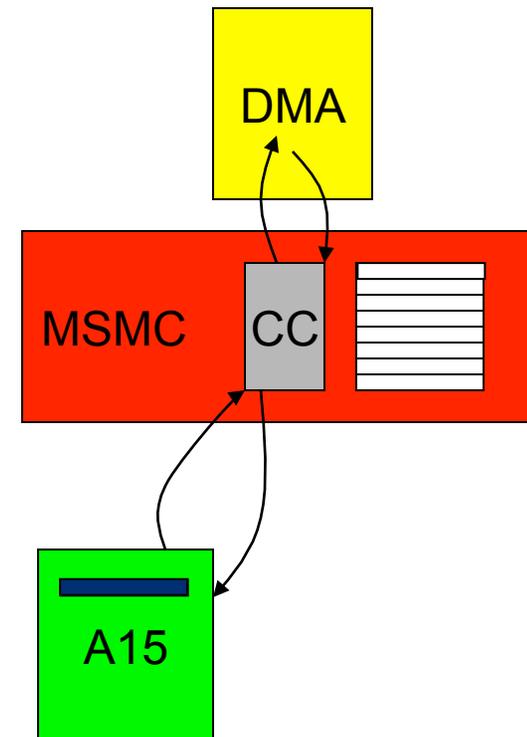
The colored blocks represent coherent memory blocks and their corresponding MOESI state at different points in time.

-  **Modified**
-  **Owned**
-  **Exclusive**
-  **Shared**
- Invalid**

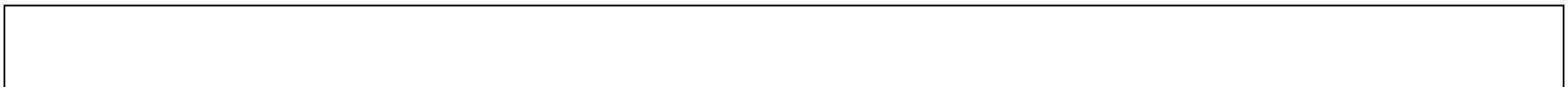


# SoC Read to Shared Space with Dirty Cache State

1. A15 has memory block A in Modified state
2. DMA issues sharable read to A
3. MSMC issues read snoop for A to A15
  - Optimization: In parallel with snoop, MSMC speculatively reads endpoint memory
4. A15 responds with updated data
5. MSMC combines snoop response with speculative read data and returns final read response to DMA

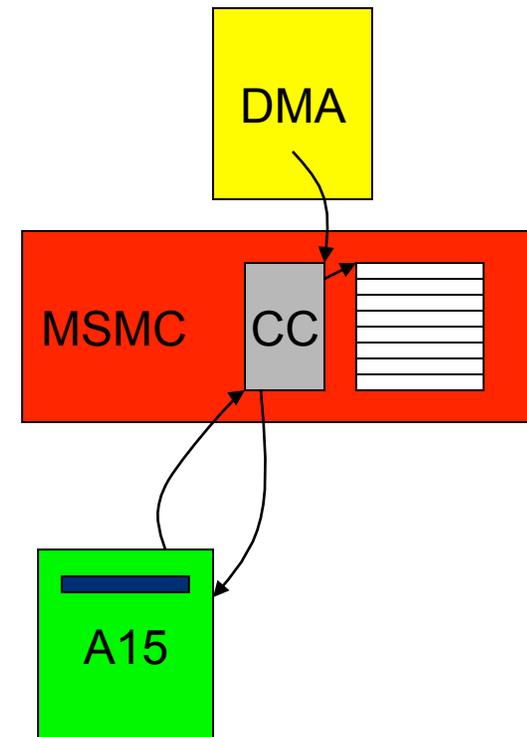


**Speculatively reading the endpoint memory allows the snoop latency and endpoint read latency to overlap, improving performance**



