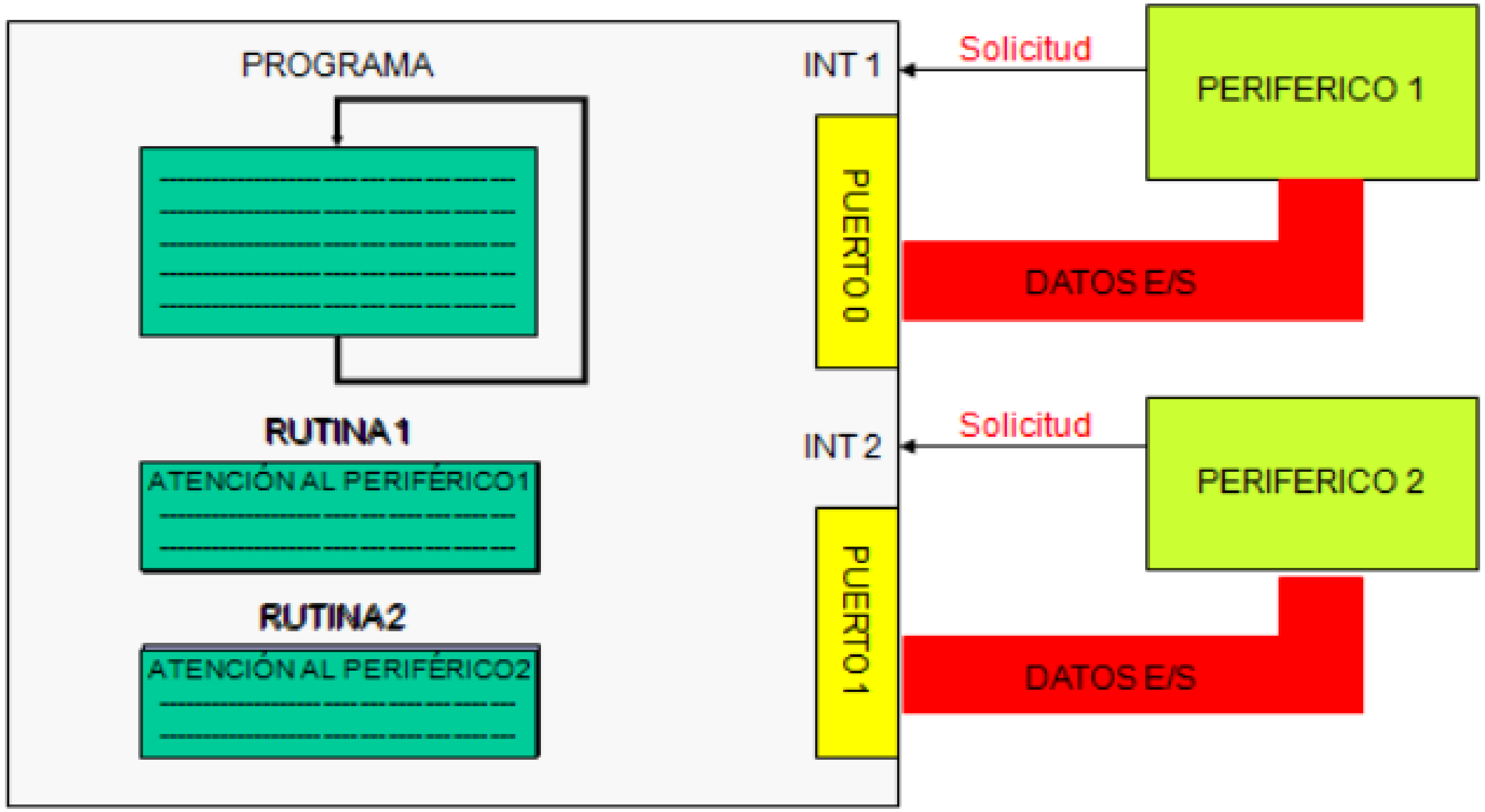
MICROS 32 BITS
STM - Interrupciones

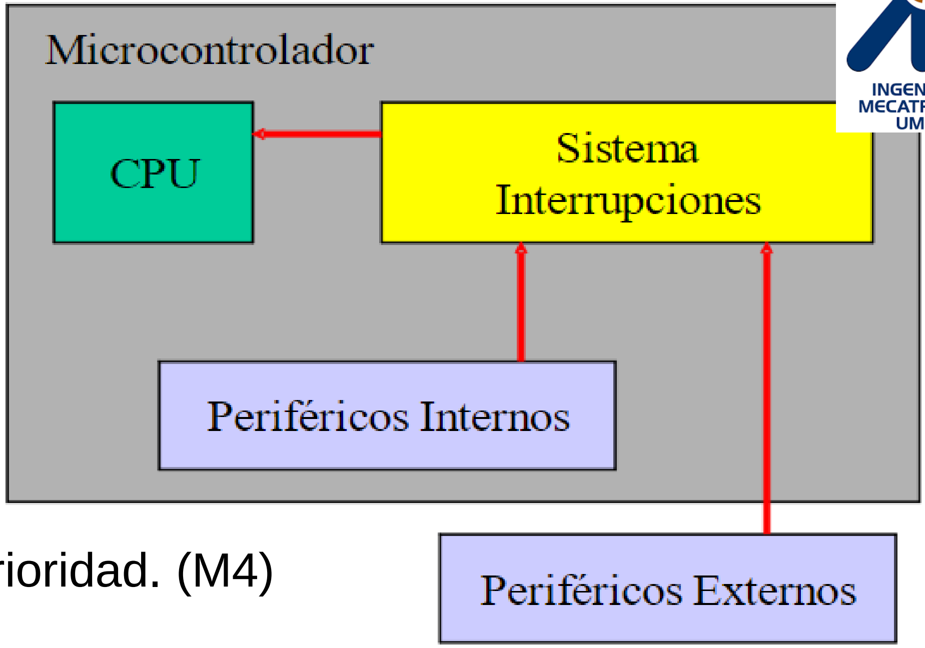
ROBINSON JIMENEZ MORENO

UNIVERSIDAD MILITAR NUEVA GRANADA

INGENIERÍA MECATRÓNICA UMNG

INTERRUPCION





Interrupciones en ARM Cortex-M

● Soporta hasta 240 fuentes de interrupción con 256 niveles de prioridad. (M4)

