INTERRUPTS IN ARM PROCESSORS

study.embeddedexpert.io

Interrupts : Interrupt vs. Polling

A single microprocessor can serve several modules by:

- **Interrupt**

When module needs service, it notifies the CPU by sending an interrupt signal. When the CPU receives the signal the CPU interrupts whatever it is doing and services the module.

- **Polling**

The CPU continuously monitors the status of a given module, when a particular status condition is met the CPU then services the module.

Interrupts : Interrupt vs. Polling

```
int main()
{
  while(1){
    . . .
   }
}

 OnSwitch_ISR{
  getData()
}
```

Interrupt

```
int main()
{
   while(1){

   if(switch = on ){
   getData(); }
   . . .
   }
  }
```

Polling

Interrupts : Interrupt Service Routine

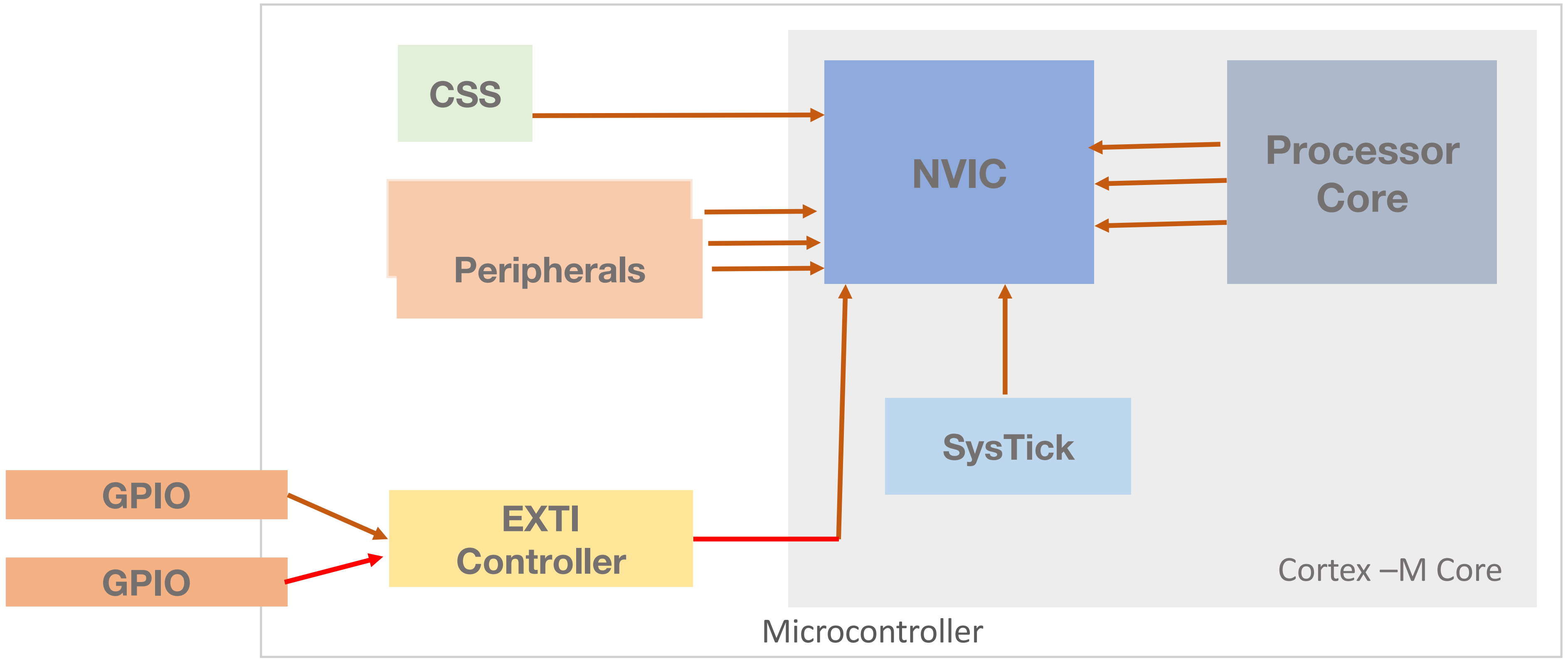
The function that gets executed when an interrupt occurs is called the Interrupt Service Routine(ISR) or the Interrupt Handler

