

## Tema 7: Interrupciones (IRQs) y EXTI (External Interrupts)

### Sistemas Digitales Basados en Microprocesadores

Universidad Carlos III de Madrid

Dpto. Tecnología Electrónica

*Nota: Las figuras utilizadas para ilustrar las características y funcionalidades del microcontrolador del curso se han obtenido de la documentación técnica disponible en <https://www.st.com/en/microcontrollers-microprocessors/stm321151-152.html>*

- 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
- Ejemplos de Uso de EXTI

# Conceptos Previos

# 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 (NMI):** Aquellas que, cada vez que ocurren, provocan la atención de la CPU
  - **Enmascarables (IRQ):** 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
- Las IRQs se pueden enmascarar de forma global (por la propia CPU), o de forma local (configurando el periférico implicado en dicha IRQ). Si el enmascaramiento es local, la configuración del periférico puede enmascarar (inhabilitar) o desenmascarar (habilitar) cada una de los distintos eventos que pueden provocar la IRQ
  - 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 **Rutinas de Atención a la Interrupción (RAI)**, sino por **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 RAIs 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** debe poner una **Espera Activa** en una RAI
  - Si la RAI 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 RAI, y volver a habilitarlas antes de salir

# El Controlador de Interrupciones Vectorizadas (NVIC)

- La Gestión de las interrupciones por parte de la CPU se lleva a cabo utilizando un controlador, que en nuestro caso se llama ***Nested Vectored Interrupt Controller (NVIC)***
  - Permite 45 fuentes de interrupción enmascarables
    - ***Mucha de ellas no se van a utilizar en este curso***
  - Puede permitir establecer niveles de prioridad programables (hasta 16)
    - ***Esto no se va a usar en este curso***
  - Establece un sistema por defecto de fuentes de interrupción y su prioridad
    - En la Tabla 28 del Reference Manual están todas las fuentes.
- El Compilador (en nuestro caso el STM32CubeIDE) ya tiene unos nombres de RAI definidos para cada fuente de IRQ
  - La tabla siguiente refleja el subconjunto de fuentes de interrupción que se van a utilizar en el curso.

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 <sup>2</sup> C1 Event	I2C1_EV_IRQHandler
32	39	I2C1_ER	I <sup>2</sup> 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

- 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

# Las Entradas de Interrupción Externa (EXTI)

# 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:

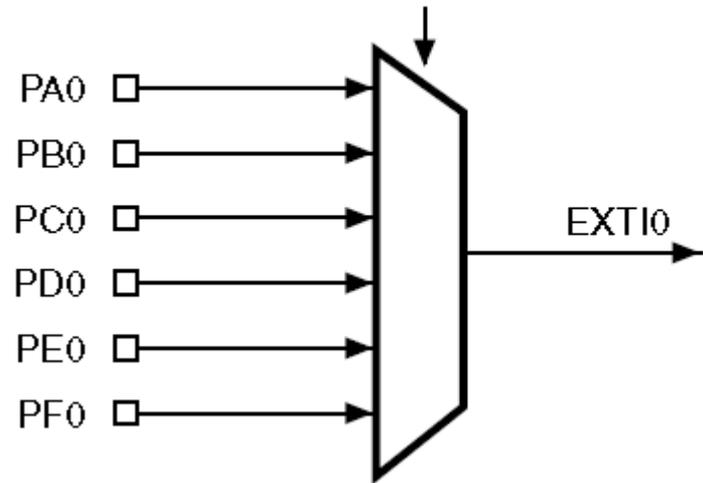
- Conjunto de registros de 16 bits (p. ej. SYSCFG → EXTICR[0]), en el que cada grupo de 4 bits indica el puerto asociado con la EXTI Line

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

- Cada registro son 4 pines

- EXTICR[0] – pin3 a pin0
- EXTICR[1] – pin7 a pin4
- EXTICR[2] – pin11 a pin8
- EXTICR[3] – pin15 a pin12

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]			
r/w	r/w	r/w	r/w												

# 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
  - Si no limpias el flag:
    - No podrás detectar nuevos eventos
    - Si estás usando RAI, entonces seguirá saltando a la RAI de forma indefinida, colgando la CPU.

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															

# Ejemplos

# Ejemplo de Uso por Eventos

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

```
/* USER CODE BEGIN 2 */

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

/* USER CODE END 2 */
```

