EECS 373

Design of Microprocessor-Based Systems

Thomas Schmid

University of Michigan

Lecture 7: Interrupts, ARM NVIC

September 28, 2010

"PLEASE FEEL FREE TO INTERRUPT IF YOU HAVE A QUESTION."

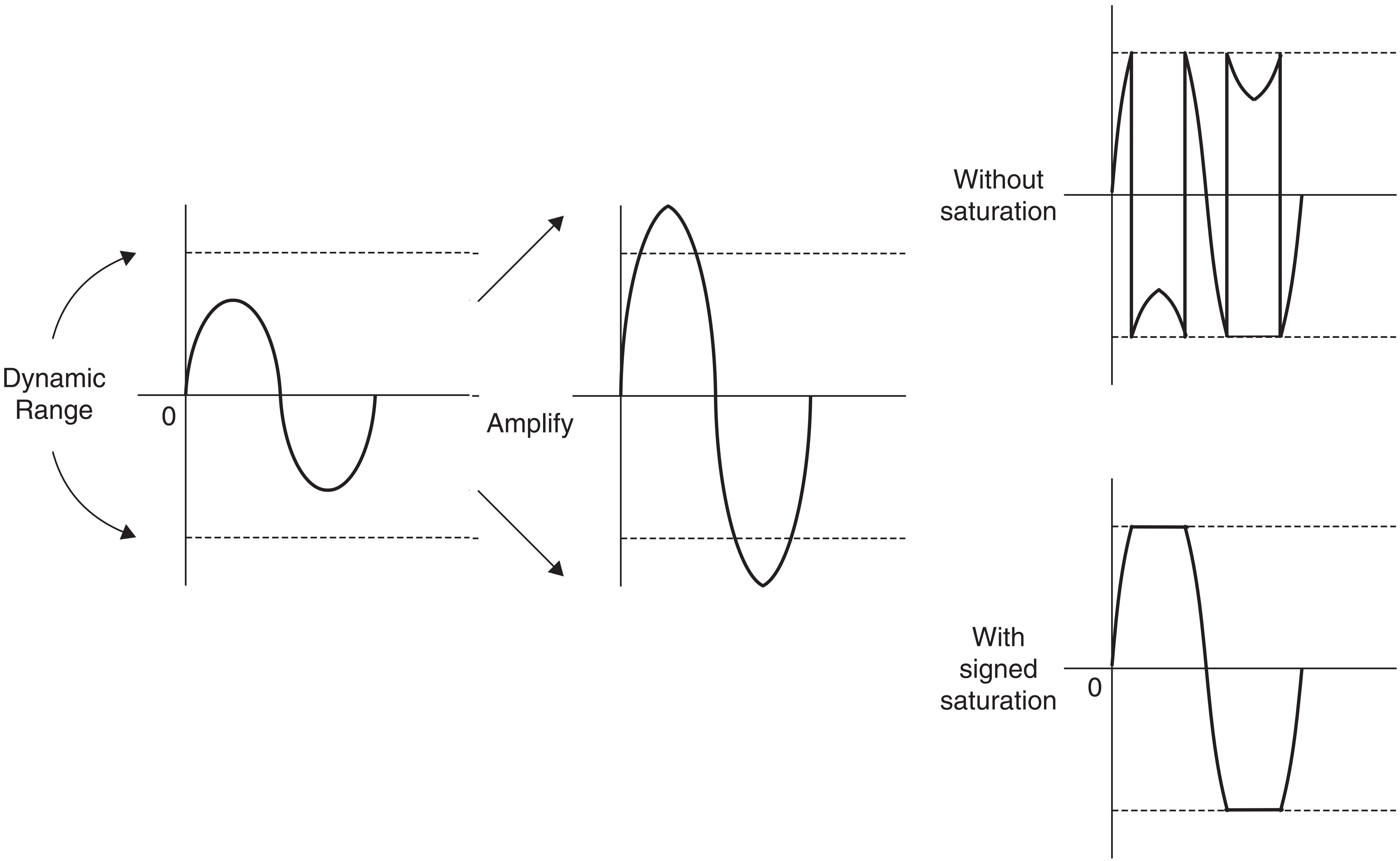
http://home.netcom.com/~swansont/science.html

Minute Quiz...

Recap of the last lecture

- Why is Reset Vector +1?
  - It's an ARM specific thing. The least significant bit in jump instructions indicates the type of instruction at that location (0: for ARM, 1: for Thumb). Since the Cortex-M3 can only execute Thumb2, this will always

The SAT instruction



From: The Definitive Guide to the ARM Cortex-M3

Saturating at 32-bit signed value to a 16-bit

SSAT.W <Rd>, #<immed>, <Rn>, {,<shift>}

```
SSAT.W R1, #16, R0
```

Input (R0)	Output (R1)	Q Bit
0x00020000	0x00007FFF	Set
0x00008000	0x00007FFF	Set
0x00007FFF	0x00007FFF	Unchanged
0x00000000	0x00000000	Unchanged
0xFFFF8000	0xFFFF8000	Unchanged
0xFFFF8001	0xFFFF8000	Set
0xFFFE0000	0xFFFF8000	Set

From: The Definitive Guide to the ARM Cortex-M3

Interrupts

Generalization of Interrupts

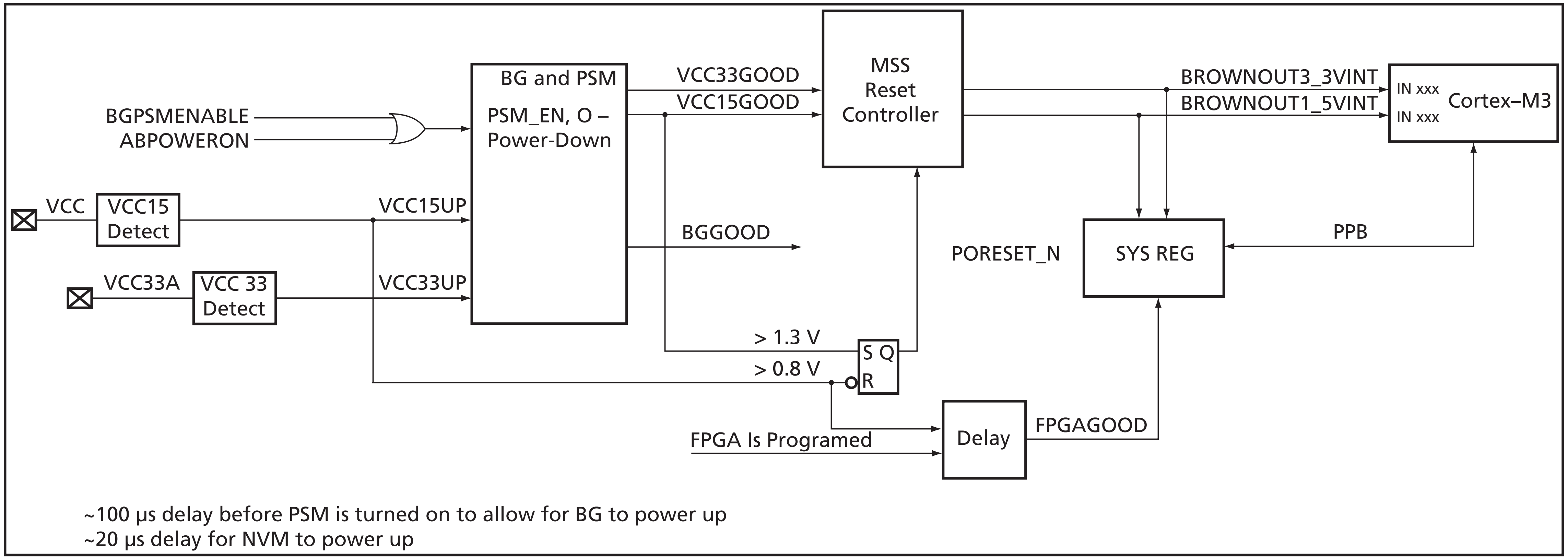
- Merriam-Webster:
  "*to break the uniformity or continuity of*"

- Informs a program of some external events
- Breaks execution flow

- Where do interrupts come from?
- How do we save state for later continuation?
- How can we ignore interrupts?
- How can we prioritize interrupts?
- How can we share interrupts?