# SoC Write to Shared Space with Dirty Cache State

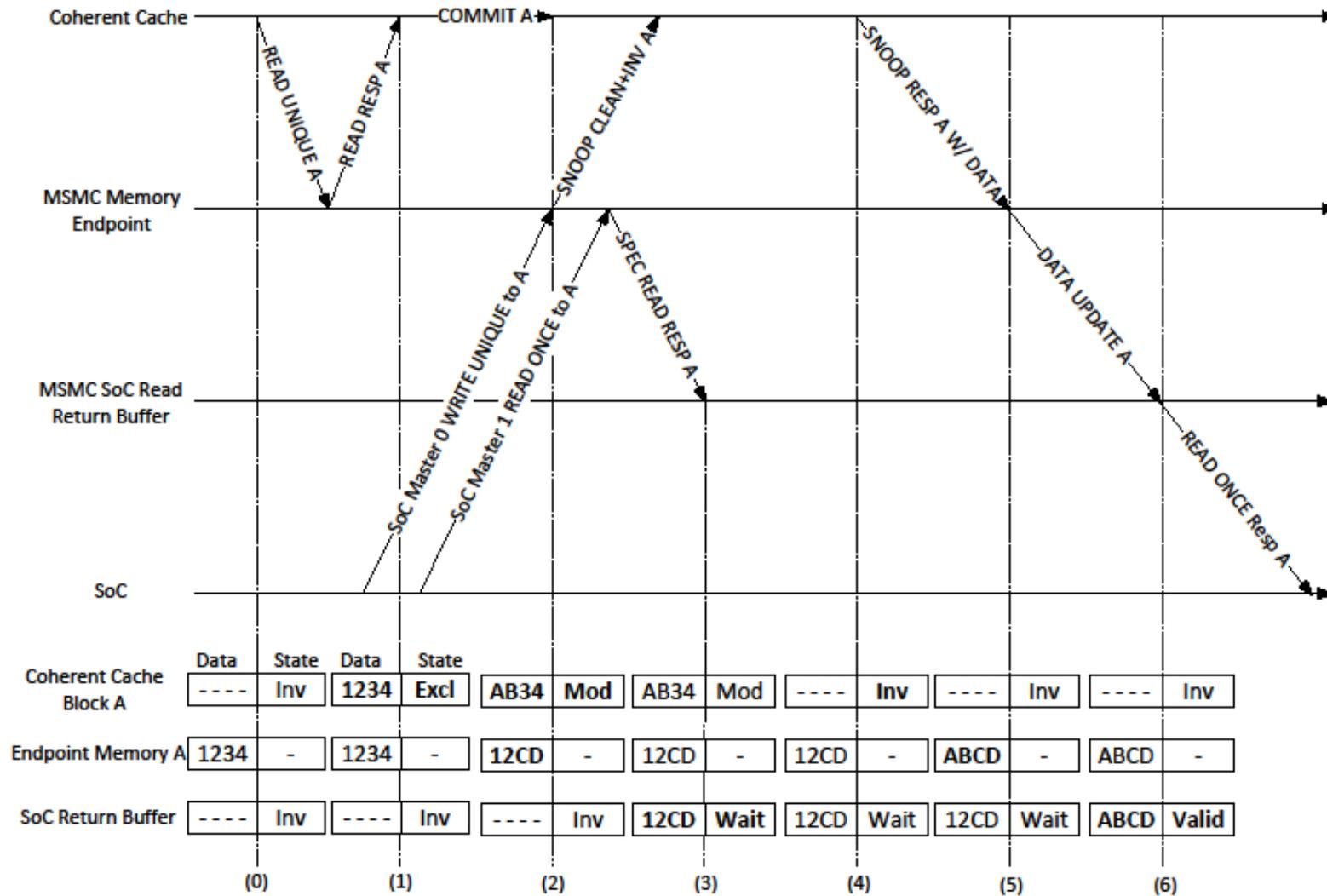
1. A15 has shared memory block A in the Modified state
2. DMA issues write to memory block A
3. MSMC issues invalidate-and-writeback snoop to A15
  - Optimization: MSMC speculatively commits write data to endpoint memory, saves updated data lanes
4. A15 responds to snoop including updated data
5. MSMC masks out stale byte lanes in snoop response data and commits to memory



**Early write commit by MSMC reduces extra required coherent write storage space by 8x and eliminates unnecessary memory stalls**



# Coherent Writes with Read Sync Example



# Memory Protection and Address Extension (MPAX)

# MPAX High-Level

1. MSMC natively operates on 40-bit addresses providing 1 tera-byte of physically addressable memory space.
2. C66X CPU, ARM A15 CPU, and Keystone2 SoC infrastructure natively operate on 32-bit addresses (4 giga-bytes of addressable memory space).

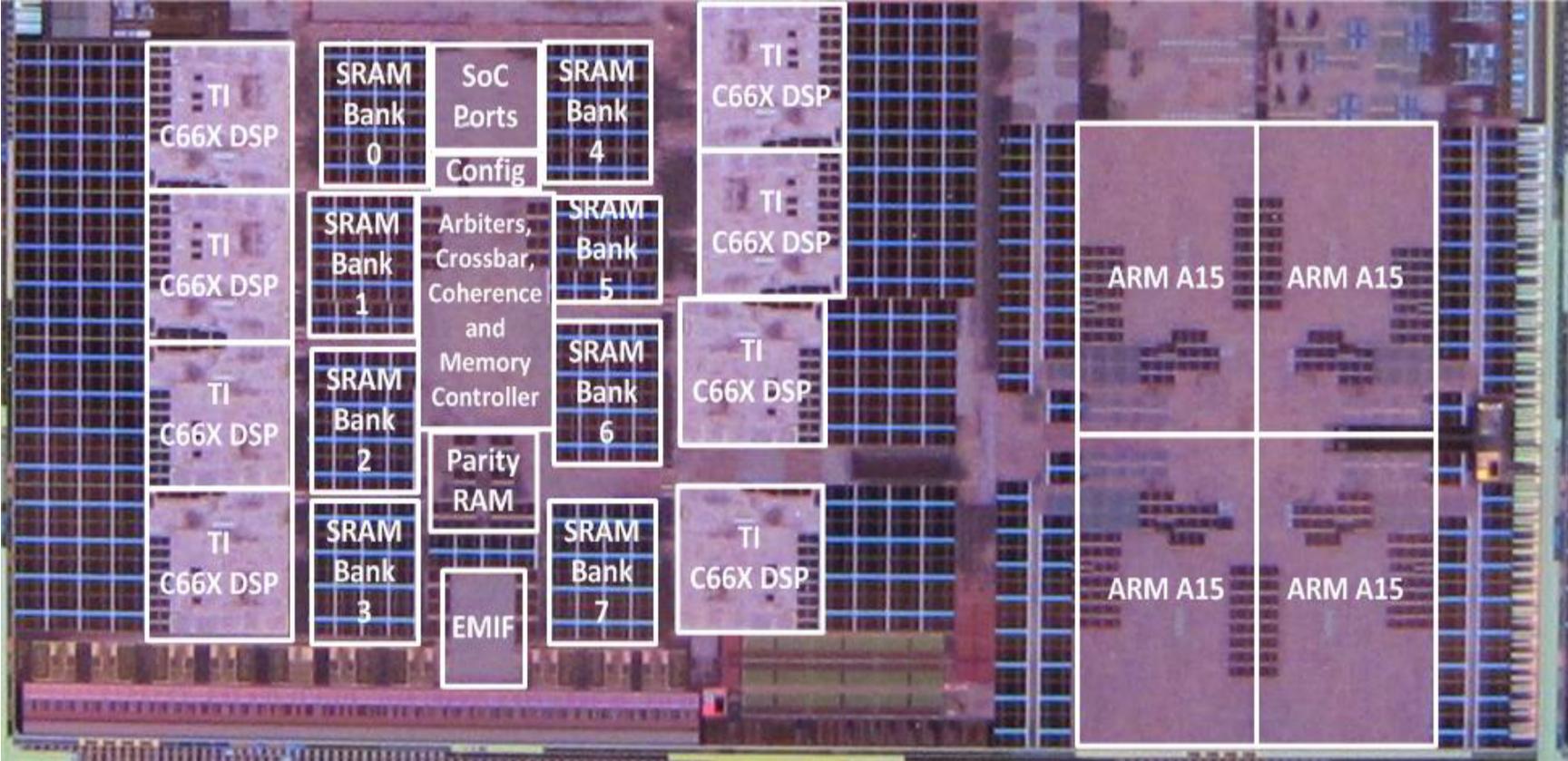
Basic Requirement: All of these 32-bit components need their address translated to 40-bits to fully utilize the MSMC memory space.

