香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

# EIE3080
# Microprocessors and Computer Systems

## Exceptions and Interrupts

**Instructor：Tin Lun LAM**

E-mail: tllam@cuhk.edu.cn

URL: https://myweb.cuhk.edu.cn/tllam

# Polling

# Polling



- **Polling**, or **polled** operation, in computer science, refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of input/output (I/O), and is also referred to as polled I/O or software-driven I/O.
- Polling is sometimes used synonymously with **busy-wait** polling (busy waiting). In this situation, when an I/O operation is required, the computer does nothing other than check the status of the I/O device until it is ready, at which point the device is accessed. In other words, the computer waits until the device is ready.
- Polling also refers to the situation where a device is repeatedly checked for readiness, and if it is not, the computer returns to a different task. Although not as wasteful of CPU cycles as busy waiting, this is generally not as efficient as the alternative to polling, interrupt-driven I/O.

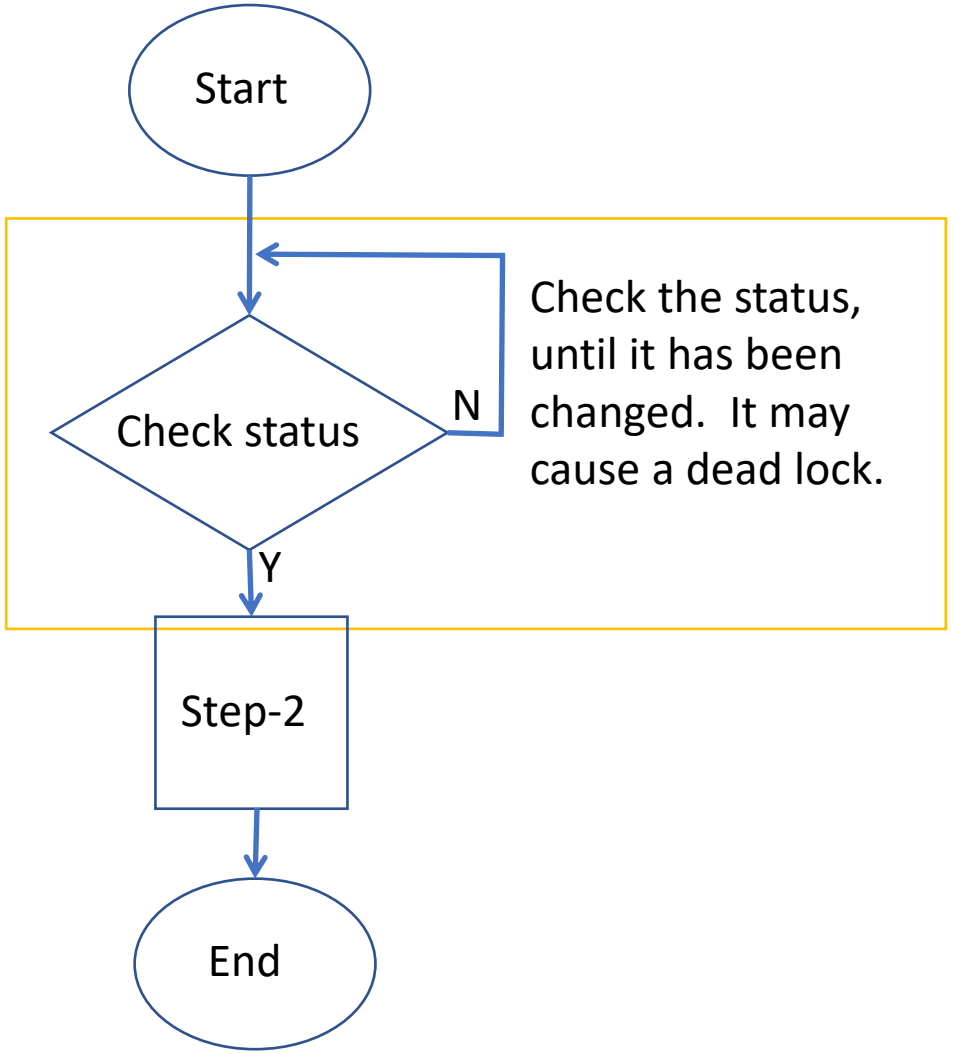
Contents from http://en.wikipedia.org/wiki/Polling_%28computer_science%29

# Polling (busy-wait)



- **Busy-waiting** or **spinning** is a technique in which a process repeatedly checks whether a condition is true, such as whether a key is pressed as in EIE3810 Lab1.

```
while (status != Y)
{
  ; // do nothing and loop back
}
```

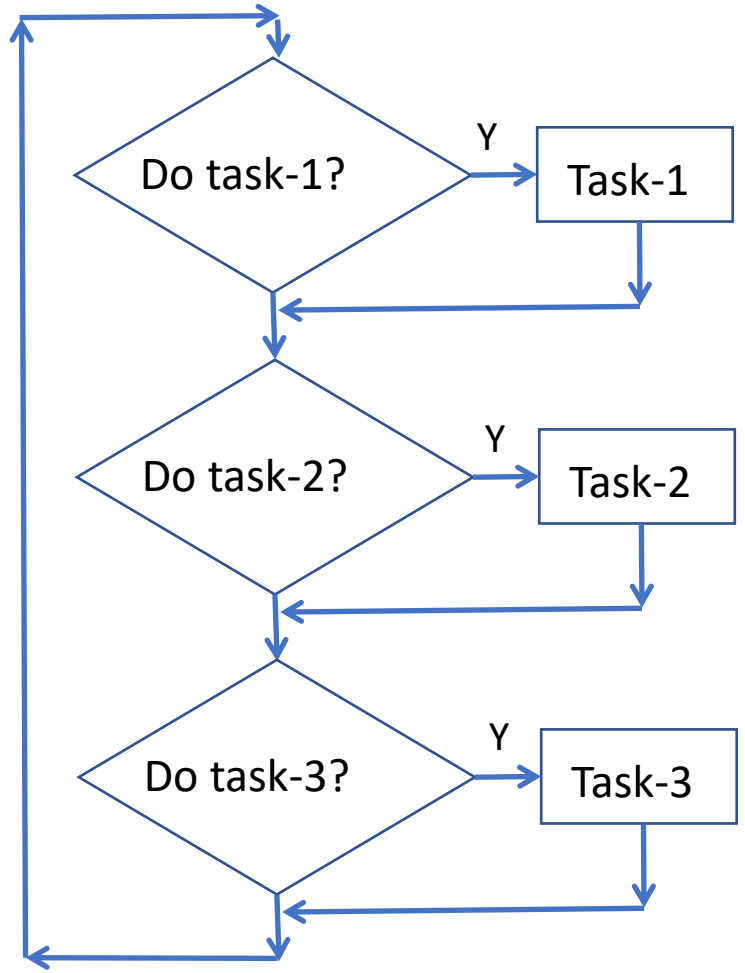



http://en.wikipedia.org/wiki/Busy_waiting

# Polling (roll call polling)



- In **roll call polling**, the polling device or process queries each element on a list in a fixed sequence.
- A timer is needed to prevent lock-ups caused by some non-responding elements.
- The scheme is inefficient if (i) the overhead for the polling messages is high or (ii) there are many elements to be polled in each polling cycle and only a few elements are active.




http://en.wikipedia.org/wiki/Polling_%28computer_science%29

# Polling



- Polling is like fire service center calling each house periodically and ask ‘ **Is your home on fire? Do you need emergency services?** ’ .
- That is very inefficient and wasteful for resources.
- Only suitable if there are only a few houses in that small village.

Pictures from http://www.tested.com

# Interrupt

# Interrupt – Definition



- An **interrupt** is a signal generated by hardware or software to tell the processor that an event needs immediate attention.
- Then the processor may pause the current code’ s execution to handle something more important/urgent.
- If the interrupt is accepted, the processor will suspend its current activities, save its state, and then execute a function called an *interrupt handler* (or an interrupt service routine, ISR) to deal with the event.
- After the interrupt handler finishes, the processor resumes normal activities.
- There are two types of interrupts: hardware interrupts and software interrupts.

http://en.wikipedia.org/wiki/Interrupt

# Interrupt – Analogy



- In case a house is on fire, you will break the glass of the ‘alarm unit’ (the hardware interrupt) to inform emergency call center (the interrupt controller).
- Of course, you hope fire engines to come and put out the fire (interrupt handler) as soon as possible (fast response time).

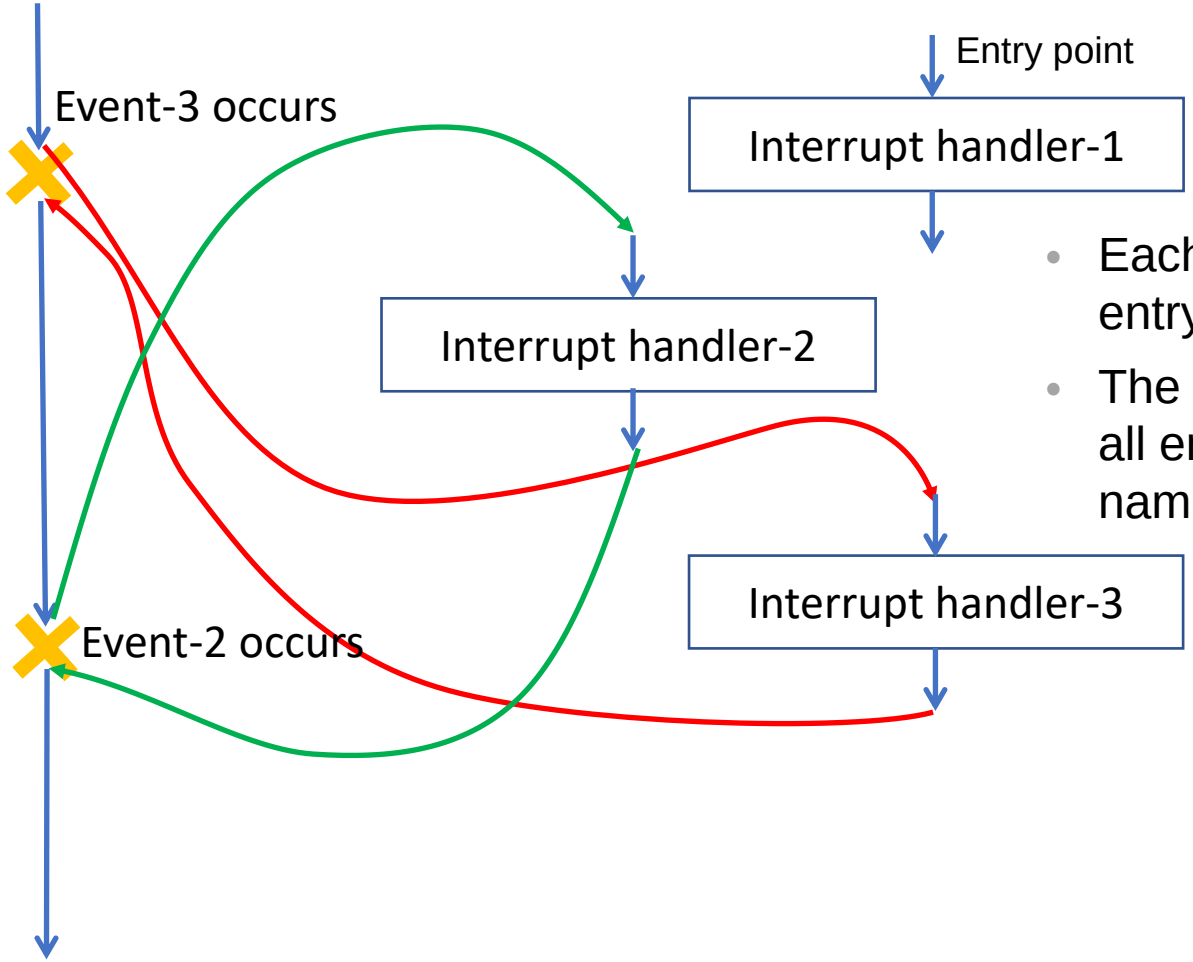
Pictures from alefiresystemsltd.co.uk, http://artworkbydon.deviantart.com

