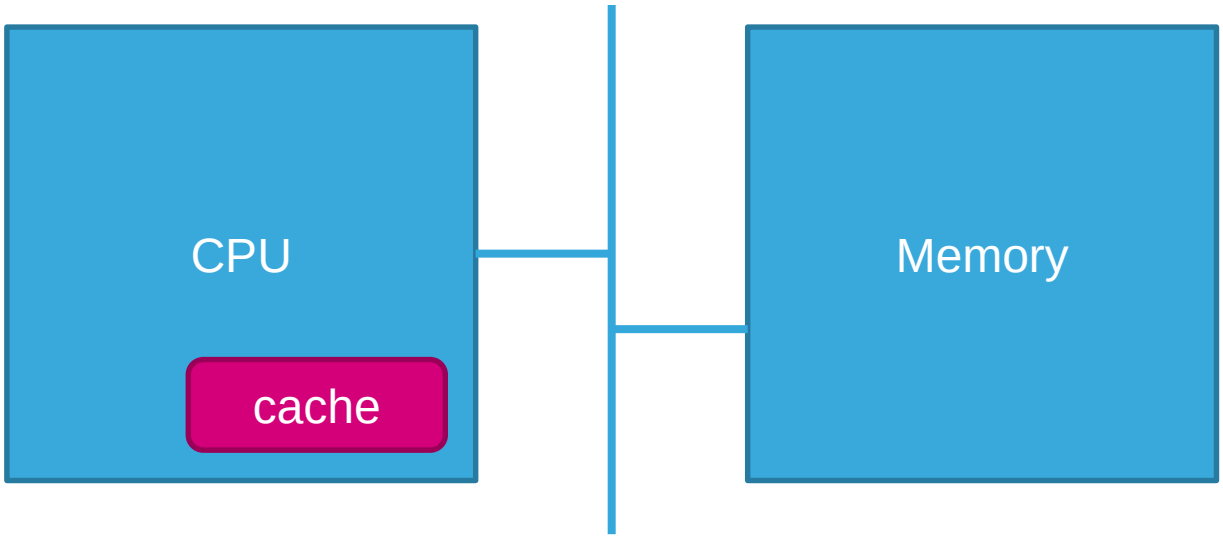
STM32F7xx Cache

life.augmented

Cache Overview

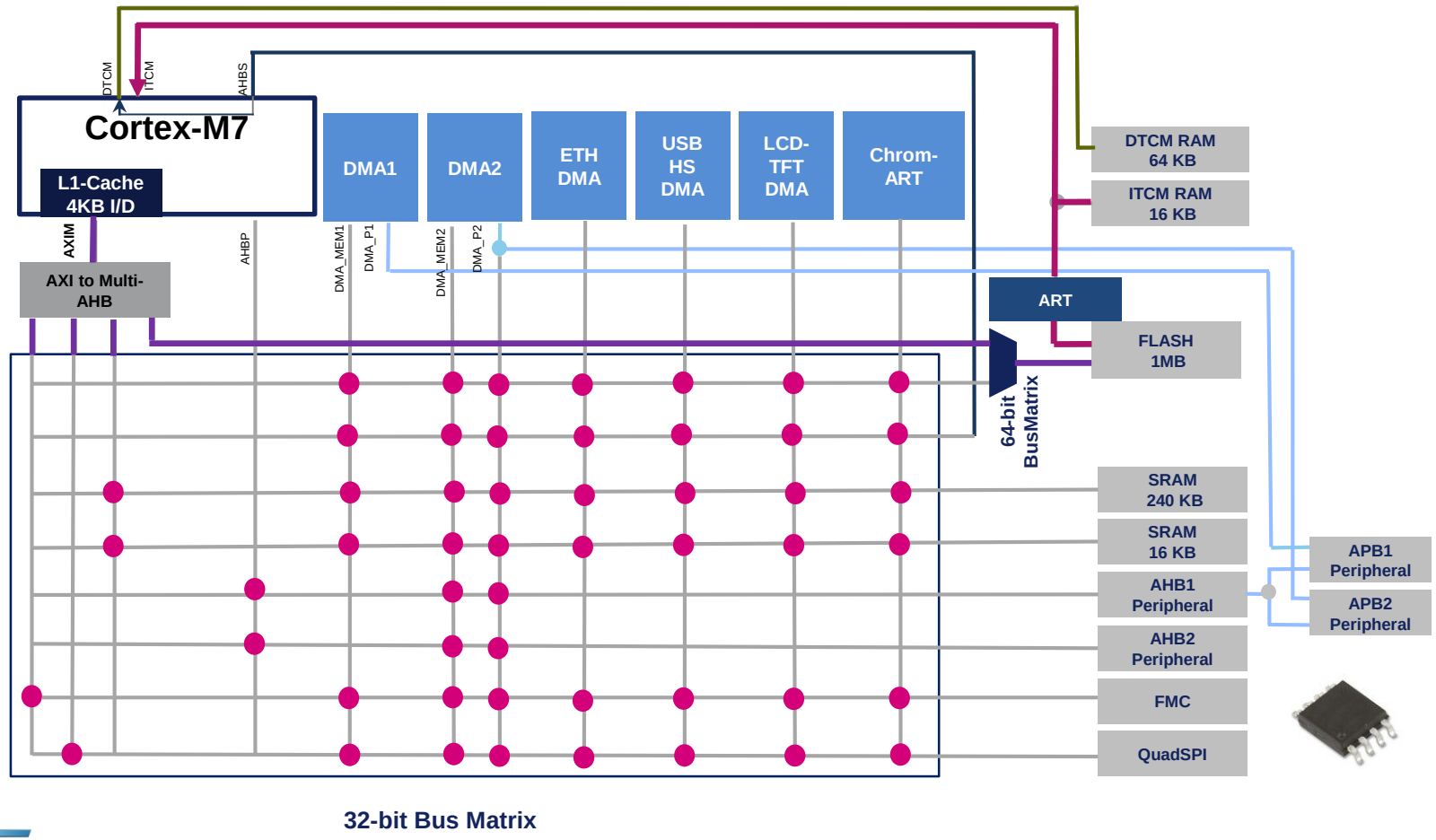
What is Cache

- Cache is located between CPU and memory.
- Save time to get instruction and data from external memory.
- Most CPU on SoC has cache inside CPU core. Cortex-M7 also.



