

Departamento de Arquitectura y Tecnología de Computadores



UNIVERSIDAD DE SEVILLA

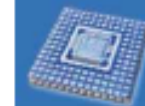


E.T.S. de Ingeniería Informática  
Avda. Reina Mercedes, S/N.  
41012 Sevilla, SPAIN



Escuela Universitaria Politécnica  
C/ Virgen de África, 7.  
41011 Sevilla, SPAIN

# Introducción al STM32: Manejo de GPIO y uso del SysTick



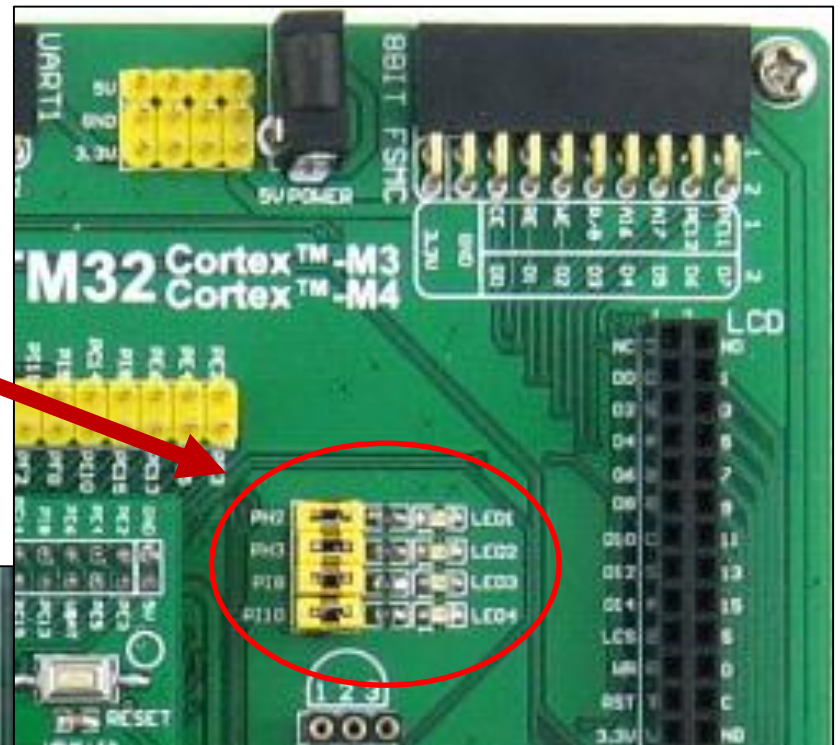
# Objetivos

- Introducir al desarrollo con ColDE de CooCox.
- Familiarizarse con las librerías de ST para el manejo de periféricos.
- Uso del GPIO del STM32.
  - Implementar un controlador para el manejo de los LEDs de la Open407i de alto nivel.
  - Implementar un controlador para leer el joystick digital de la Open407i.
- Introducir retrasos temporales activos usando el SysTick para componer animaciones.

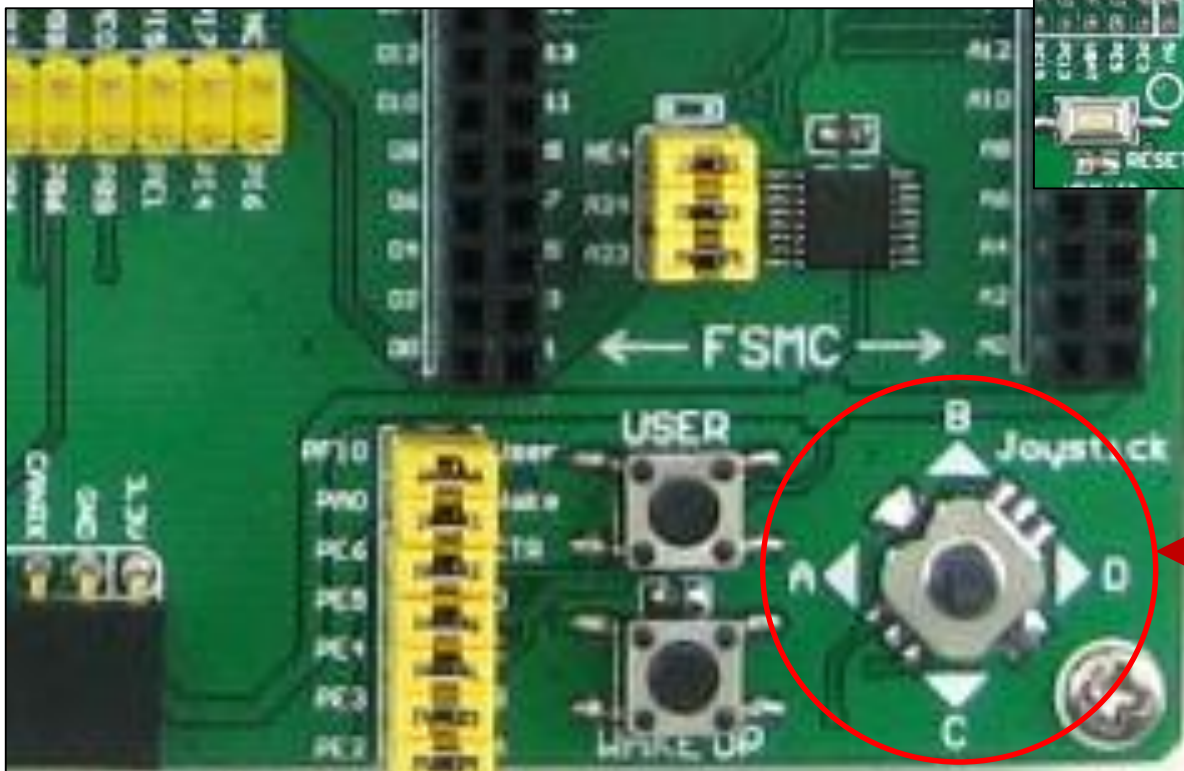
# Desarrollo de práctica

1. Elementos de la Open407i.
2. Creación de un proyecto con Co-IDE.
3. Fuentes del proyecto por defecto.
4. GPIO en el STM32.
5. Manejo de los GPIOs usando las librerías del STM32.
6. Implementación de un driver para manejo los leds de alto nivel.
7. Uso del SysTick para introducir esperas activas.
8. Implementación de un driver para leer el joystick.
9. Diseño de animaciones basadas en retraso temporal activo.

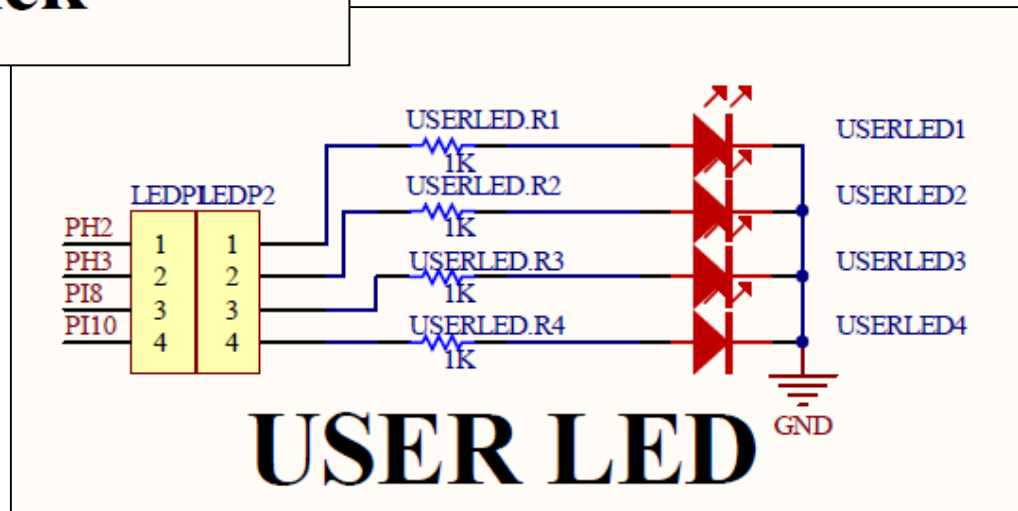
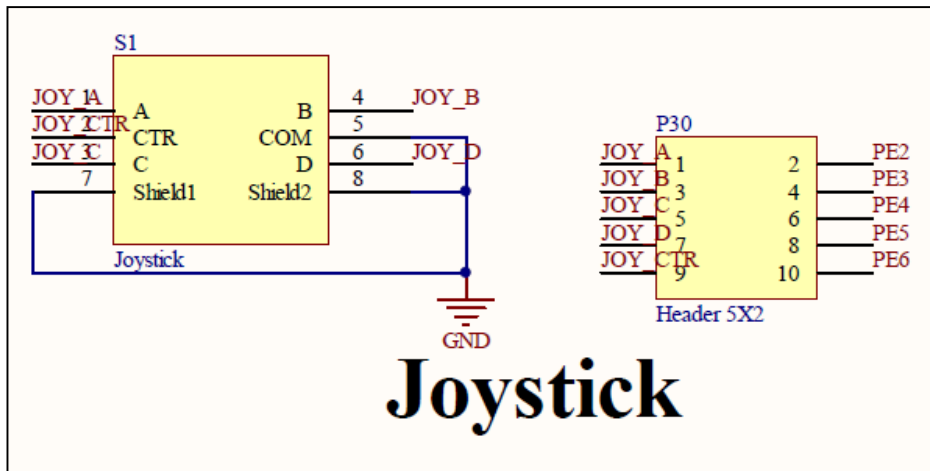
**LEDS**