In addition to this basic requirement, MPAX provides

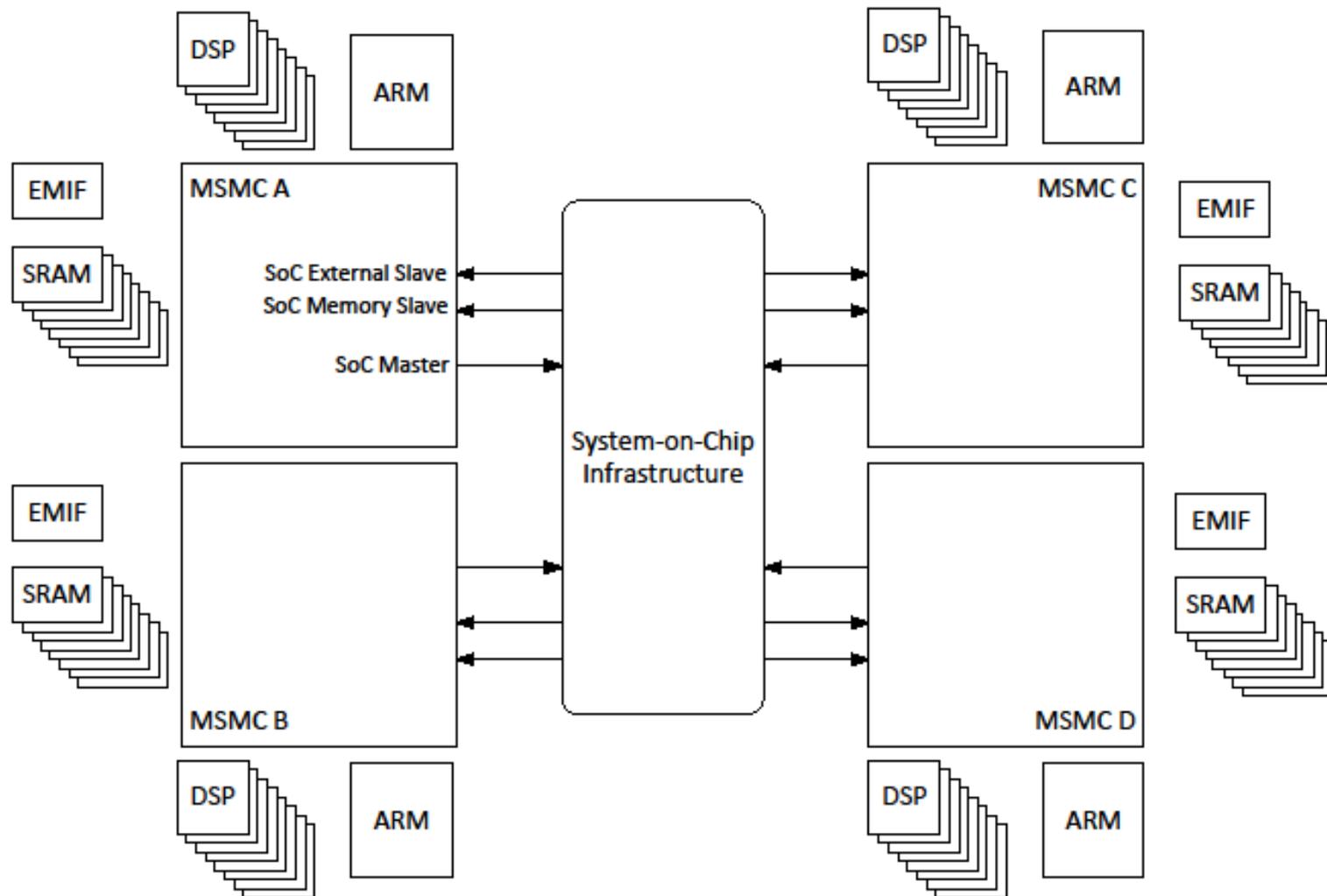
1. Address Virtualization – Incoming addresses (virtual) matched to user-defined memory partitions to determine final address (physical)
2. Memory protection – User/Supervisor/Emulation, Read/Write/Execute, Secure/Non-Secure control for each memory partition
3. Share-ability for each memory partition
4. Separate virtualized address view for each VBUSM privID



# MSMC Die Photo



# MSMC Scaling – Multi-MSMC SoC



# Conclusion

- KeyStone II's Multicore Shared Memory Controller (MSMC) allows all on-chip memories to operate at the same clock rate as the cores
  - Very low latency, high bandwidth interconnects among all cores all on-chip memories and all off-chip DDR memories
  - Features 2.8 Tbps of throughput and switching capacity
  - Provide 256-bit interface for ARM integration
- Connects directly to external DDR3 memory which reduces the latencies typically associated with external memory accesses
  - In a typical use case, most of the operating system code and data reside in external memory. With the MSMC, each access to the external memory requires far fewer CPU cycles, significantly increasing application performance
- Support non-blocking data flow among all of the CPU cores in the architecture which enables processing elements to operate near full capacity
- Provide full scalability for heterogeneous platform
  - 0 MB to 6 MB of Level 3 SRAM shared by all ARM and DSP CorePacs, configurable number of banks
  - Configurable number of DSP and ARM cores
- Cache coherency is maintained in MSMC hardware for all memories that are accessed by Cortex-A15 processors via the MSMC, thus eliminating the need for software to maintain ARM processor-to-processor and processor to-I/O memory traffic coherency.
- All memories instances in MSMC are ECC protected.
- Memory protection and address extension
- Firewall protection



