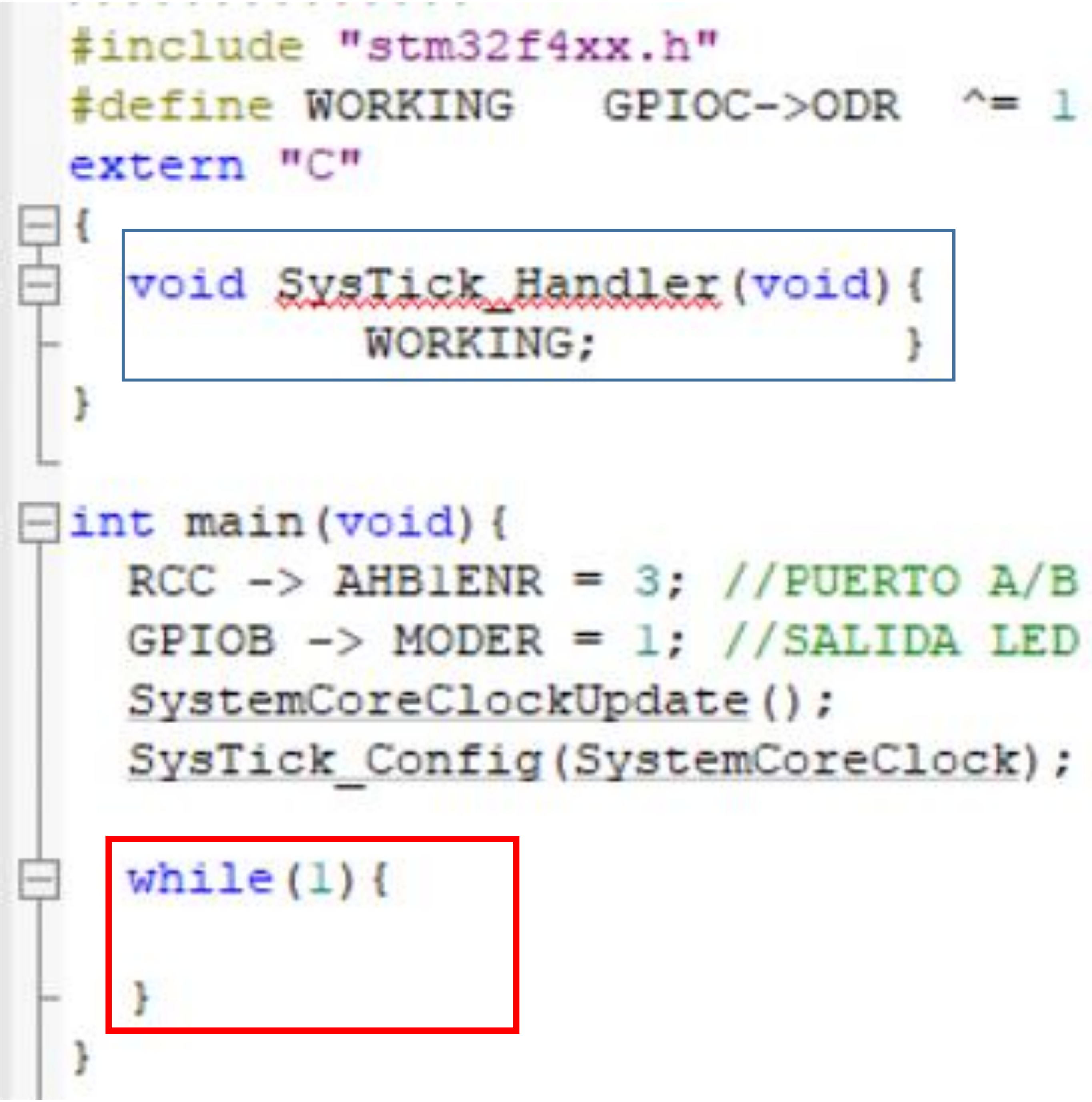
MICROS 32 BITS
STM – Interrupciones 1

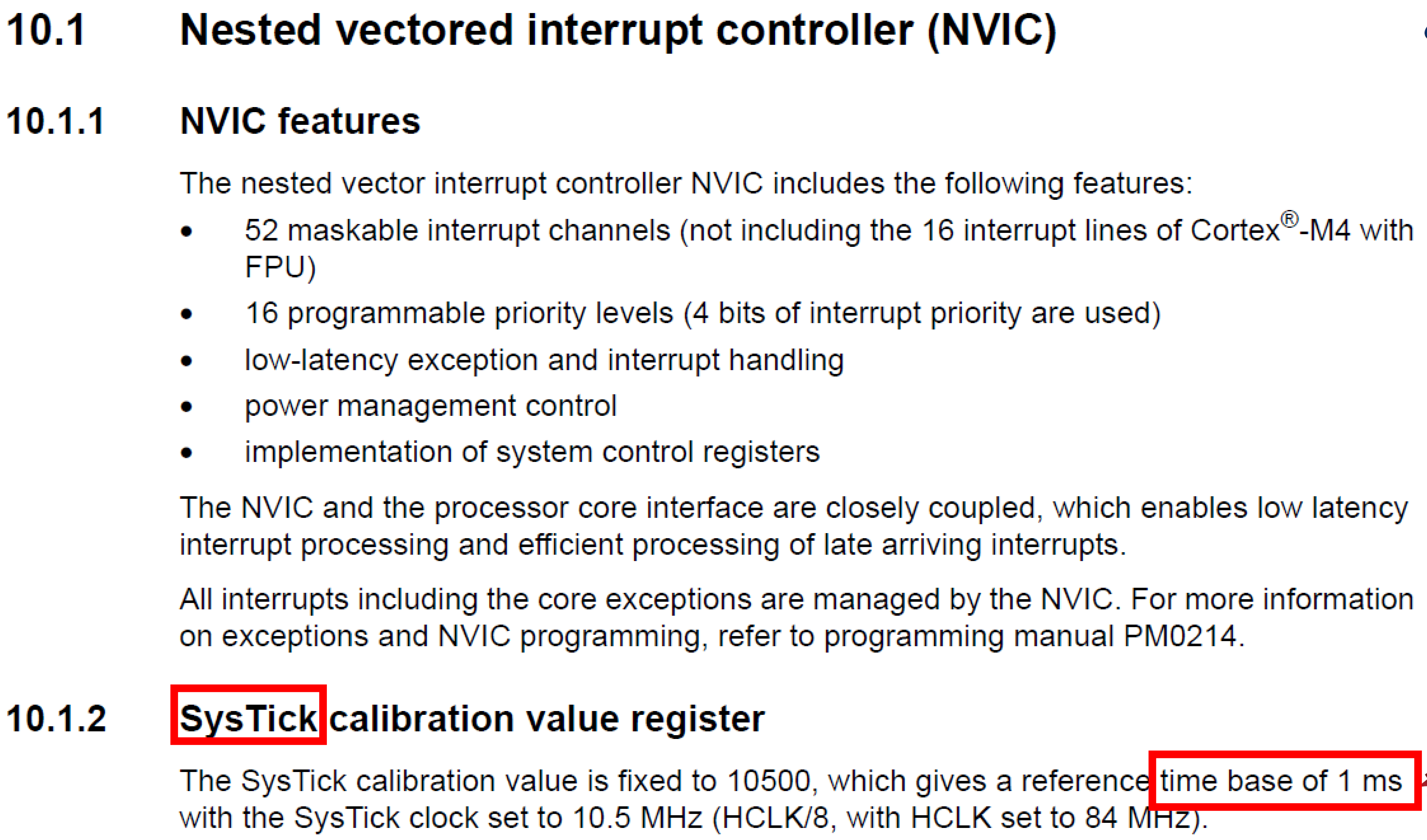
ROBINSON JIMENEZ MORENO

UNIVERSIDAD MILITAR
NUEVA GRANADA

INGENIERÍA
MECATRÓNICA
UMNG



```
#include "stm32f4xx.h"
#define WORKING   GPIOC->ODR  ^= 1
extern "C"
{
  void SysTick_Handler(void){
          WORKING;            }
}

int main(void){
  RCC -> AHB1ENR = 3; //PUERTO A/B
  GPIOB -> MODER = 1; //SALIDA LED
  SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock);

  while(1){


  }
}
```



10.1 Nested vectored interrupt controller (NVIC)

10.1.1 NVIC features

The nested vector interrupt controller NVIC includes the following features:

