|  |
| --- |
| Practica 1: |
| Introducción al STM32: Manejo de GPIO y uso del SysTick | |
| [Subtítulo del documento] |

|  |
| --- |
| Alejandro Vega  Antonio Portillo |

Objetivos

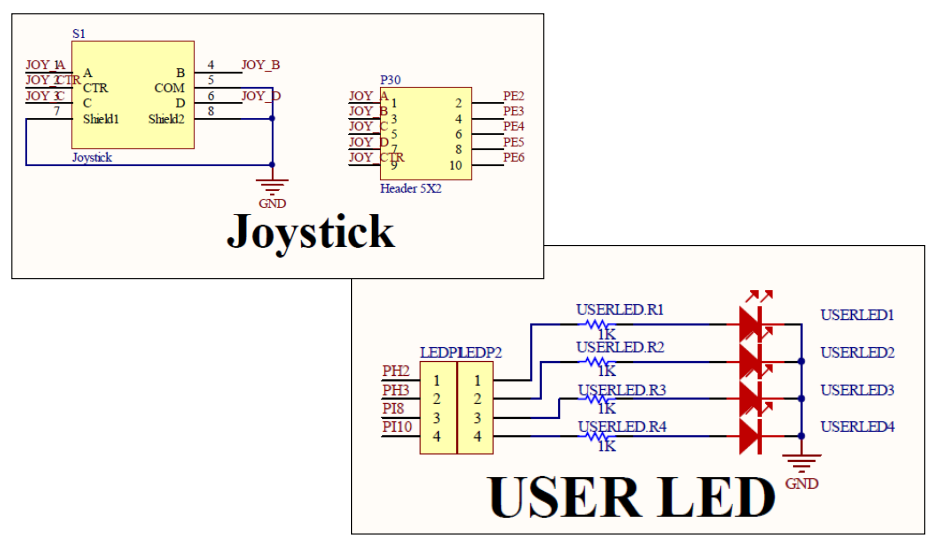
• Introducir al desarrollo con CoIDE de CooCox.  
• Familiarizarse con las librerías de ST para el manejo de periféricos.  
• Uso del GPIO del STM32.  
 – Implementar un controlador para el manejo de los LEDs de la Open407i de alto nivel.  
 – Implementar un controlador para leer el joystick digital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

Desarrollo de práctica

1. Elementos de la Open407i.

En esta práctica vamos a usar tanto los LEDs como los botones del Joystick de la placa.

Los esquemáticos son los siguientes:



* LEDs: Los pines LEDP1 y LEDP2 están unidos por pares con jumpers. Esto es porque podríamos necesitar usar esa señal pero sin activar el LED correspondiente. La señal llega a LEDP1 desde el micro, y de ahí pasa a LEDP2. El LED solo se encenderá cuando la señal en LEDP2 sea un ‘1’. Los pines asociados a estos LEDs son PH2, PH3, PI8 y PI10 con los LEDs 1, 2, 3 y 4 respectivamente.
* Joystick: Los pines asociados a los botones A, B, C y D (los que usaremos) son PE2, PE3, PE4 y PE5 respectivamente. Estos botones son activos en baja por lo que necesitan una resistencia de pull-up cada uno.

2. Creación de un proyecto con Co-IDE.

Seguimos las pautas que se marcan en la presentación de la práctica para configurar e iniciar el proyecto. Vamos a comentar las más interesantes.

Vamos a usar el IDE ‘CoIDE’ y el SOTR ‘CooCox’.

Elegimos nuestra placa (STM32F407IG)

Elegimos los componentes básicos que queremos que tenga nuestro proyecto, entre los que están la librería C, GPIO, Flash memory controller, el reset, etc...

3. Fuentes del proyecto por defecto.

Inicializamos el STM32. Tenemos varios ficheros fuente que sirven para inicializar el sistema.

Debemos abrir el fichero startup\_. El Reloj viene predeterminado a 25MHz pero nuestro reloj de cuarzo funciona a 8MHz por lo que debemos cambiar este valor (PLL\_M) para evitar el cuelgue de nuestro sistema.

