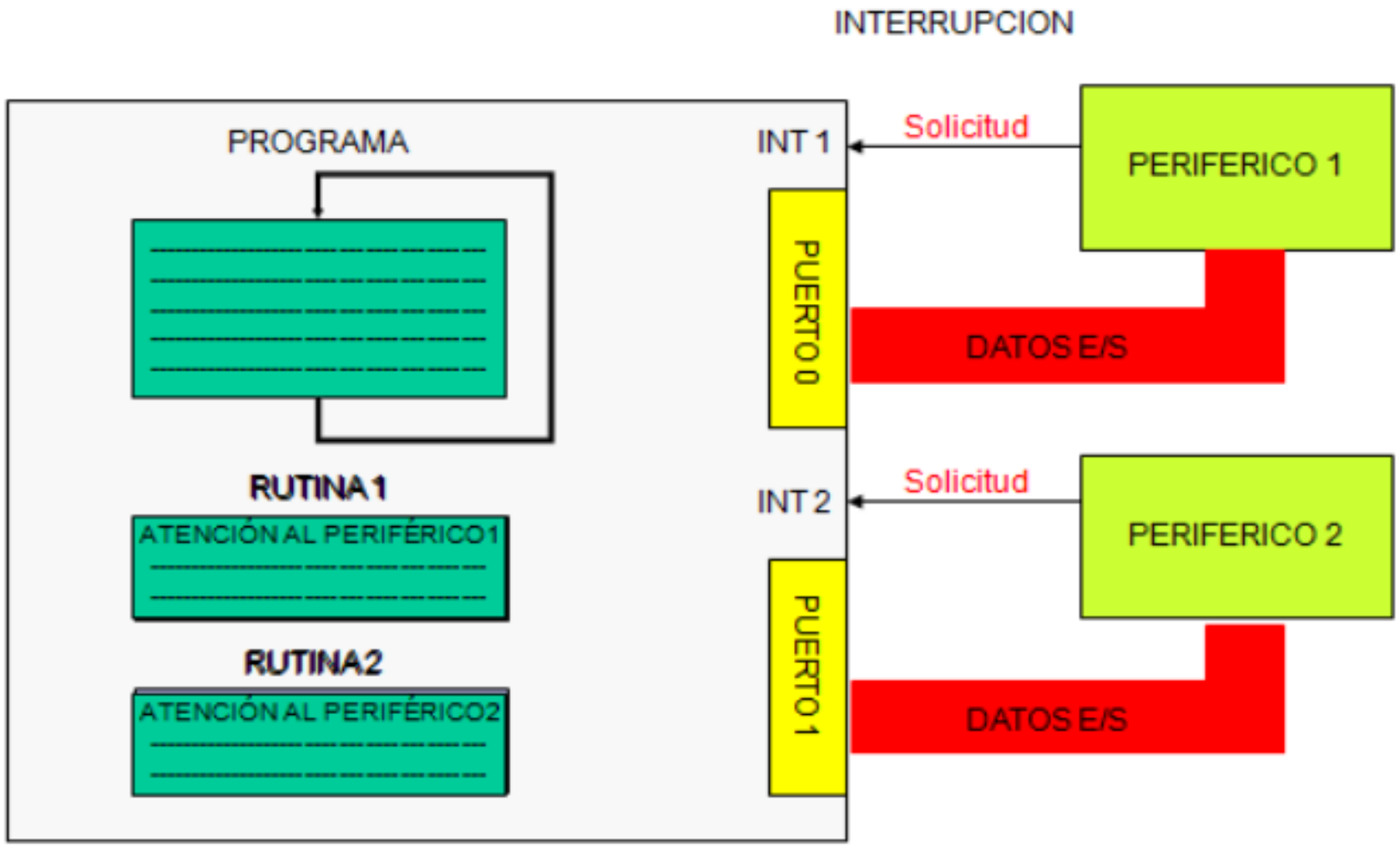
MICROS 32 BITS
STM - INTERRUPCIONES

REALIZADA POR: ROBINSON JIMENEZ MORENO

PROFESORA: LUISA FERNANDA GARCÍA

UNIVERSIDAD MILITAR NUEVA GRANADA

INGENIERÍA MECATRÓNICA UMNG

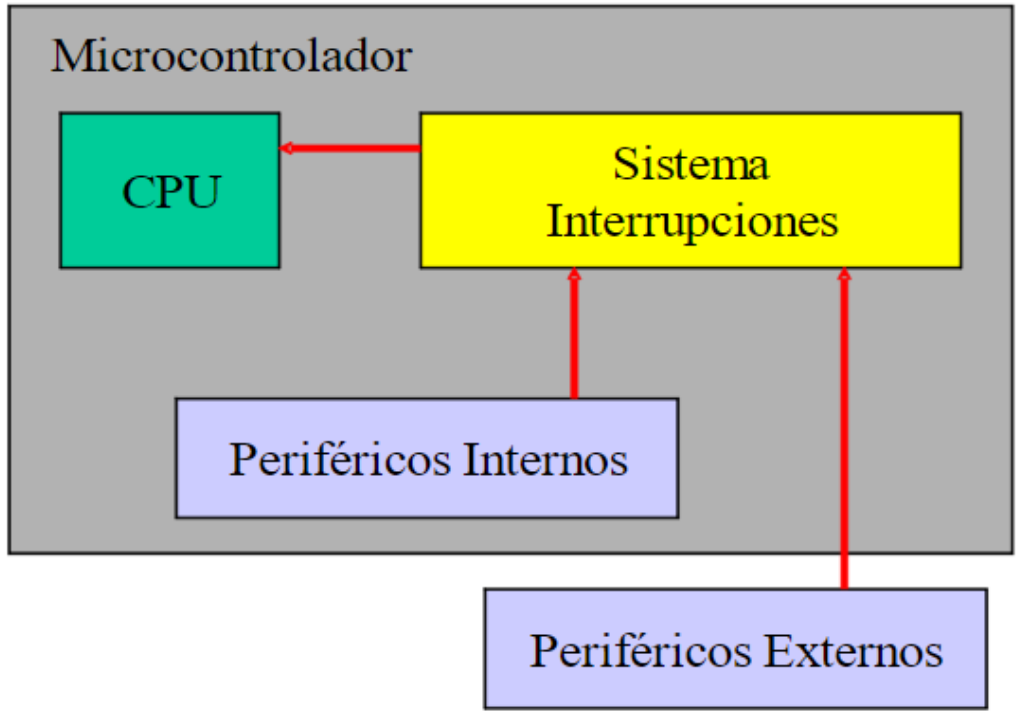

INTERRUPCION
PROGRAMA
RUTINA 1
ATENCIÓN AL PERIFÉRICO1
RUTINA2