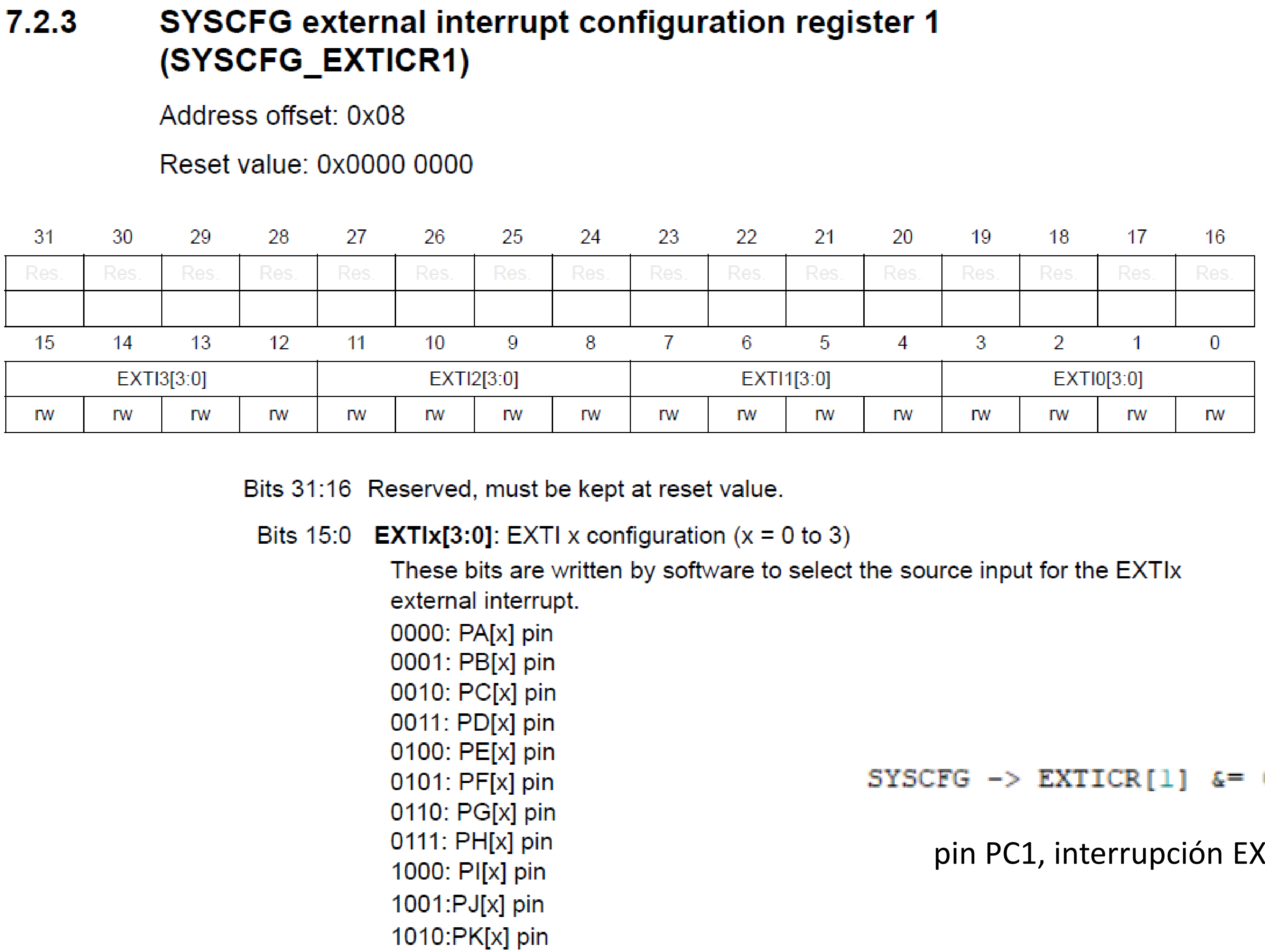
- 52 maskable interrupt channels (not including the 16 interrupt lines of Cortex®-M4 with FPU)
- 16 programmable priority levels (4 bits of interrupt priority are used)
- low-latency exception and interrupt handling
- power management control
- implementation of system control registers

The NVIC and the processor core interface are closely coupled, which enables low latency interrupt processing and efficient processing of late arriving interrupts.

All interrupts including the core exceptions are managed by the NVIC. For more information on exceptions and NVIC programming, refer to programming manual PM0214.

10.1.2 SysTick calibration value register

The SysTick calibration value is fixed to 10500, which gives a reference time base of 1 ms with the SysTick clock set to 10.5 MHz (HCLK/8, with HCLK set to 84 MHz).



7.2.3 SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

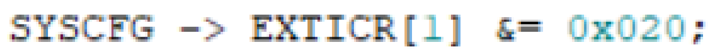
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

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.

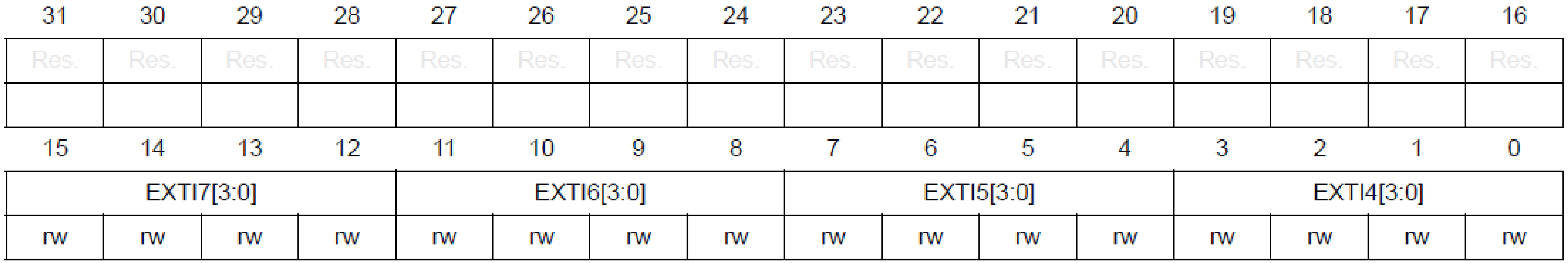
0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin
0111: PH[x] pin
1000: PI[x] pin
1001:PJ[x] pin
1010:PK[x] pin



```
SYSCFG -> EXTICR[1] &= 0x020;
```

pin PC1, interrupción EXTI1-PC

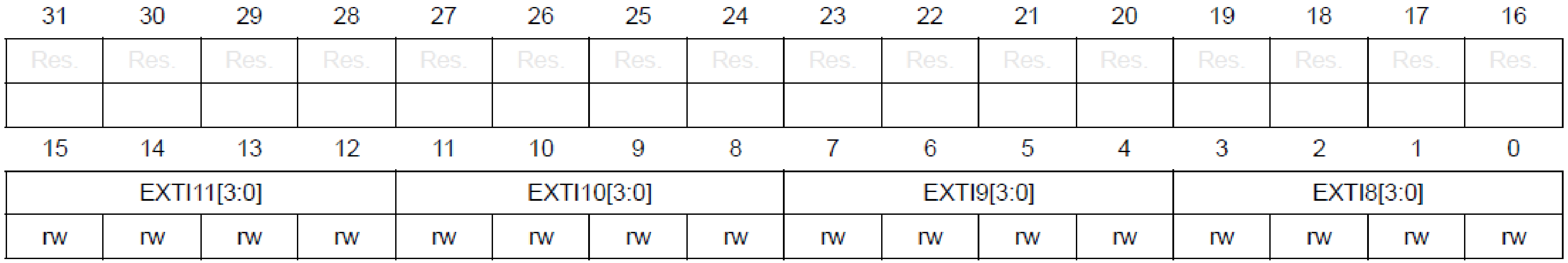
7.2.4 SYSCFG external interrupt configuration register 2 (SYSCFG_EXTICR2)



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
EXTI7[3:0]				EXTI6[3:0]				EXTI5[3:0]				EXTI4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

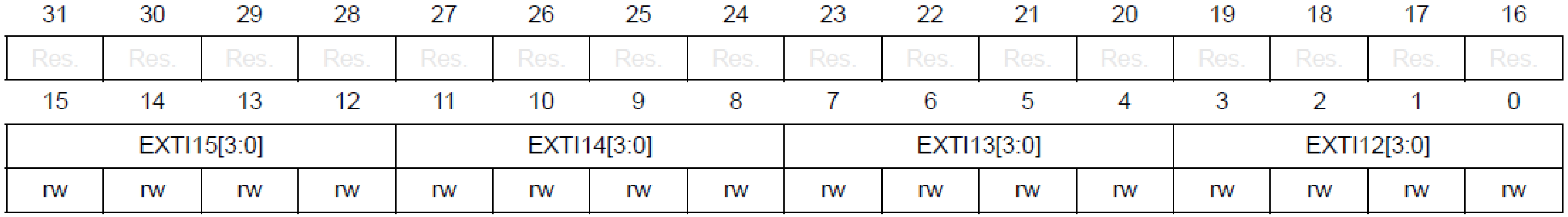
7.2.5 SYSCFG external interrupt configuration register 3 (SYSCFG_EXTICR3)



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
EXTI11[3:0]				EXTI10[3:0]				EXTI9[3:0]				EXTI8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

