Tema 8: Temporización

SOLUCIÓN DE EJERCICIOS PROPUESTOS

Ejercicio 1

Activando el internal clock en el TIM4 (en la perspectiva CubeMX), el código quedaría así:

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

```
  /* USER CODE BEGIN 2 */
  BSP_LCD_GLASS_Init();
  BSP_LCD_GLASS_BarLevelConfig(0);
  BSP_LCD_GLASS_Clear();
  // Selección del reloj interno: CR1, CR2, SMRC
  TIM4->CR1 = 0x0000;   // ARPE = 0 -> No es PWM, es TOC
                        // CEN = 0; Contador apagado
  TIM4->CR2 = 0x0000;   // Siempre "0" en este curso
  TIM4->SMCR = 0x0000;  // Siempre "0" en este curso
  // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
  TIM4->PSC = 32000;    // Preescalado=32000 -> F_contador=32000000/32000 = 1000 pasos/segundo
  TIM4->CNT = 0;        // Inicializo el valor del contador a cero
  TIM4->ARR = 0xFFFF;   // Valor recomendado = FFFF
  TIM4->CCR1 = 2000;    // 2 segundos.
  // Selección de IRQ o no: DIER
  TIM4->DIER = 0x0000;  // No se genera INT al terminar de contar -> CCyIE = 0
  // Modo de salida del contador
  TIM4->CCMR1 = 0x0000; // CCyS = 0   (TOC)
                        // OCyM = 000 (no hay salida por el pin HW asociado al TIM4)
                        // OCyPE = 0  (sin precarga)
  TIM4->CCER = 0x0000;  // CCyP = 0   (siempre en TOC)
                        // CCyE = 0   (desactivada la salida hardware)
  // Habilitación de contador
  TIM4->CR1 |= 0x0001;   // CEN = 1 -> Arranco el contador
  TIM4->EGR |= 0x0001;   // UG = 1 -> Se genera evento de actualización
  TIM4->SR = 0;          // Limpio los flags del contador
  // 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 = 0x00000400;         // EOCS = 1 (EOC to be activated after each conversion)
                                  // DELS = 000 (no delay)
                                  // CONT = 0 (single conversion)
  ADC1->SQR1 = 0x00000000;        // 1 channel in the sequence
  ADC1->SQR5 = 0x00000004;        // Channel is AIN4
  ADC1->CR2 |= 0x00000001;        // ADON = 1 (ADC powered on)
  while ((ADC1->SR&0x0040)==0);   // While ADONS = 0, i.e., ADC is not ready to convert, I wait
```

```
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    while ((TIM4->SR&0x0002)==0); // Si no han pasado 2000 pasos = 2 segundos, espero
    TIM4->SR &= ~(0x0002);        // Si he llegado, sigo en el programa y limpio el flag
    TIM4->CCR1 += 2000;           // Aumento en 2000 pasos mas = 2 sg el valor a contar
    ADC1->CR2 |= 0x40000000;      // When ADONS = 1, I start conversion (SWSTART = 1)
    while ((ADC1->SR&0x0002)==0); // Wait till conversion is finished
    valor = ADC1->DR;  // When EOC = 1, I take the result and store it in valor
    // Convert result to string
    Bin2Ascii(valor,&texto[0]);
    BSP_LCD_GLASS_Clear();
    BSP_LCD_GLASS_DisplayString((uint8_t *)texto); // Saco el valor en el LCD
    /* 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 0 */
void TIM4_IRQHandler(void) {
  if ((TIM4->SR & 0x0002)!=0)   // Si se produce un evento en el canal 1 del TIM4 (TOC), o sea,
                                // han pasado 2000 pasos = 2 segundo, ejecuto la interrupción
    {
     TIM4->SR &= ~(0x0002);        // Si he llegado, sigo en el programa y limpio el flag
     TIM4->CCR1 += 2000;           // Aumento en 2000 pasos mas = 2 sg el valor a contar
     ADC1->CR2 |= 0x40000000;      // When ADONS = 1, I start conversion (SWSTART = 1)
     }
  TIM4->SR = 0x0000;          // Limpio todos los flags del contador (aunque sólo es el CH1)
  }
/* USER CODE END 0 */
```

