
# SISTEMAS ELECTRÓNICOS DIGITALES

# CURSO 2018/19

# TRABAJO DE MICROCONTROLADOR

## Título: Control de luces

**Grupo:** Grupo B, Lunes (11:30 – 13:30)

**Integrantes:**

Ricardo Andrés Baños Rojas (52240)

Pablo Sanz Hernández (52625)

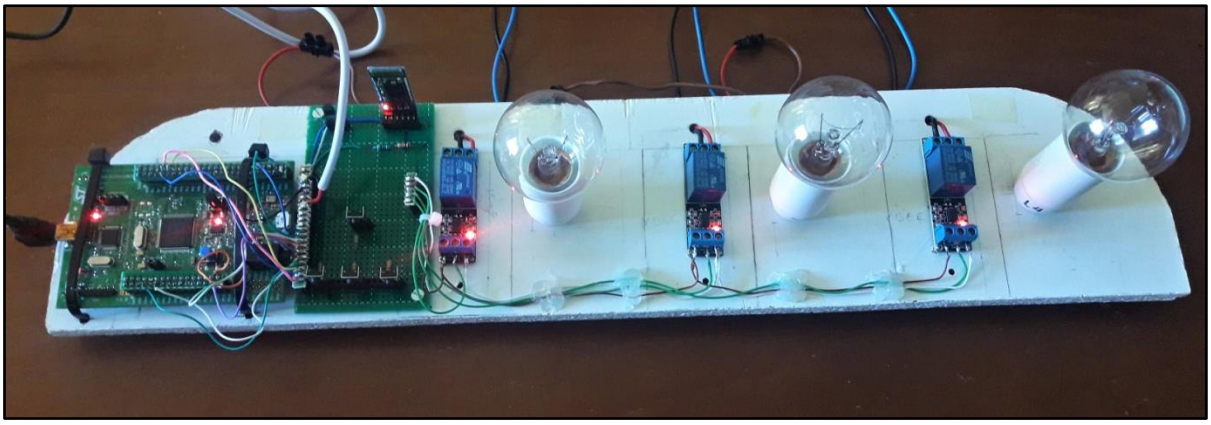
**Tutor:** Alberto Brunete Gonzalez

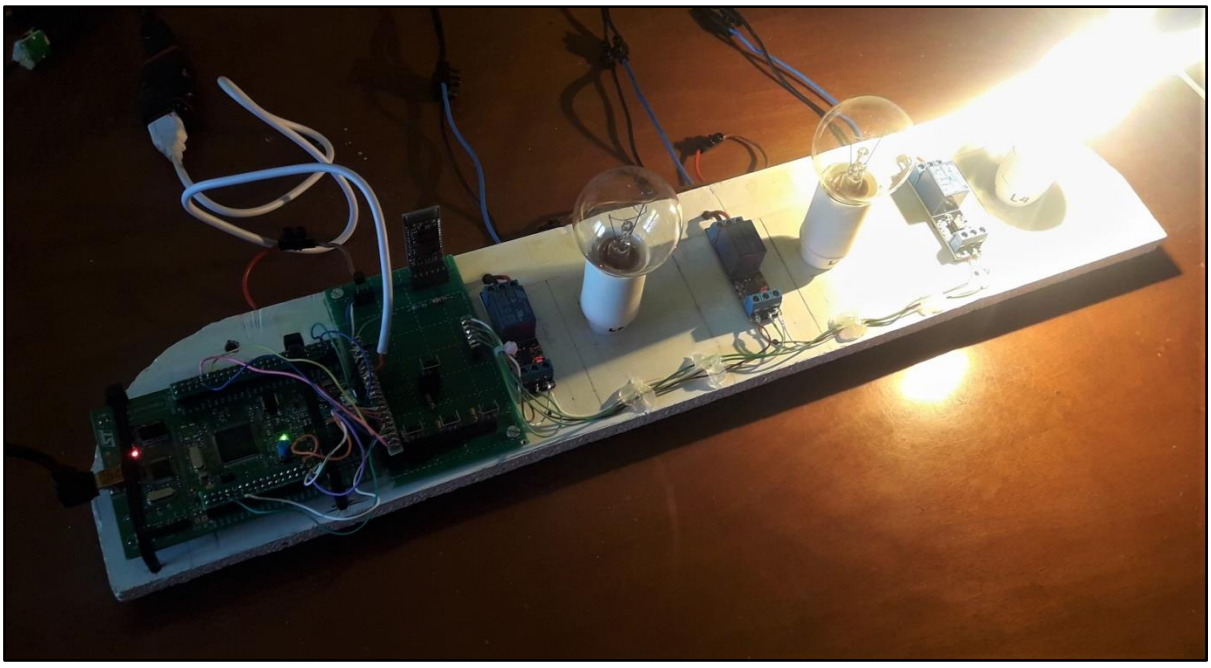
# Objetivo del proyecto

La meta que persigue nuestro sistema embebido es que el usuario pueda tener un cómodo y total control, de forma centralizada, sobre un circuito de domótica de iluminación que podría por ejemplo pertenecer a un hogar o un edificio.

Las funciones de este sistema serán:

- Apagado y encendido de luces de forma manual por pulsador y bluetooth.
- Captación de temperatura del ambiente.
- Rutina automatizada de encendido y apagado de luces.
- Apagado y encendido de una luz dependiendo de la temperatura del entorno.





