

TEMA 7: INTERRUPCIONES (IRQS) Y EXTI (EXTERNAL INTERRUPTS)

Sistemas Digitales basados en Microprocesador
Grado en Ingeniería Telemática

© Raúl Sánchez Reillo

1





ÍNDICE

- Conceptos Previos: Gestión de Interrupciones
- El Controlador de Interrupciones Vectorizadas (NVIC)
- Las Entradas de Interrupción Externa (EXTI)
- EXTI: Registros de Control
- EXTI: Registros de Datos
- EXTI: Registros de Estado
- Ejemplo de Uso de EXTI
- Ejercicios



GESTIÓN DE INTERRUPCIONES

- El mundo externo se puede comunicar con la CPU de dos formas:
 - A través de accesos controlados de la CPU a los periféricos
 - A través de interrupciones que los elementos externos a la CPU provoquen en ésta
- Las interrupciones que reciba la CPU van a poder clasificarse en dos tipos:
 - **No Enmascarables:** Aquellas que, cada vez que ocurren, provocan la atención de la CPU
 - **Enmascarables:** Aquellas que la CPU puede inhibir en determinados momentos, según el interés del programador
 - Cada fuente de interrupción puede, a su vez, tener varios eventos que la provoquen
 - Además, cada fuente de interrupción puede tener, a su vez, sus propios mecanismos de habilitación/inhibición.
 - En caso de que el mecanismo de habilitación de la fuente esté habilitada, pero no el de la CPU, entonces no se habla de interrupciones, sino de **eventos**.
 - Los eventos no se gestionan por funciones, sino por mecanismos de **espera activa**.



GESTIÓN DE INTERRUPCIONES

- Pasos que da la CPU cuando ocurre una petición de interrupción:
 - Se comprueba si está habilitada esa fuente de interrupción
 - Si no, se ignora. Si es que sí, se siguen los siguientes pasos
 - Se termina la ejecución de la instrucción en curso
 - Guarda el contexto actual (registros de estado, dirección de la instrucción a ejecutar, etc.)
 - Modifica el contenido del PC, para ubicar en él la dirección de la **Rutina de Atención a la Interrupción (RAI)**, que también se conoce como **Rutina de Servicio**, correspondiente a esa fuente de interrupción
 - Se inicia el nuevo ciclo de fetch-execute. Es decir, se inicia la ejecución de la RAI.
- Cuando se termina la RAI:
 - Se recupera el contexto anteriormente guardado
 - Se carga en el PC la dirección de la siguiente dirección a ejecutar (donde se había quedado el programa cuando saltó la RAI).
 - Se prosigue la ejecución del programa.



GESTIÓN DE INTERRUPCIONES

- Hay que recordar los conceptos básicos de Interrupciones:
 - Para evitar que la CPU se quede “colgada”, las Rutinas de Servicio deben ser lo más pequeñas posible
 - Dentro de lo razonable, ya que la Rutina de Servicio debe ser funcional (que realice la función para la que está encomendada)
 - Nunca se debe poner una *Espera Activa* en una Rutina de Servicio
 - Si la Rutina de Servicio le debe comunicar algo al programa principal, lo tendrá que hacer a través de posiciones de memoria RAM
 - Ya que, como la Interrupción puede surgir en cualquier momento, no se sabrá para qué se pueda estar utilizando cada uno de los registros internos
 - En general, es buena práctica inhibir las interrupciones al entrar en la Rutina de Servicio, y volver a habilitarlas antes de salir

TEMA 7: INTERRUPCIONES (IRQS) Y EXTI (EXTERNAL INTERRUPTS)

El controlador de Interrupciones Vectorizadas (NVIC)

6





- La Gestión de las interrupciones por parte de la CPU se lleva a cabo utilizando el ***Nested Vectored Interrupt Controller (NVIC)***
 - Permite 45 fuentes de interrupción enmascarables
 - Puede permitir establecer niveles de prioridad programables (hasta 16)
 - Establece un sistema por defecto de fuentes de interrupción y su prioridad
 - En la Tabla 28 del Reference Manual están todas las fuentes.
 - La siguiente transparencia muestra una tabla con las fuentes de interés para el curso
- El uVision4 ya tiene unos nombres de RAI definidos para cada fuente de IRQ (ver tabla)