7.2.6 SYSCFG external interrupt configuration register 4 (SYSCFG_EXTICR4)

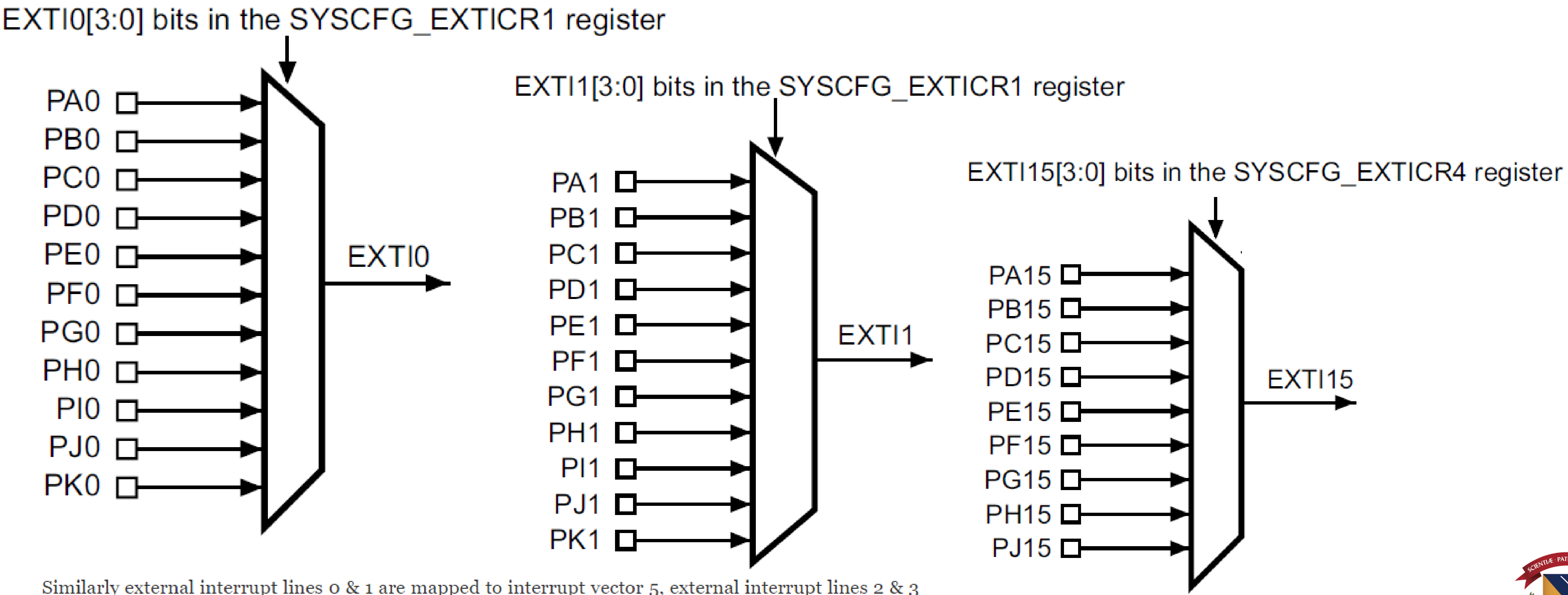


31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

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

External interrupt/event line mapping

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

EXTI1[3:0] bits in the SYSCFG_EXTICR1 register

EXTI15[3:0] bits in the SYSCFG_EXTICR4 register

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.

5.3.14 RCC APB2 peripheral clock enable register (RCC_APB2ENR)

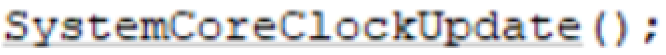
Address offset: 0x44

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



```
SystemCoreClockUpdate();
```

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

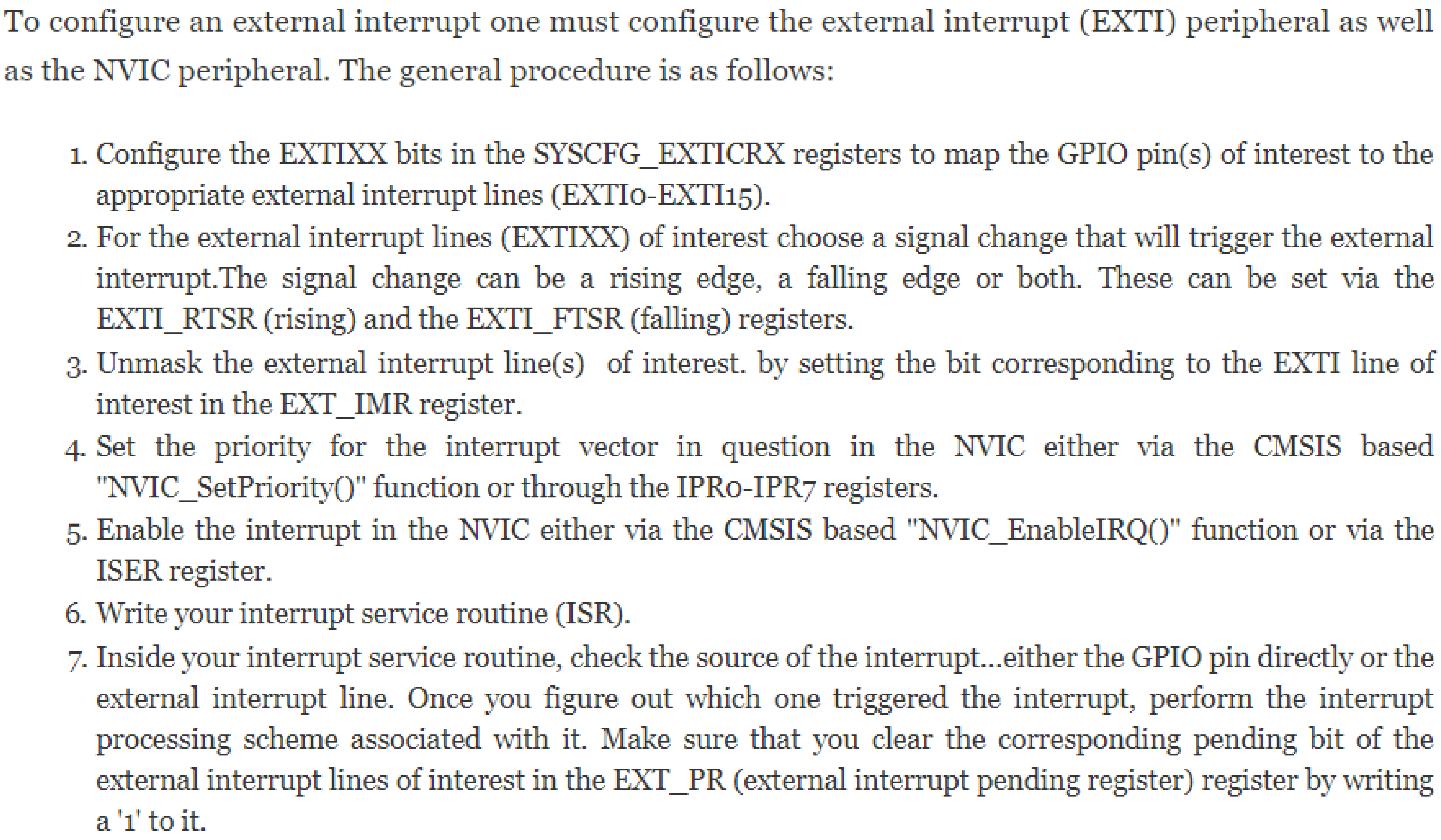
1: System configuration controller clock enabled

Bit 13 **SPI4EN:** SPI4 clock enable

This bit is set and cleared by software.

0: SPI4 clock disabled

1: SPI4 clock enabled



To configure an external interrupt one must configure the external interrupt (EXTI) peripheral as well as the NVIC peripheral. The general procedure is as follows:

1. Configure the EXTIXX bits in the SYSCFG_EXTICRX registers to map the GPIO pin(s) of interest to the appropriate external interrupt lines (EXTI0-EXTI15).
2. For the external interrupt lines (EXTIXX) of interest choose a signal change that will trigger the external interrupt.The signal change can be a rising edge, a falling edge or both. These can be set via the EXTI_RTSR (rising) and the EXTI_FTSR (falling) registers.
3. Unmask the external interrupt line(s) of interest. by setting the bit corresponding to the EXTI line of interest in the EXT_IMR register.
4. Set the priority for the interrupt vector in question in the NVIC either via the CMSIS based "NVIC_SetPriority()" function or through the IPR0-IPR7 registers.
5. Enable the interrupt in the NVIC either via the CMSIS based "NVIC_EnableIRQ()" function or via the ISER register.
6. Write your interrupt service routine (ISR).
7. Inside your interrupt service routine, check the source of the interrupt...either the GPIO pin directly or the external interrupt line. Once you figure out which one triggered the interrupt, perform the interrupt processing scheme associated with it. Make sure that you clear the corresponding pending bit of the external interrupt lines of interest in the EXT_PR (external interrupt pending register) register by writing a '1' to it.

http://hertaville.com/external-interrupts-on-the-stm32f0.html

11.9.3 Rising trigger selection register (EXTI_RTSR)

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.	TR23	TR22	TR21	TR20	TR19	TR18	TR17	TR16
								rw	rw	rw	rw	rw	rw	rw	rw

