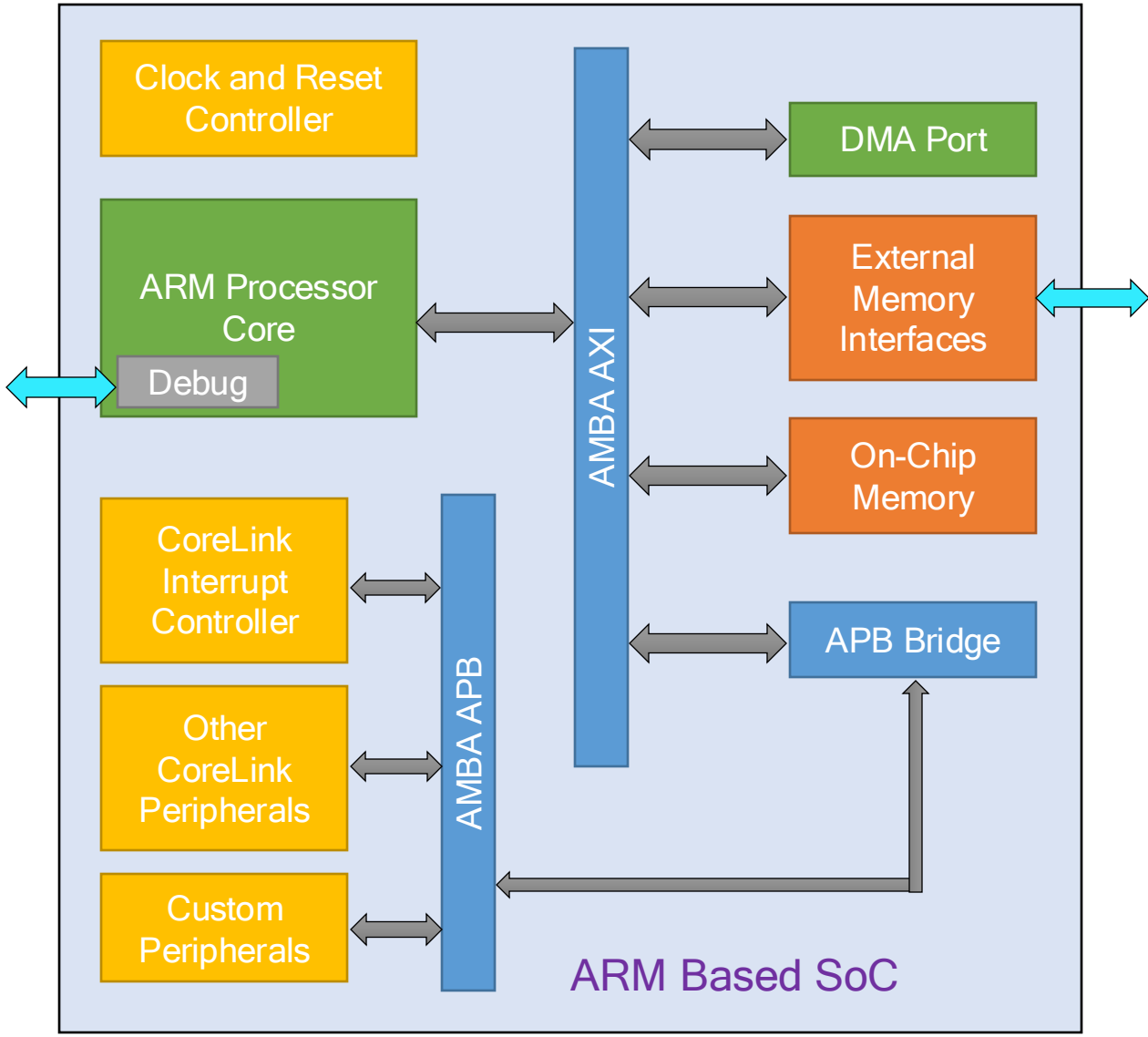
Introduction to ARM v7-M

How different from other ARM architectures

- No ARM instruction support
  - Thumb and Thumb-2 extensions only
- No Cache (M3 and M4), No MMU
- Debug is optimized for microcontroller applications
- Interrupt automatically save/restore state
- Exceptions are programmed in C
- No Coprocessor 15
  - All the registers are memory mapped
- Interrupt controller is part of the processor
- Bit banding capability
- NMI

Typical ARM based system

- ARM core was deeply embedded with the SoC
  - External debug and trace via JTAG or CoreSight interface
- Design can have both internal and external memories
  - Varying width, speed and size depending on the system requirements
- Can include ARM licensed CoreLink peripherals
  - Interrupt controller, since core has only two interrupt sources
  - Other peripheral and interfaces
- Can include on-chip memory from ARM artisan physical IP libraries
- Elements are connected using AMBA (Advanced Microcontroller Bus Architecture)



