Armv8-M Mainline Memory Model

Learning objectives

- **At the end of this module you will be able to:**
  - List the different partitions of the Armv8-M Mainline address space
  - Differentiate the Arm Architecture memory types
  - Identify the different memory attributes and their effects
  - Characterize the effects of endianness on the Armv8-M processors
  - Recognize the different Barrier instructions describe when they are required

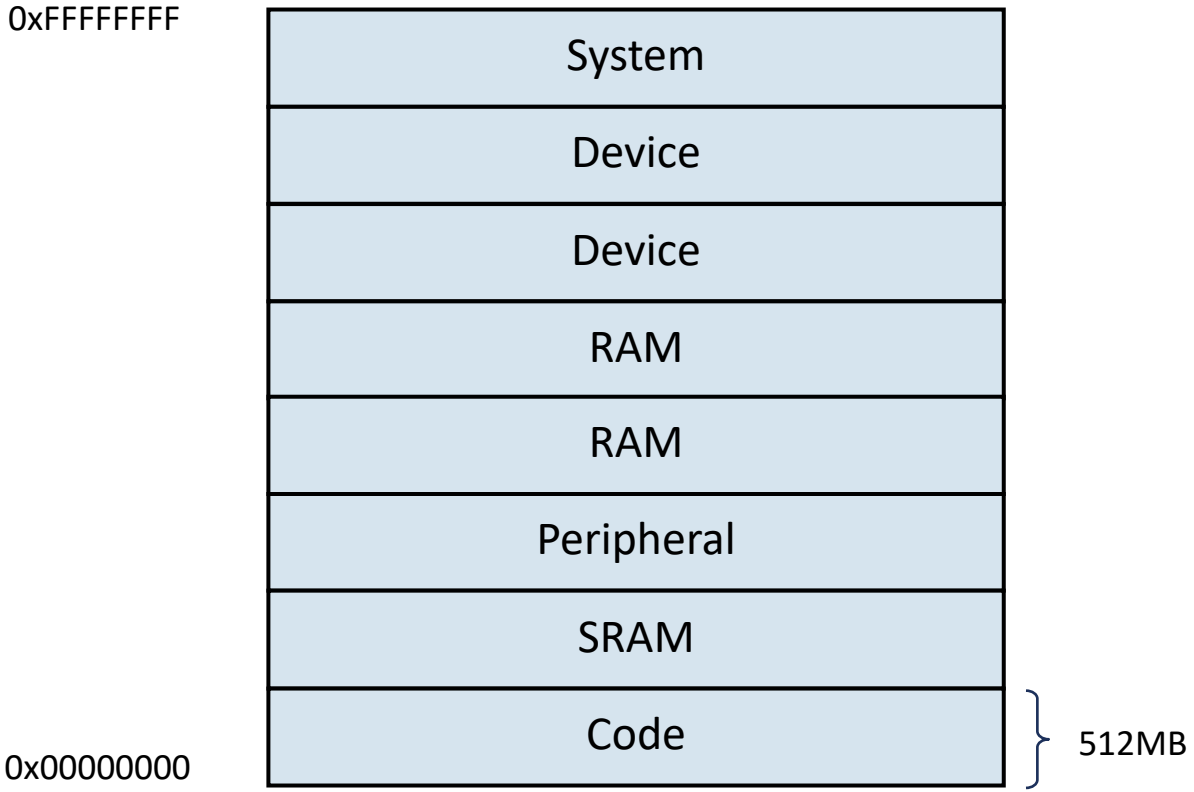
Agenda

- **Memory Address Space**
- Memory Types and Attributes
- Endianness
- Barriers

System address map

- **Armv8-M is a memory-mapped architecture**
  - Shared address space for physical memory and processor control & status registers

- **Memory is divided into 8 x 512MB segments**



Memory segments

- **Code 0x00000000 – 0x1FFFFFFF**
  - Memory to hold instructions
    - Typically ROM or flash memory
- **SRAM 0x20000000 – 0x3FFFFFFF**
  - Fast SRAM memory, usually on-chip RAM
- **2x RAM 0x60000000 – 0x9FFFFFFF**
  - Typical RAM memory, usually off-chip RAM

Code execution allowed

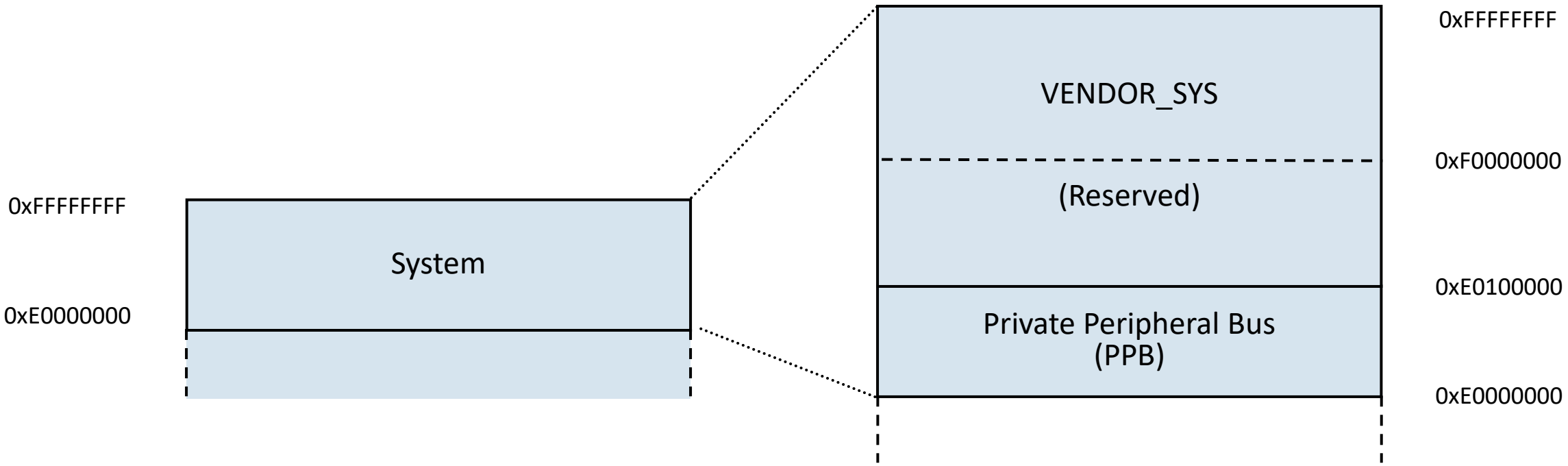
- **Peripheral 0x40000000 – 0x5FFFFFFF**
  - Peripheral memory space, on-chip
- **2x Device 0xA0000000 – 0xDFFFFFFF**
  - Peripheral memory space, off-chip
- **System 0xE0000000 – 0xFFFFFFFF**
  - Contains memory mapped registers