**JOYSTICK**

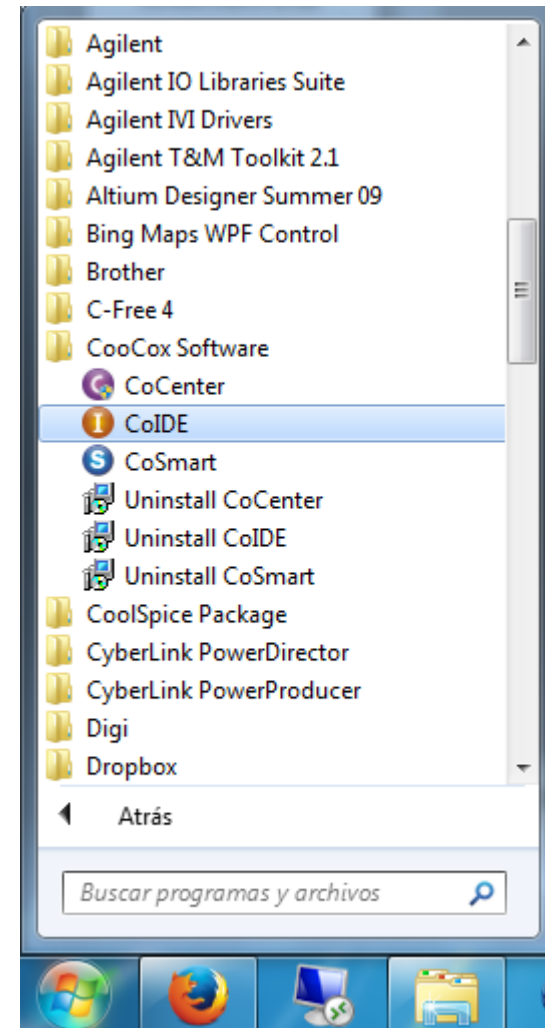


# Esquemáticos

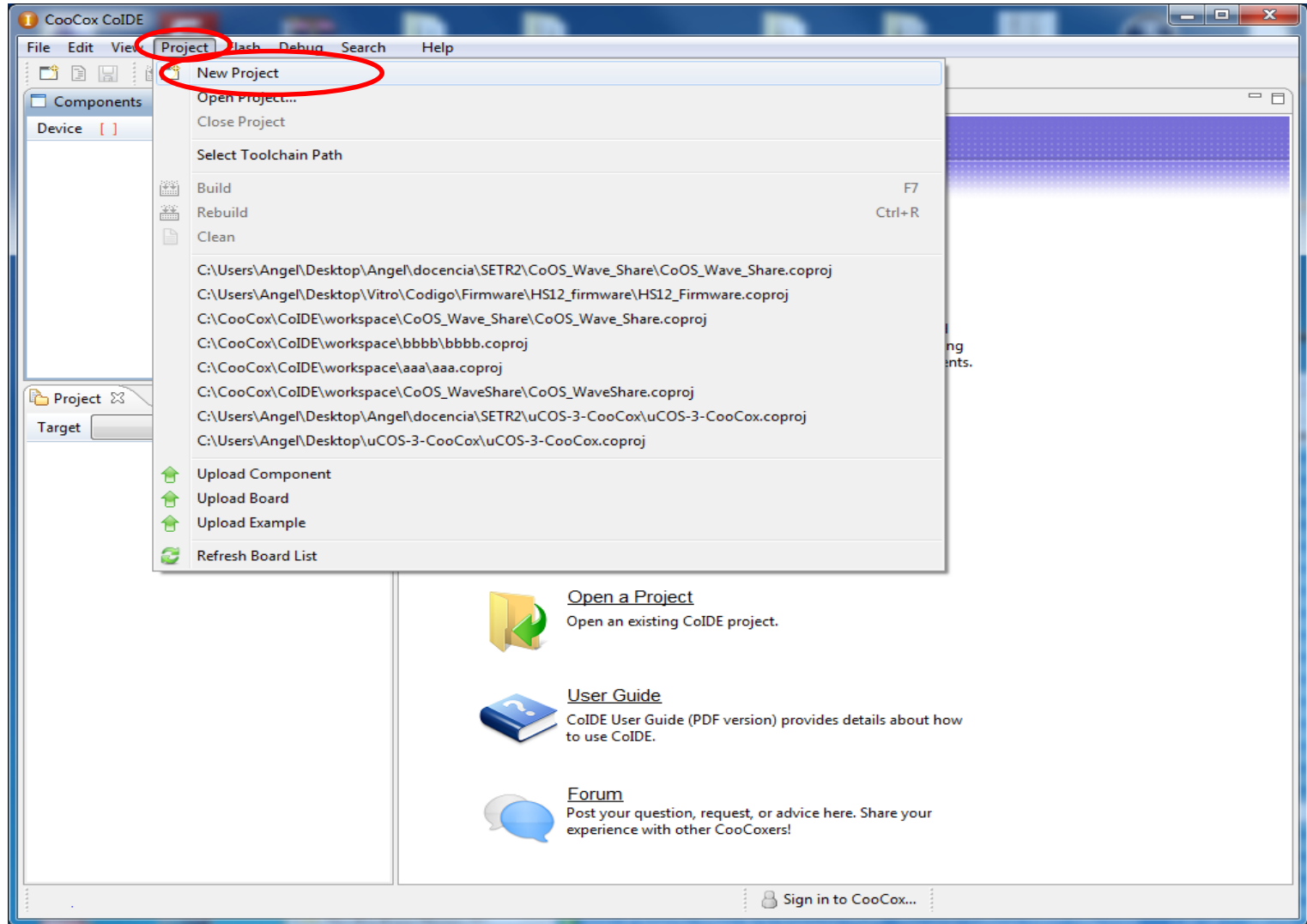


# CooCox - CoIDE

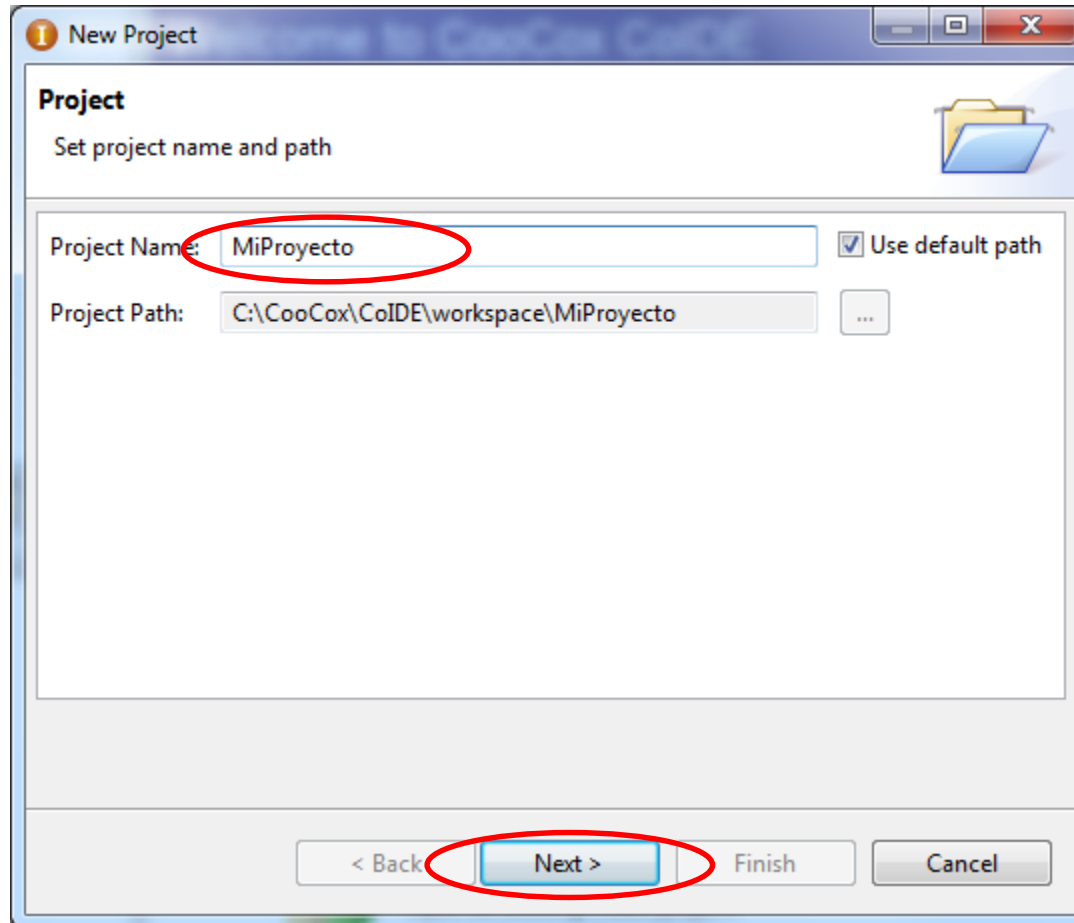
- Lanzando el CoIDE



# Creando un proyecto nuevo

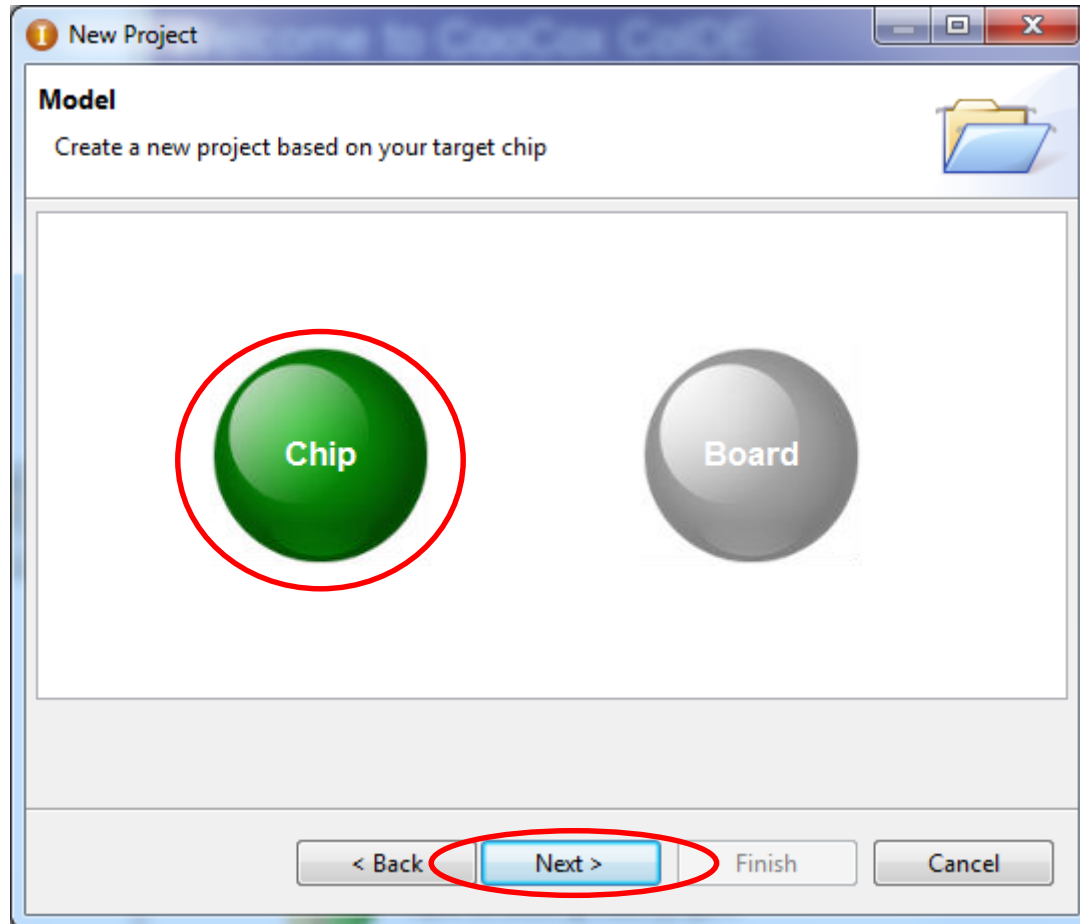


# Nombrar y definir la ruta

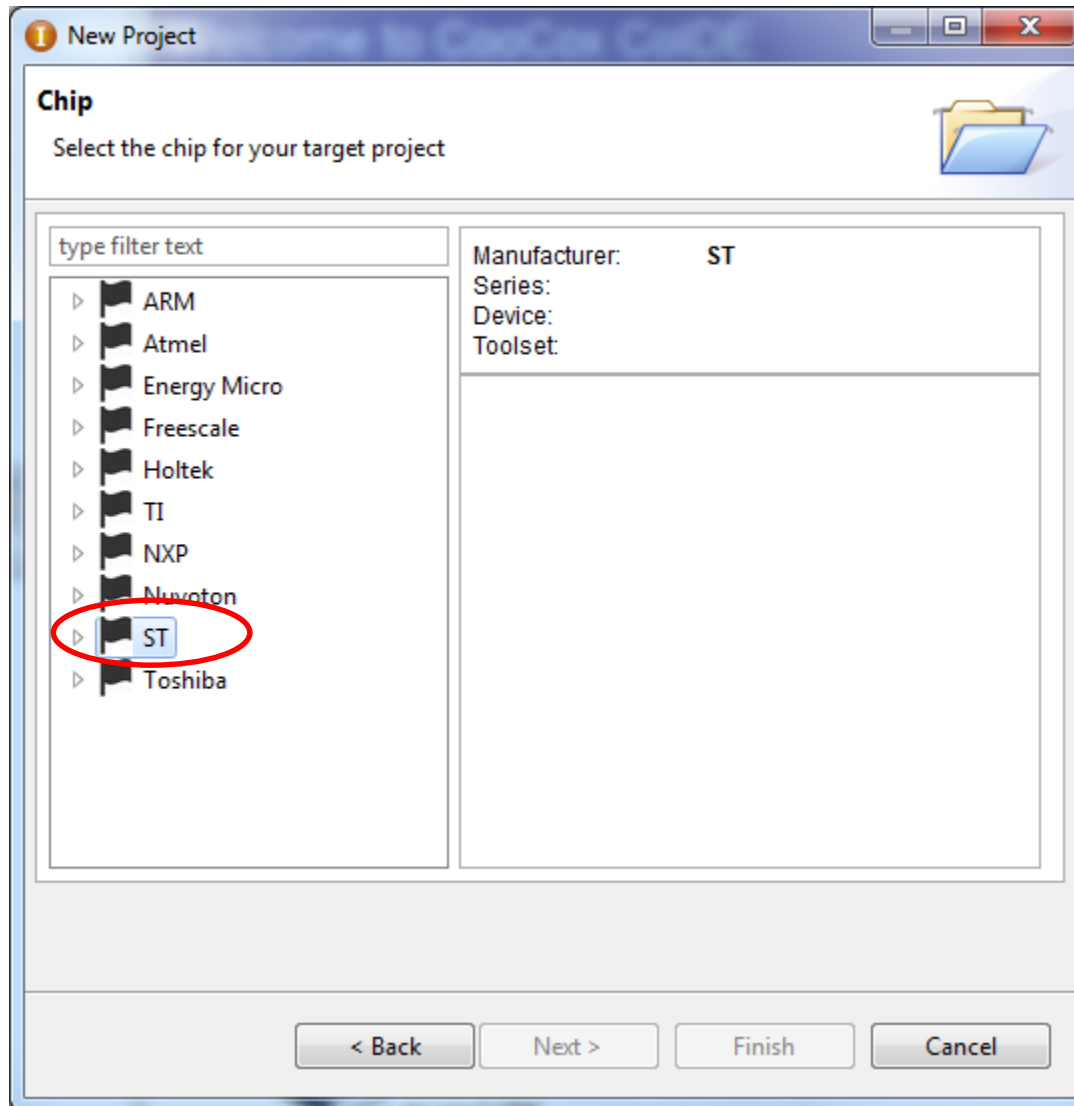




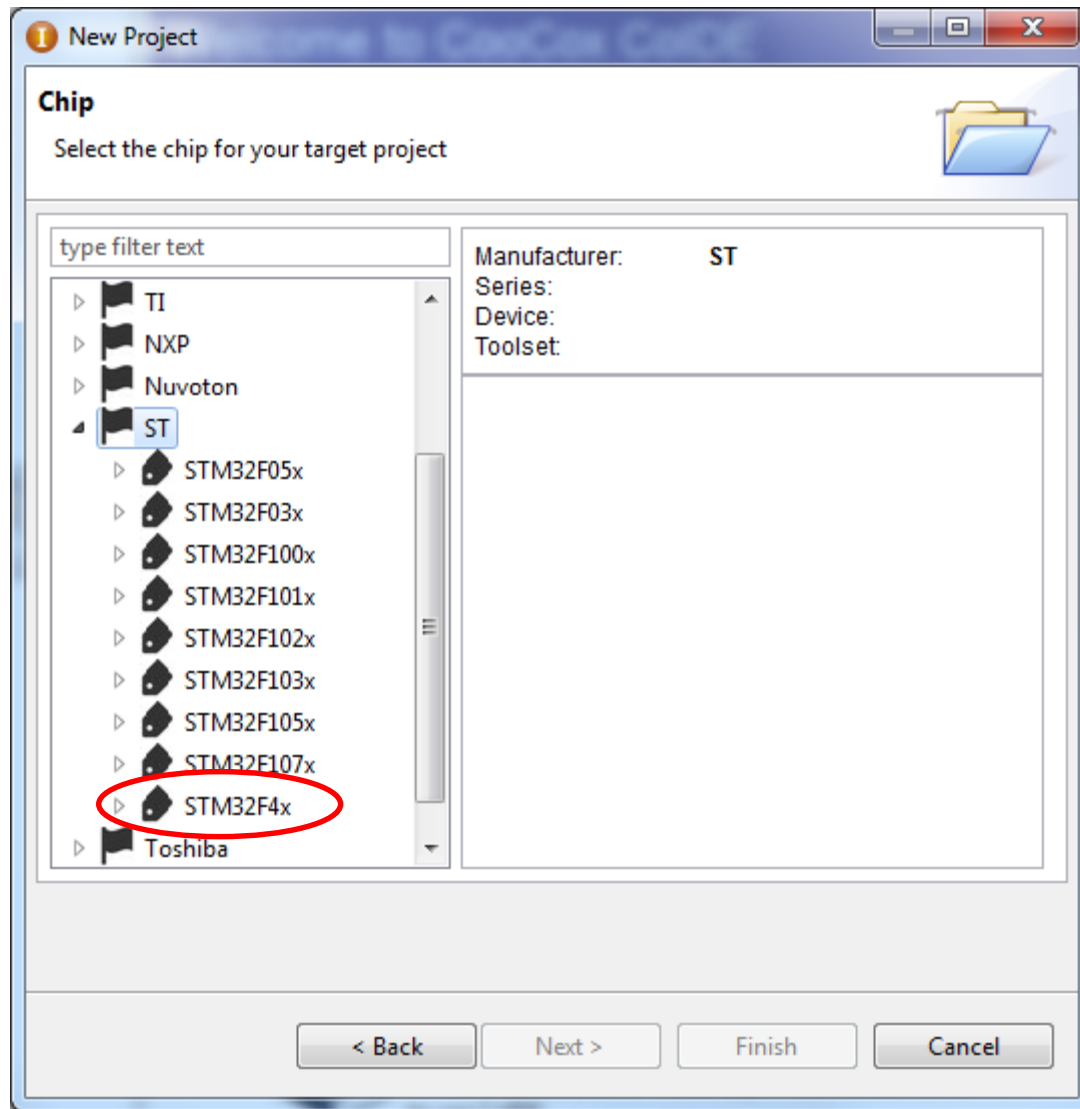
# Selección de dispositivo



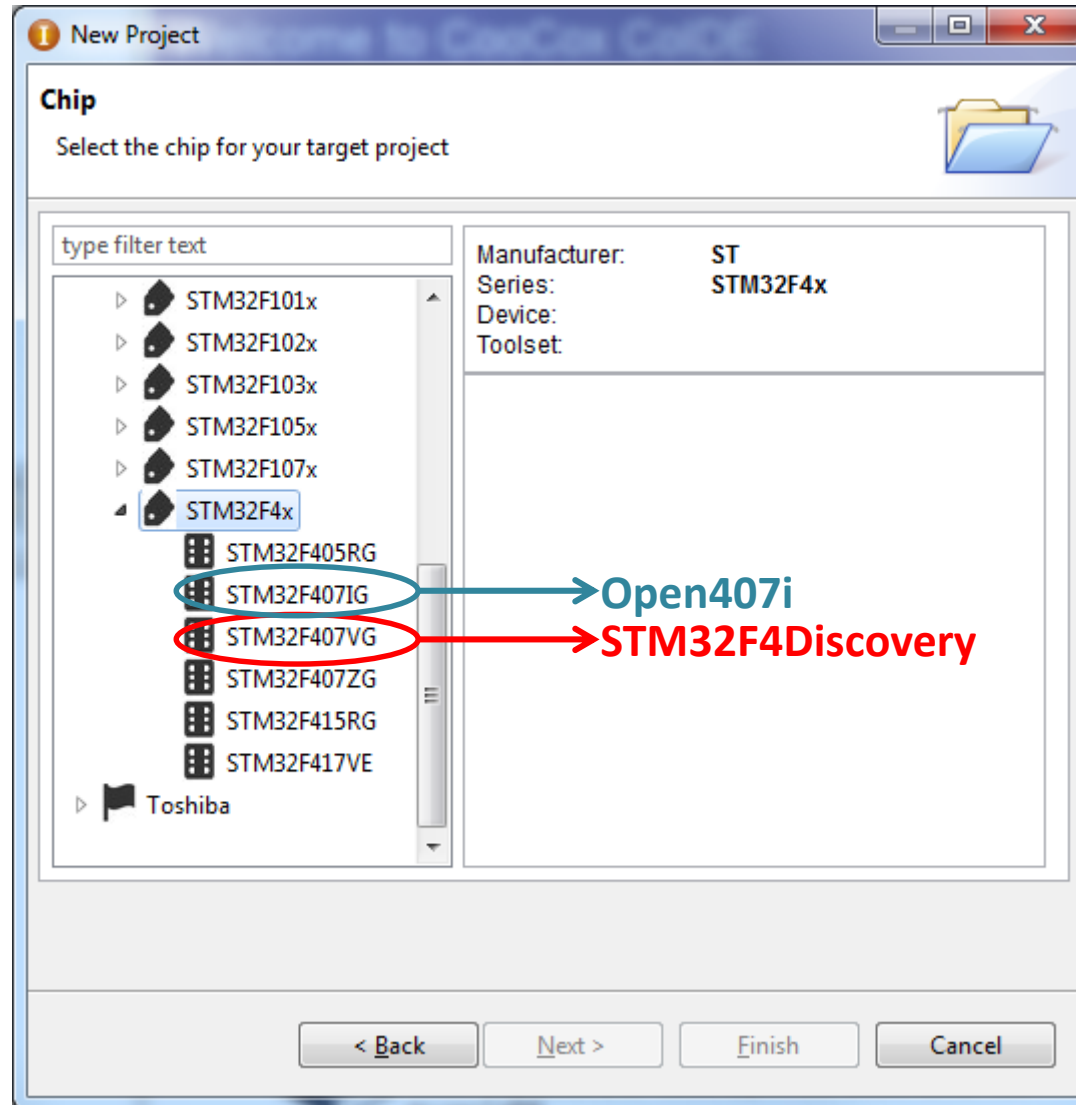
# Selección del procesador



# Selección del procesador



# Selección del procesador



FileEditViewProjectFlashDebugSearchHelp

Components

Device [ STM32F407VG ]

Common

- C Library (with 3 examples)
- M4 CMSIS Core (with 1 example)

Boot

- CMSIS BOOT (with 1 example)

Peripheral.ST

- RCC (with 2 examples)
- GPIO (with 3 examples)
- FLASH (with 1 example)
- MISC (with 1 example)

Project

Target MiProjecto

MiProjecto

cmsis

- core\_cm4\_simd.h
- core\_cm4.h
- core\_cmFunc.h
- core\_cmInstr.h

cmsis\_boot

- startup
  - startup\_stm32f4xx.c
  - stm32f4xx\_conf.h
  - stm32f4xx.h
  - system\_stm32f4xx.c
  - system\_stm32f4xx.h

cmsis\_lib

- include
  - misc.h
  - stm32f4xx\_flash.h
  - stm32f4xx\_gpio.h
  - stm32f4xx\_rcc.h
- source
  - misc.c
  - stm32f4xx\_flash.c
  - stm32f4xx\_gpio.c
  - stm32f4xx\_rcc.c

syscalls

- syscalls.c
- main.c

Step 3 Select Basic Components [ ST / STM32F407VG ]

COMMON

☒

C Library

Implement the minimal functionality required to allow newlib to link

Available

CooCo

☐

Retarget printf

Implementation of printf(), sprintf() to reduce memory footprint

Available

CooCo

☐

Semihosting

Implementation of Semihosting GetChar/SendChar

Available

CooCo

☒

M4 CMSIS Core

CMSIS core for Cortex M4

Available

CooCo

☐

STM32F4 Standard Framework

This Framework does the minimal configurations of the Controller. Supported Devices are STM32F40X and STM32F41X

Available

Moritz

BOOT

☒

CMSIS BOOT

CMSIS BOOT for STM32F4x series

Available

CooCo

PERIPHERAL.ST

☒

RCC

Reset and clock control for STM32F4xx

Available

CooCo

☐

PWR

System configuration controller for STM32F4xx

Available

CooCo

☐

CRC

Cyclic Redundancy Check for STM32F4xx

Available

CooCo

☒

GPIO

General Purpose Input/Output for STM32F4xx

Available

CooCo

☐

EXTI

External Interrupt/Line Controller for STM32F4xx

Available