# Q&A

Thank you

[kai@ti.com](mailto:kai@ti.com)



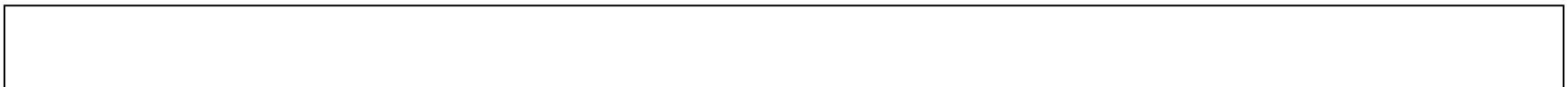
# Error Detection and Correction (EDC)

# MSMC Error Detection and Correction

MSMC protects on-chip SRAM data using hamming EDC codes capable of correcting a single-bit error and detecting any double-bit error (SEC/DED).

## Basics

- MSMC calculates an 11-bit parity signature to store alongside each 32-byte data word in the on-chip SRAM
  - 10 bits of this represent the SEC/DED hamming code
  - 1 bit represents overall parity signature valid
- When a functional request reads the SRAM, MSMC decodes the output data and the output parity signature to detect if data/parity was corrupted in the memory
  - Software can control whether single bit errors are corrected or detect-only, default is correct (no latency penalty in MSMC2)
  - Double bit errors can only be detected (not corrected)
- Sub-word writes to the memory (do not write an entire aligned 32-byte window) store invalidated parity
  - Parity signature calculation requires access to all 256 protected data bits
  - Avoids performance degradation of Read-Modify-Write cycles to always store validated parity
  - These data locations with invalidated parity signatures will be 'scrubbed' later to validate the parity...



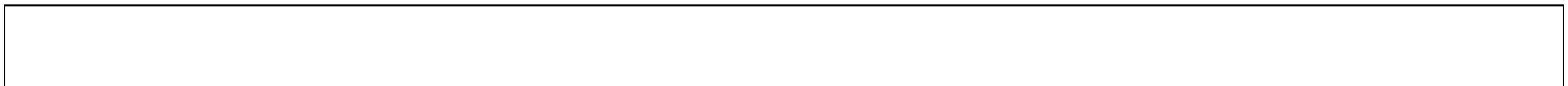
# Automatic Hardware Scrubbing

The scrubber will traverse the entire implemented memory space and perform necessary read-modify-write operations to:

1. Validate invalid parity signatures, scrubbed locations now protected
2. Correct any detected single-bit errors (valid parity required) before they possibly become uncorrectable two-bit errors
3. Detect any two-bit errors

MSMC provides software configurability and reporting through its configuration space

- Interrupts for functional and scrubbing correctable and non-correctable errors
- Counters and address logging for correctable and non-correctable scrubbing events
- Address logging for correctable and non-correctable functional EDC events
- Scrubber enable and rate control



# MPAX High-Level (cont)

Diagram (right) shows Keystone2 slave port connections to MSMC along with their address translation hardware.

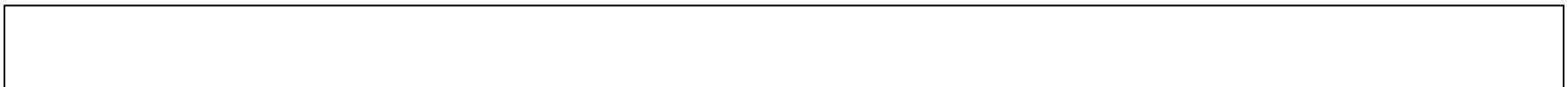
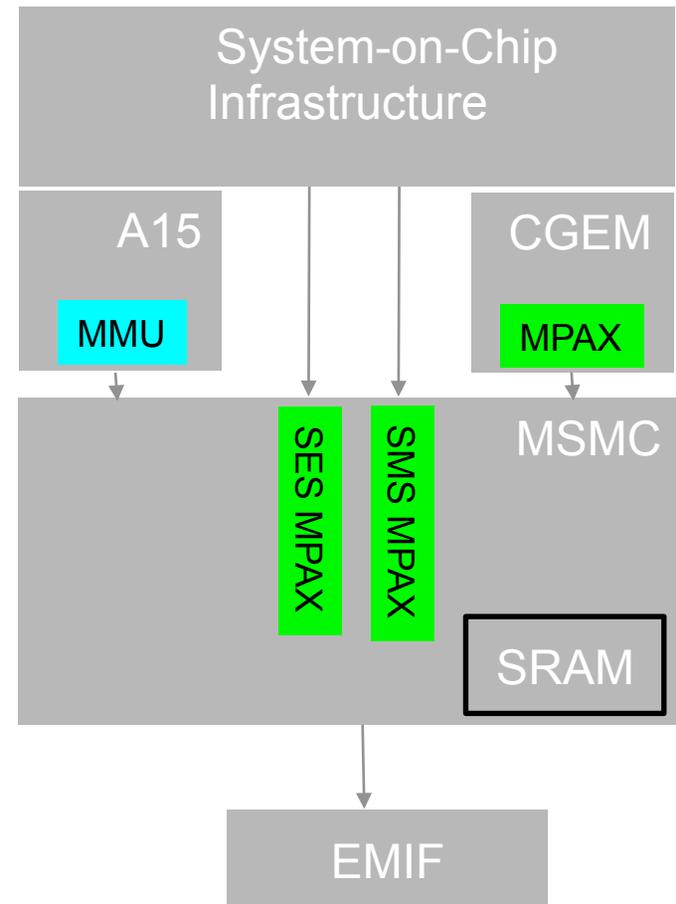
System-on-Chip infrastructure sends the following addresses to MSMC slave ports

- 0x0BC0\_0000 – 0x0BCF\_FFFF (SMS, Config)
- 0x0C00\_0000 – 0x0CFF\_FFFF (SMS, SRAM)
- 0x8000\_0000 – 0x8FFF\_FFFF (SES, External)

Configuration space addresses are not translated, but SRAM and external addresses can be setup for translation.