Interrupts : NVIC

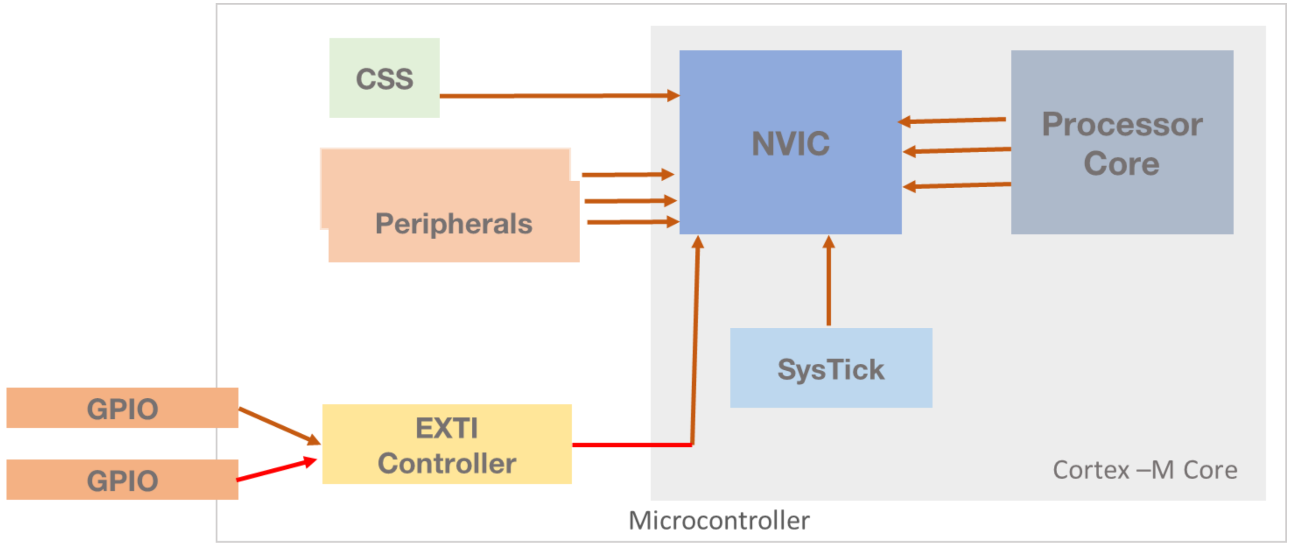
Nested Vector Interrupt Controller (NVIC)

- A dedicated hardware inside the Cortex-Microcontroller
- It is responsible for handling interrupts.