Processor modes, access levels and stacks

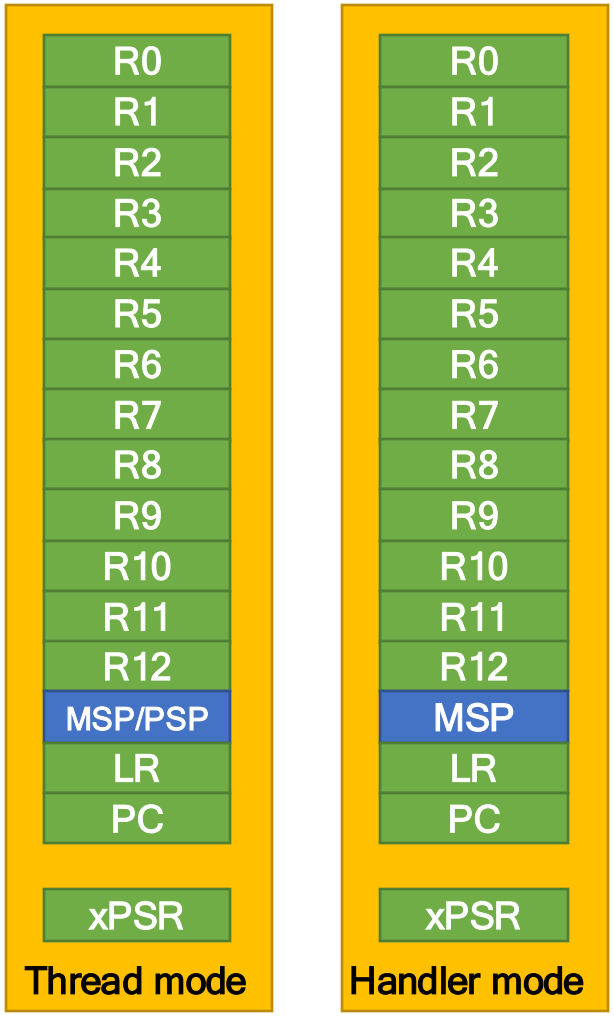
- Processor supports privileged and unprivileged operations
  - Used to control the access to memory and registers
- Two possible stacks
  - Main stack pointer (MSP) and Process stack pointer (PSP)ARM Cortex v7-M supports two processor modes

Processor mode	Usage	Access level	Stack	Description
Thread mode	Runs user code	• Can be privileged/Unprivileged • Controlled by **nPRIV** bit of CONTROL register	MSP/PSP	Default mode at reset
Handler mode	Runs kernel code	Always privileged	Always MSP	Default mode for all the exceptions/interrupts

- Processor always starts with Thread mode, when core meets any exceptions/interrupts, then core will change its mode to Handler mode, in order to service the ISR associated with exception/interrupt
- When the processor is in Thread mode, its possible to move the processor to non-privileged access level, once you move to NPAL being in Thread mode, its not possible to come back to privilege access level in Thread mode
- If you want to switch between the access levels use the CONTROL register
  - Note – It can be accessed only with privileged access level

Core registers

- Both Thread and Handler modes share same registers, except SP (which is also configurable)



Register	Usage	Access
xPSR	• IPSR register defines the exception number, if IPSR==0 then processor in thread mode else value is exception number and its in handler mode • T bit in EPSR defines current execution state of the processor, if T==0 then processor in ARM state, else processor in thumb state. Writing T=0 in cortex Mx processors causes usage fault	
CONTROL	• Selects privilege level for thread mode • Selects stack pointer (MSP/PSP) for thread mode • Defined whether fp is active in current context or not? ISB required to to ensure that changes in CONTROL register takes effect	• Privileged mode access only • R/W

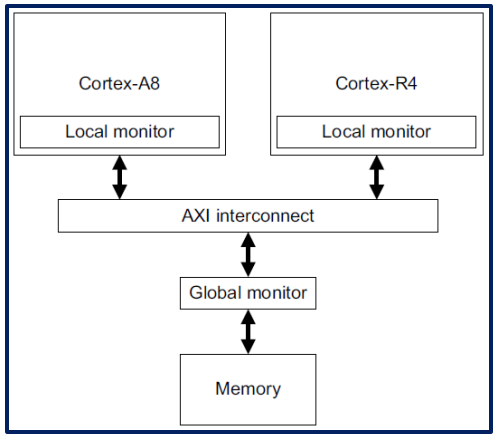
Power management

- Multiple sleep modes are supported
  - SLEEPING
  - DEEPSLEEP
    - Controlled by system control register
  - WIC based deep sleep
- Sleep now
  - WFI, WFE, SEV instructions (Wait for interrupt/event, Send event)
- Sleep on exit
  - Sleep immediately on return from last exception
- System clock gated in sleep modes
  - Sleep (SLEEPING) signal is exported allowing external system to be clock gated
  - NVIC interrupt interface stays awake
- WIC (Wake-up interrupt controller)
  - Optional external wakeup detector allows core to be fully powered down
  - Effective with state retention/power gating (SRPG) methodology

Core debug

- Single stepping
- ITM (Instruction Trace Macrocell)
  - Supports printf style software controlled trace
- Optional ETM (Embedded Trace Macrocell)
  - Provides instruction trace
- Access to all memory and registers via Debug Access Port (DAP)
- Event counters for code profiling
- Flash patch and breakpoint unit (FPB)
  - Remapping instructions from ROM to RAM
  - Breakpoint functionality
- Data watchpoint and Trace unit (DWT)
  - Detects data access to a specific address
  - Generates debug event or trace packet