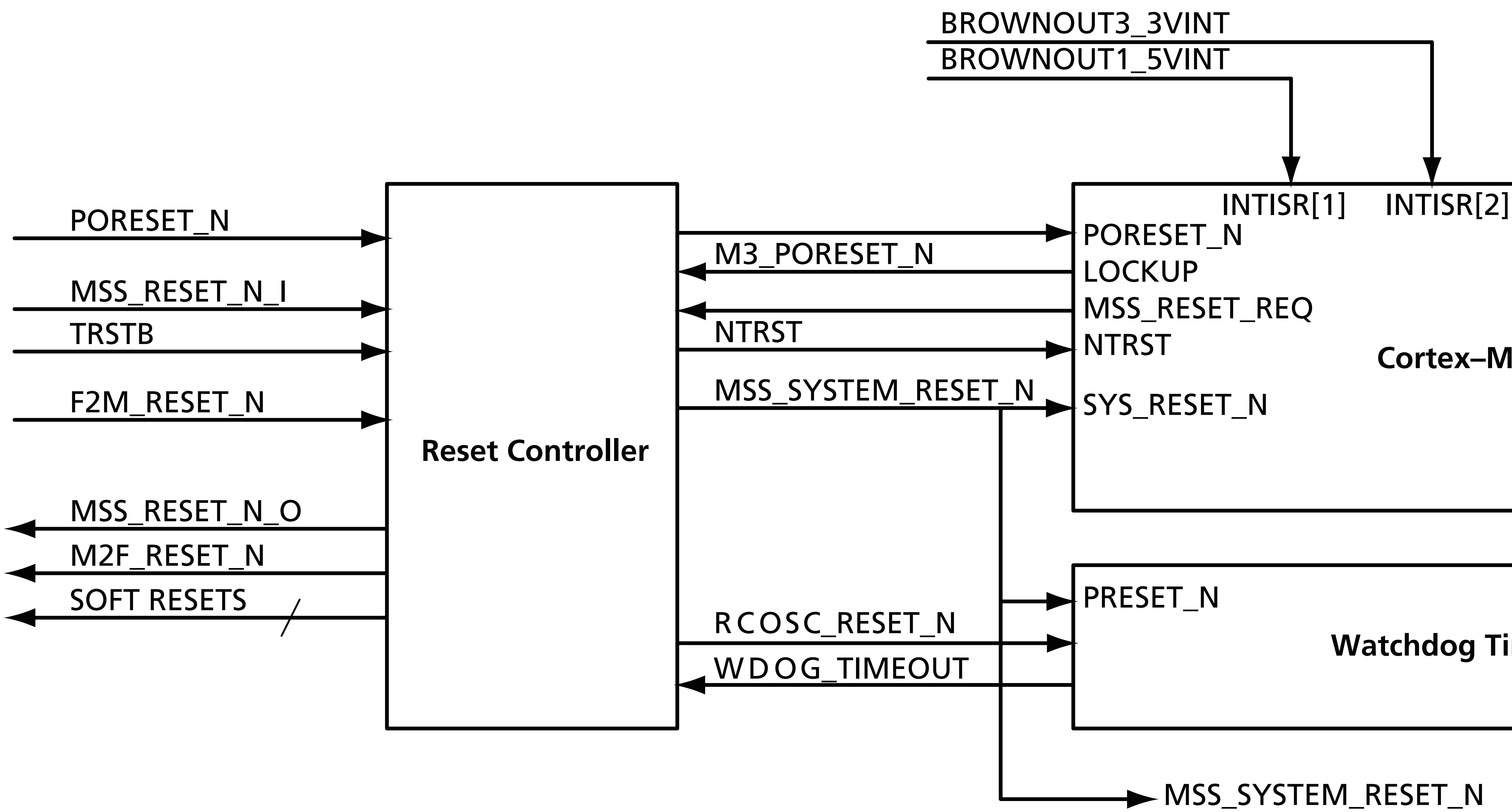
How does an embedded system boot?

The Reset Interrupt



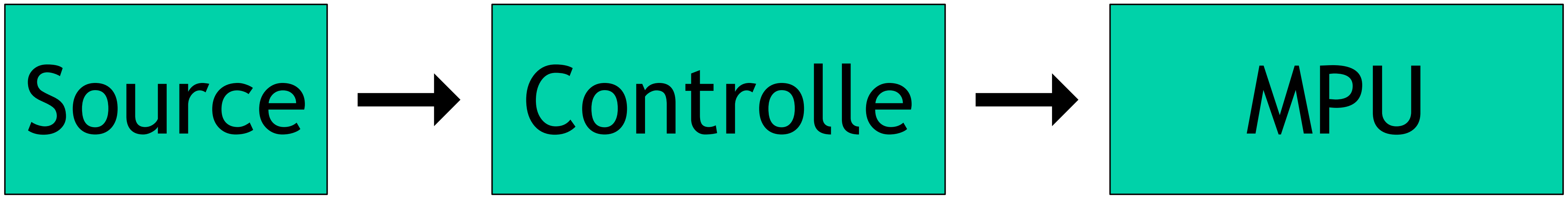
1. No power
2. System is held in RESET as long as VCC15 < 0.8V
   a. In reset: registers forced to default
   b. RC-Osc begins to oscillate
   c. MSS_CCC drives RC-Osc/4 into FSCK
   d. PORESET_N is held low
3. Once VCC15GOOD, PORESET_N goes high
   a. MSS reads from eNVM address 0x0 and 0x4

The Reset Interrupt (2)



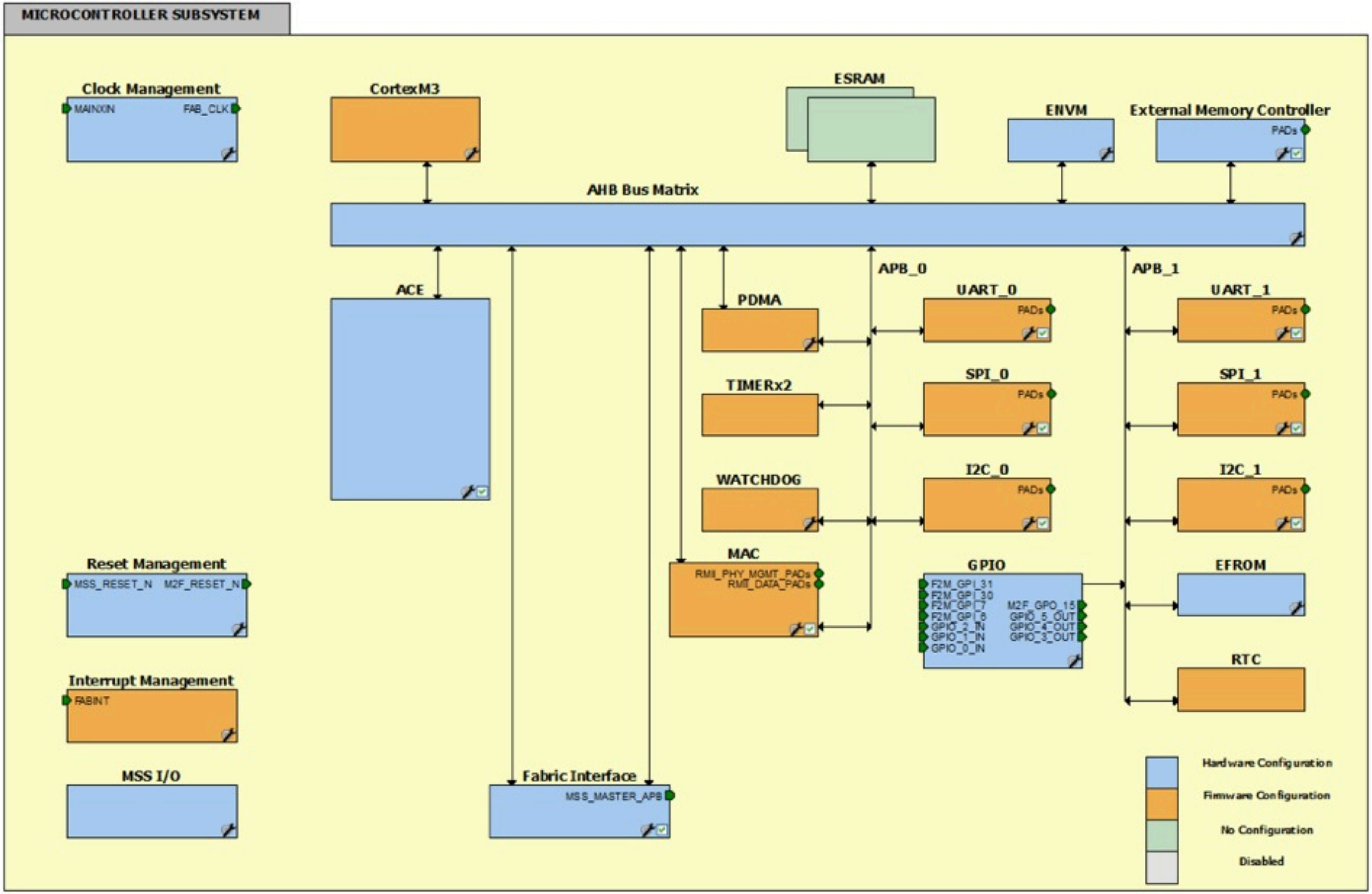
- The Reset Interrupt is Non-Maskable!