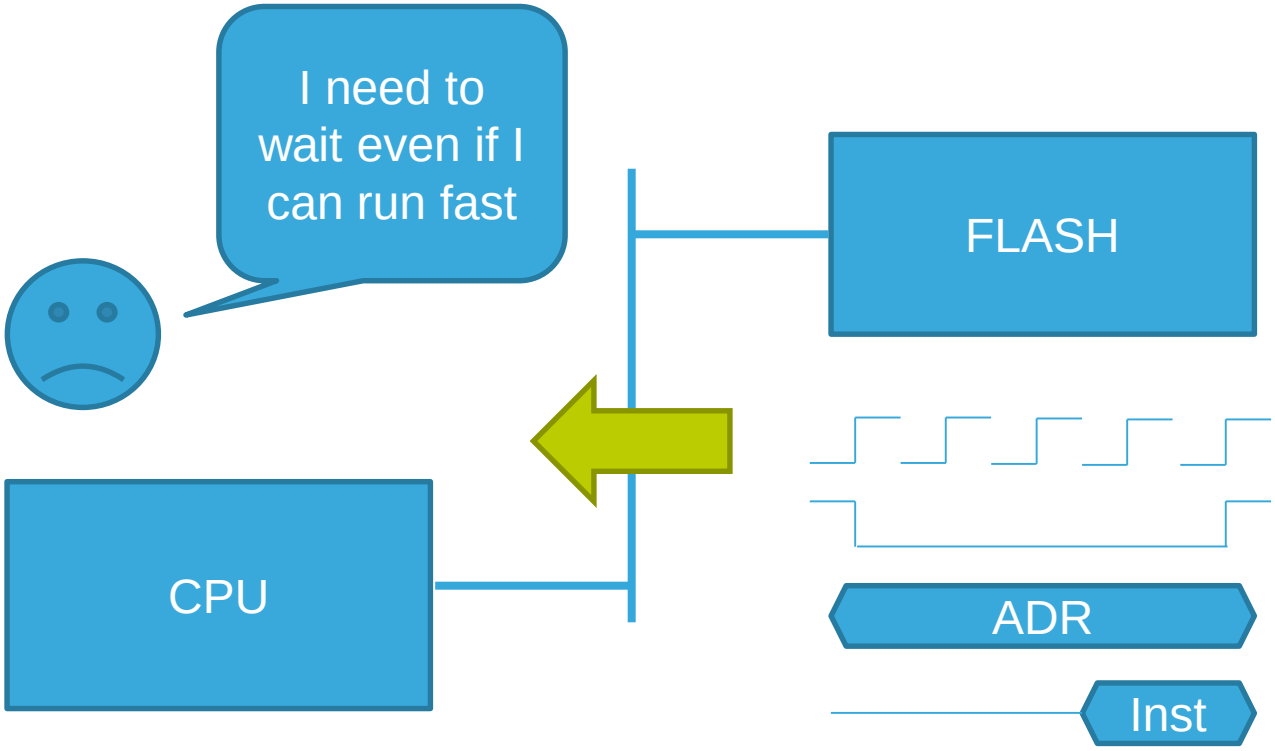
Cache & Bus Matrix

- New TCM bus compared to STM32F4xx



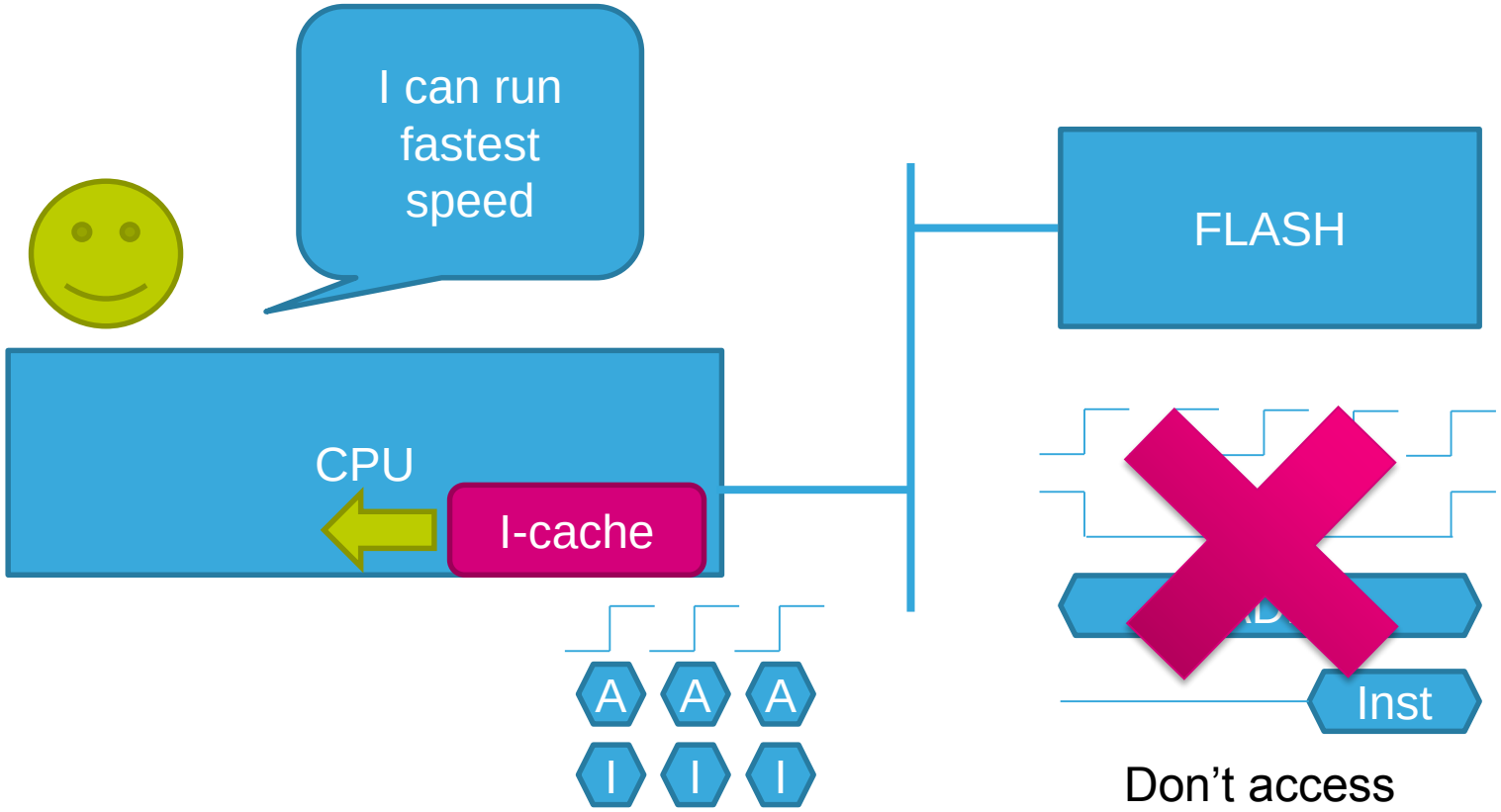
I-Cache

- Instruction Cache is used for instructions only. The time to get an instruction from the external memory is large. If the memory is external FLASH, CPU may needs 50-100ns to get the instruction.



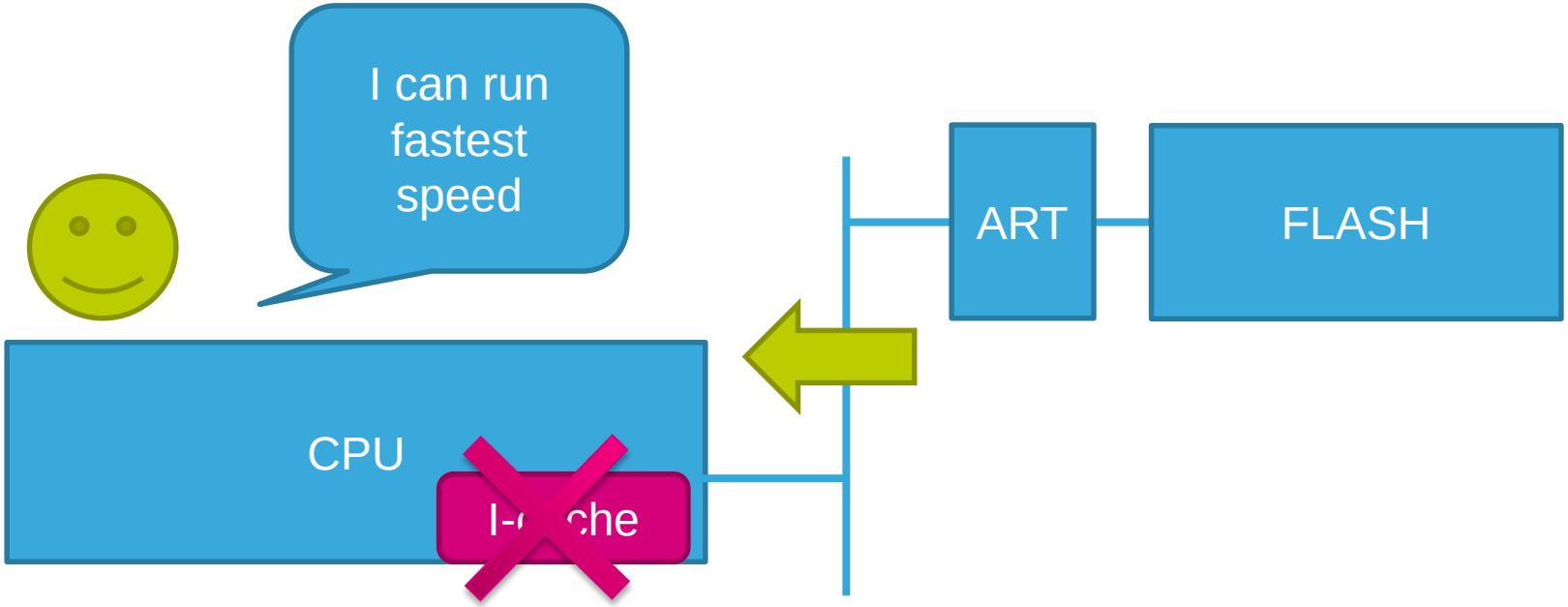
I-Cache

- I-cache stores the instructions which have already been executed. And next time, CPU can get the instruction in just 1 clock.



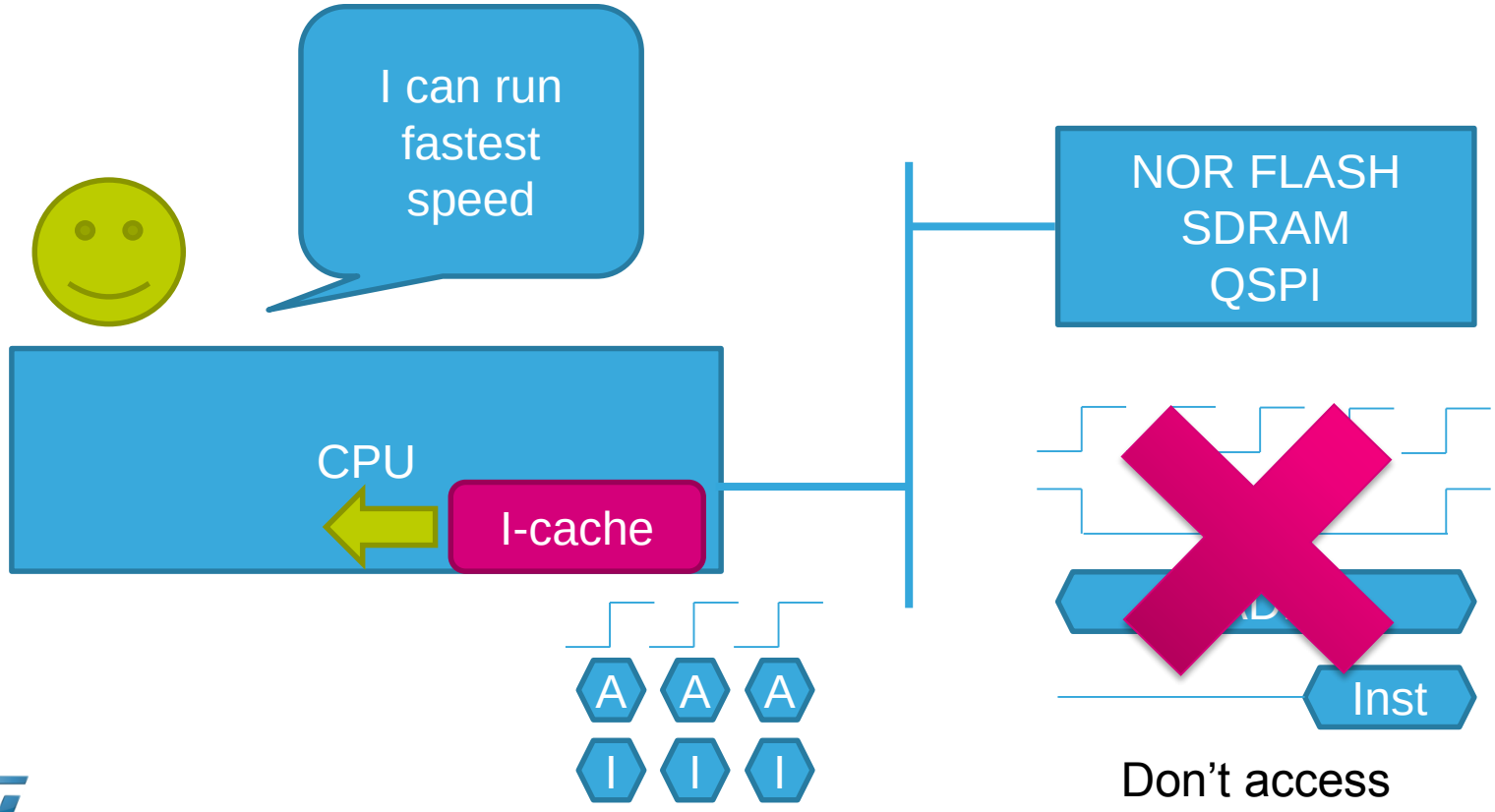
STM32 case for internal FLASH

- Thanks to ART Accelerator, STM32 CPU can read an instruction in just 1 clock from internal FLASH (Like “0-wait state”). So, I-cache is not necessary for internal flash.