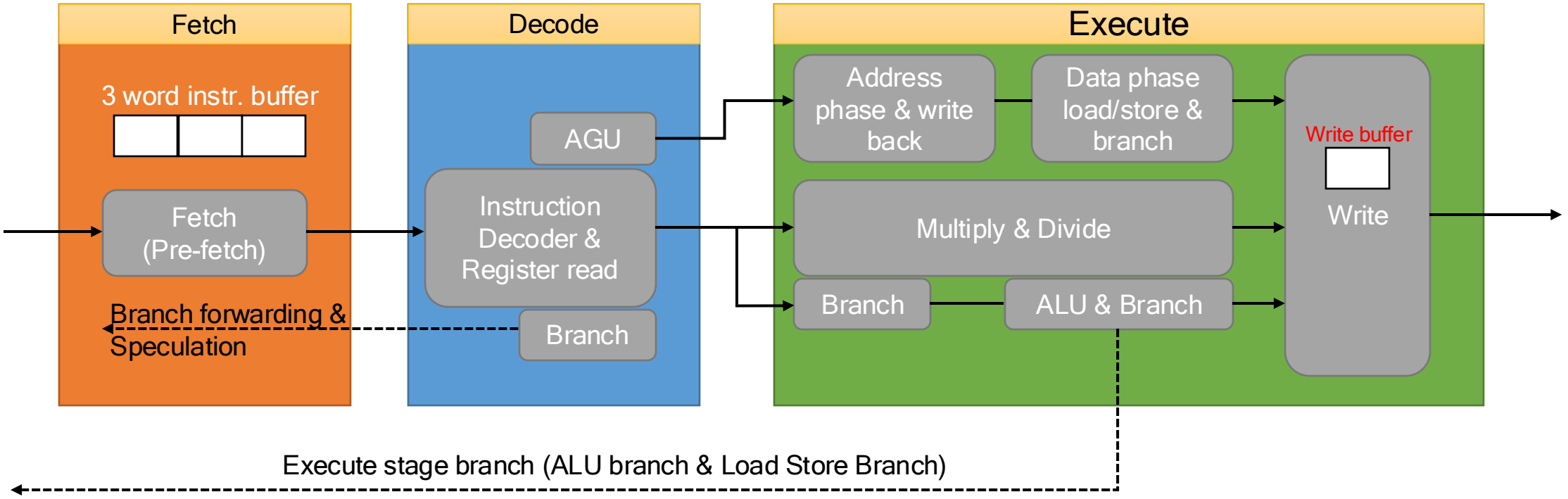
Exclusive memory access



- Exclusive monitor is a simple state machine, with the possible states open and exclusive
  - Execution of LDREX takes monitor state from open to exclusive
  - Execution of STREX brings monitor state from exclusive to open
- To support synchronization between processors, system must implement two sets of monitors
  - Global monitor
  - Local monitor
- When memory location marked as an non-shareable are checked against Local monitor only
- When memory location marked as an shareable are checked against both Local and Global monitors
- Store exclusive may be succeed even if its different memory location which is used by load exclusive instructions
  - Portable code must not make assumption on address checking
- If the location is cacheable, the synchronization takes place w/o any external bus transactions and external observers also will not get to know.
- Global monitor can tag **one address for each processor in the system**, that supports exclusive access
- LDREX will fail when others already initiated LDREX instruction
- Exclusive reservation granule (ERG)
  - When an exclusive monitor tags an address, the minimum region that can be tagged for exclusive access is called Exclusive reservation granule
  - The ERG is implementation defined in the range of 8-2048 bytes, in multiples of 2bytes
  - Portable code must not assume anything about ERG size
- Resetting monitors
  - When operating system performs context switch, it must reset local monitors to open state, to prevent false positives occurring
  - There 2 Ways to reset the local monitor
    - Clear-Exclusive instruction, CLREX will reset the local monitor
    - Use dummy load and store exclusive instructions