CooCo

☐

RTC

Real Time Clock for STM32F4xx

Available

CooCo

☐

IWDG

Independent watchdog for STM32F4xx

Available

CooCo

☐

WWDG

Window watchdog for STM32F4xx

Available

CooCo

☐

SPI

Serial peripheral interface for STM32F4xx

Available

CooCo

☐

I2C

Inter-integrated circuit interface for STM32F4xx

Available

CooCo

☐

DMA

Direct Memory Access for STM32F4xx

Available

CooCo

☒

FLASH

Flash Memory Controller for STM32F4xx

Available

CooCo

☐

TIM

Advanced-control timers for STM32F4xx

Available

CooCo

☐

ADC

Analog/Digital Convert for STM32F4xx

Available

CooCo

☐

DAC

General Purpose Input/Output for STM32F4xx

Available

CooCo

☐

FSMC

Flexible static memory controller for STM32F4xx

Available

CooCo

☐

USART

Universal synchronous asynchronous receiver transmitter for STM32F4xx

Available

CooCo

☐

RNG

Random number generator for STM32F4xx

Available

CooCo

☐

HASH

General Purpose Input/Output for STM32F4xx

Available

CooCo

☐

SDIO

Secure digital input/output interface for STM32F4xx

Available

CooCo

☐

CAN

Controller Area Network for STM32F4xx

Available

CooCo

☐

DCMI

Digital Camera Interface for STM32F4xx

Available

CooCo

☐

CRYP

Cryptographic Processor for STM32F4xx

Available

CooCo

☒

MISC

Nested vectored interrupt controller for STM32F4xx

Available

CooCo

☐

DBGMCU

Debug MCU Configuration for STM32F4xx

Available

CooCo

☐

SYSCFG

System configuration controller for STM32F4xx

Available

CooCo

☐

TEST

describing stuff

Available

lundbu

☐

Name

Description

Available

nbd (A

☐

Name

Description

Available

nbd (A

RTOS

☐

STM4CoOS

CoOS frame application for STM32F4 Discover Board

Available

janheit

☐

STM4CoOS

CoOS frame application for STM32F4 Discover Board

Available

janheit

☐

freeRTOS

fgshgm

Available

aaaaa

☐

ff

ff

Available

aaaaa

☐

f

f

Available

aaaaa

☐

fsd

sdf

Available

aaaaa

☐

sdfas

dsafgshdfgh

Available

aaaaa

☐

sdf

'p'[poiuytgrfss

Available

aaaaa

☐

FreeRTOS

niki fr4dhoue

Available

ola (A

Manufacturers Chips Peripherals Drivers Others

Help

MISC

Overview

- The nested vector interrupt controller NVIC includes the following features: 87 maskable interrupt channels (not including the 16 interrupt lines of Cortex7-M4F); 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.