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

```
  /* USER CODE BEGIN 2 */
  BSP_LCD_GLASS_Init();
  BSP_LCD_GLASS_BarLevelConfig(0);
  BSP_LCD_GLASS_Clear();
  // Selección del reloj interno: CR1, CR2, SMRC
  TIM4->CR1 = 0x0000;   // ARPE = 0 -> No es PWM, es TOC
                        // CEN = 0; Contador apagado
  TIM4->CR2 = 0x0000;   // Siempre "0" en este curso
  TIM4->SMCR = 0x0000;  // Siempre "0" en este curso
  // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
  TIM4->PSC = 32000;    // Preescalado=32000 -> F_contador=32000000/32000 = 1000 pasos/segundo
  TIM4->CNT = 0;        // Inicializo el valor del contador a cero
  TIM4->ARR = 0xFFFF;   // Valor recomendado = FFFF
  TIM4->CCR1 = 2000;    // 2 segundos.
  // Selección de IRQ o no: DIER
  TIM4->DIER = 0x0000;  // No se genera INT al terminar de contar -> CCyIE = 0
  // Modo de salida del contador
  TIM4->CCMR1 = 0x0000; // CCyS = 0   (TOC)
                        // OCyM = 000 (no hay salida por el pin HW asociado al TIM4)
```

```
                        // OCyPE = 0  (sin precarga)
  TIM4->CCER = 0x0000;  // CCyP = 0   (siempre en TOC)
                        // CCyE = 0   (desactivada la salida hardware)
  // Habilitación de contador
  TIM4->CR1 |= 0x0001;    // CEN = 1 -> Arranco el contador
  TIM4->EGR |= 0x0001;    // UG = 1 -> Se genera evento de actualización
  TIM4->SR = 0;           // Limpio los flags del contador
  // 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 = 0x00000400;         // EOCS = 1 (EOC to be activated after each conversion)
                                  // DELS = 000 (no delay)
                                  // CONT = 0 (single conversion)
  ADC1->SQR1 = 0x00000000;        // 1 channel in the sequence
  ADC1->SQR5 = 0x00000004;        // Channel is AIN4
  ADC1->CR2 |= 0x00000001;        // ADON = 1 (ADC powered on)
  while ((ADC1->SR&0x0040)==0);   // While ADONS = 0, i.e., ADC is not ready to convert, I wait
  // Selección de IRQ o no por TIM4: DIER
  TIM4->DIER = 0x0002;            // Se genera INT al terminar de contar -> CCyIE = 1
  // Habilitación de la interrupción TIM4_IRQ en el NVIC (posición 30).
  NVIC->ISER[0] |= (1 << 30);
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    while ((ADC1->SR&0x0002)==0); // Wait till conversion is finished
    valor = ADC1->DR;  // When EOC = 1, I take the result and store it in valor
    // Convert result to string
    Bin2Ascii(valor,&texto[0]);
    BSP_LCD_GLASS_Clear();
    BSP_LCD_GLASS_DisplayString((uint8_t *)texto); // Saco el valor en el LCD
    /* USER CODE END WHILE */
```

Ejercicio 3

Como el LED verde está conectado al PB7, y ese pin tiene también funcionalidad TIM4_CH2, utilizaremos la funcionalidad TOC por el canal 2, con salida hardware en modo toogle. Con eso, el bucle infinito se queda sólo para analizar si se ha pulsado el botón y cambiar el periodo del temporizador.