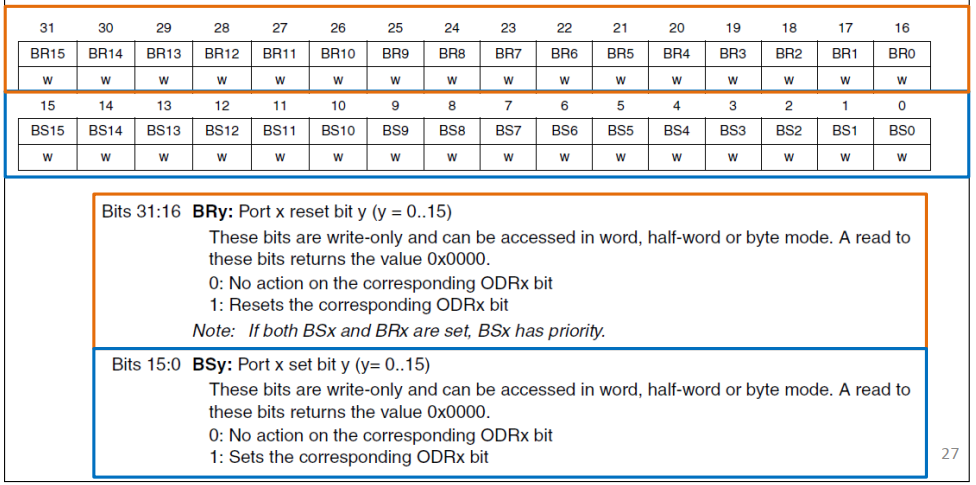
Tenemos también los ficheros System\_ donde se encuenta la función SystemInit(void) que se encarga de Inhabilitar los relojes y llamar a SetSysClock(). Por ultimo posiciona el vector de interrupciones.

Hay que incluir también las librerías de los periféricos. Para hacer esto cómodamente vamos al archivo stm32f4xx\_conf.h y descomentamos las que vamos a utilizar. Ahora solo tenemos que hacer un include de este archivo en nuestro código.

Antes de continuar compilamos para comprobar que todo está correcto.

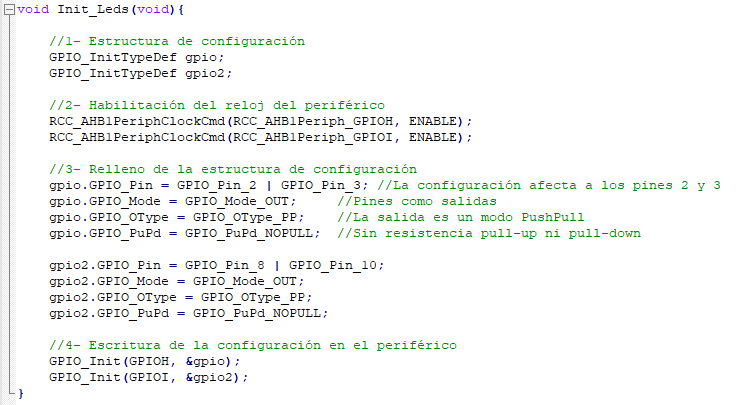
4. GPIO en el STM32.

Vamos a hacer un resumen de lo mas importante que debemos tener en cuenta acerca de los GPIO en el STM32.

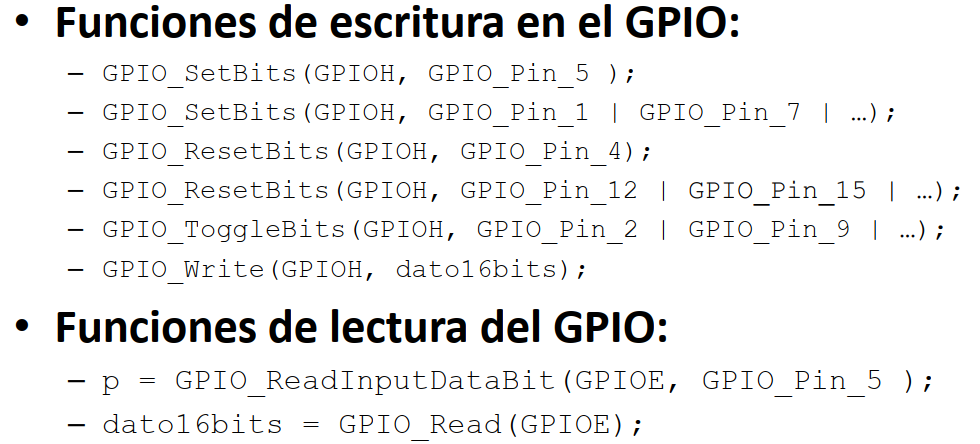
* Se clasifican con letras. GPIOA, GPIOB .. GPIOI.
* Cada uno de ellos tiene 16 bits y controla 16 pines.
* Tienen 4 modos: Salida, Entrada, Analógico y función alternativa.
* Permiten añadir resistencias internas de pull-up o pull-down.
* Existe un registro para manejar el puerto a nivel de bits. GPIOx\_BSRR. Este registro es de 32 bits y de solo escritura. 