ATENCIÓN AL PERIFÉRICO2
INT 1
INT 2
Solicitud
Solicitud
PERIFERICO 1
PERIFERICO 2
PUERTO 0
PUERTO 1
DATOS E/S
DATOS E/S

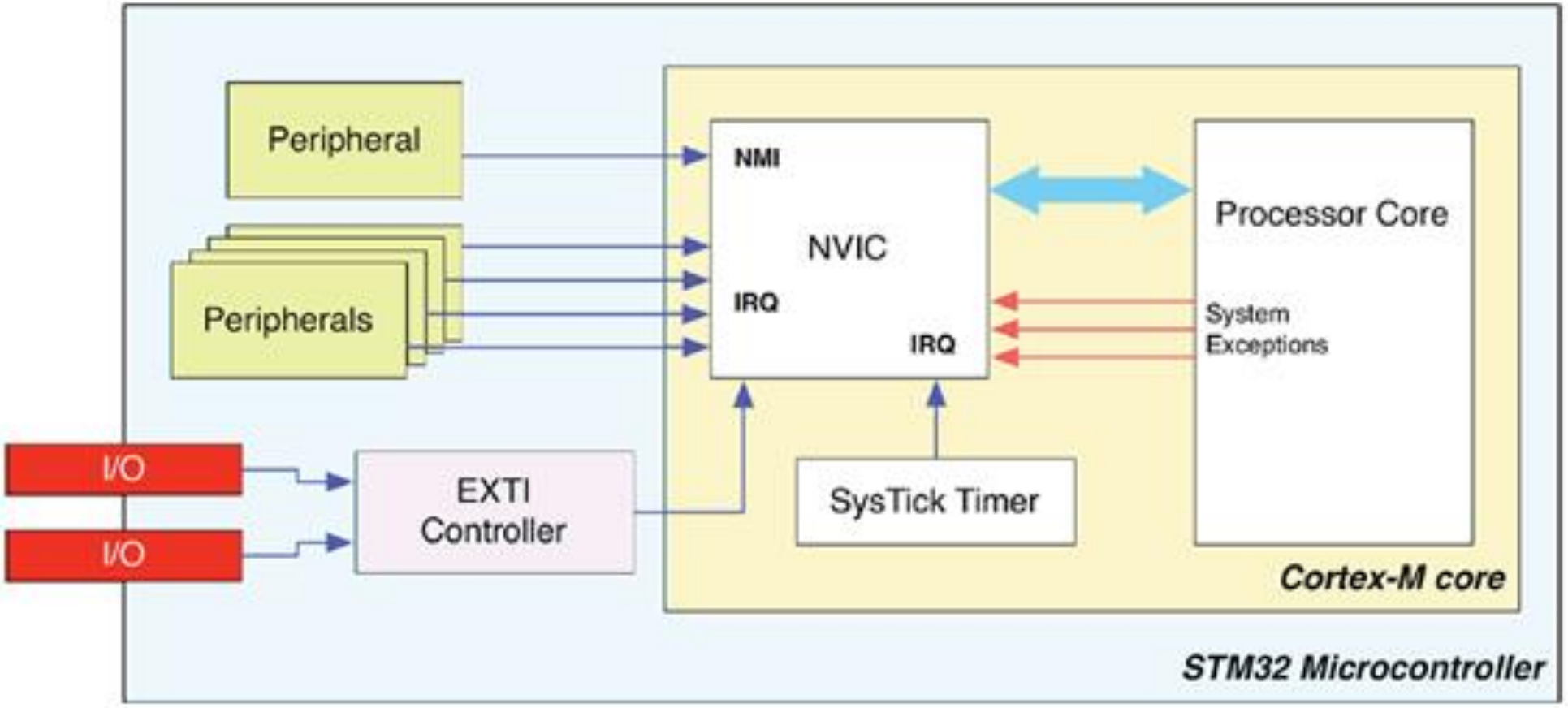
INTERRUPCIONES EN ARM CORTEX-M

Generadas por fuentes externas: La petición de interrupción se puede generar tanto por nivel como por flanco en el pin correspondiente

EXTI0,1,2,..