STM32 case for external memory

- If the instructions are on external memory (NOR FLASH, QSPI, etc), I-cache is useful. This is because the access time of external memory is not 1 clock.

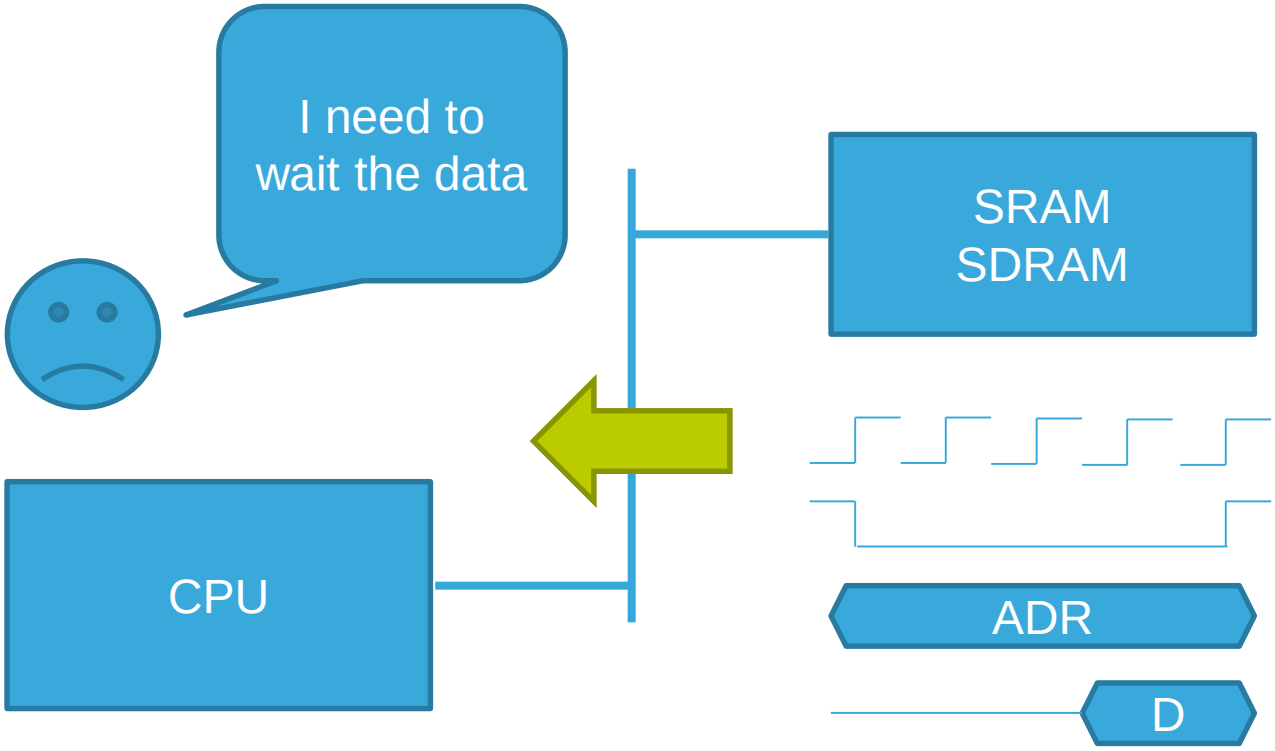


I-cache Summary

- When using ART with internal FLASH, I-cache is not necessary.
- When using external memory (NOR FLASH, SDRAM and QSPI), I-cache is very useful.

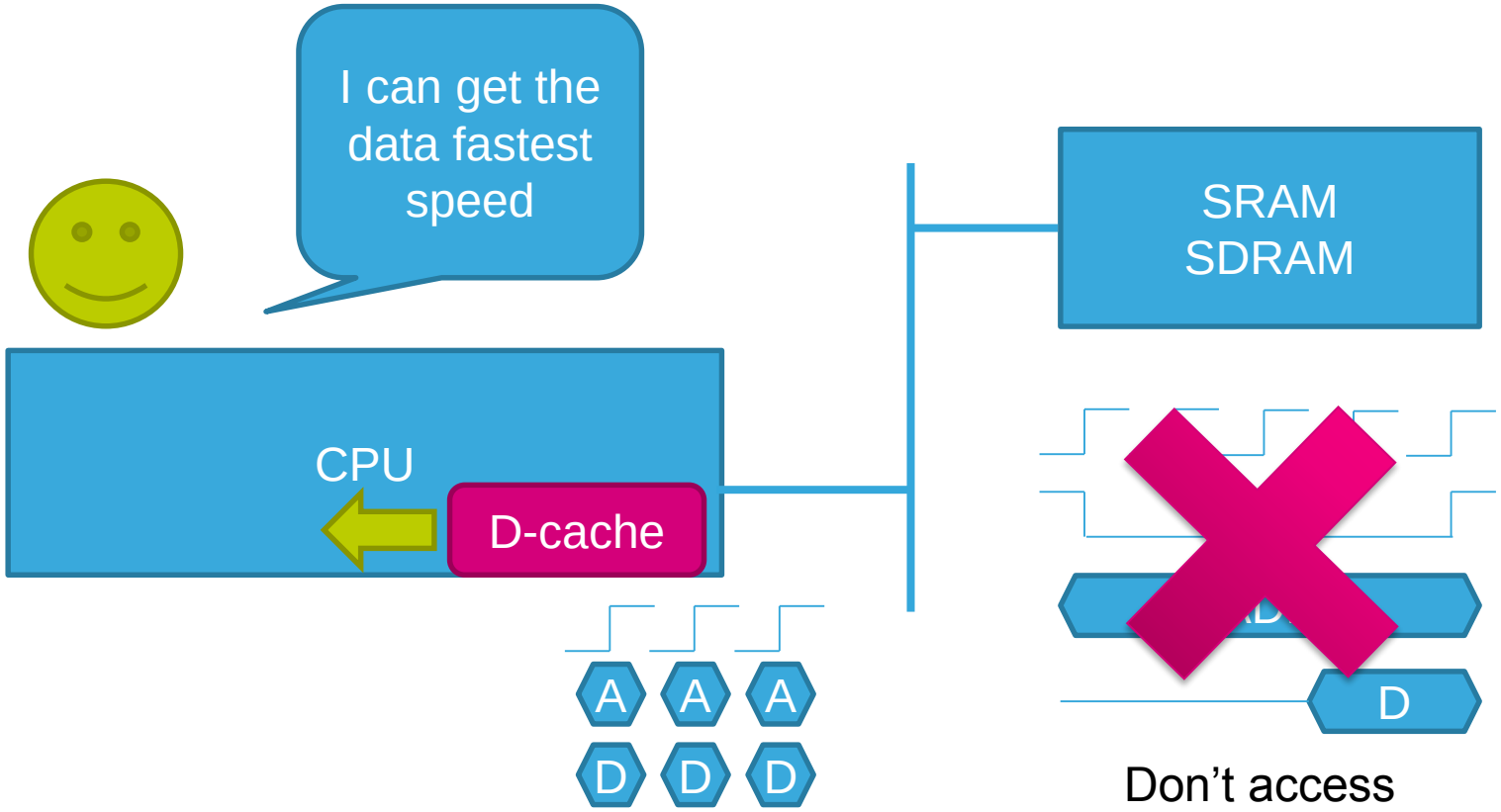
D-cache

- Data Cache is used for data only. It's similar to I-cache.
  - It takes a long time to get the data from external memory.