5. Manejo de los GPIOs usando las librerías del STM32.

Para inicializar los LEDs hacemos uso de las librerías del STM32 de la siguiente forma:

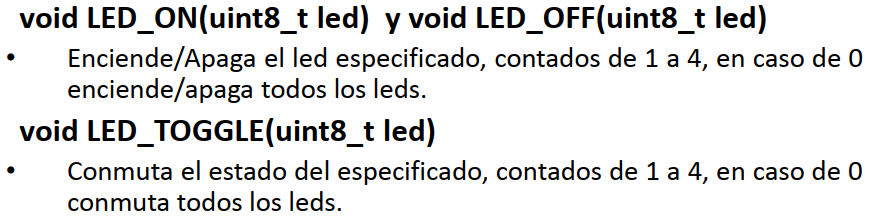


Las funciones que usaremos más adelante en el driver de los LEDs son las siguientes:

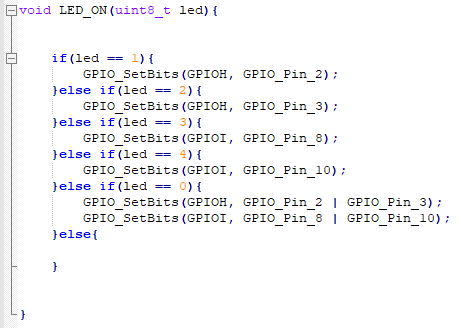


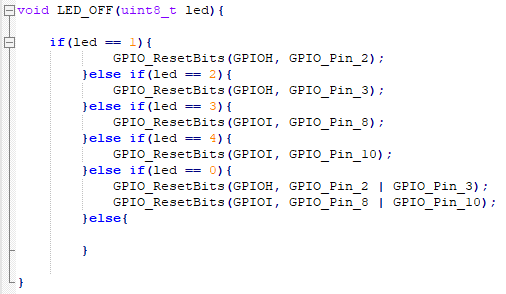
6. Implementación de un driver para manejo los leds de alto nivel.

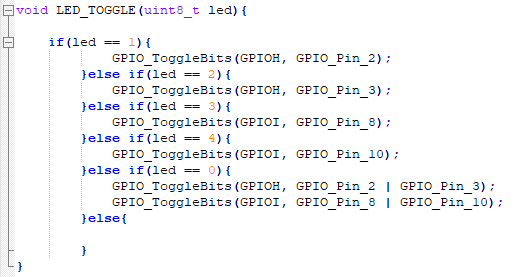
Se nos pide crear 3 funciones para el manejo de los LEDs que son las siguientes:



El código quedaría así:





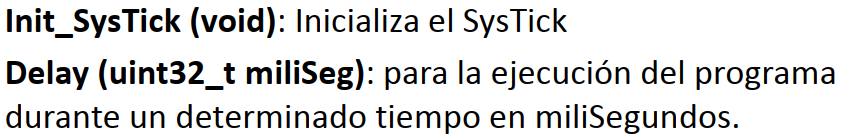


Este código junto a la función Init\_Leds(); irá en el fichero ‘led\_driver.c’ por lo que no se los puede olvidar crear y hacer el include del fichero ‘led\_driver.h’

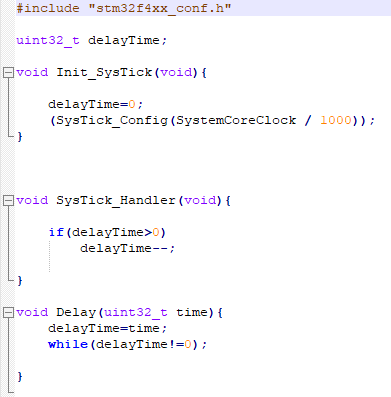
7. Uso del SysTick para introducir esperas activas.