Posición	Prioridad	Acrónimo	Descripción	Nombre de la RAI
6	13	EXTI0	EXTI Line 0	EXTI0_IRQHandler
7	14	EXTI1	EXTI Line 1	EXTI1_IRQHandler
8	15	EXTI2	EXTI Line 2	EXTI2_IRQHandler
9	16	EXTI3	EXTI Line 3	EXTI3_IRQHandler
10	17	EXTI4	EXTI Line 4	EXTI4_IRQHandler
18	25	ADC1	ADC1 global IRQ	ADC1_IRQHandler
19	26	USB_HP	USB High Priority IRQ	USB_HP_IRQHandler
20	27	USB_LP	USB Low Priority IRQ	USB_LP_IRQHandler
21	28	DAC	D/A Converter	DAC_IRQHandler
23	30	EXTI9_5	EXTI Line[9:5]	EXTI9_5_IRQHandler
28	35	TIM2	TIM2 global IRQ	TIM2_IRQHandler
29	36	TIM3	TIM3 global IRQ	TIM3_IRQHandler
30	37	TIM4	TIM4 global IRQ	TIM4_IRQHandler
31	38	I2C1_EV	I ² C1 Event	I2C1_EV_IRQHandler
32	39	I2C1_ER	I ² C1 Error	I2C1_ER_IRQHandler
35	42	SPI1	SPI1 global IRQ	SPI1_IRQHandler
37	44	USART1	USART1 global IRQ	USART1_IRQHandler
40	47	EXTI15_10	EXTI Line[15:10]	EXTI15_10_IRQHandler
41	48	RTC_Alarm	RTC Alarms	RTC_Alarm_IRQHandler



NVIC

- El NVIC se controla con los siguientes registros de 32 bits, donde cada bit se refiere a una de las posiciones de interrupción:
 - **NVIC->ISER[x] : Habilita la fuente de interrupción correspondiente al bit en el que se escriba un '1' del registro x**
 - Los bits a 0 no modifican el estado de enmascaramiento
 - ISER[0] cubre los bits para las primeras 32 fuentes de interrupción
 - ISER[1] cubre los bits para las fuentes 32 a 63, correspondiendo el bit 0 a la fuente 32
 - **NVIC->ICER[x] : Enmascara la fuente de interrupción correspondiente al bit en el que se escriba un '1' del registro x**
 - Los bits a 0 no modifican el estado de enmascaramiento
 - ICER[0] e ICER[1] análogo a ISER[x]

Name	Type	Reset	Description
ICTR	RO	-	<i>Interrupt Controller Type Register, ICTR</i>
NVIC_ISER0 - NVIC_ISER7	RW	0x00000000	Interrupt Set-Enable Registers
NVIC_ICER0 - NVIC_ICER7	RW	0x00000000	Interrupt Clear-Enable Registers
NVIC_ISPR0 - NVIC_ISPR7	RW	0x00000000	Interrupt Set-Pending Registers
NVIC_ICPR0 - NVIC_ICPR7	RW	0x00000000	Interrupt Clear-Pending Registers

TEMA 7: INTERRUPCIONES (IRQs) Y EXTI (EXTERNAL INTERRUPTS)

Las Entradas Externas de Interrupción
(EXTI)