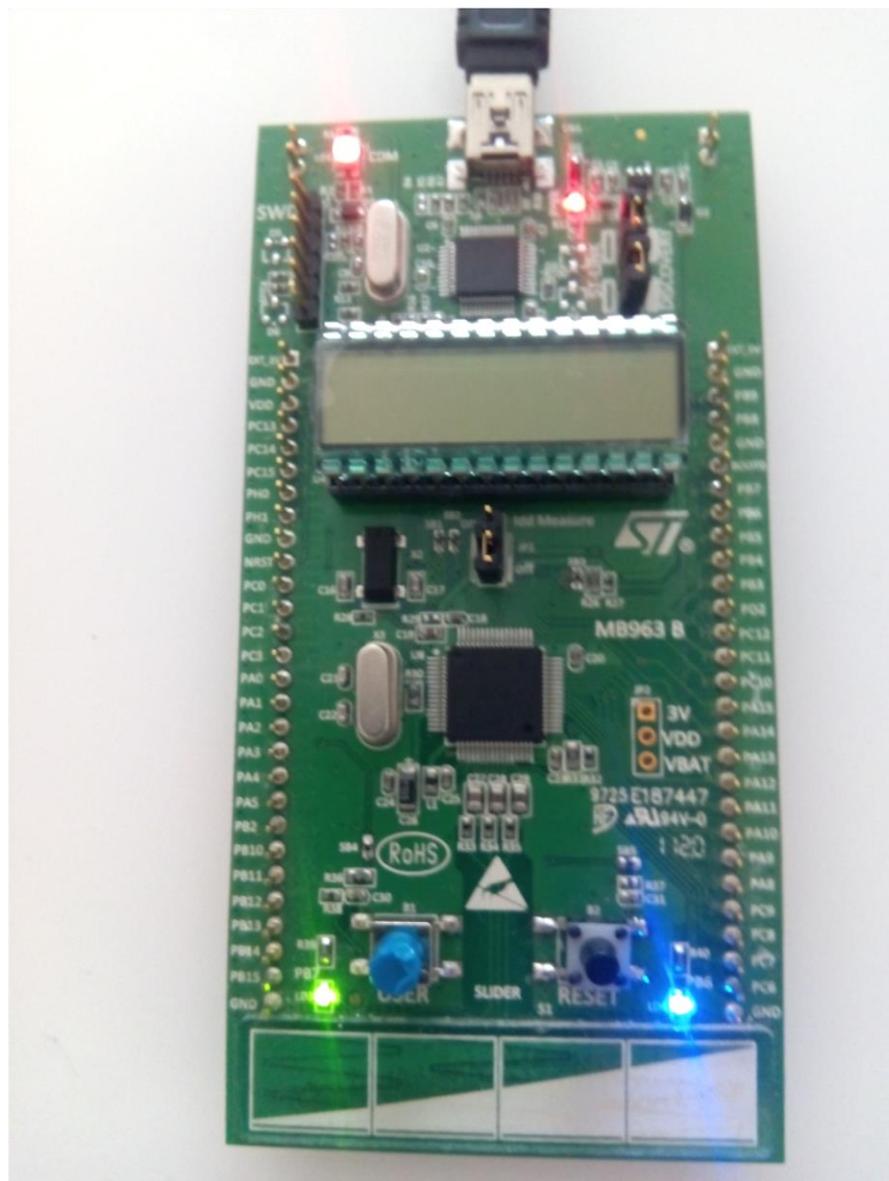
# Ejemplo de Uso por Eventos



## ○ Funcionalidad Continua:

```
/* USER CODE BEGIN WHILE */
while (1) {
    if (EXTI->PR!=0) { // Espera activa de evento. Si se pulsa PA0, o sea hay una EXTI
                        // 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->BSRR = (1<<7);
                GPIOB->BSRR = (1<<6)<<16;
                break;
        case 1: // Enciende el Led Verde y el Led Azul
                GPIOB->BSRR = (1<<7);
                GPIOB->BSRR = (1<<6);
                break;
        case 2: // Enciende el Led Azul y no el Led Verde
                GPIOB->BSRR = (1<<7)<<16;
                GPIOB->BSRR = (1<<6);
                break;
        default: // Apaga el Led Verde y el Led Azul
                GPIOB->BSRR = (1<<7)<<16;
                GPIOB->BSRR = (1<<6)<<16;
    }
}
/* USER CODE END WHILE */
}
```

# Ejemplo de Uso por Eventos



# Ejemplo de Uso por IRQs

- El siguiente ejemplo modifica el anterior, utilizando IRQs:
  - Variables Globales y RAIs:

```
/* USER CODE BEGIN PV */
unsigned char estado = 0;
unsigned char estado_ant = 0;
/* USER CODE END PV */

/* Private user code -----*/
/* USER CODE BEGIN 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
    }
}

/* USER CODE END 0 */
```

# Ejemplo de Uso por IRQs

## ○ Inicialización:

```

/* USER CODE BEGIN 2 */

// 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));
// PA0 (Boton User) como entrada (00)
GPIOA->MODER &= ~(1 << (0*2 +1));
GPIOA->MODER &= ~(1 << (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);   // Habilito la EXTI0 en el NVIC (posición 6).
                               // Si se pulsa PA0, o sea hay una EXTI
                               // en PA0 (EXTI0), salto a la RAI, consulto la EXTI0 en la RAI,
                               // cambio el estado de los LEDs en la RAI y luego sigo con el
                               // programa principal

/* USER CODE END 2 */

```

# Ejemplo de Uso por IRQs

- Funcionalidad Continua:

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

        estado_ant = estado;
        switch(estado) {
            case 0: // Enciende el Led Verde y no el Led Azul
                    GPIOB->BSRR = (1<<7);
                    GPIOB->BSRR = (1<<6)<<16;
                    break;

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

            case 2: // Enciende el Led Azul y no el Led Verde
                    GPIOB->BSRR = (1<<7)<<16;
                    GPIOB->BSRR = (1<<6);
                    break;

            default: // Apaga el Led Verde y el Led Azul
                    GPIOB->BSRR = (1<<7)<<16;
                    GPIOB->BSRR = (1<<6)<<16;

        }
    }
}
/* USER CODE END WHILE */
}
```

# Ejemplo de Uso por IRQs