Interrupts : NVIC



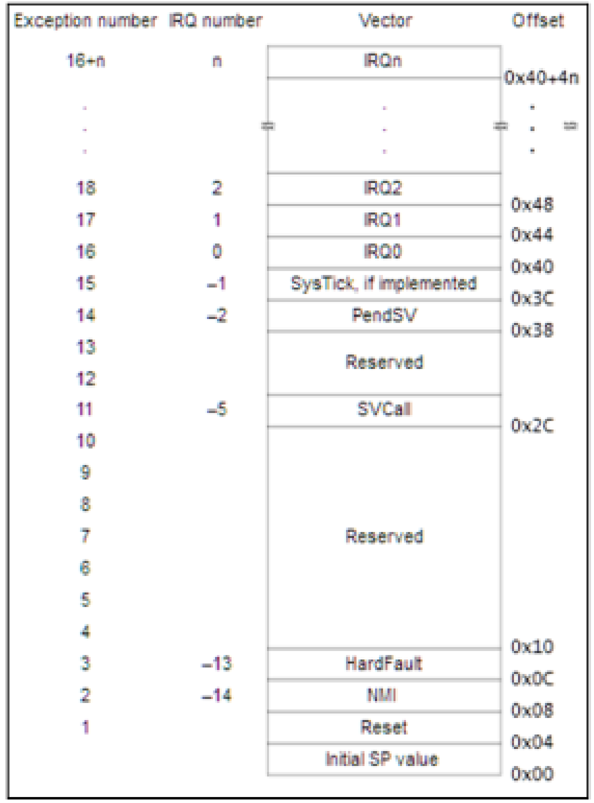
Interrupts : NVIC



- Interrupts from the processor core are known as *exceptions*.
- Interrupts from outside the processor core are known as *hardware exceptions* or *Interrupt Requests*.

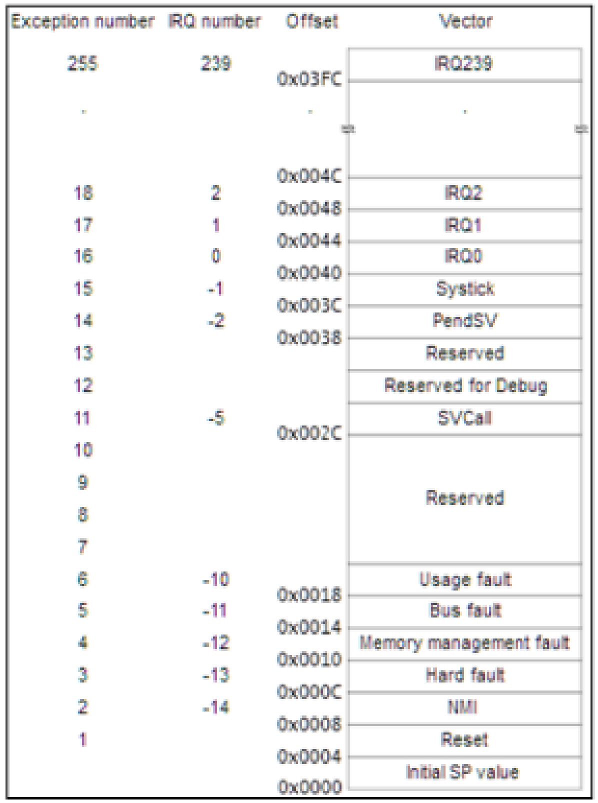
Interrupts : The Vector Table

- The vector table contains the addresses of the <u>Interrupt Handlers</u> and <u>Exception Handlers</u>.



Exception number	IRQ number	Vector	Offset
16+n	n	IRQn	0x40+4n
.		.	.
.		.	.
.		.	.
18	2	IRQ2	0x48
17	1	IRQ1	0x44
16	0	IRQ0	0x40
15	–1	SysTick, if implemented	0x3C
14	–2	PendSV	0x38
13		Reserved	
12			
11	–5	SVCall	0x2C
10		Reserved	
9			
8			
7			
6			
5			
4			0x10
3	–13	HardFault	0x0C
2	–14	NMI	0x08
1		Reset	0x04
		Initial SP value	0x00

M0+



Exception number	IRQ number	Offset	Vector
255	239	0x03FC	IRQ239
.		.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

M4/M7

Interrupts : External Interrupt (EXTI) lines

- GPIO pins are connected to EXTI lines
- It possible to enable interrupt for any GPIO pin
- Multiple pins share the same EXTI line

