Practica 1:

Introducción al STM32: Manejo de GPIO y uso del SysTick

Alejandro Vega
Antonio Carlos Portillo

Objetivos

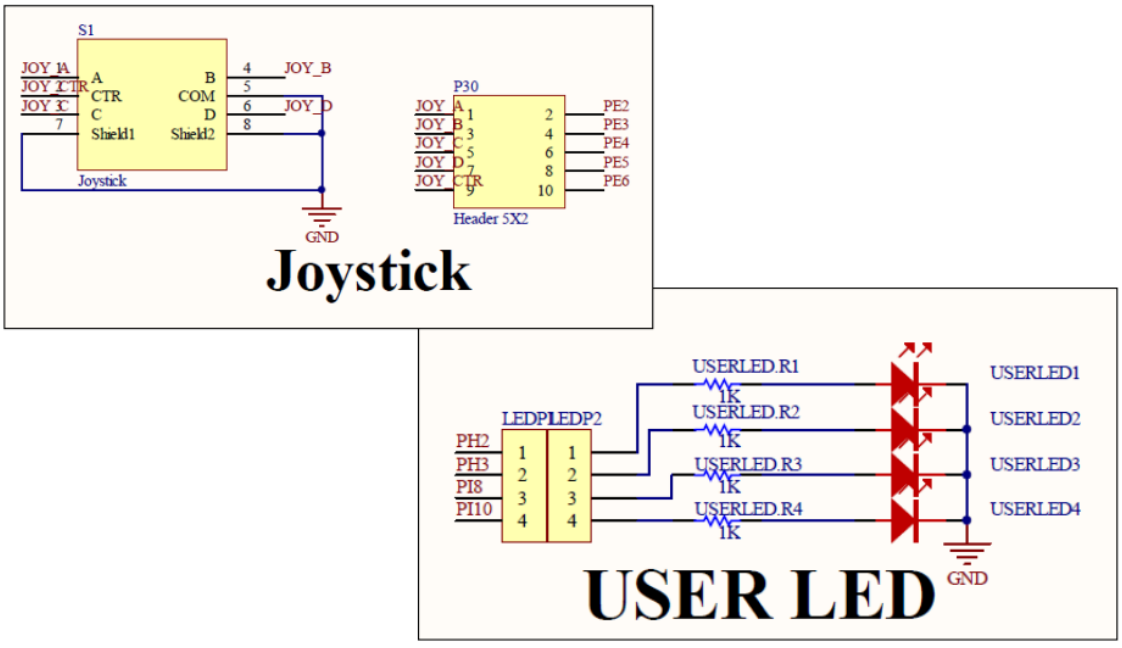
- Introducir al desarrollo con CoIDE de CooCox.
- Familiarizarse con las librerías de ST para el manejo de periféricos.
- Uso del GPIO del STM32.
  - Implementar un controlador para el manejo de los LEDs de la Open407i de alto nivel.
  - Implementar un controlador para leer el joystick digital de la 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.

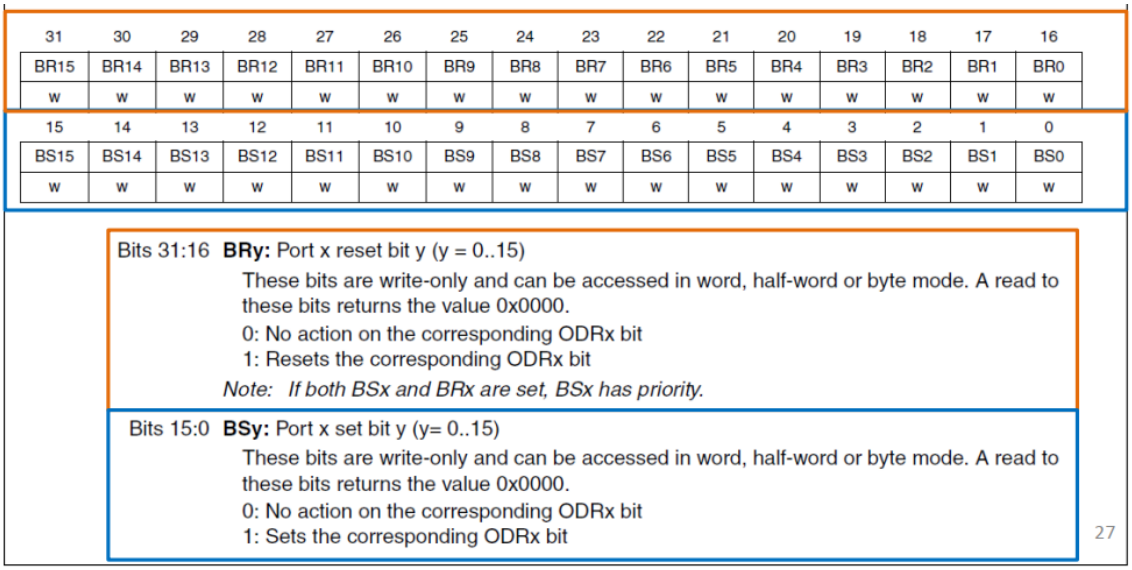
Antes de continuar compilamos para comprobar que todo está correcto.

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.