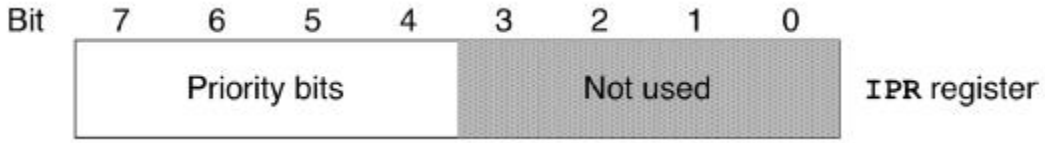
Interrupciones generadas por los periféricos internos: Temporizadores/Contadores donde cada fuente de interrupción tiene asociado su propio vector de interrupción para localizar el manejador.





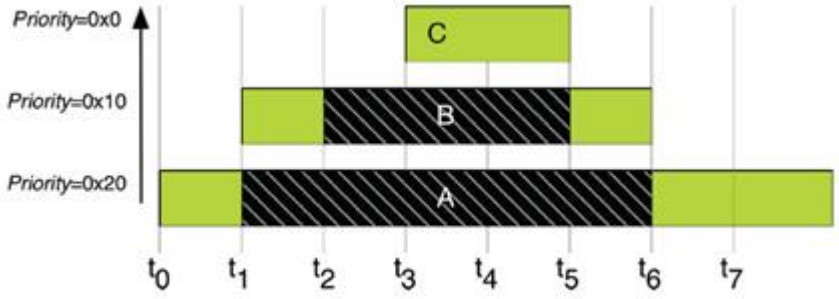
https://www.intesc.mx/interrupciones-en-microcontroladores-de-la-serie-stm32f4/

PRIORIDADES



Se tienen los dieciséis niveles de prioridad máxima: 0x00, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70, 0x80, 0x90, 0xA0, 0xB0, 0xC0, 0xD0, 0xE0, 0xF0. Cuanto menor sea este número, mayor será la prioridad. Es decir, un IRQ que tiene una prioridad igual a 0x10 tiene una prioridad más alta que un IRQ con un nivel de prioridad igual a 0xA0. Si dos interrupciones disparan al mismo tiempo, el que tenga la mayor prioridad será atendido primero.

La letra *A* representa un IRQ con una prioridad baja (0x20) que se dispara en el tiempo t0. El ISR inicia la ejecución de A-ISR, pero el B-IRQ que tiene una prioridad más alta (0x10), se dispara en el instante t1. Entonces la ejecución de A-ISR se detiene y comienza la ejecución de B-ISR. Después en un tiempo t3 se dispara C-IRQ (prioridad 0x00), la ejecución de B-ISR se detiene y se inicia la ejecución de C-ISR. Cuando la ejecución de C-ISR termina, se reanuda la ejecución de B-ISR hasta que termina. Finalmente se reanuda la ejecución de A-ISR hasta el final. A este proceso de control de interrupciones se le llama: "anidado". El mecanismo inducido por prioridades de interrupción conduce al nombre del controlador **NVIC** o **Controlador de Interrupción Vectorial Anidado**



https://www.intesc.mx/interrupciones-en-microcontroladores-de-la-serie-stm32f4/

Los ARM Cortex-M disponen de un controlador de interrupciones: Nested Vectored Interrupt Controller (NVIC) - controlador de interrupción vectorial anidado.

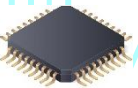
Cuando se produce una interrupción el **NVIC** compara la prioridad de esta interrupción (Pi) con el nivel de prioridad de ejecución actual (Pa),

Si **Pi > Pa** se ejecuta el manejador de la interrupción

Las interrupciones pueden ser habilitadas / inhibidas,
Up to 98 maskable interrupt channels for STM32F75xxx and STM32F74xxx (not including the 16 interrupt lines of Cortex®-M7 with FPU) • 16 programmable priority levels (4 bits of interrupt priority are used)

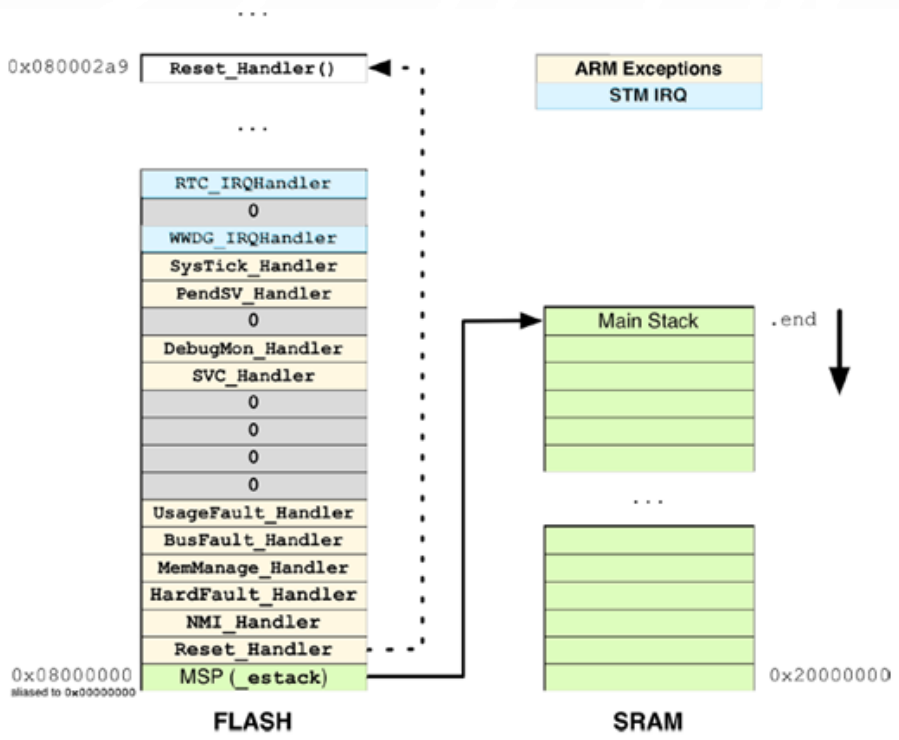
CMSIS HAL (Hardware Abstraction Layer) utiliza números de IRQ (IRQn) para identificar las interrupciones.

