Tema 6: Conversión Analógico/Digital y Digital/Analógico

SOLUCIÓN DE EJERCICIOS PROPUESTOS

Ejercicio 1

La sentencia a cambiar se encuentra en la sección USER CODE 2, y es la siguiente:

```
  /* USER CODE BEGIN 2 */

  ...
  ADC1->CR1 = 0x00000000;          // OVRIE = 0 (Overrun IRQ disabled)
                                   // RES = 00 (resolution = 12 bits)
                                   // SCAN = 0 (scan mode disabled)
                                   // EOCIE = 0 (EOC IRQ disabled)
  ...

  /* USER CODE END 2 */
```

Se cambiaría por este Código:

```
  /* USER CODE BEGIN 2 */

  ...
  ADC1->CR1 = 0x02000000;          // OVRIE = 0 (Overrun IRQ disabled)
                                   // RES = 10 (resolution = 8 bits)
                                   // SCAN = 0 (scan mode disabled)
                                   // EOCIE = 0 (EOC IRQ disabled)
  ...

  /* USER CODE END 2 */
```

Al ejecutar, el cambio que se nota es que los valores mostrados ya no son entre 0 y 1023, sino entre 0 y 255.

Ejercicio 2

Los cambios en el código son en las secciones siguientes y marcados en amarillo:

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

```
  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if ((GPIOA->IDR&0x00000001)!=0) { // If PA0 = 1 (USER pressed)
                                      // start conversión, otherwise finish
       while ((GPIOA->IDR&0x00000001)!=0) { // If PA0 = 1 (USER pressed),
                    // wait to avoid rebounds
          espera(70000);
```

```
        }
      // Start conversion
      while ((ADC1->SR&0x0040)==0); // While ADONS = 0, i.e., ADC is not ready
            // to convert, I wait
      ADC1->CR2 |= 0x40000000; // When ADONS = 1, I start conversion
                               // (SWSTART = 1)
      // Wait till conversion is finished
      while ((ADC1->SR&0x0002)==0);  // If EOC = 0, i.e., the conversion is not
                                     // finished, I wait
      valor = ADC1->DR;  // When EOC = 1, I take the result and store it in
                         // variable called valor
      // Calculate the volts
      voltios = (valor * 300);
      voltios = voltios / 4095;
      // Convert result to string
      Bin2Ascii((unsigned short)voltios,&texto[0]);
      // Show result in LCD
      BSP_LCD_GLASS_Clear();
      BSP_LCD_GLASS_DisplayString((uint8_t *)texto);
      }
  /* USER CODE END WHILE */
```

Como se puede ver, es necesario que para hacer el cálculo, se utilice una variable de un tipo "superior" (int en lugar de short), para que no se trunquen los resultados parciales. Al trabajar con centésimas de voltio, multiplicamos por 300. Al estar trabajando siempre con valores entero, no hace falta incluir operaciones en punto fijo o coma flotante.

Ejercicio 3

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 1 */
  unsigned char estado = 0;
  unsigned char canal_ADC = 4;
  unsigned short valorIN4 = 0;
  unsigned short valorIN5 = 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();

  // USER button configuration (PA0)
  // PA0 as digital input(00)
  GPIOA->MODER &= ~(1 << (0*2 +1));
  GPIOA->MODER &= ~(1 << (0*2));

  // ADC Configuration
  GPIOA->MODER |= 0x00000F00;     // PA4 and PA5 as analog
  ADC1->CR2 &= ~(0x00000001);     // ADON = 0 (ADC powered off)
  ADC1->CR1 = 0x00000000;         // OVRIE = 0 (overrun IRQ disabled)
                                  // RES = 00 (resolution = 12 bits)
                                  // 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 = 0x00100000;        // 2 channels in the sequence
  ADC1->SQR5 = 0x00000004;        // Selected channel is AIN4
  ADC1->SQR5 |= 5 <<5;            // Additional selected channel AIN5
  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_Clear();
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if ((GPIOA->IDR&0x00000001)!=0) { // Si PA0 = 1 (USER pressed) incrementa estado
      estado++;
      if (estado>1) estado = 0;
      while ((GPIOA->IDR&0x00000001)!=0) { // Si PA0 = 1 (USER pressed),
                                           // espera hasta soltar el boton
          }
    }

    // Wait till conversion is finished
    while ((ADC1->SR&0x0002)==0);  // If EOC = 0, i.e., the conversion is not
                                   // finished, I wait
    if (canal_ADC==4) {
      valorIN4 = ADC1->DR;  // When EOC = 1, I take the result
      canal_ADC = 5;
    }
    else {
      valorIN5 = ADC1->DR;  // When EOC = 1, I take the result
      canal_ADC = 4;
    }

    // Convert result to a string
    switch (estado) {
      case 0:
        Bin2Ascii(valorIN4,&texto[0]);
        break;
      default:
        Bin2Ascii(valorIN5,&texto[0]);
        break;
    }