10





ENTRADAS DE INTERRUPCIÓN EXTERNA

- Como se ha podido ver, cada periférico tiene sus propias fuentes de interrupción
- La fuente de interrupción del GPIO se denomina External Interrupts (EXTI).
- Todos los pines de los GPIO pueden ser configurados para ser utilizados como una EXTI
- Hay 16 EXTI (EXTI0 – EXTI15) distintas (una por cada pin del puerto)
 - Cada EXTI puede elegir entre cada uno de los distintos puertos (GPIOA – GPIOD)
- Cada EXTI puede configurarse para que salte:
 - Por flanco de subida
 - Por flanco de bajada
 - Por ambos flancos
- Aunque se llame External Interrupt, se puede utilizar por **evento**, simplemente no activando el NVIC.
- Una vez que se ha gestionado el evento o la IRQ, hay que limpiar el flag que indica que el evento ha ocurrido.
- Los pines a ser utilizado como EXTI deberán estar configurado como Digital Input



EXTI: REGISTROS DE CONTROL

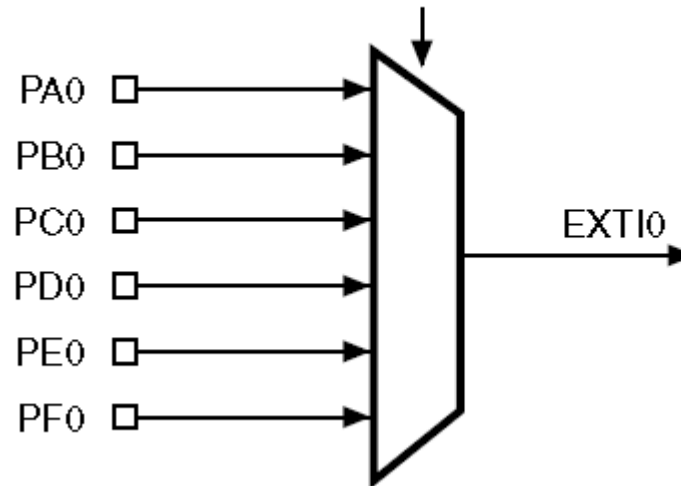
○ SYSCFG → EXTICR – EXTI Control Register:

- Conjunto de registros de 16 bits (p. ej. SYSCFG → EXTICR[0]), en el que cada grupo de 4 bits indica el puerto asociado con la EXTI Line

- 0000 – GPIOA
- 0001 – GPIOB
- 0010 – GPIOC
- 0011 – GPIOD

- Cada registro son 4 pines

EXTI0[3:0] bits in SYSCFG_EXTICR1 register



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



EXTI: REGISTROS DE CONTROL

- **EXTI→IMR** – Interrupt Mask Register:
 - Registro de 32 bits (EXTI→IMR), con sólo 16 útiles, uno por cada pin (bit0 = EXTI0, bit15 = EXTI15)
 - Un '1' habilita esa línea de EXTI
 - Un '0' enmascara esa línea de EXTI
- **EXTI→RTSR** – Rising Edge Trigger Selection Register:
 - Registro de 32 bits (EXTI→RTSR), con sólo 16 útiles, uno por cada pin (bit0 = EXTI0, bit15 = EXTI15)
 - Un '1' habilita el evento por flanco de subida
 - Un '0' inhabilita el evento por flanco de subida
- **EXTI→FTSR** – Falling Edge Trigger Selection Register:
 - Registro de 32 bits (EXTI→FTSR), con sólo 16 útiles, uno por cada pin (bit0 = EXTI0, bit15 = EXTI15)
 - Un '1' habilita el evento por flanco de bajada
 - Un '0' inhabilita el evento por flanco de bajada



EXTI: REGISTROS DE DATOS

- El EXTI no tiene Registro de Datos



EXTI: REGISTROS DE ESTADO

○ EXTI→PR – Pending Register:

- Registro de 32 bits (EXTI→PR), con sólo 16 útiles, uno por cada pin (bit0 = EXTI0, bit15 = EXTI15)
 - Un '1' indica que ha saltado ese evento
 - Un '0' indica que no ha saltado
- Para limpiar el flag que se encuentre a '1' hay que escribir un '1' en dicho bit
 - Escribir '0' no afecta al estado del bit

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

TEMA 7: INTERRUPCIONES (IRQs) Y EXTI (EXTERNAL INTERRUPTS)