# Funciones y pines utilizados del microprocesador

**1. Entradas digitales:**

- PA0 (Botón de encendido/apagado)
- PC1 (Botón para el cambio de modo de usuario) [Cable verde]
- PC13 (Botón para encendido/apagado de la lámpara 3) [Cable amarillo]
- PC14 (Botón para encendido/apagado de la lámpara 2) [Cable azul]
- PC15 (Botón para encendido/apagado de la lámpara 1) [Cable rosa]

**2. Salidas digitales:**

- PE11 (Lámpara 1) [Cable morado]
- PE12 (Lámpara 2) [Cable naranja]
- PE13 (Lámpara 3) [Cable blanco-verde]
- PD12 (Led verde del microcontrolador para indicar el modo de usuario)
- PD14 (Led rojo del microcontrolador para indicar el estado de funcionamiento)
- PD15 (Led azul del microcontrolador para indicar el sobrepaso de una determinada temperatura)

**3. Timers:**

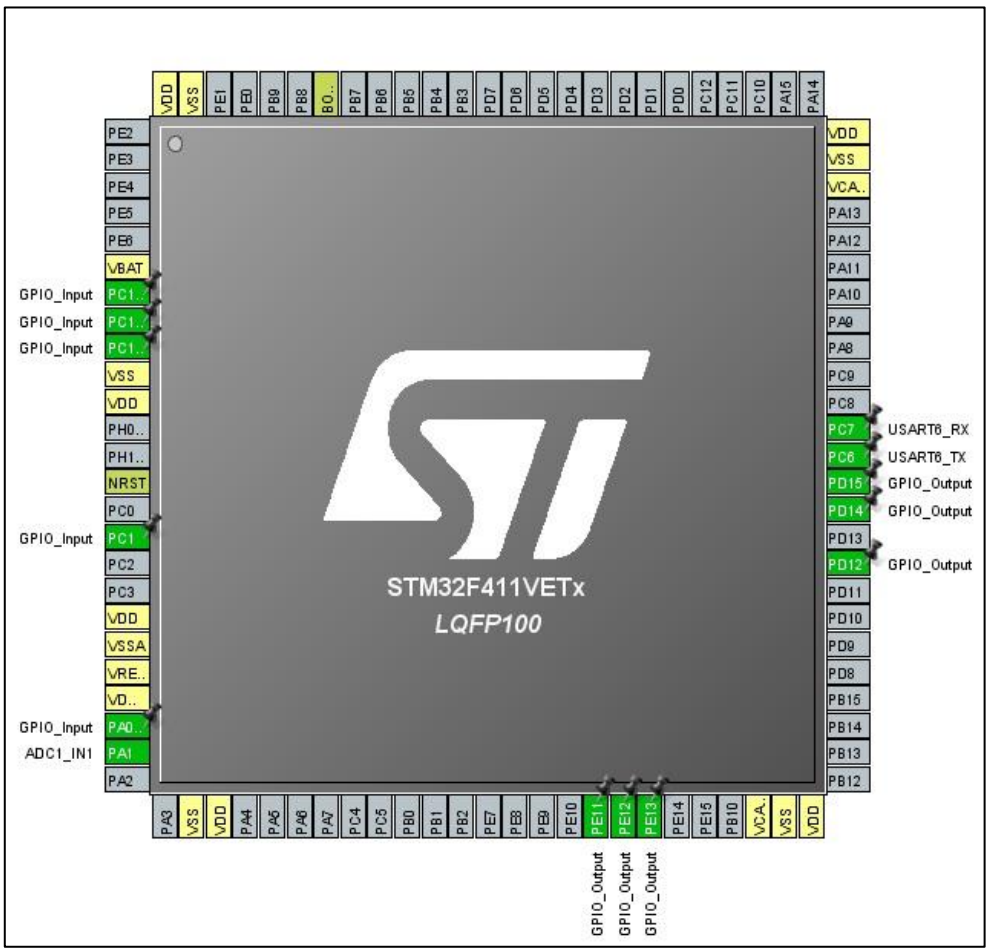
- Timer 3 (Usado en la rutina automatizada de apagado y encendido de luces)