Code execution not allowed

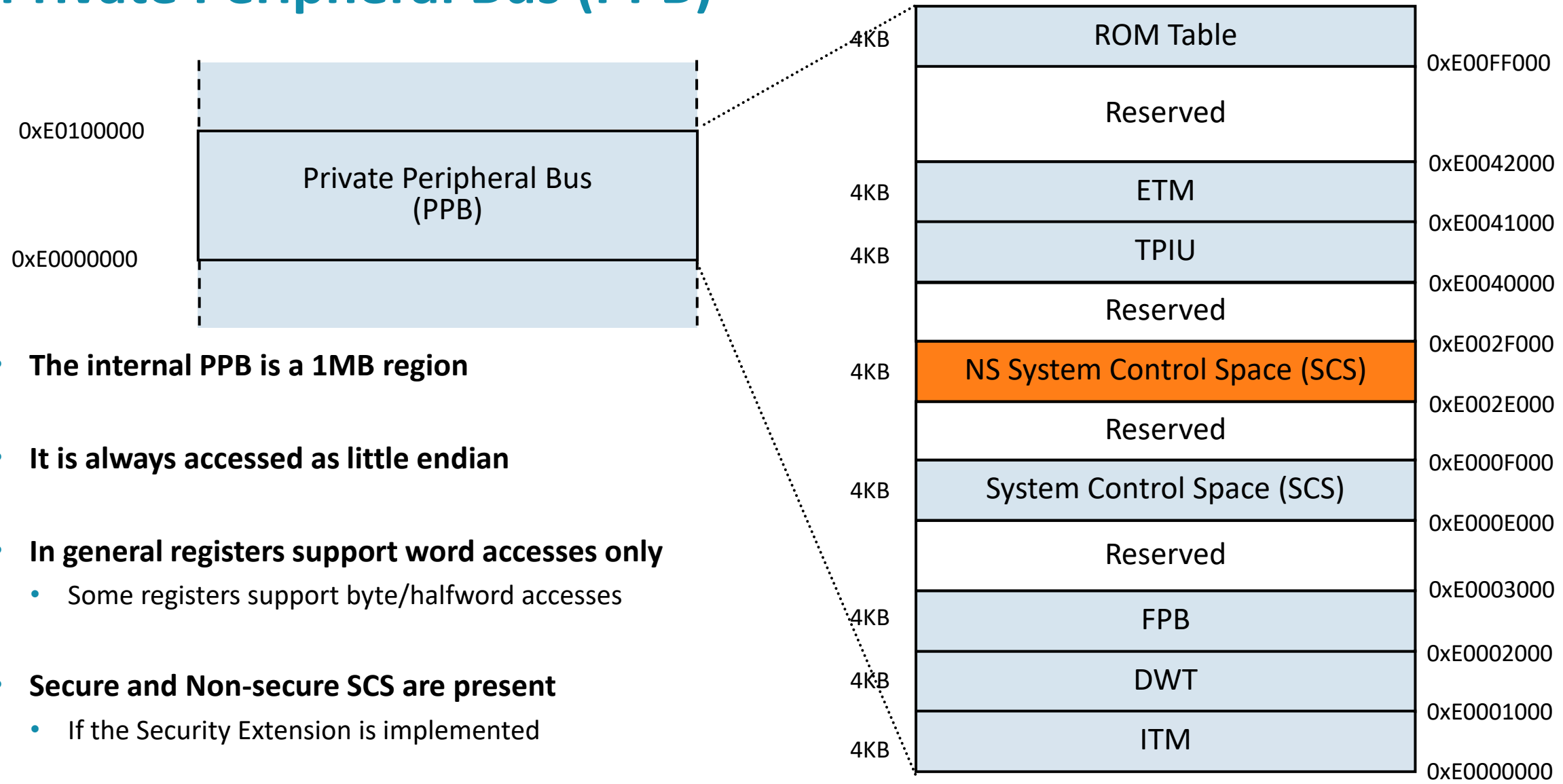
eXecute Never (XN)

System segment

- **Segment for control & configuration of the processor**
  - Including resources like NVIC, System Timer or Debug
- **Top of memory (511MB) can be used for adding additional implementation-defined system space**
  - Arm recommends that the top 256MB be used for Vendor Specific system usage