Ejemplos y Ejercicios

16





EJEMPLO DE USO POR EVENTOS (1)

- El siguiente ejemplo modifica el ejercicio de usar el botón USER para cambiar los estados del LED Verde y del LED Azul:

```
# include "stm3211xx.h"
#include "Biblioteca_SDM.h"

int main(void){
    unsigned char estado = 0;
    Init_SDM();

    // PB6 (LED Azul) como salida
    GPIOB->MODER &= ~(1 << (6*2 +1));
    GPIOB->MODER |= (1 << (6*2));

    // PB7 (LED Verde) como salida
    GPIOB->MODER &= ~(1 << (7*2 +1));
    GPIOB->MODER |= (1 << (7*2));

    // Salidas como push-pull y velocidad lenta
    GPIOB->OTYPER &= ~(1 << 6);
    GPIOB->OTYPER &= ~(1 << 7);
    GPIOB->OSPEEDR &= ~(1 << (6*2 +1));
    GPIOB->OSPEEDR &= ~(1 << (6*2));
    GPIOB->OSPEEDR &= ~(1 << (7*2 +1));
    GPIOB->OSPEEDR &= ~(1 << (7*2));

    // PA0 (Boton User) como entrada (00)
    GPIOA->MODER &= ~(1 << (0*2 +1));
    GPIOA->MODER &= ~(1 << (0*2));

    // Entradas sin pull-up, pull-down (00)
    GPIOA->PUPDR &= ~(11 << (0*2));

    // Configuración de EXTI0 por flanco de bajada
    EXTI->FTSR |= 0x01; // Un '1' habilita el evento por flanco de bajada en EXTI0
    EXTI->RTSR &= ~(0x01); // Un '0' inhabilita el evento por flanco de subida en EXTI0
    SYSCFG->EXTICR[0] = 0; // La EXTI0 la provoca el bit 0 del GPIOA (el botón USER = PA0)
    EXTI->IMR |= 0x01; // Un '1' habilita la EXTI0, no la enmascara
```



EJEMPLO DE USO POR EVENTOS (2)

