Interrupts

Eduardo Barrera

Julián Nieto

Juan Manuel López

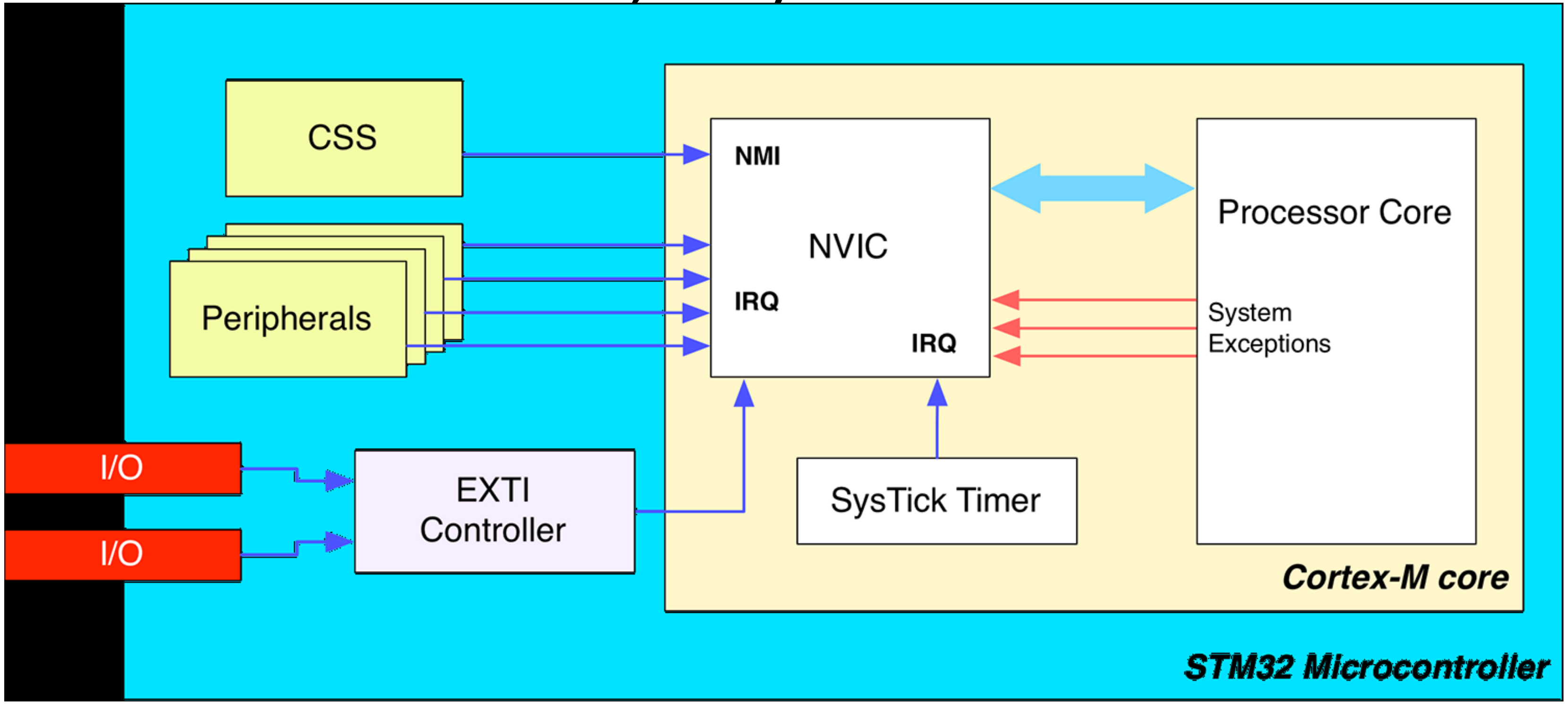
Mariano Ruiz

Interrupts

- Mechanism to handle asynchronous events.
- When an interrupt happens, the microcontroller does several actions:
  - Save the current software context and program counter
  - Jump to the Interrupt Service Routine (ISR)
- When the ISR finishes, the software context is restored and continues with program execution
- The ARM Cortex-M family provides a unit to manage the interrupts. **Nested Vectored Interrupt Controller (NVIC)**

Nested Vectored Interrupt Controller (NVIC)