MSMC implements the hardware resources at these physical addresses

- 0x0:0BC0\_0000 – 0x0:0BCF\_FFFF (Config)
- 0x0:0C00\_0000 – 0x0:0CFF\_FFFF (SRAM)
- 0x1:0000\_0000 – 0xF:FFFF\_FFFF (External)



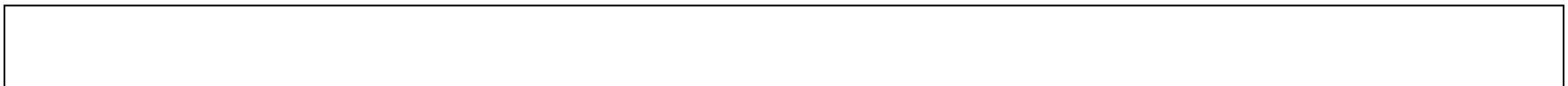
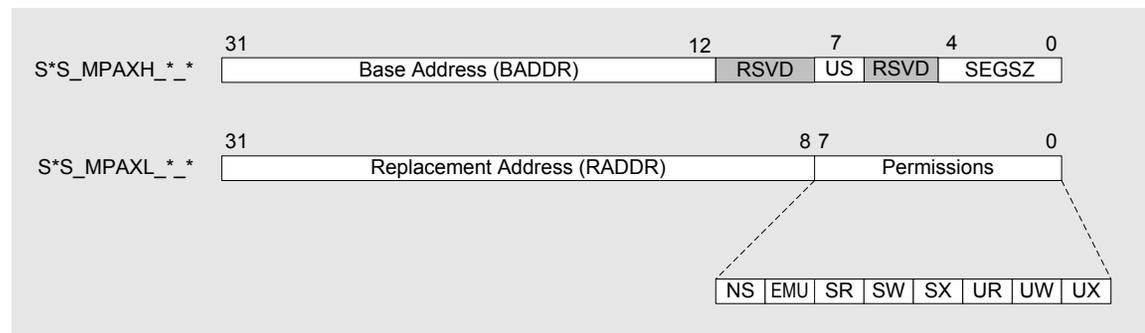
# MPAX Implementation

User configures memory partitions through memory-mapped registers residing in MSMC configuration space. Each port (SMS/SES) provides 8 memory partitions per VBUSM privID (16) totaling 256 memory partitions across both SoC slave ports.

Each memory partition is made up of two memory-mapped registers organized by port, privID, and segment number.

- SMS\_MPAXL\_0\_0, SMS\_MPAXH\_0\_0 (SMS port, privID 0, segment 0)
- SMS\_MPAXL\_1\_0, SMS\_MPAXH\_1\_0 (SMS port, privID 1, segment 0)
- ...

Each request is matched against all memory partitions that match the requests privID on that port. Requests whose address (32-bit virtual) falls within the configured memory partition address window (BADDR + SEGSZ elow) 'hit' that segment. When multiple segments are hit, MPAX selects the highest numbered hit segment for translation.



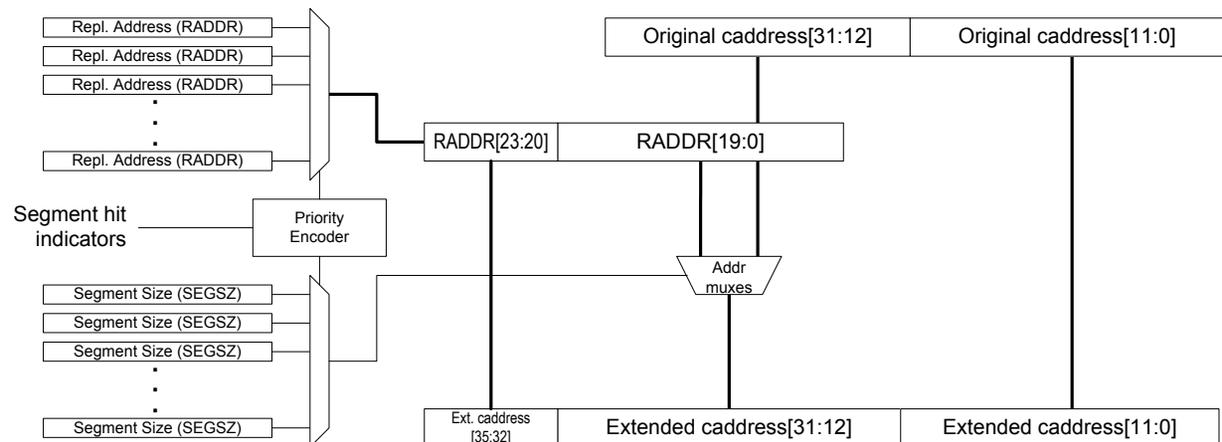
# MPAX Implementation

Once the hit segment is selected

1. MPAX replaces a number of the most significant bits of the address with the RADDR field in S\*S\_MPAXL\_\*\_\* depending on the segment size.
2. MPAX checks the request attributes (privilege level, emulation, secure) against the configured memory partition attributes (Permissions field)
3. MPAX attaches the share-ability (US field) attribute onto the request to assist the endpoint coherence controller.