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

```
/* USER CODE BEGIN 0 */
void TIM4_IRQHandler(void) {
  if ((TIM4->SR & 0x0004)!=0)   // Si se produce un evento en el canal 2 del TIM4 (TOC)
    {
    switch (estado) {
      case 0:
        TIM4->CCR2 += 500;
        break;
      case 1:
        TIM4->CCR2 += 1000;
        break;
      case 2:
        TIM4->CCR2 += 2000;
        break;
      default:
        TIM4->CCR2 += 5000;
        break;
```

```
    }
    }
  TIM4->SR = 0x0000;          // Limpio los flags del contador
  }
/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 2 */
  // USER button configuration (PA0)
  // PA0 as digital input(00)
  GPIOA->MODER &= ~(1 << (0*2 +1));
  GPIOA->MODER &= ~(1 << (0*2));
  // PB7 como salida para el TIM4
  GPIOB->MODER|=0x00000001 << (2*7 +1);  // MODER = 10 (AF) para el bit 7 del puerto B
  GPIOB->MODER&=~(0x00000001 << (2*7));
  GPIOB->AFR[0]|=(0x02 << (7*4));        // AFR[0] para decir que el PB7 tiene la AF2 (TIM4)
  // Selección del reloj interno: CR1, CR2, SMRC
  TIM4->CR1 = 0x0000;  // ARPE = 0 -> No es PWM, es TOC
                       // CEN = 0; Contador apagado
  TIM4->CR2 = 0x0000;  // Siempre "0" en este curso
  TIM4->SMCR = 0x0000; // Siempre "0" en este curso
  // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
  TIM4->PSC = 32000;   // Preescalado = 32000 -> Frecuencia del contador = 32000000/32000
                       // = 1000 pasos por segundo
  TIM4->CNT = 0;       // Inicializo el valor del contador a cero
  TIM4->ARR = 0xFFFF;  // Valor recomendado = FFFF
  TIM4->CCR2 = 500;    // Empezamos con parpadeo cada 0,5s.
  // Selección de IRQ o no: DIER
  TIM4->DIER = 0x0004; // Se genera INT al terminar de contar -> CCyIE = 1
  // Modo de salida del contador
  TIM4->CCMR1 = 0x3000;  // CC2S = 0   (TOC)
                         // OC2M = 011 (no hay salida por el pin HW asociado al TIM4)
                         // OC2PE = 0  (sin precarga)
  TIM4->CCER = 0x0010;   // CC2P = 0   (siempre en TOC)
                         // CC2E = 1   (desactivada la salida hardware)
  // Habilitación de contador
  TIM4->CR1 |= 0x0001;   // CEN = 1 -> Arranco el contador
  TIM4->EGR |= 0x0001;   // UG = 1 -> Se genera evento de actualización
  TIM4->SR = 0;          // Limpio los flags del contador
  // Habilitación de la interrupción TIM4_IRQ en el NVIC (posición 30).
  NVIC->ISER[0] |= (1 << 30);
  /* 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
      estado++;
      if (estado>3) estado = 0;
      while ((GPIOA->IDR&0x00000001)!=0) { // If PA0 = 1 (USER pressed), wait
      }
    }
    /* USER CODE END WHILE */
```