● Las mas básicas son:

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.

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

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

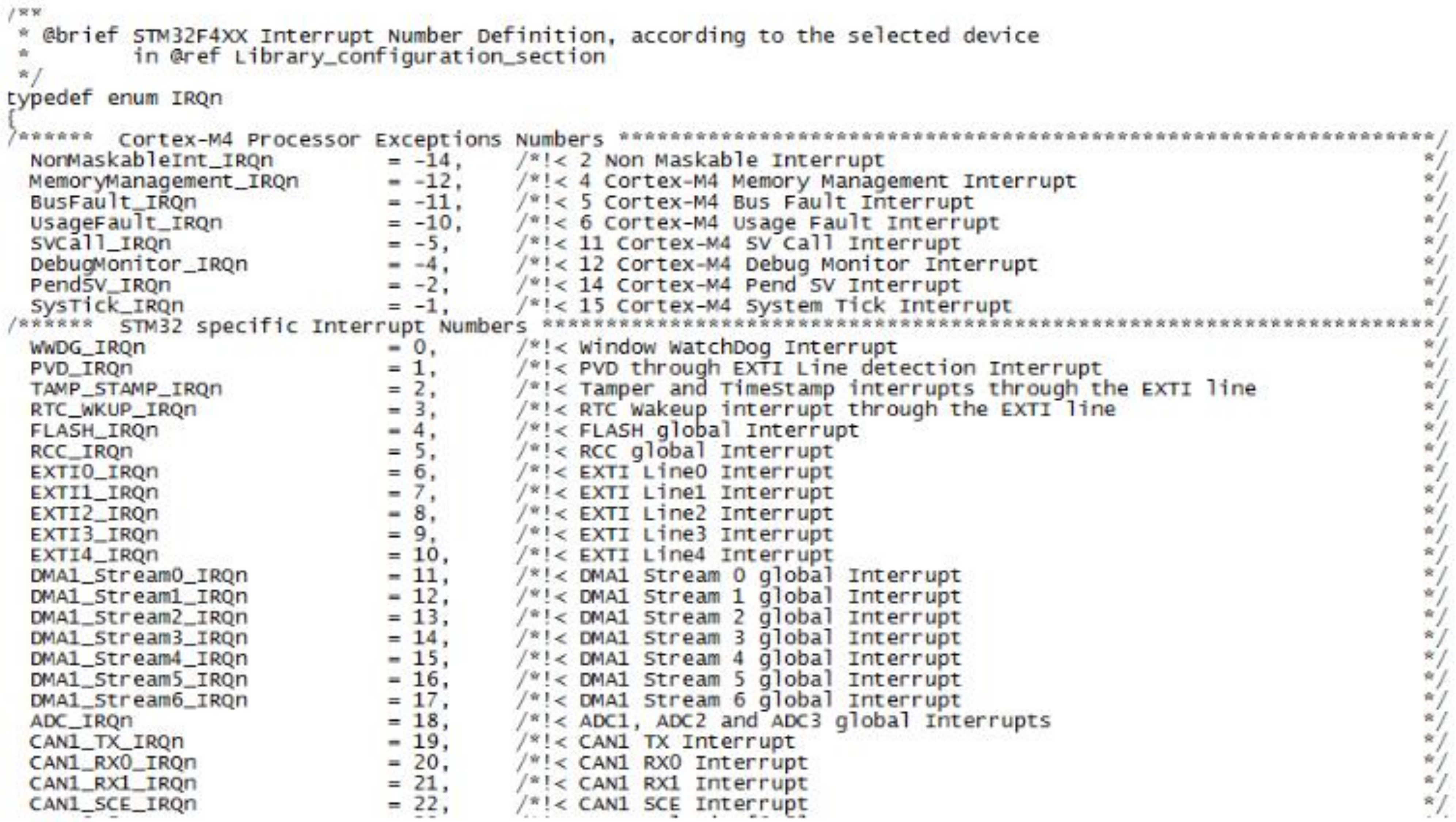
Si Pi > Pa se ejecuta el manejador de la interrupción