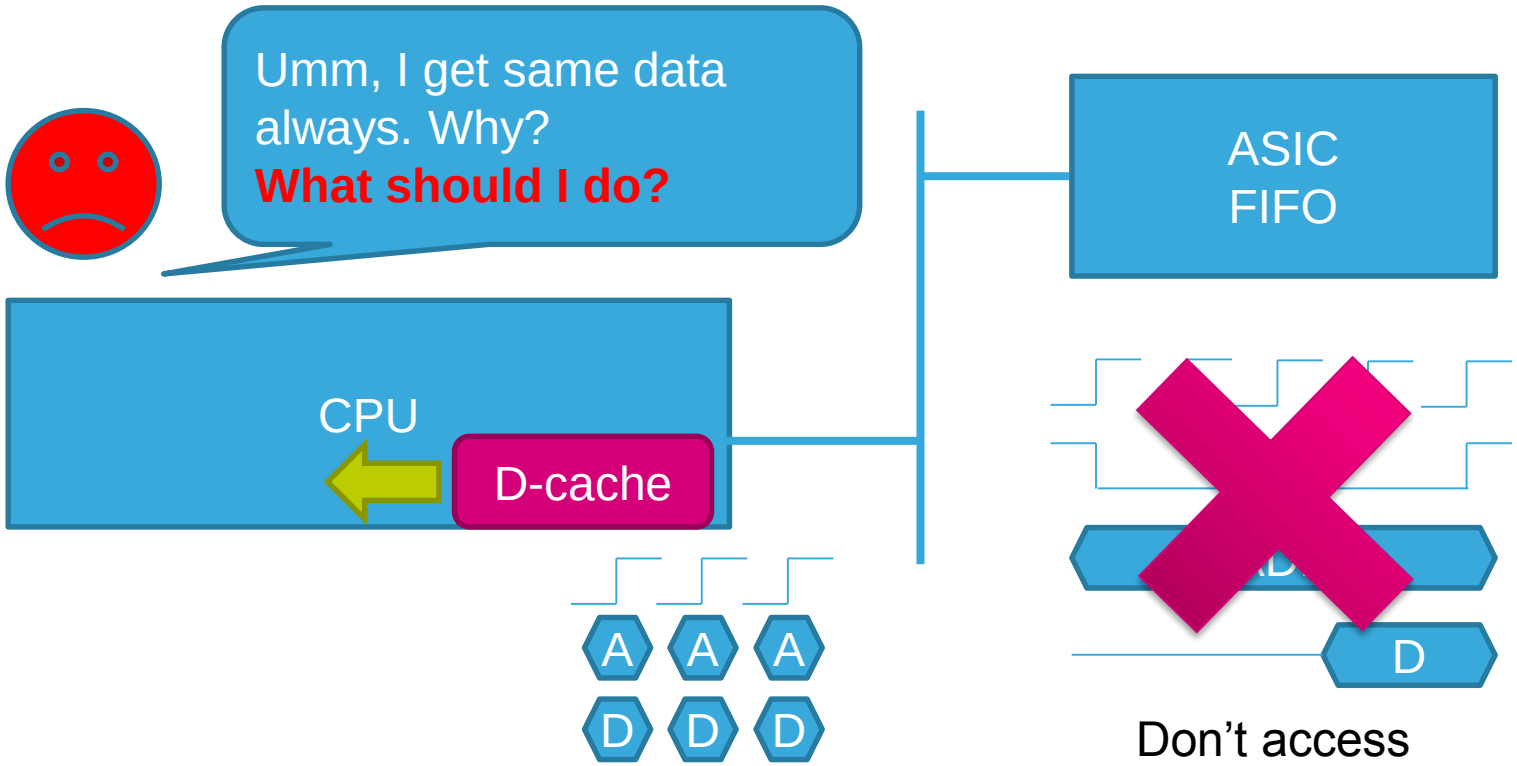
D-cache read

- D-cache stores the data which have been read already. When the same data is needed again, the CPU can get the data in just 1 clock.



STM32 case of D-cache read

- Data Coherency
  - If the connected device is normal memory, D-cache read is useful. However, If the connected device is ASIC or FIFO, CPU reads previous data from D-cache regardless of data in ASIC or FIFO changed or not.



Answer will be

- Memory types & Attributes
- Cache maintenances

Memory types & Attributes

Memory types & Attributes

Memory type		Can be cached	Merging	Restartable
Normal	Shared	No[a]	Yes	Yes
	Non-shared	Yes	Yes	Yes
Device	Shared	No	No	No
	Non-shared	No	No	No
Strongly-ordered	Shared	No	No	No

a. Unless CACR.SIWT is set to 1.

Memory Types

- Normal
  - The processor can re-order transactions for efficiency
  - Can be cached (peripheral on cache ability / shared attributes)
- Device and Strongly-ordered
  - Used only for Devices- not cached.
  - The processor preserves transactions order relative to another transaction to Device or Strongly-ordered memory.
  - The different ordering for Device and Strongly-ordered memory mean that the external memory system can buffer a write to Device memory, but must not buffer a write to Strongly-ordered memory.

Memory Attributes

- Shared
  - For memory space that is **shared with another master** with system coherency guarantees.
  - The memory system provides data synchronization between bus masters in a system with multiple bus masters, for example, a processor with a DMA controller.
  - By default data access are NOT cached in the D-cache
    - By setting CACR.SIWT to 1, normal cacheable shared location are treated as WT.
    - Other memory agent updates are not visible to Cortex®-M7 software without suitable cache maintenance.
- Non shared
  - If a memory region was defined to non-shared, when multiple bus masters accessing it, software must ensure data coherency between these bus masters.

Memory Attributes

- Execute Never (XN)
  - Means the processor prevents instruction accesses. A fault exception is generated only on execution of an instruction executed from an XN region.
- Full support of the following attributes
  - Write-Back, no write allocate (WBRA)
    - Read allocate – specific use cases (e.g memset())
  - Write-Back, write allocate (WBWA)
    - Write allocate - best for overall performance
  - Write Through, no write allocate (WT)
    - Write through – lower performance than WB. Useful for safety-critical

Memory types & Attributes

- Memory attributes are defined by MPU settings
  - Cache policy programmed in MPU_RASR register

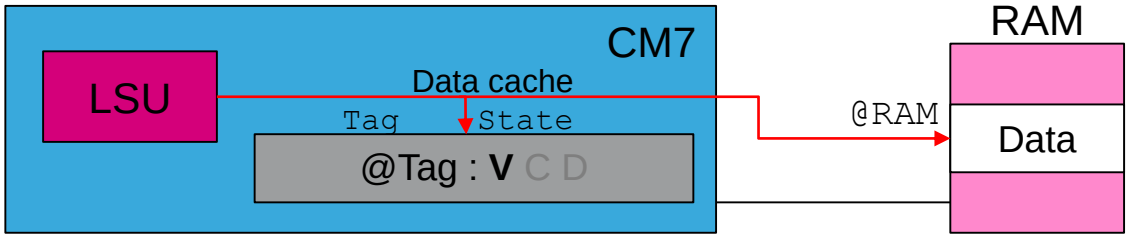
TEX	C	B	Description	Memory type
000	0	0	Strongly Ordered	Strongly Ordered and shared
000	0	1	Shared Device	Device
000	1	0	Cacheable, write-through, no write allocate	Normal
000	1	1	Cacheable write-back, no write allocate	Normal
001	1	1	Cacheable write-back; write and read allocate	Normal
010	0	0	Non-shared device	Device

Memory Default mapping and attribute

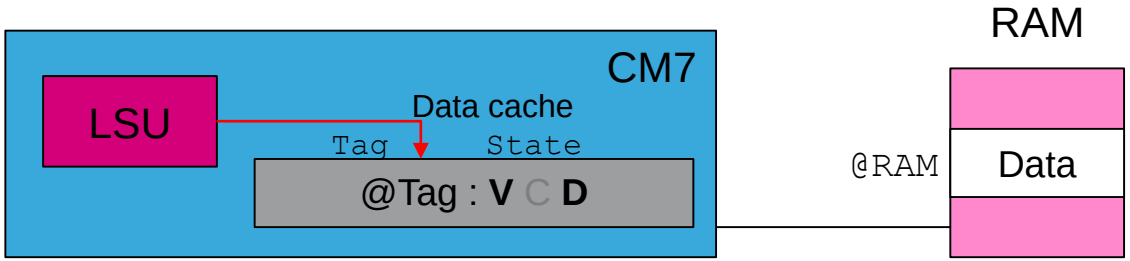
Address	Name	Memory Type	XN ?	Cache	Description
0x0000 0000 0x1FFF FFFF	Code	Normal		WT	Typically ROM or flash memory. Memory required from address 0x0 to support the vector table for system boot code on reset
0x2000 0000 0x3FFF FFFF	SRAM	Normal	-	WBWA	SRAM region typically used for on-chip RAM
0x4000 0000 0x5FFF FFFF	Peripheral	Device	XN	-	On chip peripheral address space
0x6000 0000 0x7FFF FFFF	RAM	Normal	-	WBWA	Memory with write-back, write allocate cache attribute for L2/L3 cache support
0x8000 0000 0x9FFF FFFF	RAM	Normal	-	WT	Memory with write-through cache attribute
0xA000 0000 0xBFFF FFFF	Device	Device, Shareable	XN	-	Shared device space
0xC000 0000 0xDFFF FFFF	Device	Device, non-shareable	XN	-	Non-shared device space
0xE000 0000 0xE00F FFFF	PPB	Strongly-Ordered	XN	-	1MB region reserved as the PPB. This supports key resources, including the System Control Space and debug features.
0xE010 0000 0xFFFF FFFF	Vendor_SYS	Device	XN	-	Vendor system region

Cache policy

- Description of the Inner (L1) cache policy:
  1. Write-through policy (read or write allocate)
     - Data writes occur simultaneously to the cache and to next level memory