API Reference

NVIC\_PriorityGroupConfig

Configures the priority grouping: pre-emption priority and subpriority.

NVIC\_Init

Initializes the NVIC peripheral according to the specified parameters in the NVIC\_InitStruct.

NVIC\_SetVectorTable

Sets the vector table location and Offset.

NVIC\_SystemLPConfig

Selects the condition for the system to enter low power mode.

SysTick\_CLKSourceConfig

Configures the SysTick clock source.

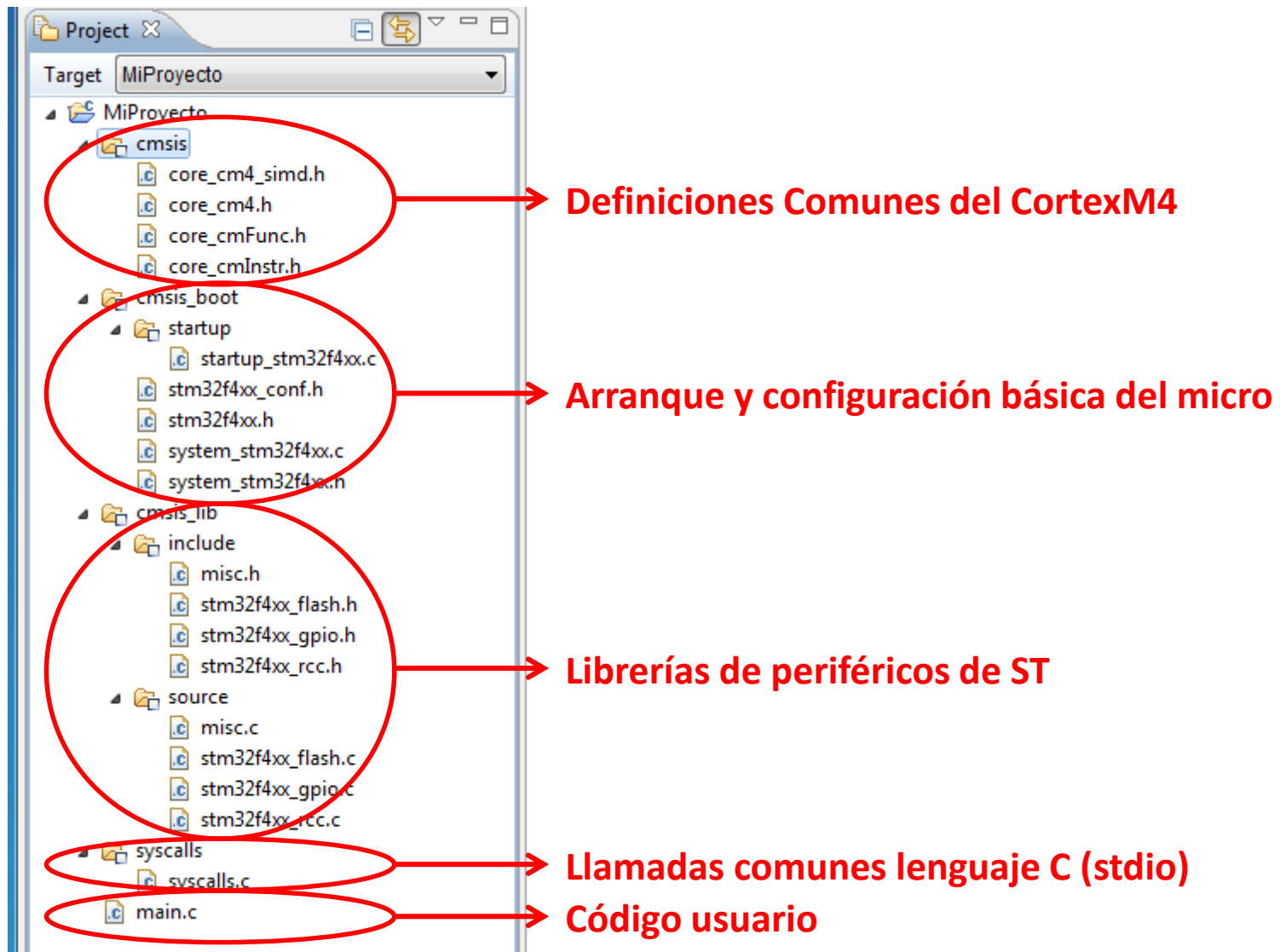
Source files

- misc.h
- misc.c

Dependency

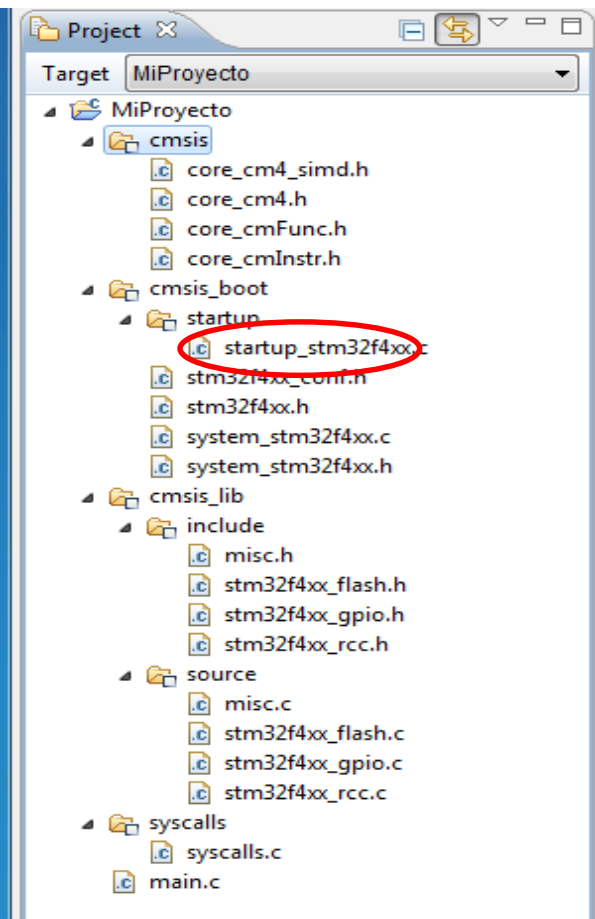
CMSIS BOOT

13



# Inicialización del STM32 (startup\_)

- Define rutinas de servicio de interrupción por defecto e inicializa el vector de interrupciones



```
148 /**
149  * @brief The minimal vector table for a Cortex M3. Note that the proper construct
150  *        must be placed on this to ensure that it ends up at physical address
151  *        0x00000000.
152  */
153 __attribute__((used, section(".isr_vector")))
154 void (* const g_pfnVectors[])(void) =
155 {
156     /*-----Core Exceptions-----*/
157     (void *)&pulStack[STACK_SIZE], /*!< The initial stack pointer */
158     Reset_Handler, /*!< Reset Handler */
159     HardFault_Handler, /*!< Hard Fault Handler */
160     MemManage_Handler, /*!< MPU Fault Handler */
161     SysTick_Handler, /*!< SysTick Handler */
162
163     /*-----External Exceptions-----*/
164     RCC_IRQHandler, /*!< 5: RCC */
165     EXTI0_IRQHandler, /*!< 6: EXTI Line0 */
166     DMA1_Stream0_IRQHandler, /*!< 11: DMA1 Stream 0 */
167     ADC_IRQHandler, /*!< 18: ADC1, ADC2 and ADC3s */
168     TIM2_IRQHandler, /*!< 28: TIM2 */
169     TIM3_IRQHandler, /*!< 29: TIM3 */
170     TIM4_IRQHandler, /*!< 30: TIM4 */
171     I2C1_EV_IRQHandler, /*!< 31: I2C1 Event */
172     I2C2_EV_IRQHandler, /*!< 33: I2C2 Event */
173     SPI1_IRQHandler, /*!< 35: SPI1 */
174     SPI2_IRQHandler, /*!< 36: SPI2 */
175     USART1_IRQHandler, /*!< 37: USART1 */
176     USART2_IRQHandler, /*!< 38: USART2 */
177     USART3_IRQHandler, /*!< 39: USART3 */
    ...
}
```

# Inicialización del STM32 (startup\_)

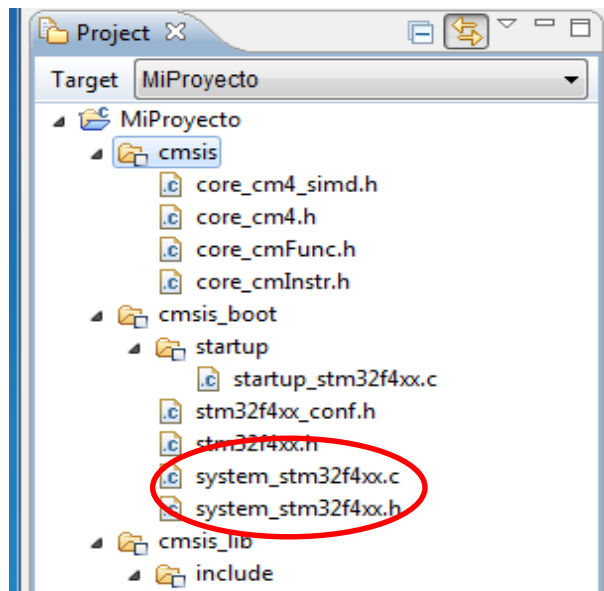
- La ejecución comienza con la interrupción de reset.
- Se inicializan los segmentos de la pila.
- Se llama al main.

```
224 /**
225  * @brief This is the code that gets called when the processor first
226  *        starts execution following a reset event. Only the absolutely
227  *        necessary set is performed, after which the application
228  *        supplied main() routine is called.
229  * @param None
230  * @retval None
231  */
232 void Default_Reset_Handler(void)
233 {
234     /* Initialize data and bss */
235     unsigned long *pulSrc, *pulDest;
236
237     /* Copy the data segment initializers from flash to SRAM */
238     pulSrc = &_sdata;
239
240     for(pulDest = &_sdata; pulDest < &_edata; )
241     {
242         *(pulDest++) = *(pulSrc++);
243     }
244
245     /* Zero fill the bss segment. This is done with inline assembly since this
246      * will clear the value of pulDest if it is not kept in a register. */
247     __asm("    ldr    r0, =_sbss\n"
248          "    ldr    r1, =_ebss\n"
249          "    mov    r2, #0\n"
250          "    .thumb_func\n"
251          "zero_loop:\n"
252          "    cmp    r0, r1\n"
253          "    it     lt\n"
254          "    strlt  r2, [r0], #4\n"
255          "    blt    zero_loop");
256 #ifdef __FPU_USED
257     /* Enable FPU.*/
258     __asm("    LDR.W R0, =0xE000ED88\n"
259          "    LDR R1, [R0]\n"
260          "    ORR R1, R1, #(0xF << 20)\n"
261          "    STR R1, [R0]");
262 #endif
263
264     /* Call the application's entry point.*/
265     main();
266 }
267
268
```



# Inicialización del Reloj (system\_)

- Inicializa el PLL a partir de un cuarzo externo (HSE) o del oscilador interno (HSI).
- OJO! Por defecto usa el HSE a 25MHz.
- En la Open407i y la Discovery el cuarzo es de 8Mhz.



```
42 * 5. This file configures the system clock as follows:
43 * =====
44 *
45 *      Supported STM32F4xx device revision    | Rev A
46 *
47 *      System Clock source                    | PLL (HSE)
48 *
49 *      SYSCLK(Hz)                             | 168000000
50 *
51 *      HCLK(Hz)                               | 168000000
52 *
53 *      AHB Prescaler                          | 1
54 *
55 *      APB1 Prescaler                         | 4
56 *
57 *      APB2 Prescaler                         | 2
58 *
59 *      HSE Frequency(Hz)                      | 25000000
60 *
61 *      PLL_M                                  | 25
62 *
63 *      PLL_N                                  | 336
64 *
65 *      PLL_P                                  | 2
66 *
67 *      PLL_Q                                  | 7
68 *
69 *      PLLI2S_N                               | NA
70 *
71 *      PLLI2S_R                               | NA
72 *
73 *      I2S input clock                        | NA
74 *
75 *      VDD(V)                                 | 3.3
76 *
77 *      Main regulator output voltage          | Scale1 mode
78 *
79 *      Flash Latency(WS)                     | 5
80 *
81 *      Prefetch Buffer                        | OFF
82 *
83 *      Instruction cache                     | ON
84 *
```