Las interrupciones puede ser habilitadas / inhibidas

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

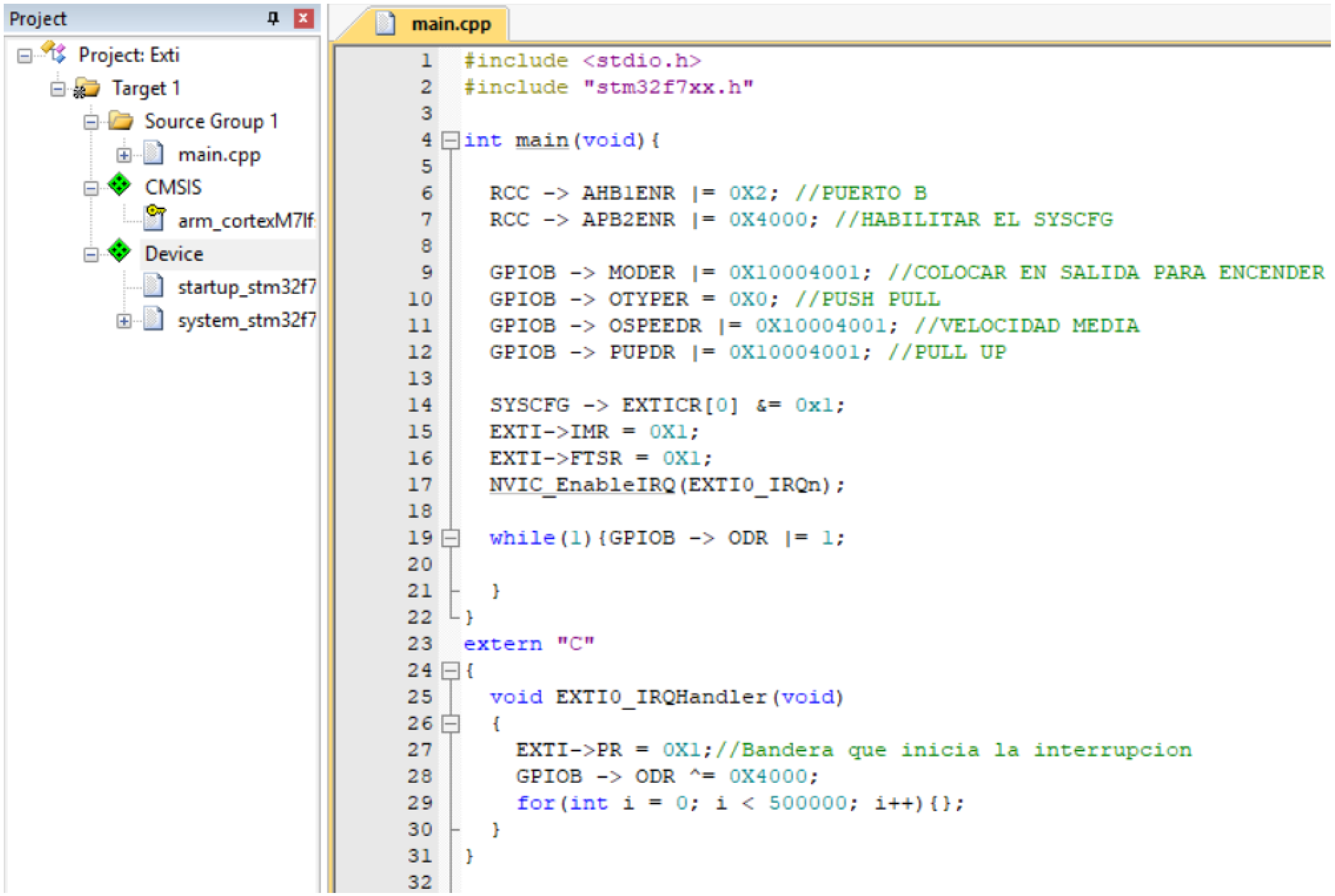
La primera interrupción de dispositivo tiene el IRQn = 0, se utilizan valores negativos de IRQn para las “processor core exceptions”. El fichero stm32f4xx.h contiene el Interrupt Number Definition: proporciona los números de interrupción (IRQn) para todas las interrupciones y excepciones.