**4. ADC:**

- PA1 (Entrada analógica IN1 para la lectura del sensor de temperatura)

**5. Comunicación serie:**

- PC6 (Pin de transmisión)
- PC7 (Pin de recepción)




Ventana de visualización de la configuración de los pines en STM32CubeMX

## Comportamiento del sistema

1. Para empezar el sistema podrá ser encendido o apagado (standby) con el botón azul de la placa. Cuando esté encendido, el usuario podrá hacer uso de todos los servicios que ofrece el microcontrolador. Cuando esté apagado (o en standby) no se permitirá al usuario hacer uso del sistema domótico de luces, estando únicamente habilitado el botón azul para cambiar el estado del sistema.

El estado en que se encuentra el sistema es representado por el led rojo de la placa, estando encendido cuando se encuentra en standby, y apagado cuando esté en funcionamiento.

2. Una vez que el sistema está en marcha, tendremos un pulsador (conectado al pin PC1 de la placa) con el que podremos pasar del modo manual al automatizado y viceversa.

- Cuando nos encontramos en el modo manual podremos encender y apagar las lámparas que queramos actuando sobre sus respectivos pulsadores o a través de un dispositivo android vía bluetooth.

- Por otro lado, cuando nos encontramos en el modo automático el sistema encenderá y apagará las luces siguiendo una secuencia ya programada, controlada por el Timer 3.

Para saber en qué modo de funcionamiento nos encontramos solo debemos fijarnos en el estado del led verde del microcontrolador. Si está encendido, significa que estamos en el modo automático y si está apagado nos encontramos en modo manual.

3. En tercer lugar cabe señalar que hay un sensor de temperatura (TMP36) el cual tiene absoluto control sobre la segunda lámpara si nos encontramos en modo manual. Cuando se ejecute por primera vez el sistema, se captura y almacena la temperatura ambiente.

Si en un determinado momento la temperatura captada por el sensor sobrepasa en dos grados Celsius a la temperatura ambiente (se excede el límite), la segunda lámpara se encenderá y el usuario no podrá actuar sobre ella por mucho que use el pulsador de dicha lámpara. Si la temperatura captada por el sensor baja del límite anteriormente mencionado, la lámpara dos se apaga y el usuario vuelve a tener el mando sobre ella.

4. Para la comunicación serie entre el microcontrolador y un dispositivo android se ha utilizado un HC-05 configurado como esclavo. Los parámetros de la comunicación serie de ambos extremos son: 9600 baudios, 8 bits de datos, ningún bit de paridad y un bit de parada.

5. Por último está el botón negro del microcontrolador, el cual podemos pulsar siempre que queramos reiniciar el sistema. Cada vez que se reinicie el sistema, el sensor actualizará el valor captado de la temperatura ambiente.