Ejercicio 4

Partiendo del ejemplo dado en las transparencias, una solución podría ser 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 */
  short DC=1;
  unsigned short valor =0;
  unsigned char 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)
  // PB7 como salida para el TIM4
  GPIOB->MODER|=0x00000001 << (2*7 +1);  // MODER = 10 (AF) para el bit 7 del puerto B
  GPIOB->MODER&=~(0x00000001 << (2*7));
  GPIOB->AFR[0]|=(0x02 << (7*4));        // AFR[0] para decir que el PB7 tiene la AF2 (TIM4)
  // PA0 (Boton USER) como entrada
  GPIOA->MODER &= ~(1 << (0*2 +1));      // MODER = 00 (entrada) para el bit 0 del puerto A
  GPIOA->MODER &= ~(1 << (0*2));
  // Selección del reloj interno: CR1, CR2, SMRC
  TIM4->CR1 = 0x0080;                    // ARPE = 1 -> Es PWM; CEN = 0; Contador apagado
  TIM4->CR2 = 0x0000;                    // Siempre "0" en este curso
  TIM4->SMCR = 0x0000;                   // Siempre "0" en este curso
  // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
  TIM4->PSC = 32000;      // Preescalado=32000 -> f_contador=32000000/32000 = 1000 pasos/segundo
  TIM4->CNT = 0;          // Inicializo el valor del contador a cero
  TIM4->ARR = 9;          // Pongo una frecuencia PWM de 100 Hz y sólo cuento 10 pasos
  TIM4->CCR2 = DC;        // El Duty cycle se pone a 1 inicialmente
  // Selección de IRQ o no: DIER
  TIM4->DIER = 0x0000;   // No se genera INT al terminar de contar -> CCyIE = 0
  // Modo de salida
  TIM4->CCMR1 = 0x6800;  // CCyS = 0   (TOC, PWM)
                // OCyM = 110 (PWM con el primer semiciclo a 1)
                         // OCyPE = 1  (con precarga)
  TIM4->CCER = 0x0010;   // CCyP = 0   (siempre en PWM)
                         // CCyE = 1   (activada la salida hardware)
// Habilitación de contador
  TIM4->CR1 |= 0x0001;   // CEN = 1 -> Arranco el contador
  TIM4->EGR |= 0x0001;   // UG = 1 -> Se genera evento de actualización
  TIM4->SR = 0;          // Limpio los flags del contador
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if ((GPIOA->IDR&0x00000001)!=0) {         // Si se pulsa el botón USER
       DC+=1;                                 // Modifico el DC cada vez que pulse
       if (DC >= TIM4->ARR) DC = 1;           // Si el DC es mayor que 10, se pone a uno de nuevo
       TIM4->CCR2 = DC;                       // Guardo el nuevo DC en CCR2
       while ((GPIOA->IDR&0x00000001)!=0)     // Espero un poco para evitar los rebotes
       }
    valor = ADC1->DR; // I take the result and copy it in a variable called valor
    // Convert result to a string
    Bin2Ascii(valor,&texto[0]);
```

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

Ejercicio 5

Partiendo del ejemplo dado en las transparencias, una solución podría ser 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 PV */
unsigned short tiempo_inicio = 0;
int tiempo;
unsigned frecuencia;
/* USER CODE END PV */
```

```
/* USER CODE BEGIN 0 */
void TIM2_IRQHandler(void) {
   if ((TIM2->SR & 0x0002)!=0) {            // Si se cumple el evento del TIC
      pulsacion=1;                          // Actualizo para que lo sepa el programa principal
      tiempo = TIM2->CCR1 - tiempo_inicio;  // El tiempo transcurrido es la resta del tiempo
      if (tiempo<0) tiempo += 0x0FFFF;      // Aseguro la diferencia correcta
      tiempo_inicio = TIM2->CCR1;           // Actualizo el nuevo tiempo inicio para la próxima
      frecuencia = 1000000 / tiempo;
      }
   TIM2->SR = 0x0000;                    // Limpio los flags del contador
   }
/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 1 */
  short DC=1;
  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();
  // PB7 como salida para el TIM4
  GPIOB->MODER|=0x00000001 << (2*7 +1);  // MODER = 10 (AF) para el bit 7 del puerto B
