Universidad Politécnica de Madrid

CONTROL DE LUCES

David Gay Alonso 52375

Raúl Güemes Sánchez 52397

A – 404

Sistemas Eléctronicos Digitales

Contenido

OBJETIVO

El objetivo del este trabajo es conseguir controlar la iluminación de tres leds en función de la luz ambiental (LDR).

INTRODUCCIÓN

En nuestro proyecto empleamos 3 leds en serie, con sus respectivas resistencias, y un LDR. Los conectaremos a los puertos de la placa discovery STM32F411 y mediante la programación de un temporizador, podremos hacer que los leds se enciendan durante un determinado tiempo, y mediante la lectura del LDR se activan los leds de manera independiente.

MATERIAL DE TRABAJO

Vamos a utilizar en este proyecto:

- 4 resistencias de 220 Ω.
- 3 leds.
- 1 LDR.
- Cables macho/hembra.
- Protoboard.
- Microprocesador STM32F411VEx.
- Programa MDK-ARMv5.

FUNCIONAMIENTO

Nuestro proyecto de arduino consiste en un control de luces en función de la luz ambiente, como ya hemos mencionado anteriormente.

Para poder medir la luminosidad empleamos un LDR (conectado al PA1), el cual nos da unos valores de lectura, a través de un ADC.

De esta manera, en el código indicamos que, si el valor del umbral está por debajo de un valor arbitrariamente escogido, entonces se encenderán los 3 leds. Es decir, cuando

hay poca luz ambiente se encienden los leds y cuando se dispone de plena luminosidad, los leds se mantendrán apagados.

Además de este funcionamiento implementamos una interrupción mediante un pulsador, cuando yo presiono el pulsador, los leds pasan a encenderse en unos determinados intervalos de tiempo. Para estos intervalos de tiempo hemos empleado el reloj síncrono de la placa.

CÓDIGO

Para llevar a cabo la implementación de la interrupción, declaro una variable global llamada:

```
52  /* USER CODE BEGIN PV */
53  /* Private variables ----
54  volatile int flag;
55  /* USER CODE END PV */
56
```

Y empleamos la siguiente función:

```
64  /* Private function prototypes ----------------------------
65  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
66  {
67    flag = 1;
68  }
```

Esta variable flag, nos servirá para la interrupción generada cuando se presiona el pulsador.

A continuación, para poder leer el valor del ADC, proporcionado por el LDR, debo declarar la siguiente variable:

```
/* USER CODE BEGIN 0 */
  uint8_t ADC_val;
/* USER CODE END 0 */
```

En la siguiente parte del código se muestra como según los valores del ADC, encendemos o apagamos los leds. Siempre y cuando no se produzca ninguna interrupción.

```
while (1)
{
/* USER CODE END WHILE */
  while(!flag)
  {
    HAL_ADC_Start(&hadc1);
    if (HAL_ADC_PollForConversion(&hadc1,5)==HAL_OK)
    {
      ADC_val=HAL_ADC_GetValue(&hadc1);
    }
    if (ADC_val < 10)
    {
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_SET);
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_6, GPIO_PIN_SET);
    }
    else
    {
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_RESET);
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_6, GPIO_PIN_RESET);
    }
    HAL_ADC_Stop(&hadc1);
    HAL_Delay(300);
  }
```

Cuando presiono el pulsador se genera la interrupción, y de esta manera, los leds se encenderán y apagarán durante unos intervalos de tiempo definidos.

```
136     while(flag)
137     {
138         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_SET);
139         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_RESET);
140         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_6, GPIO_PIN_RESET);
141         HAL_Delay(5000);
142         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_2, GPIO_PIN_RESET);
143         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_4, GPIO_PIN_SET);
144         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_6, GPIO_PIN_SET);
145         HAL_Delay(5000);
146
147     }
148
149     HAL_Delay(300);
150 }
```

Por otra parte, nuestra intención era implementar un reloj asíncrono, de tal manera que generase una interrupción con un valor determinado de frecuencia. Así, de esta manera, podríamos encender y apagar un led de manera periódica, indicando que la placa estaba funcionando correctamente.

Para implementar este reloj asíncrono introduzco el siguiente código:

```
57  /* Private function prototypes -----------------
58  void SystemClock_Config(void);
59  static void MX_GPIO_Init(void);
60  static void MX_ADC1_Init(void);
61  static void MX_TIM10_Init(void);
62
```

Aunque en el reloj global venga establecida por defecto una frecuencia predeterminada, a los periféricos les llega un valor distinto de frecuencia, debido a un prescaler que le divide ese dato entre un valor arbitrario. Por lo tanto, aunque nuestro valor predefinido de frecuencia sea de 100 MHz. A nuestro reloj le llegan 16 MHz.

Esto valor no es la frecuencia con la que va a contar.

Para definir este valor, a nuestro reloj le vamos a definir un prescaler y un número de carga, siendo el número de carga el valor de los pulsos necesarios para que nuestro temporizador genera la interrupción.

Lo anterior se logra con la siguiente ecuación:

$$Fre.\ T = \frac{Fre.APB1}{Prescaler}$$

$$Carga = \big((Fre.T) * Tiempo\big) - 1$$

Y en el código se veria reflejado de la siguiente manera:

```
250  /* TIM10 init function */
251  static void MX_TIM10_Init(void)
252  {
253
254    htim10.Instance = TIM10;
255    htim10.Init.Prescaler = 50000;
256    htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
257    htim10.Init.Period = 999;
258    htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
259    if (HAL_TIM_Base_Init(&htim10) != HAL_OK)
260    {
261      _Error_Handler(__FILE__, __LINE__);
262    }
263
264  }
```

A continuación, debemos decirle a nuestro temporizador que queremos que haga cada vez que realice la espera. Esto se hacen en el fichero “stm32f4xx_it.c”:

```
212 void TIM1_UP_TIM10_IRQHandler(void)
213 {
214   /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 0 */
215   HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_0);
216
217   /* USER CODE END TIM1_UP_TIM10_IRQn 0 */
218   HAL_TIM_IRQHandler(&htim10);
219   /* USER CODE BEGIN TIM1_UP_TIM10_IRQn 1 */
220
221   /* USER CODE END TIM1_UP_TIM10_IRQn 1 */
222 }
223
224 /* USER CODE BEGIN 1 */
225
226 /* USER CODE END 1 */
227 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Y, por último, le debemos decir a nuestro temporizador cuando debe iniciar la espera, esto se realiza en el “main.c” dentro del main:

```
105   /* USER CODE BEGIN 2 */
106 HAL_TIM_Base_Start_IT(&htim10);
107   /* USER CODE END 2 */
108
109   /* Infinite loop */
110   /* USER CODE BEGIN WHILE */
```

(Debido a razones desconocidas no ha sido posible la implementación del temporizador).