# Diagramas

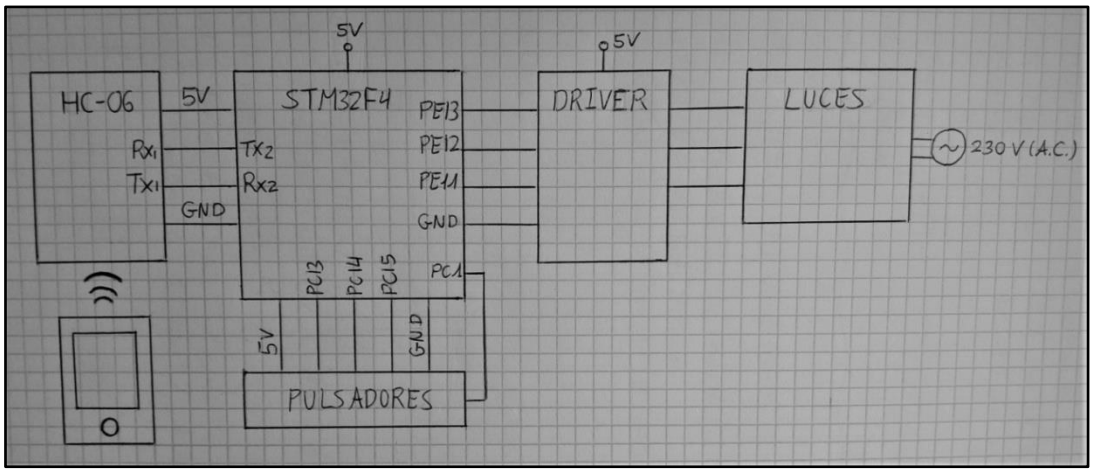



Diagrama de la composición del sistema.

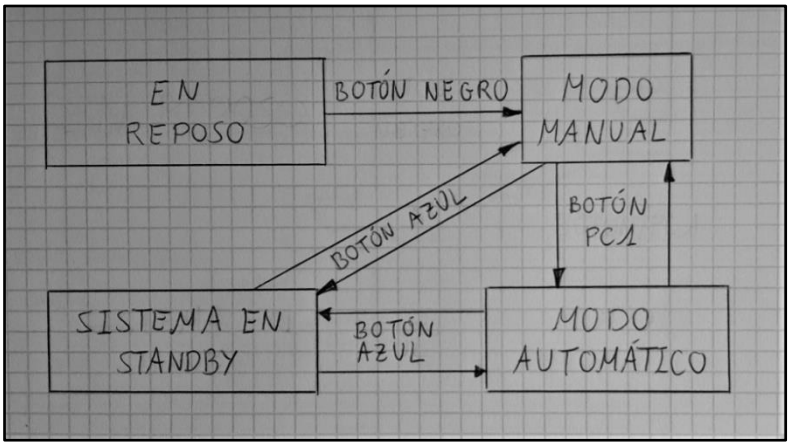



Diagrama de bloques de funcionamiento del sistema.