# Inicialización del Reloj (system\_)

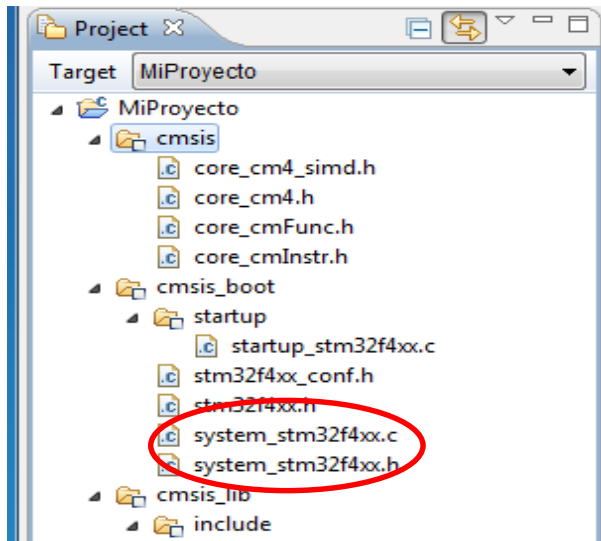
- Generar un fichero de configuración de reloj que use el HSI como fuente de reloj, y fije una frecuencia de 168MHz

```

// This file configures the system clock as follows.
=====
*
* Supported STM32F4xx device revision | Rev A
*
* System Clock source                 | PLL(HSI)
*
* SYSCLK(Hz)                         | 168000000
*
* HCLK(Hz)                           | 168000000
*
* AHB Prescaler                       | 1
*
* APB1 Prescaler                      | 4
*
* APB2 Prescaler                      | 2
*
* HSE Frequency(Hz)                  | 25000000
*
* PLL_M                               | 16
*
* PLL_N                               | 336
*
* PLL_P                               | 2
*
* PLL_Q                               | 7
*
* PLLI2S_N                           | NA
*
* PLLI2S_R                           | NA
*
* I2S input clock                     | NA
*
* VDD(V)                             | 3.3
=====
```

# Inicialización del Reloj (system\_)

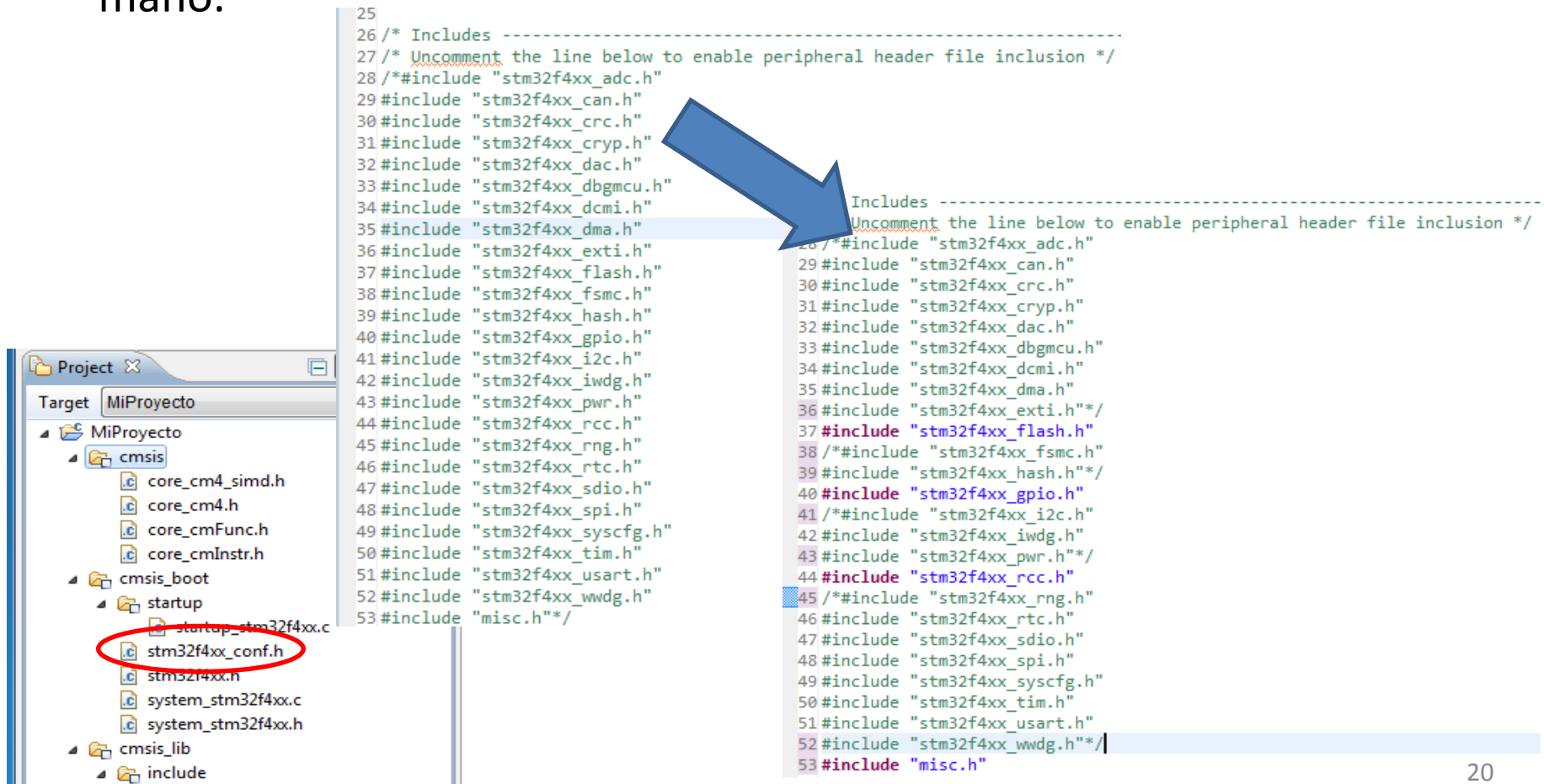
- La función de inicialización del sistema es: SystemInit (void).
- Inhabilita todos los relojes, llama a SetSysClock(), y finalmente posiciona el origen del vector de interrupciones.



```
208 void SystemInit(void)
209 {
210     /* FPU settings -----*/
211     #if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
212         SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2)); /* set CP10 and CP11 Full Access */
213     #endif
214
215     /* Reset the RCC clock configuration to the default reset state -----*/
216     /* Set HSION bit */
217     RCC->CR |= (uint32_t)0x00000001;
218
219     /* Reset CFGR register */
220     RCC->CFGR = 0x00000000;
221
222     /* Reset HSEON, CSSON and PLLON bits */
223     RCC->CR &= (uint32_t)0xFEFFFFFF;
224
225     /* Reset PLLCFGR register */
226     RCC->PLLCFGR = 0x24003010;
227
228     /* Reset HSEBYP bit */
229     RCC->CR &= (uint32_t)0xFFBFFFFFF;
230
231     /* Disable all interrupts */
232     RCC->CIR = 0x00000000;
233
234
235     /* Configure the System clock source, PLL Multiplier and Divider factors,
236        AHB/APBx prescalers and Flash settings -----*/
237     SetSysClock();
238
239     /* Configure the Vector Table location add offset address -----*/
240     #ifndef VECT_TAB_SRAM
241         SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
242     #else
243         SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */
244     #endif
245 }
```

# Inclusión de librerías de periféricos (\_conf)

- Se incluyen todas las .h de las librerías de ST.
- Forma cómoda de incluir todas las librerías de una vez.
- Por defecto vienen todas comentadas, hay que descomentarlas a mano.



The image shows a screenshot of an IDE with a project named 'MiProyecto'. In the left pane, the file tree shows the following structure:

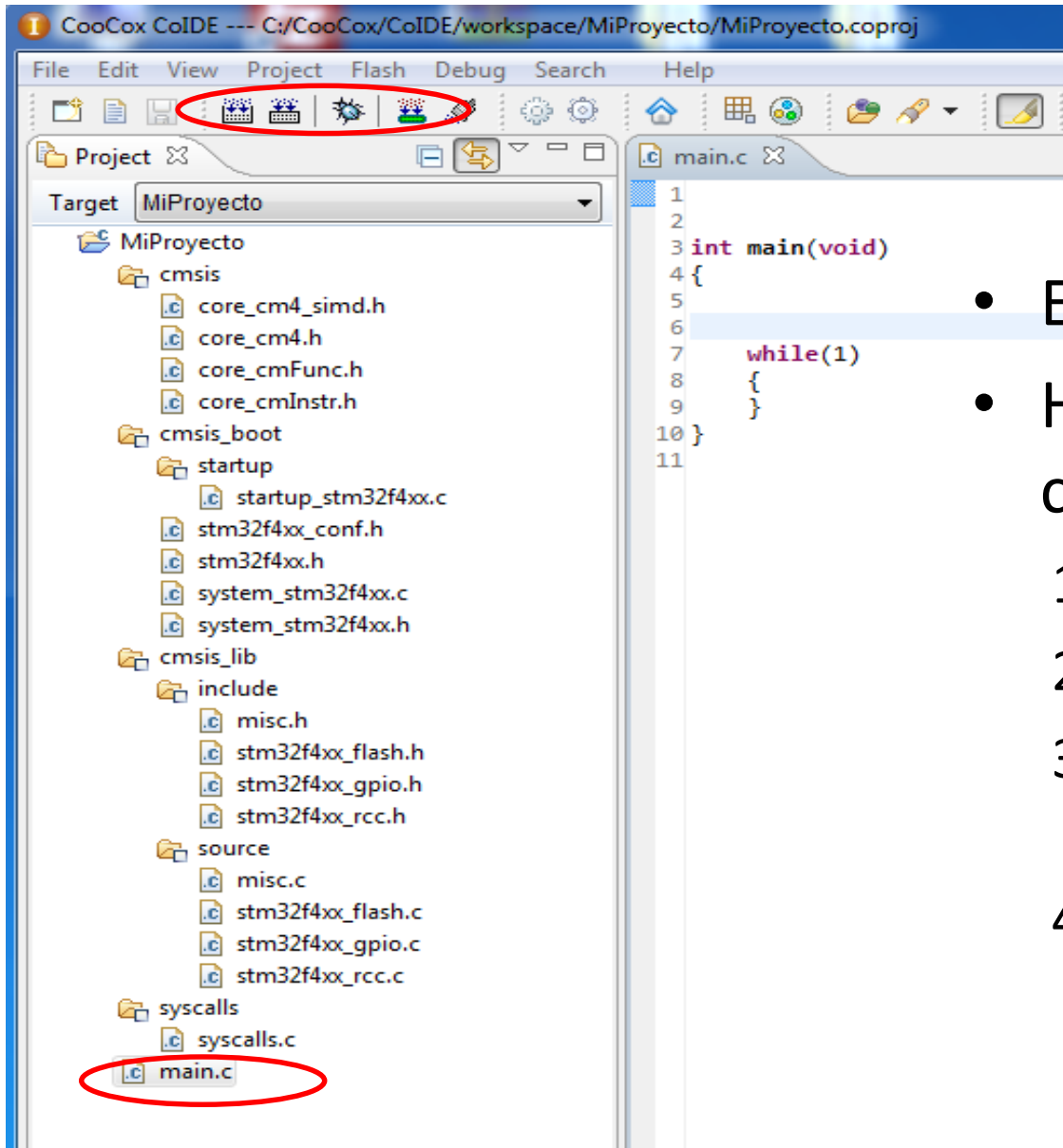
- Project
- Target: MiProyecto
- Source Files:
  - cmsis
    - core\_cm4\_simd.h
    - core\_cm4.h
    - core\_cmFunc.h
    - core\_cmInstr.h
  - cmsis\_boot
    - startup
    - startup\_stm32f4xx.c
    - stm32f4xx\_conf.h** (circled)
    - stm32f4xx.h
    - system\_stm32f4xx.c
    - system\_stm32f4xx.h
  - cmsis\_lib
    - include