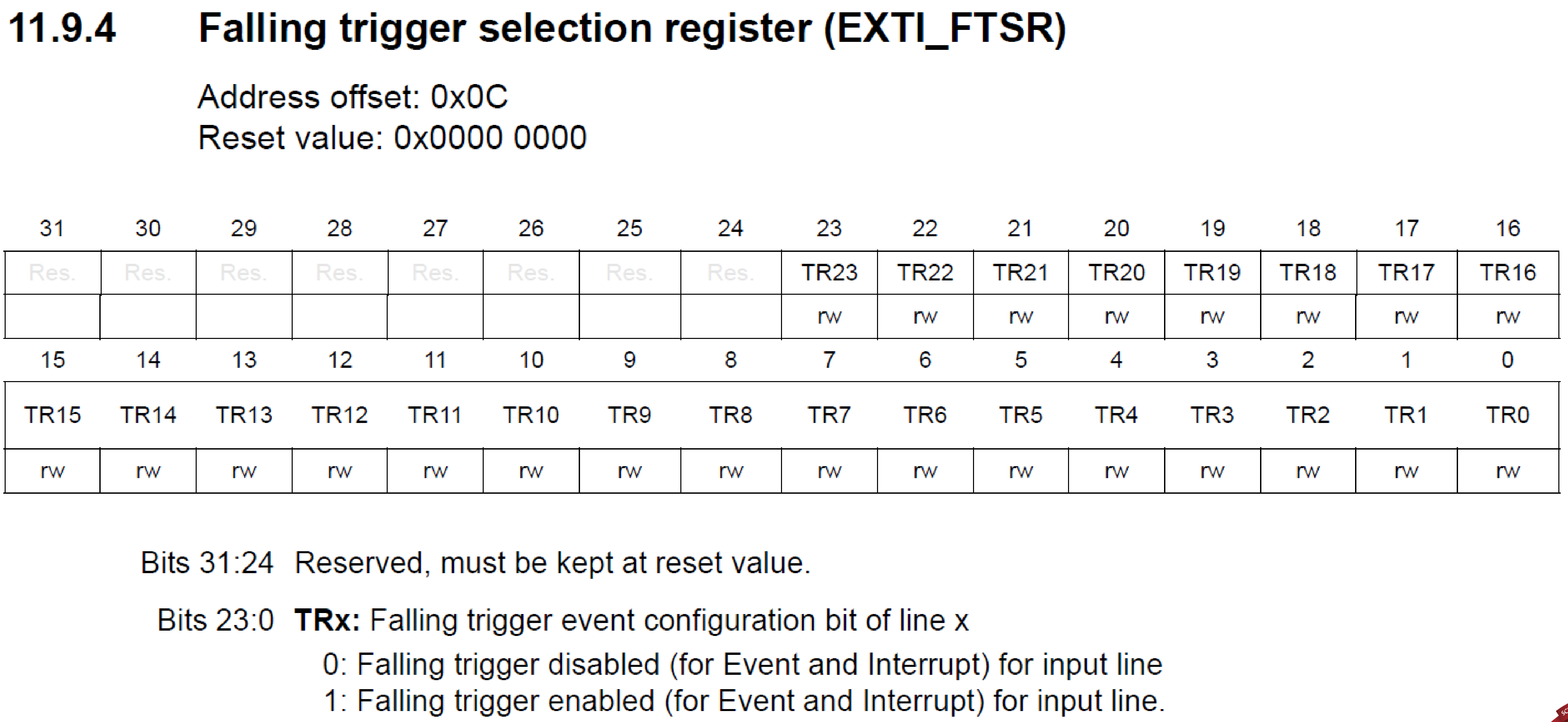
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

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



11.9.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
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.	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.

11.9.6 Pending register (EXTI_PR)

Address offset: 0x14
Reset value: undefined

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

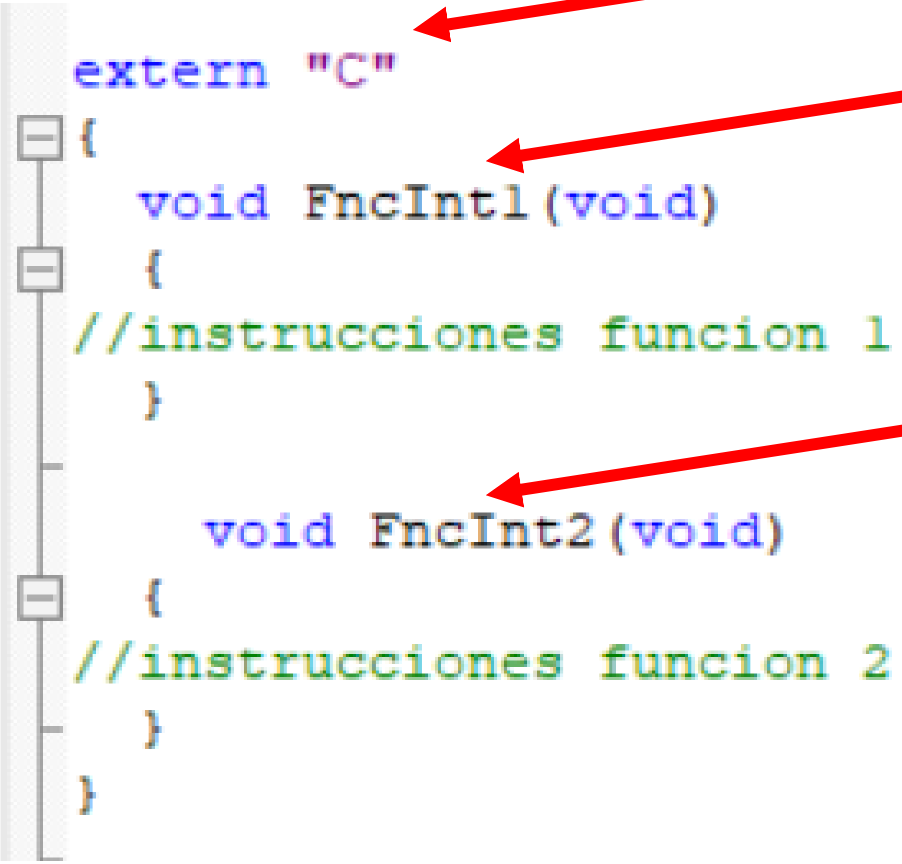
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

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

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’.

GENERALIDADES DEL CODIGO EN LAS FUNCIONES DE INTERRUPCION.

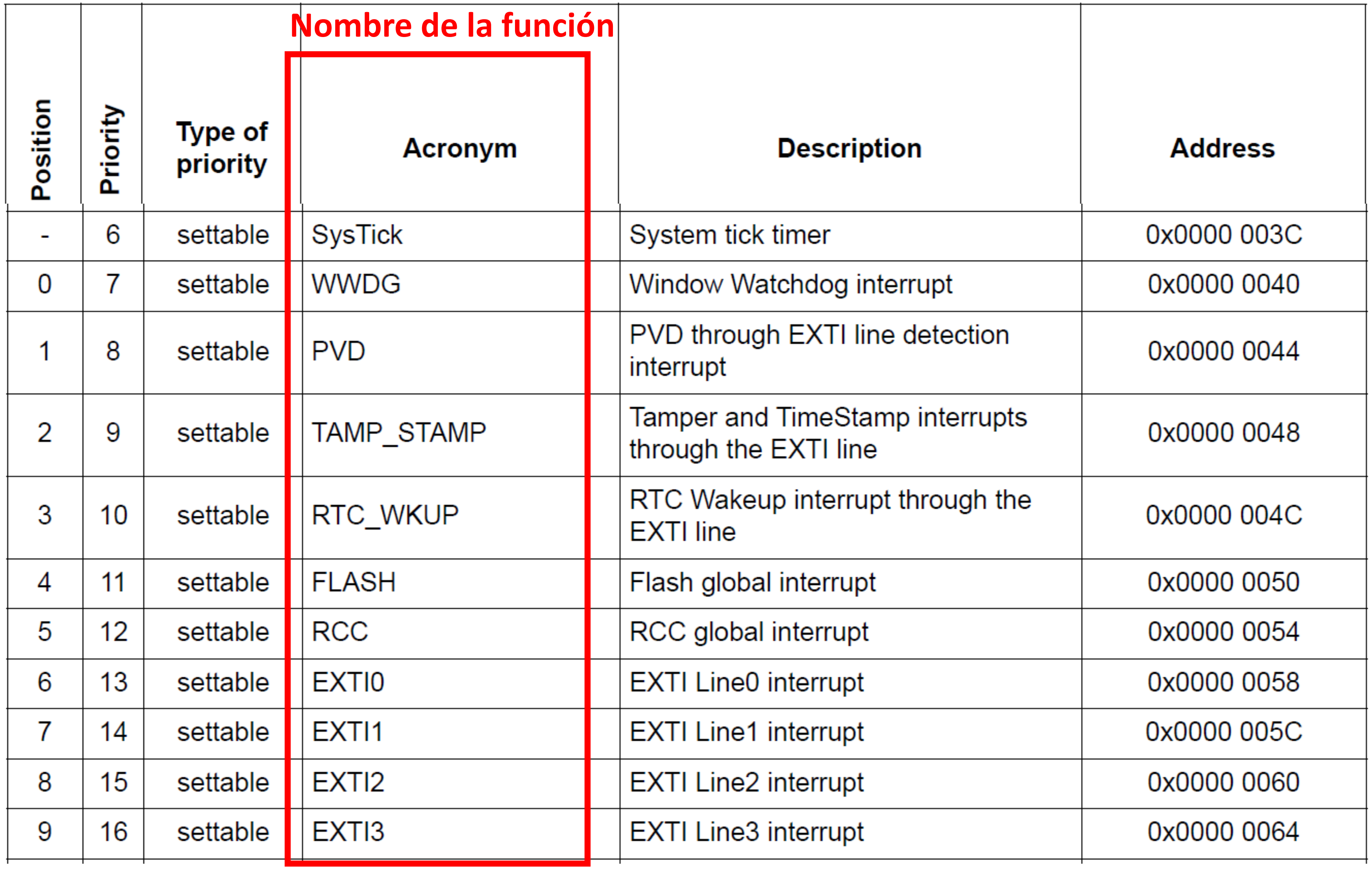


```
extern "C"
{
  void FncInt1(void)
  {
//instrucciones funcion 1
  }

    void FncInt2(void)
  {
//instrucciones funcion 2
  }
}
```