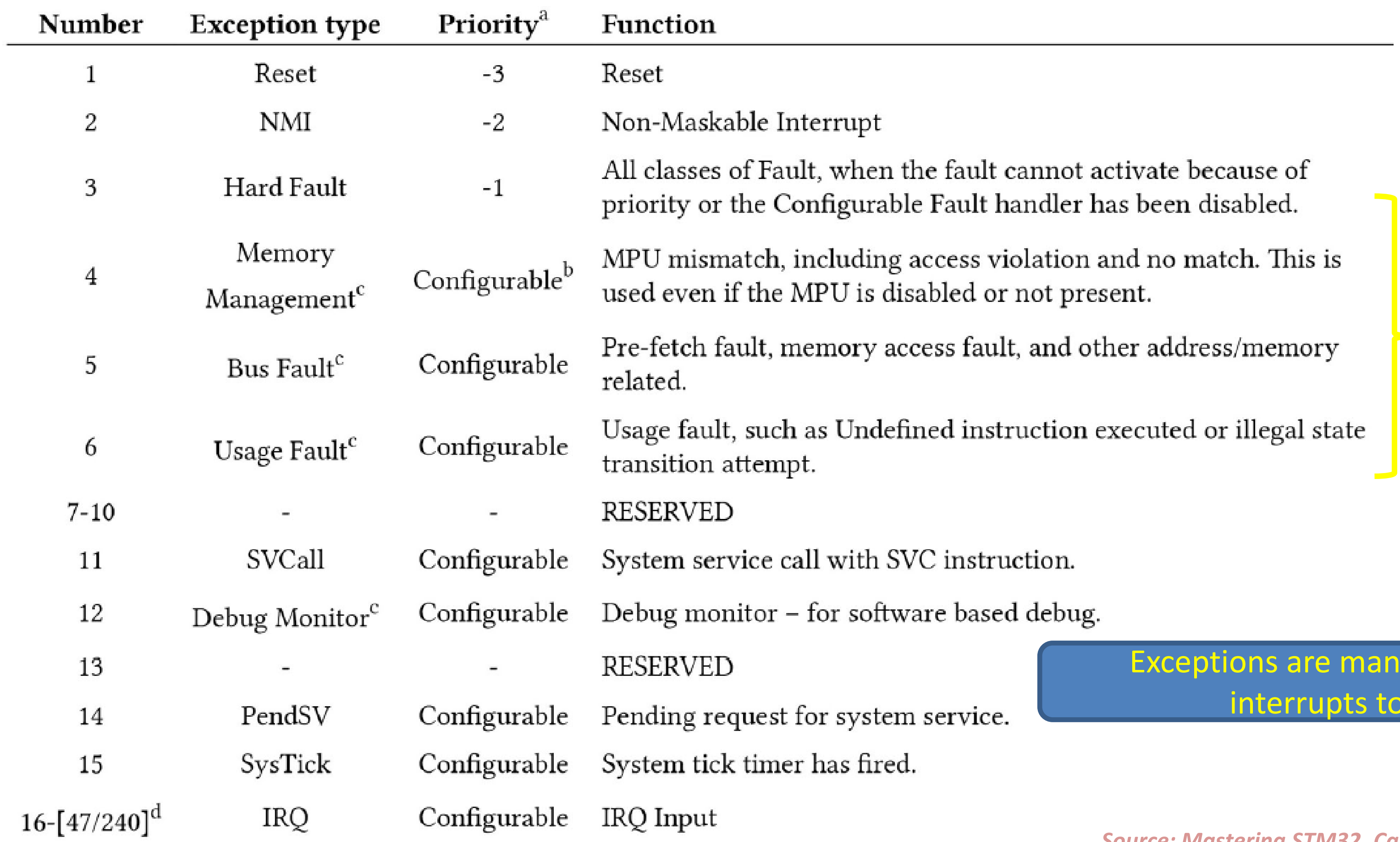
- The NVIC manages the interrupts and the exceptions (from internal peripherals, from external GPIO lines, etc)



Source: Mastering STM32. Carmine Noviello

Nested Vectored Interrupt Controller (NVIC)

- The processor knows where to jump when an event happens using the vector table (ordered by priority)



Number	Exception type	Priority[a]	Function
1	Reset	-3	Reset
2	NMI	-2	Non-Maskable Interrupt
3	Hard Fault	-1	All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled.
4	Memory Management[c]	Configurable[b]	MPU mismatch, including access violation and no match. This is used even if the MPU is disabled or not present.
5	Bus Fault[c]	Configurable	Pre-fetch fault, memory access fault, and other address/memory related.
6	Usage Fault[c]	Configurable	Usage fault, such as Undefined instruction executed or illegal state transition attempt.
7-10	-	-	RESERVED
11	SVCall	Configurable	System service call with SVC instruction.
12	Debug Monitor[c]	Configurable	Debug monitor – for software based debug.
13	-	-	RESERVED
14	PendSV	Configurable	Pending request for system service.
15	SysTick	Configurable	System tick timer has fired.
16-[47/240][d]	IRQ	Configurable	IRQ Input

Exceptions are managed like interrupts too

Source: Mastering STM32. Carmine Noviello

Nested Vectored Interrupt Controller (NVIC)