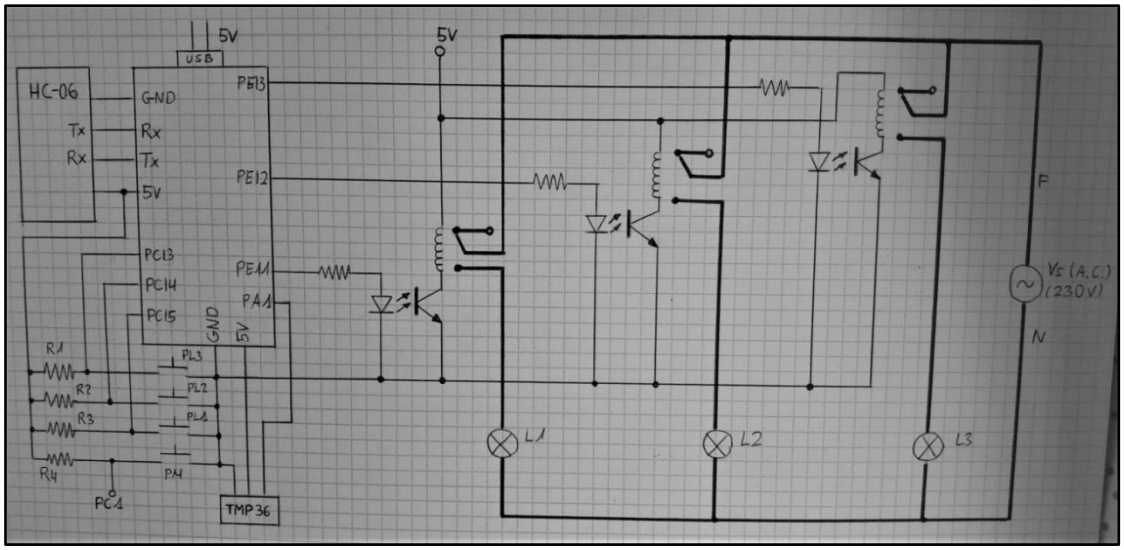



Diagrama de los circuitos de control y potencia del que está compuesto el sistema.

# Detalles de algunas funciones del programa

**1. Debounce:** en primer lugar tenemos un algoritmo que se repite bastante a lo largo del programa, utilizado para evitar los rebotes de una señal enviada por un pulsador y confirmar si dicho pulsador se ha accionado correctamente o no.

Ejemplo:

Dentro de *while (1)* en *main.c*

```
if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 1 && boton_pulsado_azul == 0){
	last_time = millis(); //Guarda el valor de millis()
	boton_pulsado_azul = 1;
}

if ((HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 1) && (millis() - last_time >
200)){
	boton_pulsado_azul = 0;
	if (EnEspera_Funcionando == 0){
		HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12, GPIO_PIN_RESET);
		HAL_GPIO_WritePin(GPIOE, GPIO_PIN_13, GPIO_PIN_RESET);
		EnEspera_Funcionando = 1;
	}
//Continua el programa…
}
```

Para entender el ejemplo hay que mencionar que *millis ()* es una función que devuelve el número de ticks de reloj que se llevan contados. La cuenta de estos ticks es llevada a cabo gracias al Timer SysTick que está configurado para que genere interrupciones cada 1 ms.

En el ejemplo, al pulsar el botón azul del microprocesador, en cuanto se detecte el primer pico de los rebotes generados, se guardará la cuenta que lleva *millis()* en una variable global llamada *last_time*. Como ya hemos tomado la medida de tiempo, no nos interesa alterar *last_time*, por lo que hacemos que *boton_pulsado_azul* sea 1 y así no entrará más al primer if.

Por tanto la única forma de entrar al segundo if es que haya pasado un tiempo de 200 ms desde que se guardó la cuenta del reloj en *last_time* y que la señal percibida del botón siga siendo 1. De esta forma queda confirmado que el botón se ha pulsado correctamente y se desarrollan las instrucciones correspondientes.

**2. Sensor de temperatura:** el sensor de temperatura fue implementado en el sistema con la finalidad de poder leer la temperatura ambiente, pasando su señal analógica a una señal digital por medio del ADC del microcontrolador.

Ejemplo:

Dentro del *while (1)* en *main.c*

```
HAL_ADC_Start(&hadc1);


if (HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK){
	adcValue = HAL_ADC_GetValue(&hadc1);
}


if(adcValue > 0){
	temperature = (float)adcValue/1024;
	temperature = temperature * 0.7;
	temperature = temperature - 0.01;
	temperature = temperature * 100;
}


if (temperatura_tomada == 0){
	temp_ambiente = temperature;
	temperatura_tomada = 1;
}
//Continúa el programa…
HAL_Delay(50);
//Continúa el programa…
```

Se definió que la resolución con la que se obtiene la señal del sensor fuera de 10 bits. Es decir, que la señal es discretizada en 1024 valores. Como el sensor utilizado es un TMP36, por su hoja de características, podemos convertir el valor que va de cero a 1023 en un voltaje y más tarde pasar ese voltaje a grados Celsius.

La primera vez que el sensor toma una medida, esta se guarda como la temperatura del ambiente en el que nos encontramos, ya que más tarde queremos que la segunda lámpara y el led azul del microcontrolador se enciendan cuando la temperatura sobrepase en dos grados a la temperatura ambiente.

Por último se usa un delay para que haya una pequeño tiempo de espera entre tomas de medida.

**3. Timer 3:** este timer de interrupciones tiene el objetivo de encender y apagar luces, cada un determinado número de segundos, siguiendo una secuencia ya programada, cuando el sistema se encuentra en modo de usuario automático.