The main editor displays the content of 'stm32f4xx\_conf.h'. The file contains two identical blocks of code, each starting with a comment: '/\* Includes -----' and '/\* Uncomment the line below to enable peripheral header file inclusion \*/'. The first block lists various peripheral headers from line 25 to 53. The second block is identical but starts at line 46. A blue arrow points from the 'stm32f4xx\_dma.h' line in the first block to the corresponding line in the second block.

```
25
26 /* Includes -----
27 /* Uncomment the line below to enable peripheral header file inclusion */
28 /*#include "stm32f4xx_adc.h"
29 #include "stm32f4xx_can.h"
30 #include "stm32f4xx_crc.h"
31 #include "stm32f4xx_cryp.h"
32 #include "stm32f4xx_dac.h"
33 #include "stm32f4xx_dbgmcu.h"
34 #include "stm32f4xx_dcmi.h"
35 #include "stm32f4xx_dma.h"
36 #include "stm32f4xx_exti.h"
37 #include "stm32f4xx_flash.h"
38 #include "stm32f4xx_fsmc.h"
39 #include "stm32f4xx_hash.h"
40 #include "stm32f4xx_gpio.h"
41 #include "stm32f4xx_i2c.h"
42 #include "stm32f4xx_iwdg.h"
43 #include "stm32f4xx_pwr.h"
44 #include "stm32f4xx_rcc.h"
45 #include "stm32f4xx_rng.h"
46 #include "stm32f4xx_rtc.h"
47 #include "stm32f4xx_sdio.h"
48 #include "stm32f4xx_spi.h"
49 #include "stm32f4xx_syscfg.h"
50 #include "stm32f4xx_tim.h"
51 #include "stm32f4xx_usart.h"
52 #include "stm32f4xx_wwdg.h"
53 #include "misc.h"*/

46 /* Includes -----
47 /* Uncomment the line below to enable peripheral header file inclusion */
48 /*#include "stm32f4xx_adc.h"
49 #include "stm32f4xx_can.h"
50 #include "stm32f4xx_crc.h"
51 #include "stm32f4xx_cryp.h"
52 #include "stm32f4xx_dac.h"
53 #include "stm32f4xx_dbgmcu.h"
54 #include "stm32f4xx_dcmi.h"
55 #include "stm32f4xx_dma.h"
56 #include "stm32f4xx_exti.h"*/
57 #include "stm32f4xx_flash.h"
58 /*#include "stm32f4xx_fsmc.h"
59 #include "stm32f4xx_hash.h"*/
60 #include "stm32f4xx_gpio.h"
61 /*#include "stm32f4xx_i2c.h"
62 #include "stm32f4xx_iwdg.h"
63 #include "stm32f4xx_pwr.h"*/
64 #include "stm32f4xx_rcc.h"
65 /*#include "stm32f4xx_rng.h"
66 #include "stm32f4xx_rtc.h"
67 #include "stm32f4xx_sdio.h"
68 #include "stm32f4xx_spi.h"
69 #include "stm32f4xx_syscfg.h"
70 #include "stm32f4xx_tim.h"
71 #include "stm32f4xx_usart.h"
72 #include "stm32f4xx_wwdg.h"*/
73 #include "misc.h"*/
```

# Main



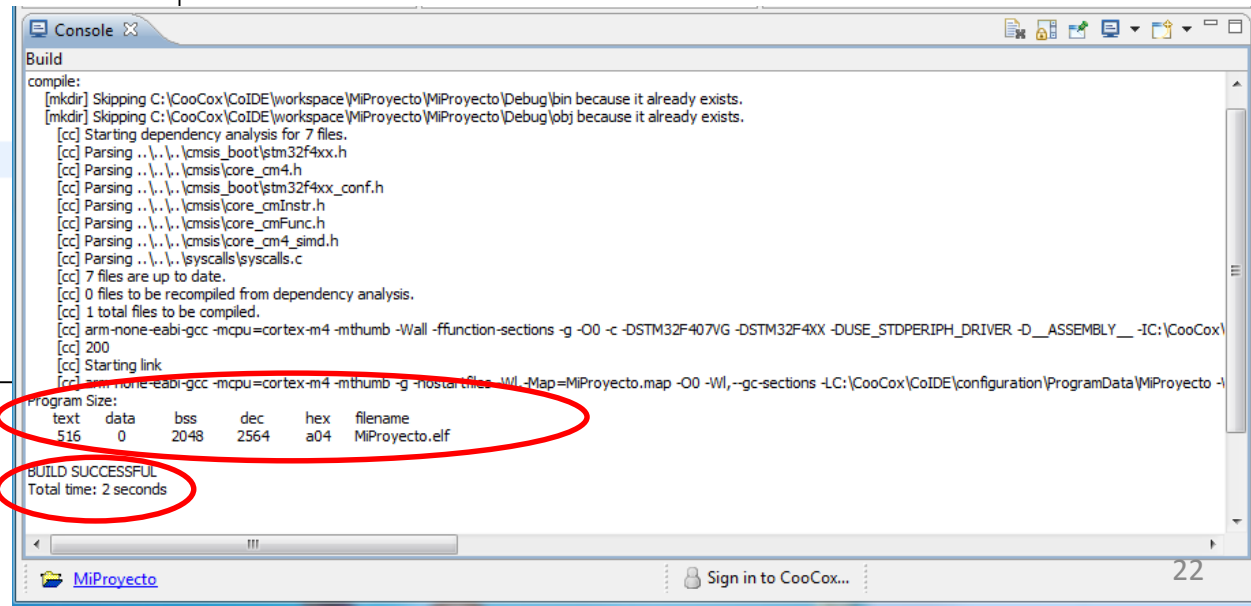
- El main aparece vacío.
- Herramientas compilación:
  1. Compilar cambios.
  2. Re-compilar todo.
  3. Entrar en modo depuración.
  4. Programar la flash

# Main

- **Antes de seguir:**

1. Incluir librerías: periféricos (*\_conf*) y del sistema (*system\_*).
2. Llamar a la inicialización del sistema: *SystemInit()*.
3. Compilar.

```
2 #include "stm32f4xx_conf.h"
3 #include "system_stm32f4xx.h"
4
5 int main(void)
6 {
7
8     SystemInit();
9
10    while(1)
11    {
12    }
13 }
14
```



```
Build
compile:
[mkdir] Skipping C:\CooCox\CoIDE\workspace\MiProyecto\MiProyecto\Debug\bin because it already exists.
[mkdir] Skipping C:\CooCox\CoIDE\workspace\MiProyecto\MiProyecto\Debug\obj because it already exists.
[cc] Starting dependency analysis for 7 files.
[cc] Parsing ..\..\..\cmsis\boot\stm32f4xx.h
[cc] Parsing ..\..\..\cmsis\core_cm4.h
[cc] Parsing ..\..\..\cmsis\boot\stm32f4xx_conf.h
[cc] Parsing ..\..\..\cmsis\core_cmInstr.h
[cc] Parsing ..\..\..\cmsis\core_cmFunc.h
[cc] Parsing ..\..\..\cmsis\core_cm4_simd.h
[cc] Parsing ..\..\..\syscalls\syscalls.c
[cc] 7 files are up to date.
[cc] 0 files to be recompiled from dependency analysis.
[cc] 1 total files to be compiled.
[cc] arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -Wall -ffunction-sections -g -O0 -c -DSTM32F407VG -DSTM32F4XX -DUSE_STDPERIPH_DRIVER -D__ASSEMBLY__ -IC:\CooCox\
[cc] 200
[cc] Starting link
[cc] arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -g -nostartfiles -Wl,-Map=MiProyecto.map -O0 -Wl,-gc-sections -LC:\CooCox\CoIDE\configuration\ProgramData\MiProyecto -l
Program Size:
text  data  bss  dec  hex  filename
516    0    2048  2564  a04  MiProyecto.elf
BUILD SUCCESSFUL
Total time: 2 seconds
```

# Tipos de datos

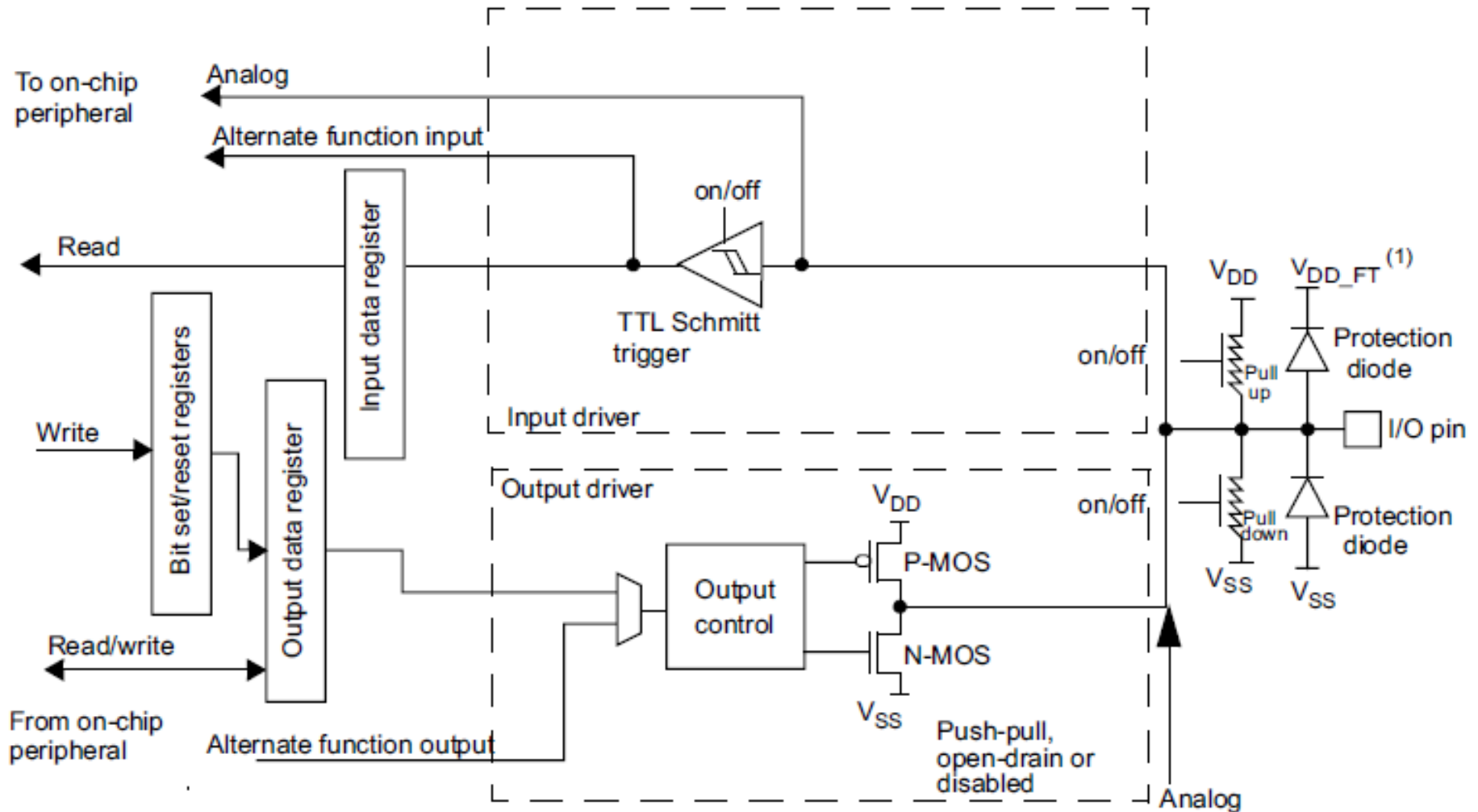
- **Comunes:**
  - char: 8 bits
  - int: 32 bits
  - float: 32 bits
  - double: 64 bits
- **Propios del STM32:**
  - 8 bits: int8\_t / uint8\_t
  - 16 bits: int16\_t / uint16\_t
  - 32 bits: int32\_t / uint32\_t

# GPIO en el STM32

- Los GPIO se clasifican con letras: GPIOA, GPIOB, GPIOC...
- Dependiendo del encapsulado tienen más o menos puertos de GPIO.
- Cada uno de ellos tiene 16 bits y controla 16 pines.
- Los GPIO tienen 4 modos:
  - **Salida:** En modo push-pull u open-drain.
  - **Entrada:** Entrada digital.
  - **Analógico:** en caso de estar conectado a un ADC/DAC.
  - **Función alternativa:** conectado internamente a un periférico, ej. un puerto serie, PWM...
- Permiten además añadir resistencias internas de **pull-up** o **pull-down**.



# GPIO en el STM32



# Registros del GPIO

#### 7.4.5 GPIO portdata register (GPIOx\_IDR) (x = A..I)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

[illegible]

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

Bits 15:0 IDRy[15:0]: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

#### 7.4.6 GPIO port output data register (GPIOx\_ODR) (x = A..I)

Address offset: 0x14

Reset value: 0x0000 0000

[illegible]

# Registros del GPIO

- Aparte de los registros de lectura/escritura de 16 bits, un registro para manejar el puerto a nivel de bits.

## 7.4.7 GPIO port bit set/reset register (GPIOx\_BSRR) (x = A..I)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 **BRy**: Port x reset bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

*Note: If both BSx and BRx are set, BSx has priority.*

Bits 15:0 **BSy**: Port x set bit y (y= 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

# Estructura de Inicialización del GPIO

- Las librerías de ST nos permite manejar los periféricos a más alto nivel, basándose en estructuras en C.
- El caso de los GPIO hay que usar una estructura de tipo: `GPIO_InitTypeDef`.
- En cada uno de sus campos se especifican las opciones de configuración de un GPIO.

## GPIO\_InitTypeDef Struct Reference

GPIO

GPIO Init structure definition. [More...](#)

```
#include <stm32f4xx_gpio.h>
```

### Data Fields

<code>uint32_t</code>	<a href="#">GPIO_Pin</a>
<a href="#">GPIO_Mode_TypeDef</a>	<a href="#">GPIO_Mode</a>
<a href="#">GPIO_Speed_TypeDef</a>	<a href="#">GPIO_Speed</a>
<a href="#">GPIO_OType_TypeDef</a>	<a href="#">GPIO_OType</a>
<a href="#">GPIO_PuPd_TypeDef</a>	<a href="#">GPIO_PuPd</a>

### Detailed Description

GPIO Init structure definition.

Pines a los que va destinada la inicialización.

Modo del pin:

1. Analógico: `GPIO_Mode_AN`
2. Entrada: `GPIO_Mode_IN`
3. Salida: `GPIO_Mode_OUT`
4. Función alternativa: `GPIO_Mode_AF`

Velocidad del gpio: `GPIO_Speed_50MHz`

Tipo de salida:

1. Push-Pull: `GPIO_OType_PP`
2. Open-Drain: `GPIO_OType_OD`

Pull-up/down:

1. Pull-Up: `GPIO_PuPd_UP`
2. Pull-Down: `GPIO_PuPd_DOWN`
3. No Pull: `GPIO_PuPd_NOPULL`

# Inicialización de dos GPIO (LEDs)

- Siempre se sigue el mismo proceso para habilitar cualquier periférico:

1. Declaración de la estructura en la que estableceremos la configuración:

```
GPIO_InitTypeDef gpio;
```

2. Habilitación del reloj del periférico (GPIOH):

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOH,  
                        ENABLE);
```

3. Rellenar la estructura de configuración(pin 2 y 3):

```
gpio.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;  
gpio.GPIO_Mode = GPIO_Mode_OUT;
```

...

4. Escribir la configuración en el periférico (GPIOH):

```
GPIO_Init(GPIOH, &gpio);
```

# Ejemplos del manejo del GPIO

- **Funciones de escritura en el GPIO:**

- `GPIO_SetBits(GPIOH, GPIO_Pin_5 );`
- `GPIO_SetBits(GPIOH, GPIO_Pin_1 | GPIO_Pin_7 | ...);`
- `GPIO_ResetBits(GPIOH, GPIO_Pin_4);`
- `GPIO_ResetBits(GPIOH, GPIO_Pin_12 | GPIO_Pin_15 | ...);`
- `GPIO_ToggleBits(GPIOH, GPIO_Pin_2 | GPIO_Pin_9 | ...);`
- `GPIO_Write(GPIOH, dato16bits);`

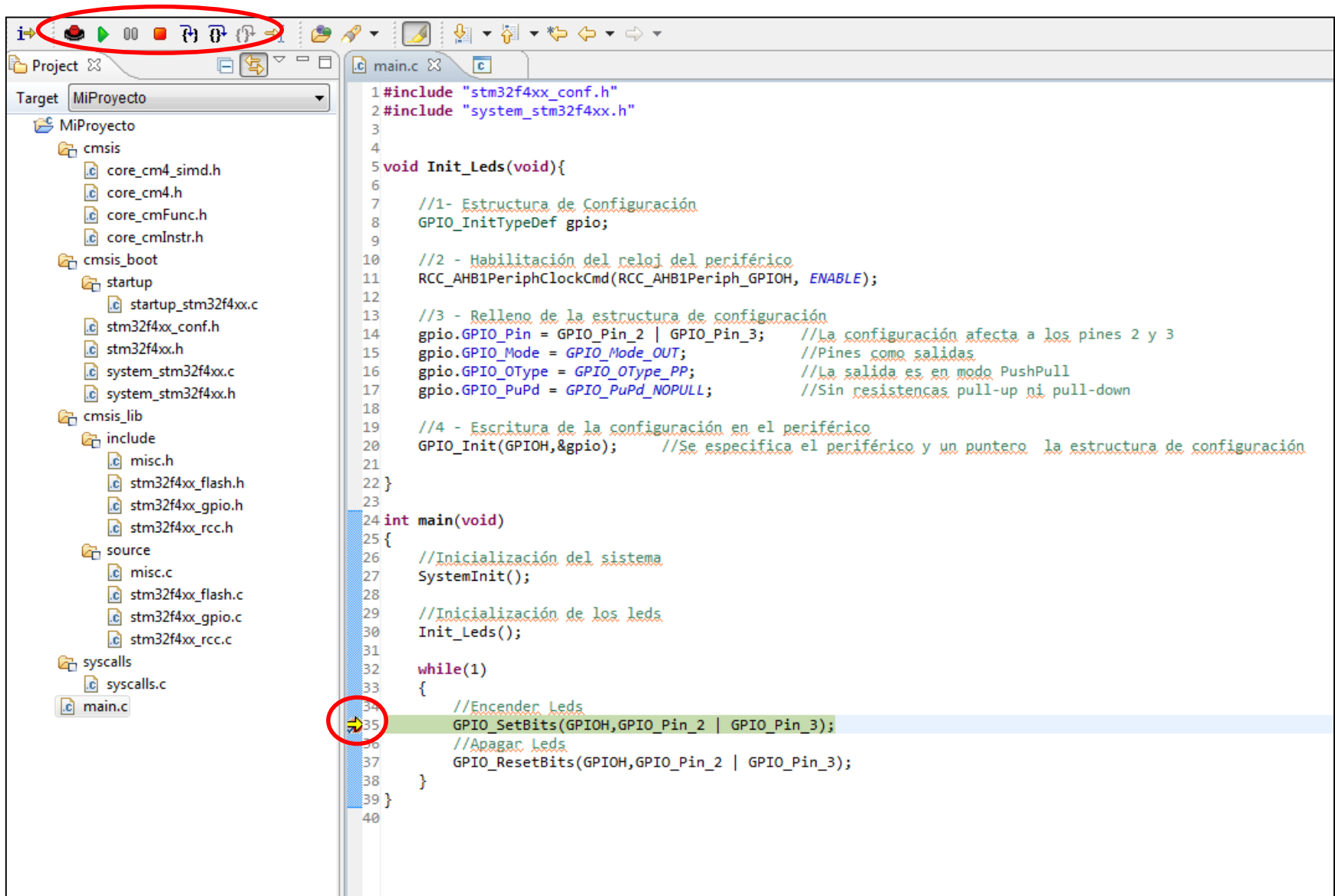
- **Funciones de lectura del GPIO:**

- `p = GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_5 );`
- `dato16bits = GPIO_Read(GPIOE);`

# Probando las inicializaciones

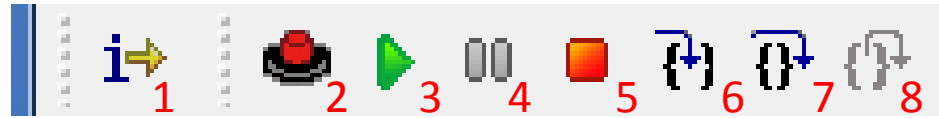
1. **Escribir una función** que configure los **pines 2 y 3** del **GPIOH** como **salidas** en modo **push-pull**, sin **pull-up/down**:
2. **Modificar el main** para que se llame a la **función de inicialización** de los LEDs.
3. **Modificar el bucle principal** que se encienda y apaguen los dos LEDs inicializados.
4. **Depurar paso a paso** para comprobar su funcionamiento.

# Depurando





# Controles de depuración



1. Mostrar Desensamblado.
2. Reset del sistema.
- 3. Ejecución.**
4. Pausar ejecución.
- 5. Salir de la sesión de depuración.**
6. Paso adentro.
7. Paso simple.
8. Paso afuera.

# Implementación de un driver Leds

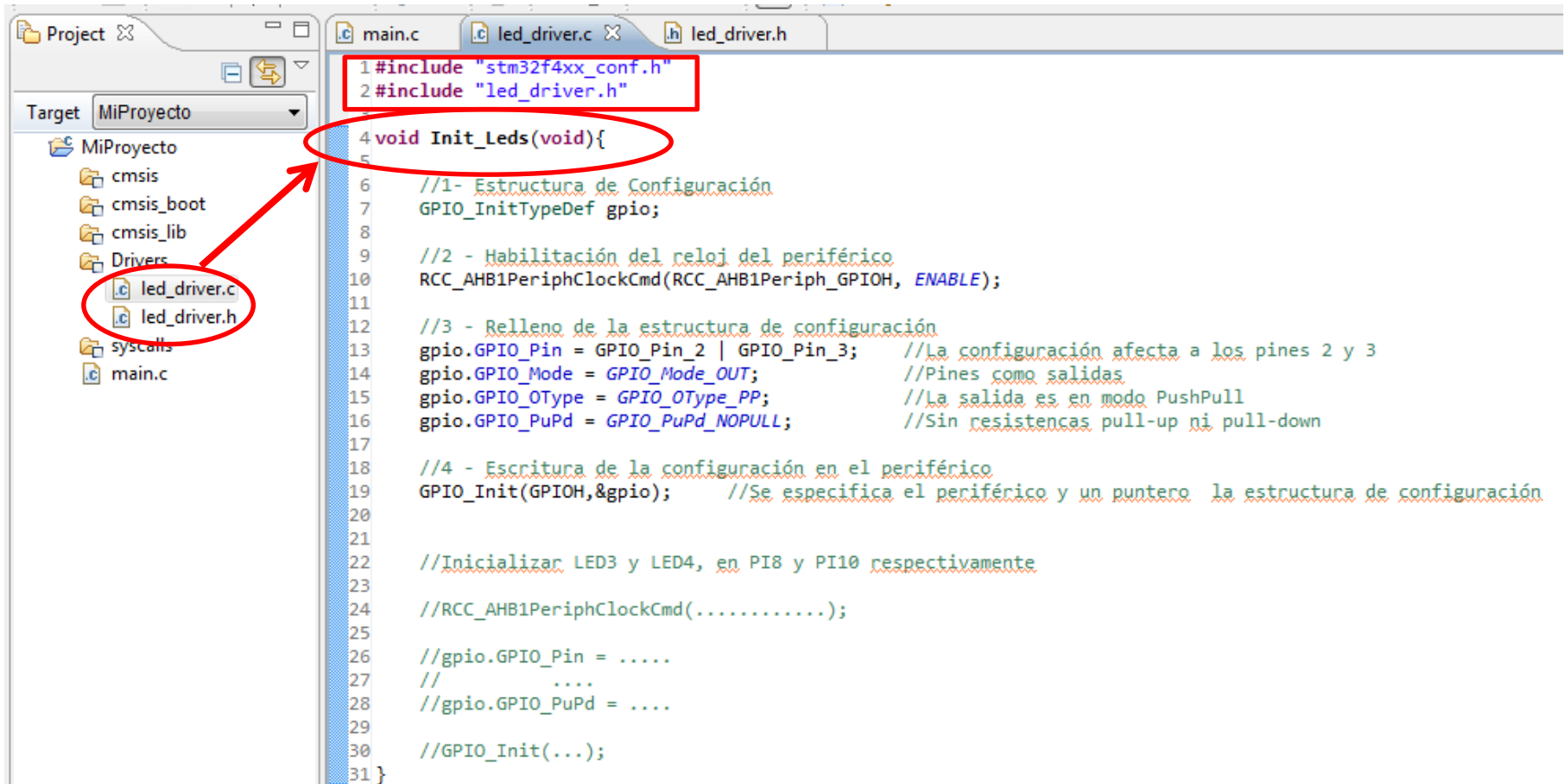
- El objetivo es **implementar un pequeño juego de funciones** que nos permitan **manejar cuatro leds desde un nivel más alto**.
- Vamos a implementar el driver en **fuentes separadas** para su **portabilidad**.
- Las funciones a implementar son:
  1. **void Init\_Leds(void)**
    - Completar con los otros dos leds.
  2. **void LED\_ON(uint8\_t led) y void LED\_OFF(uint8\_t led)**
    - Enciende/Apaga el led especificado, contados de 1 a 4, en caso de 0 enciende/apaga todos los leds.
  3. **void LED\_TOGGLE(uint8\_t led)**
    - Conmuta el estado del especificado, contados de 1 a 4, en caso de 0 conmuta todos los leds.

# Implementación de un driver Leds

- Los LEDs están conectados a **PH2 , PH3 , PI8 , PI10**.
1. Crear dos ficheros nuevos, **led\_driver.c** para las funciones, y **led\_driver.h** para los prototipos.
  2. Incluir en **led\_driver.c**, la librería de periféricos (**\_conf.h**) y **led\_driver.h**
  3. Mover la función *Init\_Leds()* del **main.c** a **led\_driver.c**.
  4. Completarla con la inicialización de los LEDs en el PI8, PI10.

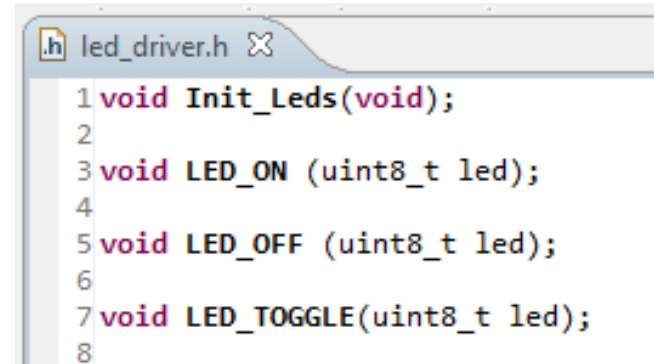
LED	Puerto
LED1	PH2
LED2	PH3
LED3	PI8
LED4	PI10

# Implementación de un driver Leds



# Implementación de un driver Leds

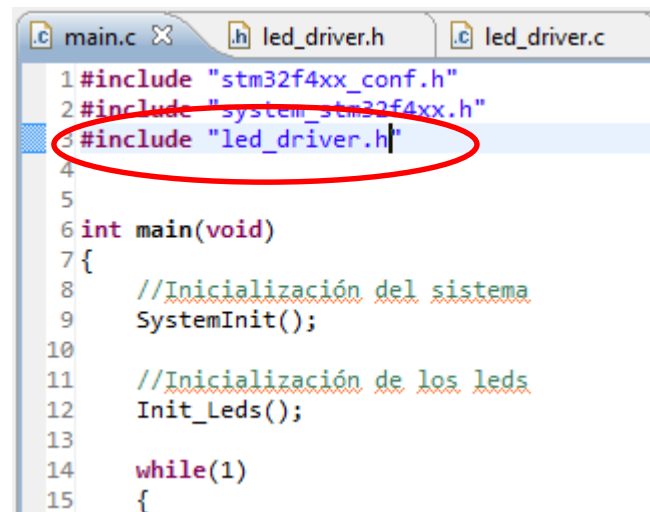
- Añadir las cabeceras de los miembros del driver en **led\_driver.h**.



A screenshot of a code editor showing the `led_driver.h` header file. The file contains four function declarations: `void Init_Leds(void);`, `void LED_ON (uint8_t led);`, `void LED_OFF (uint8_t led);`, and `void LED_TOGGLE(uint8_t led);`. The lines are numbered 1 through 8.