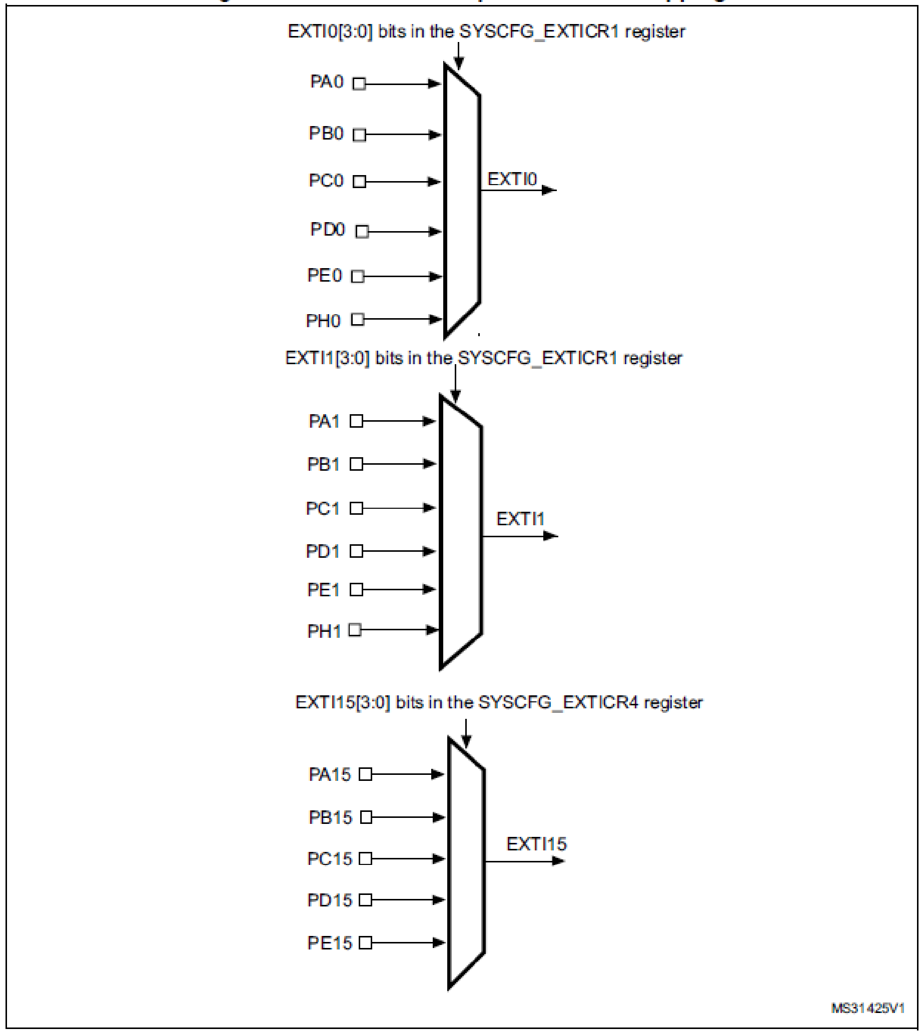
- **Pin 0 of every Port** *is connected* ***EXTI0_IRQ***
- **Pin 1 of every Port** *is connected* ***EXTI1_IRQ***
- **Pin 2 of every Port** *is connected* ***EXTI2_IRQ***
- **Pin 3 of every Port** *is connected* ***EXTI3_IRQ***

…

This means we cannot have PB0 and PA0 as input interrupt pins at the same time since they are connected to the same multiplexer i.e. EXTI0

Same for PC4 and PB4 at the same time, etc.

