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(); }
   . . .
   }
  }
```

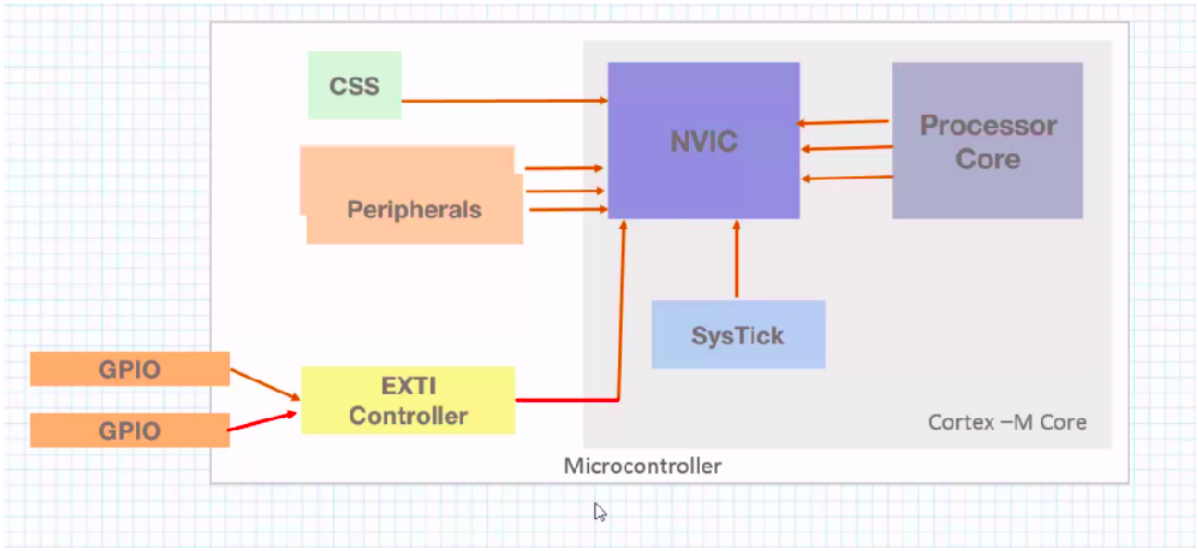
Polling

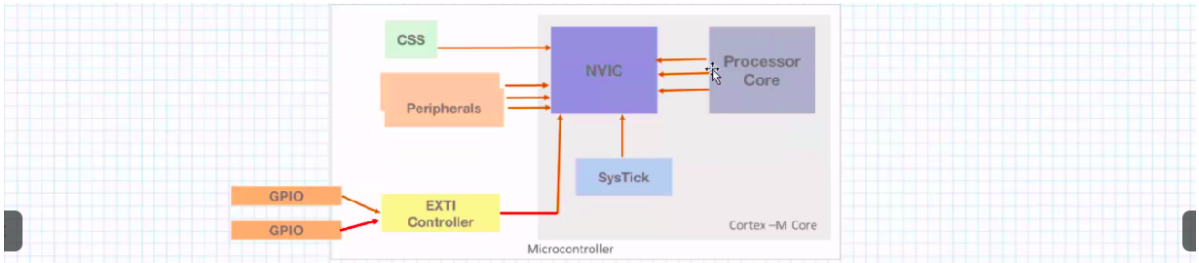
The function that gets executed when an interrupt occurs is called the Interrupt Service Routine(ISR) or the Interrupt Handler

Interrupts : *NVIC*

Nest Vector Interrupt Controller (NVIC)

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





- Interrupts from the processor core are know 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 Interrupt Handlers and Exception Handlers.

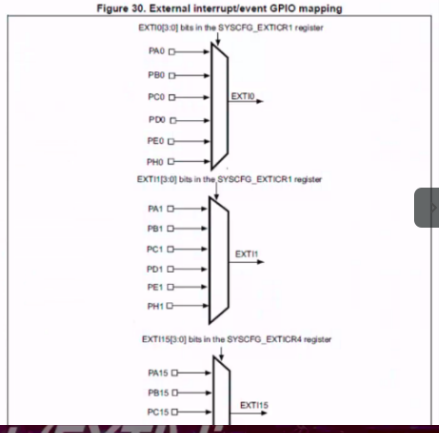
Interrupts : *External Interrupt (EXTI) lines*

- 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***

...

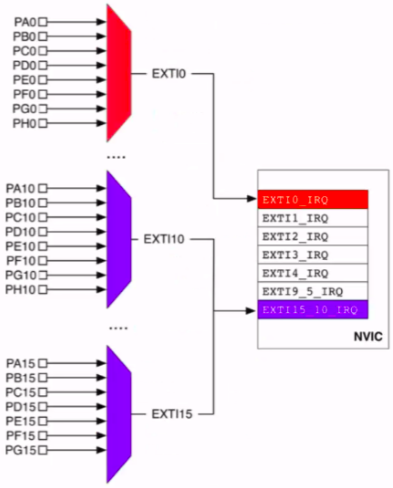
This means we cannot have PB0 and PA0 as input interrupt pins at the same time since they are connected to the same



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)

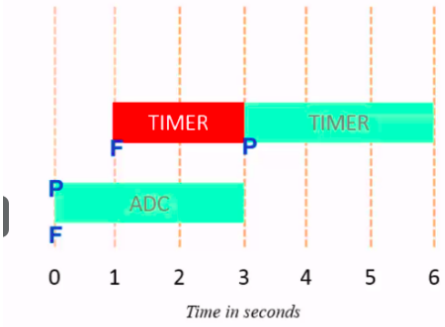
- Application code must be able to find which pin from 10 to 15 generated the interrupt.



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*



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#*

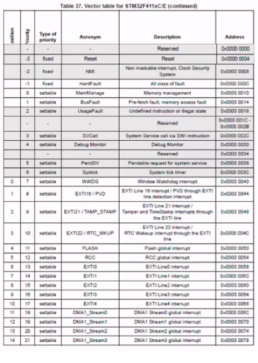


Table 37. Vector table for STM32F411xC/E (continued)

position	Priority	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 0028
	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 pfeach 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_*4n*+3	PRI_*4n*+2	PRI_*4n*+1	PRI_*4n*
⋮				
IPR0	PRI_3	PRI_2	PRI_1	PRI_0

Table 4-9 IPR bit assignments

- *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);
```