La primera interrupción de dispositivo tiene el IRQn = 0, se utilizan valores negativos de IRQn para las “processor core exceptions”. El fichero stm32f4xx.h contiene el Interrupt Number Definition: proporciona los números de interrupción (IRQn) para todas las interrupciones y excepciones.



ARM distingue entre las excepciones del sistema, que se originan dentro del núcleo de la CPU, y las excepciones de hardware procedentes de periféricos externos, también llamadas “Solicitudes de Interrupción” (IRQ). Los programadores manejan las interrupciones mediante el uso de ISRs específicos, que están codificados en un nivel superior (a menudo usando el lenguaje C). El procesador sabe dónde ubicar estas rutinas gracias a una tabla indirecta que contiene las direcciones en memoria de las Rutinas de Servicio de Interrupción. Esta tabla se denomina comúnmente tabla del vector de interrupciones que es propia para cada procesador.

Por convención, la tabla vectorial comienza en la dirección de hardware 0x00000000 en todos los procesadores basados en Cortex-M. Si la tabla vectorial reside en la memoria no volátil interna, y dado que dicha memoria en todas los MCUs STM32 está asignado desde la dirección 0x08000000, como se muestra en la imagen 4, pero en cuanto el microcontrolador enciende, enmascara esta dirección y la ve como 0x00000000.



https://www.intesc.mx/interrupciones-en-microcontroladores-de-la-serie-stm32f4/

- stm32f4xx.h

```
/**
 * @brief STM32F4XX Interrupt Number Definition, according to the selected device
 *        in @ref Library_configuration_section
 */
typedef enum IRQn
{
/******  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               */
  TAMP_STAMP_IRQn            = 2,      /*!< Tamper and TimeStamp interrupts through the EXTI line   */
  RTC_WKUP_IRQn              = 3,      /*!< RTC Wakeup interrupt through the EXTI line              */
  FLASH_IRQn                 = 4,      /*!< FLASH global Interrupt                                  */
  RCC_IRQn                   = 5,      /*!< RCC global Interrupt                                    */
  EXTI0_IRQn                 = 6,      /*!< EXTI Line0 Interrupt                                    */
  EXTI1_IRQn                 = 7,      /*!< EXTI Line1 Interrupt                                    */
  EXTI2_IRQn                 = 8,      /*!< EXTI Line2 Interrupt                                    */
  EXTI3_IRQn                 = 9,      /*!< EXTI Line3 Interrupt                                    */
  EXTI4_IRQn                 = 10,     /*!< EXTI Line4 Interrupt                                    */
  DMA1_Stream0_IRQn          = 11,     /*!< DMA1 Stream 0 global Interrupt                          */
  DMA1_Stream1_IRQn          = 12,     /*!< DMA1 Stream 1 global Interrupt                          */
  DMA1_Stream2_IRQn          = 13,     /*!< DMA1 Stream 2 global Interrupt                          */
  DMA1_Stream3_IRQn          = 14,     /*!< DMA1 Stream 3 global Interrupt                          */
  DMA1_Stream4_IRQn          = 15,     /*!< DMA1 Stream 4 global Interrupt                          */
  DMA1_Stream5_IRQn          = 16,     /*!< DMA1 Stream 5 global Interrupt                          */
  DMA1_Stream6_IRQn          = 17,     /*!< DMA1 Stream 6 global Interrupt                          */
  ADC_IRQn                   = 18,     /*!< ADC1, ADC2 and ADC3 global Interrupts                   */
  CAN1_TX_IRQn               = 19,     /*!< CAN1 TX Interrupt                                       */
  CAN1_RX0_IRQn              = 20,     /*!< CAN1 RX0 Interrupt                                      */
  CAN1_RX1_IRQn              = 21,     /*!< CAN1 RX1 Interrupt                                      */
  CAN1_SCE_IRQn              = 22,     /*!< CAN1 SCE Interrupt                                      */
```

Interrupt lines

I will show now how to configure GPIO pin to be an interrupt and how to handle it in your code with CMSIS function.

In section one (GPIOs) we have 16 interrupt lines. They are **line0** to **line15** and they also represent pin number. This means, **PA0** is connected to **Line0** and **PA13** is connected to **Line13**.

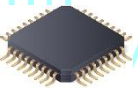
You have to know that **PB0** is also connected to **Line0** and **PC0** also and so on. This is for all pins on board, All **Px0** (where x is GPIO name) pins are connected to **Line0** and let's say all Px3 are connected to **Line3** on the Interrupt channel.

All pins with same number are connected to line with same number. They are multiplexed to one line.

IMPORTANT: You can not use two pins on one line simultaneously:

- **PA0** and **PB0** and **PC0** and so on, are connected to **Line0**, so you can use only one pin at one time to handle interrupt from there.
- **PA0** and **PA5** are connected to different lines, they can be used at the same time.

Each line can trigger an interrupt on rising, falling or rising_falling enge on signal.



Interrupt handlers

OK, now you have selected your pin you want to use. But you have to handle interrupt somehow. This process is described below.