Debe cobijar todas las funciones

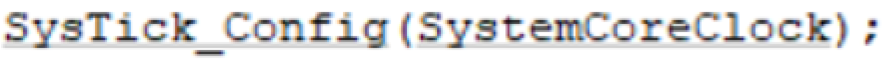
Validar el nombre de la funcion

NO debe existir mas de una
funcion con el mismo nombre



Nombre de la función

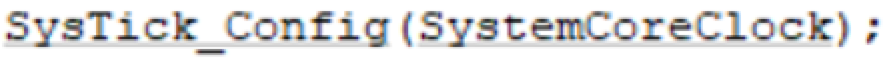
Position	Priority	Type of priority	Acronym	Description	Address
-	6	settable	SysTick	System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	PVD	PVD through EXTI line detection interrupt	0x0000 0044
2	9	settable	TAMP_STAMP	Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	RTC_WKUP	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



```
SysTick_Config(SystemCoreClock);
```

10.1.2 SysTick calibration value register

The SysTick calibration value is fixed to 10500, which gives a reference time base of 1 ms with the SysTick clock set to 10.5 MHz (HCLK/8, with HCLK set to 84 MHz).

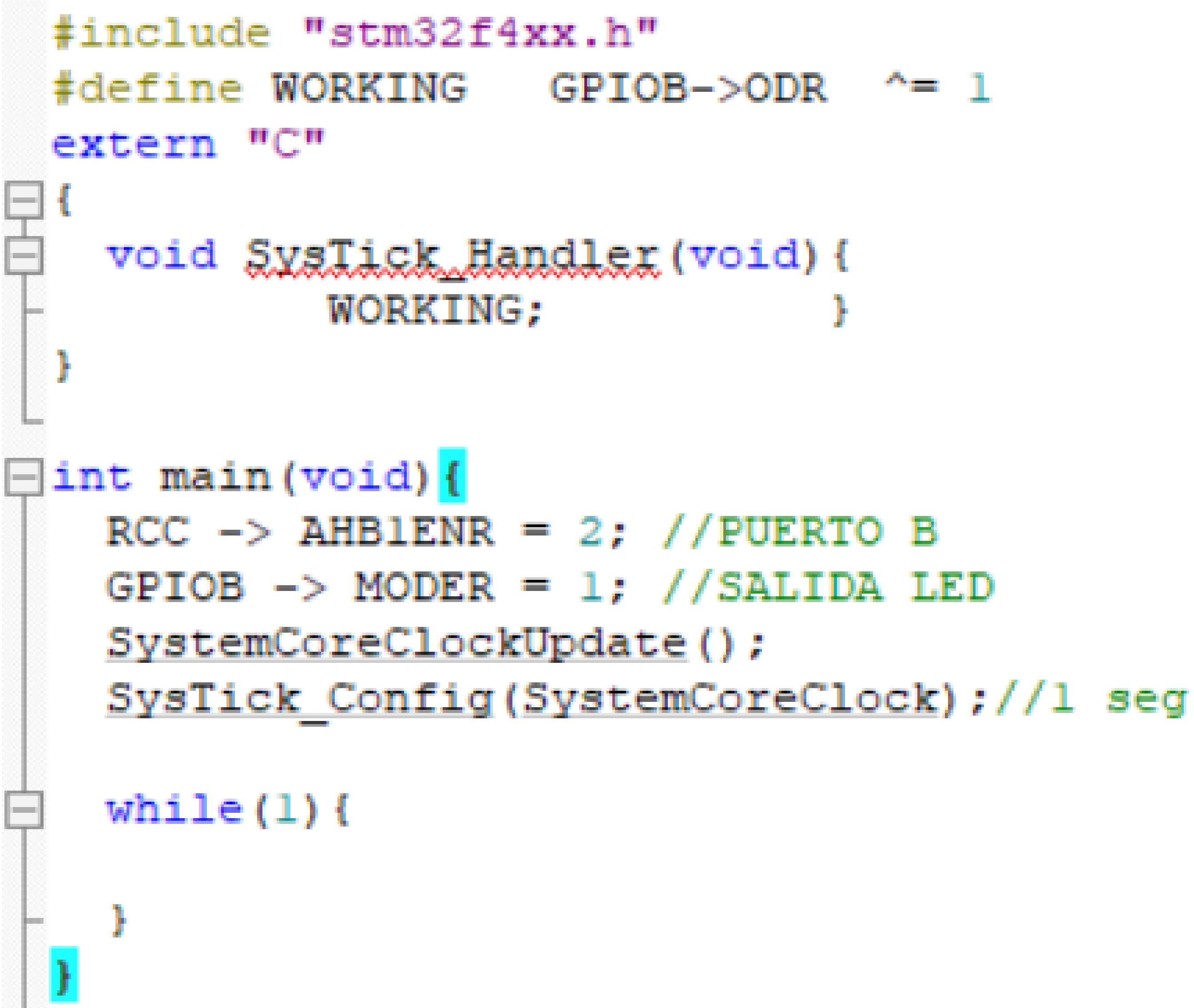


```
SysTick_Config(SystemCoreClock);
```

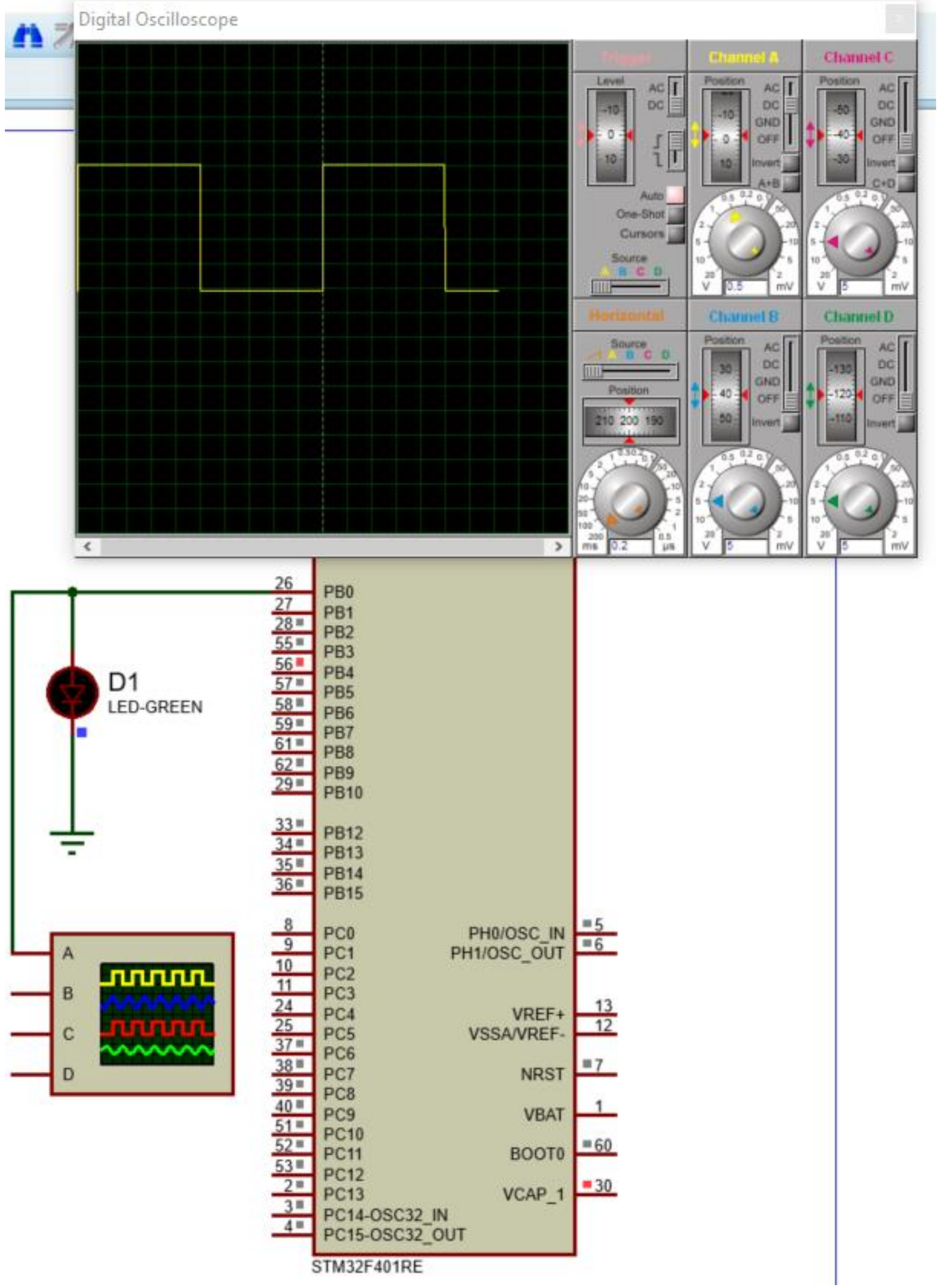
El SysTick Counter Clock es el reloj que le llega al temporizador SysTick, que en el STM32F4 es 168 Mhz.