- 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                                                */
```

...



```
#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++){};
  }
}

```

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**.

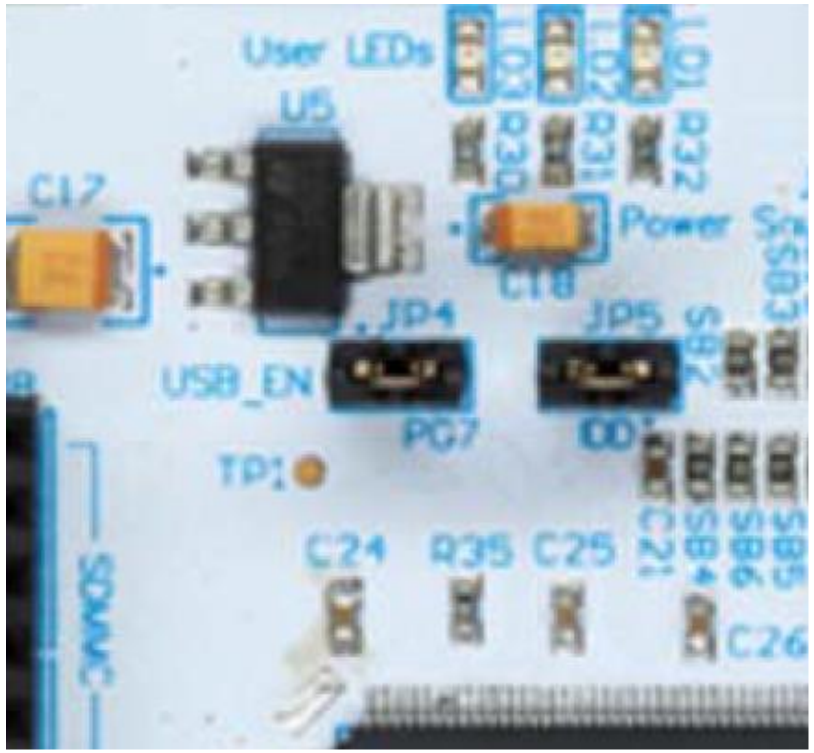
You have to know that **PB0** is also connected to **Line0** and **PC0** also and so on. This is for all pins on board, All **Px0** (where x is GPIO name) pins are connected to **Line0** and let's say all Px3 are connected to **Line3** on the Interrupt channel.

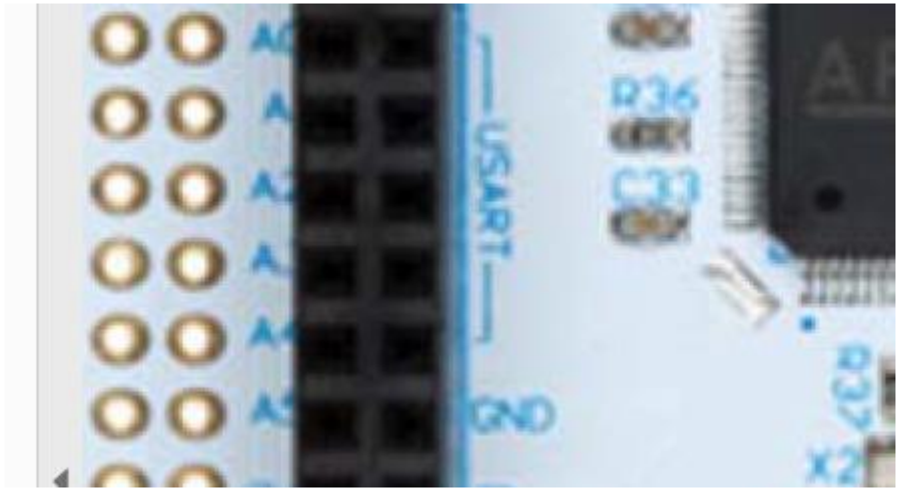