Interrupt Handling



- On the Cortex-M3
  - Source: Software, Peripheral
  - Controller: Nested Vectored Interrupt Controller (NVIC)
  - MPU: Cortex-M3 Core

Sources of Interrupts



Types of Interrupts

- Physical interrupts
  - Level-triggered
  - Edge-triggered (positive, negative)
  - Hybrid
    - Look for edges, but signal must stay for a while
    - Often used for non-maskable interrupts to avoid glitches
- Non-maskable interrupts
- Interrupt priorities
- Software interrupts

The Nested Vectored Interrupt Controller (NVIC) on the Cortex-M3

- Control registers are memory mapped
- Contains control logic for interrupt processing
- Also contains MPU, SYSTICK Timer, and Debug

- 15 internal interrupts (defined by ARM)
- Supports up to 240 external interrupts (vendor specific)
- Accessed at 0xE000E000 on any Cortex-M3!

- Register definitions can be found at:
  - ARM Cortex-M3 Technical Reference Manual v2.1, Chapter 6
  - The Definitive Guide to the ARM Cortex-M3

System Exceptions
NVIC Interrupts 1-15

Exception Number	Exception Type	Priority	Description
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt (external NMI input)
3	Hard Fault	−1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage Fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus Fault	Programmable	Bus error; occurs when AHB interface receives an error response from a bus slave (also called *prefetch abort* if it is an instruction fetch or *data abort* if it is a data access)
6	Usage Fault	Programmable	Exceptions due to program error or trying to access coprocessor (the Cortex-M3 does not support a coprocessor)
7–10	Reserved	NA	–
11	SVCall	Programmable	System Service call
12	Debug Monitor	Programmable	Debug monitor (breakpoints, watchpoints, or external debug requests)
13	Reserved	NA	–
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System Tick Timer

From: The Definitive Guide to the ARM Cortex-M3

Actel SmartFusion Interrupts

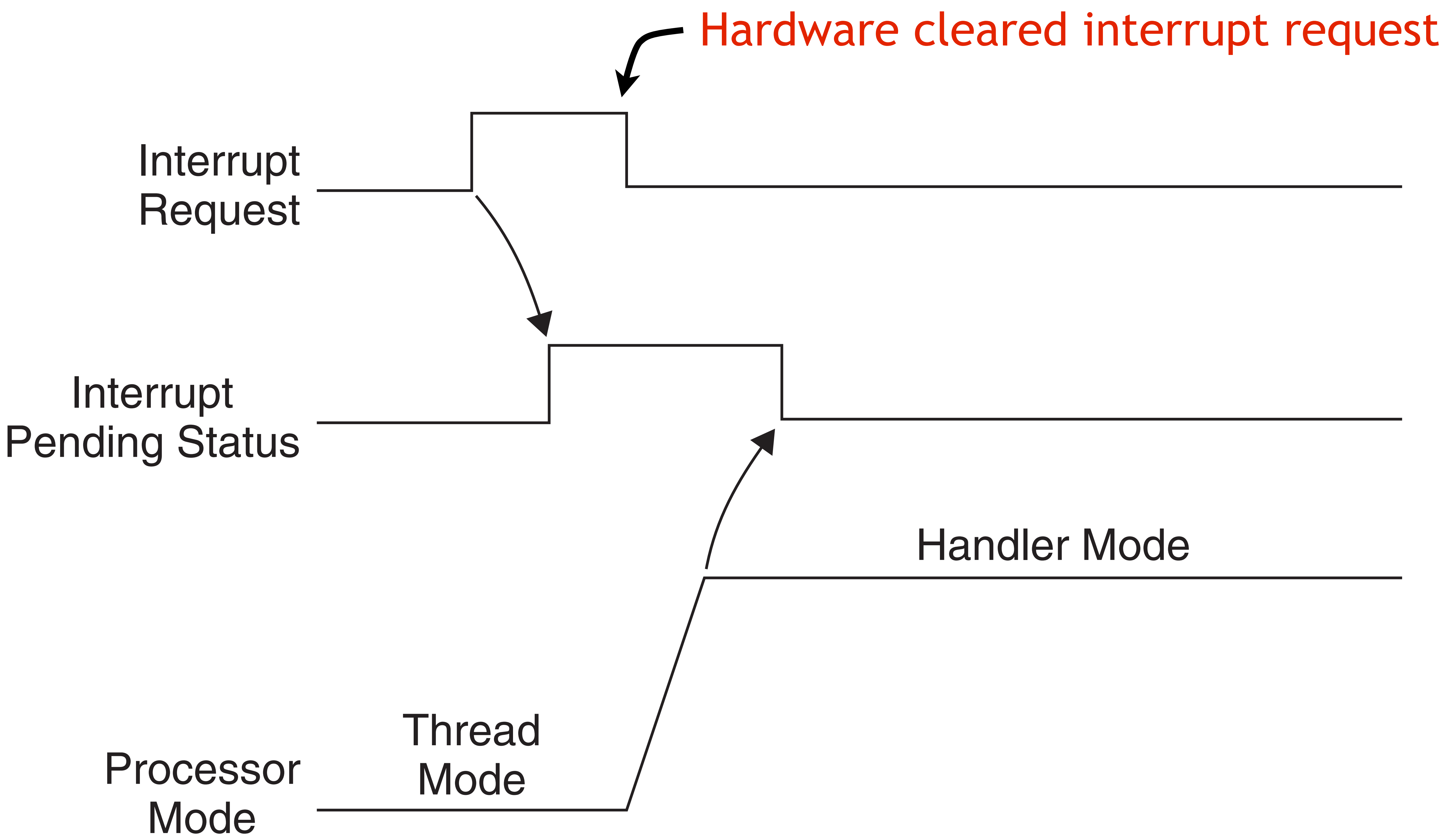
Table 1-5 • **SmartFusion Interrupt Sources**

Cortex-M3 NVIC Input	IRQ Label	IRQ Source
NMI	WDOGTIMEOUT_IRQ	WATCHDOG
INTISR[0]	WDOGWAKEUP_IRQ	WATCHDOG
INTISR[1]	BROWNOUT1_5V_IRQ	VR/PSM
INTISR[2]	BROWNOUT3_3V_IRQ	VR/PSM
INTISR[3]	RTCMATCHEVENT_IRQ	RTC
INTISR[4]	PU_N_IRQ	RTC
INTISR[5]	EMAC_IRQ	Ethernet MAC
INTISR[6]	M3_IAP_IRQ	IAP
INTISR[7]	ENVM_0_IRQ	ENVM Controller
INTISR[8]	ENVM_1_IRQ	ENVM Controller
INTISR[9]	DMA_IRQ	Peripheral DMA
INTISR[10]	UART_0_IRQ	UART_0
INTISR[11]	UART_1_IRQ	UART_1
INTISR[12]	SPI_0_IRQ	SPI_0
INTISR[13]	SPI_1_IRQ	SPI_1
INTISR[14]	I2C_0_IRQ	I2C_0
INTISR[15]	I2C_0_SMBALERT_IRQ	I2C_0
INTISR[16]	I2C_0_SMBSUS_IRQ	I2C_0
INTISR[17]	I2C_1_IRQ	I2C_1
INTISR[18]	I2C_1_SMBALERT_IRQ	I2C_1
INTISR[19]	I2C_1_SMBSUS_IRQ	I2C_1
INTISR[20]	TIMER_1_IRQ	TIMER
INTISR[21]	TIMER_2_IRQ	TIMER
INTISR[22]	PLLLOCK_IRQ	MSS_CCC
INTISR[23]	PLLLOCKLOST_IRQ	MSS_CCC
INTISR[24]	ABM_ERROR_IRQ	AHB BUS MATRIX
INTISR[25]	Reserved	Reserved
INTISR[26]	Reserved	Reserved
INTISR[27]	Reserved	Reserved
INTISR[28]	Reserved	Reserved
INTISR[29]	Reserved	Reserved
INTISR[30]	Reserved	Reserved
INTISR[31]	FAB_IRQ	FABRIC INTERFACE
INTISR[32]	GPIO_0_IRQ	GPIO
INTISR[33]	GPIO_1_IRQ	GPIO
INTISR[34]	GPIO_2_IRQ	GPIO
INTISR[35]	GPIO_3_IRQ	GPIO
INTISR[64]	ACE_PC0_FLAG0_IRQ	ACE
INTISR[65]	ACE_PC0_FLAG1_IRQ	ACE
INTISR[66]	ACE_PC0_FLAG2_IRQ	ACE
INTISR[67]	ACE_PC0_FLAG3_IRQ	ACE
INTISR[68]	ACE_PC1_FLAG0_IRQ	ACE
INTISR[69]	ACE_PC1_FLAG1_IRQ	ACE
INTISR[70]	ACE_PC1_FLAG2_IRQ	ACE
INTISR[71]	ACE_PC1_FLAG3_IRQ	ACE
INTISR[72]	ACE_PC2_FLAG0_IRQ	ACE
INTISR[73]	ACE_PC2_FLAG1_IRQ	ACE
INTISR[74]	ACE_PC2_FLAG2_IRQ	ACE
INTISR[75]	ACE_PC2_FLAG3_IRQ	ACE
INTISR[76]	ACE_ADC0_DATAVALID_IRQ	ACE
INTISR[77]	ACE_ADC1_DATAVALID_IRQ	ACE
INTISR[78]	ACE_ADC2_DATAVALID_IRQ	ACE
INTISR[79]	ACE_ADC0_CALDONE_IRQ	ACE
INTISR[80]	ACE_ADC1_CALDONE_IRQ	ACE
INTISR[81]	ACE_ADC2_CALDONE_IRQ	ACE
INTISR[82]	ACE_ADC0_CALSTART_IRQ	ACE
INTISR[83]	ACE_ADC1_CALSTART_IRQ	ACE
INTISR[84]	ACE_ADC2_CALSTART_IRQ	ACE
INTISR[85]	ACE_COMP0_FALL_IRQ	ACE
INTISR[86]	ACE_COMP1_FALL_IRQ	ACE
INTISR[87]	ACE_COMP2_FALL_IRQ	ACE
INTISR[88]	ACE_COMP3_FALL_IRQ	ACE
INTISR[89]	ACE_COMP4_FALL_IRQ	ACE
INTISR[90]	ACE_COMP5_FALL_IRQ	ACE
INTISR[91]	ACE_COMP6_FALL_IRQ	ACE
INTISR[92]	ACE_COMP7_FALL_IRQ	ACE
INTISR[93]	ACE_COMP8_FALL_IRQ	ACE
INTISR[94]	ACE_COMP9_FALL_IRQ	ACE
INTISR[95]	ACE_COMP10_FALL_IRQ	ACE