El valor de recarga es el parámetro que le pasamos a la función SysTick_Config(), que no debe exceder 0xFFFFFF.

```
void SysTick_Handler(void)
{

        // aquí el código de la rutina de servicio del timer

}
```



```
#include "stm32f4xx.h"
#define WORKING   GPIOB->ODR  ^= 1
extern "C"
{
  void SysTick_Handler(void){
          WORKING;          }
}

int main(void){
  RCC -> AHB1ENR = 2; //PUERTO B
  GPIOB -> MODER = 1; //SALIDA LED
  SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock);//1 seg

  while(1){

  }
}
```

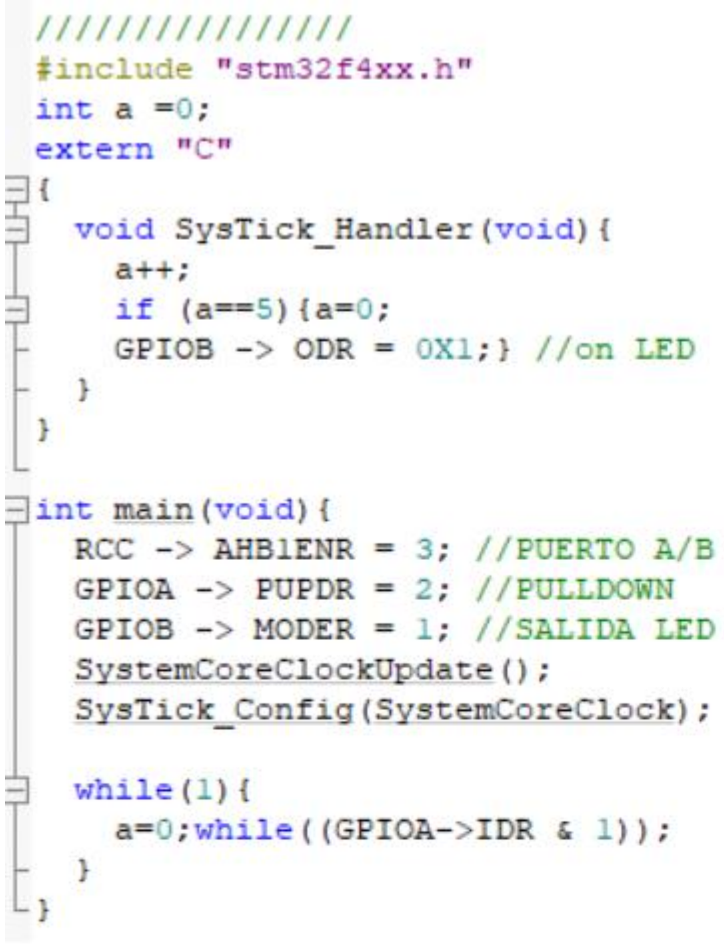


EJERCICIO:

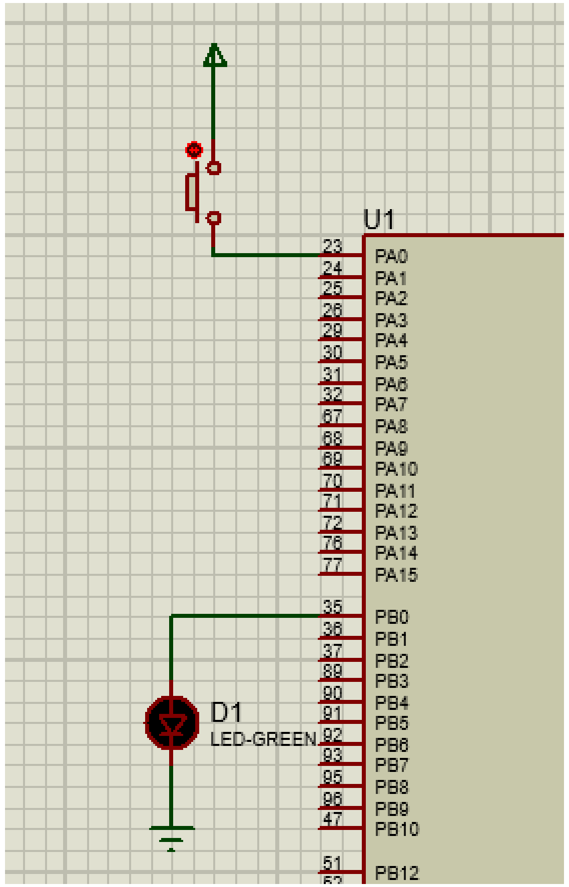
Generar una señal cuadrada de frecuencia 5Hz

Ejemplo en clase:

Emplear el pulsador para encender un led, si el pulso se mantiene mínimo 5 segundos.

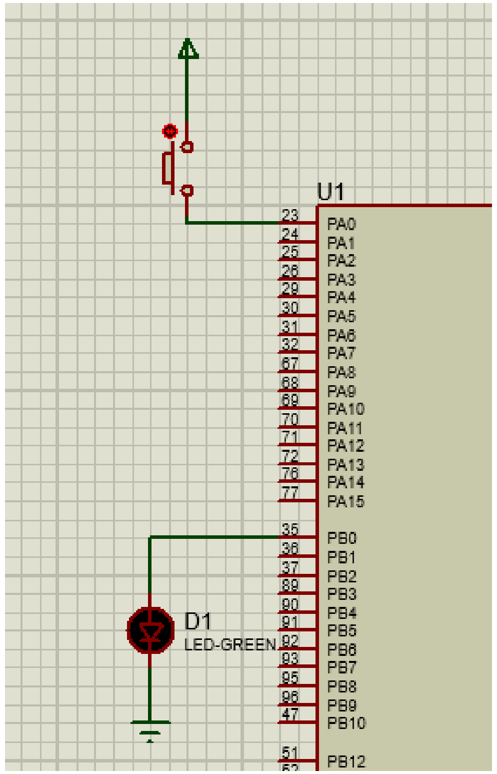


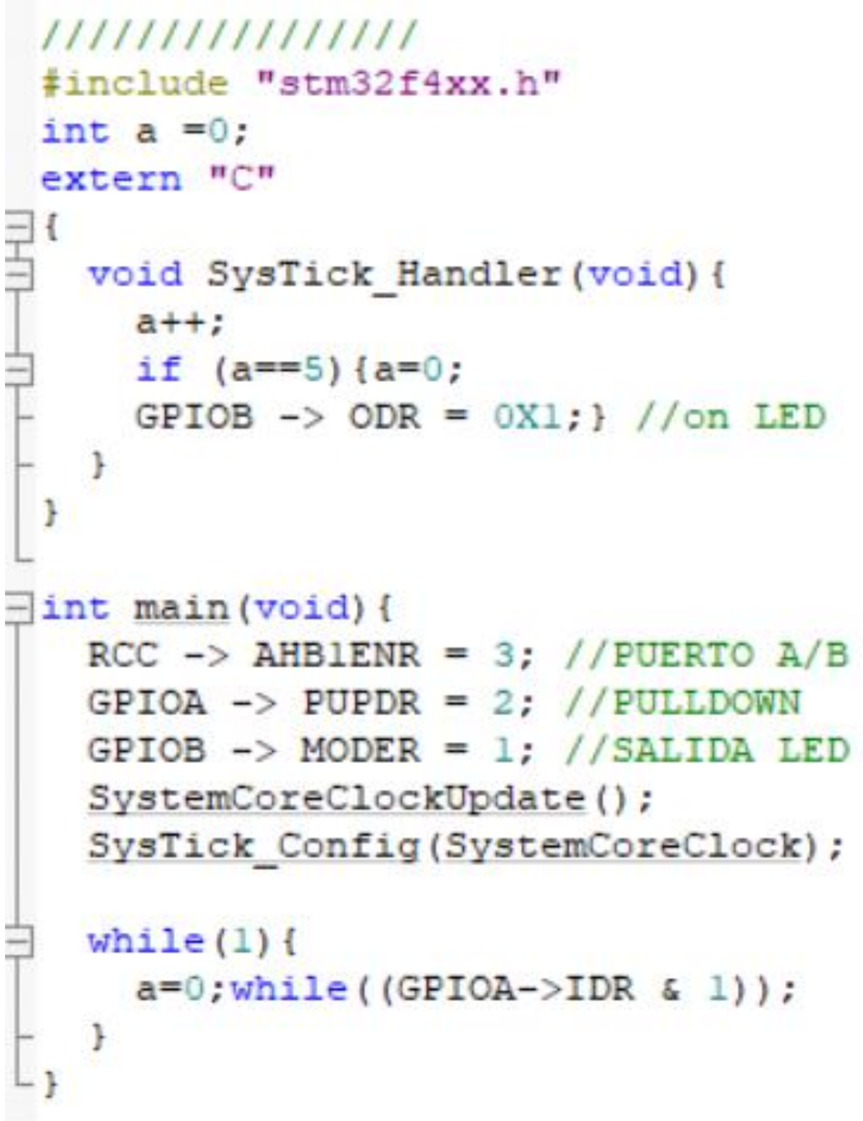
```
/////////////////
#include "stm32f4xx.h"
int a =0;
extern "C"
{
  void SysTick_Handler(void){
    a++;
    if (a==5){a=0;
    GPIOB -> ODR = 0X1;} //on LED
  }
}

int main(void){
  RCC -> AHB1ENR = 3; //PUERTO A/B
  GPIOA -> PUPDR = 2; //PULLDOWN
  GPIOB -> MODER = 1; //SALIDA LED
  SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock);

  while(1){
    a=0;while((GPIOA->IDR & 1));
  }
}
```



Ejercicio en clase:

Emplear un pulsador para encender un led, si el pulso se mantiene mínimo 5 segundos, entrada Pto A2 salida Pto B4.