All pins with same number are connected to line with same number. They are multiplexed to one line.

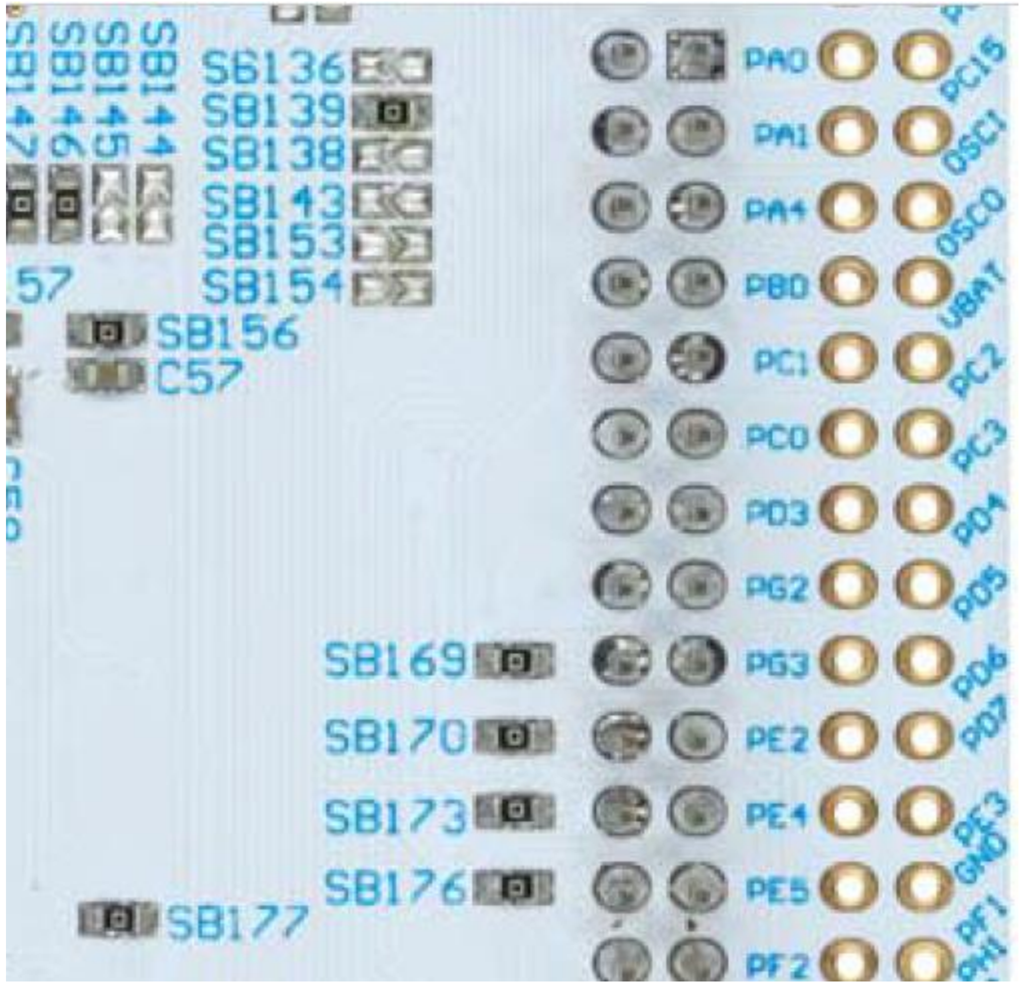
IMPORTANT: You can not use two pins on one line simultaneously:

- **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.


User LEDs
LD3
LD2
LD1
U5
R30
R31
R32
C17
Power
C18
JP4
JP5
USB_EN
PG7
IDD
TP1
C24
R35
C25
C21
SB4
SB6
SB5
C26
SDMMC


USART
R36
C33
GND
R37
X2


SB147
SB146
SB145
SB144
SB136
SB139
SB138
SB143
SB153
SB154
SB156
C57
SB169
SB170
SB173
SB176
SB177
PA0
PA1
PA4
PB0
PC1
PC0
PD3
PG2
PG3
PE2
PE4
PE5