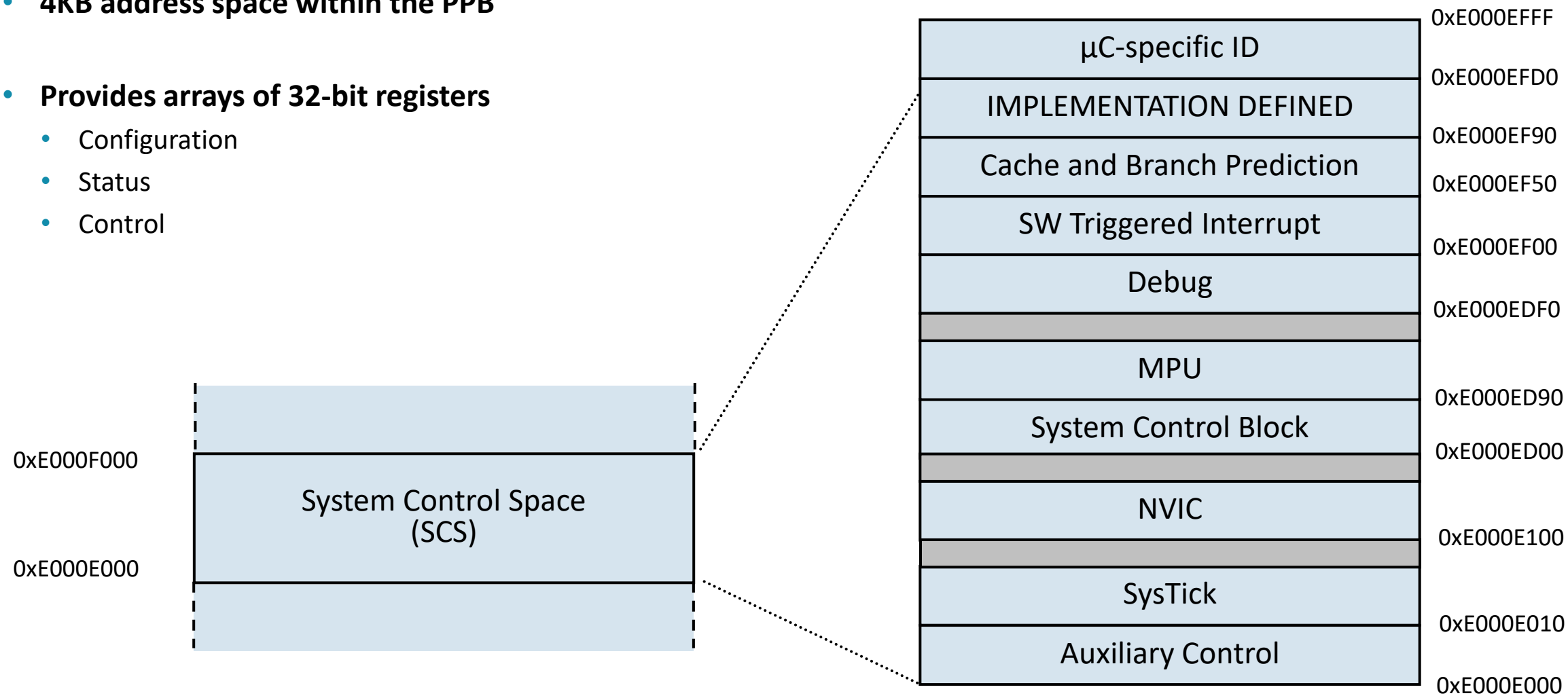
Private Peripheral Bus (PPB)



- **The internal PPB is a 1MB region**
- **It is always accessed as little endian**
- **In general registers support word accesses only**
  - Some registers support byte/halfword accesses
- **Secure and Non-secure SCS are present**
  - If the Security Extension is implemented

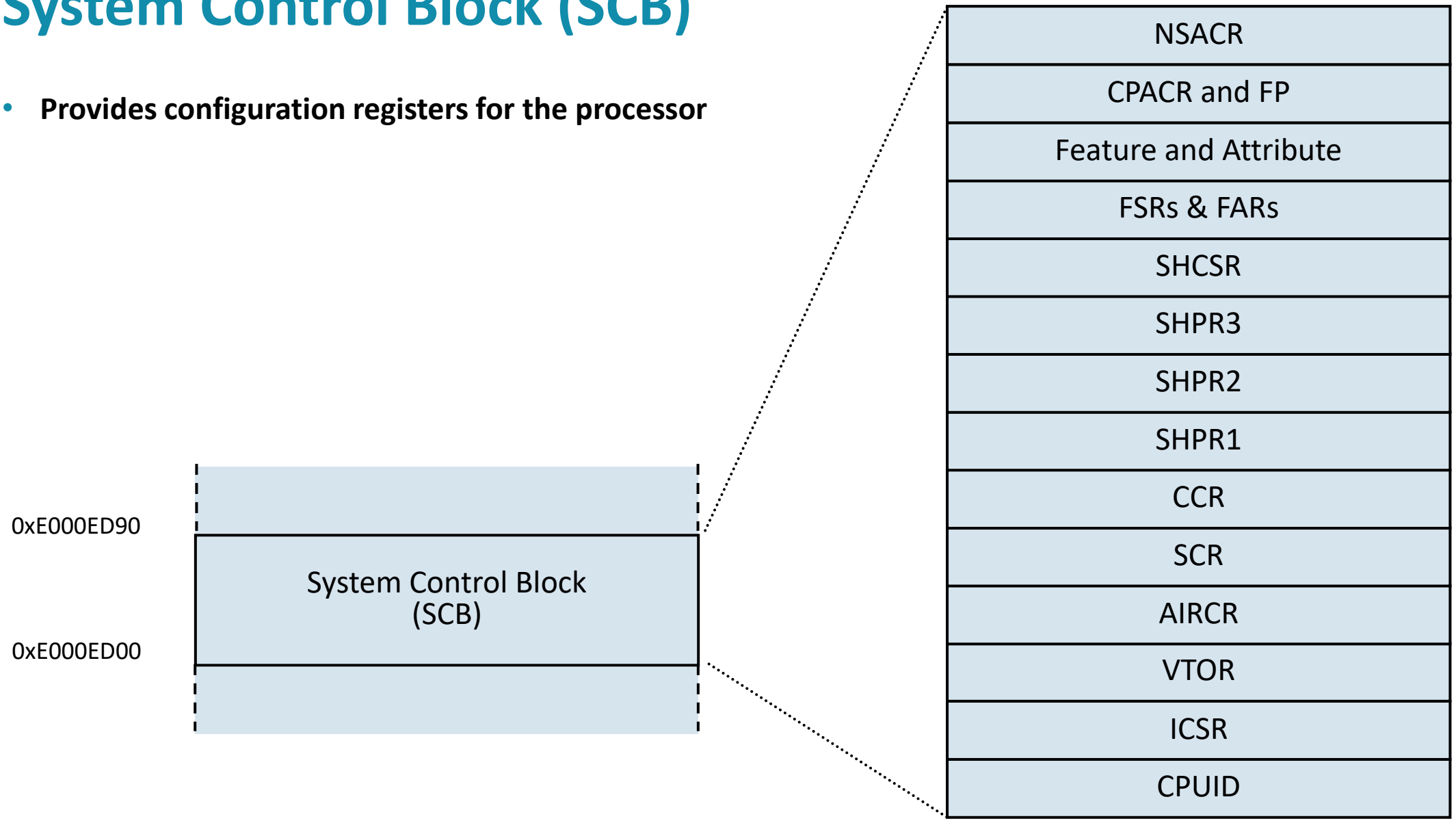
System Control Space (SCS)