ARM v7-M - MicroArchitecture

Processor pipeline



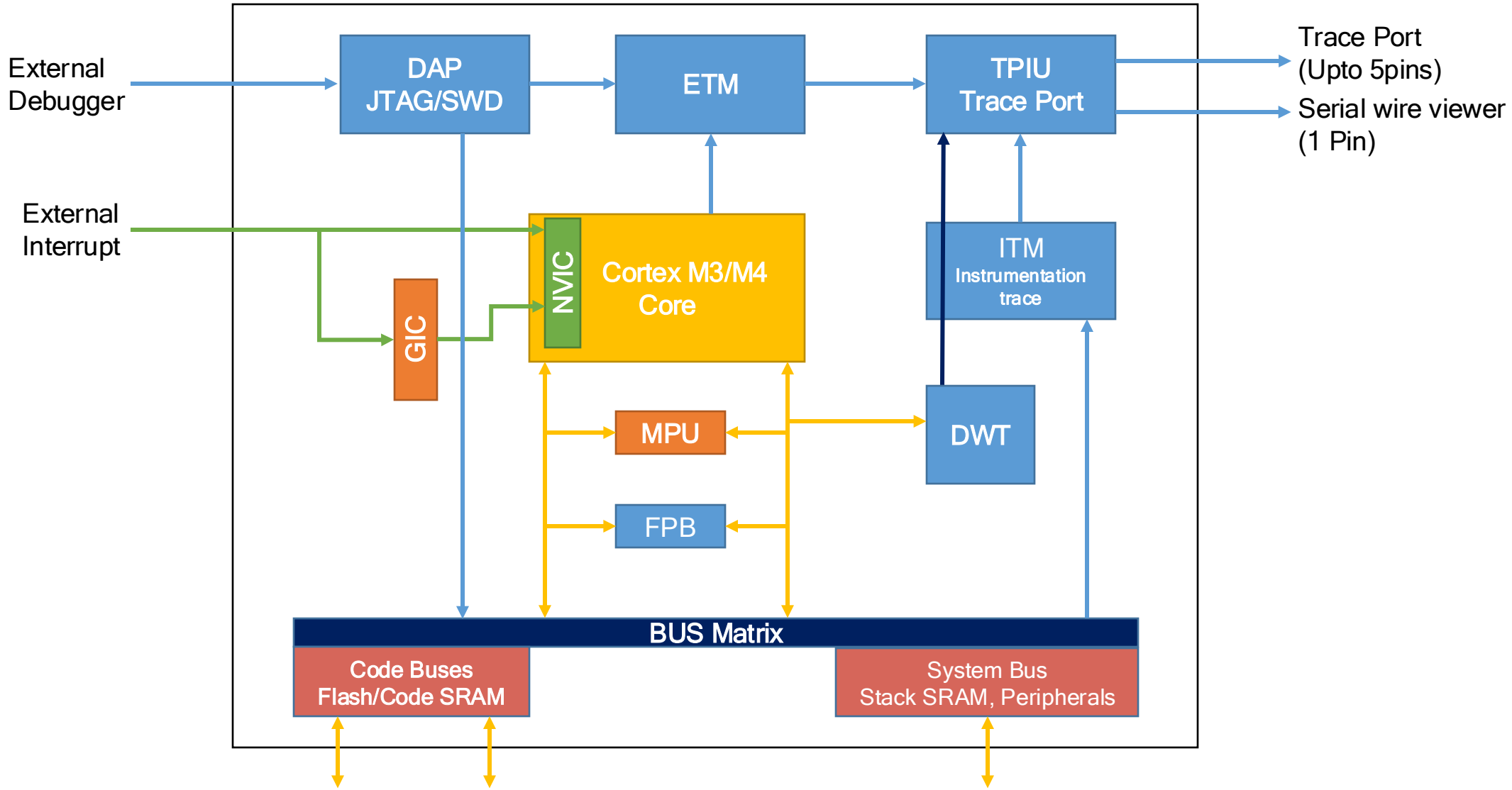
PC points to “Current instruction + 4”

