Tema 7: Interrupciones y EXTI

SOLUCIÓN DE EJERCICIOS PROPUESTOS

Ejercicio 1

El código quedaría así:

```
/* USER CODE BEGIN Includes */
#include "stm32l152c_discovery.h"
#include "stm32l152c_discovery_glass_lcd.h"
/* USER CODE END Includes */
```

```
/* USER CODE BEGIN PV */
unsigned char estado = 0;   // 0=UP; 1=DOWN
unsigned char estado_ant = 1;
/* USER CODE END PV */
```

```
/* 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 hay una interrupción externa en PA0 (EXTI0),
                              // compruebo que flanco ha sido y cambio el estado
      {
        if ((GPIOA->IDR&0x0001) == 0) estado = 0;   // flanco de bajada -> UP
        else estado = 1;                            // flanco de subida -> DOWN
        EXTI->PR = 0x01;              // Limpio el flag de EXTI0 para la siguiente vez
      }
  }
/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 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 y de subida
  EXTI->FTSR |= 0x01;         // Un '1' habilita el evento por flanco de bajada en EXTI0
  EXTI->RTSR |= 0x01;         // Un '1' habilita 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).

  BSP_LCD_GLASS_Init();
  BSP_LCD_GLASS_BarLevelConfig(0);
  BSP_LCD_GLASS_Clear();
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* 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:
          BSP_LCD_GLASS_Clear();
          BSP_LCD_GLASS_DisplayString((uint8_t *)" UP");
          break;
        default:
          BSP_LCD_GLASS_Clear();
          BSP_LCD_GLASS_DisplayString((uint8_t *)" DOWN");
          break;
      }
    }
    /* USER CODE END WHILE */
```

Ejercicio 2

La solución puede ser la dada por el siguiente código:

```
/* USER CODE BEGIN Includes */
#include "stm32l152c_discovery.h"
#include "stm32l152c_discovery_glass_lcd.h"
#include "Utiles_SDM.h"
/* USER CODE END Includes */
```

```
/* USER CODE BEGIN PV */
unsigned char estado = 0; // 0-encendido; 1-apagado
/* USER CODE END PV */
```

```
/* 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 hay una interrupción externa en PA0 (EXTI0),
                              // compruebo que flanco ha sido y cambio el estado
      {
        estado++;
        if (estado>1) estado=0;
        switch(estado){
          case 0:
            GPIOB->BSRR = (1<<6)<<16;   // Apaga LED Azul
            ADC1->CR2 |= 0x00000001;    // ADC encendido
            ADC1->CR2 |= 0x40000000;    // Arranca conversión
            break;
          case 1:
            GPIOB->BSRR = (1<<6);       // Enciende LED Azul
            ADC1->CR2 &= ~(0x00000001); // Para ADC
            break;
        }
        EXTI->PR = 0x01;               // Limpio el flag de EXTI0 para la siguiente vez
      }
  }
/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 1 */
  unsigned short valor = 0;
  uint8_t texto[7];
  /* USER CODE END 1 */
```

```
  /* 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 y de subida
  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).
  // ADC Configuration
  GPIOA->MODER |= 0x00000300;     // PA4 as analog
  ADC1->CR2 &= ~(0x00000001);     // ADON = 0 (ADC powered off)
  ADC1->CR1 = 0x00000000;         // OVRIE = 0 (overrun IRQ disabled)
                                  // RES = 00 (resolution = 12 bits)
                                  // PDI = 0 (se mantiene encendido si se para el ADC)
                                  // SCAN = 0 (scan mode disabled)
                                  // EOCIE = 0 (EOC IRQ disabled)
  ADC1->CR2 = 0x00000412;         // EOCS = 1 (EOC is activated after each conversion)
                                  // DELS = 001 (delay till data is read)
                                  // CONT = 1 (continuous conversion)
  ADC1->SQR1 = 0x00000000;        // 1 channel in the sequence
  ADC1->SQR5 = 0x00000004;        // The selected channel is AIN4
  ADC1->CR2 |= 0x00000001;        // ADON = 1 (ADC powered on)
  while ((ADC1->SR&0x0040)==0);  // If ADCONS = 0, I wait till converter is ready
  ADC1->CR2 |= 0x40000000;        // When ADCONS = 1, I start conversion (SWSTART = 1)

  BSP_LCD_GLASS_Init();
  BSP_LCD_GLASS_BarLevelConfig(0);
  BSP_LCD_GLASS_Clear();
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    valor = ADC1->DR;
    Bin2Ascii(valor,&texto[0]);
    switch(estado) {
      case 0:
        BSP_LCD_GLASS_DisplayString((uint8_t *)texto);
        break;
      default:
        BSP_LCD_GLASS_Clear();
        BSP_LCD_GLASS_DisplayString((uint8_t *)texto);
        break;
    }
    /* USER CODE END WHILE */
```