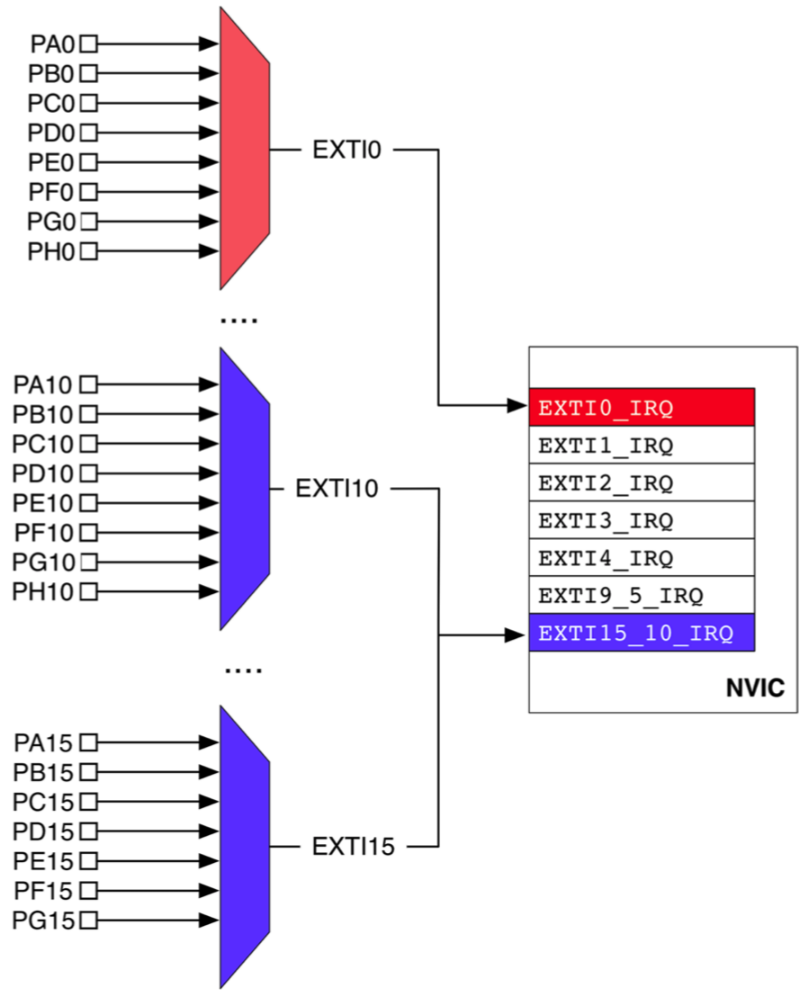
Figure 30. External interrupt/event GPIO mapping


EXTI0[3:0] bits in the SYSCFG_EXTICR1 register
PA0
PB0
PC0
PD0
PE0
PH0
EXTI0
EXTI1[3:0] bits in the SYSCFG_EXTICR1 register
PA1
PB1
PC1
PD1
PE1
PH1
EXTI1
EXTI15[3:0] bits in the SYSCFG_EXTICR4 register
PA15
PB15
PC15
PD15
PE15
EXTI15
MS31425V1

Interrupts : External Interrupt (EXTI) lines

- Pins 10 to 15 share the same IRQ inside the NVIC and therefore are serviced by the same Interrupt Service Routine (ISR)

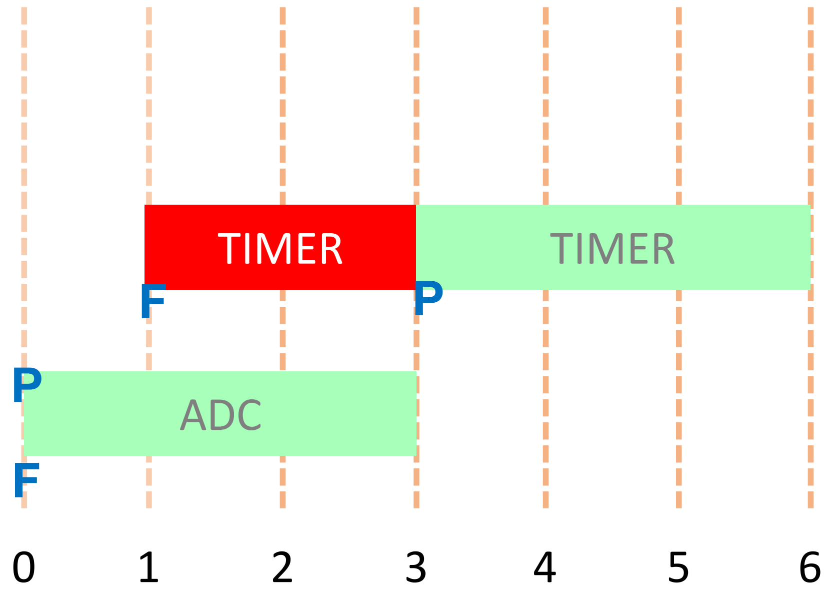
- Application code must be able to find which pin from 10 to 15 generated the interrupt.