# Handler and Vector

# Interrupt Handling



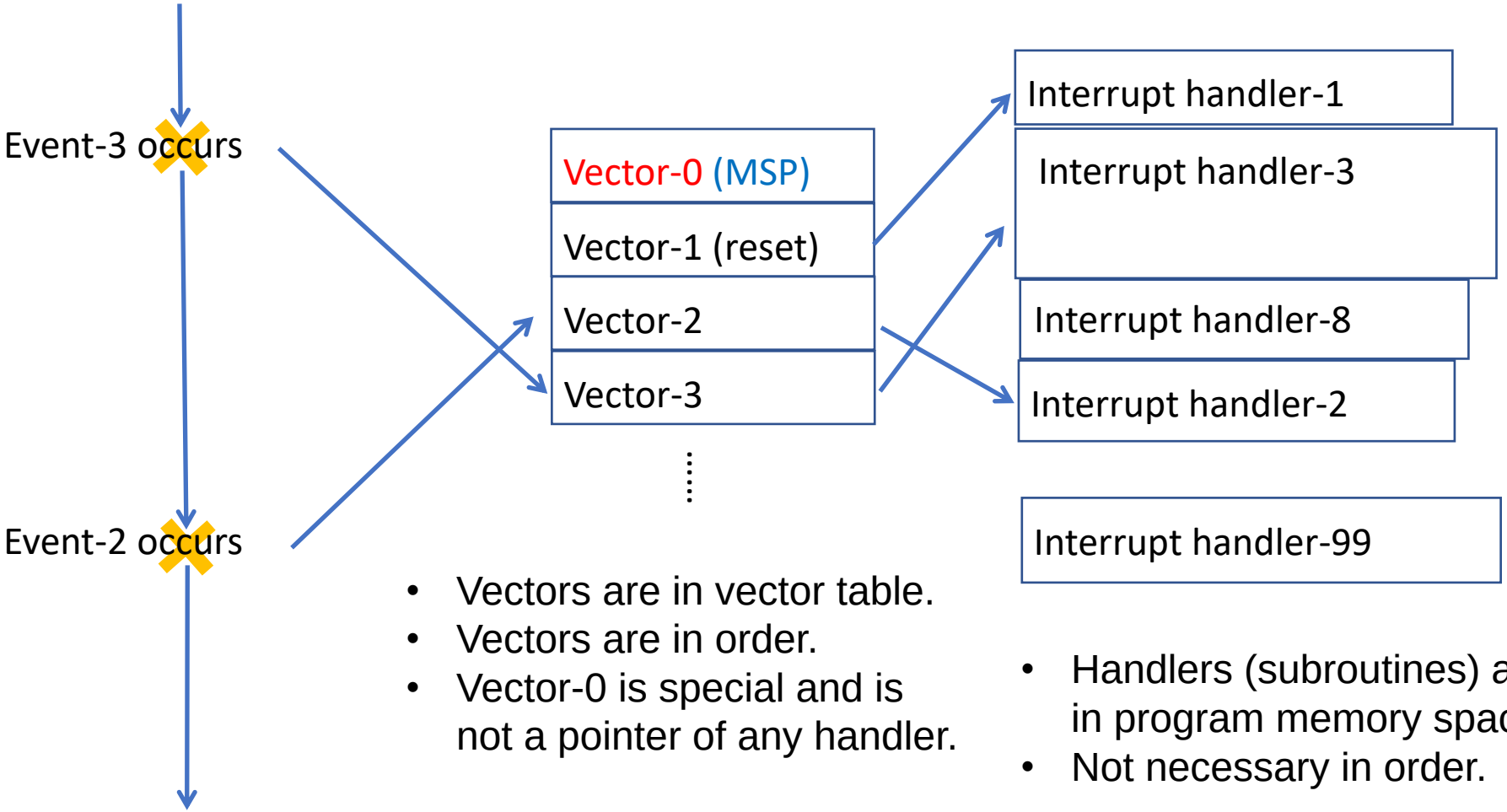
- Not necessary using while() or if() to check the status.
- Status checking (initiated by CPU) now changes to event triggering (initiated by events, e.g. key press).
- They may not be in order, unlike the case in polling.




- Each handler has an entry point.
- The table that collects all entry points is named vector table.

# Interrupt (vector and handler)






- Vectors are in vector table.
- Vectors are in order.
- Vector-0 is special and is not a pointer of any handler.

- Handlers (subroutines) are in program memory space.
- Not necessary in order.

# Nested Vectored Interrupt Controller (NVIC)



- All Cortex-M processors provide a Nested Vectored Interrupt Controller (NVIC) for interrupt handling.
- In addition to interrupt requests, there are other events that need servicing and we called them "exceptions." In ARM terminology, an interrupt is one type of exception.
- The pieces of program code that handle exceptions are often called exception handlers. They are part of the compiled program image.
- In a typical Cortex-M microcontroller, the NVIC receives interrupt requests from various sources.

- **Exception types:** *: The highest priority
  - Reset *
  - Non-maskable Interrupts (NMI) *
  - Faults *
  - PendSV
  - SVCall
  - External Interrupt ← 240 **external interrupt inputs, IRQ**
  - SysTick Interrupt

255

Other than 240 exceptions are **system exceptions**.

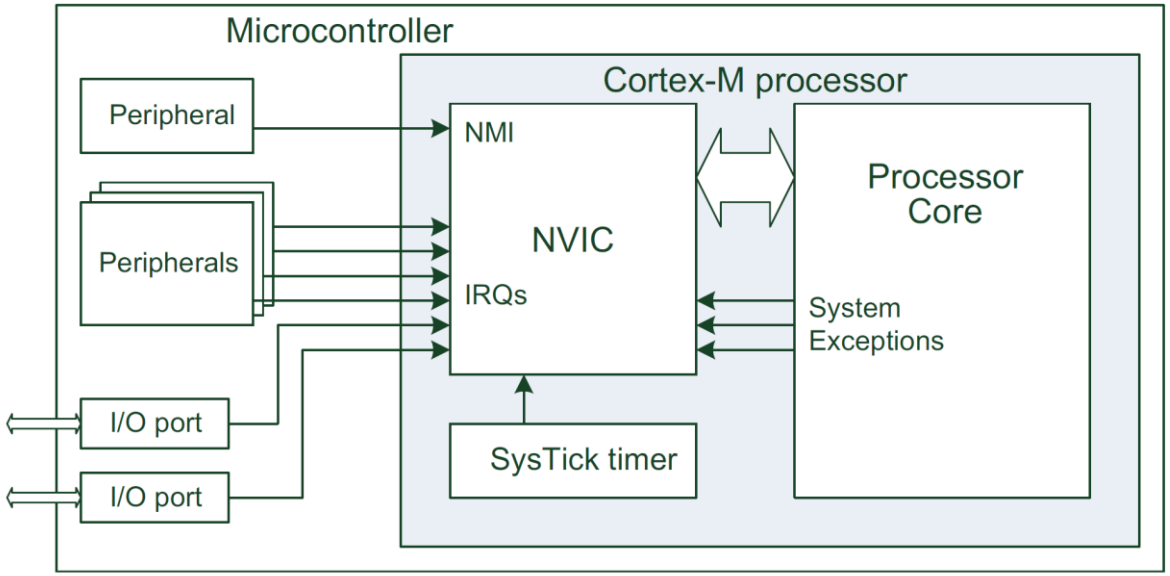



**FIGURE 7.1**

Various sources of exceptions in a typical microcontroller

# Exception Types



**Table 3.4** Exception Types in Cortex-M3

| Exception Number | Exception Type | Priority | Function |
|---|---|---|---|
| 1 | Reset | –3 (Highest) | Reset |
| 2 | NMI | –2 | Nonmaskable interrupt |
| 3 | Hard fault | –1 | All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked by exception masking |
| 4 | MemManage | Settable | Memory management fault; caused by MPU violation or invalid accesses (such as an instruction fetch from a nonexecutable region) |
| 5 | Bus fault | Settable | Error response received from the bus system; caused by an instruction prefetch abort or data access error |
| 6 | Usage fault | Settable | Usage fault; typical causes are invalid instructions or invalid state transition attempts (such as trying to switch to ARM state in the Cortex-M3) |
| 7–10 | — | — | Reserved |
| 11 | SVC | Settable | Supervisor call via SVC instruction |
| 12 | Debug monitor | Settable | Debug monitor |
| 13 | — | — | Reserved |
| 14 | PendSV | Settable | Pendable request for system service |
| 15 | SYSTICK | Settable | System tick timer |
| 16–255 | IRQ | Settable | IRQ input #0–239 |

system exception 15

external interrupt 240

# Exceptions (Cortex-M3 vector table)



- At 0x00, it is the starting value of the MSP, not an interrupt vector.

**Table 3.5** Vector Table Definition after Reset

| Exception Number | Address Offset | Exception Vector |
|---|---|---|
| 18–255 | 0x48–0x3FF | IRQ #2–239 |
| 17 | 0x44 | IRQ #1 |
| 16 | 0x40 | IRQ #0 |
| 15 | 0x3C | SYSTICK |
| 14 | 0x38 | PendSV |
| 13 | 0x34 | Reserved |
| 12 | 0x30 | Debug monitor |
| 11 | 0x2C | SVC |
| 7–10 | 0x1C–0x28 | Reserved |
| 6 | 0x18 | Usage fault |
| 5 | 0x14 | Bus fault |
| 4 | 0x10 | MemManage fault |
| 3 | 0x0C | Hard fault |
| 2 | 0x08 | NMI |
| 1 | 0x04 | Reset |
| 0 | 0x00 | Starting value of the MSP |

exception

external interrupt

system exception

Pictures from text book

# External Interrupt



- Exception # 16-255 are the external interrupt
- *The external interrupt input pin number might not match the interrupt input number on the NVIC.*
- For example, the first few interrupt inputs on NVIC might be assigned to internal peripherals (not accessible from chip outside), and external interrupt pins could be assigned to the remaining interrupt inputs.