- El siguiente ejemplo modifica el ejercicio de usar el botón USER para cambiar los estados del LED Verde y del LED Azul:

```
while (1) {
    if (EXTI->PR!=0)          // Espera activa del evento. Si se pulsa PA0, o sea hay una interrupción
                             // externa en PA0 (EXTI0),pero no salto al NVIC, consulto las EXTI0 en
                             // el programa principal, cambio el estado de los LEDs en el
                             // programa principal y sigo con el programa cuando acabe
    {
        estado++;           // Cambio el estado de los LEDs
        if (estado > 3) estado = 0; // Si he llegado al estado 3, vuelvo al 0
        EXTI->PR = 0x01;     // Limpio el flag de EXTI0 para la siguiente vez
    }
    switch(estado) {
        case 0: // Enciende el LED Verde y no el LED Azul
                GPIOB->BSRRL = (1<<7);
                GPIOB->BSRRH = (1<<6);
                break;

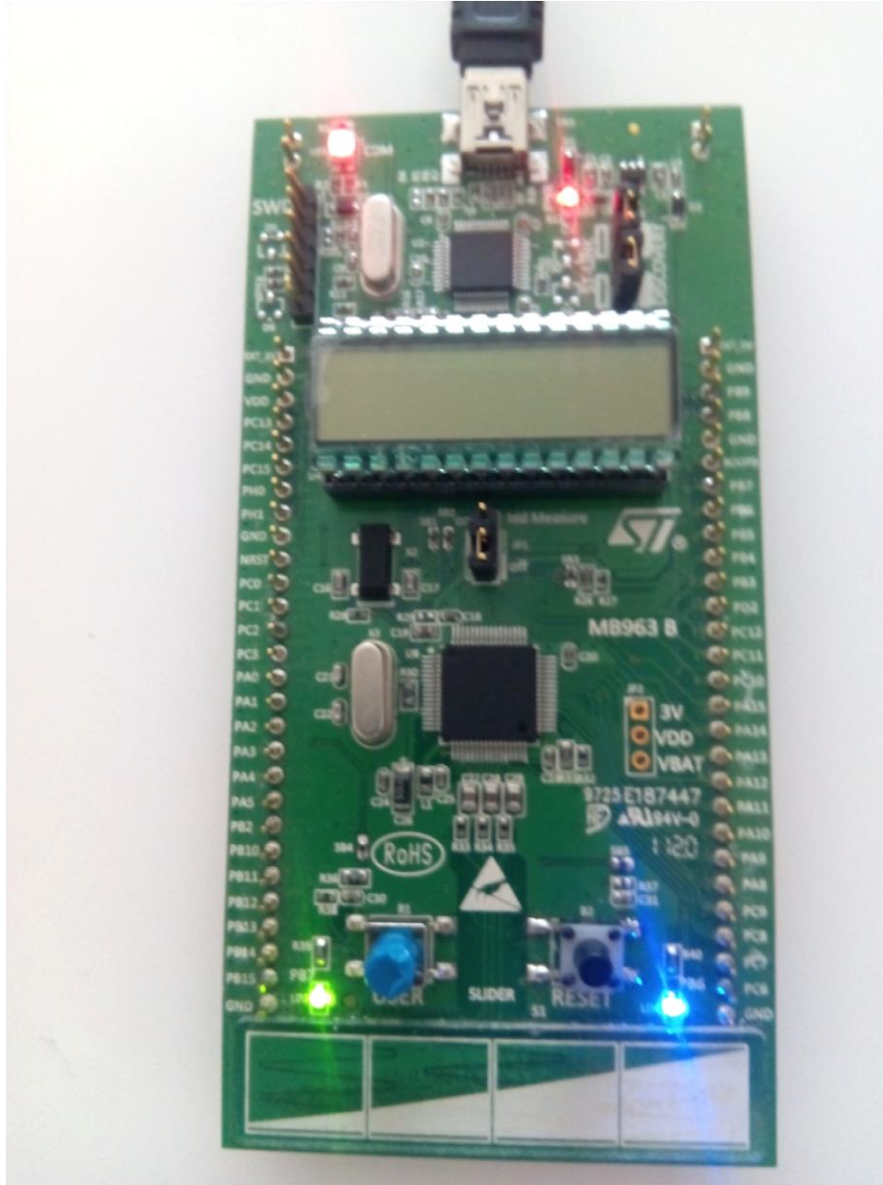
        case 1: // Enciende el LED Verde y el LED Azul
                GPIOB->BSRRL = (1<<7);
                GPIOB->BSRRL = (1<<6);
                break;

        case 2: // Enciende el LED Azul y no el LED Verde
                GPIOB->BSRRH = (1<<7);
                GPIOB->BSRRL = (1<<6);
                break;

        default: // Apaga el LED Verde y el LED Azul
                GPIOB->BSRRH = (1<<7);
                GPIOB->BSRRH = (1<<6);
    }
}
}
```



PRUEBA DEL EJEMPLO EXPLICADO





EJEMPLO DE USO POR IRQs (1)

- El siguiente ejemplo modifica el anterior, utilizando IRQs:

```
#include "stm3211xx.h"
#include "Biblioteca_SDM.h"

unsigned char estado = 0;
unsigned char estado_ant = 0;

void EXTI0_IRQHandler(void)    // Programa de tratamiento de la EXTI0.
                               // El PC salta aquí en cuanto se produzca la EXTI0
{
    if (EXTI->PR!=0)           // Si se pulsa PA0, o sea hay una interrupción
                               // externa en PA0 (EXTI0), cambio el estado de los LEDs
                               // y sigo con el programa
    {
        estado++;              // Cambio el estado de los LEDs
        if (estado > 3) estado = 0; // Si he llegado al estado 3, vuelvo al 0
        EXTI->PR = 0x01;       // Limpio el flag de EXTI0 para la siguiente vez
    }
}
```



EJEMPLO DE USO POR IRQs (2)