- The processor knows where to jump when an event happens using the vector table

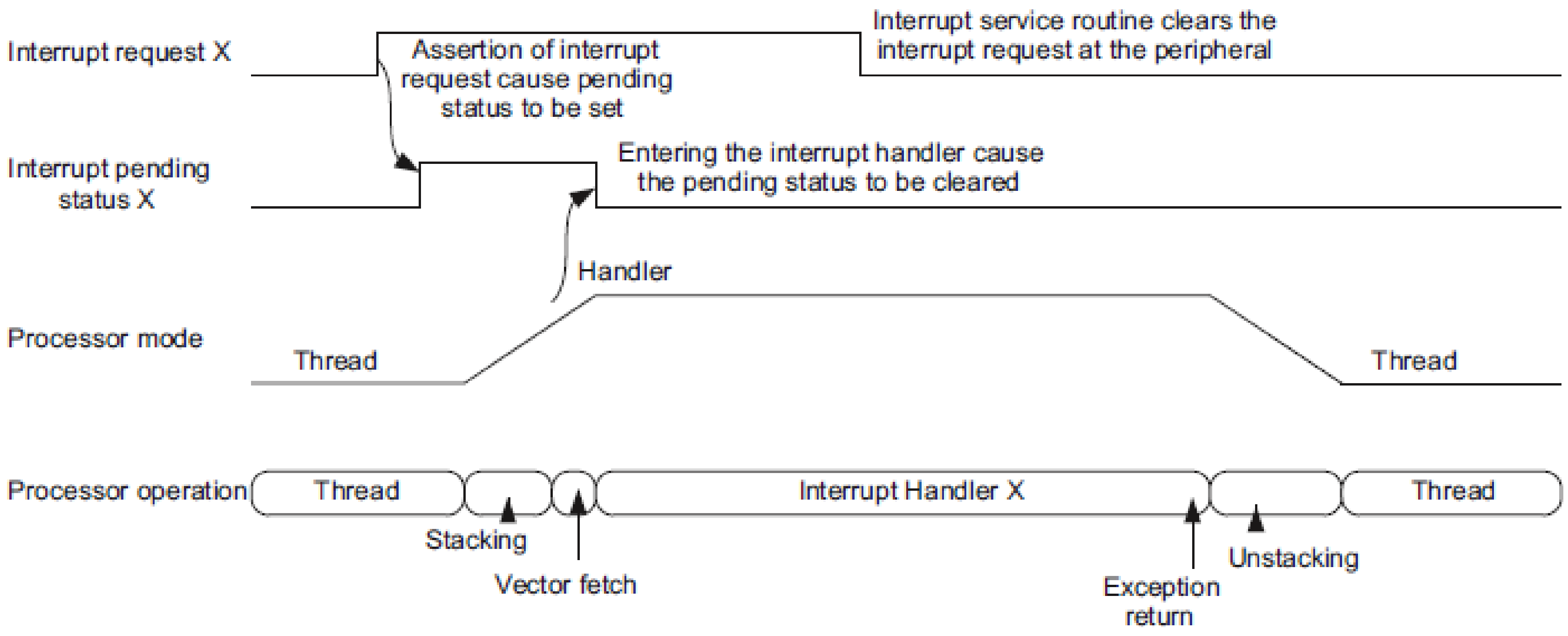
Number	Exception type	Priority[a]	Function
1	Reset	-3	Reset
2	NMI	-2	Non-Maskable Interrupt
3	Hard Fault	-1	All classes of Fault, when the fault cannot activate because of priority or the Configurable Fault handler has been disabled.
4	Memory Management[c]	Configurable[b]	MPU mismatch, including access violation and no match. This is used even if the MPU is disabled or not present.
5	Bus Fault[c]	Configurable	Pre-fetch fa related.
6	Usage Fault[c]	Configurable	Usage fault, transition a
7-10	-	-	RESERVED
11	SVCall	Configurable	System serv
12	Debug Monitor[c]	Configurable	Debug mon
13	-	-	RESERVED
14	PendSV	Configurable	Pending req
15	SysTick	Configurable	System ick
16-[47/240][d]	IRQ	Configurable	IRQ Inp t

startup_stm32f429xx.s file

```
; Vector Table Mapped to Address 0 at Reset
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors
                EXPORT  __Vectors_End
                EXPORT  __Vectors_Size

__Vectors       DCD     __initial_sp               ; Top of Stack
                DCD     Reset_Handler              ; Reset Handler
                DCD     NMI_Handler                ; NMI Handler
                DCD     HardFault_Handler          ; Hard Fault Handler
                DCD     MemManage_Handler          ; MPU Fault Handler
                DCD     BusFault_Handler           ; Bus Fault Handler
                DCD     UsageFault_Handler         ; Usage Fault Handler
                DCD     0                          ; Reserved
                DCD     0                          ; Reserved
                DCD     0                          ; Reserved
                DCD     0                          ; Reserved
                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 through EXTI Line detection
                DCD     TAMP_STAMP_IRQHandler             ; Tamper and TimeStamps through the EXTI line
                DCD     RTC_WKUP_IRQHandler               ; RTC Wakeup through the EXTI line
```

Interrupt Lifecycle

- Interrupt lifecycle example


Interrupt request X
Assertion of interrupt request cause pending status to be set
Interrupt service routine clears the interrupt request at the peripheral
Interrupt pending status X
Entering the interrupt handler cause the pending status to be cleared
Handler
Processor mode
Thread
Thread
Processor operation
Thread
Interrupt Handler X
Thread
Stacking
Vector fetch
Exception return
Unstacking

- The ISR routine must fulfil this criterium:
  - Reduce execution time avoiding loops and intensive operations