    // Show result in the LCD
    BSP_LCD_GLASS_DisplayString((uint8_t *)texto);
    /* USER CODE END WHILE */
```

Ejercicio 4

Hay que cambiar 2 sentencias en el código. Por un lado, hay que cambiar los valores de wave y ponerlos en números entre 0 y 4095 (en lugar de entre 0 y 255), y la segunda es utilizar el registro DAC->DHR12R2 en lugar del DAC->DHR8R2:

```
  /* USER CODE BEGIN 2 */

  unsigned short i = 0;
  unsigned short wave[16] = {0, 32, 64, 128, 256, 512, 1024, 2048, 4095, 2048, 1024, 512, 256,
128, 64, 32};

  // DAC Configuration
  GPIOA->MODER |= 0x00000C00;       // PA5 as analog signal
  DAC->CR = 0x00010000;             // Configuration and enabling of DAC number 2

  /* USER CODE END 2 */
```

```
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    for (i=0; i<16; i++) {          // Get, one by one, the 14 values
       DAC->DHR12R2 = wave[i];
       espera(1000000);
       }
    /* USER CODE END WHILE */
```

Al ejecutarse no se aprecia ninguna diferencia (ya que se han cambiado también los valores de wave). Si no se hubieran cambiado los valores de wave, entonces la onda generada sería 16 veces más pequeña en amplitud.

La función espera(x) es necesaria, para mantener el valor analógico durante el tiempo que sea necesario, antes de cambiar al siguiente valor. Con eso se controla el periodo de la señal. A mayor x, mayor el tiempo entre muestras, y por tanto mayor periodo de la señal.

Ejercicio 5

Basándose en el código de la generación de onda, el código sería 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 1 */
  unsigned short i = 0;
  unsigned short onda[16] = {0, 2, 4, 8, 16, 32, 64, 128, 255, 128, 64, 32, 16, 8, 4, 2};
  unsigned char estado = 0;
  unsigned short multiplo = 0;  // Puls0 = 0; Puls1 = 5; Puls2 = 10; Puls3 = 16;
  /* USER CODE END 1 */
```

```
  /* USER CODE BEGIN 2 */
  BSP_LCD_GLASS_Init();
  BSP_LCD_GLASS_BarLevelConfig(0);
  BSP_LCD_GLASS_Clear();
  // USER button configuration (PA0)
  // PA0 as digital input(00)
  GPIOA->MODER &= ~(1 << (0*2 +1));
  GPIOA->MODER &= ~(1 << (0*2));

  // DAC Configuration
  GPIOA->MODER |= 0x00000C00;       // PA5 as analog signal
  DAC->CR = 0x00010000;             // Configuration and enabling of DAC number 2
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if ((GPIOA->IDR&0x00000001)!=0) { // If PA0 = 1 (USER pressed)
      estado++;
      if (estado>3) estado = 0;
      switch(estado) {
        case 0:
          multiplo = 0;
          BSP_LCD_GLASS_DisplayString((uint8_t *)" PULS0");
          break;
        case 1:
          multiplo = 5;
          BSP_LCD_GLASS_DisplayString((uint8_t *)" PULS1");
          break;
        case 2:
```

```
          multiplo = 10;
          BSP_LCD_GLASS_DisplayString((uint8_t *)" PULS2");
          break;
        default:
          multiplo = 16;
          BSP_LCD_GLASS_DisplayString((uint8_t *)" PULS3");
          break;
      }
      while ((GPIOA->IDR&0x00000001)!=0) { // USER released?
        }
    }
    for (i=0; i<16; i++) {          // Get, one by one, the 16 values (one each second)
       DAC->DHR12R2 = onda[i] * multiplo;
       espera(500000);
       }
    /* USER CODE END WHILE */
```

Ejercicio 6

Basándose en el código anterior, la solución sería la 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 1 */
  unsigned short i = 0;
  unsigned short onda[16] = {0, 2, 4, 8, 16, 32, 64, 128, 255, 128, 64, 32, 16, 8, 4, 2};
  unsigned short valor_ADC = 0;
  unsigned int valor_DAC = 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 = 0x00000000;         // OVRIE = 0 (overrun IRQ disabled)
                                  // RES = 00 (resolution = 12 bits)
                                  // 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)
  // DAC Configuration
  GPIOA->MODER |= 0x00000C00;       // PA5 as analog signal
  DAC->CR = 0x00010000;             // Configuration and enabling of DAC number 2
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
```

```
  for (i=0; i<16; i++) {          // Get, one by one, the 16 values (one each second)
    valor_ADC = ADC1->DR;
    valor_DAC = (onda[i] * valor_ADC)/ 255;
    DAC->DHR12R2 = (unsigned short)valor_DAC;
    Bin2Ascii(valor_ADC,&texto[0]);
    BSP_LCD_GLASS_DisplayString((uint8_t *)texto);

    espera(500000);
    }
  /* USER CODE END WHILE */
```