Trabajo de Microprocesadores

Aplicación domótica para control de luces de una vivienda

GRUPO 23

SISTEMAS ELECTRÓNICOS DIGITALES

CURSO 2022/2023

Alumnos:	**Nº de Matrícula:**
Aspiroz de la Calle, Miguel	54918
Bustillo Ergui, Jaime	54920
Pacheco Guiop, Jefferson Eduardo	55581

ÍNDICE

1. INTRODUCCIÓN
2. DIAGRAMAS
3. MAQUETA
4. CÓDIGO
    a. TRABAJO_MICROS.ioc
    b. main.c
    c. stm32f4xx_it.c

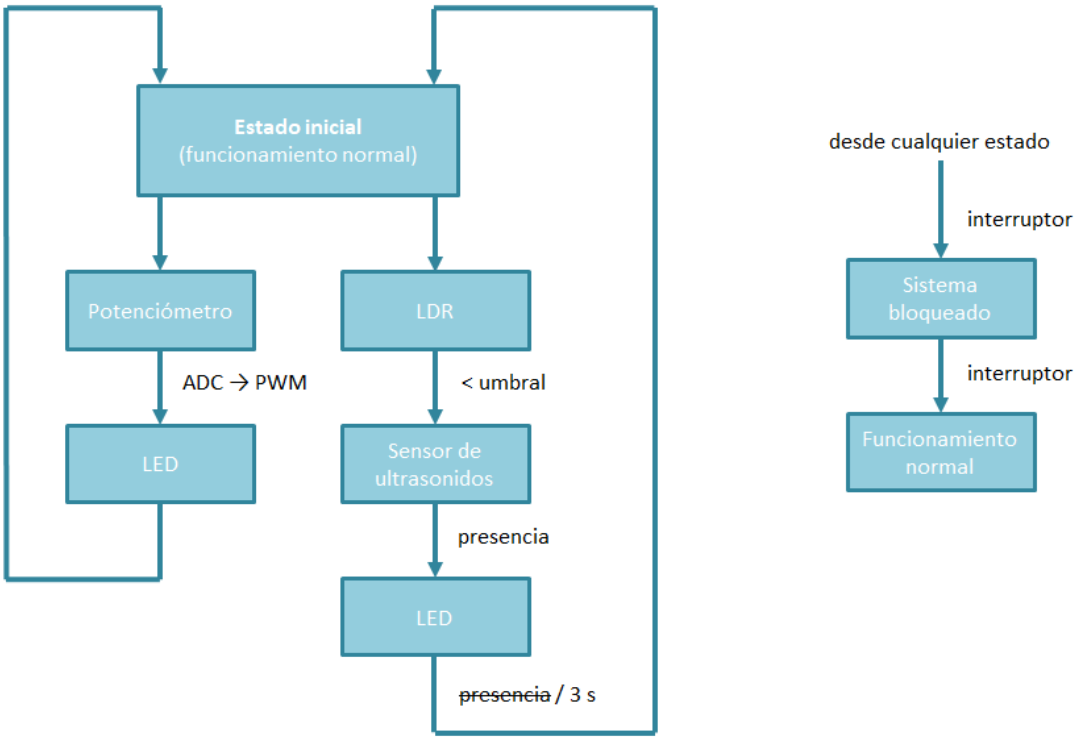
1. INTRODUCCIÓN

Para el trabajo de programación de microprocesadores de la asignatura de Sistemas Electrónicos Digitales se ha optado por llevar a cabo la primera propuesta ofertada por los profesores.

Propuesta:
Aplicación de domótica: se pretende simular el control de luces de una vivienda con distintos focos luminosos, activados por diferentes métodos.
El proyecto se puede imaginar como una estancia con varios puntos de iluminación. Uno de ellos, que servirá a modo de luz ambiente, se activará bajo los condicionantes de la detección de presencia y, como medida de ahorro energético, la medición de una baja intensidad luminosa. Otro está pensado como una luz focalizada, comparable a un flexo, que podrá ser regulada manualmente según las necesidades específicas del usuario. Por último, habrá una luz que indicará que, de forma manual, se habrá interrumpido y bloqueado el funcionamiento del sistema de control.