**Table 7.2** List of External Interrupts

| Exception Number | Exception Type | Priority |
|---|---|---|
| 16 | External Interrupt #0 | Programmable |
| 17 | External Interrupt #1 | Programmable |
| … | … | … |
| 255 | External Interrupt #239 | Programmable |

**Interrupt number on NVIC**

# Address of exception



- There are two addresses involved in exception: the address of the vectors and the value of the vectors (the starting address of the vector's corresponding handler).
- Q: How does the processor locate the starting address of the exception handler? A: Information of the starting address of each handler (the vector value) is stored in the vector table in the memory.
- By default, the vector table starts at memory address 0x0, and the vector address is arranged according to the exception number times four.

Address of the vector

Value of the vector (the starting address of the handler)

**Table 7.6** Exception Vector Table After Power Up

| 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 |

# STM32F103 (XL) Vector Tables



- Our STM32F103ZET6 (XL) only uses **60 external interrupt inputs**.
- STM32 line product (i.e. with Ethernet function) has 68 external interrupt inputs.

**Table 62. Vector table for XL-density devices**

| Position | Priority | Type of priority | Acronym | Description | Address |
|---|---|---|---|---|---|
| | - | - | - | Reserved | 0x0000_0000 |
| | -3 | fixed | Reset | Reset | 0x0000_0004 |
| | -2 | fixed | NMI | Nonmaskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector. | 0x0000_0008 |
| | -1 | fixed | HardFault | All class of fault | 0x0000_000C |
| | 0 | settable | MemManage | Memory management | 0x0000_0010 |
| | 1 | settable | BusFault | Prefetch fault, memory access fault | 0x0000_0014 |
| | 2 | settable | UsageFault | Undefined instruction or illegal state | 0x0000_0018 |
| | - | - | - | Reserved | 0x0000_001C - 0x0000_002B |
| | 3 | settable | SVCall | System service call via SWI instruction | 0x0000_002C |
| | 4 | settable | Debug Monitor | Debug monitor | 0x0000_0030 |
| | - | - | - | Reserved | 0x0000_0034 |
| | 5 | settable | PendSV | Pendable request for system service | 0x0000_0038 |
| | 6 | settable | SysTick | Systick timer | 0x0000_003C |
| 0 | 7 | settable | WWDG | Window watchdog interrupt | 0x0000_0040 |

Pictures from RM0008

# STM32F103 (XL) Vector Tables



Q: Could you find the interrupts for EXTI-0 to EXTI-15?

Table 62. Vector table for XL-density devices (continued)

| Position | Priority | Type of priority | Acronym | Description | Address |
|---|---|---|---|---|---|
| 1 | 8 | settable | PVD | PVD through EXTI Line detection interrupt | 0x0000_0044 |
| 2 | 9 | settable | TAMPER | Tamper interrupt | 0x0000_0048 |
| 3 | 10 | settable | RTC | RTC global interrupt | 0x0000_004C |
| 4 | 11 | settable | FLASH | Flash global interrupt | 0x0000_0050 |
| 5 | 12 | settable | RCC | RCC global interrupt | 0x0000_0054 |
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt | 0x0000_0058 |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt | 0x0000_005C |
| 8 | 15 | settable | EXTI2 | EXTI Line2 interrupt | 0x0000_0060 |
| 9 | 16 | settable | EXTI3 | EXTI Line3 interrupt | 0x0000_0064 |
| 10 | 17 | settable | EXTI4 | EXTI Line4 interrupt | 0x0000_0068 |
| 11 | 18 | settable | DMA1_Channel1 | DMA1 Channel1 global interrupt | 0x0000_006C |
| 12 | 19 | settable | DMA1_Channel2 | DMA1 Channel2 global interrupt | 0x0000_0070 |
| 13 | 20 | settable | DMA1_Channel3 | DMA1 Channel3 global interrupt | 0x0000_0074 |
| 14 | 21 | settable | DMA1_Channel4 | DMA1 Channel4 global interrupt | 0x0000_0078 |
| 15 | 22 | settable | DMA1_Channel5 | DMA1 Channel5 global interrupt | 0x0000_007C |
| 16 | 23 | settable | DMA1_Channel6 | DMA1 Channel6 global interrupt | 0x0000_0080 |
| 17 | 24 | settable | DMA1_Channel7 | DMA1 Channel7 global interrupt | 0x0000_0084 |
| 18 | 25 | settable | ADC1_2 | ADC1 and ADC2 global interrupt | 0x0000_0088 |
| 19 | 26 | settable | USB_HP_CAN_TX | USB high priority or CAN TX interrupts | 0x0000_008C |
| 20 | 27 | settable | USB_LP_CAN_RX0 | USB low priority or CAN RX0 interrupts | 0x0000_0090 |
| 21 | 28 | settable | CAN_RX1 | CAN RX1 interrupt | 0x0000_0094 |
| 22 | 29 | settable | CAN_SCE | CAN SCE interrupt | 0x0000_0098 |
| 23 | 30 | settable | EXTI9_5 | EXTI Line[9:5] interrupts | 0x0000_009C |
| 24 | 31 | settable | TIM1_BRK_TIM9 | TIM1 Break interrupt and TIM9 global interrupt | 0x0000_00A0 |
| 25 | 32 | settable | TIM1_UP_TIM10 | TIM1 Update interrupt and TIM10 global interrupt | 0x0000_00A4 |
| 26 | 33 | settable | TIM1_TRG_COM_TIM11 | TIM1 Trigger and Commutation interrupts and TIM11 global interrupt | 0x0000_00A8 |
| 27 | 34 | settable | TIM1_CC | TIM1 Capture Compare interrupt | 0x0000_00AC |
| 28 | 35 | settable | TIM2 | TIM2 global interrupt | 0x0000_00B0 |
| 29 | 36 | settable | TIM3 | TIM3 global interrupt | 0x0000_00B4 |
| 30 | 37 | settable | TIM4 | TIM4 global interrupt | 0x0000_00B8 |

Table 62. Vector table for XL-density devices (continued)

| Position | Priority | Type of priority | Acronym | Description | Address |
|---|---|---|---|---|---|
| 31 | 38 | settable | I2C1_EV | I2C1 event interrupt | 0x0000_00BC |
| 32 | 39 | settable | I2C1_ER | I2C1 error interrupt | 0x0000_00C0 |
| 33 | 40 | settable | I2C2_EV | I2C2 event interrupt | 0x0000_00C4 |
| 34 | 41 | settable | I2C2_ER | I2C2 error interrupt | 0x0000_00C8 |
| 35 | 42 | settable | SPI1 | SPI1 global interrupt | 0x0000_00CC |
| 36 | 43 | settable | SPI2 | SPI2 global interrupt | 0x0000_00D0 |
| 37 | 44 | settable | USART1 | USART1 global interrupt | 0x0000_00D4 |
| 38 | 45 | settable | USART2 | USART2 global interrupt | 0x0000_00D8 |
| 39 | 46 | settable | USART3 | USART3 global interrupt | 0x0000_00DC |
| 40 | 47 | settable | EXTI15_10 | EXTI Line[15:10] interrupts | 0x0000_00E0 |
| 41 | 48 | settable | RTCAlarm | RTC alarm through EXTI line interrupt | 0x0000_00E4 |
| 42 | 49 | settable | USBWakeUp | USB wakeup from suspend through EXTI line interrupt | 0x0000_00E8 |
| 43 | 50 | settable | TIM8_BRK_TIM12 | TIM8 Break interrupt and TIM12 global interrupt | 0x0000_00EC |
| 44 | 51 | settable | TIM8_UP_TIM13 | TIM8 Update interrupt and TIM13 global interrupt | 0x0000_00F0 |
| 45 | 52 | settable | TIM8_TRG_COM_TIM14 | TIM8 Trigger and Commutation interrupts and TIM14 global interrupt | 0x0000_00F4 |
| 46 | 53 | settable | TIM8_CC | TIM8 Capture Compare interrupt | 0x0000_00F8 |
| 47 | 54 | settable | ADC3 | ADC3 global interrupt | 0x0000_00FC |
| 48 | 55 | settable | FSMC | FSMC global interrupt | 0x0000_0100 |
| 49 | 56 | settable | SDIO | SDIO global interrupt | 0x0000_0104 |
| 50 | 57 | settable | TIM5 | TIM5 global interrupt | 0x0000_0108 |
| 51 | 58 | settable | SPI3 | SPI3 global interrupt | 0x0000_010C |
| 52 | 59 | settable | UART4 | UART4 global interrupt | 0x0000_0110 |
| 53 | 60 | settable | UART5 | UART5 global interrupt | 0x0000_0114 |
| 54 | 61 | settable | TIM6 | TIM6 global interrupt | 0x0000_0118 |
| 55 | 62 | settable | TIM7 | TIM7 global interrupt | 0x0000_011C |
| 56 | 63 | settable | DMA2_Channel1 | DMA2 Channel1 global interrupt | 0x0000_0120 |
| 57 | 64 | settable | DMA2_Channel2 | DMA2 Channel2 global interrupt | 0x0000_0124 |
| 58 | 65 | settable | DMA2_Channel3 | DMA2 Channel3 global interrupt | 0x0000_0128 |
| 59 | 66 | settable | DMA2_Channel4_5 | DMA2 Channel4 and DMA2 Channel5 global interrupts | 0x0000_012C |

Pictures from RM0008

# EXTI9_5 / EXTI15_10 shares the same interrupt handler



```
void EXTI9_5_IRQHandler(void){
        if(EXTI_GetITStatus(EXTI_Line9)!=RESET){                    …
                EXTI_ClearITPendingBit(EXTI_Line9);
        }
        if(EXTI_GetITStatus(EXTI_Line8)!=RESET){                    …
                EXTI_ClearITPendingBit(EXTI_Line8);
        }
        …
}
```

# Interrupt (Handler)



- Each interrupt has its own interrupt handler.
- The number of hardware interrupts is limited by the number of interrupt request (IRQ) lines to the processor, but there may be hundreds of different software interrupts.
- Interrupts are a commonly used technique for computer multitasking, especially in real-time computing. Such a system is said to be interrupt-driven.