PF2
PC15
OSCI
OSCO
VBAT
PC2
PC3
PD4
PD5
PD6
PD7
PE3
GND
PF1

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:

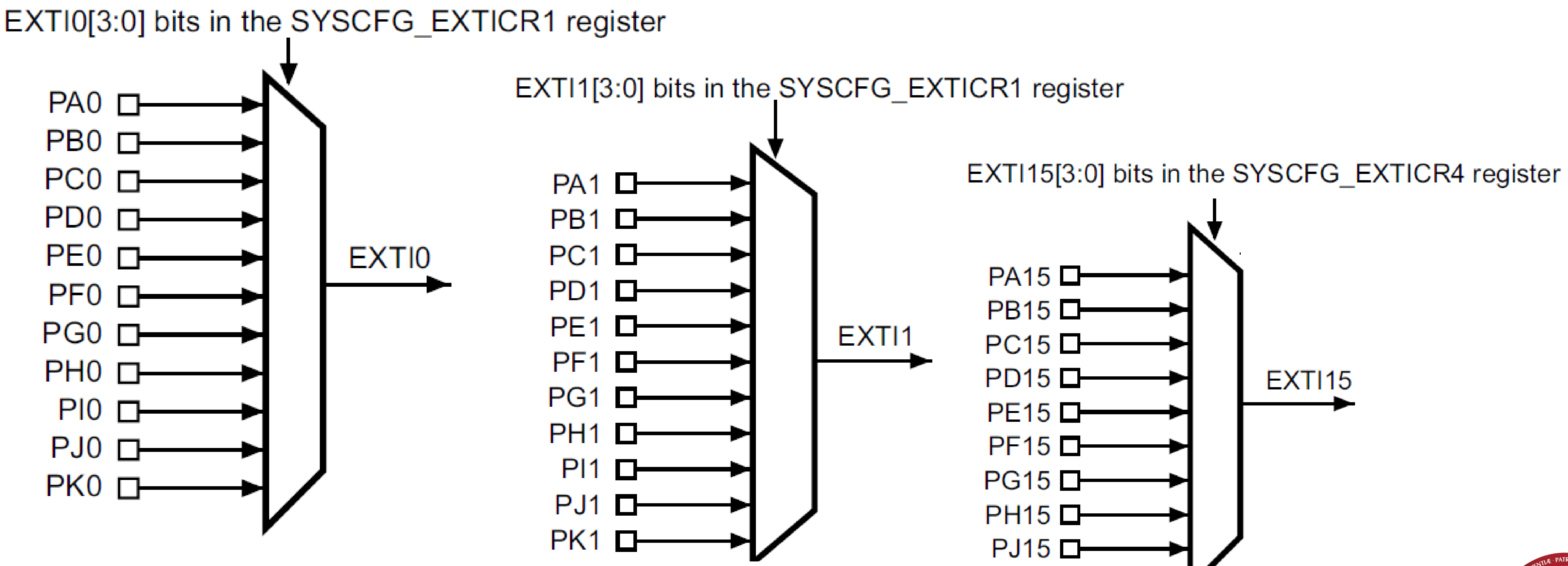
Irq	Handler	Description
EXTI0_IRQn	EXTI0_IRQHandler	Handler for pins connected to line 0
EXTI1_IRQn	EXTI1_IRQHandler	Handler for pins connected to line 1
EXTI2_IRQn	EXTI2_IRQHandler	Handler for pins connected to line 2
EXTI3_IRQn	EXTI3_IRQHandler	Handler for pins connected to line 3
EXTI4_IRQn	EXTI4_IRQHandler	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

SYSCFG -> EXTICR[3]

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