Interrupts : States

- Disabled : *This is the default state*
- Enabled : *Interrupt is enabled*
- Pending : *Waiting to be serviced*
- Active : *Being serviced*

Interrupts : States- Pending vs. Active



Time in seconds

Legend

🟩	**Active State**
🟥	**Pending State**
P	**Pending State Cleared**
F	**Interrupt fired**

Let's assume
ADC Interrupt has a higher priority than TIMER interrupt

ADC Interrupt fires at time t = 0.
This is indicated by F

Since there is no other interrupt, the pending state is cleared and the interrupt becomes active.
This is indicated by P

At time t=1 TIMER interrupt fires
This is indicated by F

Since it has a lower priority than the ADC *interrupt it remains in the pending state*

At time t=3 ADC interrupt completes its execution

Since there is no other interrupt with a higher priority , the pending state of the TIMER interrupt is cleared and the interrupt becomes active.
This is indicated by P

Interrupts : Priorities

- Priorities allow us to set which interrupt should execute first.

- They also allow us to set which interrupt can interrupt which.

Interrupts : Vector Table and IRQ#

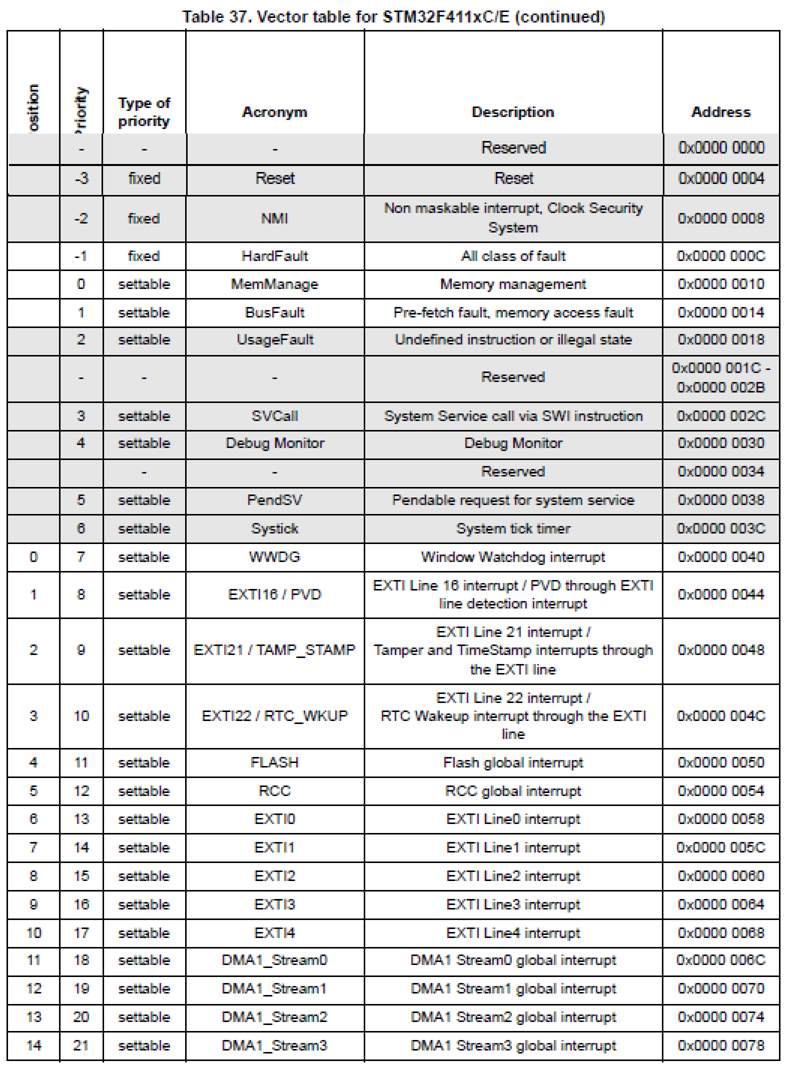


Table 37. Vector table for STM32F411xC/E (continued)

osition	riority	Type of priority	Acronym	Description	Address
	-	-	-	Reserved	0x0000 0000
	-3	fixed	Reset	Reset	0x0000 0004
	-2	fixed	NMI	Non maskable interrupt, Clock Security System	0x0000 0008
	-1	fixed	HardFault	All class of fault	0x0000 000C