Acronyms

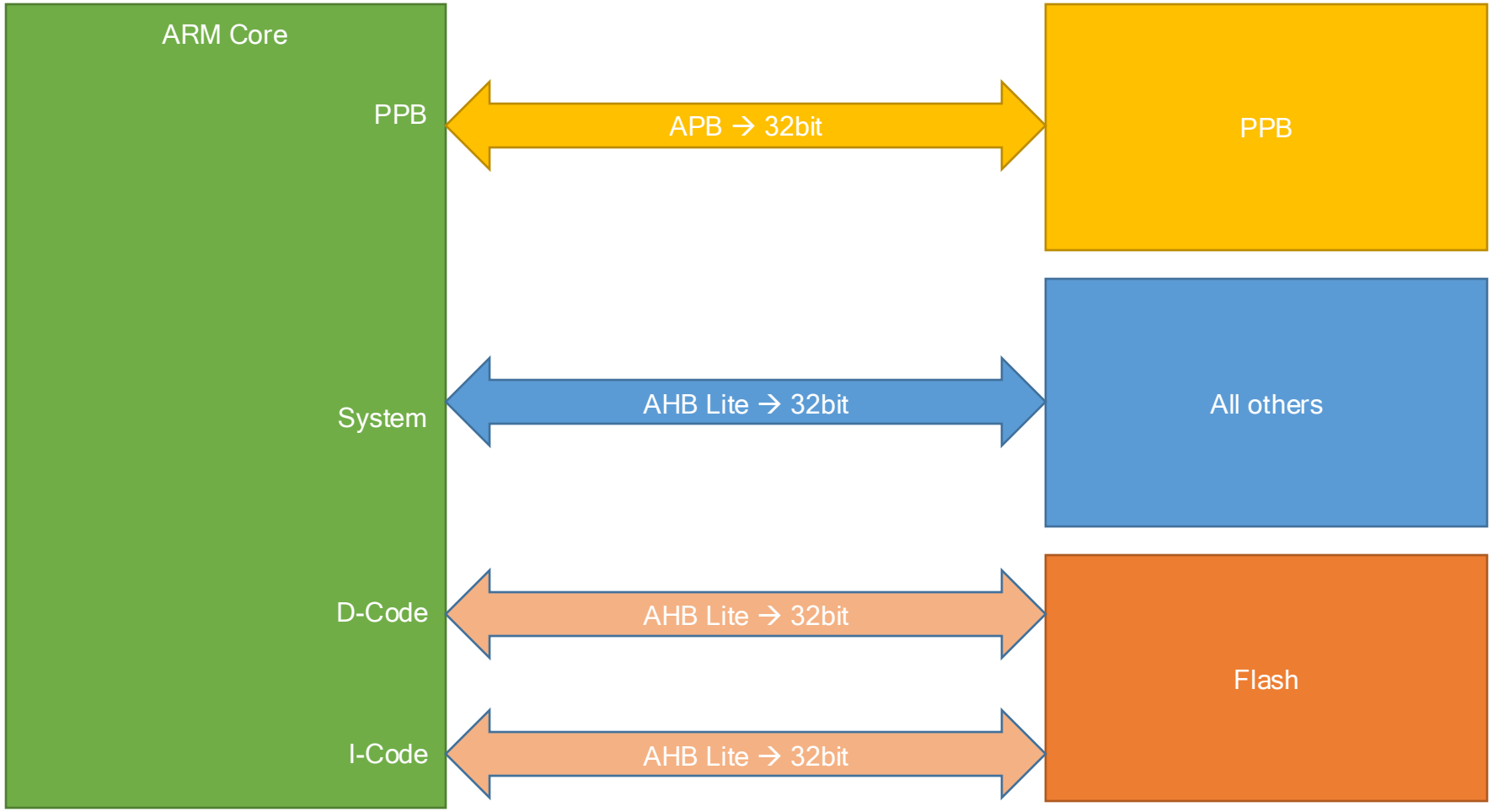
- AGU – Address Generation Unit

- Data is placed in the buffer so processor can continue the execution
- Data can be written to memory when possible
- Read require write buffer to have drained before access
- Write buffer allows store to complete before store cycle completes on BUS

High level Cortex M4 bus interfaces



High level Cortex M4 bus interfaces

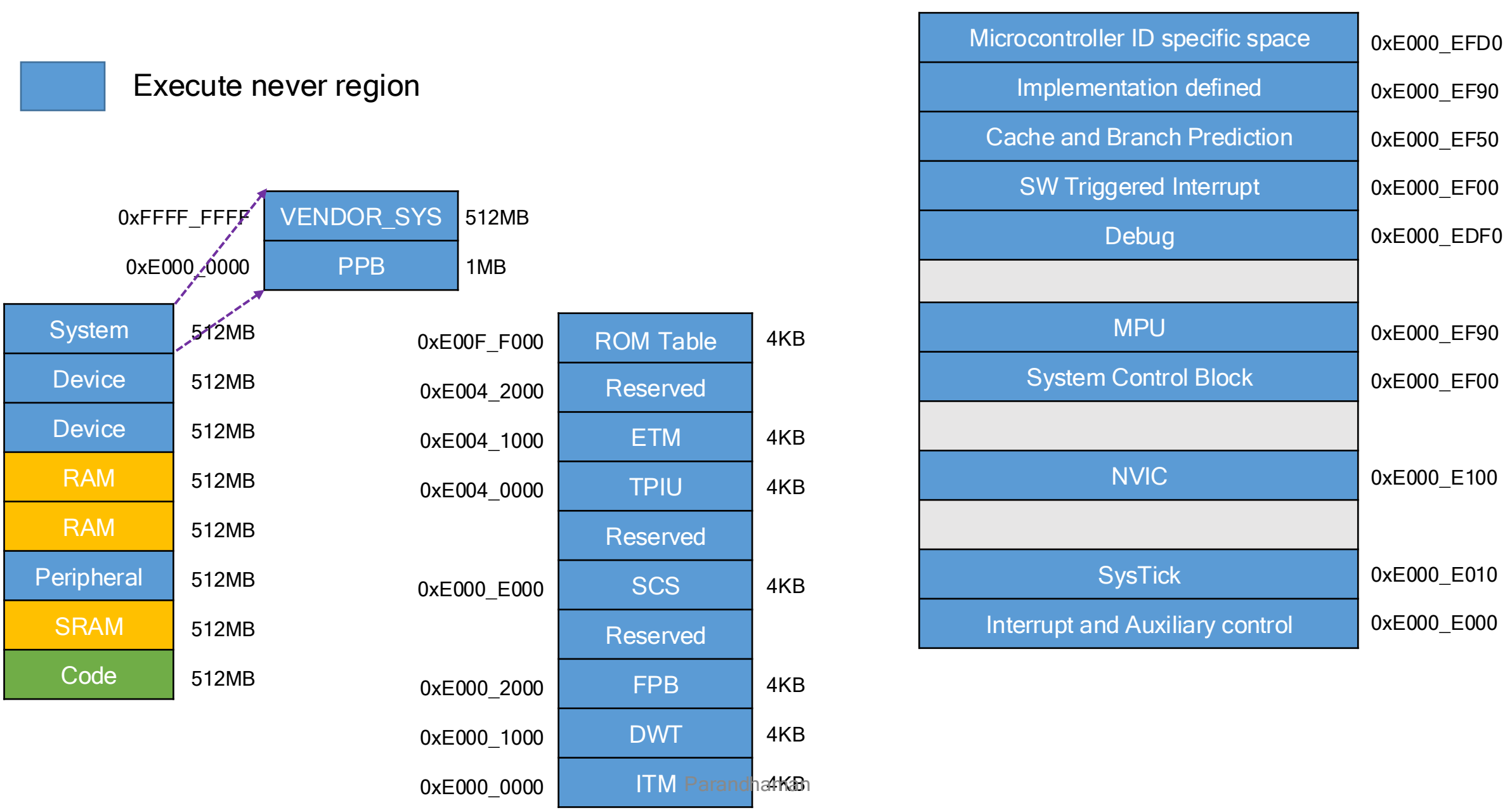


Stack initialization tips

- Evaluate your targeted application. Decide the amount of stack that would be needed for the worst case scenario of your application run time
- Know your processors stack consumption model (FA, FD, EA, ED)
- Decide the stack placement in the RAM (middle, end, external memory)
- In many applications there could be second stage stack init. For example, if you want to allocate stack in external SDRAM then first start with internal RAM, in the main or startup code initialize the SDRAM, then change the stack pointer to point to SDRAM
- If you're using ARM cortex Mx based processor's make sure first location of the vector table contains the initial stack address(MSP). The startup code of the project usually does this
- You may also use the linker script to decide the stack, heap and other RAM area boundaries. Startup script usually fetches boundary details from startup script
- In an RTOS scenario, the kernel code may use MSP to trace its own stack and configure PSP for user's task stack