- **4KB address space within the PPB**
- **Provides arrays of 32-bit registers**
  - Configuration
  - Status
  - Control



System Control Block (SCB)

- **Provides configuration registers for the processor**



SCB registers overview

NSACR	**Non-secure Access Control Register**
CPACR and FP	**Coprocessor Access Control and FP registers**
Cache	**Cache ID, Size and Type registers**
Feature and Attribute	**Reserved for CPUID feature registers**
FSRs & FARs	**Fault status and address registers**
SHCSR	**System Handler Control and State Register**
SHPRx	**System Handler Priority registers 1, 2 and 3**
CCR	**Configuration and Control Register**
SCR	**System Control Register**
AIRCR	**Application Interrupt and Reset Control Register**
VTOR	**Vector Table Offset Register**
ICSR	**Interrupt Control and State Register**
CPUID	**CPUID Base Register**

SCB Registers - Cache and Coprocessor

NSACR	**Non-secure Access Control Register**
CPACR	**Coprocessor Access Control Register**
CSSELR	**Cache Size Selection Register**
CCSIDR	**Cache Size ID Register**
CTR	**Cache Type Register**
CLIDR	**Cache Level ID Register**
Reserved	
AFSR	**Auxiliary Fault Status Register**

Agenda

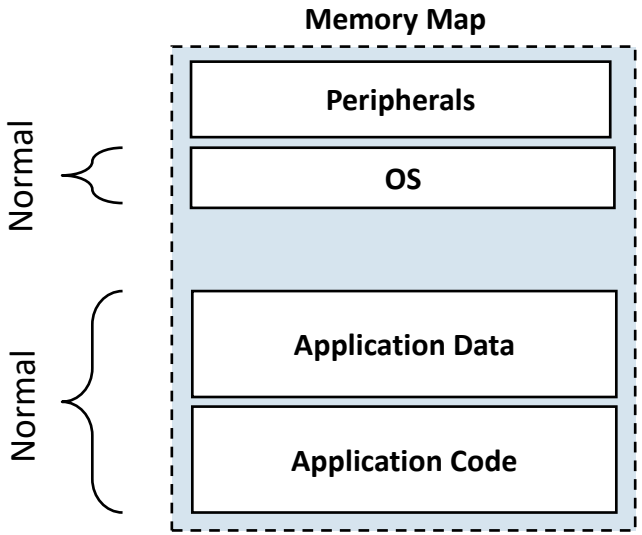
- Memory Address Space
- **Memory Types and Attributes**
- Endianness
- Barriers

Memory Types

- **Each defined memory region has a specified memory *type***
- **The memory type affects how the processor can access the region**
- **There are two mutually exclusive memory types**
  - Normal memory
  - Device memory
- **Additional attributes are specified to control:**
  - Access permissions
  - Execute permissions
  - Shareability
  - Cacheability

Normal memory (1)

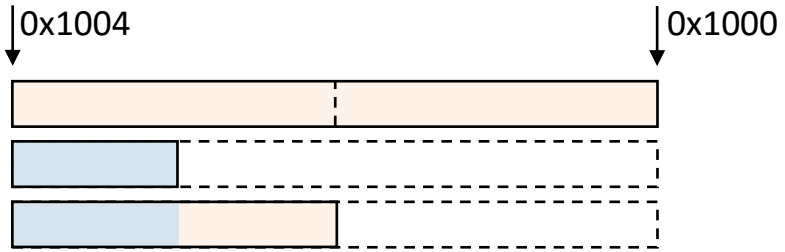
- **Addresses defined as Normal are used for code and most data regions**
- **Normal memory gives the best performance because it imposes the fewest restrictions**
  - Allows the processor to re-order, repeat and merge accesses
- **For optimal performance, application code and data should be marked as Normal**
  - Ordering can still be enforced when required using explicit barrier operations
- **Address regions marked as Normal can be accessed *speculatively***
  - Data or instructions fetched from memory before being explicitly referenced
  - Speculative access may, for example, be caused by:
    - Branch prediction
    - Out of order data loads
    - Speculative cache line fills



Normal memory (2)

- **Normal memory implements a “weakly ordered” memory model**
  - There is no requirement for Normal accesses to complete in order with respect to other Normal and Device accesses
  - However, a PE must handle dependencies

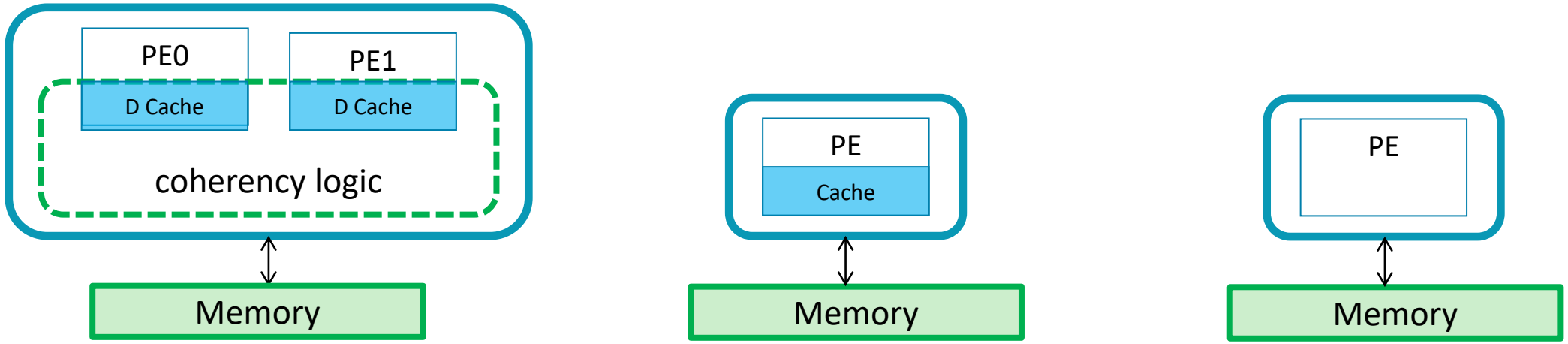
```
STR  r0, [0x1000]
STRB r0, [0x1003]
LDRH r0, [0x1002]
```