	0	settable	MemManage	Memory management	0x0000 0010
	1	settable	BusFault	Pre-fetch 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	System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	EXTI16 / PVD	EXTI Line 16 interrupt / PVD through EXTI line detection interrupt	0x0000 0044
2	9	settable	EXTI21 / TAMP_STAMP	EXTI Line 21 interrupt / Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	EXTI22 / RTC_WKUP	EXTI Line 22 interrupt / 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
12	19	settable	DMA1_Stream1	DMA1 Stream1 global interrupt	0x0000 0070
13	20	settable	DMA1_Stream2	DMA1 Stream2 global interrupt	0x0000 0074
14	21	settable	DMA1_Stream3	DMA1 Stream3 global interrupt	0x0000 0078

- Part of the vector table for stm32f411

Interrupts : Priorities

Some interrupt priorities are defined by ARM, these cannot be changed. E.g.:

- RESET : *Priority of* -3
- NMI : *Priority of* -2
- HardFault : *Priority of* -1

Lower number = Higher priority

Interrupts : Priorities in M3/M4/M7

- Priority of each interrupt is defined using one of the Interrupt Priority Registers (IPR)

- Each Interrupt Request(IRQ) uses 8-bits inside a single IPR register

- Therefore **one IPR** register allows us to configure the priorities of **four** different Interrupt Requests

- Example : **IPR0** holds the priorities of *IRQ0,IRQ1,IRQ2 and IRQ3*

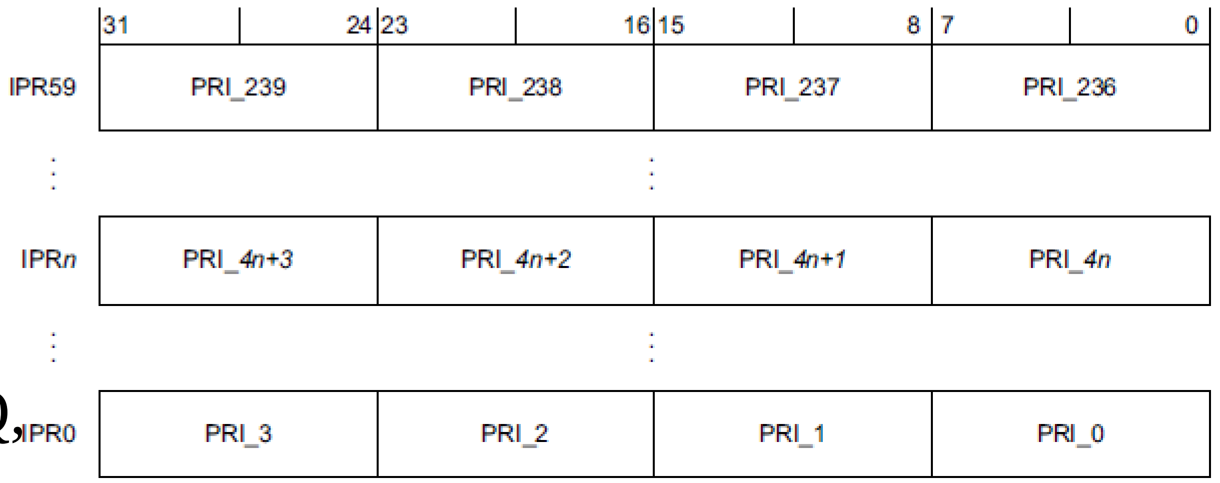
- There are 60 Interrupt Priority Registers : *IPR0 – IPR59*