```
1 void Init_Leds(void);
2
3 void LED_ON (uint8_t led);
4
5 void LED_OFF (uint8_t led);
6
7 void LED_TOGGLE(uint8_t led);
8
```

- Incluir **led\_driver.h** en el main.



A screenshot of a code editor showing the `main.c` source file. The file includes three header files: `"stm32f4xx_conf.h"`, `"system_stm32f4xx.h"`, and `"led_driver.h"`. The `"led_driver.h"` line is circled in red. Below the includes, the `main` function is defined, which calls `SystemInit()` and `Init_Leds()`, and enters a `while(1)` loop.

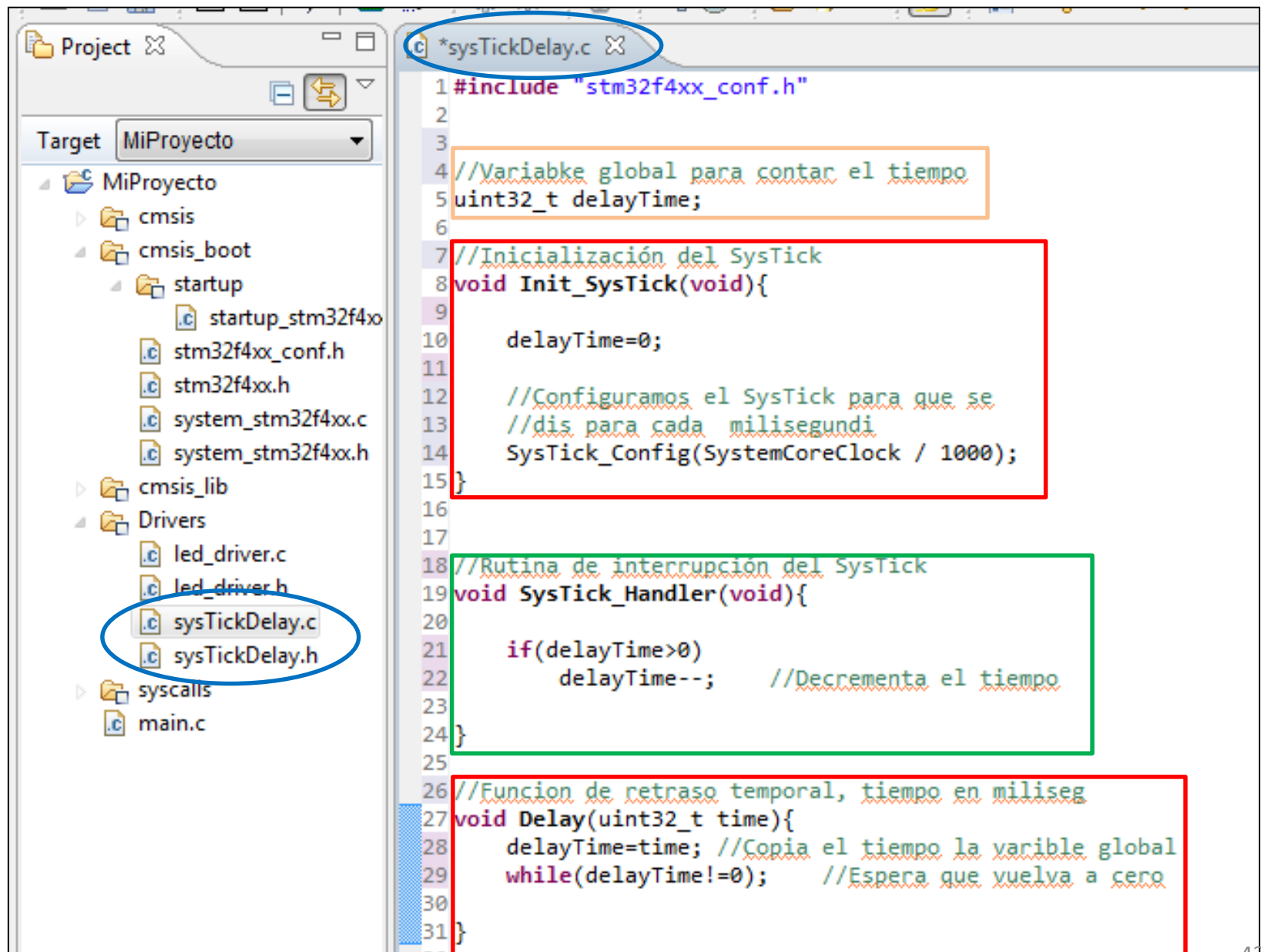
```
1 #include "stm32f4xx_conf.h"
2 #include "system_stm32f4xx.h"
3 #include "led_driver.h"
4
5
6 int main(void)
7 {
8     //Iniciación del sistema
9     SystemInit();
10
11     //Iniciación de los leds
12     Init_Leds();
13
14     while(1)
15     {
```

# Retraso Temporal: SysTick

- El **SysTick** es un timer especial de 24 bits que produce una **interrupción periódica no enmascarable**.
- El período de la interrupción se fija cada **cierto números de ciclos del reloj** del sistema del STM32.
- Comúnmente usada como **tick del sistema** por los **sistemas operativos en tiempo real**.
- Nosotros lo vamos a usar para **introducir esperas activas temporales**.

# Retraso Temporal: SysTick

- Copiar y añadir al proyecto los ficheros:
  - **sysTickDelay.c**
  - **sysTickDelay.h**
- Nos proporcionan dos funciones:
  - **Init\_SysTick (void)**: Inicializa el SysTick
  - **Delay (uint32\_t miliSeg)**: para la ejecución del programa durante un determinado tiempo en miliSegundos.





# Integrando el Retraso Temporal al main

- Se han de hacer 2 cosas en **main.c** para poder llamar a la función **Delay()**:
  1. Incluir **sysTickDelay.h** en **main.c**:
    - `#include "sysTickDelay.h"`
  2. Llamar a **Init\_SysTick()** antes del bucle principal

**Modificar el main para que los leds parpadeen cada 500miliSeg**

# Implementación de un driver de joystick

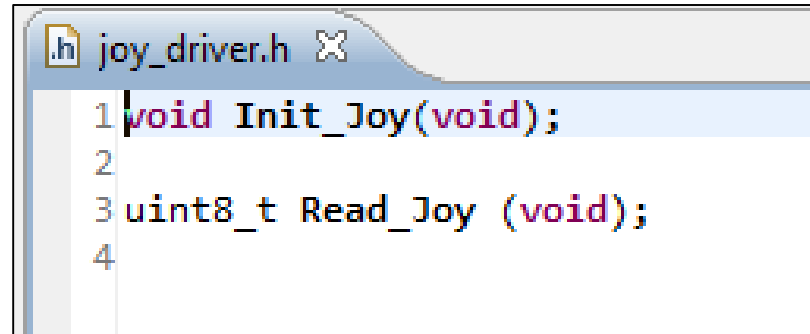
- La Open407i incluye un pequeño joystick de 4 botones (A, B, C, D).
- Crear dos ficheros nuevos:
  - **joy\_driver.c** y **joy\_driver.h**
- Implementar dos funciones:
  - No olvidar incluir **stm32f4xx\_conf.h**
  - **void Init\_Joy(void)**: inicialice los GPIO del joystick como entradas.
  - **uint8\_t Read\_Joy(void)**: devuelve el botón que se está pulsando (de 1 a 4, o 0 en caso de que no se pulse ningún botón). Hacer uso de la función:

- **GPIO\_ReadInputDataBit(puerto, pin)**

Tecla	GPIO	Devuelve
A	PE2	1
B	PE3	2
C	PE4	3
D	PE5	4
Ninguna	-	0

# Implementación de un driver de joystick

- No olvidar añadir los prototipos de las funciones en **joy\_driver.h**!



```
.h joy_driver.h X
1 void Init_Joy(void);
2
3 uint8_t Read_Joy (void);
4
```

# Integrando el driver del joystick en el main

- Modificar **main.c**:
  1. **Si no está ninguna tecla pulsada:** se conmutan todos los leds
  2. **Si alguna tecla está pulsada:** sólo conmuta el led al que corresponda la tecla

Tecla	LED
A	LED1
B	LED2
C	LED3
D	LED4
Ninguna	Todos

```
main.c X
1#include "stm32f4xx_conf.h"
2#include "system_stm32f4xx.h"
3#include "led_driver.h"
4#include "sysTickDelay.h"
5#include "joy_driver.h"
6
7
8int main(void)
9{
10    //Variable para guardar el boton pulsado
11    uint8_t joyButton;
12
13    //Inicialización del sistema
14    SystemInit();
15
16    //Inicialización de los leds
17    Init_Leds();
18
19    //Inicialización el SysTick
20    Init_SysTick();
21
22    //Inicialización del JoyStick
23    Init_Joy();
24
25    while(1)
26    {
27        //Leemos el joystick
28        joyButton=Read_Joy();
29
30        //Conmutamos el led correspondiendte al joystick
31        // 0 para todos los leds
32        LED_TOGGLE(joyButton);
33
34        //Espera n milisegundos
35        Delay(100);
36    }
37
38
39}
```

# Sistema multi-animación

- Haciendo uso de la función *Delay()*, lanzar animaciones distintas cada vez que se pulse una tecla del joystick.
- **Consejos:**
  - Agrupar las **distintas animaciones** en **funciones individuales**.
  - **Invocar a las funciones** de las animaciones apoyándose en una **estructura switch/case**.
- Ejemplo:

```
void animation1(void) {  
    int i;  
    LED_OFF(0);  
    for(i=0; i<10; i++) {  
        LED_TOOGLE(0);  
        Delay(200);  
    }  
    LED_OFF(0);  
}
```

# Ejemplos de animaciones