  GPIOB->MODER&=~(0x00000001 << (2*7));
  GPIOB->AFR[0]|=(0x02 << (7*4));        // AFR[0] para decir que el PB7 tiene la AF2 (TIM4)
  // PA0 (Boton USER) como entrada
  GPIOA->MODER &= ~(1 << (0*2 +1));      // MODER = 00 (entrada) para el bit 0 del puerto A
  GPIOA->MODER &= ~(1 << (0*2));
  // Selección del reloj interno: CR1, CR2, SMRC
  TIM4->CR1 = 0x0080;                    // ARPE = 1 -> Es PWM; CEN = 0; Contador apagado
  TIM4->CR2 = 0x0000;                    // Siempre "0" en este curso
  TIM4->SMCR = 0x0000;                   // Siempre "0" en este curso
  // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
  TIM4->PSC = 32000;      // Preescalado=32000 -> f_contador=32000000/32000 = 1000 pasos/segundo
  TIM4->CNT = 0;          // Inicializo el valor del contador a cero
  TIM4->ARR = 9;          // Pongo una frecuencia PWM de 100 Hz y sólo cuento 10 pasos
  TIM4->CCR2 = DC;        // El Duty cycle se pone a 1 inicialmente
  // Selección de IRQ o no: DIER
  TIM4->DIER = 0x0000;   // No se genera INT al terminar de contar -> CCyIE = 0
  // Modo de salida
  TIM4->CCMR1 = 0x6800;  // CCyS = 0   (TOC, PWM)
               // OCyM = 110 (PWM con el primer semiciclo a 1)
                        // OCyPE = 1  (con precarga)
  TIM4->CCER = 0x0010;   // CCyP = 0   (siempre en PWM)
```

```
                        // CCyE = 1   (activada la salida hardware)
  // Habilitación de contador
  TIM4->CR1 |= 0x0001;    // CEN = 1 -> Arranco el contador
  TIM4->EGR |= 0x0001;    // UG = 1 -> Se genera evento de actualización
  TIM4->SR = 0;           // Limpio los flags del contador
  // PA5 como AF para el TIM2
  GPIOA->MODER|=0x00000001 << (2*5 +1);     // MODER = 10 (AF) para el bit 0 del puerto A
  GPIOA->MODER&=~(0x00000001 << (2*5));
  GPIOA->AFR[0]|=(0x01 << (5*4));           // AFR para decir que el P0A es AF1 (TIM2)
  GPIOA->AFR[0]&=~(0x0E << (5*4));
  // Selección del reloj interno: CR1, CR2, SMRC
  TIM2->CR1 = 0x0000;     // ARPE = 0 -> No es PWM, es TIC; CEN = 0; Contador apagado
  TIM2->CR2 = 0x0000;     // CCyIE = 0 -> No se provoca interrupción con el TIMER2
  TIM2->SMCR = 0x0000;    // Siempre "0" en este curso
  // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
  TIM2->PSC = 31;         // Preescalado=32 -> F_contador=32000000/32 = 1.000.000 pasos/seg
  TIM2->CNT = 0;          // Inicializo el valor del contador a cero
  TIM2->ARR = 0xFFFF;     // Valor recomendado si no es PWM
  // Selección de IRQ o no: DIER
  TIM2->DIER = 0x0002;    // Se genera INT al terminar de contar -> CCyIE = 1
  // Modo de salida del contador
  TIM2->CCMR1 = 0x0001;   // CCyS = 1 (TIC); OCyM = 000 y OCyPE = 0 (siempre en TIC)
  TIM2->CCER = 0x0001;    // CCyNP:CCyP = 00 (activo a flanco de subida)
                          // CCyE = 1        (captura habilitada para TIC)
  // Habilitación de contador
  TIM2->CR1 |= 0x0001;    // CEN = 1 -> Arranco el contador
  TIM2->EGR |= 0x0001;    // UG = 1 -> Se genera evento de actualización
  TIM2->SR = 0;           // Limpio los flags del contador
  NVIC->ISER[0] |= (1 << 28);  // Habilita la IRQ del TIM 2 en el NVIC (posición 28)
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if ((GPIOA->IDR&0x00000001)!=0) {         // Si se pulsa el botón USER
      DC+=1;                                 // Modifico el DC cada vez que pulse
      if (DC >= TIM4->ARR) DC = 1;           // Si el DC es mayor que 10, se pone a uno de nuevo
      TIM4->CCR2 = DC;                       // Guardo el nuevo DC en CCR2
      while ((GPIOA->IDR&0x00000001)!=0);    // Espero un poco para evitar los rebotes
      }
    Bin2Ascii((unsigned short)frecuencia,texto);            // Convierte el DC a texto
    BSP_LCD_GLASS_Clear();
    BSP_LCD_GLASS_DisplayString(texto); // Saca el DC por el LCD
    /* USER CODE END WHILE */
```