ARM v7-M Memory Map

Processor memory map



ARM v7-M Exceptions

Exception model

- Exception modes
  - Inactive → Not pending or active
  - Active → Exception processing has been started but not yet completed
  - Pending → Exception event has been generated but processing not started yet
  - Active and Pending → Exception processing has been started but there is a pending exception from same source
- When enabled interrupt asserted
  - Interrupt is taken and serviced by ISR
  - Core runs at handler mode
  - While return from interrupt, original priority level restored
- When enabled interrupt asserted, with equal or lower priority
  - Interrupt is pended to run (will go to active mode when higher priority interrupt was completed)
- When disabled interrupt asserted
  - Interrupt is pended to run (will go to active mode when interrupt gets enabled)
- Interrupt nesting is always enabled
  - To avoid nesting set all the interrupts to the same priority

Exceptions in cortex Mx processors

- Exceptions can be caused by various events
  - Internal - SVC, PendSV, Faults, SysTick
  - External - Reset, Interrupts, Faults
- Cortex Mx processors has total of 255 exceptions/interrupts
  - 15 – System exceptions
  - 240 – external interrupts

Name	Exception Number	Exception priority No.	Vector address	Default status
Reset	1	-3	0x0000_0004	Enabled, can't be masked
Non Maskable interrupt (NMI)	2	-2	0x0000_0008	Enabled, can't be masked
Hard Fault	3	-1	0x0000_000C	Enabled, can be masked
Memory management Fault	4	(0-255) Programmable	0x0000_0010	Disabled
Bus Fault	5	(0-255) Programmable	0x0000_0014	Disabled
Usage Fault	6	(0-255) Programmable	0x0000_0018	Disabled
SVCall	11	(0-255) Programmable	0x0000_002C	Triggers only when inst. executed
Debug monitor			0x0000_0030	Disabled
PendSV	14	(0-255) Programmable	0x0000_0038	Disabled
SysTick	15	(0-255) Programmable	0x0000_003C	Disabled and triggers when SysTick timer enabled and expires
Interrupts 0-245	16 to 16+N	(0-255) Programmable	0x0000_0040	

Pre-empt priority and sub-priority

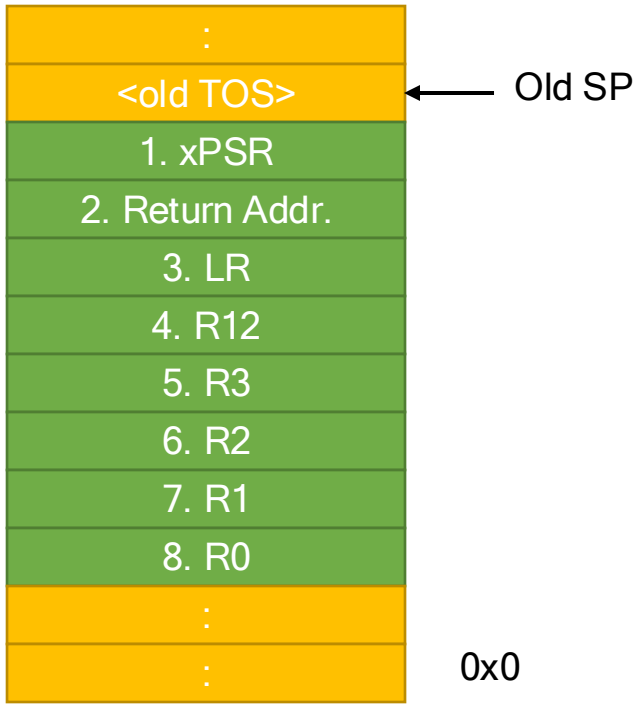
- What if two interrupts of same priority hit the processor at the same time?
  - Then sub-priority value will be checked against them

Priority group	Pre-empt priority field	Sub-priority field
0 (default)	bit[7:1]	bit[0]
1	bit[7:2]	bit[1:0]
2	bit[7:3]	bit[2:0]
3	bit[7:4]	bit[3:0]
4	bit[7:5]	bit[4:0]
5	bit[7:6]	bit[5:0]
6	bit[7:7]	bit[6:0]
7	None	bit[7:0]

- Application Interrupt and Reset Control Register [10:8] (part of SCS) used to select the priority group
- What if two interrupts with same priority group (same pre-empt priority and sub-priority) hit the processor at the same time?
  - Interrupt with lowest IRQ number will be allowed first
- NOTE – We cannot change the priority for Reset, NMI and HardFault handlers

Exception entry

- Complete current instruction on most of the cases
- Set pending bit of that interrupt
- Stacking and vector fetch
  - Stacking → Full descending stack
    - Stack must be 8bytes aligned
    - STKALIGN bit in configuration and control register must be 0
    - 8 registers are pushed → PC, R0-R3, R12, LR, xPSR
    - Follow AAPCS
- LR gets modified for interrupt return
- Set active bit of that interrupt
- Clear pending bit of that interrupt
- Change processor mode to handler mode and force to use MSP
- Start executing ISR