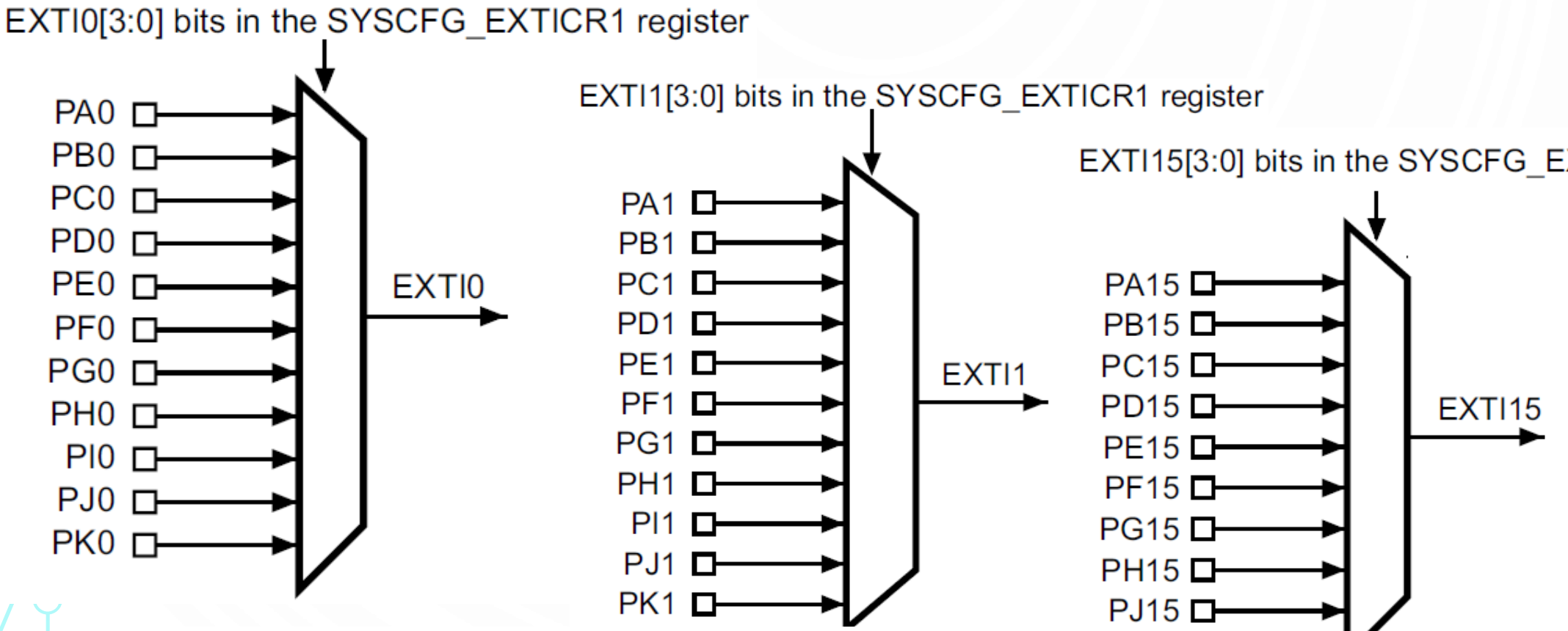
STM32F4 has 7 interrupt handlers for GPIO pins. They are in table below:

Irq	Handler	Description
EXTI0_IRQn	EXTI0_IRQHandler	Handler for pins connected to line 0
EXTI1_IRQn	EXTI1_IRQHandler	Handler for pins connected to line 1
EXTI2_IRQn	EXTI2_IRQHandler	Handler for pins connected to line 2
EXTI3_IRQn	EXTI3_IRQHandler	Handler for pins connected to line 3
EXTI4_IRQn	EXTI4_IRQHandler	Handler for pins connected to line 4
EXTI9_5_IRQn	EXTI9_5_IRQHandler	Handler for pins connected to line 5 to 9
EXTI15_10_IRQn	EXTI15_10_IRQHandler	Handler for pins connected to line 10 to 15

This table show you which **IRQ** you have to set for **NVIC** (first column) and function names to handle your interrupts (second column). You have probably also figured, that only lines 0 to 4 have own IRQ handler. Yes, lines 5-9 have the same interrupt handler and this is also for lines 10 to 15.

External interrupt/event line mapping

Up to 168 GPIOs are connected to the 16 external interrupt/event lines in the following manner:



Similarly external interrupt lines 0 & 1 are mapped to interrupt vector 5, external interrupt lines 2 & 3 are mapped to interrupt vector 6 and external interrupt lines 4 through 15 are mapped to interrupt vector 7.

SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 0 to 3)

These bits are written by software to select the source input for the EXTIx external interrupt.

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin
0111: PH[x] pin
1000: PI[x] pin
1001:PJ[x] pin
1010:PK[x] pin

SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)

Address offset: 0x14

Reset value: 0x0000 0000

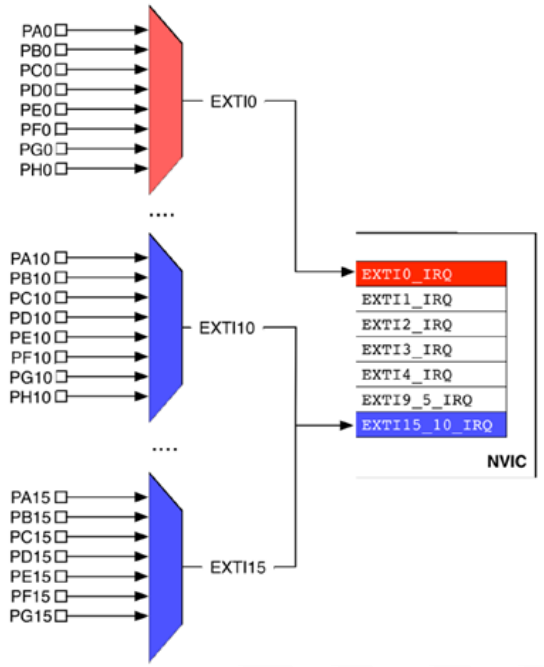
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI15[3:0]				EXTI14[3:0]				EXTI13[3:0]				EXTI12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 12 to 15)

These bits are written by software to select the source input for the EXTIx external interrupt.