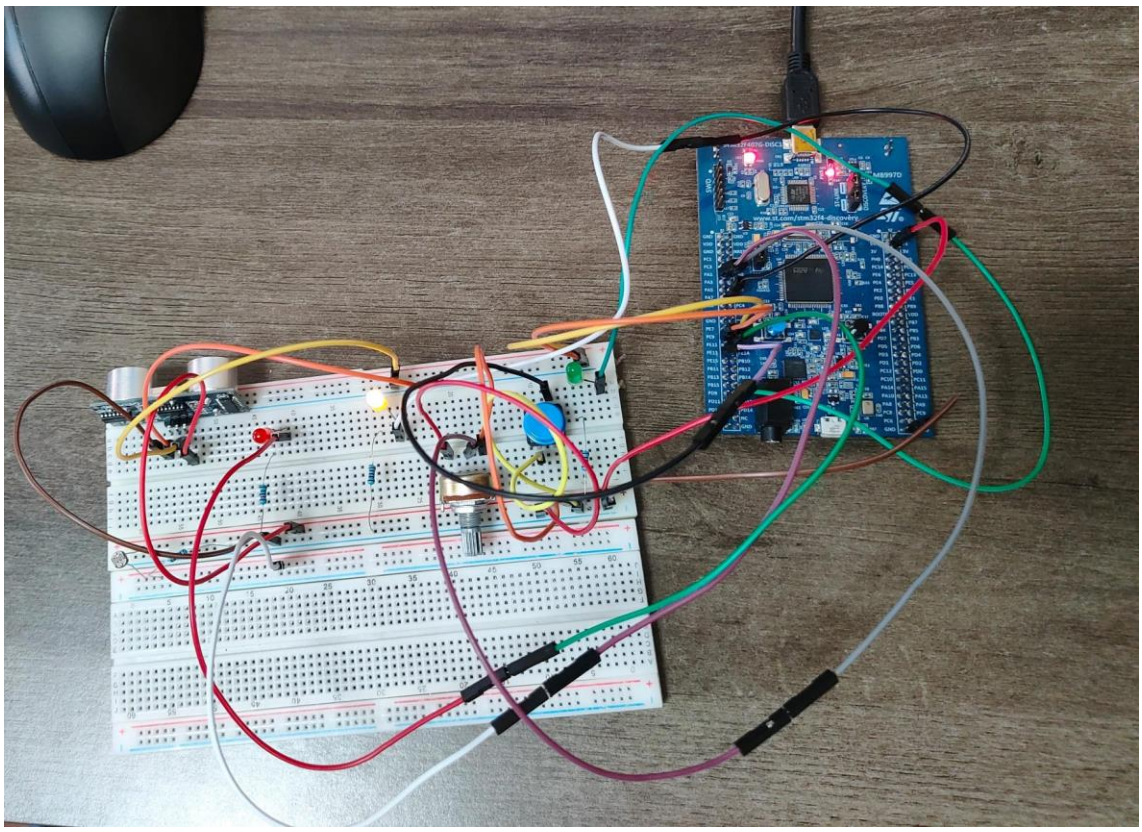
2. DIAGRAMAS

En la imagen inferior se presenta un diagrama de bloques que muestra el funcionamiento básico del programa.



3. MAQUETA

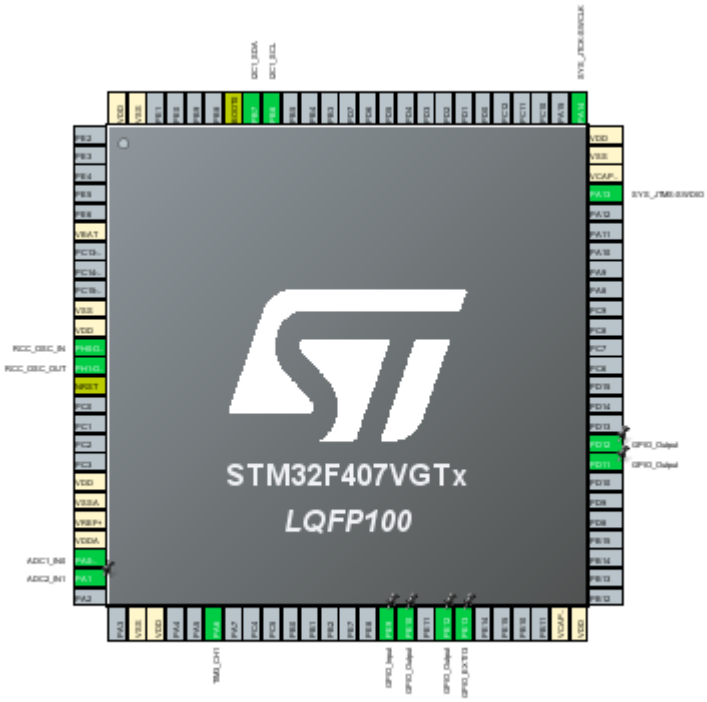
El montaje consiste en una placa STM32F407 a la que se conectan una serie de componentes con la ayuda de una *protoboard*. El sensor de ultrasonidos, utilizado para la detección de presencia, está conectado a un pin de entrada digital (PE9) y otro de salida (PE10). La medida de la intensidad luminosa se hace con un LDR, en serie con una resistencia de 1 kilohmio, y que va a un pin de entrada analógica (PA1). El LED cuyo estado depende de estos dos sensores se conecta a una salida digital (PE12). La luz focalizada se regula con un potenciómetro, el cual llega hasta otra entrada analógica (PA0). El LED asociado a él se alimenta con un pin analógico (PA6). El bloqueo del sistema de control se activa mediante un pulsador, usando otro pin (PE13) para gestionar las interrupciones. El LED que indica que se ha bloqueado el sistema se conecta a un pin de salida digital (PD12). Cabe destacar que los tres LED llevan en serie una resistencia de 220 ohmios y que todos los demás componentes tienen conexión a 5 V y a tierra.