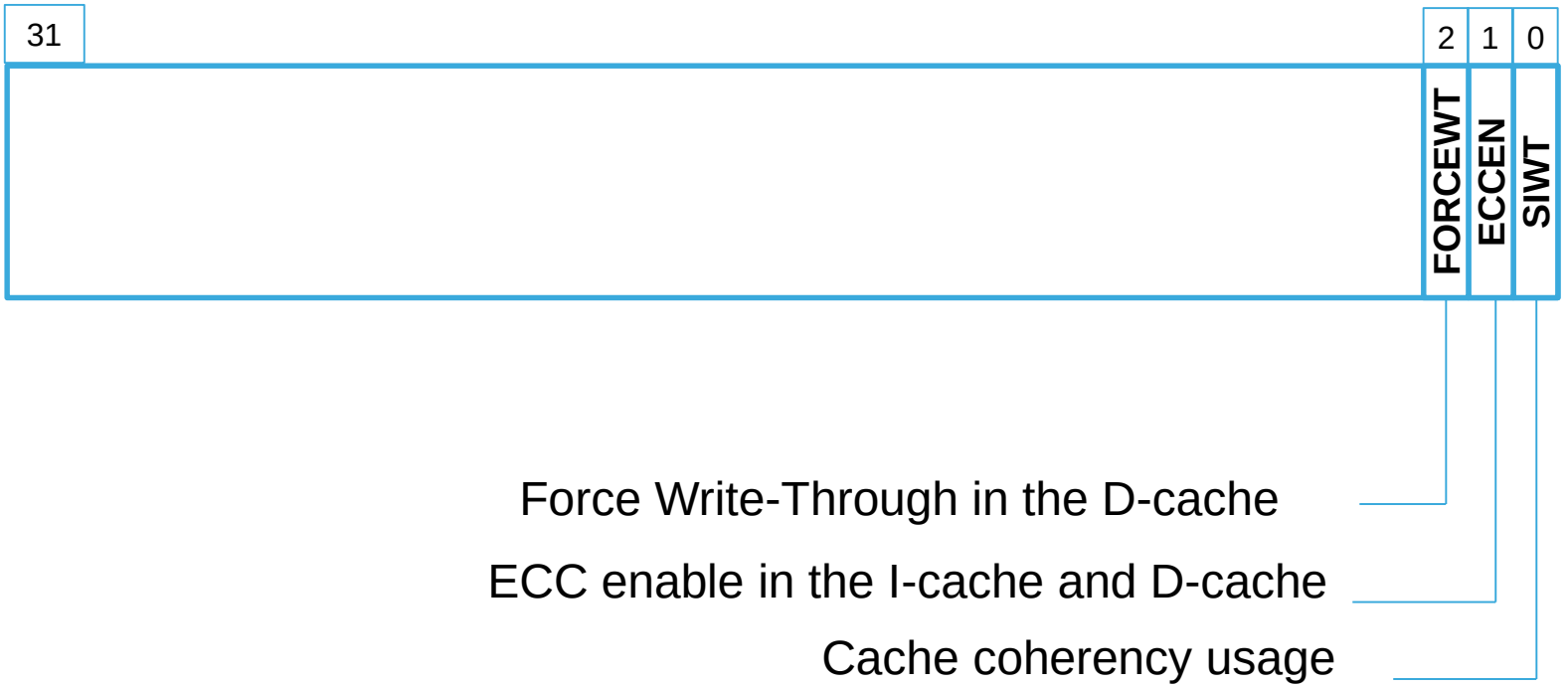


  2. Write-back policy (read or write allocate)
     - Data write occurs only to the cache



L1 Cache Control Registers

- The CACR controls the L1 ECC and le L1 cache coherency usage model
- CACR – 0xE000EF9C (R/W)



Software ordering of memory accesses

- The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions. This is because:
  - The processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
  - The processor has multiple bus interfaces.
  - Memory or devices in the memory map have different wait states.
  - Some memory accesses are buffered or speculative.

Memory system ordering

A1 \ A2	Normal Access	Device access		Stronly-ordered access
		Non-Shareable	Shareable	
Normal access	-	-	-	-
Device access, Non-shareable	-	<	-	<
Device access, Shareable	-	-	<	<
Strongly-ordered access	-	<	<	<

- Means that the memory system does not guarantee the ordering of the accesses.

< Means that accesses are observed in program order, that is, A1 is always observed before A2.

SW Memory ordering and Barriers

- If the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The processor provides the following memory barrier instructions:

- **DMB**: The Data Memory Barrier (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions.

- **DSB:** The Data Synchronization Barrier (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute.

- **ISB:** The Instruction Synchronization Barrier (ISB) ensures that the effect of all completed memory transactions is recognizable by subsequent instructions.

Cache Coherency

- Two options
  - Use Shared attribute for all shared regions of memory
    - This will by default prevent these regions from being cached in D-Cache
    - Lower performance since all accesses go to L2
    - Easiest for SW since caches are transparent for these regions of memory
  - SW cache maintenance
    - Cortex-M7 writes need to be made globally visible
      - Use Write-Through memory attribute by MPU
      - Use CACR.SIWT (Shared = Write Through)
      - D-Cache clean of D-Cache and invalidate for all updated locations
    - Other masters writes need to be visible to Cortex-M7
      - Invalidate updated locations in Cortex-M7 D-Cache.

Cache Maintenance

Cache Terminology

- Cache hit
  - Lookup finds desired entry cache
- Cache miss:
  - Lookup does not find desired entry in cache
- Cache allocation
  - Put a new entry (cache line) into the cache. Occurs on cache miss
  - Write-allocate: allocate on reads and writes
  - Read allocate: allocate on reads only
- Line fill
  - Read request on the bus for an entire cache-line
  - This line, once available, will be subsequently allocated to cache

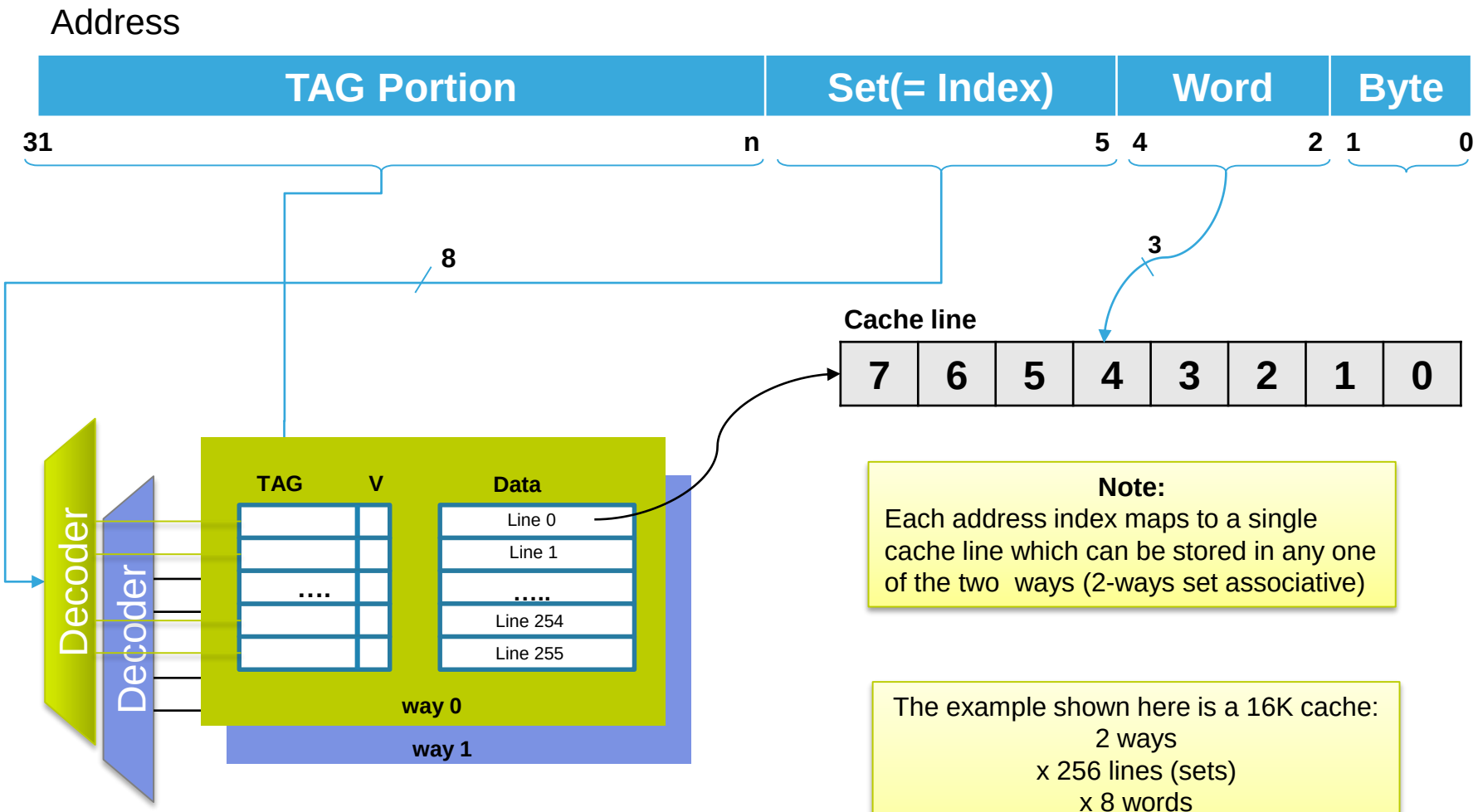
Cache Terminology

- Eviction:
  - Write of an entire cache line on the bus
  - Occurs when a dirty cache line is replaced by a new cache line
  - Because it is dirty, the new data needs to be written back to L2.
- Write-back:
  - Stores update cache only leaving cache lines ‘dirty’
  - If this line is replaced by a new one, its data needs to be written to L2
- Write through:
  - Stores update cache and L2 at the same time
  - Lines can never be dirty

Cache features

- I-cache
  - 2-way set-associative
    - Lower-cost than 4-way associative with almost identical performance
- D-cache
  - 4-way set-associative
  - Support for dual-issue of loads without use of dual-ported memories
    - Achieved through micro-Tag structure
    - 32-entry 4 way associative structure- contains subset of main Tag RAMs
    - Updated on main cache hits and new allocation