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin
0111: PH[x] pin
1001:PJ[x] pin
1010:PK[x] pin

Note: PK[15:12] are not used



5.3.14 RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	LTDC EN	Res.	Res.	SAI2EN	SAI1EN	SPI6EN	SPI5EN	Res.	TIM11 EN	TIM10 EN	TIM9 EN
					rw			rw	rw	rw	rw		rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	SYSCFG EN	SPI4 EN	SPI1 EN	SDMMC1 EN	ADC3 EN	ADC2 EN	ADC1 EN	Res.	Res.	USART6 EN	USART1 EN	Res.	Res.	TIM8 EN	TIM1 EN
	rw	rw	rw	rw	rw	rw	rw			rw	rw			rw	rw

Bit 16 **TIM9EN:** TIM9 clock enable
This bit is set and cleared by software.
0: TIM9 clock disabled
1: TIM9 clock enabled

Bit 15 Reserved, must be kept at reset value.

Bit 14 **SYSCFGEN:** System configuration controller clock enable
This bit is set and cleared by software.
0: System configuration controller clock disabled
1: System configuration controller clock enabled

Bit 13 **SPI4EN:** SPI4 clock enable
This bit is set and cleared by software.
0: SPI4 clock disabled
1: SPI4 clock enabled

11.9.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	IM23	IM22	IM21	IM20	IM19	IM18	IM17	IM16
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:0 **IMx:** Interrupt mask on line x

0: Interrupt request from line x is masked

1: Interrupt request from line x is not masked

The eight other EXTI lines are connected as follows:

- EXTI line 16 is connected to the PVD output
- EXTI line 17 is connected to the RTC Alarm event
- EXTI line 18 is connected to the USB OTG FS Wakeup event
- EXTI line 19 is connected to the Ethernet Wakeup event
- EXTI line 20 is connected to the USB OTG HS (configured in FS) Wakeup event
- EXTI line 21 is connected to the RTC Tamper and TimeStamp events
- EXTI line 22 is connected to the RTC Wakeup event
- EXTI line 23 is connected to the LPTIM1 asynchronous event

Pending register (EXTI_PR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PR23	PR22	PR21	PR20	PR19	PR18	PR17	PR16
								rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 23:0 **PRx:** Pending bit

0: No trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line.

This bit is cleared by programming it to '1'.

Rising trigger selection register (EXTI_RTSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TR23	TR22	TR21	TR20	TR19	TR18	TR17	TR16
								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:0 **TRx:** Rising trigger event configuration bit of line x

0: Rising trigger disabled (for Event and Interrupt) for input line

1: Rising trigger enabled (for Event and Interrupt) for input line

Falling trigger selection register (EXTI_FTSR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TR23	TR22	TR21	TR20	TR19	TR18	TR17	TR16
								rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:24 Reserved, must be kept at reset value.

Bits 23:0 **TRx:** Falling trigger event configuration bit of line x

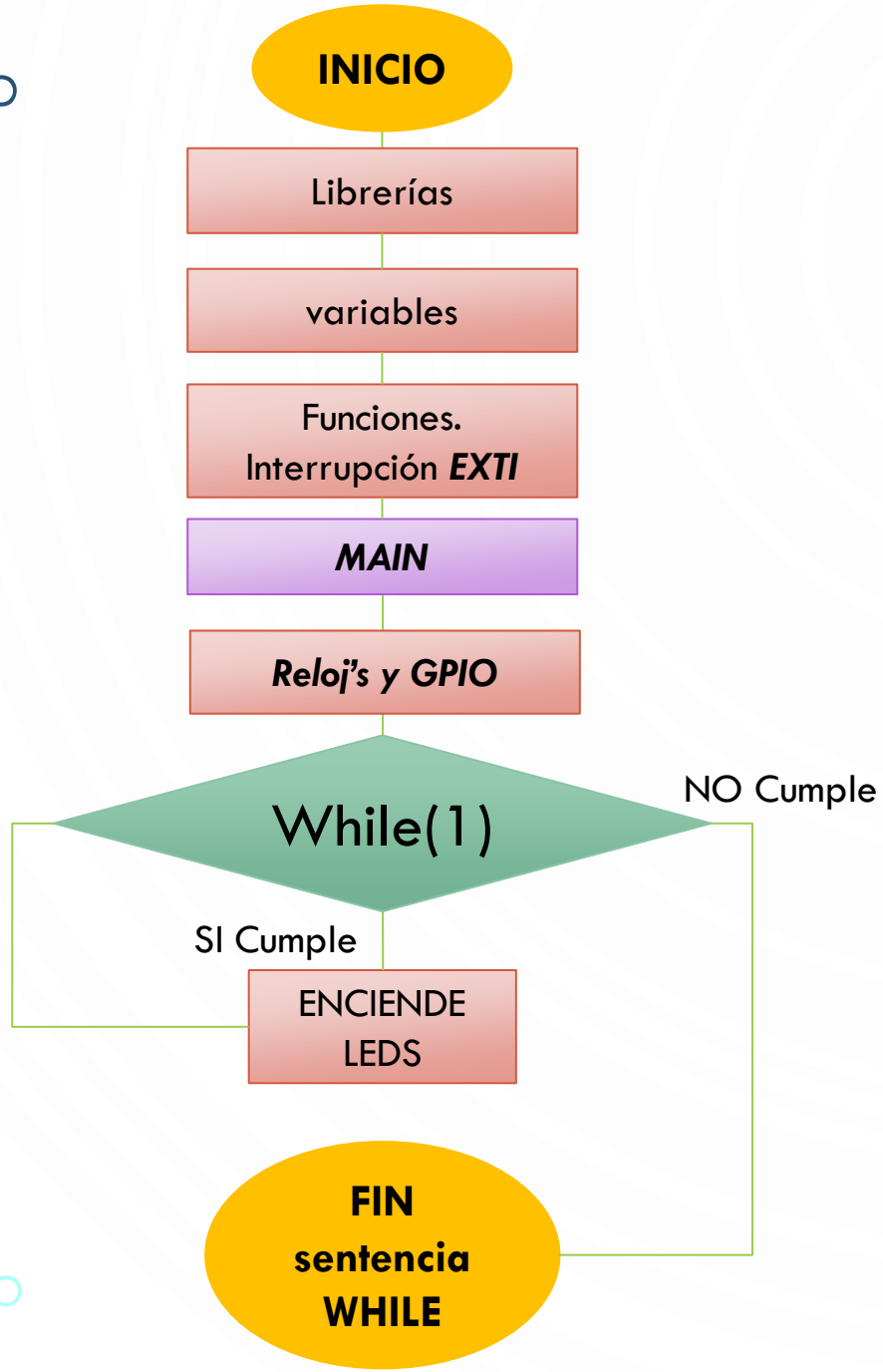
0: Falling trigger disabled (for Event and Interrupt) for input line