- **In the example, the accesses are to overlapping addresses**
  - PE must ensure the memory is updated as if the **`STR`** and **`STRB`** occurred in order
    - These accesses might be merged into a single access
  - The **`LDRH`** must return the most up-to-date value

Normal memory (3)

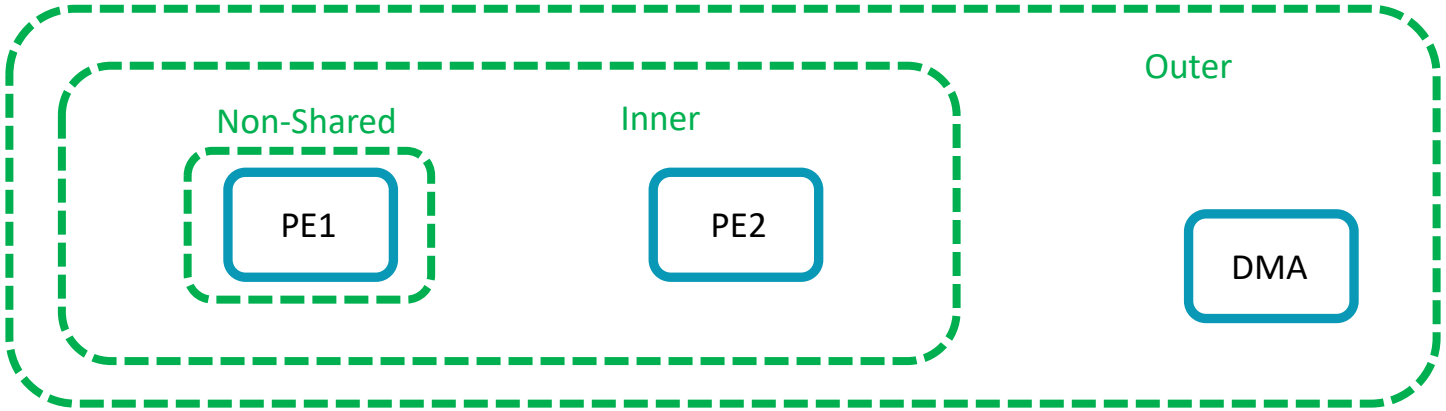
- **Different types of Normal memory can become weaker to allow extra behaviours**
  - E.g. Cacheable means the data *may* be cached not that it *must* be cached
- **The shareable attribute indicates whether other masters need to access data**
  - Complex systems may make data visible using extra coherency logic
  - PEs with non-coherent caches can share data by forcing memory to be not cached
  - Non-cacheable PEs don't need to do anything special
- **Software that specifies the correct memory type for the desired behaviour will then work correctly on any implementation**



Normal memory - shareable attribute

- **The shareable attribute is used to define whether a location is shared with multiple processors**
  - Non-shareable – only used by this observer
  - Inner Shareable / Outer Shareable – shared with other observers
    - The division between inner and outer is IMPLEMENTATION DEFINED

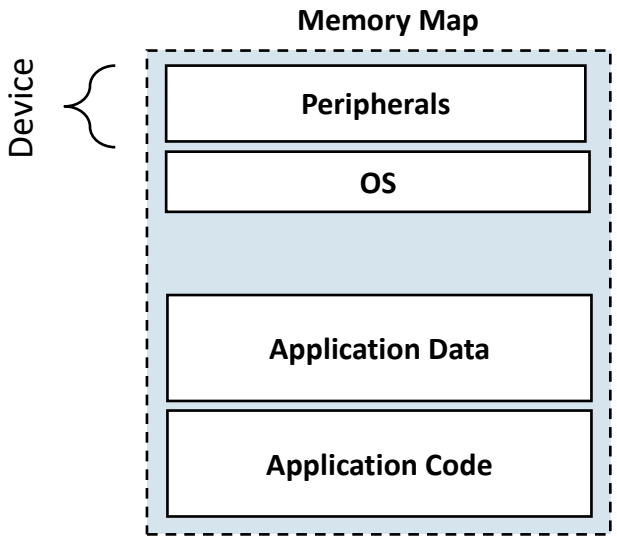
- **Example:**



- **These attributes can define sets of observers for which the shareability attributes make the data/unified caches transparent for data accesses**

Device memory (1)

- **The Device type is used for regions where accesses can have side-effects**
  - For example – a write to a peripheral's control register may trigger an interrupt as a side effect
  - Typically only used for peripherals
- **Device type imposes more restrictions on the core**
- **Attempting to execute from a region marked as Device is UNPREDICTABLE**
- **Speculative *data* accesses can not be performed to Device regions**
  - Speculative *instruction* fetches are allowed to any executable address
  - **Device regions should always be marked Execute Never**



Device memory (2)

- **Four variants of Device are available:**
  - Device-nGnRnE - most restrictive
  - Device-nGnRE
  - Device-nGRE
  - Device-GRE - least restrictive

- **Gathering (G/nG)**
  - Determines whether multiple accesses can be merged into a single bus transaction
  - nG: number/size of accesses on the bus = number/size of accesses in code

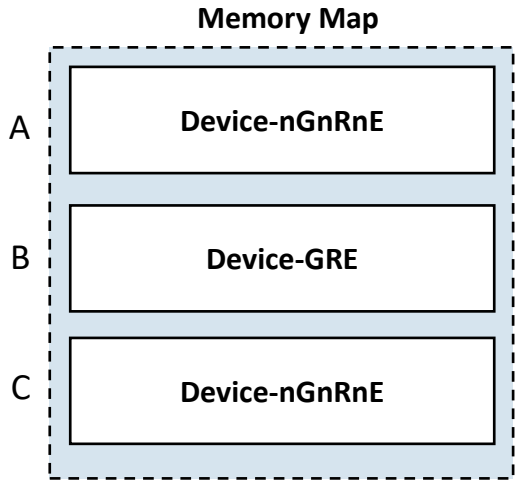
- **Re-ordering (R/nR)**
  - Determines whether accesses to same device can be re-ordered
  - nR: accesses to the same IMPLEMENTATION DEFINED block size will appear on the bus in program order

- **Early Write Acknowledgement (E/nE)**
  - Indicates to the memory system whether a buffer can send acknowledgements
  - nE: The response should come from the end slave, not buffering in interconnect