Exception return

- On interrupt entry, the LR stores the EXC_RETURN value
  - Previous LR is saved to stack so it can properly be restored
- EXC_RETURN is used to correctly return from the interrupt

Bits	31:28	27:5	4	3	2	1	0
Description	EXC_RETURN Indicator	Reserved	FPU Frame Type	Return mode	Return Stack	Reserved	CPU State
Value	0xF	0x7F_FFFF	1 (Basic) 0 (Extended)	1 (Thread) 0 (Handler)	0 (Main SP) 1 (Process SP)	0	1 (Reserved)

EXC_RETURN	Condition
0xFFFF_FFE1	Return to handler mode and restore extended stack frame
0xFFFF_FFE9	Return to thread mode using main stack and restore extended stack frame
0xFFFF_FFED	Return to thread mode using process stack and restore extended stack frame
0xFFFF_FFF1	Return to handler mode and restore basic stack frame
0xFFFF_FFF9	Return to thread mode using main stack and restore basic stack frame
0xFFFF_FFFD	Return to thread mode using process stack and restore basic stack frame

Fault exceptions

- Cortex Mx processor has 4 types of fault exceptions

Fault	Status	Priority	
HardFault	Enabled by default, Can be enabled using FAULTMASK register	Non-configurable	
MemManage	Disabled by default	Configurable	
BusFault	Disabled by default	Configurable	
UsageFault	Disabled by default	Configurable	

- Causes of Faults
  - Divide by zero (if enabled)
  - Undefined instruction
  - Attempt to execute code from memory region which is marked as execute never (XN) to prevent code injection
  - MPU guarded memory region access violation by the code
  - Unaligned data access (if enabled)
  - Returning to thread mode keeping active interrupt alive
  - Bus error (ex, no response from memory device)
  - Executing SVC instruction inside SVC handler or calling function in SVC handler which eventually execute hidden SVC instruction
  - Debug monitor settings and related exceptions

Causes of Faults

- Hard Fault is an exception that occurs because of an error during exception processing or because of exception can not be handled by any other exception mechanism
- Hard Fault causes
  - Escalation of configurable fault exceptions
  - Bus error returned during vector fetch
  - Executing breakpoint instruction when both halt mode and debug monitor is disabled
  - Executing SVC instruction inside SVC handler
- MemMange Fault exception causes
  - Memory access violation
    - Can be triggered from processor core itself or by an MPU
- Bus Fault exception causes
  - Error response during access to memory devices
  - Unprivileged access to PPB (Private Peripheral Bus)
  - If bus error happens during vector fetch then its always escalated to hard fault exception
- Usage Fault exception causes
  - Execution of undefined instruction
  - Execution of floating point instructions by keeping floating point unit disabled
  - Trying to switch ARM ISA
  - Trying to return thread mode while exception/interrupt still active
  - Unaligned data access / Divide by zero

ARM v7-M Exceptions handlers

Fault handlers

- Which Fault occurred (Example, usage fault can be caused by multiple reasons, we need to find exact reason)
  - Check Fault status register and print to the user exactly why fault happened
- Where in the code causes this issue?
  - Trace stack frame and print it
  - Fault address registers also gives you the address at which fault happened

SVC handler

PendSV handler

ARM v7-M System Control and ID Registers

System Control and ID Registers

Register	Description
CPUID	• Provides identification information about the processor
Interrupt Control and State Register (ICSR)	• Provides software control of setting/clearing pending on NMI, PendSV, SysTick • Exception number of currently active exception, 0 incase of there is no exception • Exception number of enabled interrupt but currently its pending (if any)
Vector Table Offset Register (VTOR)	• Holds vector table base address, note: [6:0] bits are reserved
Application interrupt and reset control register(AIRCR)	• Setting bit 0 causes local system reset • Clears active state of fixed and configurable interrupts • Setting bit 2 causes System reset request → asserts external signal SYSRESETREQ • Sets priority group value (means tells how many bits which defines priority and how many bit defines sub-priority) • Tells endianess of the processor
System Control Register (SCR)	• Sleep on exit • Deep sleep • Can I wakeup when there is an pending exception (inactive to pending)
Configuration and Control Register (CCR)	• Can I enter thread mode in exception handlers? • Can unprivileged software access Software triggered interrupt register? • Can I enable unaligned access trap? • Can I enable divide by zero trap? • 4 byte or 8byte stack alignment? • Enable/disable → Data cache, Instruction cache, Branch prediction

System Handler Priority Register (SHPR0-3)	• 8bits of priority for each system exception→ 4 exceptions/register = 12 Exceptions
	• Total 16 exceptions
	  • First 4 exceptions are fixed priority
	  • Next 12 exceptions can be prioritized using system handler priority register

System Control and ID Registers – cont.