54 more ACE specific interrupts

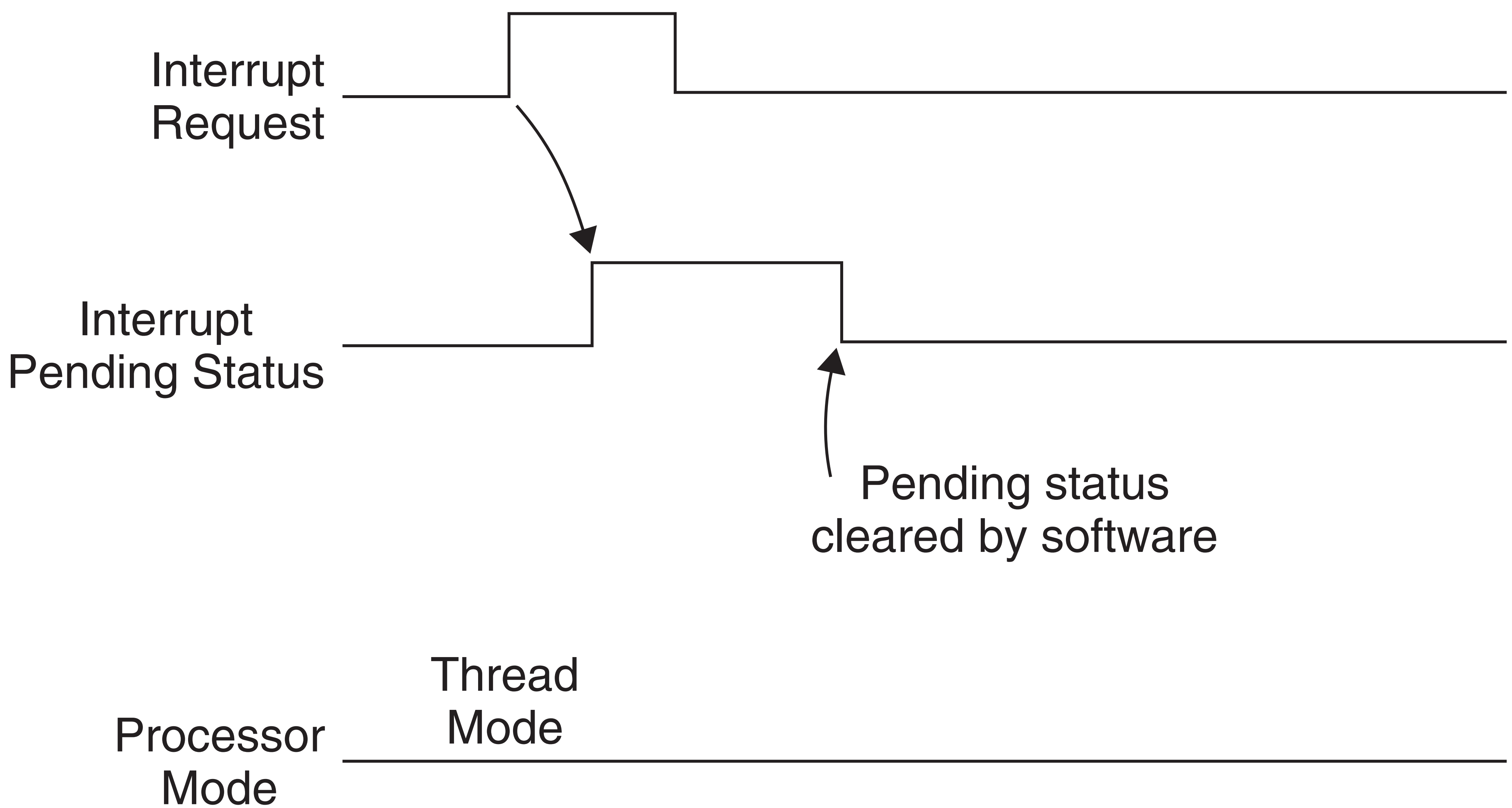
GPIO_3_IRQ to GPIO_31_IRQ cut

Pending Interrupts



From: The Definitive Guide to the ARM Cortex-M3

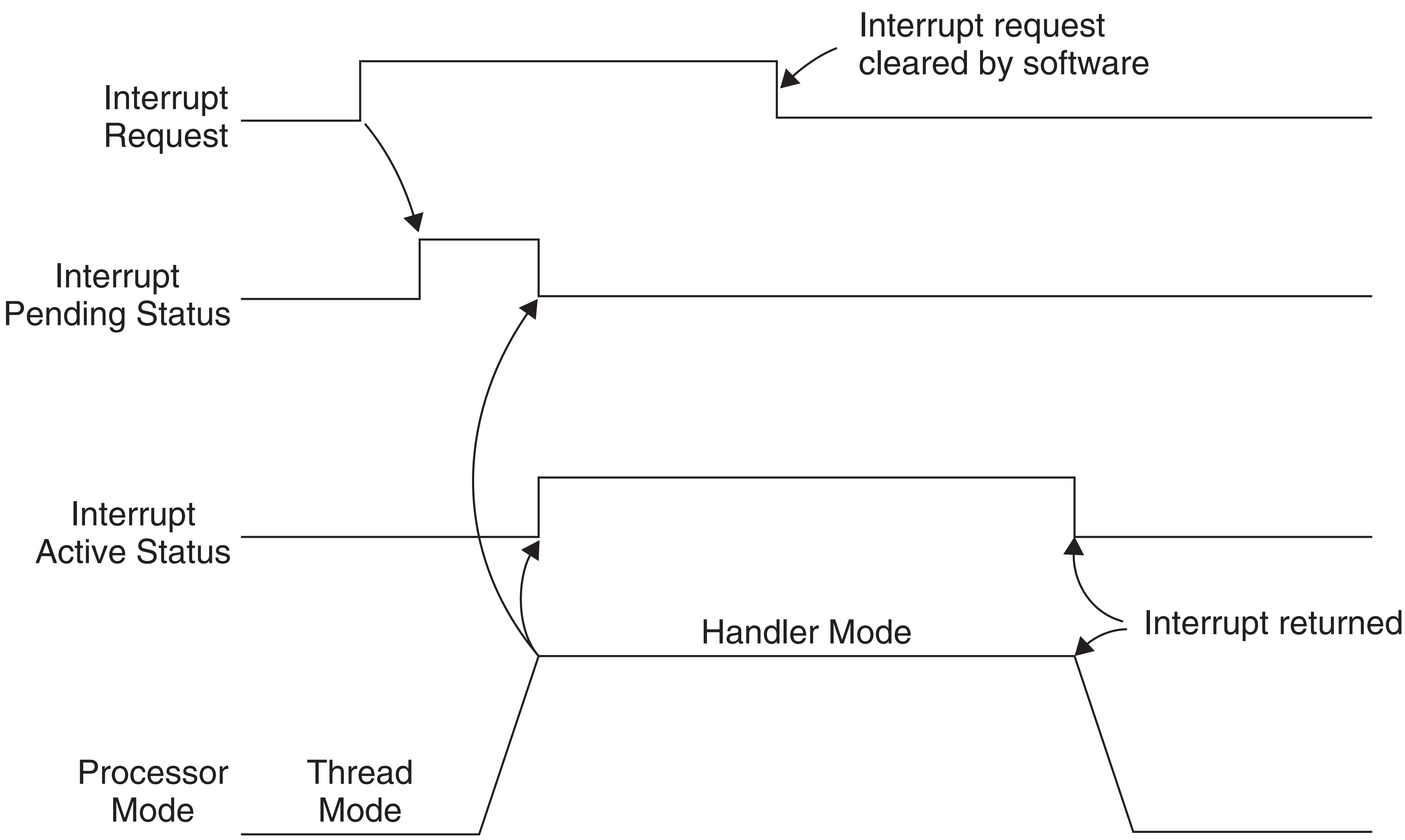
Pending Interrupts (2)