Ejercicio 6

Partiendo de la solución del ejercicio 5, ahora se va a cambiar el flanco activo para la interrupción, de forma que se calcule el DC. Los cambios más importantes se pueden ver resaltados en amarillo.

```
/* 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 short tiempo_inicio = 0;
int pulso;
int periodo;
unsigned duty_cycle;
/* USER CODE END PV */
```

```
/* USER CODE BEGIN 0 */
void TIM2_IRQHandler(void) {
  if ((TIM2->SR & 0x0002)!=0) {              // Si se cumple el evento del TIC
    if ((GPIOA->IDR&0x00000020)!=0) {        // Flanco subida: inicio de ciclo
      periodo = TIM2->CCR1 - tiempo_inicio; // El tiempo transcurrido es la resta del tiempo
      if (periodo<0) periodo += 0x0FFFF;     // Aseguro la diferencia correcta
      tiempo_inicio = TIM2->CCR1;            // Actualizo el nuevo tiempo inicio para la próxima
      duty_cycle = (pulso * 100)/periodo;
    }
    else {                                   // Flanco bajada: calculo pulso
      pulso = TIM2->CCR1 - tiempo_inicio;    // El tiempo transcurrido es la resta del tiempo
      if (pulso<0) pulso += 0x0FFFF;         // Aseguro la diferencia correcta
    }
    TIM2->SR = 0x0000;                       // Limpio los flags del contador
  }
}
/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 1 */
  short DC=1;
  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();
  // PB7 como salida para el TIM4
  GPIOB->MODER|=0x00000001 << (2*7 +1);  // MODER = 10 (AF) para el bit 7 del puerto B
  GPIOB->MODER&=~(0x00000001 << (2*7));
  GPIOB->AFR[0]|=(0x02 << (7*4));        // AFR[0] para decir que el PB7 tiene la AF2 (TIM4)
  // PA0 (Boton USER) como entrada
  GPIOA->MODER &= ~(1 << (0*2 +1));      // MODER = 00 (entrada) para el bit 0 del puerto A
  GPIOA->MODER &= ~(1 << (0*2));
  // Selección del reloj interno: CR1, CR2, SMRC
  TIM4->CR1 = 0x0080;                    // ARPE = 1 -> Es PWM; CEN = 0; Contador apagado
  TIM4->CR2 = 0x0000;                    // Siempre "0" en este curso
  TIM4->SMCR = 0x0000;                   // Siempre "0" en este curso
  // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
  TIM4->PSC = 32000;      // Preescalado=32000 -> f_contador=32000000/32000 = 1000 pasos/segundo
  TIM4->CNT = 0;          // Inicializo el valor del contador a cero
  TIM4->ARR = 9;          // Pongo una frecuencia PWM de 100 Hz y sólo cuento 10 pasos
  TIM4->CCR2 = DC;        // El Duty cycle se pone a 1 inicialmente
  // Selección de IRQ o no: DIER
  TIM4->DIER = 0x0000;   // No se genera INT al terminar de contar -> CCyIE = 0
  // Modo de salida
  TIM4->CCMR1 = 0x6800;  // CCyS = 0   (TOC, PWM)
                         // OCyM = 110 (PWM con el primer semiciclo a 1)
                         // OCyPE = 1  (con precarga)
  TIM4->CCER = 0x0010;   // CCyP = 0   (siempre en PWM)
                         // CCyE = 1   (activada la salida hardware)
  // Habilitación de contador
  TIM4->CR1 |= 0x0001;   // CEN = 1 -> Arranco el contador
  TIM4->EGR |= 0x0001;   // UG = 1 -> Se genera evento de actualización
  TIM4->SR = 0;          // Limpio los flags del contador
  // PA5 como AF para el TIM2
  GPIOA->MODER|=0x00000001 << (2*5 +1);    // MODER = 10 (AF) para el bit 0 del puerto A
  GPIOA->MODER&=~(0x00000001 << (2*5));
  GPIOA->AFR[0]|=(0x01 << (5*4));          // AFR para decir que el P0A es AF1 (TIM2)
  GPIOA->AFR[0]&=~(0x0E << (5*4));
  // Selección del reloj interno: CR1, CR2, SMRC
  TIM2->CR1 = 0x0000;     // ARPE = 0 -> No es PWM, es TIC; CEN = 0; Contador apagado
  TIM2->CR2 = 0x0000;     // CCyIE = 0 -> No se provoca interrupción con el TIMER2
  TIM2->SMCR = 0x0000;    // Siempre "0" en este curso
  // Configuración del funcionamiento del contador: PSC, CNT, ARR y CCRx
  TIM2->PSC = 31;         // Preescalado=32 -> F_contador=32000000/32 = 1.000.000 pasos/seg
  TIM2->CNT = 0;          // Inicializo el valor del contador a cero
  TIM2->ARR = 0xFFFF;     // Valor recomendado si no es PWM
```

```
  // Selección de IRQ o no: DIER
  TIM2->DIER = 0x0002;    // Se genera INT al terminar de contar -> CCyIE = 1
  // Modo de salida del contador
  TIM2->CCMR1 = 0x0001;   // CCyS = 1 (TIC); OCyM = 000 y OCyPE = 0 (siempre en TIC)
  TIM2->CCER = 0x000B;    // CCyNP:CCyP = 11 (activo a flanco de subida)
                          // CCyE = 1        (captura habilitada para TIC)
  // Habilitación de contador
  TIM2->CR1 |= 0x0001;    // CEN = 1 -> Arranco el contador
  TIM2->EGR |= 0x0001;    // UG = 1 -> Se genera evento de actualización
  TIM2->SR = 0;           // Limpio los flags del contador
  NVIC->ISER[0] |= (1 << 28);  // Habilita la IRQ del TIM 2 en el NVIC (posición 28)
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if ((GPIOA->IDR&0x00000001)!=0) {     // Si se pulsa el botón USER
      DC+=1;                              // Modifico el DC cada vez que pulse
      if (DC >= TIM4->ARR) DC = 1;        // Si el DC es mayor que 10, se pone a uno de nuevo
      TIM4->CCR2 = DC;                    // Guardo el nuevo DC en CCR2
      while ((GPIOA->IDR&0x00000001)!=0); // Espero un poco para evitar los rebotes
      }
    Bin2Ascii((unsigned short)duty_cycle,texto); // Convierte el DC a texto
    BSP_LCD_GLASS_DisplayString(texto);   // Saca el DC por el LCD
    /* USER CODE END WHILE */
```