Contents from http://en.wikipedia.org/wiki/Interrupt

# Relocation of vector table

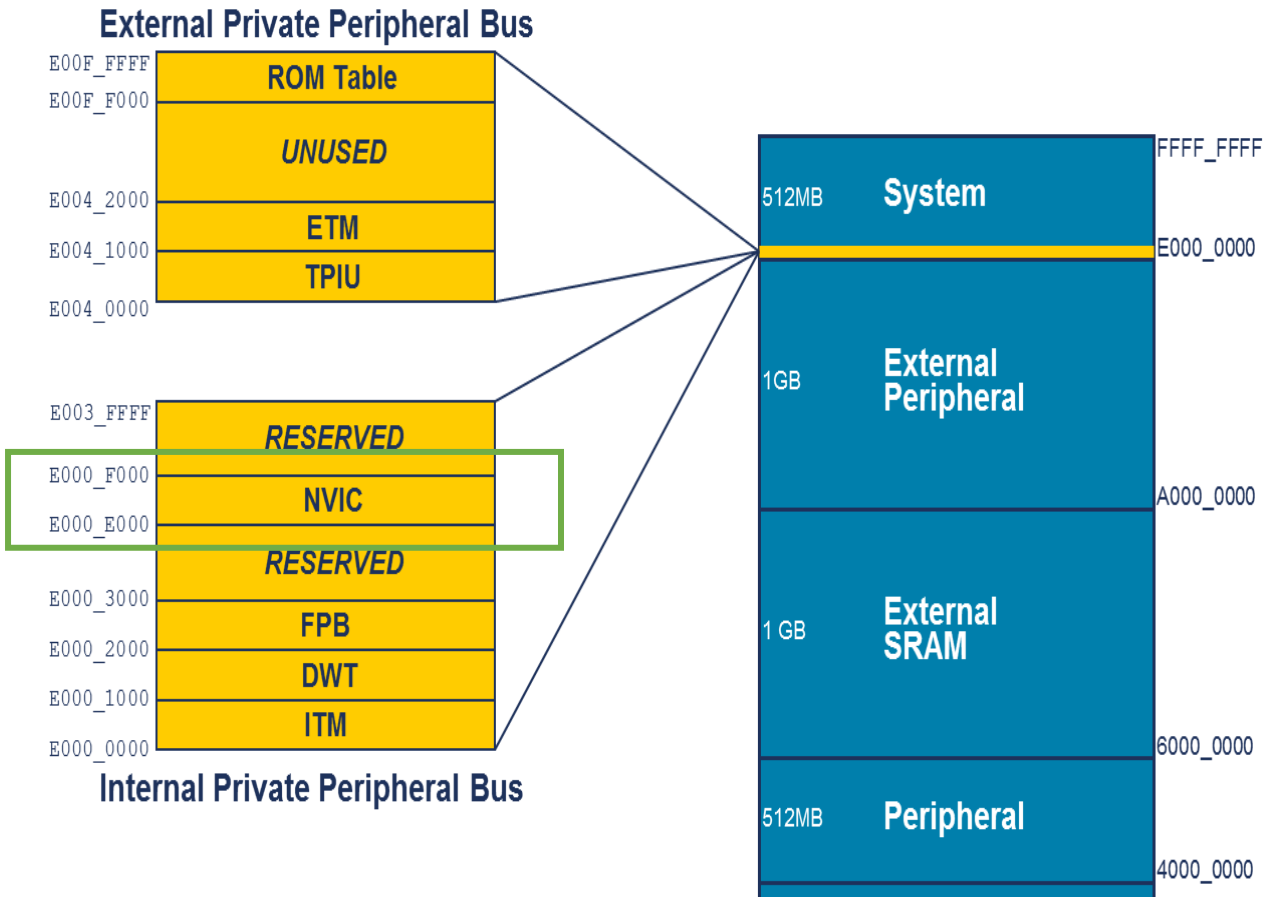


- Since the address 0x0 should be boot code, usually it will be either Flash memory or ROM devices, and the value cannot be changed at run time.
- However, the vector table can be relocated to other memory locations in the code or Random Access Memory (RAM) region where the RAM is, so that we can change the handlers during run time. (See Textbook 11.4 relocation example)
- This is done by setting a register in the NVIC called the vector table offset register *(address 0xE000ED08) (Q: Which region of the memory space is that?). The address offset should be aligned to the vector* table size (extended to the next larger power of 2, if not power of 2).

# Relocation of vector table



- For example, if there are 32 IRQ inputs, the total number of exceptions will be 32 + 16 (system exceptions) = 48. Extending it to the power of 2 makes it 64. Multiplying it by 4 (4 bytes per vector) makes it 256 bytes (0x100). Therefore, the vector table offset can be programmed as 0x0, 0x100, 0x200, and so on.



# Vector Table Offset Register



- The vector table offset register (VTOR) contains TBLBASE and TBLOFF, as shown in the following.
- If dynamic changing of exception handlers is needed, at the beginning of the boot image, you need to have the following (at least): Initial main stack pointer value, Reset vector, NMI vector, and Hard fault vector.
- These are required because the NMI and hard fault can potentially occur during your boot process.
- Other exceptions will not take place unless the exception are enabled.
- When the booting process is done, you can define a part of your SRAM as the new vector table and relocate the vector table to the new one, which is writable.

**Table 7.7** Vector Table Offset Register (Address 0xE000ED08)

| Bits | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 29 | TBLBASE | R/W | 0 | Table base in code (0) or RAM (1) |
| 28:7 | TBLOFF | R/W | 0 | Table offset value from code region or RAM region |

Q: Why only 22 bits for TBLOFF?

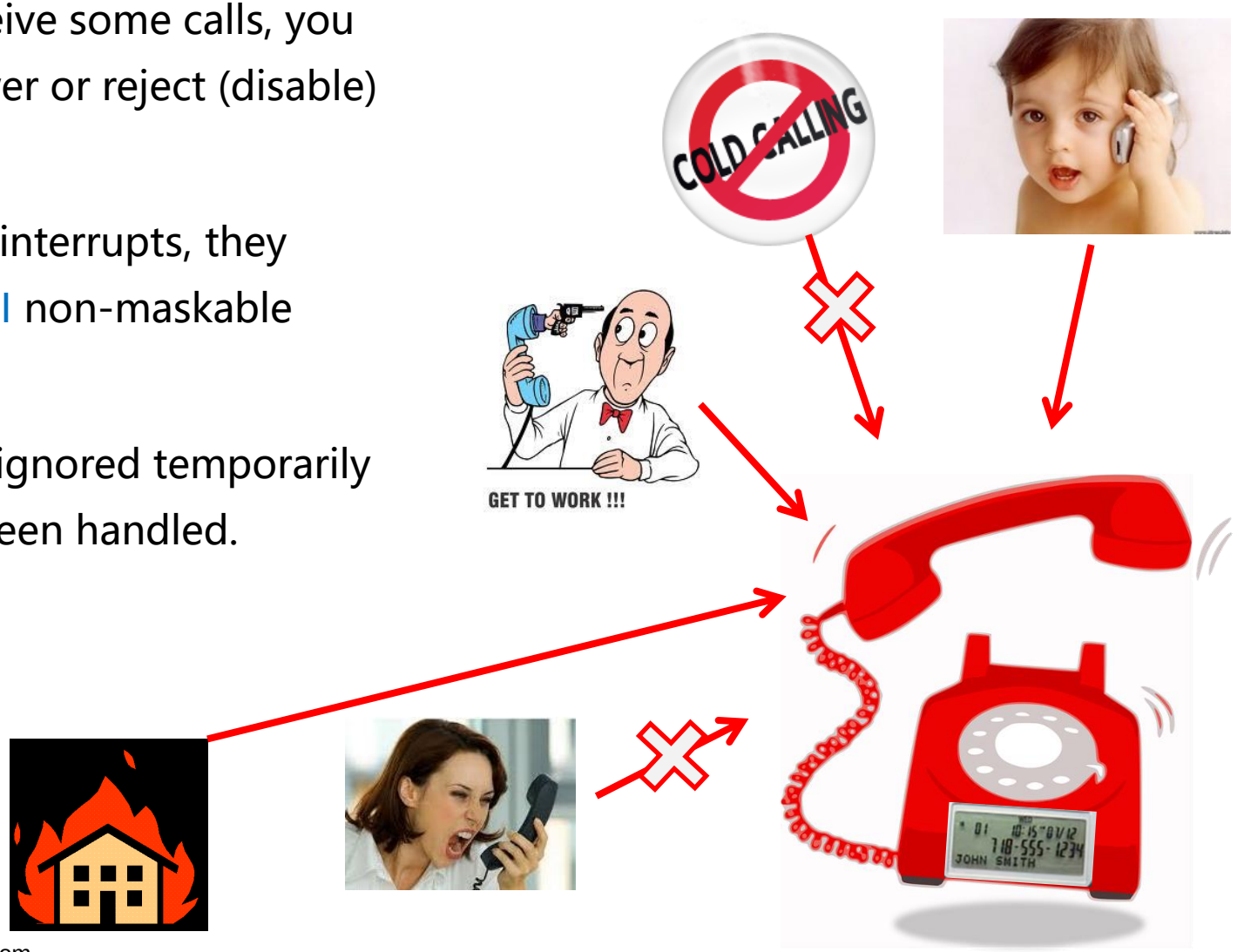
Hint: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/Ciheijba.html

# Enable / Disable

# Enable and Disable



- If you don't want to receive some calls, you can set their priority lower or reject (disable) them.
- But for some important interrupts, they cannot be rejected. (NMI non-maskable interrupt)
- Some interrupts can be ignored temporarily if an emergency job is been handled.




Contents from http://fibromodem.com, http://careerguru.co, http://apextwo.com

# Enable and Disable



- The Interrupt Enable register is programmed through two addresses; SETENAs (0xE000 E100 ~ 0xE000 E11C) and CLRENAs (0xE000 E180 ~ 0xE000 E19C).
- To set the enable bit, you need to write to the SETENA register address.
- To clear the enable bit, you need to write to the CLRENA register address.
- Enabling or disabling an interrupt will not affect other interrupt enable states. The SETENA/CLRENA registers are 32 bits wide; each bit represents one interrupt input.
- As there could be more than 32 external interrupts in the Cortex-M3 processor, you might find more than one SETENA and CLRENA register—for example, SETENA0, SETENA1, and so on (see Table 8.1). Only the enable bits for interrupts that exist are implemented. So, if you have only 32 interrupt inputs, you will only have SETENA0 and CLRENA0.
- The SETENA and CLRENA registers can be accessed as word, half word, or byte. As the first 16 exception types are system exceptions, external Interrupt #0 has a start exception number of 16.

Contents from Text book

# Enable and Disable

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

**Table 8.1** Interrupt Set Enable Registers and Interrupt Clear Enable Registers (0xE000E100-0xE000E11C, 0xE000E180-0xE000E19C)

| Address | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 0xE000E100 | SETENA0 | R/W | 0 | Enable for external Interrupt #0–31<br>bit[0] for Interrupt #0 (exception #16)<br>bit[1] for Interrupt #1 (exception #17)<br>…<br>bit[31] for Interrupt #31 (exception #47)<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status |
| 0xE000E104 | SETENA1 | R/W | 0 | Enable for external Interrupt #32–63<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status |
| 0xE000E108 | SETENA2 | R/W | 0 | Enable for external Interrupt #64–95<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status |
| … | — | — | — | — |
| 0xE000E180 | CLRENA0 | R/W | 0 | Clear enable for external Interrupt #0–31<br>bit[0] for Interrupt #0<br>bit[1] for Interrupt #1<br>…<br>bit[31] for Interrupt #31<br>Write 1 to clear bit to 0;<br>write 0 has no effect<br>Read value indicates the current enable status |
| 0xE000E184 | CLRENA1 | R/W | 0 | Clear enable for external Interrupt #32–63<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current enable status |
| 0xE000E188 | CLRENA2 | R/W | 0 | Clear enable for external Interrupt #64–95<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current enable status |
| … | — | — | — | — |

Contents from Text book

# Enable and Disable



Table 6-1 shows the NVIC registers.