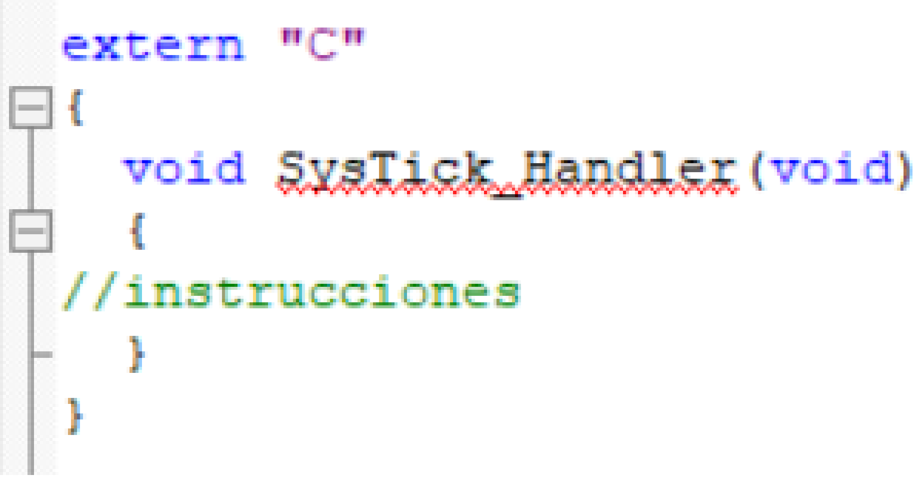




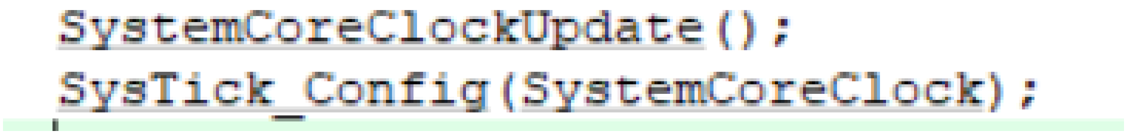
```
////////////////
#include "stm32f4xx.h"
int a =0;
extern "C"
{
  void SysTick_Handler(void){
    a++;
    if (a==5){a=0;
    GPIOB -> ODR = 0X1;} //on LED
  }
}

int main(void){
  RCC -> AHB1ENR = 3; //PUERTO A/B
  GPIOA -> PUPDR = 2; //PULLDOWN
  GPIOB -> MODER = 1; //SALIDA LED
  SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock);

  while(1){
    a=0;while((GPIOA->IDR & 1));
  }
}
```

Ejercicio en clase:

Implementar un contador, basado en interrupción del SysTick, para realizar un programa que lleve la cuenta automática tipo reloj, de segundos en unidades, decenas y centenas para un sistema microcontrolado con visualización por display.



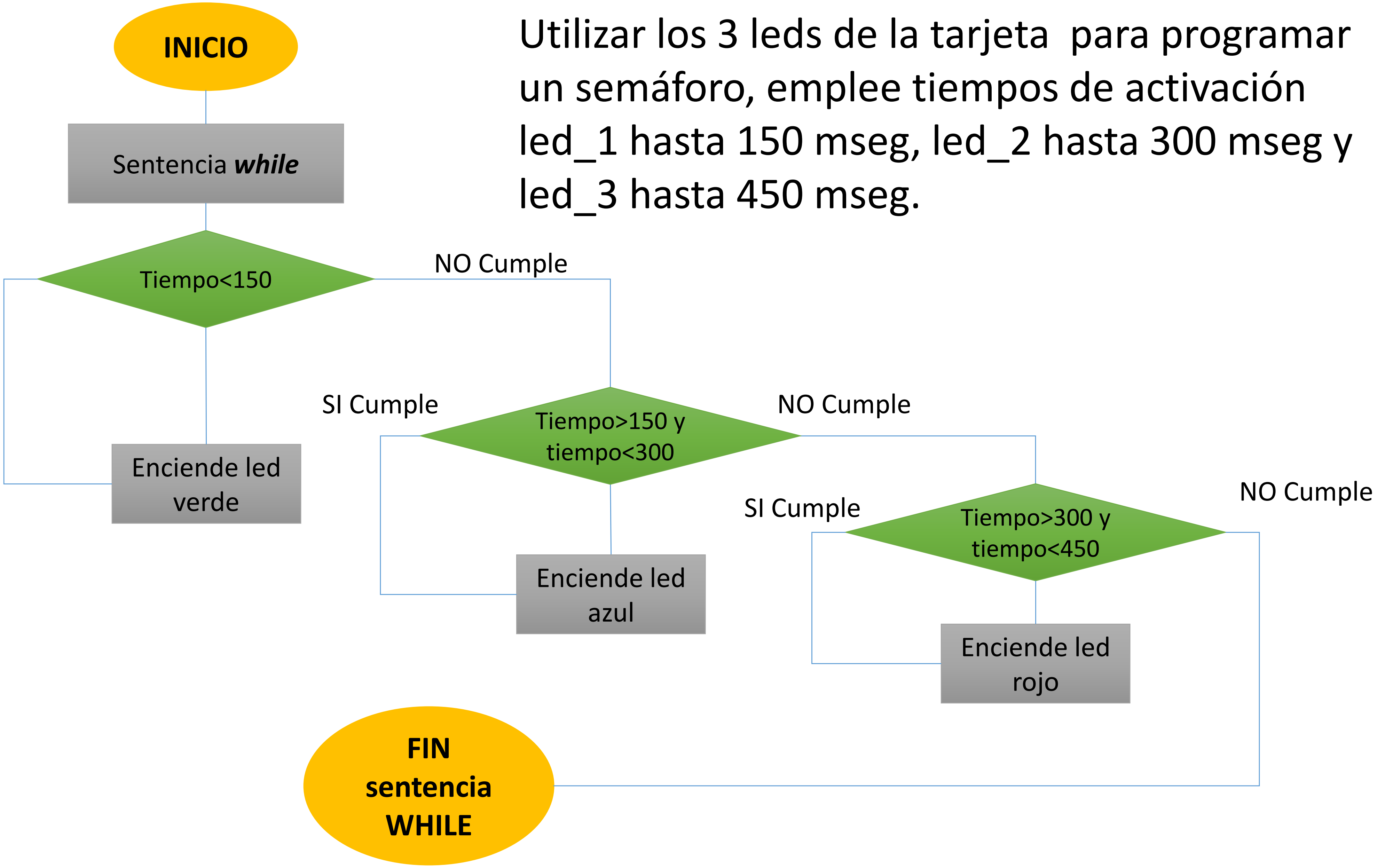
```
extern "C"
{
  void SysTick_Handler(void)
  {
//instrucciones
  }
}
```



```
SystemCoreClockUpdate();
SysTick_Config(SystemCoreClock);
```

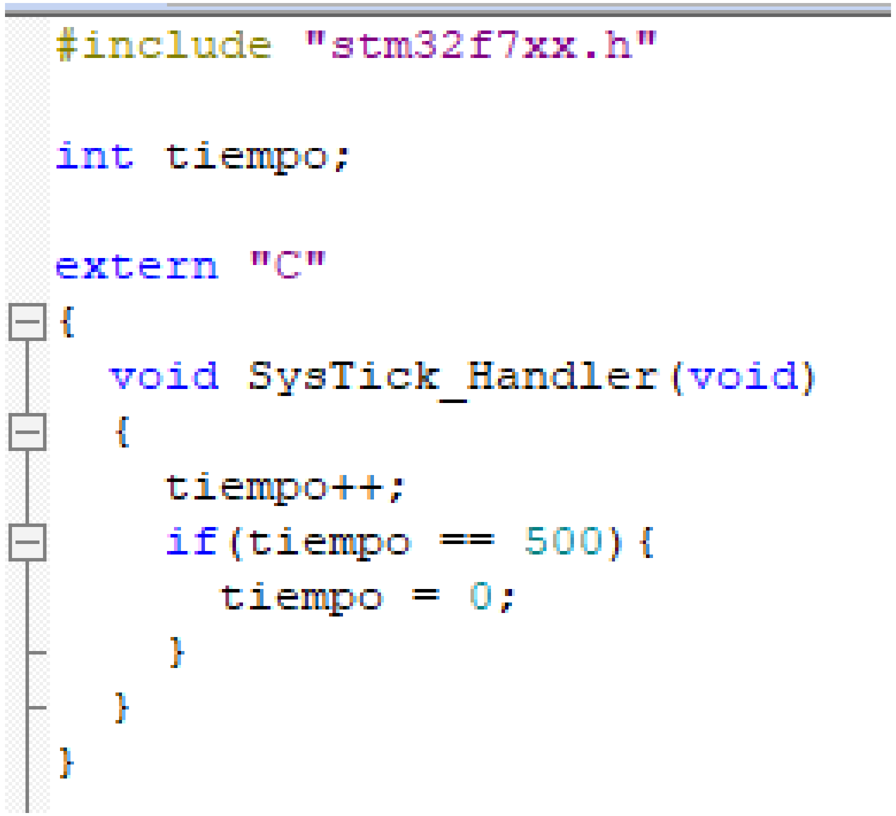
EJEMPLO:

Utilizar los 3 leds de la tarjeta para programar un semáforo, emplee tiempos de activación led_1 hasta 150 mseg, led_2 hasta 300 mseg y led_3 hasta 450 mseg.