- Software clears pending status while PRIMASK/ FAULTMASK is 1

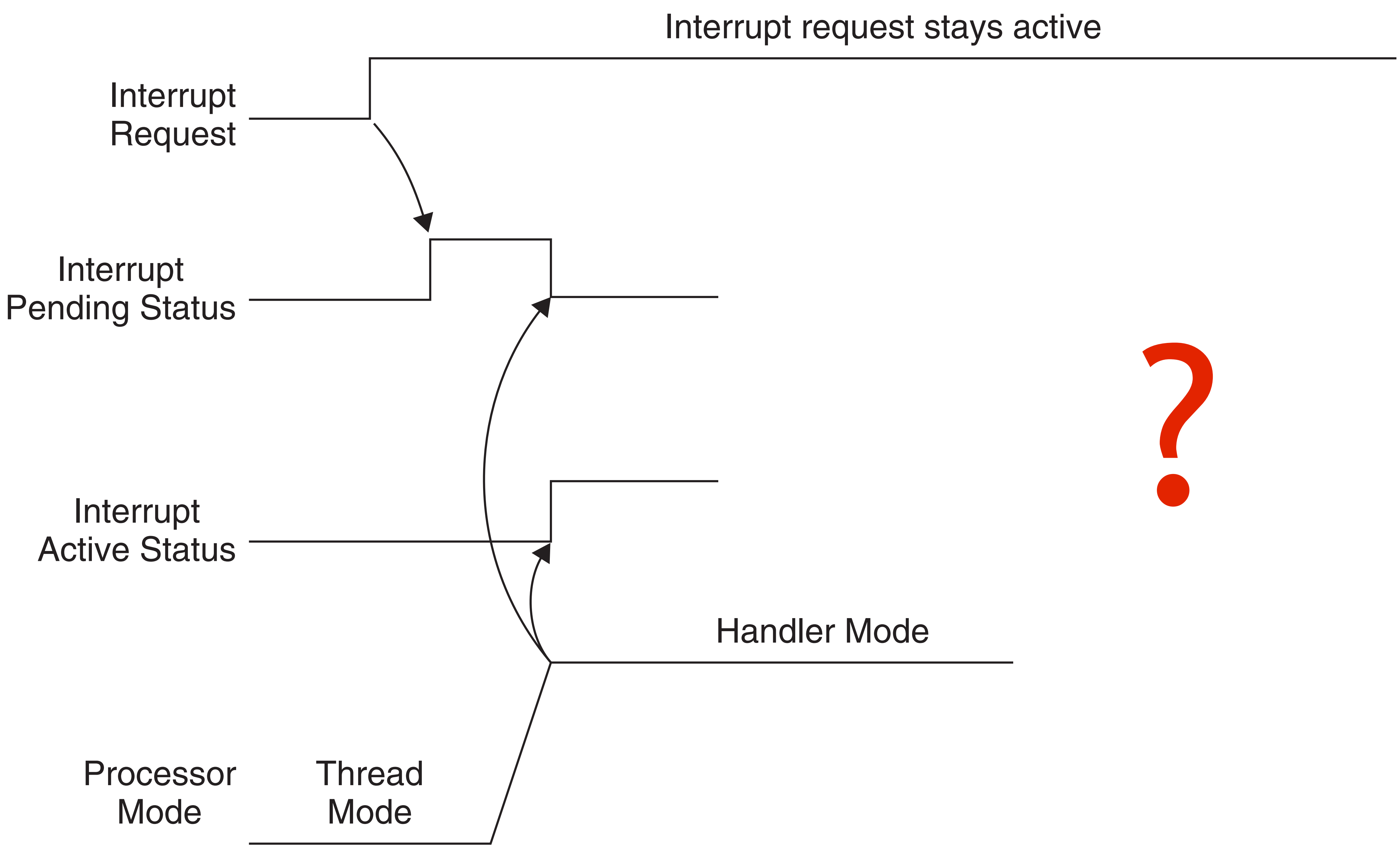
From: The Definitive Guide to the ARM Cortex-M3

Active Status set during handler execution



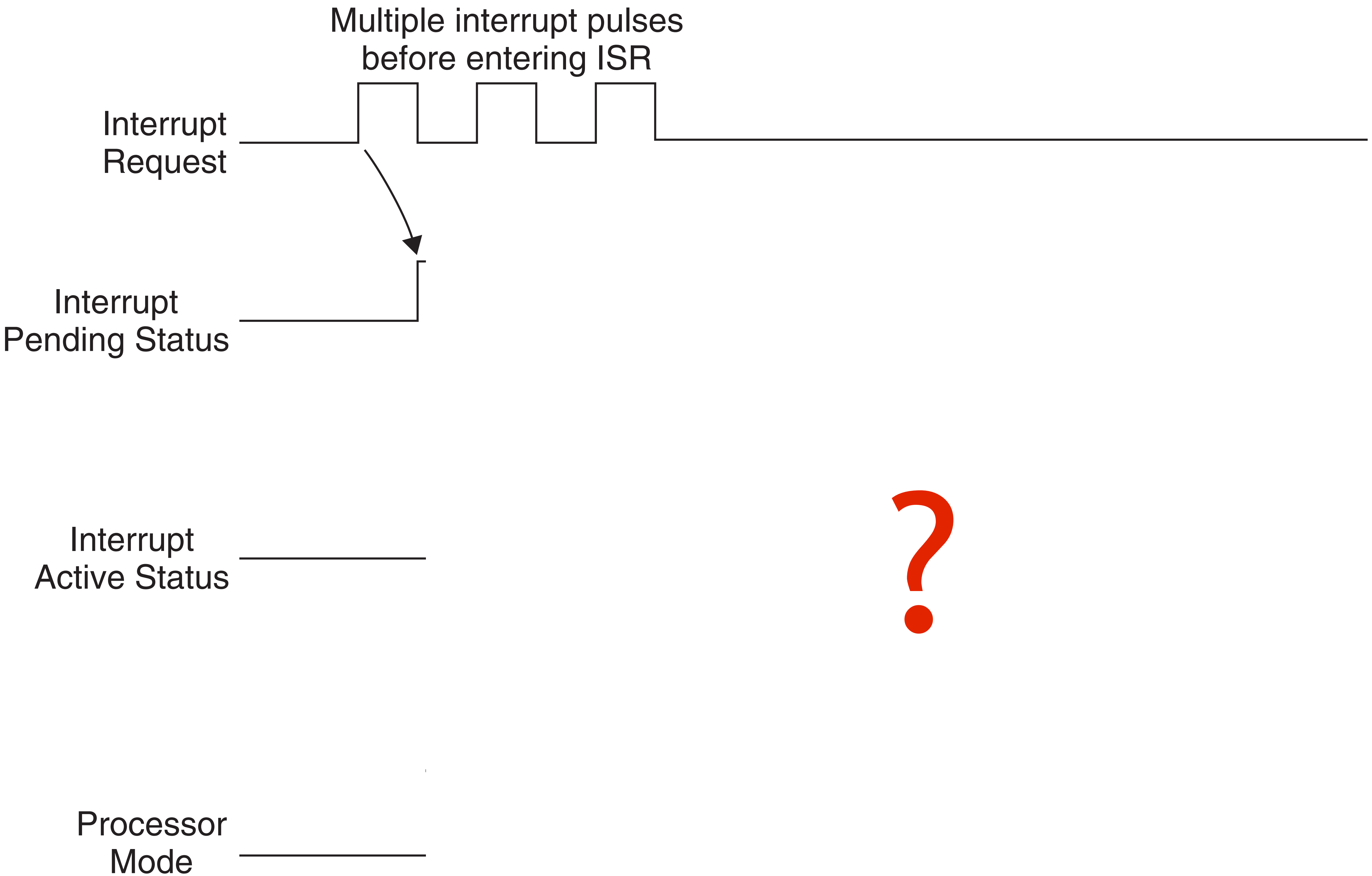
From: The Definitive Guide to the ARM Cortex-M3