**Table 6-1 NVIC registers**

| Address | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0xE000E004 | ICTR | RO | - | *Interrupt Controller Type Register, ICTR* |
| 0xE000E100 - 0xE000E11C | NVIC_ISER0 - NVIC_ISER7 | RW | 0x00000000 | Interrupt Set-Enable Registers |
| 0xE000E180 - 0E000xE19C | NVIC_ICER0 - NVIC_ICER7 | RW | 0x00000000 | Interrupt Clear-Enable Registers |
| 0xE000E200 - 0xE000E21C | NVIC_ISPR0 - NVIC_ISPR7 | RW | 0x00000000 | Interrupt Set-Pending Registers |
| 0xE000E280 - 0xE000E29C | NVIC_ICPR0 - NVIC_ICPR7 | RW | 0x00000000 | Interrupt Clear-Pending Registers |
| 0xE000E300 - 0xE000E31C | NVIC_IABR0 - NVIC_IABR7 | RO | 0x00000000 | Interrupt Active Bit Register |
| 0xE000E400 - 0xE000E4EC | NVIC_IPR0 - NVIC_IPR59 | RW | 0x00000000 | Interrupt Priority Register |

Contents from DDI0337

# Active Status



- Each external interrupt has an active status bit. When the processor starts the interrupt handler, that bit is set to 1 and cleared when the interrupt return is executed.
- However, during an Interrupt Service Routine (ISR) execution, a higher-priority interrupt might occur and cause preemption. During this period, although the processor is executing another interrupt handler, the previous interrupt is still defined as active.
- The active registers are 32 bit but can also be accessed using half word or byte-size transfers. If there are more than 32 external interrupts, there will be more than one active register. The active status registers for external interrupts are read-only (see Table 8.4).

**Table 8.4** Interrupt Active Status Registers (0xE000E300-0xE000E31C)

| Address | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 0xE000E300 | ACTIVE0 | R | 0 | Active status for external Interrupt #0–31<br>bit[0] for Interrupt #0<br>bit[1] for Interrupt #1<br>…<br>bit[31] for Interrupt #31 |
| 0xE000E304 | ACTIVE1 | R | 0 | Active status for external Interrupt #32–63 |
| … | — | — | — | — |

Contents from Text book

# Active Bit



- An Active Bit Register is used to determine which interrupts are active.
- Each flag in the register corresponds to one of the 32 interrupts.

**Table 6-1 NVIC registers**

| Address | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0xE000E004 | ICTR | RO | - | *Interrupt Controller Type Register, ICTR* |
| 0xE000E100 - 0xE000E11C | NVIC_ISER0 - NVIC_ISER7 | RW | 0x00000000 | Interrupt Set-Enable Registers |
| 0xE000E180 - 0E000xE19C | NVIC_ICER0 - NVIC_ICER7 | RW | 0x00000000 | Interrupt Clear-Enable Registers |
| 0xE000E200 - 0xE000E21C | NVIC_ISPR0 - NVIC_ISPR7 | RW | 0x00000000 | Interrupt Set-Pending Registers |
| 0xE000E280 - 0xE000E29C | NVIC_ICPR0 - NVIC_ICPR7 | RW | 0x00000000 | Interrupt Clear-Pending Registers |
| 0xE000E300 - 0xE000E31C | NVIC_IABR0 - NVIC_IABR7 | RO | 0x00000000 | Interrupt Active Bit Register |
| 0xE000E400 - 0xE000E4EC | NVIC_IPR0 - NVIC_IPR59 | RW | 0x00000000 | Interrupt Priority Register |

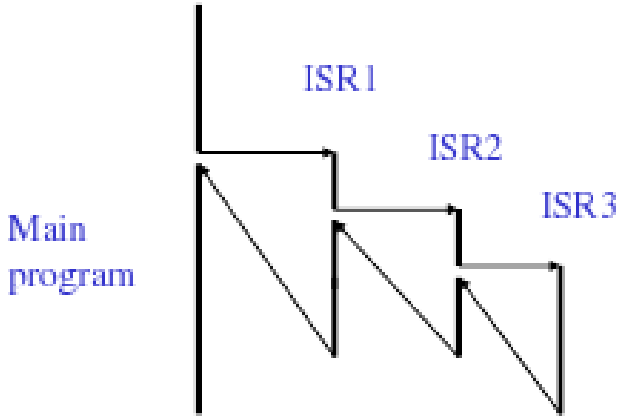
# Priority

# Nested Interrupt



- "Interrupt in an interrupt"
- Nested interrupt is that an interrupt being served (ISR1) and another interrupt request of higher priority (ISR2) comes. Then the current ISR1 will be suspended and the microprocessor will serve the higher priority interrupt (ISR2) first. It can be many levels of nested interrupts if the subsequent interrupt requests are of higher priority than that of the interrupt being served.

  - What interrupt request(s) cannot interrupt current ISR?

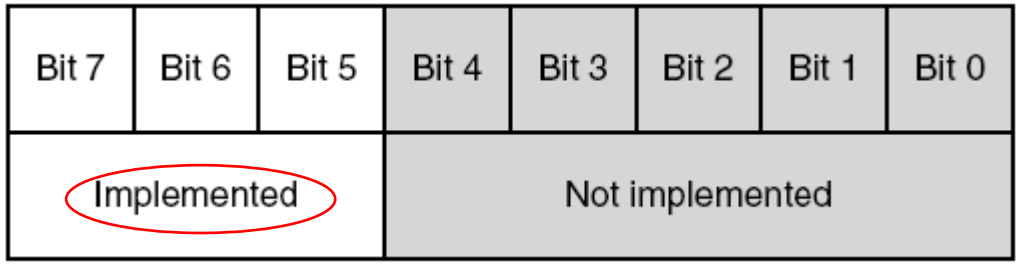



# Definitions of Exception Priority



- A higher-priority (smaller number in priority level) exception can preempt a lower-priority (larger number in priority level) exception.
- Some exceptions have negative and fixed numbers - reset, NMI, and hard fault. These are the interrupts with highest priority.
- Others have programmable priority level, which range from 0 to 255.
- Maximum 256 levels (8-bit) of programmable priority level can be set in Interrupt Priority-Level Register.
- But usually few number of levels are implemented; and least significant bits of the register are ignored. Why?

(This is because having large number of priority levels can increases the complexity of the NVIC and can increase power consumption and reduce the speed of the design.)




A Priority Level Register with 3 Bits Implemented

# Interrupt Priority Register



Table 6-1 shows the NVIC registers.

**Table 6-1 NVIC registers**

| Address | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0xE000E004 | ICTR | RO | - | *Interrupt Controller Type Register, ICTR* |
| 0xE000E100 - 0xE000E11C | NVIC_ISER0 - NVIC_ISER7 | RW | 0x00000000 | Interrupt Set-Enable Registers |
| 0xE000E180 - 0E000xE19C | NVIC_ICER0 - NVIC_ICER7 | RW | 0x00000000 | Interrupt Clear-Enable Registers |
| 0xE000E200 - 0xE000E21C | NVIC_ISPR0 - NVIC_ISPR7 | RW | 0x00000000 | Interrupt Set-Pending Registers |
| 0xE000E280 - 0xE000E29C | NVIC_ICPR0 - NVIC_ICPR7 | RW | 0x00000000 | Interrupt Clear-Pending Registers |
| 0xE000E300 - 0xE000E31C | NVIC_IABR0 - NVIC_IABR7 | RO | 0x00000000 | Interrupt Active Bit Register |
| 0xE000E400 - 0xE000E4EC | NVIC_IPR0 - NVIC_IPR59 | RW | 0x00000000 | Interrupt Priority Register |

Contents from DDI0337

# STM32F103xE interrupt controller

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Contents from STM32 data sheet

### 2.3.7 Nested vectored interrupt controller (NVIC)

The STM32F103xC, STM32F103xD and STM32F103xE performance line embeds a nested vectored interrupt controller able to handle up to 60 maskable interrupt channels (not including the 16 interrupt lines of Cortex®-M3) and 16 priority levels.

- Closely coupled NVIC gives low latency interrupt processing
- Interrupt entry vector table address passed directly to the core
- Closely coupled NVIC core interface
- Allows early processing of interrupts
- Processing of *late arriving* higher priority interrupts
- Support for tail-chaining
- Processor state automatically saved
- Interrupt entry restored on interrupt exit with no instruction overhead

This hardware block provides flexible interrupt management features with minimal interrupt latency.