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

27

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:

```
void Init_Leds(void){

    //1- Estructura de configuración
    GPIO_InitTypeDef gpio;
    GPIO_InitTypeDef gpio2;

    //2- Habilitación del reloj del periférico
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOH, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOI, ENABLE);

    //3- Relleno de la estructura de configuración
    gpio.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3; //La configuración afecta a los pines 2 y 3
    gpio.GPIO_Mode = GPIO_Mode_OUT;     //Pines como salidas
    gpio.GPIO_OType = GPIO_OType_PP;    //La salida es un modo PushPull
    gpio.GPIO_PuPd = GPIO_PuPd_NOPULL;  //Sin resistencia pull-up ni pull-down

    gpio2.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_10;
    gpio2.GPIO_Mode = GPIO_Mode_OUT;
    gpio2.GPIO_OType = GPIO_OType_PP;
    gpio2.GPIO_PuPd = GPIO_PuPd_NOPULL;

    //4- Escritura de la configuración en el periférico
    GPIO_Init(GPIOH, &gpio);
    GPIO_Init(GPIOI, &gpio2);
}
```

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

- **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);`

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:

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.

void LED_TOGGLE(uint8_t led)

- Conmuta el estado del especificado, contados de 1 a 4, en caso de 0 conmuta todos los leds.

El código quedaría así:

```
void LED_ON(uint8_t led){


    if(led == 1){
        GPIO_SetBits(GPIOH, GPIO_Pin_2);
    }else if(led == 2){
        GPIO_SetBits(GPIOH, GPIO_Pin_3);
    }else if(led == 3){
        GPIO_SetBits(GPIOI, GPIO_Pin_8);
    }else if(led == 4){
        GPIO_SetBits(GPIOI, GPIO_Pin_10);
    }else if(led == 0){
        GPIO_SetBits(GPIOH, GPIO_Pin_2 | GPIO_Pin_3);
        GPIO_SetBits(GPIOI, GPIO_Pin_8 | GPIO_Pin_10);
    }else{

    }


}
```

```
void LED_OFF(uint8_t led){

    if(led == 1){
        GPIO_ResetBits(GPIOH, GPIO_Pin_2);
    }else if(led == 2){
        GPIO_ResetBits(GPIOH, GPIO_Pin_3);
    }else if(led == 3){
        GPIO_ResetBits(GPIOI, GPIO_Pin_8);
    }else if(led == 4){
        GPIO_ResetBits(GPIOI, GPIO_Pin_10);
    }else if(led == 0){
        GPIO_ResetBits(GPIOH, GPIO_Pin_2 | GPIO_Pin_3);
        GPIO_ResetBits(GPIOI, GPIO_Pin_8 | GPIO_Pin_10);
    }else{

    }

}
```

```
void LED_TOGGLE(uint8_t led){

    if(led == 1){
        GPIO_ToggleBits(GPIOH, GPIO_Pin_2);
    }else if(led == 2){
        GPIO_ToggleBits(GPIOH, GPIO_Pin_3);
    }else if(led == 3){
        GPIO_ToggleBits(GPIOI, GPIO_Pin_8);
    }else if(led == 4){
        GPIO_ToggleBits(GPIOI, GPIO_Pin_10);
    }else if(led == 0){
        GPIO_ToggleBits(GPIOH, GPIO_Pin_2 | GPIO_Pin_3);
        GPIO_ToggleBits(GPIOI, GPIO_Pin_8 | GPIO_Pin_10);
    }else{

    }

}
```

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.

Init_SysTick (void): Inicializa el SysTick

Delay (uint32_t miliSeg): para la ejecución del programa durante un determinado tiempo en miliSegundos.

El fichero 'sysTickDelay.c' consiste en el siguiente código:

```
#include "stm32f4xx_conf.h"

uint32_t delayTime;

void Init_SysTick(void){

    delayTime=0;
    (SysTick_Config(SystemCoreClock / 1000));
}



void SysTick_Handler(void){

    if(delayTime>0)
        delayTime--;

}

void Delay(uint32_t time){
    delayTime=time;
    while(delayTime!=0);

}
```

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.

```
void Init_Joy(void){

    GPIO_InitTypeDef gpio;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE,ENABLE);


    gpio.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5;
    gpio.GPIO_Mode = GPIO_Mode_IN;
    gpio.GPIO_PuPd = GPIO_PuPd_UP;

    GPIO_Init(GPIOE,&gpio);

}
```

La función que nos piden es esta:

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)**

Que hemos implementado de la siguiente forma.

```
uint8_t Read_Joy(void){
    uint8_t res = 0;



    if(!GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_2)){
        res=1;
    }else if(!GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_3)){
        res=2;
    }else if(!GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_4)){
        res=3;
    }else if(!GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_5)){
        res=4;
    }

    return res;

}
```

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.

```
#include "stm32f4xx_conf.h"
#include "system_stm32f4xx.h"
#include "sysTickDelay.h"
#include "led_driver.h"
#include  "joy_driver.h"
```

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.

```
uint8_t i;
//Inicializa sistema
SystemInit();

//Inicializa leds
Init_Leds();
//Delay de reloj
Init_SysTick();
//Inicia joystick
Init_Joy();
```

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:

```
while(1)
{


    i = Read_Joy();


    if(i==1){
        animation1();
    }else if(i==2){
        animation2();
    }else if(i==3){
        animation3();
    }else if(i==4){
        animation4();
        Delay(200);
    }else{
        LED_TOGGLE(i);
        Delay(100);
    }
```

```
void animation2(void){
    int i;
    LED_OFF(0);


    for(i=0; i<4; i++){



        if(i==0){
           LED_ON(1);
        Delay(2400);
        }else if(i==1){
            LED_ON(3);
            Delay(500);
        }else if(i==2){
            LED_ON(4);
            Delay(200);
        }else if(i==3){
            LED_ON(0);
            Delay(2000);
        }


    }
}
```

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

Conclusiones

En conclusión, se ha podido realizar la práctica de manera satisfactoria. Siendo una primera aproximación al entorno CoIDE de CooCox y al uso de las distintas funciones del mismo, así como a la placa de STM32.