I-Cache 2-Way Set-Associative

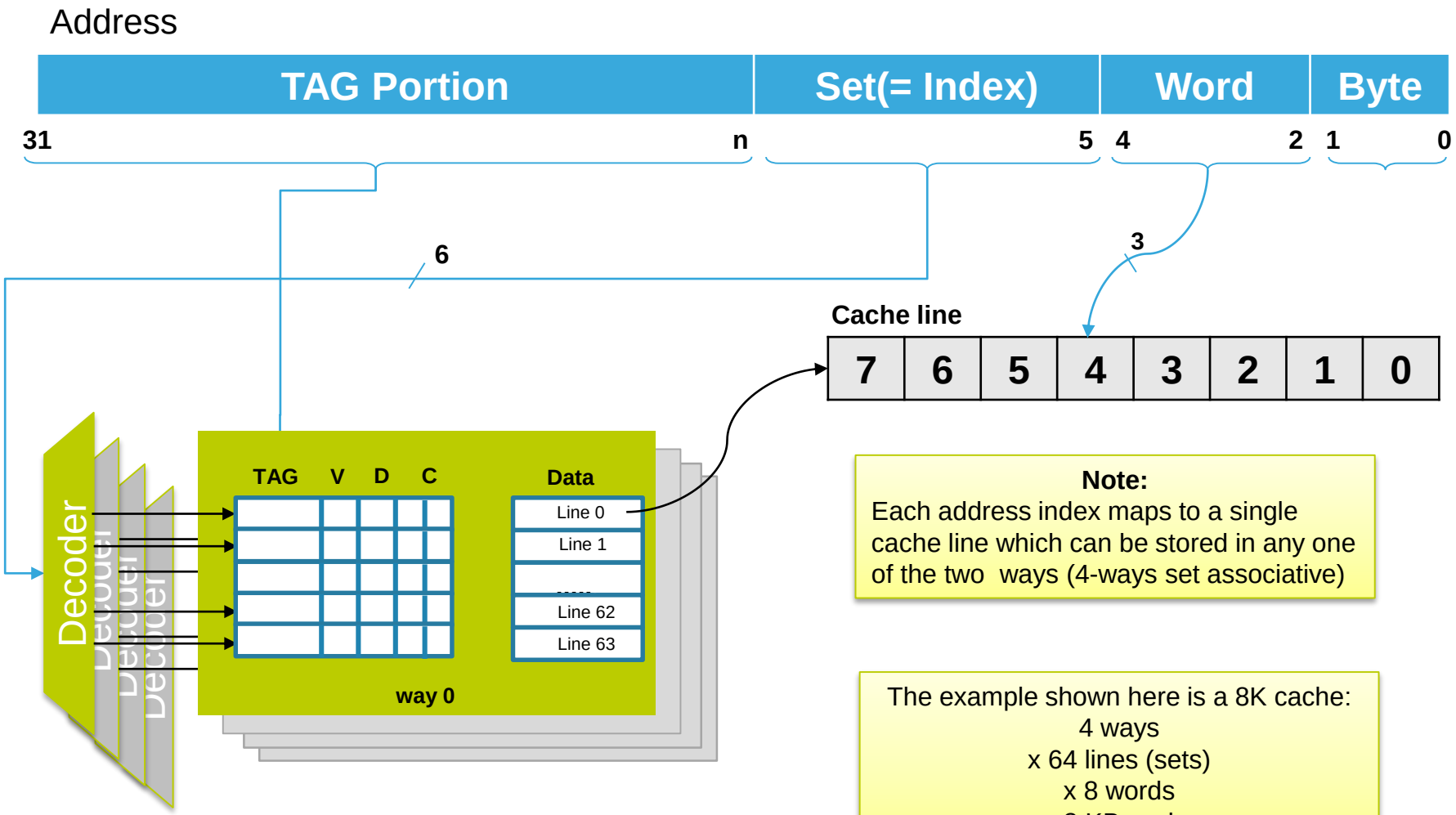


Note:
Each address index maps to a single cache line which can be stored in any one of the two ways (2-ways set associative)

The example shown here is a 16K cache:
2 ways
x 256 lines (sets)
x 8 words
= 16 KB cache

V – valid bit

D-Cache 4-Way Set-Associative



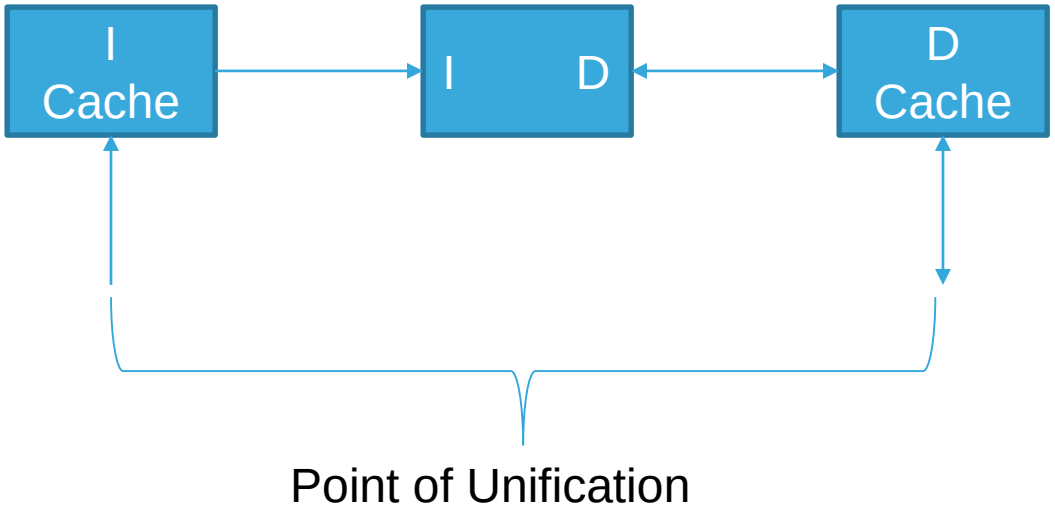
Note:
Each address index maps to a single cache line which can be stored in any one of the two ways (4-ways set associative)

The example shown here is a 8K cache:
4 ways
x 64 lines (sets)
x 8 words
= 8 KB cache

V – valid bit, D-dirty bit, C – 2 bits outer cacheability attributes

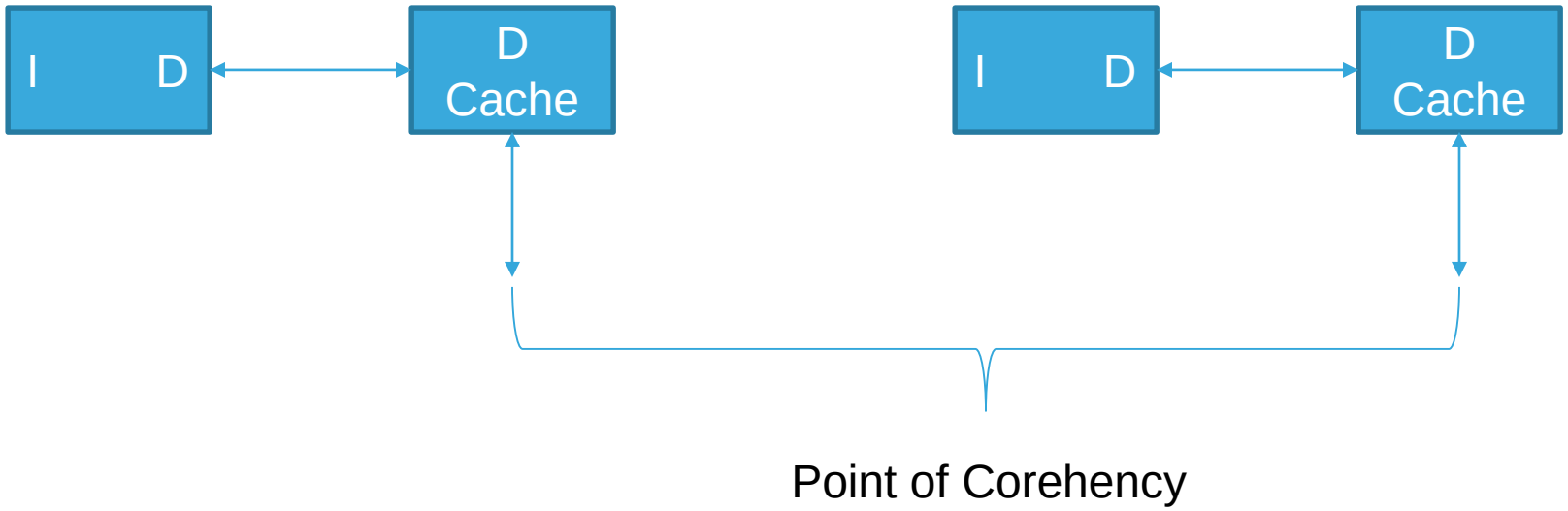
Point of Unification (PoU)

- Instruction and data accesses see the same copy of memory
- PoU is 1 – D-Cache and I-Cache accesses are unified at L2



Point of Coherency (PoC)

- All agents see the same copy of memory
- PoC is 1 – Data accesses are coherent at L2 or beyond



Cache Maintenance Operations

- All cache maintenance operation are through the memory mapped System Control Space (SCS) region of the internal PPB memory space

Address	Register Name	Function
0xE000EF50	ICIALLU	I-cache invalidate all
0xE000EF58	ICIMVAU	I-cache invalidate by MVA to PoU
0xE000EF5C	DCIMVAC	D-cache invalidate by MVA to PoC
0xE000EF60	DCISW	D-cache invalidate by set-way
0xE000EF64	DCCMVAU	D-cache clean by MVA to PoU
0xE000EF68	DCCMVAC	D-cache clean by MVA to PoC
0xE000EF6C	DCCSW	D-cache clean by set-way
0xE000EF70	DCCIMVAC	D-cache clean and invalidate by MVA to PoC
0xE000EF74	DCCISW	D-cache clean and invalidate by set-way

Cache Maintenance Operations

CMSIS function	Descriptions
void SCB_EnableICache(void)	Invalidate and then enable instruction cache
void SCB_DisableICache(void)	Disable instruction cache and invalidate its contents
void SCB_InvalidateICache(void)	Invalidate instruction cache
void SCB_EnableDCache(void)	Invalidate and then enable data cache
void SCB_DisableDCache(void)	void SCB_InvalidateDCache(void) Invalidate
void SCB_CleanDCache(void)	Clean data cache
Void SCB_CleanInvlaidateDCache(void)	Clean and invalidate data cache