Ordering of Device accesses

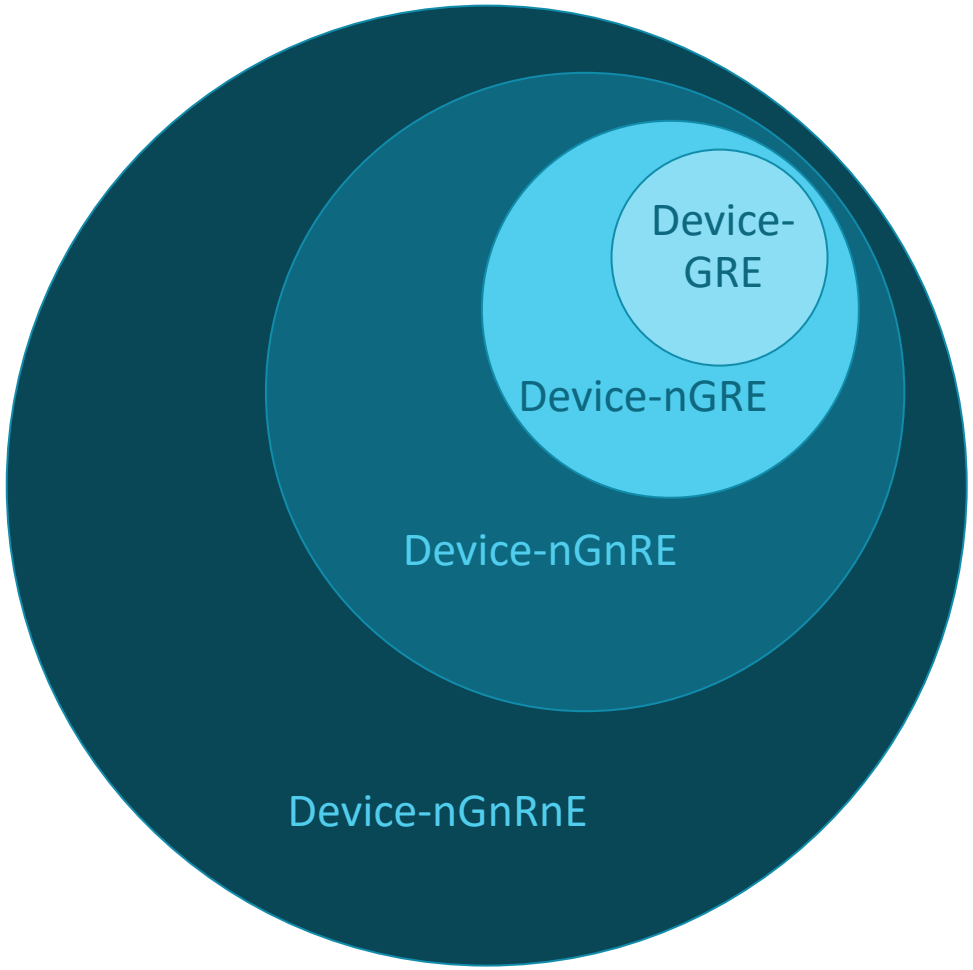
- **Example instruction sequence:**

```
LDR    r0, A
LDR    r1, B
LDR    r2, A+8
LDR    r3, C
LDR    r4, B+8
```



- **Effect of ordering rules:**
  - The two accesses to region A are guaranteed to be in program order with respect to each other
  - The two accesses to region B are NOT guaranteed to be in program order with respect to each other, or the accesses to regions A and C
  - It is IMPLEMENTATION DEFINED whether the accesses to region A will occur in program order with respect to accesses to region C

Stronger to weaker Device memory



- **Device-nGnRnE is the *strongest* memory type**
  - Defines rules memory accesses must obey
- **As the memory type *weakens* those rules are relaxed**
  - E.g. allowing gathering
- **Access rules for stronger memory are still legal for the weaker memory types**
  - For example: the Reordering attribute means accesses *may* be reordered but not that they *must* be reordered
- **An implementation may then use the same behavior for different memory types**
  - Must use the behaviour of the strongest type
  - Bus infrastructures may not be able to express all memory types
- **Software can specify the weakest type necessary for correct operation**
  - If accesses are upgraded to a stronger type the behaviour will still be correct

Address map overview

Address	Name	Memory Type(s)	XN	Cache	Description / Supported Memory
0xE0000000- 0xFFFFFFFF	System	Device nGnRnE	XN	-	• Vendor system region (VENDOR_SYS) • Private Peripheral Bus (PPB)
0xC0000000- 0xDFFFFFFF	Device	Device nGnRE Shareable	XN	-	• Peripherals accessible only to the PE
0xA0000000- 0xBFFFFFFF	Device	Device nGnRE Shareable	XN	-	• Peripherals accessible to all Requesters
0x80000000- 0x9FFFFFFF	RAM	Normal	-	WT	• Memory with WT cache attributes
0x60000000- 0x7FFFFFFF	RAM	Normal	-	WBWA	• Write-back, Write-allocate L2/L3
0x40000000- 0x5FFFFFFF	Peripheral	Device nGnRE Shareable	XN	-	• On-chip peripheral address space
0x20000000- 0x3FFFFFFF	SRAM	Normal	-	WBWA	• SRAM • On-chip RAM
0x00000000- 0x1FFFFFFF	Code	Normal	-	WT	• ROM • Flash Memory

- **XN indicates an eXecute Never region**
  - Any attempt to execute code from an XN region faults, generates a HardFault exception
- **The Cache column indicates the cache policy (write-through or write-back write allocate)**
  - See appendix for more information about caches and cache policies

Agenda

- Memory Address Space
- Memory Types and Attributes
- **Endianness**
- Barriers

Endianness

- **Endianness determines how contents of registers relate to the contents of memory**
  - Arm registers are word (4 bytes) width
  - Arm addresses memory as a sequence of bytes
- **Arm processors are little-endian**
  - But can be configured to access big-endian memory systems (selected at reset)