Ejercicio 3

La solución propuesta tiene las siguientes peculiaridades:

- Como la RAI es compleja y, sobre todo, como el programa principal tiene escrituras en el LCD, es preferible que las conversiones no sean tan seguidas, y por lo tanto compensa poner un retardo de 255 ciclos (DELS=111)
- En la ejecución, como sólo se actualiza el LCD atendiendo a las diferencias entre media y media_ant, no siempre se consigue el 0 y el 3000. Además, es preferible hacer la actualización de media_ant antes de el proceso costoso en tiempo que supone el escribir en el LCD.
- Para minimizar los cálculos, se trabaja en valores de ADC (no en voltios) y sólo se convierte a voltios cuando se va a mostrar por pantalla el resultado.

El código resultante para el main.c es el siguiente:

```
/* USER CODE BEGIN Includes */
#include "stm32l152c_discovery.h"
```

```
#include "stm32l152c_discovery_glass_lcd.h"
#include "Utiles_SDM.h"
/* USER CODE END Includes */
```

```
/* USER CODE BEGIN PD */
#define LIMITE  14   //  = 10 * 4096 / 3000 = (100mV en 12bits)
/* USER CODE END PD */
```

```
/* USER CODE BEGIN PV */
unsigned short valores[5] = {0,0,0,0,0};
unsigned short i=4;
unsigned int media = 0;
unsigned int media_ant = 0;
int diferencia = 0;
/* USER CODE END PV */
```

```
/* USER CODE BEGIN 0 */
void ADC1_IRQHandler(void)   // Programa de tratamiento de la IRQ del ADC.
  {
    unsigned char j=0;
    if ((ADC1->SR&0x0002)!=0)  // Si salta el evento de EOC
    {
      i++;
      if (i>4) i=0;
      valores[i] = ADC1->DR;
      media = 0;
      for (j=0; j<5; j++) media+=valores[j];
      media = media/5;
      diferencia = media - media_ant;
    }
    ADC1->SR = 0x00; // Limpia todos los flags
  }
/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 1 */
  unsigned int voltios = 0;
  uint8_t texto[7];
  /* USER CODE END 1 */
```

```
  /* USER CODE BEGIN 2 */
  BSP_LCD_GLASS_Init();
  BSP_LCD_GLASS_BarLevelConfig(0);
  BSP_LCD_GLASS_Clear();
  // ADC Configuration
  GPIOA->MODER |= 0x00000300;     // PA4 as analog
  ADC1->CR2 &= ~(0x00000001);     // ADON = 0 (ADC powered off)
  ADC1->CR1 = 0x00000020;         // OVRIE = 0 (overrun IRQ disabled)
                                  // RES = 00 (resolution = 12 bits)
                                  // SCAN = 0 (scan mode disabled)
                                  // EOCIE = 1 (EOC IRQ enabled)
  ADC1->CR2 = 0x00000472;         // EOCS = 1 (EOC is activated after each conversion)
                                  // DELS = 111 (delay till data is read)
                                  // CONT = 1 (continuous conversion)
  ADC1->SQR1 = 0x00000000;        // 1 channel in the sequence
  ADC1->SQR5 = 0x00000004;        // The selected channel is AIN4
  ADC1->CR2 |= 0x00000001;        // ADON = 1 (ADC powered on)
  while ((ADC1->SR&0x0040)==0);   // If ADCONS = 0, I wait till converter is ready
  ADC1->CR2 |= 0x40000000;        // When ADCONS = 1, I start conversion (SWSTART = 1)
  NVIC->ISER[0] |= (1 << 18);     // Habilito la IRQ del ADC1 en el NVIC (posición 18).
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if (diferencia < 0) diferencia = diferencia * (-1);
    if (diferencia > LIMITE) {
```

```
    media_ant = media;
    voltios = media * 3000 / 4095;
    Bin2Ascii((unsigned short)voltios,&texto[0]);
    BSP_LCD_GLASS_DisplayString((uint8_t *)texto);
  }
  /* USER CODE END WHILE */
```