Using Interrupts

- After reset, all interrupts are disabled, except Reset, NMI, and Hard Fault
- Enabling an interrupt IRQ using HAL:
  - **void** HAL_NVIC_EnableIRQ (IRQn_Type IRQn);

IRQn_Type is an enumerated datatype defined in stm32f429xx.h file

- Disabling an interrupt:
  - **void** HAL_NVIC_DisableIRQ (IRQn_Type IRQn);
- Obviously, the peripheral must be configured to generate interrupts

Using Interrupts

- An ISR must be defined
  - First, inside the ISR the associate pending bit must be cleared
  - Later the interrupt code is executed

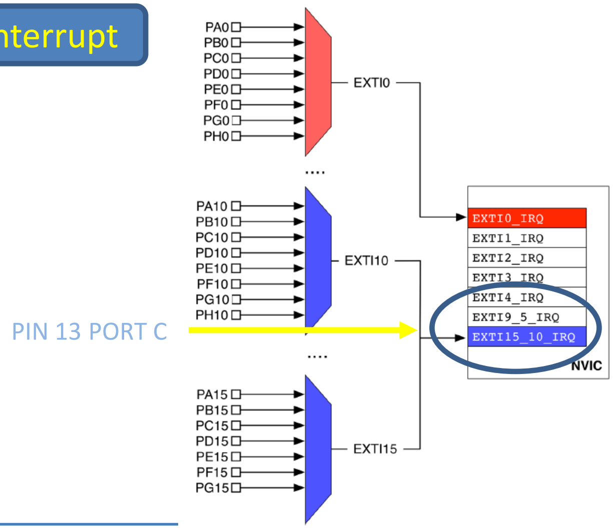
Using Interrupts

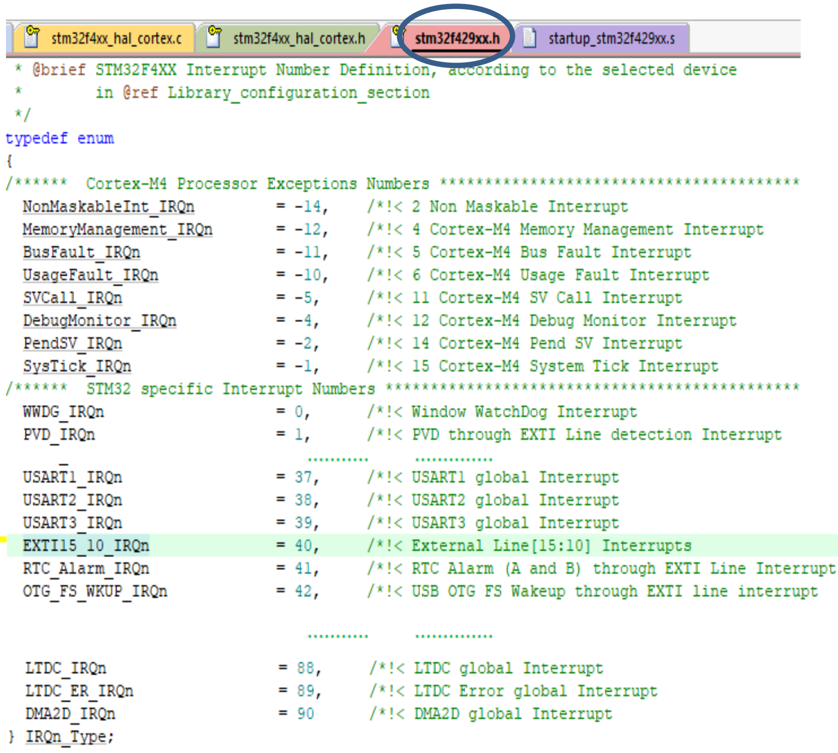
This code shows how manage an interrupt for an external GPIO

A switch is connected to PIN 13 PORT C

```
//Ports 10 to 15 use the EXTI15_10 IRQ line.

 HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);



//ISR implementation

void EXTI15_10_IRQHandler(void) {
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
  // ISR Body;
}
```