1: Falling trigger enabled (for Event and Interrupt) for input line.

11.5 Manual ARM.

Hardware interrupt selection To configure a line as interrupt sources, use the following procedure:

1. Configure the corresponding mask bit (EXTI_IMR)
2. Configure the Trigger selection bits of the interrupt lines (EXTI_RTSR and EXTI_FTSR)
3. Configure the enable and mask bits that control the NVIC IRQ channel mapped to the external interrupt controller (EXTI) so that an interrupt coming from one of the 24 lines can be correctly acknowledged.



main.cpp

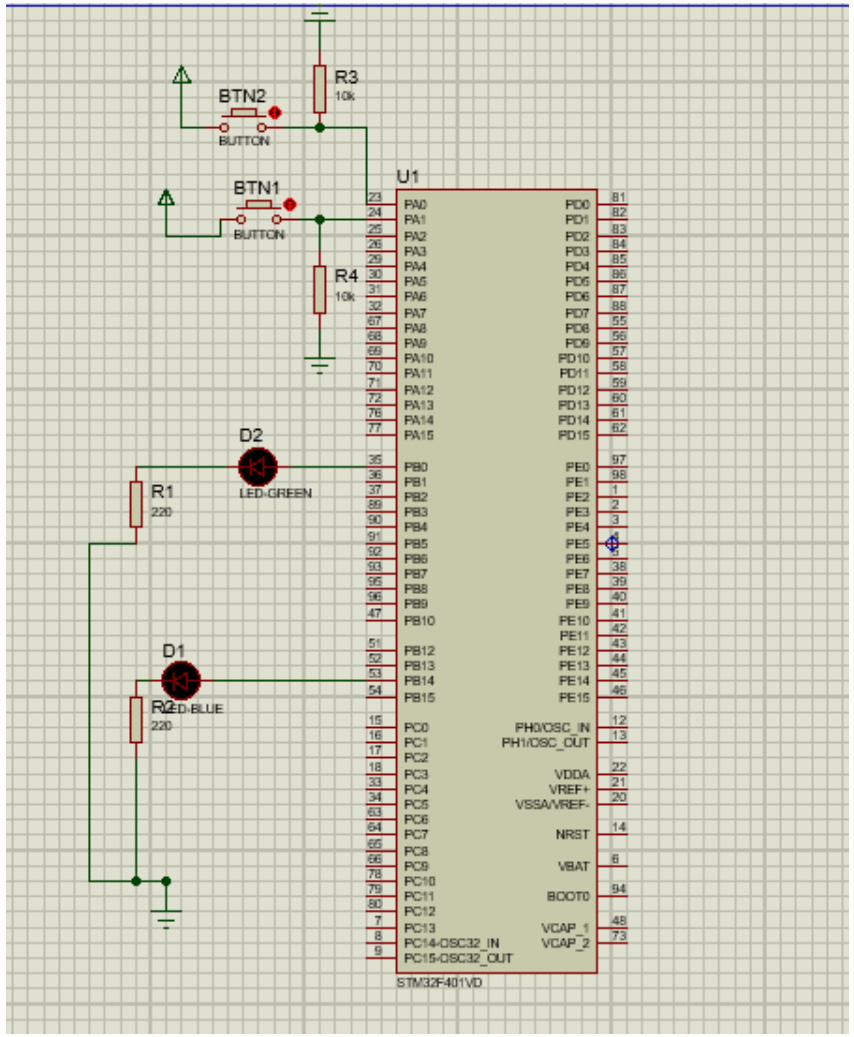
```cpp
#include <stdio.h>
#include "stm32f7xx.h"

int main(void){

  RCC -> AHB1ENR |= 0X2; //PUERTO B
  RCC -> APB2ENR |= 0X4000; //HABILITAR EL SYSCFG

  GPIOB -> MODER |= 0X10004001; //COLOCAR EN SALIDA PARA ENCENDER
  GPIOB -> OTYPER = 0X0; //PUSH PULL
  GPIOB -> OSPEEDR |= 0X10004001; //VELOCIDAD MEDIA
  GPIOB -> PUPDR |= 0X10004001; //PULL UP

  SYSCFG -> EXTICR[0] &= 0x1;
  EXTI->IMR = 0X1;
  EXTI->FTSR = 0X1;
  NVIC_EnableIRQ(EXTI0_IRQn);

  while(1){GPIOB -> ODR |= 1;

  }
}
extern "C"
{
  void EXTI0_IRQHandler(void)
  {
    EXTI->PR = 0X1;//Bandera que inicia la interrupcion
    GPIOB -> ODR ^= 0X4000;
    for(int i = 0; i < 500000; i++){};
  }
}
```

```
////////////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include "stm32f4xx.h"

int main(void){

  RCC -> AHB1ENR |= 0X2; //PUERTO B
  RCC -> APB2ENR |= 0X4000; //HABILITAR EL SYSCFG

  GPIOB -> MODER |= 0X10004001; //COLOCAR EN SALIDA PARA ENCENDER LOS LEDS
  GPIOB -> OTYPER = 0X0; //PUSH PULL
  GPIOB -> OSPEEDR |= 0X10004001; //VELOCIDAD MEDIA
  GPIOB -> PUPDR |= 0X10004001; //PULL UP

  SYSCFG -> EXTICR[0] &= 0x0; //Puerto A
  EXTI->IMR = 0X03;  //Linea 0 y 1
  EXTI->FTSR = 0X3; //Linea 0 y 1
  NVIC_EnableIRQ(EXTI0_IRQn);
  NVIC_EnableIRQ(EXTI1_IRQn);

  while(1){
  }
}
extern "C"
{
  void EXTI0_IRQHandler(void)
  {
    EXTI->PR |= 0X1;//Bandera que inicia la interrupcion
    GPIOB -> ODR ^= 0X4000;
    for(int i = 0; i < 1000000; i++){};
  }
  void EXTI1_IRQHandler(void)
  {
    EXTI->PR |= 0X2;//Bandera que inicia la interrupcion
    GPIOB -> ODR ^= 0X1;
    for(int i = 0; i < 1000000; i++){};
  }
}
```