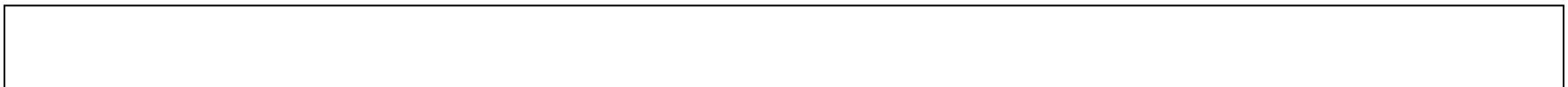
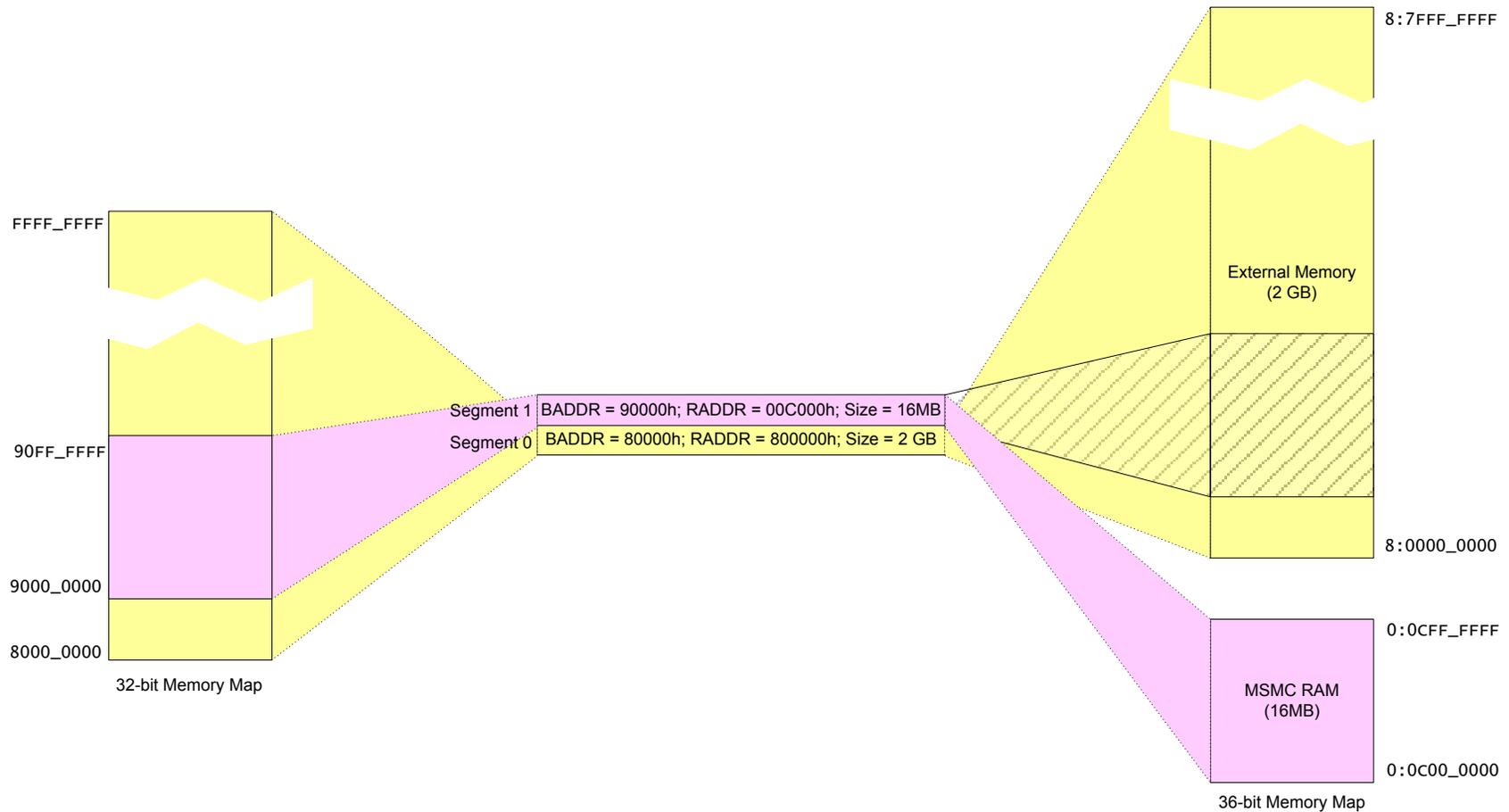
## Important Details

1. Requests arriving on the SMS port can only be remapped inside the MSMC SRAM physical address window (0x0:0C00\_0x0:0CFF\_FFFF).
2. Requests arriving on the SES port can be mapped to either external or MSMC SRAM physical address ranges



# MPAX Example Mapping

This example demonstrates overlapping segment priority as well as mapping external memory virtual address space to MSMC RAM physical address space (SES port only).



# MSMC2: Security Firewall

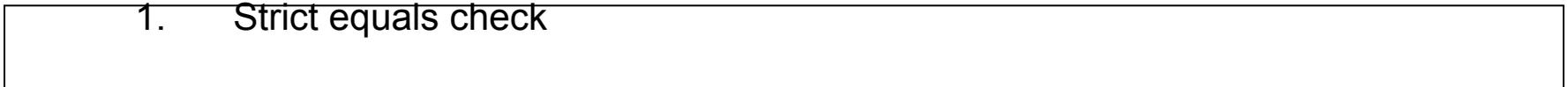
# MSMC Endpoint Security Firewall Basics

## Two types of security attributes

1. Transaction Attribute – This read or write request is secure or non-secure
2. Memory Page Attribute – Describes secure status of a memory page

## Two levels of security checks

1. Master-side security check – MPAX/MMU
  1. Greater than or equals check
  2. Secure demotion for accesses to non-secure memory page
2. Slave-side security check – MSMC Firewall
  1. Strict equals check