1.- Enable Interrupt





```
 * @brief STM32F4XX Interrupt Number Definition, according to the selected device
 *        in @ref Library_configuration_section
 */
typedef enum
{
/******  Cortex-M4 Processor Exceptions Numbers ****************************************
  NonMaskableInt_IRQn         = -14,    /*!< 2 Non Maskable Interrupt
  MemoryManagement_IRQn       = -12,    /*!< 4 Cortex-M4 Memory Management Interrupt
  BusFault_IRQn               = -11,    /*!< 5 Cortex-M4 Bus Fault Interrupt
  UsageFault_IRQn             = -10,    /*!< 6 Cortex-M4 Usage Fault Interrupt
  SVCall_IRQn                 = -5,     /*!< 11 Cortex-M4 SV Call Interrupt
  DebugMonitor_IRQn           = -4,     /*!< 12 Cortex-M4 Debug Monitor Interrupt
  PendSV_IRQn                 = -2,     /*!< 14 Cortex-M4 Pend SV Interrupt
  SysTick_IRQn                = -1,     /*!< 15 Cortex-M4 System Tick Interrupt
/******  STM32 specific Interrupt Numbers **********************************************
  WWDG_IRQn                   = 0,      /*!< Window WatchDog Interrupt
  PVD_IRQn                    = 1,      /*!< PVD through EXTI Line detection Interrupt
                           ...........   ..............
  USART1_IRQn                 = 37,     /*!< USART1 global Interrupt
  USART2_IRQn                 = 38,     /*!< USART2 global Interrupt
  USART3_IRQn                 = 39,     /*!< USART3 global Interrupt
  EXTI15_10_IRQn              = 40,     /*!< External Line[15:10] Interrupts
  RTC_Alarm_IRQn              = 41,     /*!< RTC Alarm (A and B) through EXTI Line Interrupt
  OTG_FS_WKUP_IRQn            = 42,     /*!< USB OTG FS Wakeup through EXTI line interrupt

                           ...........   ..............

  LTDC_IRQn                   = 88,     /*!< LTDC global Interrupt
  LTDC_ER_IRQn                = 89,     /*!< LTDC Error global Interrupt
  DMA2D_IRQn                  = 90      /*!< DMA2D global Interrupt
} IRQn_Type;
```

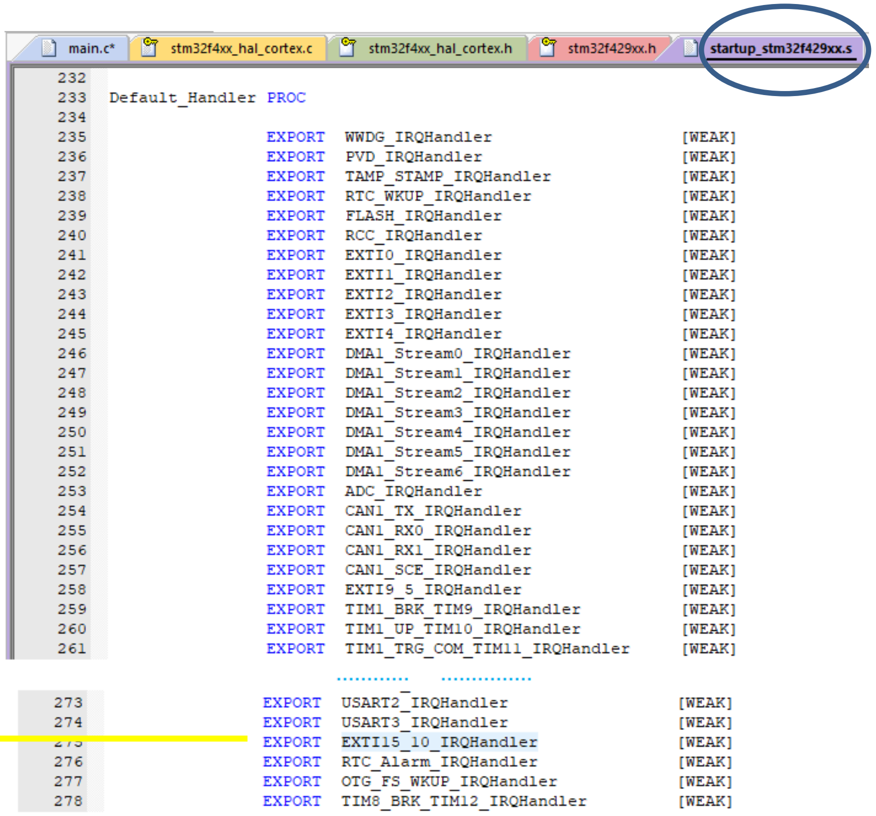
Using Interrupts

This code shows how manage interrupt for an external GPIO.
A switch is connected to PIN 13 PORT C

```
//Ports 10 to 15 use the EXTI15_10 IRQ line.

  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);



//ISR implementation

void EXTI15_10_IRQHandler(void) {
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
  // ISR Body;
}
```