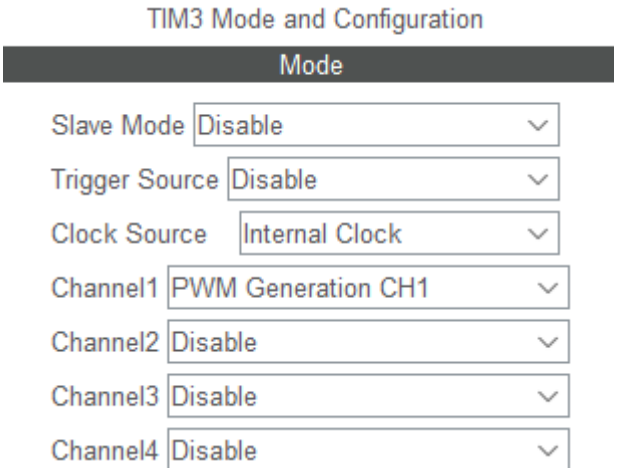
4. CÓDIGO

a. TRABAJO_MICROS.ioc

Se parte de un código generado de forma automática a partir de la configuración de los pines. El *pinout* utilizado para el proyecto se muestra en la imagen inferior.

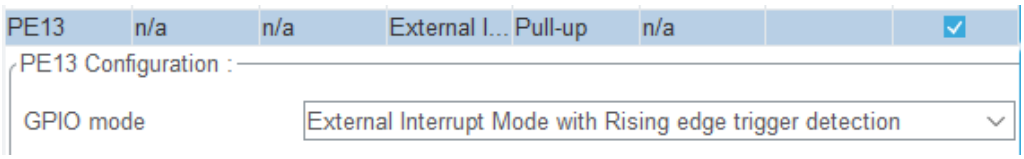


Los pines analógicos PA0 y PA1 se usan como entradas de los CAD, es decir, el potenciómetro y el LDR. El PA6 sirve para controlar el LED asociado al potenciómetro mediante el temporizador 3, por lo que se debe configurar para enviar pulsos PWM por el canal 1 del TIM3:



El sensor de ultrasonidos HC-SR04 requiere una conexión de entrada GPIO, en este caso se ha elegido el pin PE9, y otra de salida, el pin PE10. El LED

cuyo funcionamiento depende del LDR y el sensor de ultrasonidos se conecta al pin de salida GPIO PE12. El pulsador que generara la interrupción llega al pin PE13, por lo que este se debe configurar con el modo *GPIO External Interrupt*:



El LED asociado al pulsador de la interrupción se conecta al pin PD12, configurado como salida GPIO.

b. main.c

Tras incluir los ficheros de cabecera necesarios (en este caso solo el *main.h*), se comienza declarando las funciones, *#define* y variables globales que se van a utilizar. Las dos primeras líneas corresponden a funciones relacionadas con el uso del sensor de ultrasonidos como detector de presencia y se explicarán en detalle más adelante.

```
72 void HC_SR04_Trigger(void);
73 float HC_SR04_GetDistance(void);
```

A continuación, se usan varios *#define* a modo de alias para referirse a los pines del LED y del sensor de ultrasonidos y para establecer los umbrales de distancia y luminosidad que posteriormente se utilizarán.

```
76 #define TRIGGER_PIN GPIO_PIN_10
77 #define TRIGGER_PORT GPIOE
78 #define ECHO_PIN GPIO_PIN_9
79 #define ECHO_PORT GPIOE
80 #define OUTPUT_PIN GPIO_PIN_12
81 #define OUTPUT_PORT GPIOE
82 #define DISTANCE_THRESHOLD 2
83 #define BRIGHTNESS_THRESHOLD 50
```