Little-endian memory system

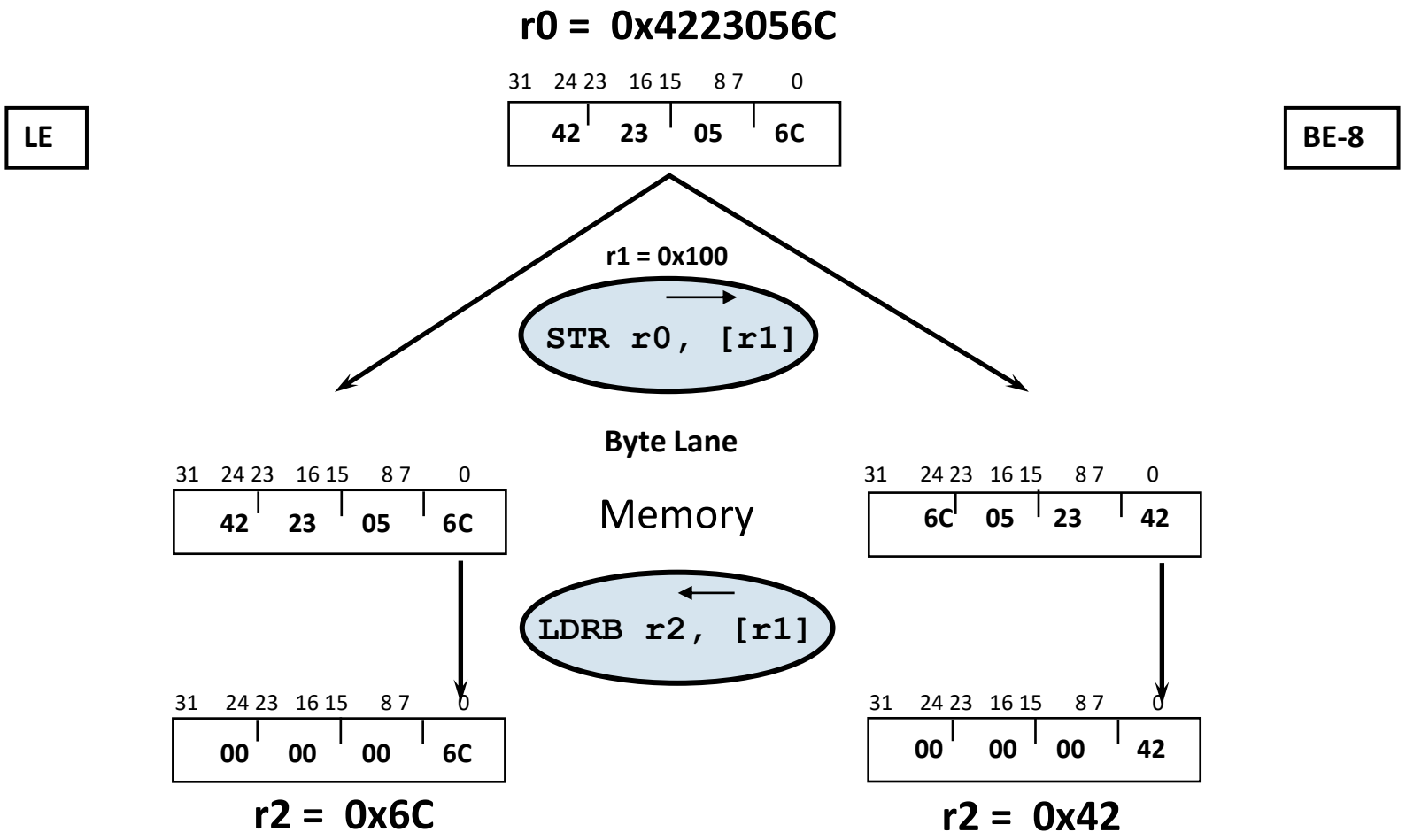
- **Least significant byte is at lowest address**

Big-endian memory system

- **Most significant byte is at lowest address**

- **Arm Cortex-M Profile cores support two models of endianness**
  - **LE** **Little-Endian**
  - **BE-8** **Byte invariant Big-Endian** (introduced in architecture v6)
- **Instruction fetches and PPB accesses are always LE on Cortex-M cores**

Effect of endian configuration



Agenda

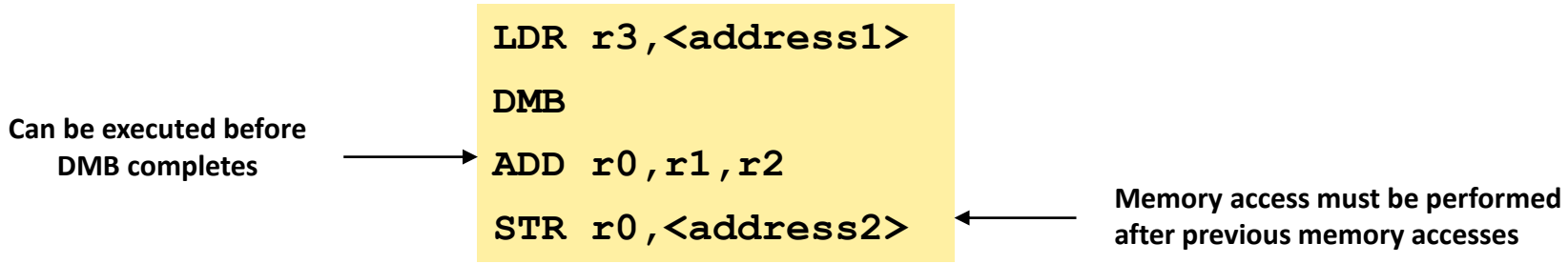
- Memory Address Space
- Memory Types and Attributes
- Endianness
- **Barriers**

Memory barrier instructions

- **Classic Arm processors, such as the ARM7TDMI, execute instructions and complete data accesses in program order**
- **Some of the later Arm processors, for example, Cortex-A15, can optimize the order of instruction execution and data accesses**
- **In some situations, it might be necessary to ensure that an operation has completed before continuing execution**
- **Arm processors provide barrier instructions to:**
  - Guarantee completion of any preceding load and store instructions
  - Flush any prefetched instructions
- **Where a PE does not perform out-of-order execution there are some situations where barriers are required, for example:**
  - Implementation requirements
    - Memory map switching and self-modifying code
  - Armv8-M architectural and portable code requirements, for example:
    - CONTROL register or VTOR updates and MPU Programming

Data Memory Barrier (DMB)