2.- Coding ISR

3.- **Clear** pending flag

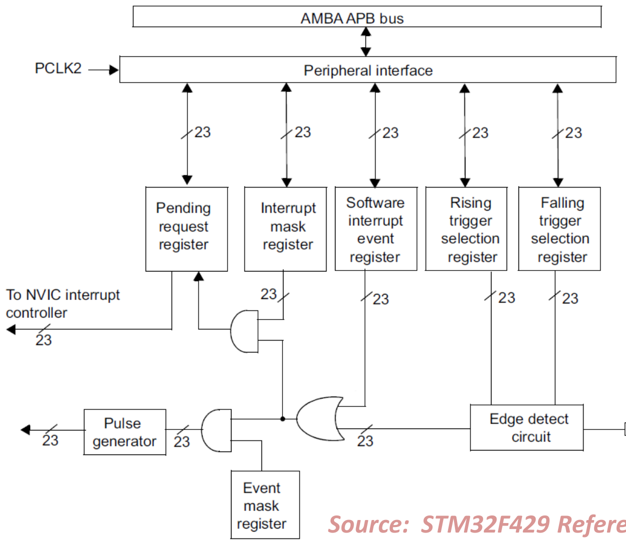


```

Default_Handler PROC

                EXPORT  WWDG_IRQHandler                   [WEAK]
                EXPORT  PVD_IRQHandler                    [WEAK]
                EXPORT  TAMP_STAMP_IRQHandler             [WEAK]
                EXPORT  RTC_WKUP_IRQHandler               [WEAK]
                EXPORT  FLASH_IRQHandler                  [WEAK]
                EXPORT  RCC_IRQHandler                    [WEAK]
                EXPORT  EXTI0_IRQHandler                  [WEAK]
                EXPORT  EXTI1_IRQHandler                  [WEAK]
                EXPORT  EXTI2_IRQHandler                  [WEAK]
                EXPORT  EXTI3_IRQHandler                  [WEAK]
                EXPORT  EXTI4_IRQHandler                  [WEAK]
                EXPORT  DMA1_Stream0_IRQHandler           [WEAK]
                EXPORT  DMA1_Stream1_IRQHandler           [WEAK]
                EXPORT  DMA1_Stream2_IRQHandler           [WEAK]
                EXPORT  DMA1_Stream3_IRQHandler           [WEAK]
                EXPORT  DMA1_Stream4_IRQHandler           [WEAK]
                EXPORT  DMA1_Stream5_IRQHandler           [WEAK]
                EXPORT  DMA1_Stream6_IRQHandler           [WEAK]
                EXPORT  ADC_IRQHandler                    [WEAK]
                EXPORT  CAN1_TX_IRQHandler                [WEAK]
                EXPORT  CAN1_RX0_IRQHandler               [WEAK]
                EXPORT  CAN1_RX1_IRQHandler               [WEAK]
                EXPORT  CAN1_SCE_IRQHandler               [WEAK]
                EXPORT  EXTI9_5_IRQHandler                [WEAK]
                EXPORT  TIM1_BRK_TIM9_IRQHandler          [WEAK]
                EXPORT  TIM1_UP_TIM10_IRQHandler          [WEAK]
                EXPORT  TIM1_TRG_COM_TIM11_IRQHandler     [WEAK]

                    ............    ..............

                EXPORT  USART2_IRQHandler                 [WEAK]
                EXPORT  USART3_IRQHandler                 [WEAK]
                EXPORT  EXTI15_10_IRQHandler              [WEAK]
                EXPORT  RTC_Alarm_IRQHandler              [WEAK]
                EXPORT  OTG_FS_WKUP_IRQHandler            [WEAK]
                EXPORT  TIM8_BRK_TIM12_IRQHandler         [WEAK]
```

Using Interrupts

This code shows how manage interrupt for an external GPIO.
A switch is connected to PIN 13 PORT C

```
//Ports 10 to 15 use the EXTI15_10 IRQ line.

  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);



//ISR implementation

void EXTI15_10_IRQHandler(void) {
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
  // ISR Body;
}
```



Source: STM32F429 Reference manual

¡¡¡¡ IMPORTANT !!!!
Do not forget to configure the peripheral to work in interrupt MODE

Configure the peripheral before interrupts are used

```
  __HAL_RCC_GPIOC_CLK_ENABLE();

/*Configure GPIO pin : PC13 - USER BUTTON */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

See: stm32l4xx_hal_gpio.h

Using Interrupts

Use this method!

- HAL interrupt Model.

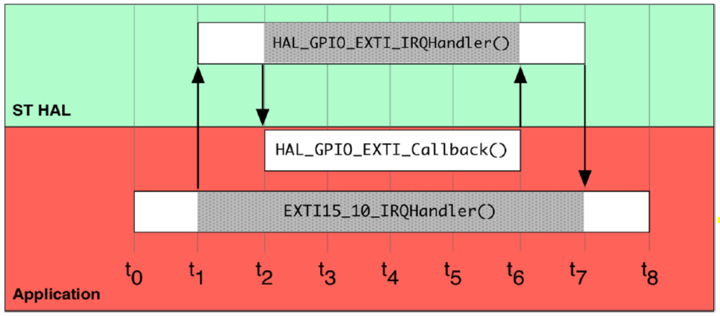
```
//Ports 10 to 15 use the EXTI15_10 IRQ line.

  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);



//ISR implementation

void EXTI15_10_IRQHandler(void) {
  __HAL_GPIO_EXTI_CLEAR_IT(GPIO_PIN_13);
  // ISR Body;
}
```

The peripheral interrupt flag is cleared

```
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);


void EXTI15_10_IRQHandler(void) {
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_13);
}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {

    //ISR Body

}
```

HAL provides a higher degree of abstraction

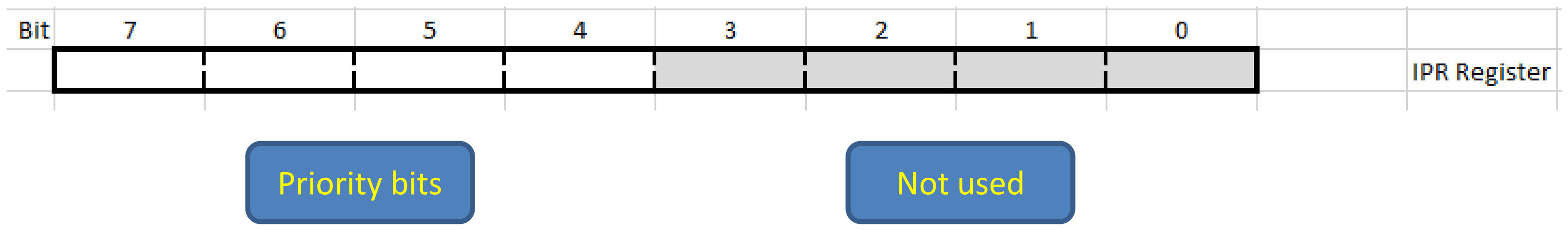
Look for the function in the file stm32F4xx_hal_YYY.c
With YYY for specific peripheral.


HAL_GPIO_EXTI_IRQHandler()
ST HAL
HAL_GPIO_EXTI_Callback()
EXTI15_10_IRQHandler()
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8
Application

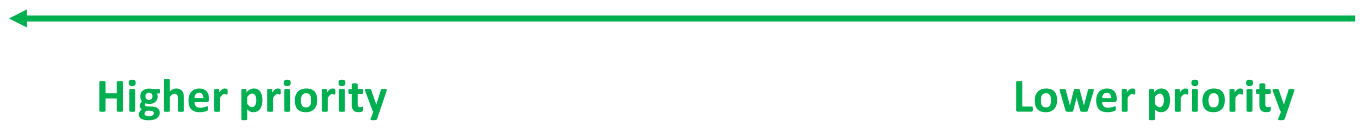
Source: Mastering STM32. Carmine Noviello

Interrupts Priority

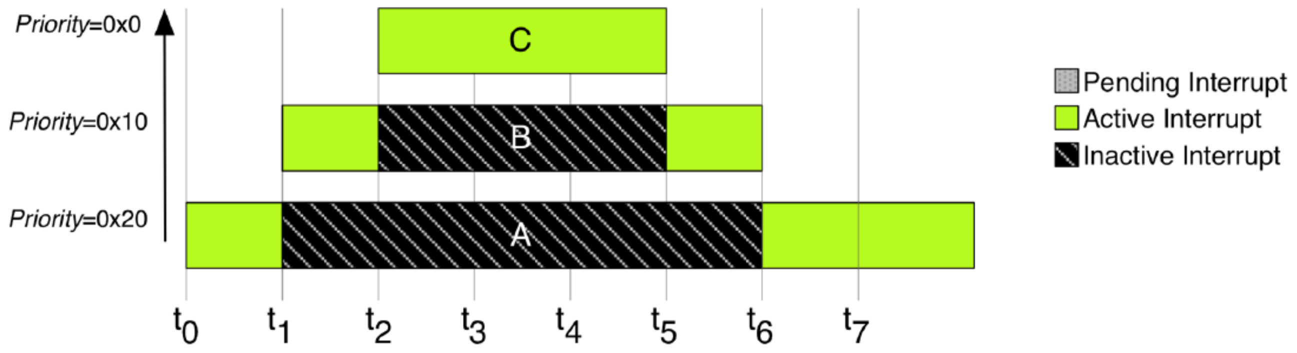
- ARM Cortex-M architecture has the capability to assign priority to interrupts
- Priority is handled using an eight-bit register (M3/M4/M7)



16 priority levels 0x00, 0x10, 0x20 ----- 0xD0, 0xE0, 0xF0.



Interrupt Preemption (concept)



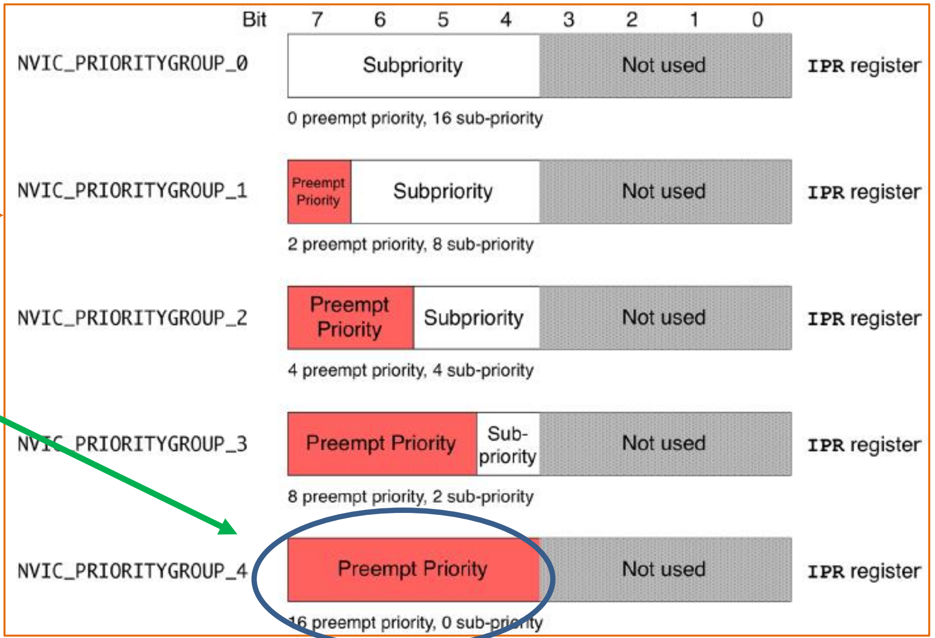
Interrupts Priority

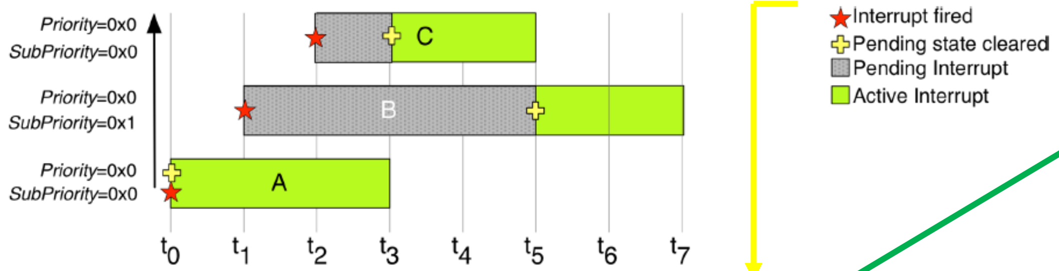
- Priority / Subpriority

Scheme: Preemption Priority / Sub-priority
Five working groups can be configured

Bits AIRCR in System Control Block register let assign a Priority / Subpriority scheme.

Source: Mastering STM32. Carmine Noviello



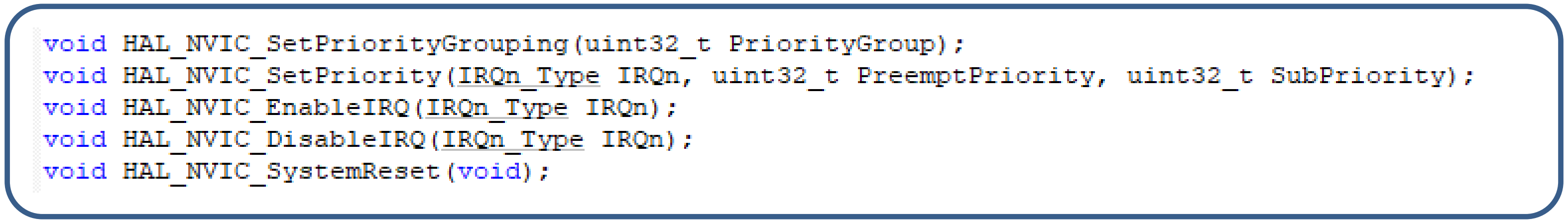


HAL_Init()
function configures NVIC_PRIORITYGROUP_4