- There are 60 x 4 = *240 Interrupt Requests (IRQ)*

Interrupts : Priorities in M3/M4/M7

- *8 –bits* to configure the priority of an IRQ implies there are 2^8 = **255 priority levels**

  - STM32 microcontrollers use only the *4 upper bits* to configure the priority of each IRQ, this implies that in STM32 MCUs there are 2^4 = **16 priority levels**



	31 … 24	23 … 16	15 … 8	7 … 0
IPR59	PRI_239	PRI_238	PRI_237	PRI_236
⋮		⋮		
IPR*n*	PRI_4*n*+3	PRI_4*n*+2	PRI_4*n*+1	PRI_4*n*
⋮		⋮		
IPR0	PRI_3	PRI_2	PRI_1	PRI_0

Table 4-9 IPR bit assignments

- ***IPRn*** = *IRQ(4n+3), IRQ(4n+2),IRQ(4n +1) and IRQ(4n)*

Interrupts : Priorities in M3/M4/M7

- *The **16 priority levels** :*

0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70,

0x80, 0x90, 0xA0, 0xB0, 0xC0, 0xD0, 0xE0, 0xF0

- *Highest Priority* = 0x00 = 0

- *Lowest Priority* = 0xF0 = 16

Interrupts : Priorities in M3/M4/M7

- To find the IPR number, we divide the **IRQ** number by 4,the remainder will determine which byte it is in the IPR register.
- Because **only the highest 4 bits** are used for priority, the priority number needs to be multiplied by 16 or **left shift 4 bits**
- To simply the calculation, the **NVIC_IPRx** are defined as an array of *8-bit* registers *IP[x]* in the core_cm3.h, core_cm4.h, core_cm7.h files.

Such that the priority of **IRQx** is controlled by **IP[x]**

E.g.

Setting TIM2 interrupt priority to 3

TIM2 interrupt is IRQ 28 *(we can find this in the stm32f411xe.h)*

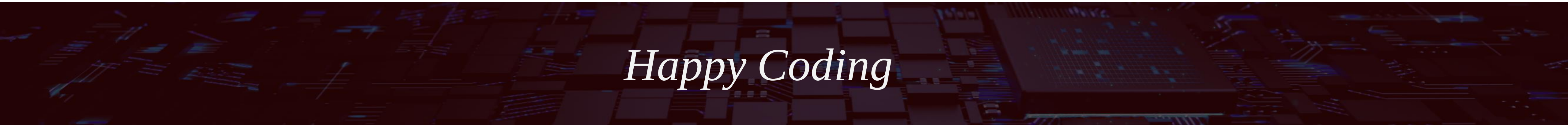
```
NVIC->IP[28] = 3 << 4;
```

or

```
NVIC_SetPriority(TIM2_IRQn,3);
```

Interrupts : Sub-priorities

- The Interrupt Priority Registers (IPR) can also be divided into sub-priorities
- In this configuration there are a series of bits defining **preemption priority** and a series of bits defining the **sub-priority**
- The sub-priority will determine which IRQ will be executed first in the case of multiple pending IRQs



Happy Coding