El SysTick es un timer especial que se ejecuta en 2º plano. En esta práctica vamos a ‘usarlo mal’ ya que lo vamos a hacer es un método que lo dejará bloqueado esperando un evento. Pero como solo lo necesitamos para una tarea nos servirá (lo vamos a usar para introducir esperas activas temporales).

Debemos copiar y añadir los ficheros ‘sysTickDelay.c’ y ‘sysTickDelay.h’ a nuestro proyecto los cuales nos proporcionarán las siguientes funciones.



El fichero ‘sysTickDelay.c’ consiste en el siguiente código:



void Init\_SysTick(void): Inicializamos el sysTick y lo configuramos para que se dispare cada milisegundo.

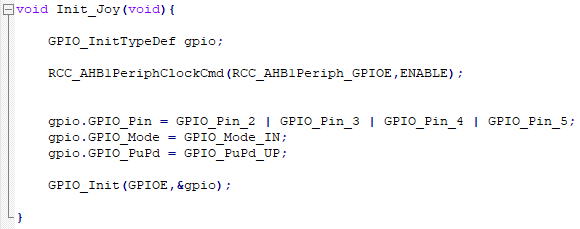
void SysTick\_Handler(void): Rutina de interrupción que haremos que se encargue de decrementar el tiempo.

void Delay(uint32\_t time): Función que usaremos para el retraso temporal. Implementa nuestra espera activa.

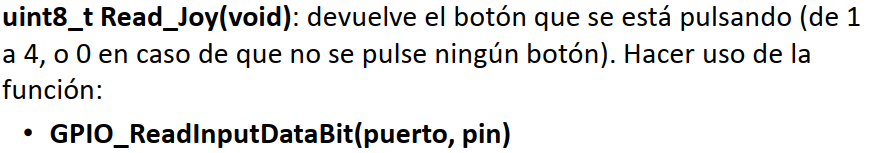
8. Implementación de un driver para leer el joystick.

Creamos los archivos ‘joy\_driver.c’ y ‘joy\_driver.h’.

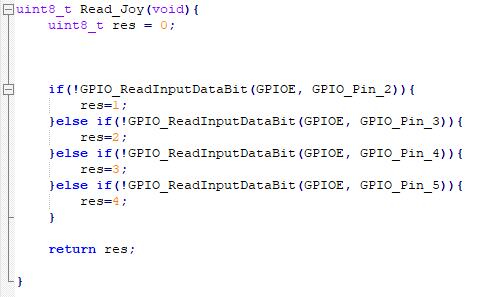
De forma análoga a como hicimos con los Leds debemos inicializar los GPIO correspondientes, esta vez como entrada y con una resistencia de Pull-up.



La función que nos piden es esta:



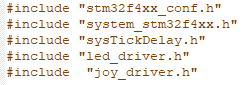
Que hemos implementado de la siguiente forma.



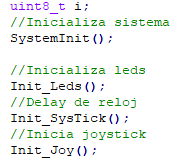
9. Diseño de animaciones basadas en retraso temporal activo.

Ahora queda jugar con todo lo que hemos configurado para ver que funciona. Para ello vamos a crear varias animaciones con los Leds que se dispararán cuando pulsemos algún botón.

Es importante incluir en nuestro main todas las librerías que vamos a utilizar, incluidas las que hemos creado nosotros.

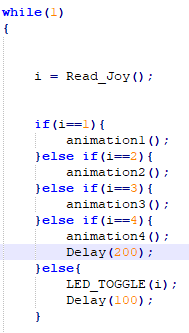
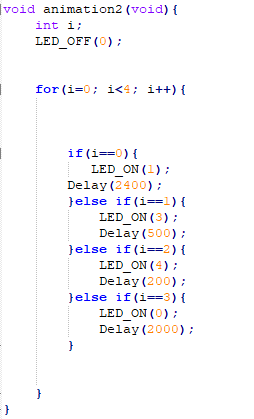


Es también critico acordarnos de usar las funciones de inicialización que hemos creado antes de entrar en el bucle de ejecución. Crearemos también una variable entera de 8 bits sin signo que usaremos para leer el joystick.



Dependiendo de la lectura del joystick usaremos una animación u otra. Como no es el objetivo de esta práctica, hemos implementado animaciones de forma bastante aleatoria. Pondremos el código de la animación2 como ejemplo.

Este es el código de nuestro bucle while:

Nota: También se podría haber usado una estructura switch/case para invocar las funciones de animación. De hecho, habría sido mas correcto.

Conclusiones