Interrupt Request not Cleared



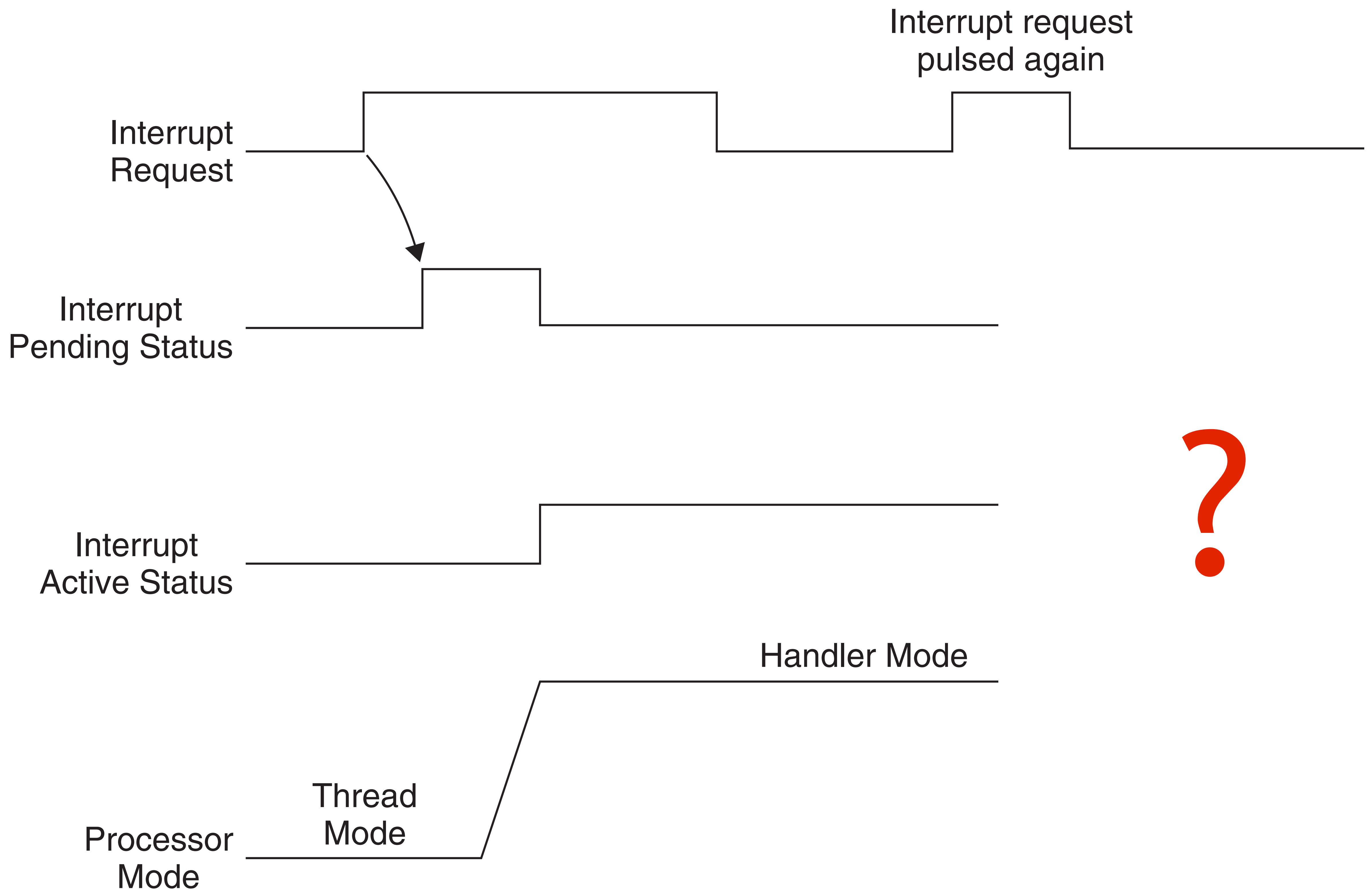
From: The Definitive Guide to the ARM Cortex-M3

Multiple Interrupt Pulses



From: The Definitive Guide to the ARM Cortex-M3

New Interrupt Request after Pending Cleared



From: The Definitive Guide to the ARM Cortex-M3

Configuring the NVIC

- Interrupt Set Enable and Clear Enable
  - 0xE000E100-0xE000E11C, 0xE000E180-0xE000E19C

0xE000E100	SETENA0	R/W	0	Enable for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status

0xE000E180	CLRENA0	R/W	0	Clear enable for external interrupt #0–31 bit[0] for interrupt #0 bit[1] for interrupt #1 … bit[31] for interrupt #31 Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current enable status

Configuring the NVIC (2)

- Set Pending & Clear Pending
  - 0xE000E200-0xE000E21C, 0xE000E280-0xE000E29C

0xE000E200	SETPEND0	R/W	0	Pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to set bit to 1; write 0 has no effect Read value indicates the current status
0xE000E280	CLRPEND0	R/W	0	Clear pending for external interrupt #0–31 bit[0] for interrupt #0 (exception #16) bit[1] for interrupt #1 (exception #17) … bit[31] for interrupt #31 (exception #47) Write 1 to clear bit to 0; write 0 has no effect Read value indicates the current pending status

Configuring the NVIC (3)

- Interrupt Active Status Register
  - 0xE000E300-0xE000E31C

Address	Name	Type	Reset Value	Description
0xE000E300	ACTIVE0	R	0	Active status for external interrupt #0–31 bit[0] for interrupt #0 bit[1] for interrupt #1 ... bit[31] for interrupt #31
0xE000E304	ACTIVE1	R	0	Active status for external interrupt #32–63
...	–	–	–	–

Interrupt Priority

- What do we do if several interrupts arrive at the same time?
- NVIC allows to set priorities for (almost) every interrupt
- 3 fixed highest priorities, up to 256 programmable priorities
  - 128 preemption levels
  - Not all priorities have to be implemented by a vendor!

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented			Not implemented, read as zero				

  - SmartFusion has 32 priority levels, i.e., 0x00, 0x08, …, 0xF8
- Higher priority interrupts can pre-empt lower priorities
- Priority can be sub-divided into priority groups
  - splits priority register into two halves, *preempt priority* and *subpriority*
  - preempt priority: indicates if an interrupt can preempt another
  - subpriority: used if two interrupts of same group arrive concurrently

Interrupt Priority (2)

- Interrupt Priority Level Registers
  - 0xE000E400-0xE000E4EF

Address	Name	Type	Reset Value	Description
0xE000E400	PRI_0	R/W	0 (8-bit)	Priority-level external interrupt #0
0xE000E401	PRI_1	R/W	0 (8-bit)	Priority-level external interrupt #1
...	–	–	–	–
0xE000E41F	PRI_31	R/W	0 (8-bit)	Priority-level external interrupt #31
...	–	–	–	–