Modifique para que una interrupción sea por flanco de subida y la otra de bajada.

¿En que pines ingresan las interrupciones?

Modifique para incluir un tercer led activado por interrupción externa 5



```c
////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include "stm32f4xx.h"

int main(void){

  RCC -> AHB1ENR |= 0X2; //PUERTO B
  RCC -> APB2ENR |= 0X4000; //HABILITAR EL SYSCFG

  GPIOB -> MODER |= 0X00000555; //COLOCAR EN SALIDA PARA ENCENDER LOS LEDS
  GPIOB -> OTYPER = 0X0; //PUSH PULL
  GPIOB -> OSPEEDR |= 0X00000555; //VELOCIDAD MEDIA
  GPIOB -> PUPDR |= 0X00000555; //PULL UP

  SYSCFG -> EXTICR[3] |= 0X0022;     // PINES EXTI3  C  0000 0000 0010 0010  13 12
  SYSCFG -> EXTICR[2] |= 0X2220;     // PINES EXTI2  C  0010 0010 0010 0000  11 10 9 8
  EXTI->IMR = 0X3E00;                // IMR    0011 1110 0000 0000
  EXTI->RTSR = 0X3E00;               // RISING EDGE     0011 1110 0000 0000

  NVIC_EnableIRQ(EXTI9_5_IRQn);      // LLAMADO FUNCION NVIC DE LOS PINES  9 AL 5
  NVIC_EnableIRQ(EXTI15_10_IRQn);    // LLAMADO FUNCION NVIC DE LOS PINES 15 AL 10

  while(1){
  }
}
```

```c
extern "C"
{
void EXTI9_5_IRQHandler(void){
  EXTI->PR |= (1UL<<9);
  GPIOB -> ODR = (1UL<<0);
  for(int i = 0; i < 1000000; i++){};
}

void EXTI15_10_IRQHandler(void){              // FUNCION EXTI15_10

  if(EXTI->PR & (1UL<<10)){
    EXTI->PR |= (1UL<<10);
    GPIOB -> ODR = (1UL<<1);
    for(int i = 0; i < 1000000; i++){};
  }
  else if(EXTI->PR & (1UL<<11)){
    EXTI->PR |= (1UL<<11);
    GPIOB -> ODR = (1UL<<2);
    for(int i = 0; i < 1000000; i++){};
  }
  else if(EXTI->PR & (1UL<<12)){
    EXTI->PR |= (1UL<<12);
    GPIOB -> ODR = (1UL<<5);
    for(int i = 0; i < 1000000; i++){};
  }
   else if(EXTI->PR & (1UL<<13)){
    EXTI->PR |= (1UL<<13);
    GPIOB -> ODR = (1UL<<2);
    for(int i = 0; i < 1000000; i++){};
  }
}
}
```

TAREA

Implementar un contador programable de 0-9 salida por display 7 segmentos.
El valor de inicio del contador debe ingresarse por medio de un teclado por entradas de interrupción.
Se debe poder configurar modo ascendente o descendente, con tecla de inicio y pausa, también usando interrupciones.