Register	Description
System Hander Control and State Register (SHCSR)	• Enable/disable of system exceptions • Status (Active/Pending) of system exceptions

Configurable Fault Status Register (CFSR)	• [7:0] → MemMange Fault status
	  • Execute never fetch occurred
	  • Data access violation (memMange fault address register indicates at which address fault happened)
	  • Stack error during exception entry
	  • Stack error during exception exit
	  • Also indicates contents of memMange fault address register is valid or not??
	• [15:8] → BusFault status
	  • Bus error during instruction prefetch
	  • Precise or imprecise fault address
	  • Stack error during exception entry
	  • Stack error during exception exit
	  • Also indicates contents of Bus fault address register is valid or not??
	• [31:16] → Usage fault status
	  • Undefined instruction usage fault
	  • Invalid 'T' bit ot 'IT' bit
	  • Integrity check error on EXC_RETURN
	  • Coprocessor error
	  • Unaligned access error
	  • Divide by zero error

Hard Fault Status Register (HFSR)	• Vector table read fault occurred • Due to forced escalation from configurable priority exception • Debug event occurred

Debug Fault Status Register (DFSR)	• Shows which debug event occurred
	  • Halt request debug event
	  • Atleast one breakpoint debug event
	  • Atleast one debug event generated by DWT
	  • Vector Catch and Assertion of external debug request

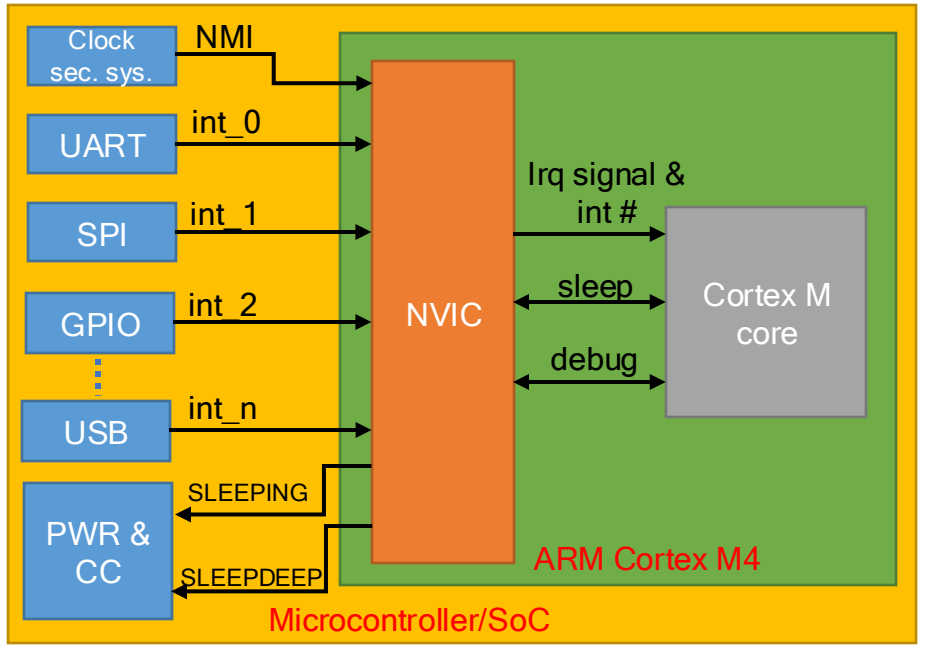
System Control and ID Registers – cont.

Register	Description
MemMange Fault Address Register (MMFAR)	• Shows the address of the memory location that caused an MPU fault
BusFault Address Register (BFAR)	• Shows the address associated with a precise data access fault.
Auxiliary Fault Status Register, AFSR	
Coprocessor Access Control Register, CPACR	

NVIC

NVIC - Nested Vectored Interrupt Controller

- ARM cortex M processor supports 240 external interrupts
- These interrupts are managed and configured by NVIC
  - Enable/Disable/Pend various interrupts and read the status of active and pending interrupts
  - Configure priority and priority grouping of various interrupts
- Its called as ‘nested’ because it supports, pre-empting a lower priority interrupt handler when high priority interrupt arrives
- What are those 240 interrupts?
  - This is highly vendor MCU specific and its triggered by various on-chip peripherals like GPIO, I2C, DMA, CAN, SPI, UART, USB, etc.
  - STM32F407xx MCU delivers 83 different interrupts to the NVIC
  - TIVA MCU delivers 154 different interrupts to the NVIC



NVIC registers

- Cortex Mx processors has total of 255 exceptions/interrupts

Register	Read value	Write value	Description
Interrupt Set Enable Registers (ISER0 - ISER7)	0 → interrupt disabled 1 → interrupt enabled	0 → no effect 1 → enable interrupt	• Used to enable the interrupt
Interrupt Clear Enable Registers (ICER0 - ICER7)	0 → interrupt disabled 1 → interrupt enabled	0 → no effect 1 → disable interrupt	• Used to disable the interrupt
Interrupt Set Pending Registers (ISPR0 - ISPR7)	0 → interrupt is not pending 1 → interrupt is pending	0 → no effect 1 → changes interrupt state to pending	• Used to pend the interrupt
Interrupt Clear Pending Registers (ICPR0 - ICPR7)	0 → interrupt is not pending 1 → interrupt is pending	0 → no effect 1 → removes pending state of the register	• Used to clear pending interrupt
Interrupt Active Bit Registers (IABR0 - IABR7)	0 → interrupt not active 1 → interrupt active	-	• Used to check which interrupt was in currently execution
Interrupt Priority Registers (IPR0 - IPR59)	Priority value of 4 interrupts	Priority value of 4 interrupts	• Used to configure the priority • 8bits used to define the interrupt priority, hence 1 register holds priority of 4 interrupts • 240/4 = 60 registers are required
Software triggered Interrupt Register (STIR)	-	Interrupt number [0:8]	• Used to trigger the interrupt through sw

SysTick

MPU

FPU