Ejemplo:

En *main.c*

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	if (htim->Instance == TIM3){	//Si la fuente es TIM3

		if (user_mode == 1){

			if (paso == 0){
				HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11);
				paso++;
			}
//Continúa el programa…
}
```

La forma más fácil de escribir el handler de interrupciones es simplemente llamando a *HAL_TIM_IRQHandler ().* El handler provisto por HAL verificará automáticamente qué evento causó la interrupción e invocará uno de los handler especializados, que en nuestro caso se trata de *HAL_TIM_PeriodElapsedCallback ().*

Una vez dentro de dicho handler se confirma que la fuente de la interrupción es el Timer 3 y, realizado esto, se desarrollan los distintos pasos de la secuencia.

## Versiones del programa

**Primera versión**

**1. Cambios**

- El proyecto permite encender y apagar las luces de forma manual y cambiar al modo automático (que por ahora funciona con Delays).

- Con el botón azul (que llama a una interrupción cada vez que es pulsado) alternamos entre estado de funcionamiento y el estado apagado/standby.

**2. Problemas**

Una vez que se entra en modo automático no se puede volver al modo manual. Lo único que queda es resetearlo todo con el botón negro del microprocesador.

**3. Soluciones**

Utilizar un Timer de interrupciones (se implementará en la versión 3).

**Segunda versión**

**1. Cambios**

- Se han añadido los leds rojo y verde del microcontrolador para que el usuario pueda ver en qué estado y modo se encuentra el sistema.

**Tercera versión**

**1. Cambios**

- Se ha añadido un debounce para todos los botones excepto el de cambio de modo de usuario ya que por el momento parece no necesitarlo.

- Se ha hecho más corto el periodo del Timer 2 para que capture con una mayor exactitud la señal del pulsador de cambio de modo de usuario.

- Como no lo necesitaba, se ha convertido el botón azul del microprocesador en una entrada digital normal y corriente, por lo tanto ya no llama a una interrupción cada vez que es accionado.

**2. Problemas**

Se quiso hacer un debounce con un while para esperar un tiempo umbral y así comparar la señal de entrada del principio y del final de dicho tiempo pero empezó a dar muchos problemas y a haber conflictos entre diferentes acciones.

**3. Soluciones**

Simplemente para no añadir otro loop utilizamos el while(1) del main.

**Cuarta versión**

**1. Cambios**

- Se han sustituido los delays del modo automático por tiempos del Timer 3.

- El Timer 3 se habilita cuando se haya pulsado el botón para pasar a modo automático y se deshabilita cuando se pase al modo manual.

**Quinta versión**

**1. Cambios**

- Se ha añadido el sensor de temperatura para hacer uso del conversor de señal analógica a digital del microcontrolador.

- Se ha habilitado el led azul de la placa que se encenderá cuando el sensor haya detectado que se pasa una determinada temperatura límite.

- Se ha hecho que la segunda lámpara se encienda cuando el sensor detecte que la temperatura se encuentra a dos grados por encima de la temperatura ambiente. Cuando el usuario se encuentre en ese caso, no podrá influir en el comportamiento de la lámpara mediante su pulsador. Si baja con respecto a ese límite (temperatura ambiente + 2 grados), la luz se apagará dejando de nuevo el mando al usuario.

**Sexta versión**

**1. Cambios**

- Se ha eliminado el Timer 2 porque ya no era necesario cada determinado tiempo verificar si se había accionado el pulsador de cambio de modo de usuario; ahora funciona como cualquier otro pulsador del sistema.

- Se ha añadido comunicación serie mediante bluetooth, haciendo uso de un HC-05 configurado como esclavo. Podremos encender y apagar las luces, cambiar de modo de usuario, cambiar de estado de funcionamiento y leer la temperatura en el momento que queramos con un dispositivo android.

**Séptima versión**

**1. Problemas**

- Las anteriores versiones se estaban realizando con leds externos como simulación del circuito de potencia donde se situarían las lámparas. A la hora de sustituir dichos leds por lámparas de corriente alterna debemos usar relés como drivers para comunicar el circuito de mando con el de potencia. Sin embargo, la lógica de control de los relés invierte la señal digital recibida del microcontrolador. Es decir, si por ejemplo, el microcontrolador manda un SET a la primera lámpara para que se encienda, la lógica electrónica del relé lo invertirá a un RESET haciendo así que la lámpara se apague).

**2. Soluciones**

- Simplemente hay que cambiar el software para que los valores de las señales digitales se manden de forma invertida al circuito de potencia.

# Programa del sistema

## Main.c