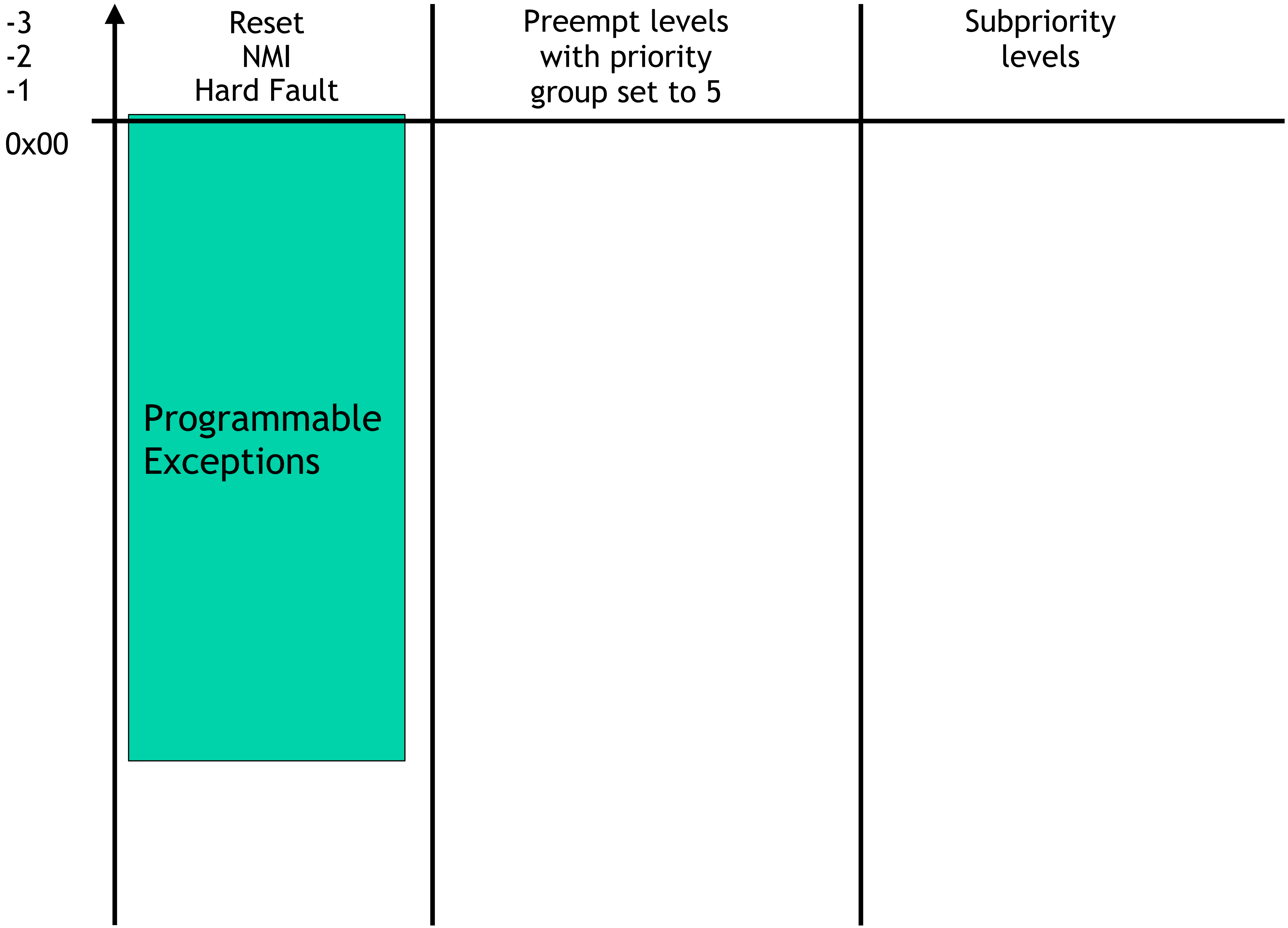
Preemption Priority and Subpriority

Priority Group	Preempt Priority Field	Subpriority Field
0	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]	Bit [6:0]
7	None	Bit [7:0]

Application Interrupt and Reset Control Register (Address 0xE000ED0C)

Bits	Name	Type	Reset Value	Description
31:16	VECTKEY	R/W	–	Access key; 0x05FA must be written to this field to write to this register, otherwise the write will be ignored; the read-back value of the upper half word is 0xFA05
15	ENDIANNESS	R	–	Indicates endianness for data: 1 for big endian (BE8) and 0 for little endian; this can only change after a reset
10:8	PRIGROUP	R/W	0	Priority group
2	SYSRESETREQ	W	–	Requests chip control logic to generate a reset
1	VECTCLRACTIVE	W	–	Clears all active state information for exceptions; typically used in debug or OS to allow system to recover from system error (Reset is safer)
0	VECTRESET	W	–	Resets the Cortex-M3 processor (except debug logic), but this will not reset circuits outside the processor

Exercise: How many preemption priorities and subpriority levels do we get on the Smart Fusion if we set Priority Group to 5?



PRIMASK, FAULTMASK, and BASEPRI

- What if we quickly want to disable all interrupts?

- Write 1 into PRIMASK to disable all interrupt except NMI
  - MOV R0, #1
  - MSR PRIMASK, R0
- Write 0 into PRIMASK to enable all interrupts
- FAULTMASK is the same as PRIMASK, but also blocks hard fault (priority -1)

- What if we want to disable all interrupts below a certain priority?
- Write priority into BASEPRI
  - MOV R0, #0x60
  - MSR BASEPRI, R0

What exactly is an interrupt handler?

Vector Table

- Upon an interrupt, the Cortex-M3 needs to know the address of the interrupt handler (function pointer)
- After powerup, vector table is located at 0x00000000

Address	Exception Number	Value (Word Size)
0x00000000	–	MSP initial value
0x00000004	1	Reset vector (program counter initial value)
0x00000008	2	NMI handler starting address
0x0000000C	3	Hard fault handler starting address
…	…	Other handler starting address

- Can be relocated to change interrupt handlers at runtime (vector table offset register)

Vector Table in SoftConsole

- Located in startup_a2fxxxm3.s

```
g_pfnVectors:
    .word   _estack
    .word   Reset_Handler
    .word   NMI_Handler
    .word   HardFault_Handler
    .word   MemManage_Handler
    .word   BusFault_Handler
    .word   UsageFault_Handler
    .word   0
    .word   0
```

```
/*==============================================
 * Vector table
 */
    .global     g_pfnVectors
    .section    .isr_vector,"a",%progbits
    .type       g_pfnVectors, %object
    .size       g_pfnVectors, .-g_pfnVectors

```

- Put at 0x00000000 in linker script

```
SECTIONS
{
  .text :
  {
    CREATE_OBJECT_SYMBOLS
    __text_load = LOADADDR(.text);
    __text_start = .;
    *(.isr_vector)
```

Interrupt Handlers

```
/*==========================================
 * Reset_Handler
 */
    .global Reset_Handler
    .type   Reset_Handler, %function
Reset_Handler:
_start:
```

```
/*=============================================
 * NMI_Handler
 */
    .weak   NMI_Handler
    .type   NMI_Handler, %function
NMI_Handler:
    B .

/*=============================================
 * HardFault_Handler
 */
    .weak   HardFault_Handler
    .type   HardFault_Handler, %function
HardFault_Handler:
    B .

```

Interrupt Handler in GNU C

- We can overwrite the predefined interrupt handlers

```
__attribute__((__interrupt__)) void Timer1_IRQHandler()
{
    MSS_TIM1_disable_irq();
    MSS_TIM1_clear_irq();
    …
    NVIC_ClearPendingIRQ( Timer1_IRQn );
}


int main()
{
    MSS_TIM1_enable_irq();
    NVIC_EnableIRQ( Timer1_IRQn );
    …
    while(1){}
}
```

Interrupt Service Routines

1. Automatic saving of registers upon exception
   - PC, PSR, R0-R3, R12, LR pushed on the stack
2. While bus busy, fetch exception vector
3. Update SP to new location
4. Update IPSR (low part of PSR) with new exception number
5. Set PC to vector handler
6. Update LR to special value EXC_RETURN

- Several other NVIC registers get updated
- Latency: as short as 12 cycles

Return from ISR

- 3 ways to return from an ISR

Return Instruction	Description
BX <*reg*>	If the EXC_RETURN value is still in LR, we can use the *BX LR* instruction to perform the interrupt return.
POP {PC}, or POP {...., PC}	Very often the value of LR is pushed to the stack after entering the exception handler. We can use the POP instruction, either a single POP or multiple POPs, to put the EXC_RETURN value to the program counter. This will cause the processor to perform the interrupt return.
LDR, or LDM	It is possible to produce an interrupt return using the LDR instruction with PC as the destination register.