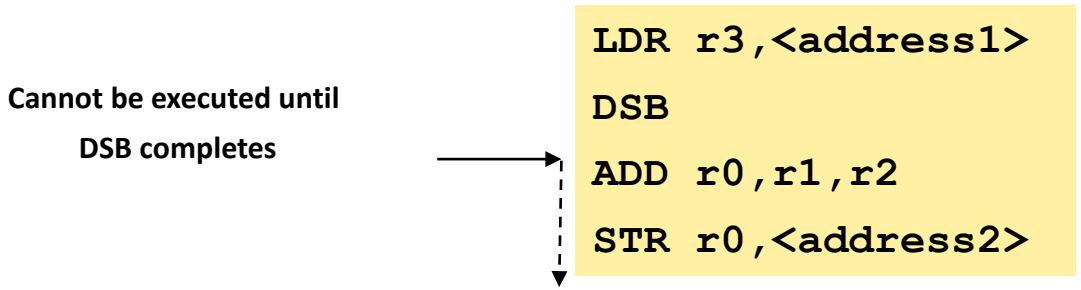
- **Ensures that all explicit memory accesses that appear in program order before the DMB instruction are observed before any explicit memory accesses that appear in program order after the DMB instruction**



- **Memory accesses to address regions marked as Device, such as the System Control Block, do not require the use of DMB instructions relative to other address regions marked as Device**
  - Relative to address regions marked as Normal they will require barriers

Data Synchronization Barrier (DSB)

- **Ensures that all explicit memory transactions complete before it finishes**
- **No further instructions may execute until the DSB instruction completes**
  - This includes instructions that update system control registers



Instruction Synchronization Barrier (`ISB`)

- **Generates a Context Synchronization Operation (CSO)**
  - A CSO guarantees visibility of any change to a system control register
- **Ensures that the pipeline of the processor is flushed**
  - Instructions in pipeline stages are cleaned out
    - Including all instruction buffers
  - Instructions are fetched again from memory system
- **Other CSOs include**
  - Exception entry
  - Exception return
  - Debug event entry
  - Debug event return

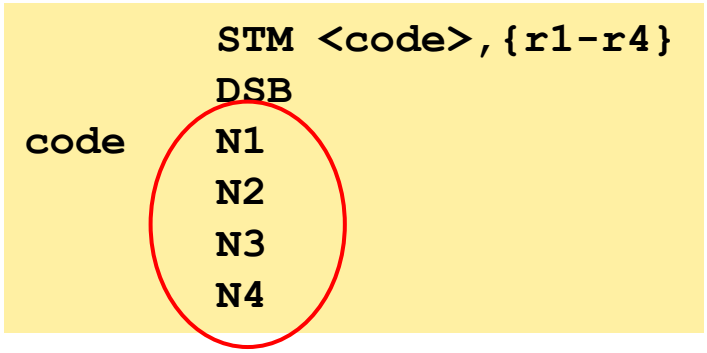
Compiler barriers

- **Memory barrier instructions are interpreted by the processor**
  - They do not guarantee to have the same affect on an Arm C/C++ compiler
- **Compilers also provide barrier intrinsics to prevent optimizations across code sequences**
- **CMSIS functions use both types of barriers if necessary, e.g., `SCB_EnableDCache()`**

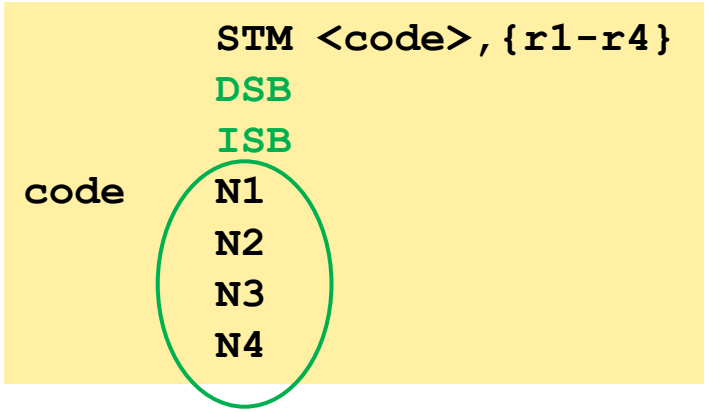
```
do {
       SCB->DCISW = (((sets << SCB_DCISW_SET_Pos) & SCB_DCISW_SET_Msk) |
                     ((ways << SCB_DCISW_WAY_Pos) & SCB_DCISW_WAY_Msk)  );
       #if defined ( __CC_ARM )
         __schedule_barrier();
       #endif
     } while (ways-- != 0U);
   } while(sets-- != 0U);
   __DSB();

   SCB->CCR |=  (uint32_t)SCB_CCR_DC_Msk;  /* enable D-Cache */

   __DSB();
   __ISB();
```

Example: self modifying code

- **Scenario: “r1-r4 contain instructions N1-N4 that will overwrite I1-I4”**
- **Question: “How can you ensure N1-N4 will be executed if I1-I4 are in pipeline?”**
- **Answer:**
  - Ensure the `STM` completes using a **DSB**
  - Flush the pipeline using an Instruction Synchronization Barrier (**ISB**)



Instructions `I1` – `I4` might be
in the processor pipeline in a prefetch buffer



ISB ensures that all instruction
buffers and the pipeline are flushed

Example: CONTROL Register

- **The CONTROL register is a 2-bit special-purpose register**
  - CONTROL.nPRIV defines the execution privilege in Thread mode
  - CONTROL.SPSEL defines the stack to be used

```
MOVS r0, #2
MSR  CONTROL, r0
ISB
I1
I2
```

ISB ensures that all instruction buffers and the pipeline are flushed

- **If the program updates the CONTROL register, an ISB ensures that the new CONTROL configuration is used by subsequent instructions**

Further usage of memory barriers

- **In addition to the previous examples, there might be other situations where memory barriers are required**

- **Architectural and portability requirements**
  - If you require your code to run on another binary compatible processor, e.g. Cortex-A53, then it might be necessary to insert a memory barrier

- **Device-specific requirements**
  - If your device contains any additional features such as multiple masters, caches or write buffers, there might be more situations where memory barriers are required
    - Additional support to ensure system correctness might be required

arm

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה