Interrupt-Driven Input/Output on the STM32F407 Microcontroller

Textbook: Chapter 11 (Interrupts)

ARM Cortex-M4 User Guide (Interrupts, exceptions, NVIC)

- Sections 2.1.4, 2.3 – Exceptions and interrupts
- Section 4.2 – Nested Vectored Interrupt Controller

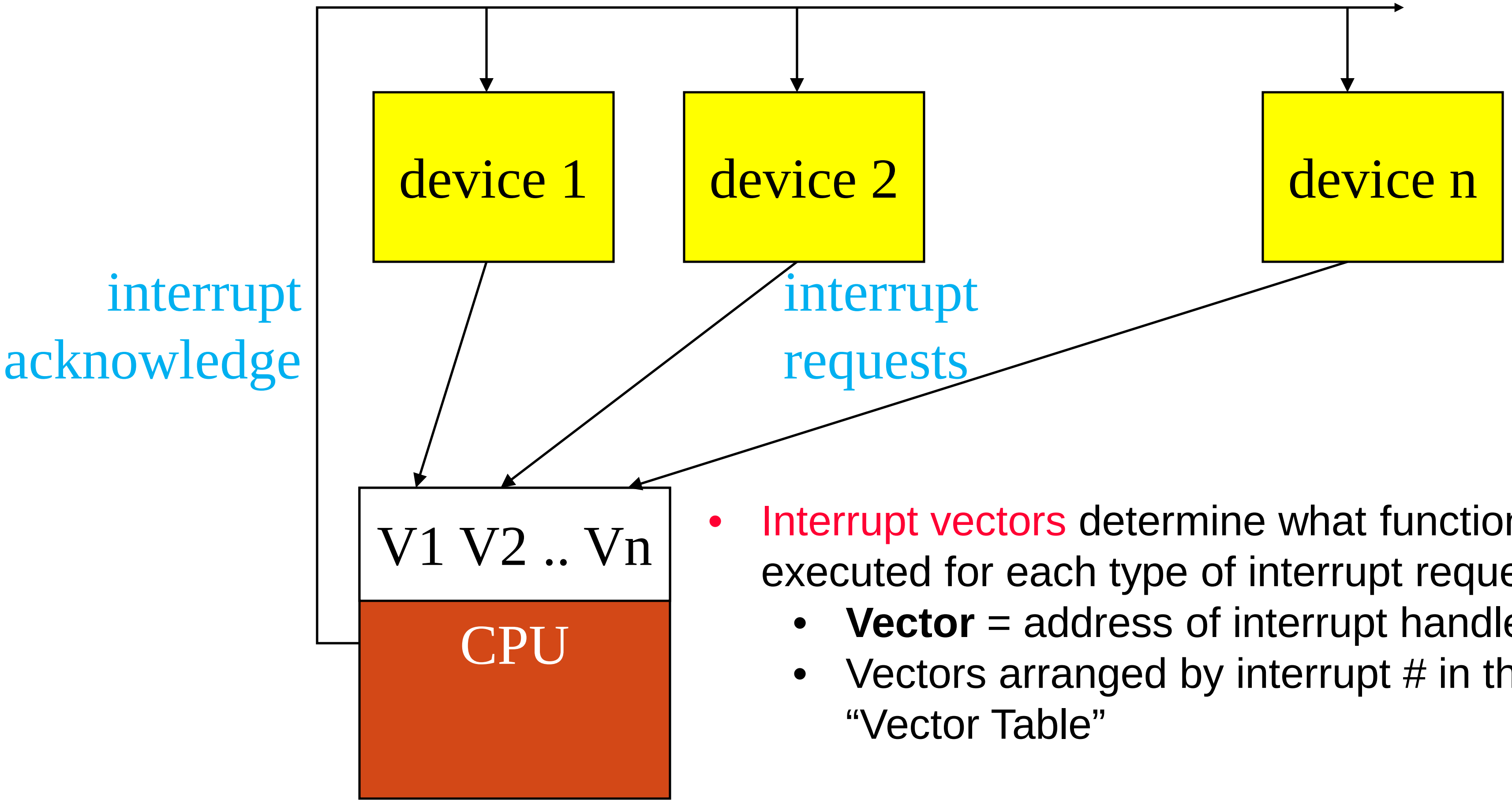
STM32F4xx Tech. Ref. Manual:

- Chapter 8: External interrupt/wakeup lines
- Chapter 9: SYSCFG external interrupt config. registers

Outline

- Interrupt vectors and vector table
- Interrupt masks and priorities
- Cortex Nested Vectored Interrupt Controller (NVIC)
- STM32F4 external interrupt signals (EXTI0 – EXTI15)
- System design when interrupts used

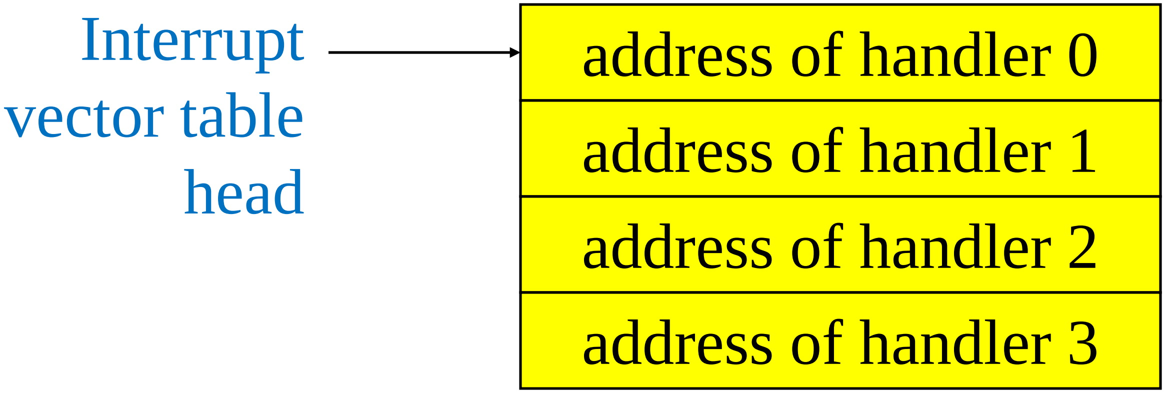
Prioritized, vectored interrupts



- Interrupt vectors determine what function is executed for each type of interrupt request.
  - **Vector** = address of interrupt handler
  - Vectors arranged by interrupt # in the "Vector Table"
- Priorities determine what interrupt gets the CPU first.

Interrupt vectors

- Interrupt vector = **address** of handler function
  - Allow different devices to be handled by different code.
- Interrupt vector table:
  - Directly supported by CPU architecture and/or
  - Supported by a separate interrupt-support device/function



Interrupt vector table head

address of handler 0
address of handler 1
address of handler 2
address of handler 3

Cortex-M CPU and peripheral exceptions

	Priority[1]	IRQ#[2]	Notes
Reset	-3/fixed		Power-up or warm reset
NMI	-2/fixed	-14	Non-maskable interrupt from peripheral or software
HardFault	-1/fixed	-13	Error during exception processing or no other handler
MemManage	0/settable	-12	Memory protection fault (MPU-detected)
BusFault	1/settable	-11	AHB data/prefetch aborts
UsageFault	2/settable	-10	Instruction execution fault - undefined instruction, illegal unaligned access
SVCcall	3/settable	-5	System service call (SVC) instruction
DebugMonitor	4/settable		Break points/watch points/etc.
PendSV	5/settable	-2	Interrupt-driven request for system service
SysTick	6/settable	-1	System tick timer reaches 0
IRQ0	7/settable	0	Signaled by peripheral or by software request
IRQ1 (etc.)	8/settable	1	Signaled by peripheral or by software request

CPU Exceptions

Vendor peripheral interrupts
IRQ0 .. IRQ44

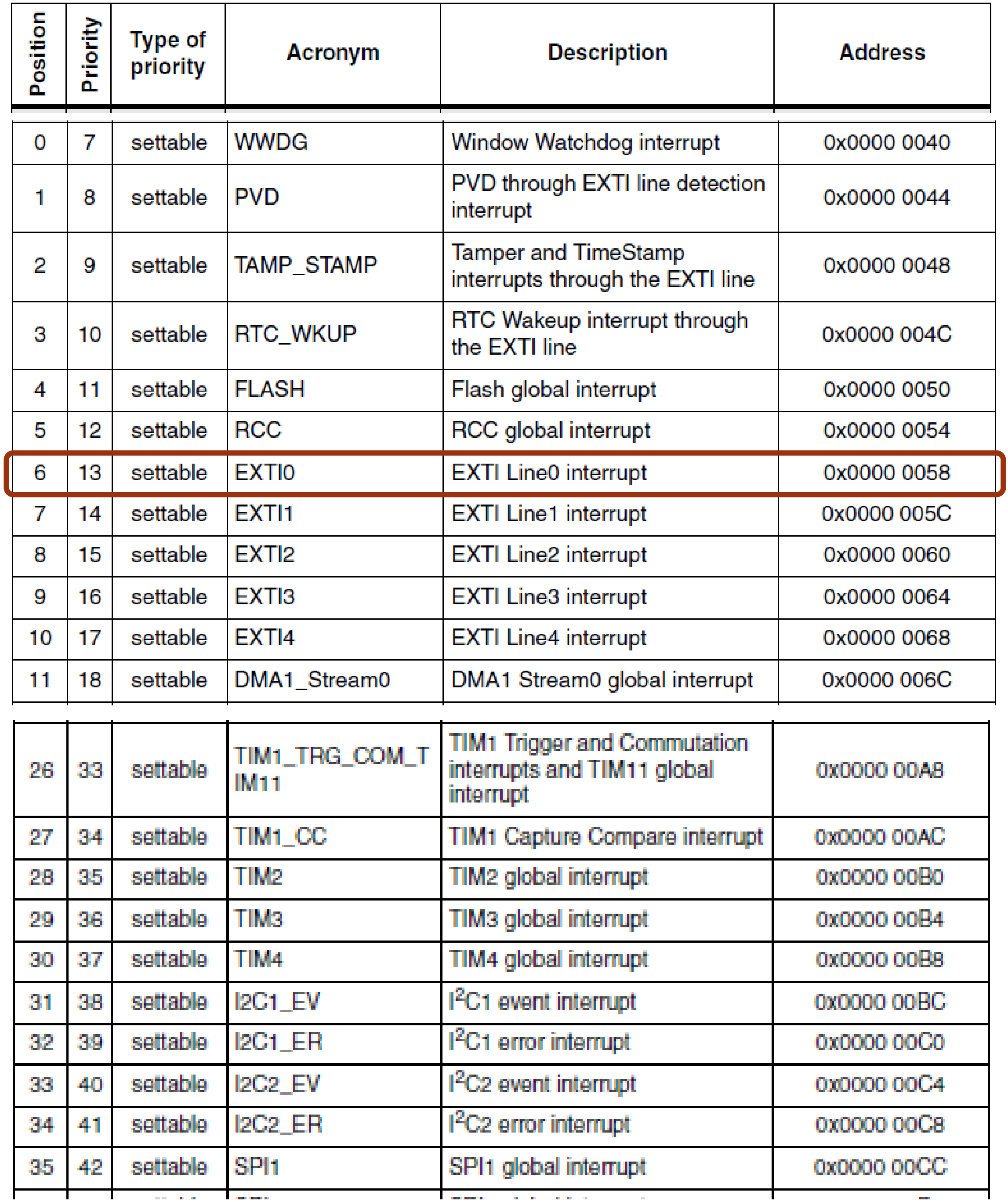
[1] *Lowest priority # = highest priority*

[2] *IRQ# used in CMSIS function calls*

STM32F4xx Peripherals: Interrupt Vector Table

Tech. Ref. Manual: Table 43

Also - refer to startup code



Position	Priority	Type of priority	Acronym	Description	Address
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	PVD	PVD through EXTI line detection interrupt	0x0000 0044
2	9	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	RTC_WKUP	RTC Wakeup interrupt through the EXTI line	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_Stream0	DMA1 Stream0 global interrupt	0x0000 006C
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
31	38	settable	I2C1_EV	I^{2}C1 event interrupt	0x0000 00BC
32	39	settable	I2C1_ER	I^{2}C1 error interrupt	0x0000 00C0
33	40	settable	I2C2_EV	I^{2}C2 event interrupt	0x0000 00C4
34	41	settable	I2C2_ER	I^{2}C2 error interrupt	0x0000 00C8
35	42	settable	SPI1	SPI1 global interrupt	0x0000 00CC

External interrupts

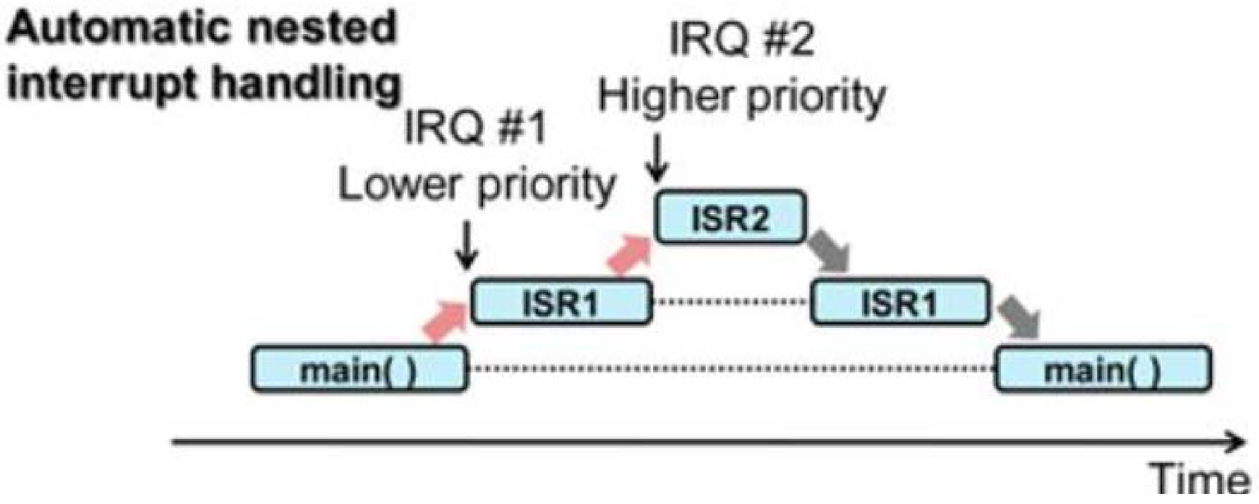
Timer interrupts

STM32F4 vector table from startup code *(partial)*

```
__Vectors
        DCD     __initial_sp                    ; Top of Stack
        DCD     Reset_Handler                   ; Reset Handler
        DCD     NMI_Handler                     ; NMI Handler
            ……
        DCD     SVC_Handler                     ; SVCall Handler
        DCD     DebugMon_Handler                ; Debug Monitor Handler
        DCD     0                               ; Reserved
        DCD     PendSV_Handler                  ; PendSV Handler
        DCD     SysTick_Handler                 ; SysTick Handler
        ; External Interrupts
        DCD     WWDG_IRQHandler                   ; Window WatchDog
        DCD     PVD_IRQHandler                    ; PVD via EXTI Line detection
        DCD     TAMP_STAMP_IRQHandler             ; Tamper/TimeStamps via EXTI
        DCD     RTC_WKUP_IRQHandler               ; RTC Wakeup via EXTI line
        DCD     FLASH_IRQHandler                  ; FLASH
        DCD     RCC_IRQHandler                    ; RCC
        DCD     EXTI0_IRQHandler                  ; EXTI Line0
        DCD     EXTI1_IRQHandler                  ; EXTI Line1
        DCD     EXTI2_IRQHandler                  ; EXTI Line2
        DCD     EXTI3_IRQHandler                  ; EXTI Line3
```

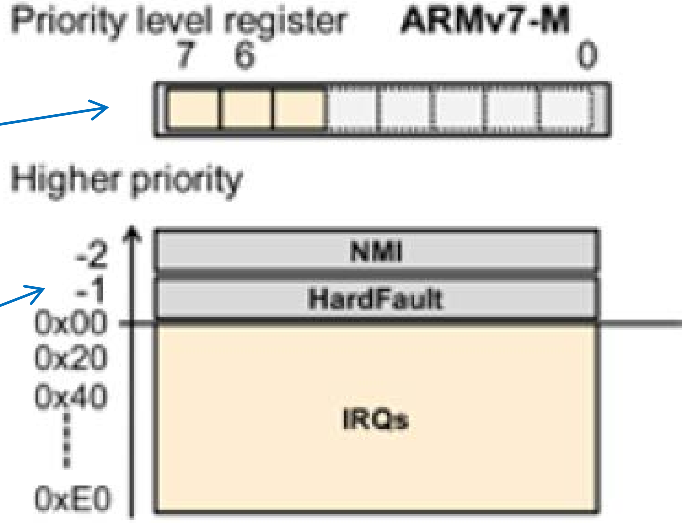
Use these names for interrupt handler functions

Prioritized interrupts



- Up to 256 priority levels
  - 8-bit priority value
  - Implementations may use fewer bits

    STM32F4xx uses upper 4 bits of each priority byte => 16 levels
  - NMI & HardFault priorities are fixed
  - Lowest # = Highest priority



Special CPU registers

Special Cortex-M Assembly Language Instructions

CPSIE I ;Change Processor State/Enable Interrupts (sets PRIMASK = 0)

CPSID I ;Change Processor State/Disable Interrupts (sets PRIMASK = 1)

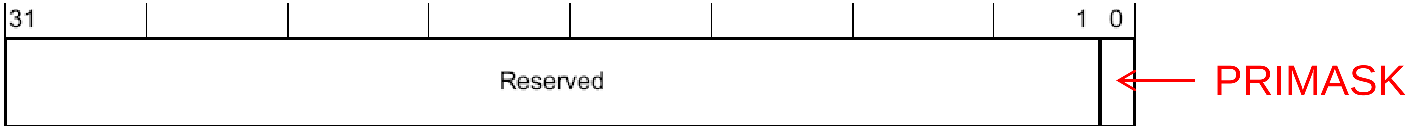
CMSIS[1] C functions to clear/set PRIMASK

__enable_irq(); //enable interrupts (set PRIMASK=0)

__disable_irq(); //disable interrupts (set PRIMASK=1)

(double-underscore at beginning)

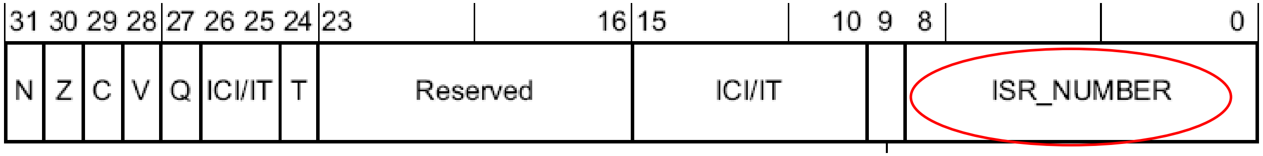
Prioritized Interrupts Mask Register (PRIMASK)



PRIMASK = 1 prevents (masks) activation of all exceptions with configurable priority

PRIMASK = 0 permits (enables/unmasks) exceptions

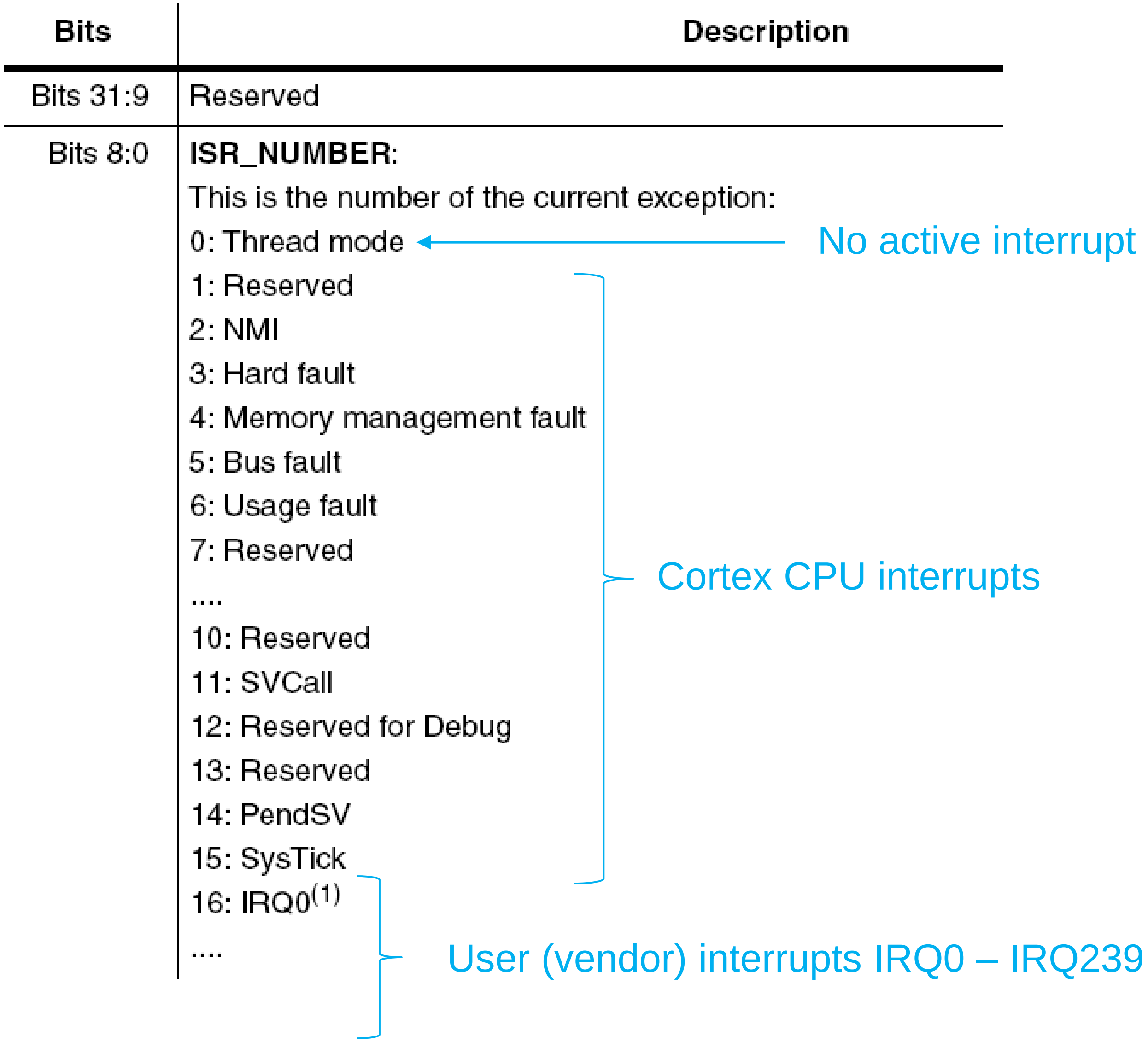
Processor Status Register (PSR)



# of current exception
(lower priority cannot interrupt)

[1] **Cortex Microcontroller Software Interface Standard** – Functions for all ARM Cortex-M CPUs, defined in project header files: *core_cmFunc.h, core_cm3.h*

Interrupt Program Status Register (ISPR)



Bits	Description
Bits 31:9	Reserved
Bits 8:0	**ISR_NUMBER**: This is the number of the current exception: 0: Thread mode 1: Reserved 2: NMI 3: Hard fault 4: Memory management fault 5: Bus fault 6: Usage fault 7: Reserved .... 10: Reserved 11: SVCall 12: Reserved for Debug 13: Reserved 14: PendSV 15: SysTick 16: IRQ0(1) ....

No active interrupt

Cortex CPU interrupts

User (vendor) interrupts IRQ0 – IRQ239

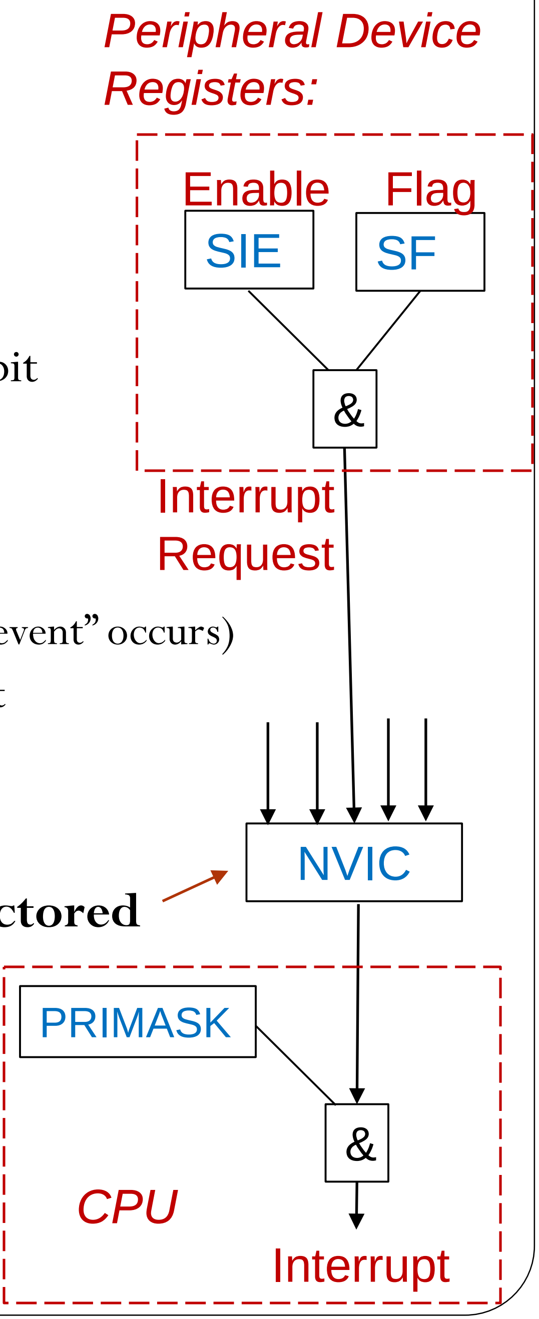
ARM Cortex-M Interrupts

In the Device:

- Each potential interrupt source has a separate **arm** **(enable)** bit
  - Set for those devices from which interrupts, are to be accepted
  - Deactivate in those devices from which interrupts are not allowed
- Each potential interrupt source has a separate **flag** bit
  - hardware sets the flag when it wishes to request an interrupt (an "event" occurs)
  - software clears the flag in ISR to signify it is processing the request
  - flags can be tested by software if interrupts not desired

In the CPU:

- Cortex-M CPUs receive interrupt requests via the **Nested Vectored Interrupt Controller (NVIC)**
  - NVIC sends highest priority request to the CPU
- Interrupt **enable** conditions in processor
  - Global interrupt enable bit, I, in PRIMASK register
  - Priority level, BASEPRI, of allowed interrupts (0 = all)

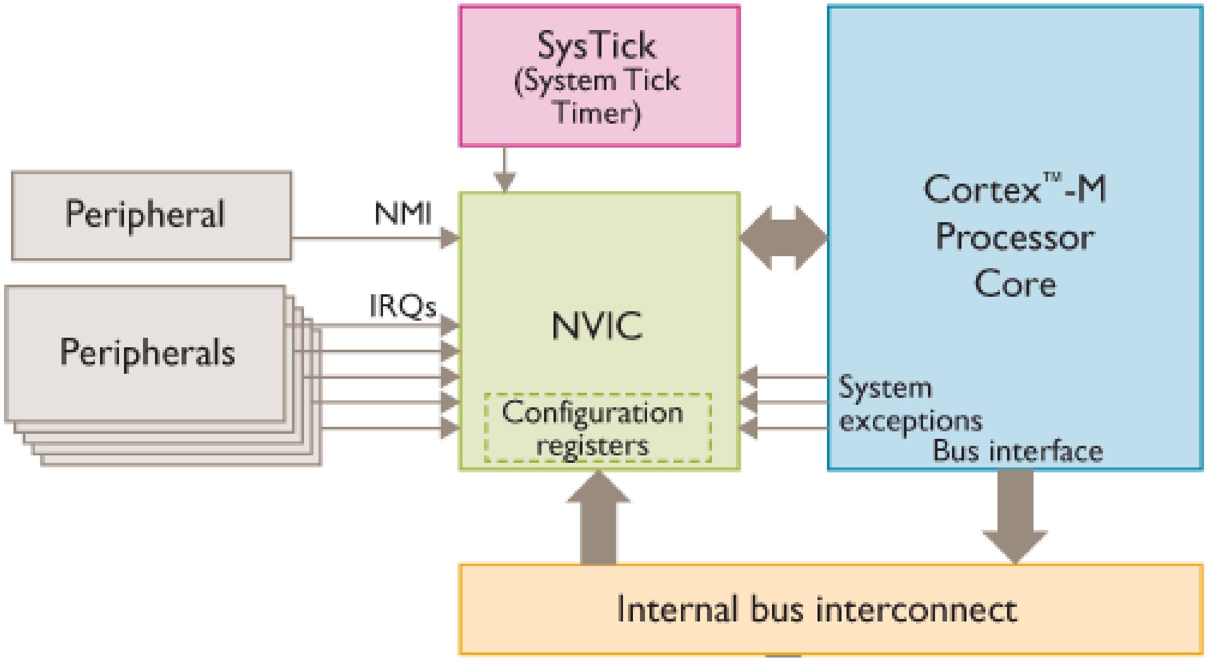


Interrupt Conditions

- Four conditions must be true simultaneously for an interrupt to occur:
  1. Trigger: hardware action sets source-specific flag in the peripheral device
  2. Arm: control bit for each possible source is set within the peripheral device
  3. Level: interrupt level must be less than BASEPRI (base priority)
  4. Enable: interrupts globally enabled in CPU (I=0 in PRIMASK)
- Interrupt remains **pending** if trigger is set but any other condition is not true
  - Interrupt serviced once all conditions become true
- Need to **acknowledge** interrupt
  - Clear trigger flag to prevent endless interrupts!

Nested Vectored Interrupt Controller

- NVIC manages and prioritizes external interrupts in Cortex-M
  - 82 IRQ sources from STM32F4xx peripherals
- NVIC interrupts CPU with IRQ# of highest-priority IRQ signal
  - CPU uses IRQ# to access the vector table & get intr. handler start address



Nested Vectored Interrupt Controller (NVIC)

- Hardware unit that coordinates interrupts from multiple sources
  - Separate enable flag for each interrupt source
    - Set/clear via **NVIC_ISERx/ICERx** registers
  - Separate priority level for each interrupt source
    - Define in **NVIC_IPRx** registers
  - Set/clear interrupts to/from pending state
    - Pending state entered when interrupt request detected
    - Pending state cleared automatically when ISR complete, unless subsequent interrupt request detected while in the ISR
    - Manually set/clear pending state via **NVIC_ISPRx/ICPRx** registers
    - Can also trigger interrupts through software if desired

- Higher priority interrupts can interrupt lower priority ones
  - Lower priority interrupts are not sent to the CPU until higher priority interrupt service has been completed

Nested Vectored Interrupt Controller (NVIC)

- Each interrupt source is in one of four states
  - Inactive – no interrupt service requested or in progress
  - Pending – interrupt request latched by NVIC; not yet serviced by CPU
    - Become pending if interrupt signal HIGH and interrupt not active
    - Also become pending if rising edge detected on interrupt signal
  - Active – interrupt service by CPU is in progress
    - State changes from Pending to Active when CPU enters the ISR
    - State changes from Active to Inactive when CPU exits the ISR, unless:
    - State changes from Active to Pending if interrupt signal still HIGH when CPU exits the ISR or if state is "Pending and Active" (can re-enter the ISR)
  - Pending and Active – new interrupt request detected (rising edge or pulse) while CPU is servicing a previous request (IRQ can be re-entered)

NVIC Interrupt Enable Registers

- Three "set interrupt enable" registers – **`NVIC_ISER0 , NVIC_ISER1 , NVIC_ISER2`**
  - One "enable" bit per IRQ - with 32 per register
  - Write 1 to a bit to set the corresponding interrupt enable bit
  - Writing 0 has no effect

Registers	IRQ numbers	Interrupt numbers
NVIC_ISER0/NVIC_ICER0	0-31	16-47
NVIC_ISER1/NVIC_ICER1	32-63	48-79
NVIC_ISER2/NVIC_ICER2	64-95	80-111

- Three corresponding "clear interrupt enable" registers **`NVIC_ICER0 , NVIC_ICER1 , NVIC_ICER2`**
  - Write 1 to clear the interrupt enable bit (disable the interrupt)
  - Writing 0 has no effect

NVIC Set-Clear Pending Registers

- Three interrupt "clear-pending" registers – **`NVIC_ICPR0, NVIC_ICPR1, NVIC_ICPR2`**
  - One "pending" flag for each IRQ (32 in each register)
  - Write 1 to a bit to remove a pending state from an interrupt
  - Writing 0 has no effect
- Pending state normally cleared automatically when the processor enters the interrupt handler (ISR)
  - If interrupt signal still active when CPU returns from the ISR, the state changes to pending again (new interrupt triggered)
  - If interrupt signal pulses while CPU is in the ISR, the state changes to pending again (new interrupt triggered)
- Three corresponding interrupt "set-pending" registers

  `NVIC_ISPR0 , NVIC_ISPR1, NVIC_ISPR2`
  - Write 1 to force the interrupt state to pending
  - Writing 0 has no effect

NVIC interrupt priority registers

Interrupt Priority Registers NVIC_IPRx (x=0..20)

- 8-bit priority field for each interrupts (4-bit field in STM32F4)
  - Four 8-bit priority values per register

byte 3	byte 2	byte 1	byte 0

  - STMicroelectronics uses upper 4 bits of each byte
  - 0 = highest priority level
  - IPR Register# x = IRQ# DIV 4
  - Byte offset within the IPR register = IRQ# MOD 4

Example: IRQ45

- 45/4 = 11 with remainder 1 (register NVIC_IPR11, byte offset 1)

  Write priority<<8 to NVIC_IPR11
- 45/32 = 1 with remainder 13:

  Write 1<<13 to NVIC_ISER1 to enable the interrupt

NVIC register addresses

NVIC_ISER0/1/2 = 0xE000E100/104/108

NVIC_ICER0/1/2 = 0xE000E180/184/188

NVIC_ISPR0/1/2 = 0xE000E200/204/208

NVIC_ICPR0/1/2 = 0xE000E280/284/288

NVIC_IPR0/1/2/…/20 = 0xE00E400/404/408/40C/…./500

```
;Example – Enable EXTI0 with priority 5 (EXTI0 = IRQ6)
NVIC_ISER0  EQU  0xE000E100         ;bit 6 enables EXTI0
NVIC_IPR1   EQU  0xE000E404         ;3rd byte = EXTI0 priority
        ldr     r0,=NVIC_ISER0
        mov     r1,#0x0040                  ;Set bit 6 of ISER0 for EXTI0
        str     r1,[r0]
        ldr     r0,=NVIC_IPR1               ;IRQ6 priority in IPR1[23:16]
        ldr     r1,[r0]                     ;Read IPR1
        bic     r1,#0x00ff0000              ;Clear [23:16] for IRQ6
        orr     r1,#0x00500000              ;Bits [23:20] = 5
        str     r1,[r0]                     ;Upper 4 bits of byte = priority
```

CMSIS[1] functions

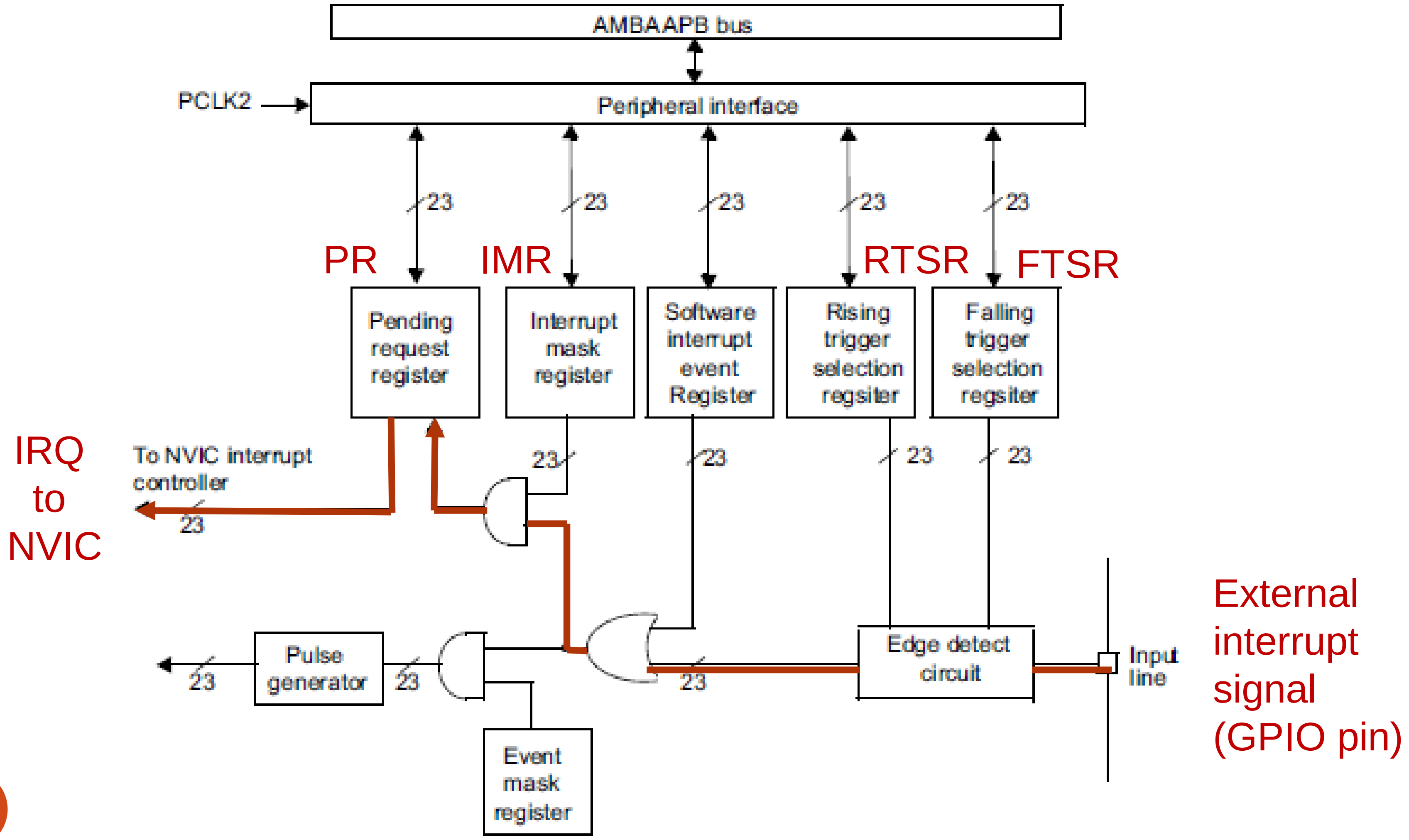
- NVIC_Enable(IRQn_Type IRQn)
- NVIC_Disable(IRQn_Type IRQn)
- NVIC_SetPending(IRQn_Type IRQn)
- NVIC_ClearPending(IRQn_Type IRQn)
- NVIC_GetPending(IRQn_Type IRQn)
- NVIC_SetPriority(IRQn_Type IRQn,unit32_t priority)
- NVIC_GetPriority(IRQn_Type IRQn)

[1] CMSIS = Cortex Microcontroller Software Interface Standard

- Vendor-independent hardware abstraction layer for Cortex-M
- Facilitates software reuse
- Other CMSIS functions: System tick timer, Debug interface, etc.

STM32F4xx external interrupt/event controller

- External devices can interrupt CPU via GPIO pins
  (Some microcontrollers have dedicated interrupt pins)
- Up to 16 external interrupts (EXTI0-EXTI15), plus 7 internal events



STM32F4xx external interrupt sources

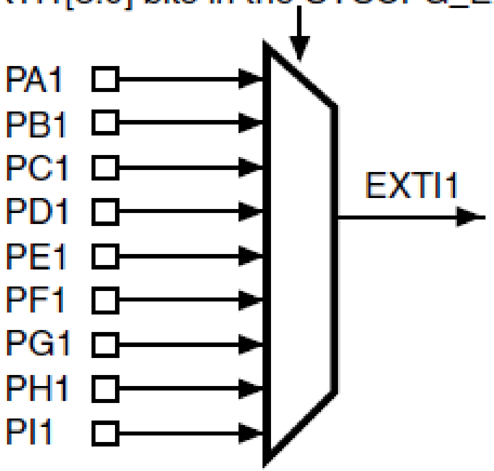
(select in System Configuration Module – SYSCFG)

- 16 multiplexers select GPIO pins as external interrupts EXTI0..EXTI15
- Mux inputs selected via 4-bit fields of EXTICR[k] registers (k=0..3)
  - EXTIx = 0 selects PAx, 1 selects PBx, 2 selects PCx, etc.
  - EXTICR[0] selects EXTI3-EXTI0; EXTICR[1] selects EXTI7-EXTI4, etc

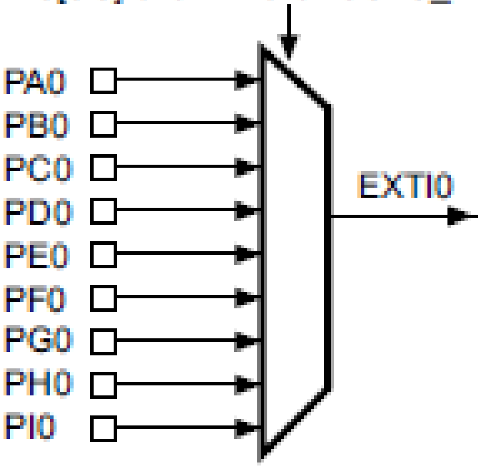
SYSCFG_EXTICR1 is
SYSCFG->EXTICR[0]

15 12	11 8	7 4	3 0
EXTI3	EXTI2	EXTI1	EXTI0

EXTI1[3:0] bits in the SYSCFG_EXTICR1 register



EXTI0[3:0] bits in the SYSCFG_EXTICR1 register



Example: Select pin PC2 as external interrupt EXTI2

```
SYSCFG->EXTICR[0] &= 0xF0FF;   //clear EXTI2 bit field
SYSCFG->EXTICR[0] |=  0x0200;   //set EXTI2 = 2 to select PC2
```

STM32L1xx EXTI configuration registers

- Register bits 15-0 control EXTI15-EXTI0, respectively
- **EXTI_IMR** – interrupt mask register
  - 1 unmasks (enables) the corresponding interrupt
  - 0 masks (disables) the interrupt
- **EXTI_RTSR/FTSR** – rising/falling trigger selection register
  - 1 to enable rising/falling edge to trigger the interrupt/event
  - 0 to ignore the rising/falling edge
- **EXTI_PR** – interrupt pending register
  - bit set to 1 by hardware if interrupt/event occurred *(bit is readable)*
  - clear bit by writing 1 *(writing 0 has no effect)*
  - **interrupt handler** must write **1** to this bit to clear the pending state of the interrupt (to cancel the IRQn request)

Example: Configure EXTI2 as rising-edge triggered

```
EXTI->RTSR |= 0x0004;     //Bit2=1 to make EXTI2 rising-edge trig.
EXTI->IMR   = 0x0004;     //Bit2=1 to enable EXTI2
EXTI->PR   |= 0x0004;     //Bit2=1 to clear EXTI2 pending status
```

Clearing pending status needs to be done in the interrupt handler after every interrupt.

Example: Enable EXTI0 as rising-edge triggered, selecting PC0 as EXTI0.

```
;System Configuration Registers
SYSCFG      EQU       0x40013800
EXTICR1     EQU       0x08
;External Interrupt Registers
EXTI        EQU       0x40013C00
IMR         EQU       0x00        ;Interrupt Mask Register
RTSR        EQU       0x08        ;Rising Trigger Select
FTSR        EQU       0x0C        ;Falling Trigger Select
PR          EQU       0x14        ;Pending Register

     ;select PC0 as EXTI0
     ldr   r1,=SYSCFG            ;SYSCFG selects EXTI sources
     ldrh  r2,[r1,#EXTICR1]      ;EXTICR1 = sources for EXTI0 - EXTI3
     bic   r2,#0x000f            ;Clear EXTICR1[3-0] for EXTI0 source
     orr   r2,#0x0002            ;EXTICR1[3-0] = 2 to select PC0 as EXTI0 source
     strh  r2,[r1,#EXTICR1]      ;Write to select PC0 as EXTI0
     ;configure and enable EXTI0 as rising-edge triggered
     ldr   r1,=EXTI         ;EXTI register block
     mov   r2,#1            ;bit #0 for EXTI0 in each of the following registers
     str   r2,[r1,#RTSR]    ;Select rising-edge trigger for EXTI0
     str   r2,[r1,#PR]      ;Clear any pending event on EXTI0
     str   r2,[r1,#IMR]     ;Enable EXTI0
```

Interrupt Rituals

- Things you must do in every ritual
  - Initialize data structures (counters, pointers)
  - Arm interrupt in the peripheral device
    - Enable a flag to trigger an interrupt (ex. **IMR** of the EXTI module)
    - Clear the flag (to ignore any previous events) (ex. **PR** of the EXTI module)
  - Configure NVIC
    - Enable interrupt (set bit of **NVIC_ISERx**)
    - Set priority (set bits of **NVIC_IPRx**)
  - Enable CPU Interrupts
    - Assembly code `CPSIE I`
    - C code `EnableInterrupts();`

Project setup for interrupt-driven applications

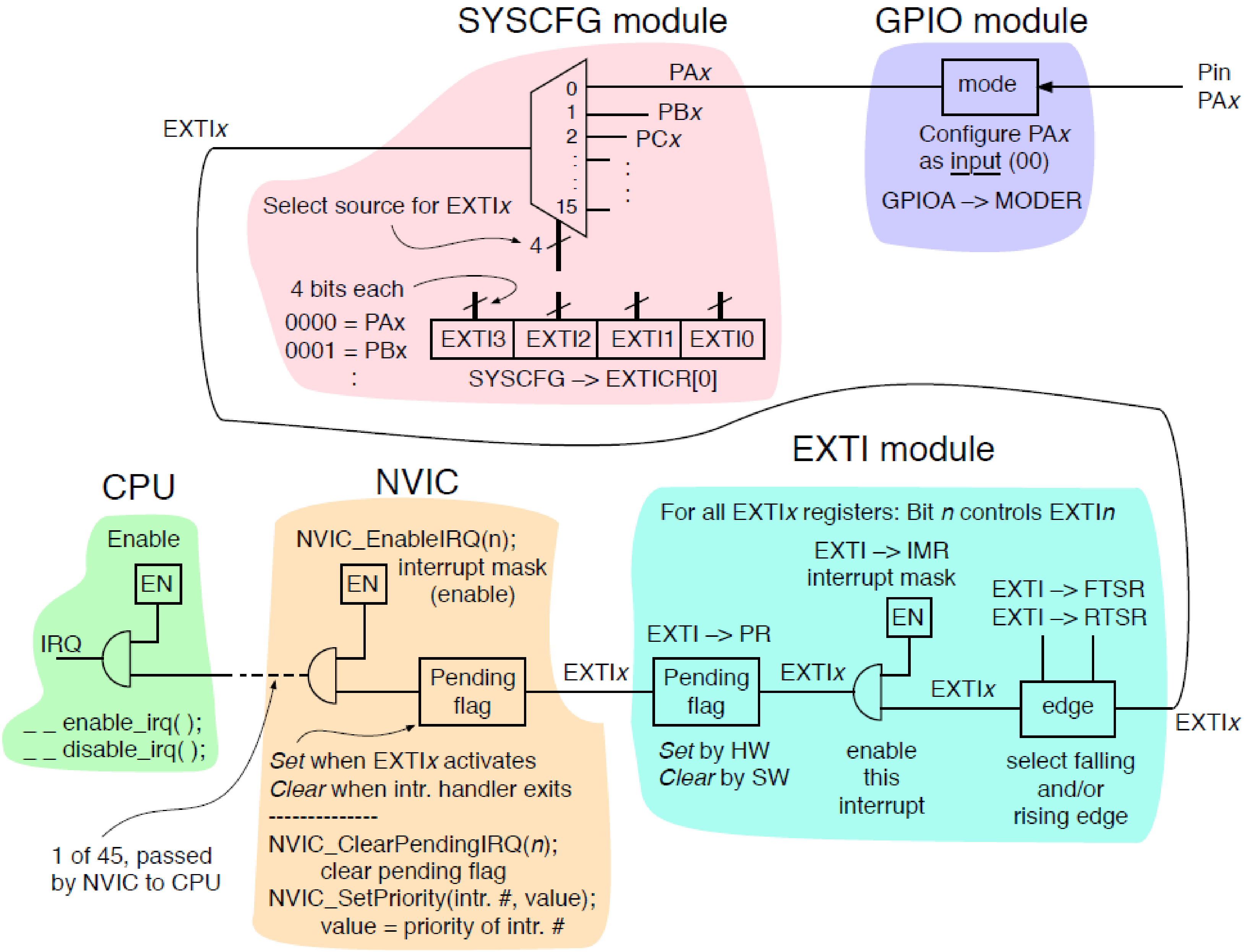
- Write the interrupt handler for each peripheral
  - Clear the flag that requested the interrupt (acknowledge the intr. request)
  - Perform the desired actions, communicating with other functions via shared global variables
  - Use function names from the vector table

    Example: *void EXTI4_IRQHandler () { statements }*
- Perform all initialization for each peripheral device:
  - Initialize the device, "arm" its interrupt, and clear its "flag"

    Example: External interrupt EXTIn
    - Configure GPIO pin as a digital input
    - Select the pin as the EXTIn source (in SYSCFG module)
    - Enable interrupt to be requested when a flag is set by the desired event (rising/falling edge)
    - Clear the pending flag (to ignore any previous events)
  - NVIC
    - Enable interrupt: *NVIC_EnableIRQ (IRQn);*
    - Set priority: *NVIC_SetPriority (IRQn, priority);*
    - Clear pending status: *NVIC_ClearPendingIRQ (IRQn);*
- Initialize counters, pointers, global variables, etc.
- Enable CPU Interrupts: *__enable_irq();*

(diagram on next slide)

Signal flow/setup for External Interrupt EXTI*x*, *x* = 0…15



SYSCFG module

PA*x*
PB*x*
PC*x*
0
1
2
15
EXTI*x*

Select source for EXTIx

4

4 bits each
0000 = PAx
0001 = PBx
:

EXTI3 | EXTI2 | EXTI1 | EXTI0

SYSCFG –> EXTICR[0]

GPIO module

mode

Pin PA*x*

Configure PA*x* as <u>input</u> (00)

GPIOA –> MODER

EXTI module

For all EXTI*x* registers: Bit *n* controls EXTI*n*

EXTI –> IMR
interrupt mask

EN

EXTI –> FTSR
EXTI –> RTSR

EXTI –> PR

Pending flag

EXTI*x*

EXTI*x*

edge

EXTI*x*

Set by HW
Clear by SW

enable this interrupt

select falling and/or rising edge

CPU

Enable

EN

IRQ

_ _enable_irq( );
_ _disable_irq( );

1 of 45, passed by NVIC to CPU

NVIC

NVIC_EnableIRQ(n);

EN

interrupt mask (enable)

Pending flag

EXTI*x*

Set when EXTI*x* activates
Clear when intr. handler exits

NVIC_ClearPendingIRQ(*n*);
clear pending flag
NVIC_SetPriority(intr. #, value);
value = priority of intr. #

Interrupt Service Routine (ISR)

- Interrupt service routine (interrupt handler) name must be in the interrupt vector table (ex. EXTI0_IRQHandler)
- Things you must do in every interrupt service routine
  - Acknowledge
    - clear the flag that requested the interrupt (ex. **PR** of EXTI module)
    - SysTick is exception; automatic acknowledge
  - Save/restore contents of R4-R11, if used. (AAPCS)
  - Perform the requested service
  - Communicate via shared global variables

Sources of interrupt “overhead”

- Handler execution time.
- Interrupt mechanism overhead.
- Register save/restore.
- Pipeline-related penalties (advanced processors).
- Cache-related penalties (advanced processors).
- Interrupt “latency” = time from activation of interrupt signal until event serviced.
- ARM worst-case latency to respond to interrupt is 27 cycles:
    - 2 cycles to synchronize external request.
    - Up to 20 cycles to complete current instruction (worst are LDM/STM).
    - 3 cycles for data abort.
    - 2 cycles to enter interrupt handling state.