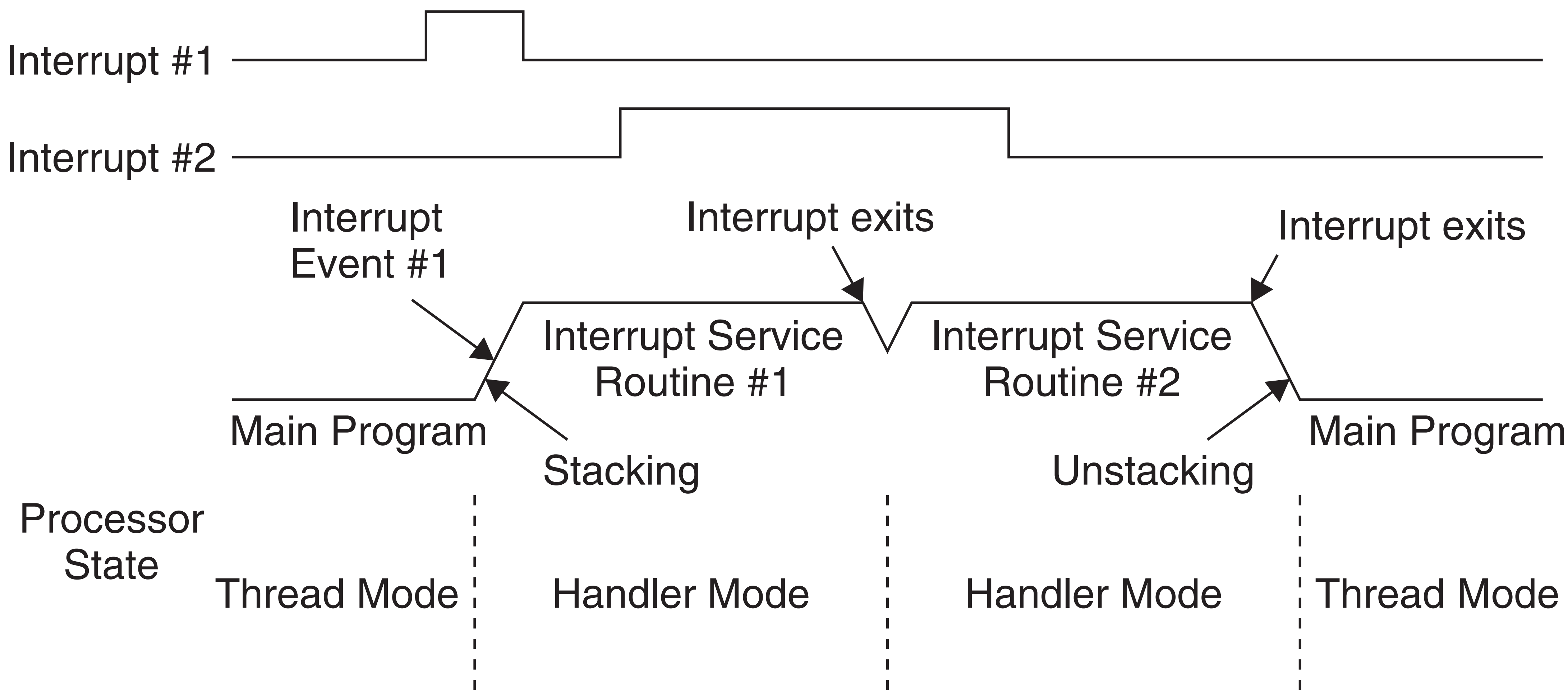
- Unstack and reset SP
- Update NVIC registers

Nested Interrupts

- Built into the Cortex-M3 (not every MCU has this)
- Make sure main stack is large enough!

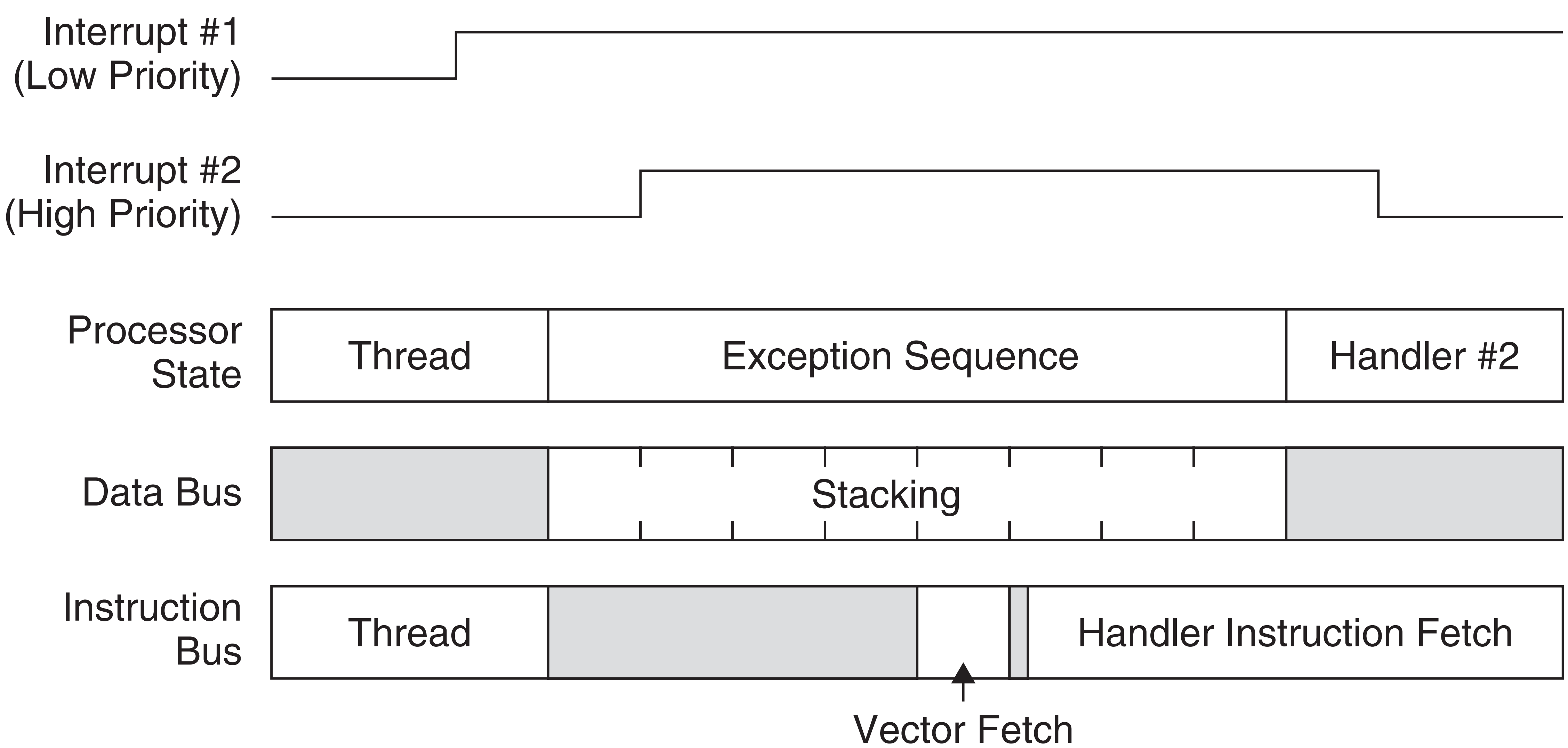
- Two methods:
  - Tail Chaining
  - Late Arrival (preemption)

Tail Chaining



- If first interrupt has same or higher priority
- Skip stacking/unstacking for efficiency

Late Arrival (Preemption)



- Main stack must be able to hold maximum number of preemptions!

Different Concepts of Interrupt Sharing

- Number of potential interrupts usually larger than interrupt lines availability on Core
- One peripheral often only has one interrupt
- Different types of events are stored in a status register

- Example, UART
  - IIR, 0x40000008

3:0	Interrupt identification bits	R	0b0001	0b0110 = Highest priority. Receiver line status interrupt due to overrun error, parity error, framing error or break interrupt. Reading the Line Status Register resets this interrupt. 0b0100 = Second priority. Receive data available interrupt modem status interrupt. Reading the Receiver Buffer Register (RBR) or the FIFO drops below the trigger level resets this interrupt. 0b1100 = Second priority. Character timeout indication interrupt occurs when no characters have been read from the RX FIFO during the last four character times and there was at least one character in it during this time. Reading the Receive Buffer Register (RBR) resets this interrupt. 0b0010 = Third priority. Transmit Holding Register Empty interrupt. Reading the IIR or writing to the Transmit Holding Register (THR) resets the interrupt. 0b0000 = Fourth priority. Modem status interrupt due to Clear to Send, Data Set Ready, Ring Indicator, or Data Carrier Detect being asserted. Reading the Modem Status Register resets this interrupt. This register is read only; writing has no effect. Also see Table 15-9.

ISR Sharing, i.e., Callbacks in C

- There is only one interrupt handler
- Functions have to "subscribe" for events
- Callbacks
  - Driver provides function to register a function pointer
  - Driver stores function pointers in list
  - Upon interrupt, each registered function gets called

```
typedef void (*radioalarm_handler_t)(void);
radioalarm_handler_t radio_alarm_fired;

void RadioAlarm_init(radioalarm_handler_t handler)
{
    radio_alarm_fired = handler;
}

__attribute__((__interrupt__)) void Timer1_IRQHandler()
{
   alarm_state = FREE;
   MSS_TIM1_disable_irq();
    MSS_TIM1_clear_irq();
    NVIC_ClearPendingIRQ( Timer1_IRQn );
   (*(radio_alarm_fired))(); // call the callback function
}
```

Common Problems and Pit-Falls

- Too many interrupts
  - Your core can’t keep up with handling interrupts
- Concurrency issues
  - One interrupt handler modifies global variables
  - Can be avoided using atomic sections protected through PRIMASK
- Lost interrupts
  - It can happen that an interrupt doesn’t get treated by the Core
  - State machine and peripheral has to be aware of this possibility
  - Danger for deadlocks

Summary

- Overwrite default Interrupt Handler

- Initialization
  - Enable interrupt in NVIC
  - Enable interrupt in Peripheral

- Upon Interrupt
  - Clear interrupt in Peripheral
  - Clear pending bit in NVIC
  - *Potentially disable interrupts temporarely*