Luego, se definen una serie de variables globales. Las dos primeras se usan para medir y limitar el tiempo de encendido del LED, las tres siguientes almacenan los valores de los conversores analógico-digital y la conversión de la primera de estas al rango de PWM, y la última variable es simplemente un *flag* para saber si la interrupción está activa o no.

```
85 uint32_t distance_timer = 0;
86 uint32_t LED_timeout = 3000;  // Timeout de 3 segundos
87
88 uint32_t adcValue = 0; // Variable para almacenar el valor del ADC1 (potenciómetro)
89 uint32_t ldrValue = 0; // Variable para almacenar el valor del ADC2 (LDR)
90 uint32_t pwmValue = 0;
91
92 uint8_t interruption_active = 0; // Verifica el estado del pulsador para la interrupción
```

Ya dentro de la función *main*, después de la inicialización de todos los periféricos configurados, se inicializan también los CAD y la generación de la señal PWM en el canal 1 del temporizador 3.

```
145     HAL_ADC_Start(&hadc1);
146     HAL_ADC_Start(&hadc2); //inicialización
147     HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
```

Ahora, comienza el bucle infinito que contiene el comportamiento básico del programa. Lo primero de todo es asegurar que la interrupción no está activa, ya que en caso contrario el sistema de control se bloquearía.

```
153   while (1)
154   {
155       if (interruption_active == 0) //Se verifica si la interrupcion está activa antes de ejecutar el resto del código
156       {
```

El valor ajustado en el potenciómetro se obtiene del primer CAD y se almacena en la variable *adcValue*, para ser posteriormente convertida al rango del PWM y guardar el nuevo valor en *pwmValue* para controlar el ciclo de trabajo de la señal PWM y regular así el brillo del LED asociado.

```
158         // Ajuste con ADC1 (potenciómetro)
159         if (HAL_ADC_PollForConversion(&hadc1, 100) == HAL_OK) // Espera a que la medida esté lista
160         {
161          adcValue = HAL_ADC_GetValue(&hadc1);
162 
163           // Ajustamos el valor del ADC (0-4095) al rango del PWM (0-720)
164           pwmValue = (adcValue * 720) / 4095;
165           __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, pwmValue);
166             HAL_ADC_Start(&hadc1); // Reiniciar la conversión del ADC
167         }
```

El siguiente fragmento de código establece las condiciones de encendido y apagado del LED que se considera como la luz ambiente de la estancia. Se empieza obteniendo el valor del segundo CAD, que se corresponde con la intensidad luminosa medida por el LDR. En caso de que no se llegue al umbral mínimo establecido, se obtiene la distancia con el sensor de ultrasonidos gracias a la función *HC_SR04_GetDistance()*. Si hay presencia en el rango de detección, el LED se mantendrá siempre encendido y se apagará solo en caso de que desaparezca dicha presencia y no se detecte de nuevo durante los 3 segundos fijados en *LED_timeout*. Nótese que al desaparecer la presencia se comienza a medir el tiempo con *HAL_GetTick()* y este se almacena en *distance_timer*. Por último, si la medida hecha anteriormente por el LDR supera el umbral se mantendrá apagado el LED.

```
169         // Medida de la luminosidad con ADC2 (LDR)
170         if (HAL_ADC_PollForConversion(&hadc2, 100) == HAL_OK) // Espera a que la medida esté lista
171         {
172             ldrValue = HAL_ADC_GetValue(&hadc2); // Obtención del valor
173 
174             if (ldrValue < BRIGHTNESS_THRESHOLD) // Comparación con el umbral mínimo de luminosidad
175             {
176                 float distance = HC_SR04_GetDistance();
177 
178                 if (distance > 0 && distance < DISTANCE_THRESHOLD) // Detección de presencia
179                 {
180                     if (HAL_GetTick() - distance_timer >= LED_timeout) // Encender el LED durante un tiempo determinado
181                     {
182                         HAL_GPIO_WritePin(OUTPUT_PORT, OUTPUT_PIN, GPIO_PIN_RESET);
183                         //Display_Alert();
184                         //lcd_clear();
185                         //lcd_send_string ("ALERTA DE INTRUSO");
186                     }
187                     else
188                     {
189                         HAL_GPIO_WritePin(OUTPUT_PORT, OUTPUT_PIN, GPIO_PIN_SET);
190                         //Display_SafeZone();
191                         //lcd_clear();
192                         //lcd_send_string ("ZONA PROTEGIDA");
193                     }
194                 }
195                 else
196                 {
197                     distance_timer = HAL_GetTick();
198                     HAL_GPIO_WritePin(OUTPUT_PORT, OUTPUT_PIN, GPIO_PIN_SET);
199                     //Display_SafeZone();
200                 }
```

```
            }
            else
            {
                HAL_GPIO_WritePin(OUTPUT_PORT, OUTPUT_PIN, GPIO_PIN_RESET);
            }
            HAL_ADC_Start(&hadc2); // Reiniciar la conversión del ADC
        }
    }

    HAL_Delay(10);
```