- El siguiente ejemplo modifica el anterior, utilizando IRQs:

```
int main(void){
    Init_SDM();

    // PB6 (LED Azul) como salida
    GPIOB->MODER &= ~(1 << (6*2 +1));
    GPIOB->MODER |= (1 << (6*2));

    // PB7 (LED Verde) como salida
    GPIOB->MODER &= ~(1 << (7*2 +1));
    GPIOB->MODER |= (1 << (7*2));

    // Salidas como push-pull y velocidad lenta
    GPIOB->OTYPER &= ~(1 << 6);
    GPIOB->OTYPER &= ~(1 << 7);
    GPIOB->OSPEEDR &= ~(1 << (6*2 +1));
    GPIOB->OSPEEDR &= ~(1 << (6*2));
    GPIOB->OSPEEDR &= ~(1 << (7*2 +1));
    GPIOB->OSPEEDR &= ~(1 << (7*2));

    // PA0 (Boton User) como entrada (00)
    GPIOA->MODER &= ~(1 << (0*2 +1));
    GPIOA->MODER &= ~(1 << (0*2));

    // Entradas sin pull-up, pull-down (00)
    GPIOA->PUPDR &= ~(11 << (0*2));

    // Configuración de EXTI0 por flanco de bajada
    EXTI->FTSR |= 0x01;           // Un '1' habilita el evento por flanco de bajada en EXTI0
    EXTI->RTSR &= ~(0x01);       // Un '0' inhabilita el evento por flanco de subida en EXTI0
    SYSCFG->EXTICR[0] = 0;      // La EXTI0 la provoca el bit 0 del GPIOA (el botón USER = PA0)
    EXTI->IMR |= 0x01;          // Un '1' habilita la EXTI0, no la enmascara
    NVIC->ISER[0] |= (1 << 6);  // No hay espera activa del evento. Habilito la EXTI0 en el NVIC (posición 6).
                                // Si se pulsa PA0, o sea hay una interrupción
                                // externa en PA0 (EXTI0), salto al NVIC, consulto la EXTI0 en el NVIC,
                                // cambio el estado de los LEDs en el NVIC y luego sigo con el programa principal
}
```



EJEMPLO DE USO POR IRQs (3)