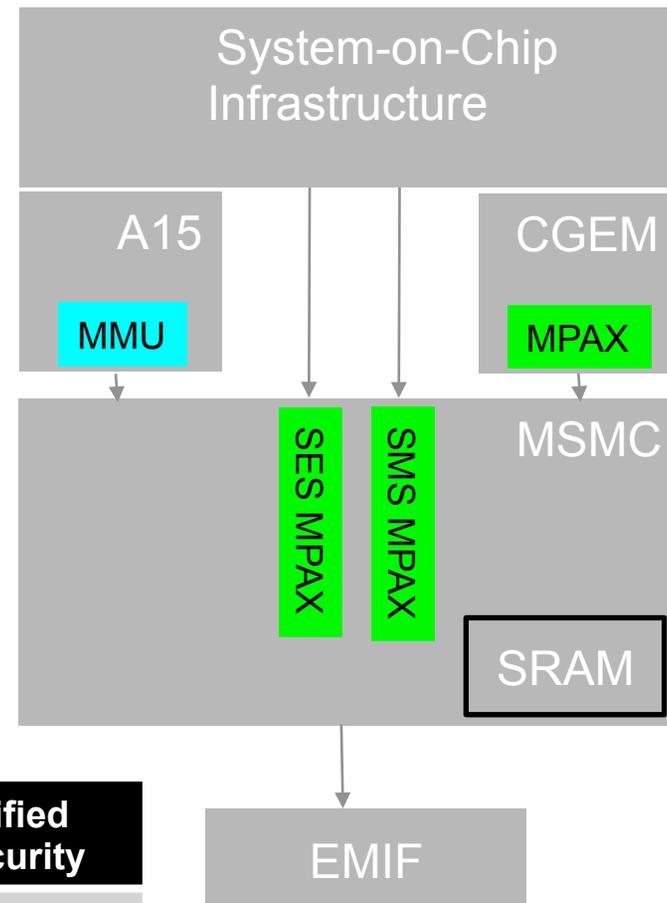


# Master Side Security

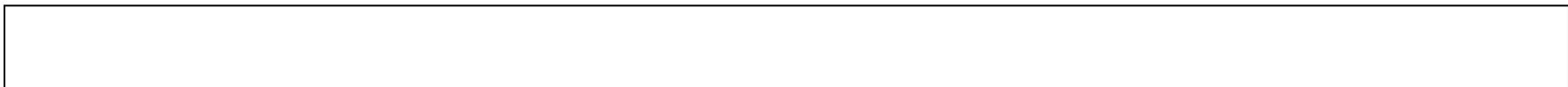
## Master's view of memory pages

1. Virtual to Physical Address Translation
2. RWX, Secure/Non-Secure
3. Share-ability

Master-side secure check compares request secure attribute and memory page attribute to generate final secure state of access.



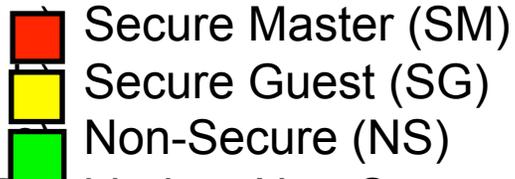
Request Security	Page Security	Result, Modified Request Security
Secure	Non-secure	Pass, Non-secure
Secure	Secure	Pass, Secure
Non-secure	Non-secure	Pass, Non-secure
Non-secure	Secure	Fail, -