Cache Maintenance Operations

- Architecturally
  - DSB is required to ensure completion of all previous cache maintenance operations
  - ISB is required to ensure that I-Cache maintenance operations are visible to subsequent instruction fetches.
- These barriers are a **requirement** on Cortex-M7
- A single DSB can be used after a sequence of cache maintenance operation – you don't need a barrier after each.
- Cache maintenance operations can continue in the background without stalling the pipeline
  - Execute a DSB to ensure all previous cache maintenance operations have completed
- For I-cache maintenance operations, an ISB is required

Initializing and enabling L1 caches

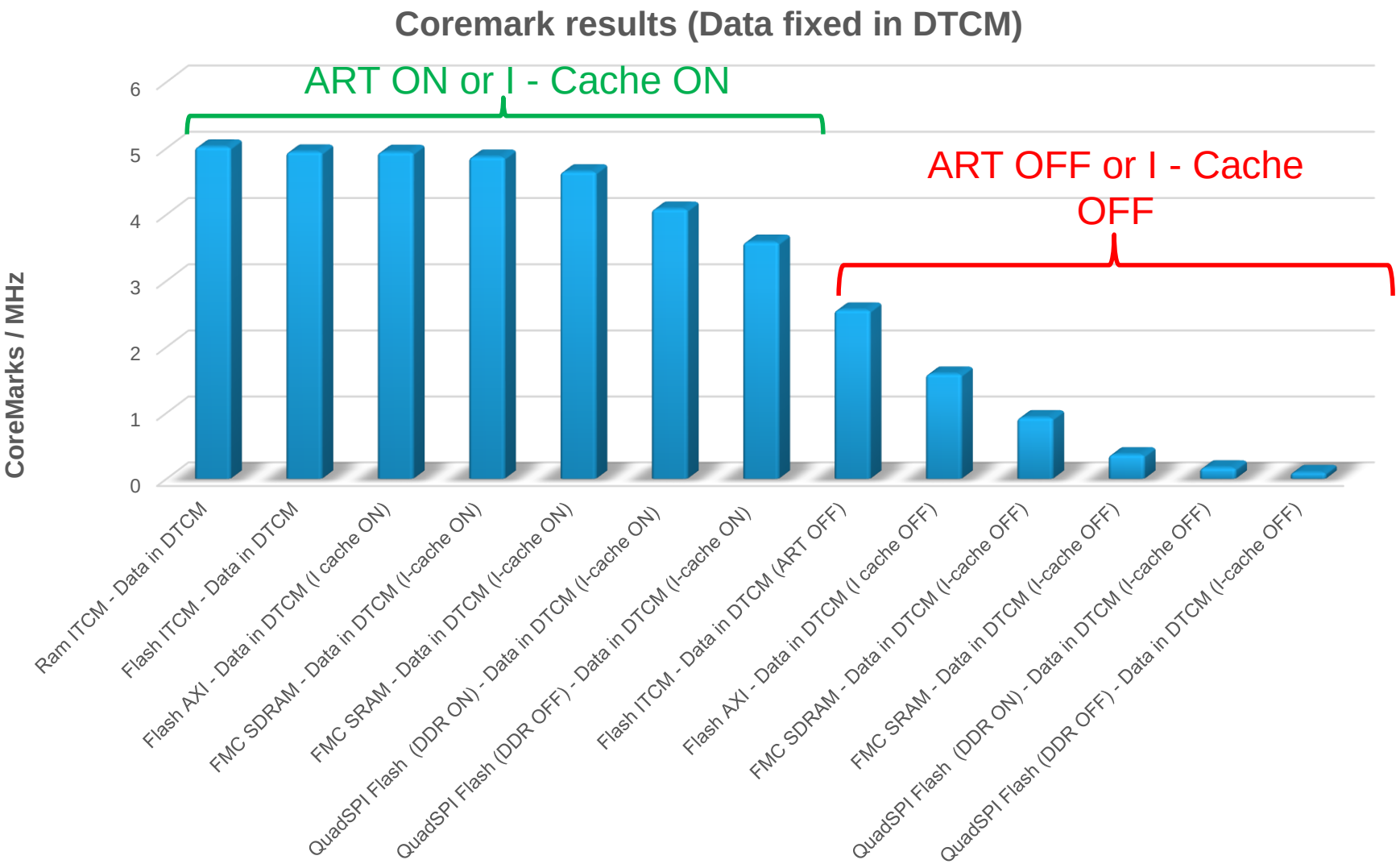
- Both the I-cache and D-cache must be completely invalidated before they are enabled on power-on reset
  - Cache line valid bits are held in Tag RAMs
  - Failure to do this could cause **UNPREDICTABLE** behavior
  - Note: on soft resets, it is possible to avoid invalidation the caches if the contents of the RAMs before reset were reliable
- I-Cache:
  - Invalidate the entire I-Cache using the ICIALLU register
- D-Cache:
  - Recommended method is to iterate through the entire D-Cache and invalidate
- To ensure data coherency, the D-cache should be cleaned before it is disabled
  - Required only if **Write-Back** caching is used
  - Failure to do so could result in loss of data

Cache Maintenance For DMA OP

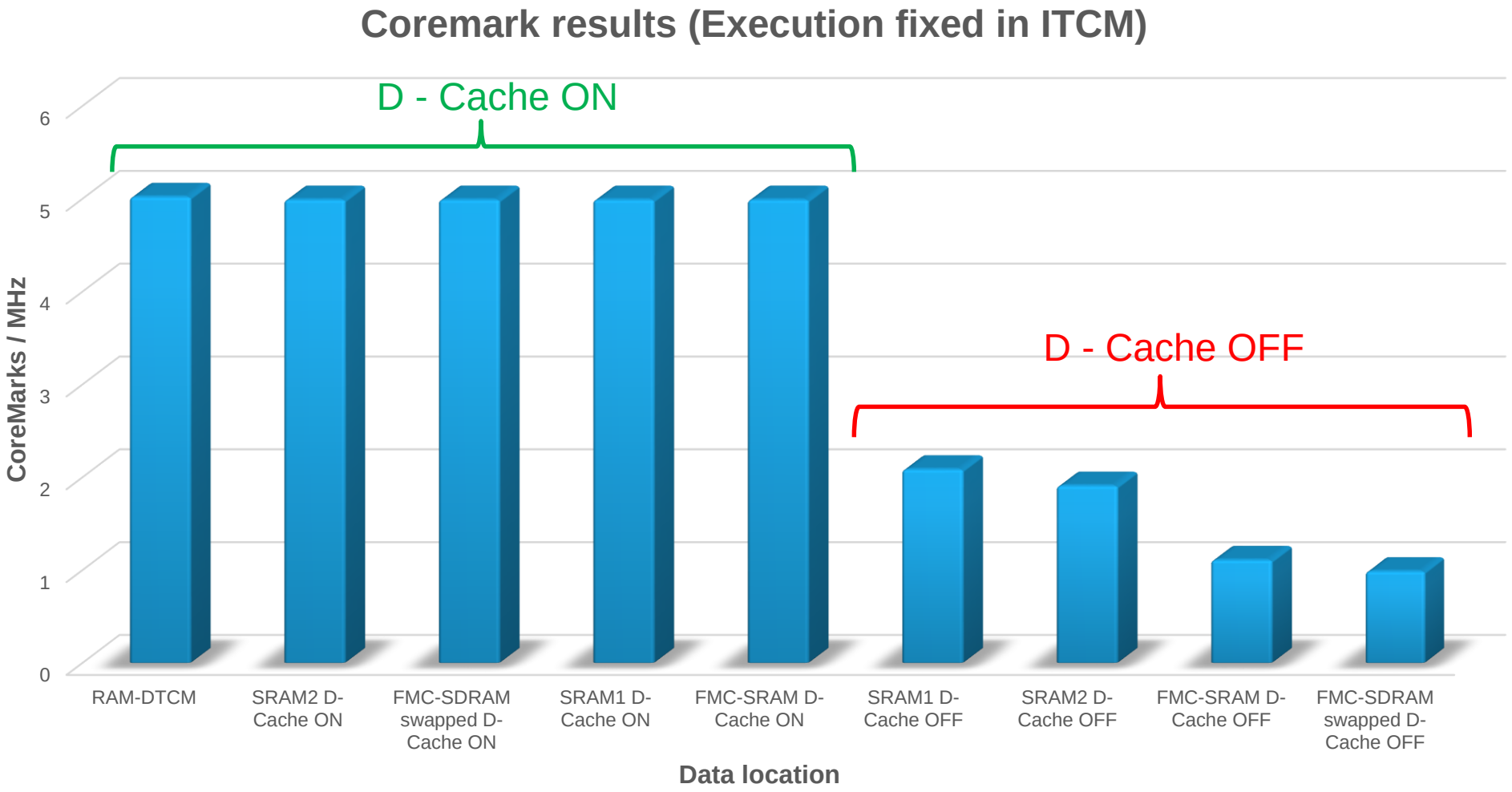
- Memory to Peripheral
  - Cache clean (by addr) should be performed after data buffer prepared by CPU
    - SCB_CleanDCache() or SCB_CleanDCache_by_Addr()
- Peripheral to Memory
  - Cache invalidate (by addr) should be performed after DMA transfer complete
    - SCB_InvalidateDCache() or SCB_InvalidateDCache_by_Addr()
- Memory to Memory
  - Cache clean (by addr) should be performed after data buffer prepared by CPU
  - Cache invalidate (by addr) should be performed after DMA transfer complete
    - SCB_CleanDCache() or SCB_CleanDCache_by_Addr()
    - SCB_InvalidateDCache() or SCB_InvalidateDCache_by_Addr()