Pictures from Text book

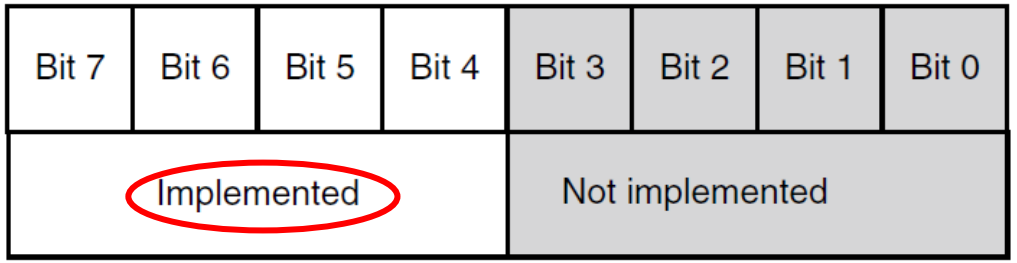


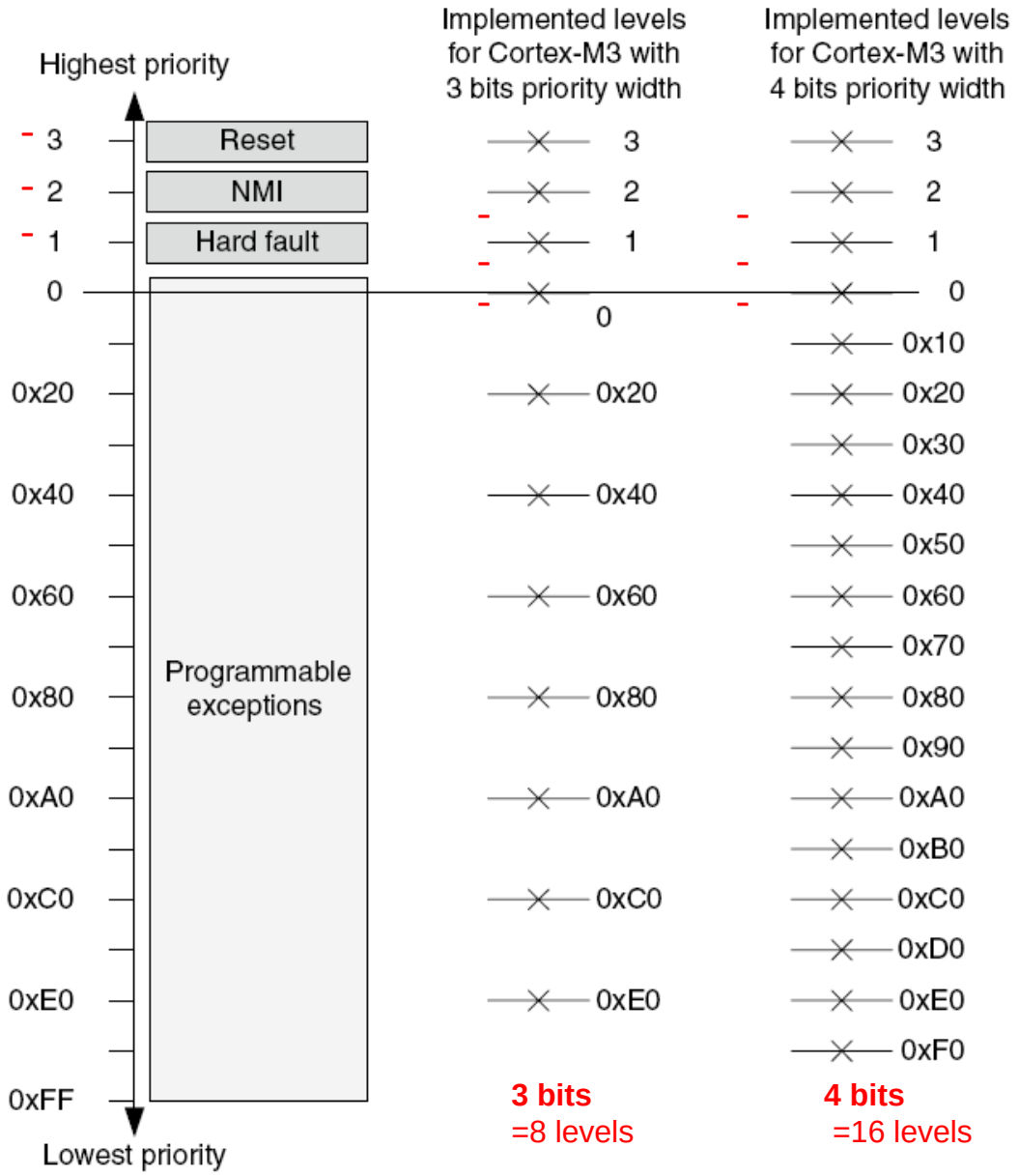

**FIGURE 7.2**

A Priority Level Register with 4 Bits Implemented.

# Available Priority Levels with 3-Bit or 4-Bit Priority Width



- More priority bits give more levels, but it also increases gate counts and hence the power consumption.
- For the Cortex-M3, the minimum number of implemented priority register widths is 3 bits (eight levels).



# Priority Group



- Priority Group divides Priority Level Register into two halves.
- The upper part (left bits) are used to determine the preempt priority, and the lower part (right bits) are to determine the subpriority (Table 7.5).

**Table 7.5** Definition of Preempt Priority Field and Subpriority Field in a Priority Level Register in Different Priority Group Settings

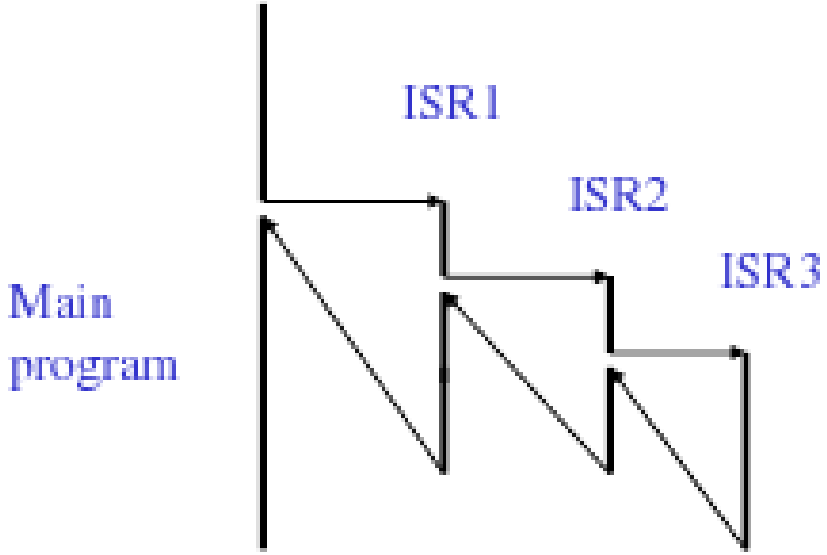
| Priority Group | Preempt Priority Field | Subpriority Field |
|---|---|---|
| 0 | Bit [7:1] | Bit [0] |
| 1 | Bit [7:2] | Bit [1:0] |
| 2 | Bit [7:3] | Bit [2:0] |
| 3 | Bit [7:4] | Bit [3:0] |
| 4 | Bit [7:5] | Bit [4:0] |
| 5 | Bit [7:6] | Bit [5:0] |
| 6 | Bit [7] | Bit [6:0] |
| 7 | None | Bit [7:0] |

Pictures from Text book

# Preempt Priority and Subpriority Levels



- The *preempt (group) priority level* defines whether an interrupt can take place when the processor is already running another interrupt handler. (for the nested interrupt case)
- The *subpriority level* is used only when two exceptions with the same preempt priority level occur at the same time. In this case, the exception with higher subpriority (lower value) will be handled first.

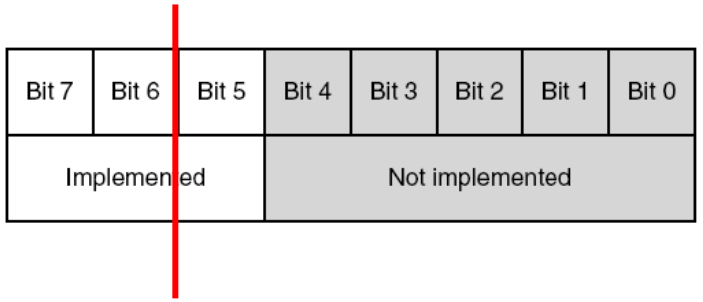


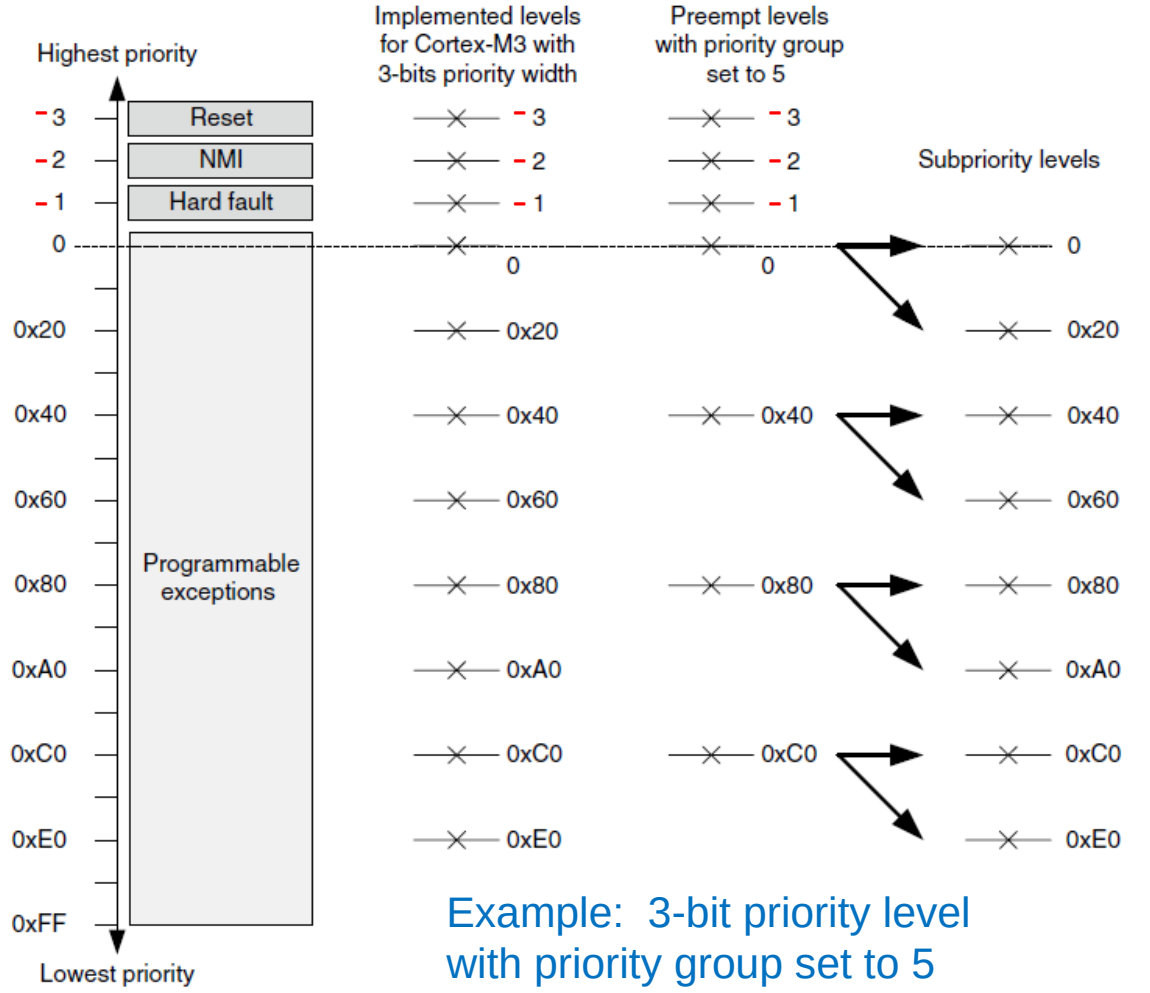

Contents from Text book

# Example of priority group setting



- Assume:
  1. 3 bits are implemented in Priority Level Register; and
  2. PRIGROUP=5




Example: 3-bit priority level with priority group set to 5

**FIGURE 7.5**

Available Priority Levels with 3-Bit Priority Width and Priority Group Set to 5.

Contents from Text book

# Preempt priority level



- With the priority grouping, the maximum width of preempt priority is 7, so there can be 128 preempt priority levels. (priority group is set to 0)
- When the priority group is set to 7, all exceptions with a programmable priority level will be in the same level, and no preemption between these exceptions will take place, except that hard fault, NMI, and reset can preempt these exceptions.

**Table 7.5** Definition of Preempt Priority Field and Subpriority Field in a Priority Level Register in Different Priority Group Settings

| 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] |

Pictures from Text book

# Other examples



- 3 bits implemented; PRIGROUP=1

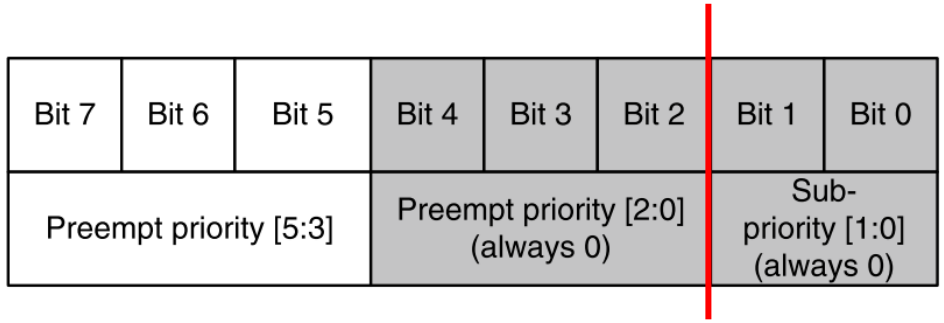



**FIGURE 7.6**

Definition of Priority Fields in an 8-Bit Priority Level Register with Priority Group Set to 1.

| 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] |

- 8 bits implemented; PRIGROUP=0

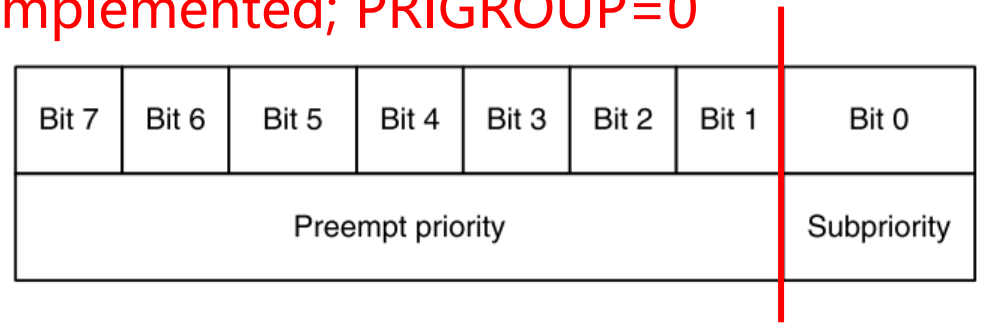



**FIGURE 7.8**

Definition of Priority Fields in an 8-Bit Priority Level Register with Priority Group Set to 0.



Pictures from Text book

# PRIGROUP



- The 3-bit PRIGROUP define Preempt priority level and Subpriority level. Figure 7.4 shows the case when PRIGROUP=5.

**Table 7.5** Definition of Preempt Priority Field and Subpriority Field in a Priority Level Register in Different Priority Group Settings

| 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] |

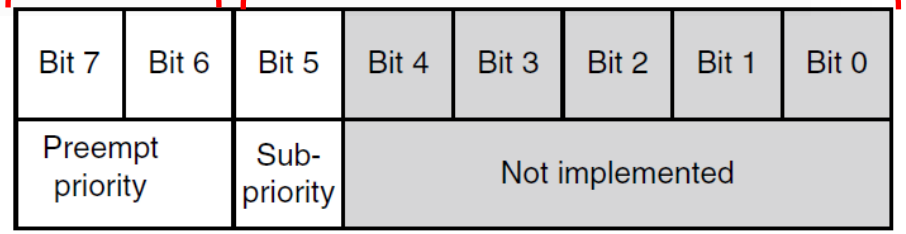



**FIGURE 7.4**

Definition of Priority Fields in a 3-Bit Priority Level Register with Priority Group Set to 5.

Pictures from Text book

# Configuration of Priority Group



- Using the 3-bit *Priority Group* register in a configuration register (Application Interrupt and Reset Control Register) in the NVIC (Table 7.4), the number of bit used for preempt priority-level implementation is determined.

**Table 7.4** 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 |

Pictures from Text book

# Question



- Q: What if both Preempt priority level and Subpriority level are the same?


Contents from Text book

# Interrupt Priority



- When two interrupts are asserted at the same time with exactly the same preempt priority level as well as subpriority level, the interrupt with the smaller exception number has higher priority. (IRQ #0 has higher priority than IRQ #1.)
- To avoid unexpected changes of priority levels for interrupts, be careful when writing to the Application Interrupt and Reset Control register (address 0xE000ED0C). In most cases, after the priority group is configured, there is no need to use this register except to generate a reset.

Table 62. Vector table for XL-density devices (continued)

| Position | Priority | Type of priority | Acronym | Description |
|---|---|---|---|---|
| 1 | 8 | settable | PVD | PVD through EXTI Line detection interrupt |
| 2 | 9 | settable | TAMPER | Tamper interrupt |
| 3 | 10 | settable | RTC | RTC global interrupt |
| 4 | 11 | settable | FLASH | Flash global interrupt |
| 5 | 12 | settable | RCC | RCC global interrupt |
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt |
| 8 | 15 | settable | EXTI2 | EXTI Line2 interrupt |
| 9 | 16 | settable | EXTI3 | EXTI Line3 interrupt |
| 10 | 17 | settable | EXTI4 | EXTI Line4 interrupt |
| 11 | 18 | settable | DMA1_Channel1 | DMA1 Channel1 global interrupt |
| 12 | 19 | settable | DMA1_Channel2 | DMA1 Channel2 global interrupt |
| 13 | 20 | settable | DMA1_Channel3 | DMA1 Channel3 global interrupt |
| 14 | 21 | settable | DMA1_Channel4 | DMA1 Channel4 global interrupt |
| 15 | 22 | settable | DMA1_Channel5 | DMA1 Channel5 global interrupt |
| 16 | 23 | settable | DMA1_Channel6 | DMA1 Channel6 global interrupt |

# Question



- Q: Can an interrupt interrupt itself?


Contents from Text book

# Pending

# Pending



- When an interrupt input is asserted, it will be *pended*, meaning it is put into a state of waiting for the processor to process the request.
- Even if the interrupt source deserts the interrupt, the pended interrupt status will still cause the interrupt handler to be executed when the priority is allowed.
- Once the interrupt handler is started, the pending status is cleared automatically, as shown in Figure 7.9

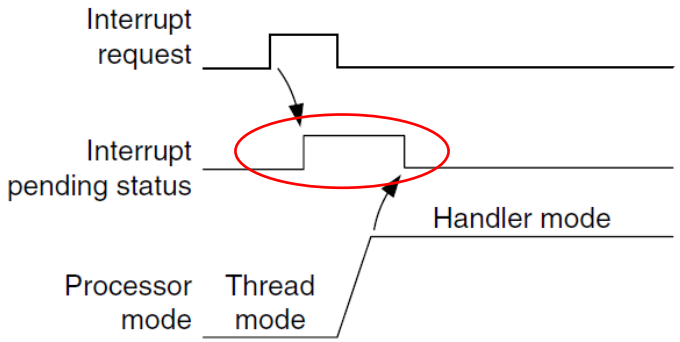



**FIGURE 7.9**

Interrupt Pending.

Contents from Text book

# Pending Cleared Before Processor Takes Action



- Q: What if the interrupt really wants to withdraw the request?
- The pending status can be cleared by software writing to NVIC interrupt control registers to cancel the interrupt (if the interrupt was not taken immediately, e.g. because PRIMASK/FAULTMASK is set to 1.) (Figure 7.10).
- The pending status of the interrupt can be accessed in the NVIC and is writable, so you can clear a pending interrupt or use software to pend a new interrupt by setting the pending register.

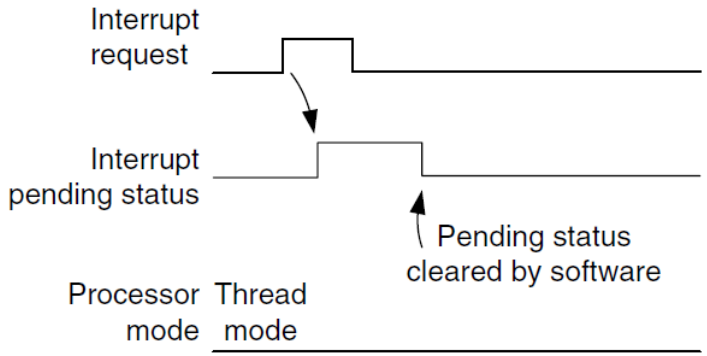



**FIGURE 7.10**

Interrupt Pending Cleared Before Processor Takes Action.

Contents from Text book

# Interrupt – Set Pending and Clea Pending

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

**Table 8.2** Interrupt Set-Pending Registers and Interrupt Clear-Pending Registers (0xE000E200-0xE000E21C, 0xE000E280-0xE000E29C)

| Address | Name | Type | Reset Value | Description |
|---|---|---|---|---|
| 0xE000E200 | SETPEND0 | R/W | 0 | Pending for external Interrupt #0–31<br>bit[0] for Interrupt #0 (exception #16)<br>bit[1] for Interrupt #1 (exception #17)<br>…<br>bit[31] for Interrupt #31 (exception #47)<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status |
| 0xE000E204 | SETPEND1 | R/W | 0 | Pending for external Interrupt #32–63<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status |
| 0xE000E208 | SETPEND2 | R/W | 0 | Pending for external Interrupt #64–95<br>Write 1 to set bit to 1; write 0 has no effect<br>Read value indicates the current status |
| … | — | — | — | — |
| 0xE000E280 | CLRPEND0 | R/W | 0 | Clear pending for external Interrupt #0–31<br>bit[0] for Interrupt #0 (exception #16)<br>bit[1] for Interrupt #1 (exception #17)<br>…<br>bit[31] for Interrupt #31 (exception #47)<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current pending status |
| 0xE000E284 | CLRPEND1 | R/W | 0 | Clear pending for external Interrupt #32–63<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current pending status |
| 0xE000E288 | CLRPEND2 | R/W | 0 | Clear pending for external Interrupt #64–95<br>Write 1 to clear bit to 0; write 0 has no effect<br>Read value indicates the current pending status |
| … | — | — | — | — |

Contents from Text book

# Set Pending and Clear Pending



Table 6-1 shows the NVIC registers.

**Table 6-1 NVIC registers**

| Address | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0xE000E004 | ICTR | RO | - | *Interrupt Controller Type Register, ICTR* |
| 0xE000E100 - 0xE000E11C | NVIC_ISER0 - NVIC_ISER7 | RW | 0x00000000 | Interrupt Set-Enable Registers |
| 0xE000E180 - 0E000xE19C | NVIC_ICER0 - NVIC_ICER7 | RW | 0x00000000 | Interrupt Clear-Enable Registers |
| 0xE000E200 - 0xE000E21C | NVIC_ISPR0 - NVIC_ISPR7 | RW | 0x00000000 | Interrupt Set-Pending Registers |
| 0xE000E280 - 0xE000E29C | NVIC_ICPR0 - NVIC_ICPR7 | RW | 0x00000000 | Interrupt Clear-Pending Registers |
| 0xE000E300 - 0xE000E31C | NVIC_IABR0 - NVIC_IABR7 | RO | 0x00000000 | Interrupt Active Bit Register |
| 0xE000E400 - 0xE000E4EC | NVIC_IPR0 - NVIC_IPR59 | RW | 0x00000000 | Interrupt Priority Register |

Contents from DDI0337

# Exception Pending Status



- When the processor starts to execute an interrupt, the interrupt becomes active and the pending bit will be cleared automatically (Figure 7.11).
- When an interrupt is active, you cannot start processing the same interrupt again, until the interrupt service routine is terminated with an interrupt return (also called an exception exit).
- Then the active status is cleared, and the interrupt can be processed again if the pending status is 1. It is possible to resend an interrupt before the end of the interrupt service routine.

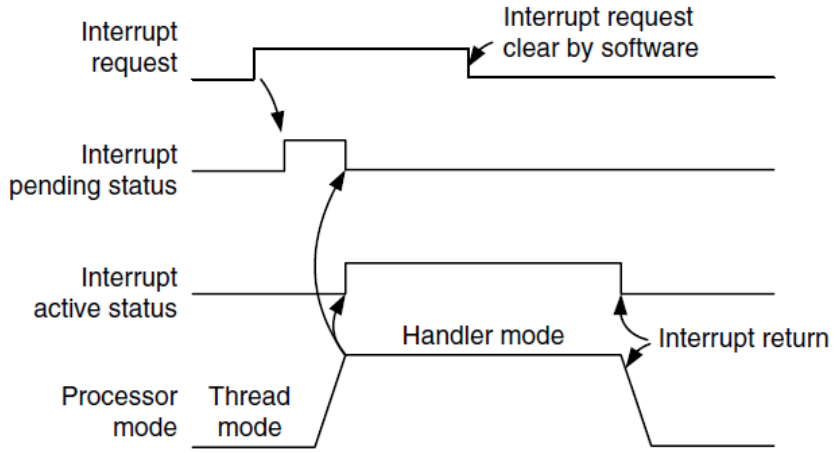



**FIGURE 7.11**

Interrupt Active Status Set as Processor Enters Handler.

Contents from Text book

# Continuous Interrupt Request



- Q: What if an interrupt source continues to hold the interrupt request signal active?
- The interrupt will be pended again at the end of the interrupt service routine, as shown in Figure 7.12.

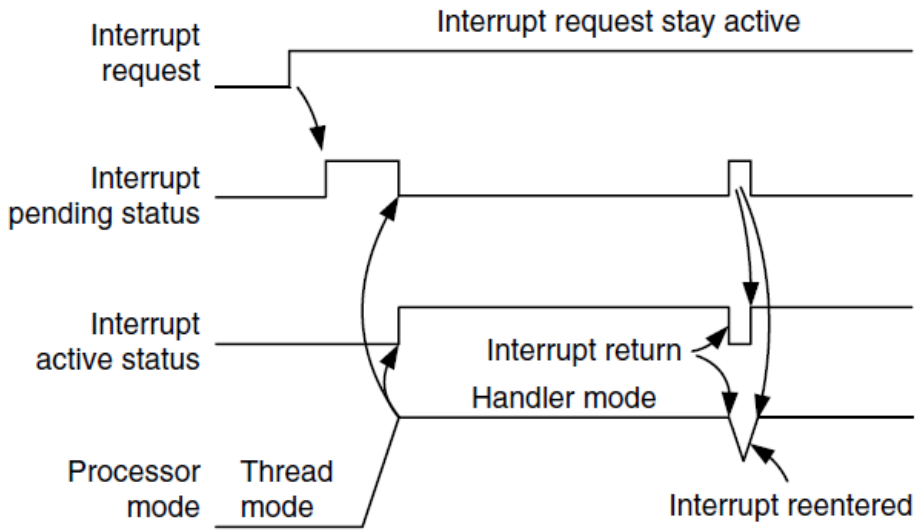



**FIGURE 7.12**

Continuous Interrupt Request Pends Again After Interrupt Exit.

Contents from Text book

# Multiple Interrupt Pulses before Entering ISR



- Q: What if an interrupt is pulsed several times before the processor starts processing it?
- The interrupt will be treated as one single interrupt request as illustrated in Figure 7.13.

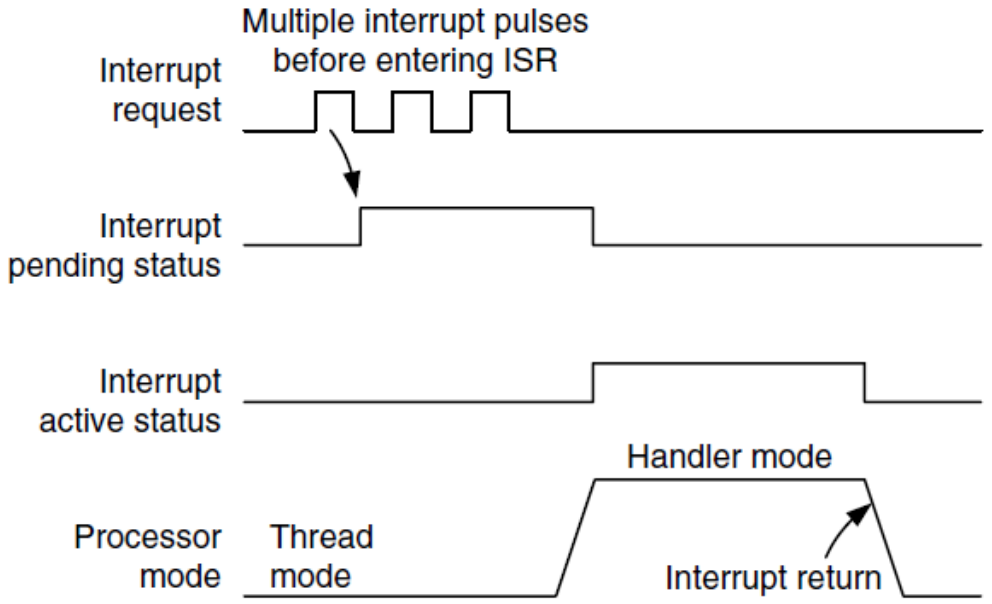



**FIGURE 7.13**

Interrupt Pending Only Once, Even with Multiple Pulses Before the Handler.

Contents from Text book

# Interrupt Pending Occurs During The Handler



- Q: What if an interrupt is deserted and then pulsed again during the interrupt service routine?
- The interrupt will be pended again, as shown in Figure 7.14.

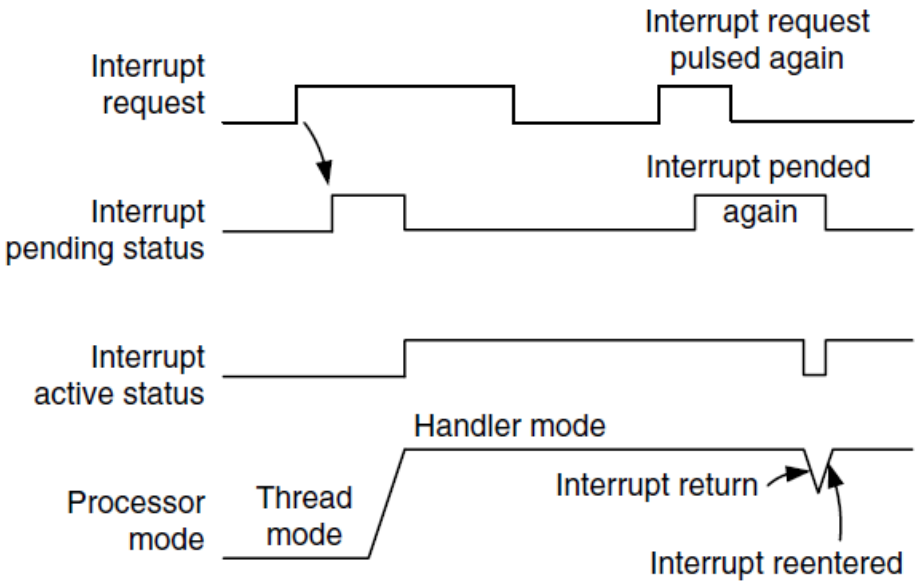



**FIGURE 7.14**

Interrupt Pending Occurs Again during the Handler.

Contents from Text book

# Adaptive change of exception priority in the middle of program

# Three registers for adaptive configuration



- Three registers related to exception priority: BASEPRI, PRIMASK and FAULTMASK.

**Table 3.2** Cortex-M3 Interrupt Mask Registers

| Register Name | Description |
|---|---|
| PRIMASK | A 1-bit register, when this is set, it allows nonmaskable interrupt (NMI) and the hard fault exception; all other interrupts and exceptions are masked. The default value is 0, which means that no masking is set. |
| FAULTMASK | A 1-bit register, when this is set, it allows only the NMI, and all interrupts and fault handling exceptions are disabled. The default value is 0, which means that no masking is set. |
| BASEPRI | A register of up to 8 bits (depending on the bit width implemented for priority level). It defines the masking priority level. When this is set, it disables all interrupts of the same or lower level (larger priority value). Higher priority interrupts can still be allowed. If this is set to 0, the masking function is disabled (this is the default). |

Contents from Text book

## PRIMASK

- The register is used to disable all exceptions except NMI and hard fault.
- It effectively changes the current priority level to 0 (highest programmable level).
- In C programming, you can use the intrinsic functions provided in CMSIS compliant device driver libraries or provided in the compiler to set and clear PRIMASK.
- PRIMASK is useful for temporarily disabling all interrupts for critical tasks. When PRIMASK is set, if a fault takes place, the hard fault handler will be executed.

Contents from Text book

## FAULTMASK

- It is just like PRIMASK except that it changes the effective current priority level to -1, so that even the hard fault handler is blocked.
- Only the NMI can be executed when FAULTMASK is set.
- It can be used by fault handlers to raise its priority to -1, so that they can have access to some features for hard fault exception (more information on this is provided in Chapter 12).

Contents from Text book

# FAULTMASK



- In C programming with CMSIS compliant driver libraries, you can use the intrinsic functions provided in device driver libraries to set and clear FAULTMASK.
- You can also access the FAULTMASK register using MRS and MSR instructions.
- FAULTMASK is cleared automatically upon exiting the exception handler except return from NMI handler.
- Both FAULTMASK and PRIMASK registers cannot be set in the user state.

## BASEPRI

- This register is used if you want to disable only the interrupts with priority lower than a certain level.
- To do this, simply write the required masking priority level to the BASEPRI register.
- For example, if you want to block all exceptions with priority level equal to or lower than 0x60, you can write the value (0x60) to BASEPRI.

```
MOV   R0, #0x60
MSR   BASEPRI, R0 ; Disable interrupts with priority
                  ; 0x60-0xFF
```

Contents from Text book

# BASEPRI_MAX



- The BASEPRI register can also be accessed using the BASEPRI_MAX register name. It is actually the same register, but with BASEPRI_MAX, it gives you a conditional write operation.
- When BASEPRI_MAX is used, the processor hardware automatically compares the current and the new value and only allows the update if it is to be changed to a higher priority level; it cannot be changed to lower priority levels.
- As far as hardware is concerned, BASEPRI and BASEPRI_MAX are the same register, but in the assembler code they use different register name coding.
- Q: What is the highest BASEPRI that can be set?


Contents from Text book

# End