```
void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup);
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority);
```

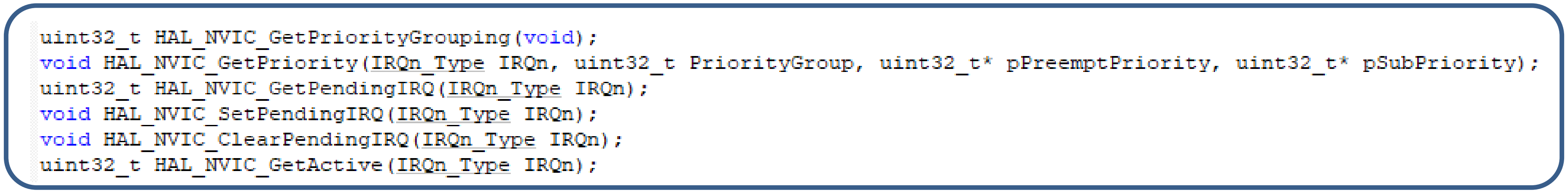
HAL Functions and Handlers

Initialization and de-initialization



```
void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup);
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority);
void HAL_NVIC_EnableIRQ(IRQn_Type IRQn);
void HAL_NVIC_DisableIRQ(IRQn_Type IRQn);
void HAL_NVIC_SystemReset(void);
```

Peripheral Control



```
uint32_t HAL_NVIC_GetPriorityGrouping(void);
void HAL_NVIC_GetPriority(IRQn_Type IRQn, uint32_t PriorityGroup, uint32_t* pPreemptPriority, uint32_t* pSubPriority);
uint32_t HAL_NVIC_GetPendingIRQ(IRQn_Type IRQn);
void HAL_NVIC_SetPendingIRQ(IRQn_Type IRQn);
void HAL_NVIC_ClearPendingIRQ(IRQn_Type IRQn);
uint32_t HAL_NVIC_GetActive(IRQn_Type IRQn);
```

IRQn_Type see stm32f29xx.h file

PriorityGroup see stm32f4xx_hal_cortex.h