- El siguiente ejemplo modifica el anterior, utilizando IRQs:

```
while (1) {
    if (estado_ant != estado) // Compruebo que la EXTI0 ha cambiado el estado,
                               // Si no, no hago nada
    {
        estado_ant = estado;

        switch(estados) {
            case 0: // LED Verde ON; LED Azul OFF
                GPIOB->BSRRL = (1<<7);
                GPIOB->BSRRH = (1<<6);
                break;

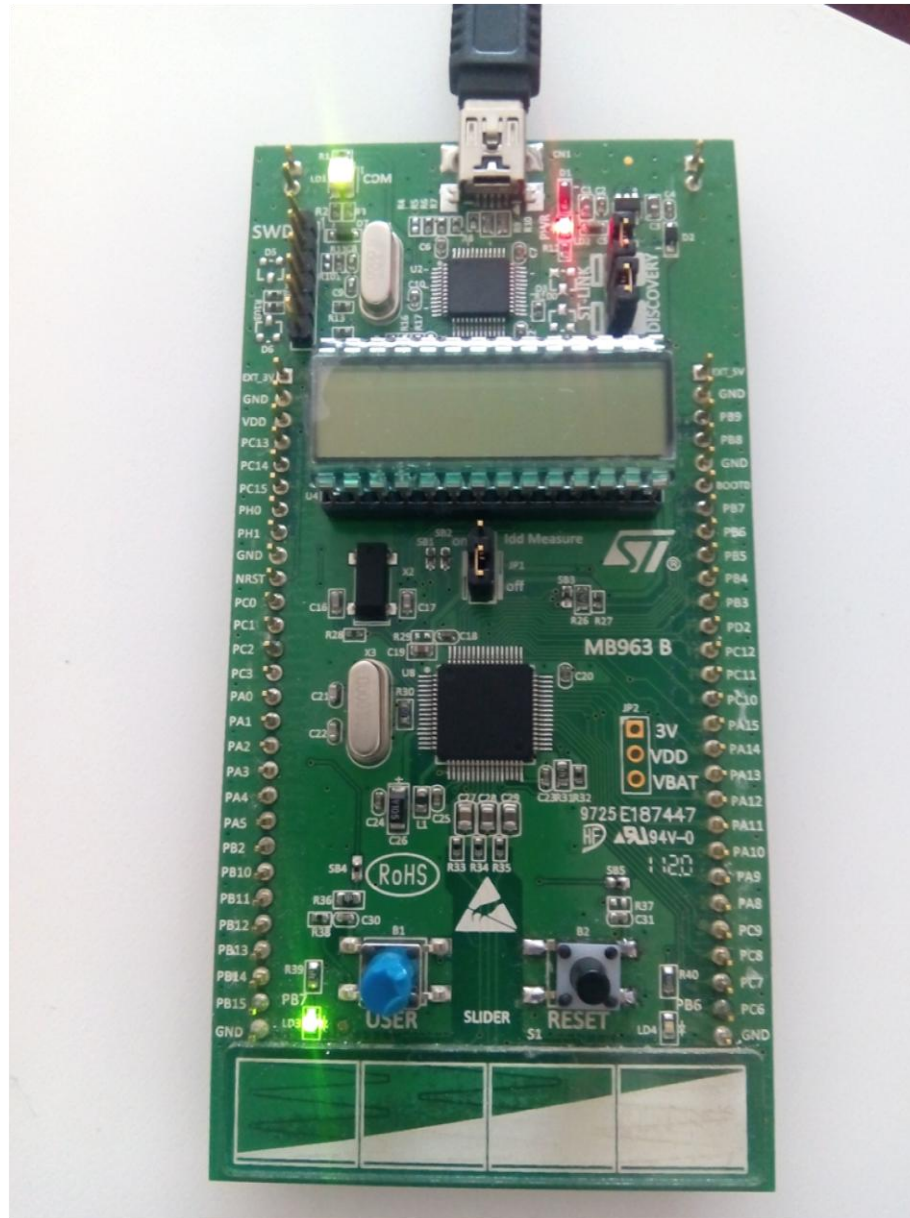
            case 1: // LED Verde ON; LED Azul ON
                GPIOB->BSRRL = (1<<7);
                GPIOB->BSRRH = (1<<6);
                break;

            case 2: // LED Verde OFF; LED Azul ON
                GPIOB->BSRRH = (1<<7);
                GPIOB->BSRRL = (1<<6);
                break;

            default: // LED Verde OFF; LED Azul OFF
                GPIOB->BSRRH = (1<<7);
                GPIOB->BSRRH = (1<<6);
                break;
        }
    }
}
```



PRUEBA DEL EJEMPLO EXPLICADO





EJERCICIOS

- 1) Análisis de los Ejemplos: Realice el diagrama de flujo de los ejemplos, cree los proyectos y al escribir el código comente lo que hace cada línea (a nivel funcional). Ejecútelo y depúrelo.
- 2) Realice un programa en el que se esté convirtiendo continuamente con el ADC, mostrando el valor por el LCD. Además, detectará por interrupciones la pulsación del botón USER, de forma que, al pulsarlo, pare el conversor y encienda el LED Azul, y si se vuelve a pulsar, se vuelva al estado inicial (ADC encendido y LED Azul apagado).
- 3) Realice un programa que, por interrupciones, convierta continuamente el ADC y vaya calculando la media de las últimas 5 medidas. La media se debe calcular en la RAI, y se mostrará el resultado en el LCD sólo si la media es 100 mV distinta a la última mostrada (tanto por arriba, como por abajo). Recuerde que mostrar cosas en el LCD no se puede hacer NUNCA en una RAI, debido al tiempo de espera que introducen las funciones del LCD.