EXTI0[3:0] bits in the SYSCFG_EXTICR1 register
PA0
PB0
PC0
PD0
PE0
PF0
PG0
PH0
PI0
PJ0
PK0
EXTI0
EXTI1[3:0] bits in the SYSCFG_EXTICR1 register
PA1
PB1
PC1
PD1
PE1
PF1
PG1
PH1
PI1
PJ1
PK1
EXTI1
EXTI15[3:0] bits in the SYSCFG_EXTICR4 register
PA15
PB15
PC15
PD15
PE15
PF15
PG15
PH15
PJ15
EXTI15

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.

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

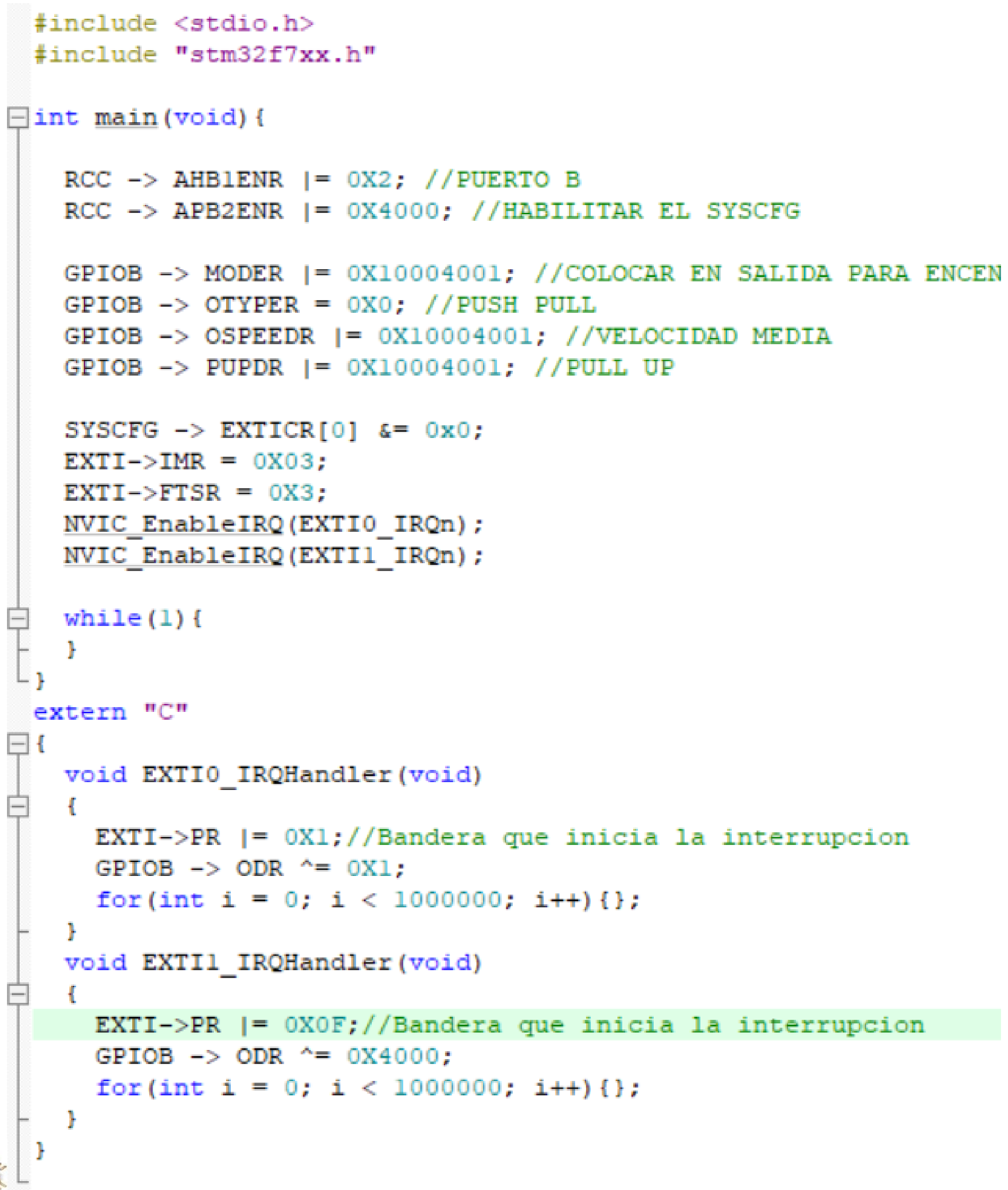
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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.