# Endpoint Firewall Security

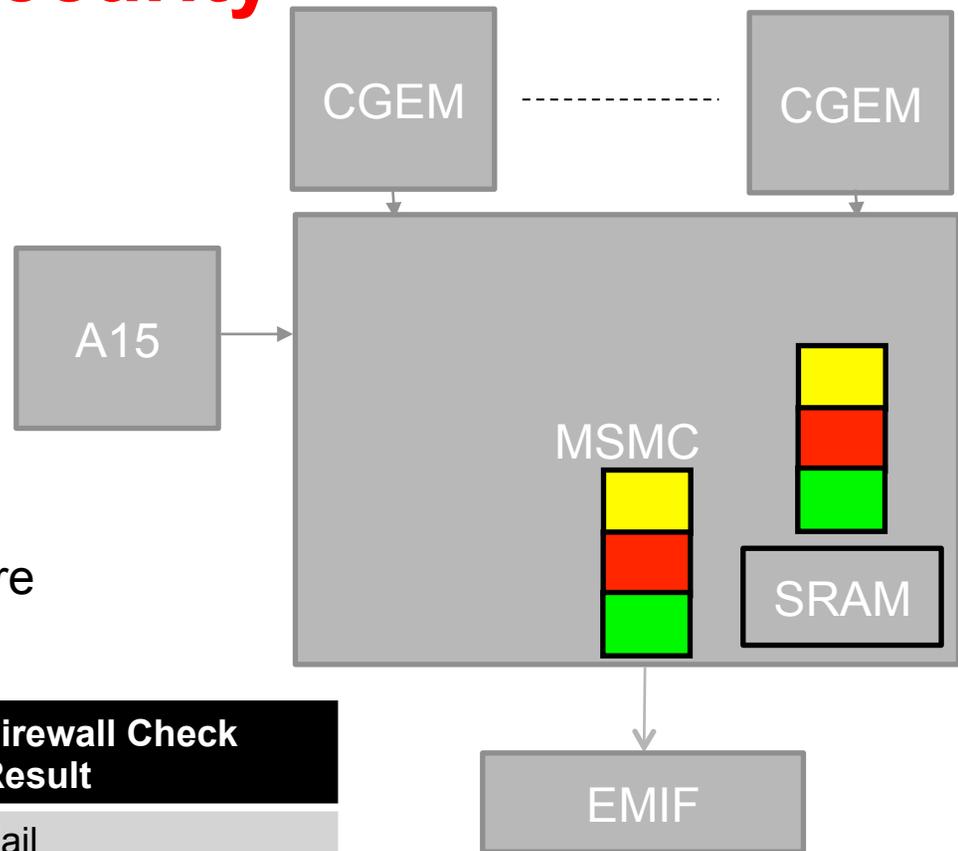
Slave's view of memory map

1. Physical address ranges
2. All memory is defined as

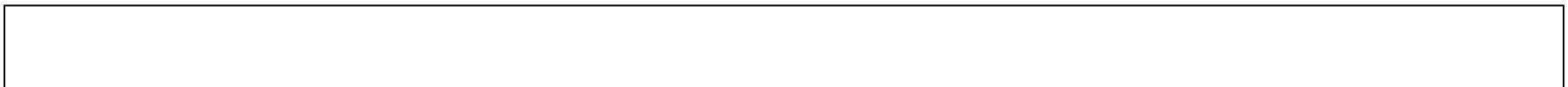


3. Disabled on Non-Secure Devices

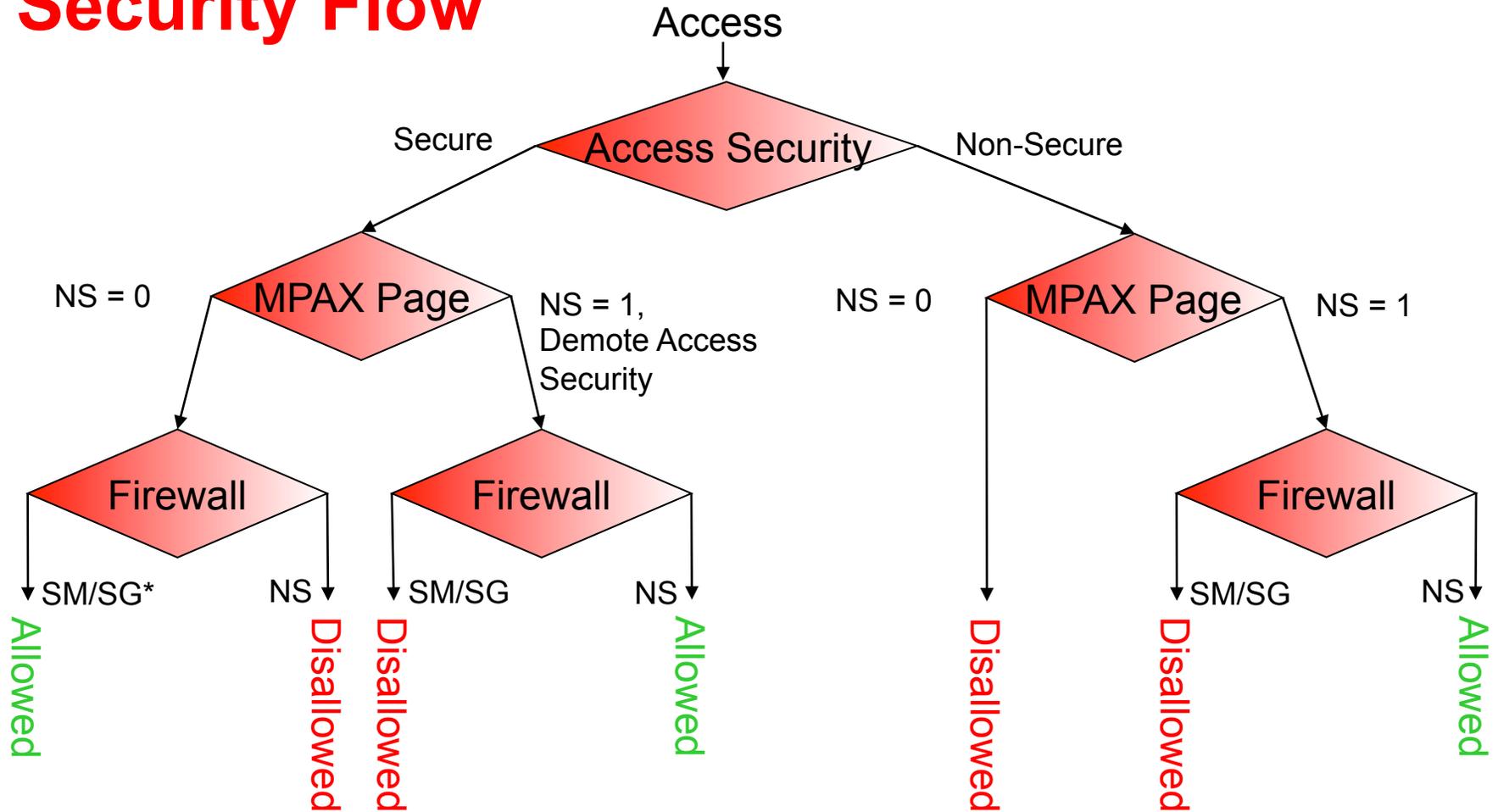
Endpoint firewall only sees final secure access attribute



Modified Request Security	Page Security	Firewall Check Result
Secure	Non-secure	Fail
Secure	Secure	Pass
Non-secure	Non-secure	Pass
Non-secure	Secure	Fail



# Security Flow



**Key**

- Secure
- Non-Secure

\* If access hits SM region cmstid must match SM\_MSTID\_INT in order to succeed, otherwise disallowed



# Firewall Implementation

MSMC provides four secure memory regions of programmable size and location

1. Secure Master SRAM
2. Secure Guest SRAM
3. Secure Master External
4. Secure Guest External

Segments are programmed via memory mapped registers with a physical memory base address and segment size (4KB – 1 TB). If segments overlap, the overlapped region belongs to the highest secure owner level, SM > SG > NS.

The firewall configuration register (SMIDCFG) must be configured by the Secure Master and provides

1. NS bit – Set to '1' to disable firewall and consider all SRAM and external memory non-secure
2. LK bit – Set to '1' to unlock Secure Master memory segments and turn them into Secure Guest segments, must reset to re-lock SM segments
3. Readable reflection of the current Secure Master VBUSM mstid