La configuración del reloj del sistema, conversores analógico-digital, temporizador y pines de entrada/salida no merece la pena ser comentada exhaustivamente, puesto que está generada de forma automática. Quizás lo único que quepa destacar sea el hecho de haber elegido el modo de canal PWM para el temporizador TIM3.

```
  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
  {
    Error_Handler();
  }
```

Por último, se van a ver las funciones utilizadas para el correcto funcionamiento del sensor de ultrasonidos HC-SR04 utilizado como detector de presencia.
La primera es la encargada de enviar un pulso de activación al sensor, lo cual desencadena la emisión de ultrasonidos para medir distancia. Esta función consiste simplemente en poner el pin correspondiente a nivel alto durante un breve pulso.

```
void HC_SR04_Trigger(void)
{
  HAL_GPIO_WritePin(TRIGGER_PORT, TRIGGER_PIN, GPIO_PIN_SET);
  HAL_Delay(1);
  HAL_GPIO_WritePin(TRIGGER_PORT, TRIGGER_PIN, GPIO_PIN_RESET);
}
```

Por otro lado, la segunda función se ocupa de la propia medición del sensor y del cálculo para la conversión a distancia. Tras declarar las variables locales necesarias, se llama a la función anterior para enviar el pulso de activación y, una vez empezada la emisión de ultrasonidos, se almacena el tiempo en *start_time*. Cuando se dejan de recibir los ultrasonidos reflejados, se mide el tiempo con *HAL_GetTick()* y se guarda en *end_time*. La variable *pulse_duration* se calcula como la diferencia entre los tiempos medidos y la distancia recorrida por los ultrasonidos se obtiene como la multiplicación de dicha variable por la velocidad del sonido (en cm/µs) y dividido entre 2 para considerar el trayecto de ida y vuelta. Este resultado, de tipo *float*, es el valor de retorno de la función.

```c
float HC_SR04_GetDistance(void)
{
  uint32_t start_time = 0;
  uint32_t end_time = 0;
  float distance = 0;

  /* Envío de pulso de activación */
  HC_SR04_Trigger();

  /* Espera a que comience el pulso de eco */
  while (HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN) == GPIO_PIN_RESET)
  {
  }

  /* Medida del tiempo en el que se inicia el envío */
  start_time = HAL_GetTick();

  /* Espera al fin del pulso de eco */
  while (HAL_GPIO_ReadPin(ECHO_PORT, ECHO_PIN) == GPIO_PIN_SET)
  {
  }

  /* Medida del tiempo en el que finaliza */
  end_time = HAL_GetTick();

  /* Cálculo de la duración del pulso */
  uint32_t pulse_duration = end_time - start_time;

  /* Cálculo de la distancia según la duración del pulso de eco y la velocidad del sonido */
  distance = (pulse_duration * 0.0343) / 2; // Velocidad del sonido (cm/us) / 2 (ida y vuelta)

  return distance;
}
```

c. stm32f4xx_it.c

Este archivo contiene la función necesaria para gestionar las interrupciones, la cual llama al *callback* que se ha implementado. El parámetro de la función es un pin GPIO y, en caso de que se detecte que se ha activado el PE13, o sea, que se ha apretado el pulsador, se evalúa el *flag* de la interrupción. Si está a nivel bajo, se pone a 1 y se enciende el LED conectado al pin PD12 para indicar que el sistema de control está bloqueado. Para desbloquearlo se necesitaría un nuevo flanco de subida en el pin PE13, lo que implicaría que el valor de *interruption_active* pasaría a 0 y el LED correspondiente se apagaría.

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == GPIO_PIN_13) // Verificar si la interrupción proviene del pin PE13
  {
    if (interruption_active == 0)
    {
      interruption_active = 1;
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET); // Encender el LED adicional en PD12
    }
    else
    {
      interruption_active = 0;
      HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET); // Apagar el LED adicional en PD12
    }
  }
}
```