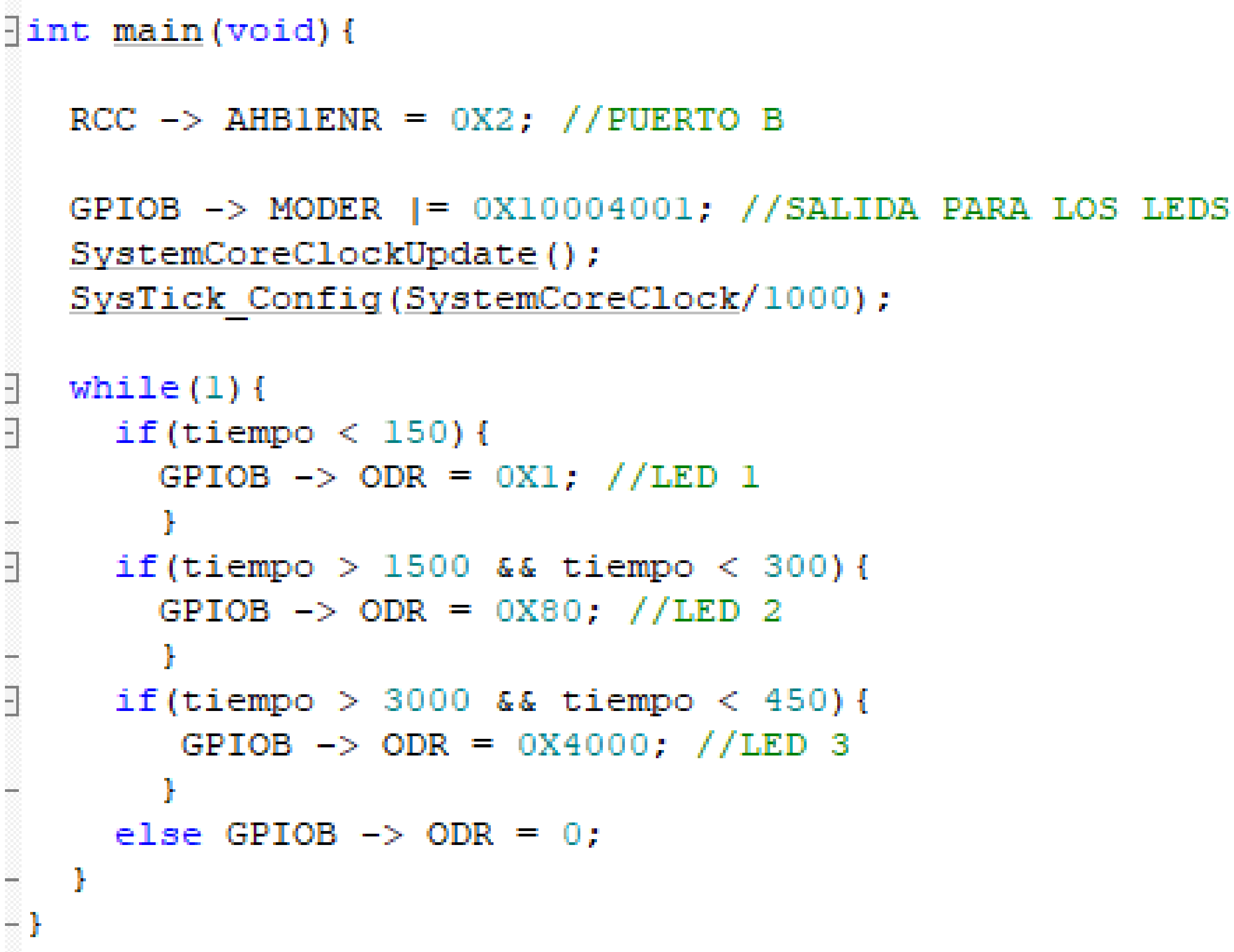


EJEMPLO:

Utilizar los 3 leds de la tarjeta para programar un semáforo, emplee tiempos de activación led_1 hasta 150 mseg, led_2 hasta 300 mseg y led_3 hasta 450 mseg.



```
#include "stm32f7xx.h"

int tiempo;

extern "C"
{
  void SysTick_Handler(void)
  {
    tiempo++;
    if(tiempo == 500){
      tiempo = 0;
    }
  }
}
```



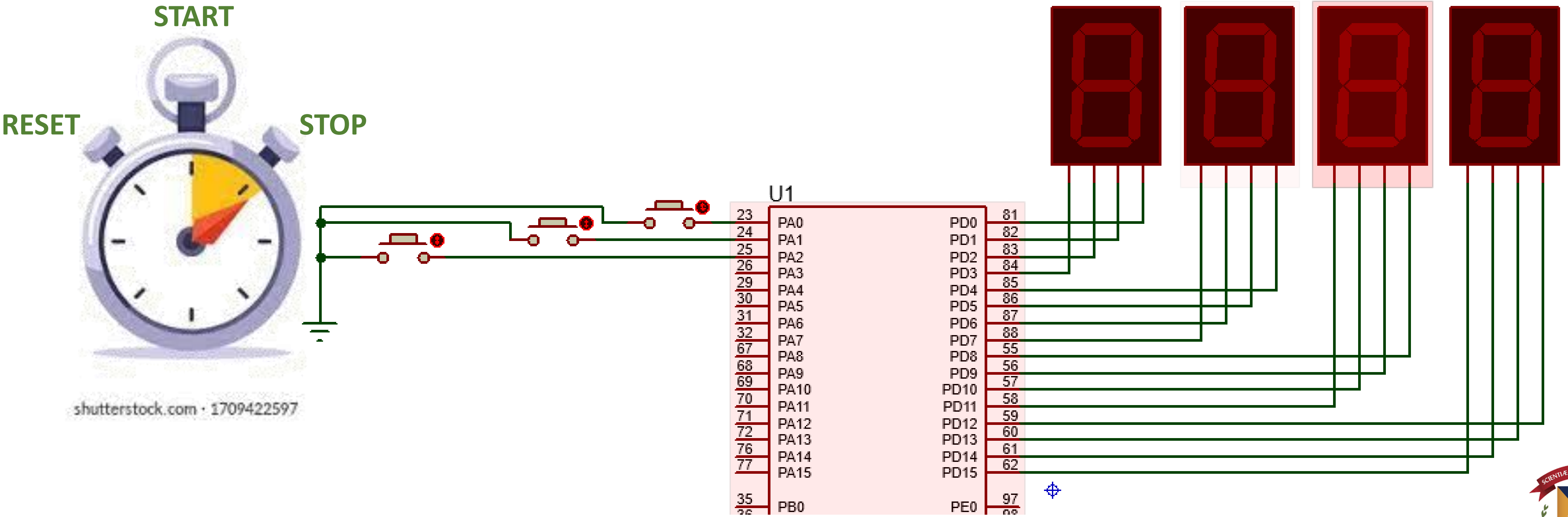
```
int main(void){

  RCC -> AHB1ENR = 0X2; //PUERTO B

  GPIOB -> MODER |= 0X10004001; //SALIDA PARA LOS LEDS
  SystemCoreClockUpdate();
  SysTick_Config(SystemCoreClock/1000);

  while(1){
    if(tiempo < 150){
      GPIOB -> ODR = 0X1; //LED 1
      }
    if(tiempo > 1500 && tiempo < 300){
      GPIOB -> ODR = 0X80; //LED 2
      }
    if(tiempo > 3000 && tiempo < 450){
       GPIOB -> ODR = 0X4000; //LED 3
      }
    else GPIOB -> ODR = 0;
  }
}
```

Ejercicio en clase:

Emplear el pulsador de la tarjeta para incrementar un 10% el tiempo de encendido/apagado de un led (tarjeta), cada vez que es pulsado.

TAREA

Diseñar un cronometro basado en systick manejado por 3 pulsadores, con salida por cuatro displays con decodificador ubicados en el mismo puerto, estos visualizan minutos y segundos (decenas y unidades cada uno). Debe existir un pulsador de inicio, otro de reset y otro de pausa, según el siguiente montaje.



shutterstock.com · 1709422597