Benefit by using Cache

Coremark Benchmark



Coremark Benchmark



Quiz

1. Which memory attribute are cacheable?
2. List the cache maintenance operation?
3. When memory barriers must be used?

STM32F756x Hands on

MCD Application Team

15W04 Tunis

V1.0

life.augmented

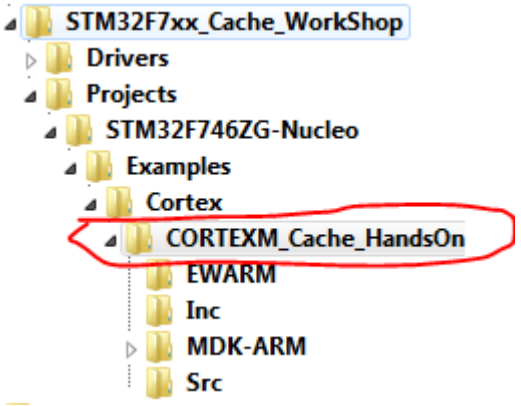
Cortex-M7 Cache Maintenance and Data coherency

Objective

- The purpose of this example is to get familiarized with ARM Cortex-M7 data cache coherency
  - Demonstrated how to ensures data coherency between DMA and CPU
  - List example of L1 cache configuration and maintenance operation

Hands-On directory

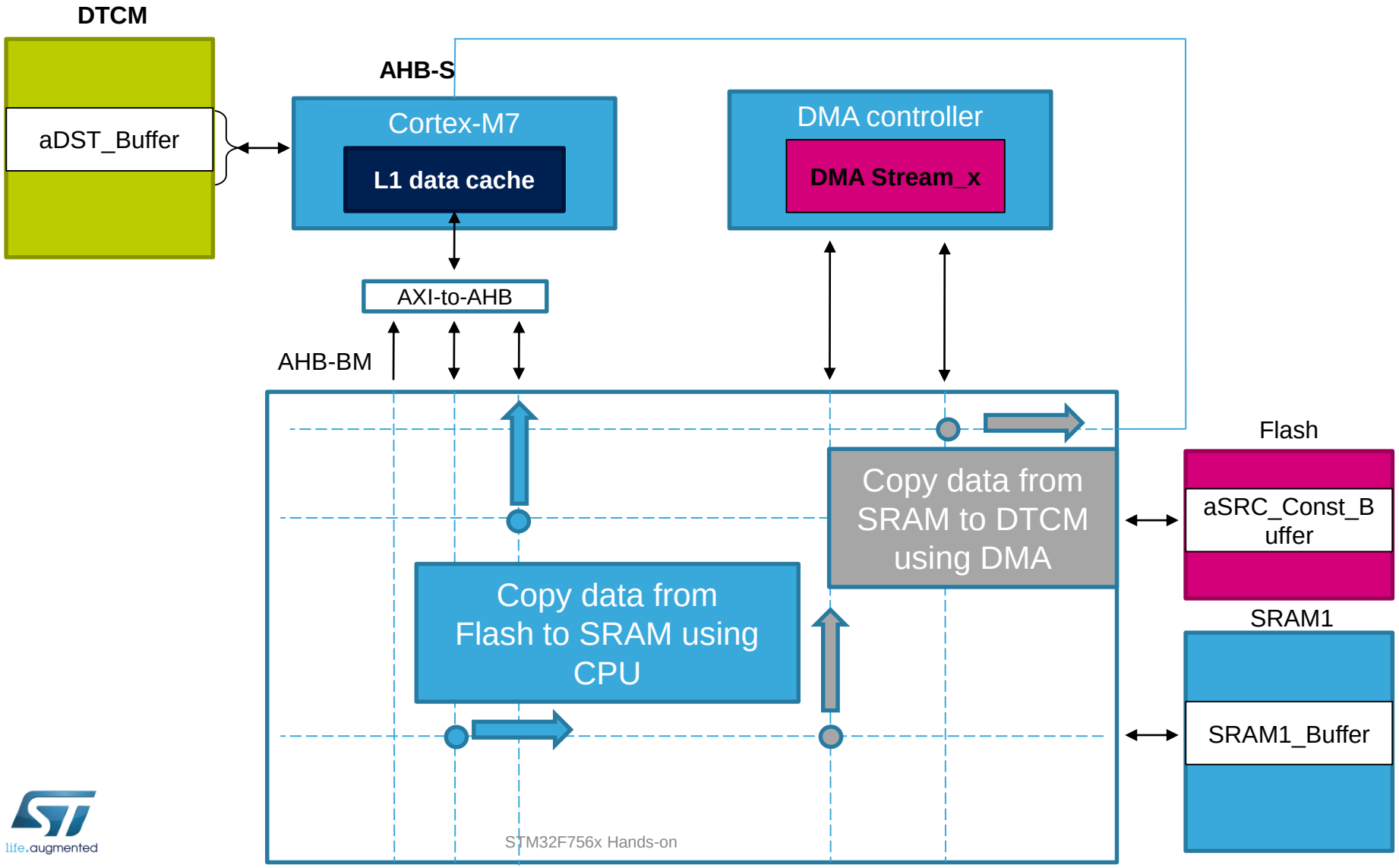
- STM32F7xx_Cache_WorkShop\Projects\STM32F746ZG-Nucleo\Examples\Cortex\CORTEXM_Cache_HandsOn
  - Use MDK-ARM Toolchain



Hands-on Description

- CPU fills first 128 bytes of SRAM1 at address 0x20010000 with 0x55 pattern.
- CPU copy 128 byte constant pattern from Flash memory (aSRC_Const_Buffer) to SRAM1 at address 0x20010000 (pBuffer).
- CPU configures and enables DMA to perform data transfer from SRAM1 at 0x20010000 to aDST_Buffer defined in DTCM RAM.
  - LED1 is ON when DMA transfer complete occurs.
- CPU compares data read by DMA aDST_Buffer with constant pattern from Flash memory (aSRC_Const_Buffer)
  - LED2 is ON if a mismatch is detected between data in aDST_Buffer and aSRC_Const_Buffer buffers.
  - If compare is successful LED3 is ON

Data transfer paths



Cortex-M7 Data coherency – Expected results

nds-on

- **Step-0**: Build the project and try the example
  - By default Data cache is disabled. Expected result LED1& LED3 are ON
- **Step-1**: Enable Data cache, CMSIS defines the function to turn on D-Cache.
  - In main.c file , add data cache enable function . To be added at Step1.
  - Build the project and Run. Doesn't work?
  - Expected result LED1 & LED2 are ON
- **Step-2**: Perform cache maintenance operation after writing data to cacheable memory region, by forcing a D-cache clean operation by software (all dirty lines will be write-back to SRAM1). CMSIS defines the function to clean D-Cache.
  - In main.c file, add clean data cache function at step2
  - Build the project and Run. Does it work?
  - Expected result LED1 & LED3 is ON

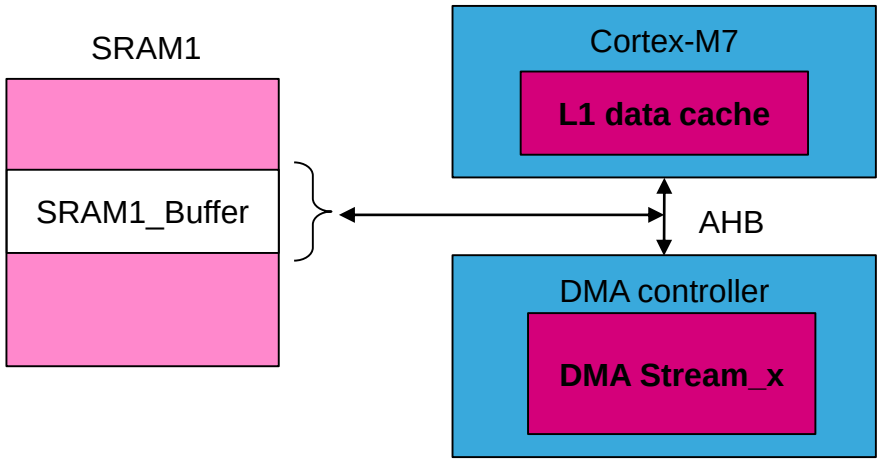
Cortex-M7 Data coherency – Expected results

- **Step-3**: Perform cache maintenance operation by modifying the MPU attribute of SRAM1 from write-back (default) to write-through policy. Enable MPU and configure SRAM1 attribute space to WT
  - In main.c file comment /* SCB_CleanDCache(); */
  - In main.c file uncomment /* MPU_Config(); */
  - Build the project and Run. Doesn't work? Fix MPU configuration.
  - Expected result LED1 & LED3 are ON

- **Step-4**: Perform cache maintenance operation , by forcing the write-through policy for all writes. This can be enabled by setting "Force Write-Through in the D-cache" bit in "CACR controls register".
  - In main.c file comment /* MPU_Config(); */
  - In main.c file uncomment /* __FORCE_WRITE_THROUGH(); */
  - Build the project and Run. Doesn't work?
  - Expected result LED1 & LED3 are ON

Data coherency between Cortex-M7 and DMA

- In this example of data buffer shared between the Cortex-M7 and a DMA:



- The data coherency between the core and the DMA is ensured by either:
  1. Cache inhibiting on SRAM1 buffers
  2. The SRAM1 buffer is cache enabled with write-back policy with coherency ensured by software (clean or invalidate D-Cache)
  3. The SRAM1 buffer is cache enabled with write-through policy
  - In this example DMA destination buffer is not cacheable by firmware architecture definition (placed in DTCM). incoherency between the CM7 and a DMA can be seen also on shared reception buffers.

Thank you

www.st.com/stm32