```
#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 ENCEN
  GPIOB -> OTYPER = 0X0; //PUSH PULL
  GPIOB -> OSPEEDR |= 0X10004001; //VELOCIDAD MEDIA
  GPIOB -> PUPDR |= 0X10004001; //PULL UP

  SYSCFG -> EXTICR[0] &= 0x0;
  EXTI->IMR = 0X03;
  EXTI->FTSR = 0X3;
  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 ^= 0X1;
    for(int i = 0; i < 1000000; i++){};
  }
  void EXTI1_IRQHandler(void)
  {
    EXTI->PR |= 0X0F;//Bandera que inicia la interrupcion
    GPIOB -> ODR ^= 0X4000;
    for(int i = 0; i < 1000000; i++){};
  }
}
```

INGENIERÍA MECATRÓNICA UMNG

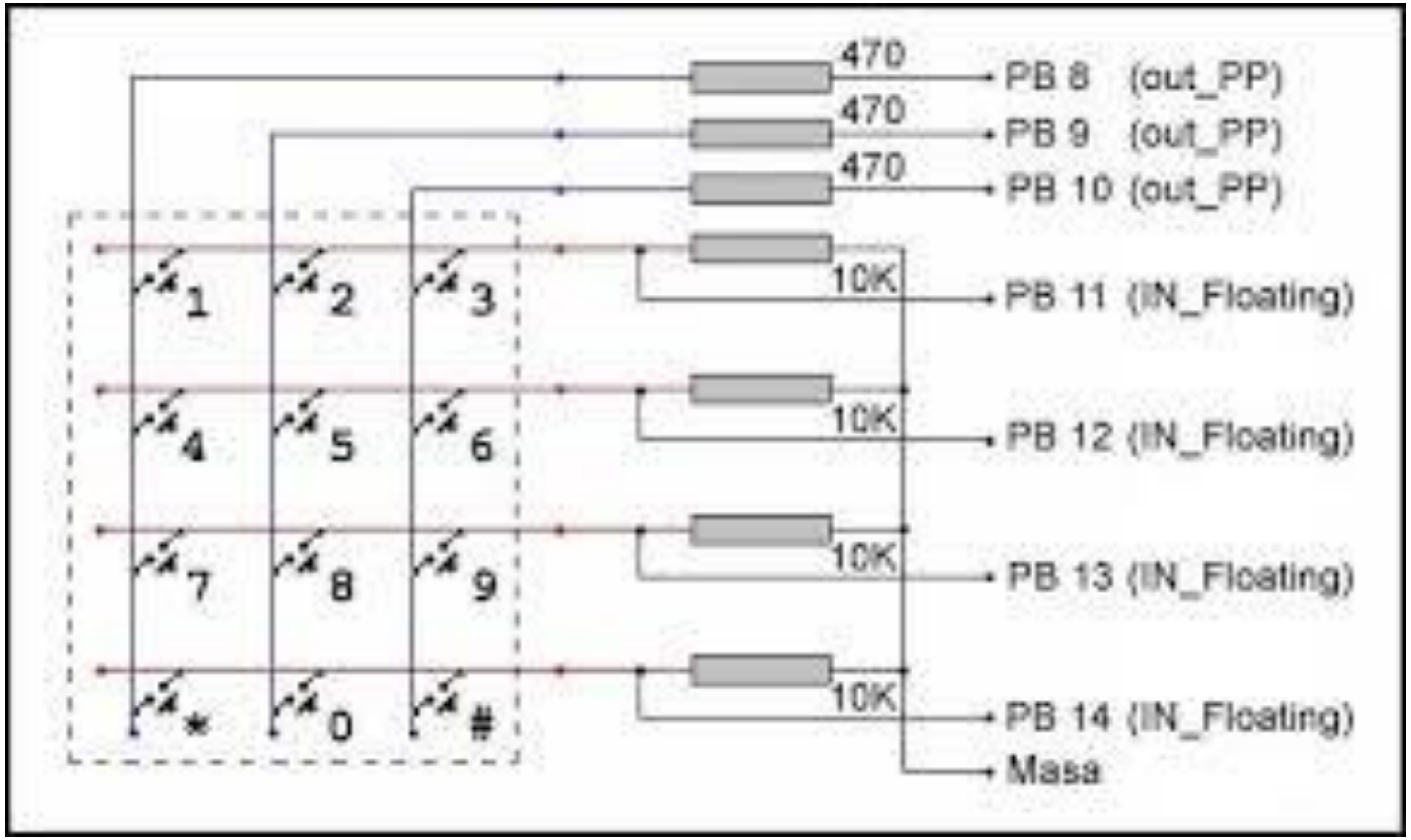
En que pines ingresan las interrupciones?

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

Modifique para incluir un tercer led activado por interrupción externa 5

UNIVERSIDAD MILITAR NUEVA GRANADA

MANEJAR TECLADO MATRICIAL MEDIANTE INTERRUPCION



TAREA

Implementar un contador programable de 0-9 salida por display 7 segmentos que lea un teclado por entradas de interrupción, debe poder configurar modo ascendente o descendente, con tecla de inicio y pausa. Una vez finalice la cuenta el número final deberá titilar cada segundo y medio.