```
#include "main.h"
#include <string.h>
#include "stm32f4xx_hal.h"

#define REF_Debounce 100
#define DIF_Temp 1

ADC_HandleTypeDef hadc1;
TIM_HandleTypeDef htim3;
UART_HandleTypeDef huart6;

//Para modos de funcionamiento
volatile int EnEspera_Funcionando = 0;
volatile int user_mode = 0;
int Sis_BT = 0;

//Para el debounce
extern uint32_t last_time;
extern uint32_t counter_ms;

//Para estado de los pulsadores
int boton_pulsado_modo = 0;
int boton_pulsado_azul = 0;
int boton_pulsado_L1 = 0;
int boton_pulsado_L2 = 0;
int boton_pulsado_L3 = 0;
int N = 0;

//Para marcar en qué momento de la secuencia de user_mode = 1 se encuentra
volatile int paso = 0;
```

```
//Para el conversor analógico-digital
uint32_t adcValue;
float temperature;
float temp_ambiente;
int temperatura_tomada;
int sensor_activa;

//Para comunicación serie bluetooth
char rx_buffer[10];
char tx_buffer[10];

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM3_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART6_UART_Init(void);
static void MX_NVIC_Init(void);
void ConfigureSysTick(void);
uint32_t millis(void);

int main(void){
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM3_Init();
  MX_ADC1_Init();
  MX_USART6_UART_Init();
  MX_NVIC_Init();

  //Inicialización de timers
  SysTick_Config(SystemCoreClock/1000); //Para interrupciones cada 1ms
  HAL_TIM_Base_Start_IT(&htim3);
  NVIC_DisableIRQ(TIM3_IRQn);

  //Inicialización de variables
  user_mode = 0;
  EnEspera_Funcionando = 0;
```

```
  boton_pulsado_modo = 0;
  boton_pulsado_azul = 0;
  boton_pulsado_L1 = 0;
  boton_pulsado_L2 = 0;
  boton_pulsado_L3 = 0;
  paso = 0;

  //Para el oonversor analógico-digital
  adcValue = 0;
  temperature = 0;
  temp_ambiente = 0;
  temperatura_tomada = 0;
  sensor_activa = 0;

  while (1)
  {
              HAL_ADC_Start(&hadc1);

              if (HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK){
                     adcValue = HAL_ADC_GetValue(&hadc1); //Si la conversión es
exitosa, leemos el valor adc.
              }

              if(adcValue > 0){
                     temperature = (float)adcValue/1024;
                     temperature = temperature * 0.7; //Obtener temperatura en
forma de voltaje
                     temperature = temperature - 0.01;      //Resta el offset
de la recta
                     temperature = temperature * 100; //Obtener temperatura en
forma de grados
              }

              if (temperatura_tomada == 0){
                     temp_ambiente = temperature;
                     int int_temp_ambiente = (int)temp_ambiente;
```

```
                    HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer, sprintf
(tx_buffer, "Temperatura ambiente: %d\n", int_temp_ambiente), 500);


                    temperatura_tomada = 1;

             }


             if (temperature >= (temp_ambiente + DIF_Temp)){

                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_SET);

             }


             else{

                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, GPIO_PIN_RESET);


             }


             HAL_Delay(50);


             HAL_UART_Receive(&huart6, (uint8_t*)rx_buffer, 10, 500);


                    //MOSTRAR TEMPERTAURA

                    if(rx_buffer[0] == 'T' && rx_buffer[1] == 'E' &&
rx_buffer[2] == 'M' && rx_buffer[3] == 'P'){

                           int int_temperatura = temperature;

                           HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Temperatura: %d\n", int_temperatura), 500);

                           memset (rx_buffer, 0, 10);

                    }


                    //LUZ 1

                    else if(rx_buffer[0] == 'L' && rx_buffer[1] == '1' &&
rx_buffer[2] == 'O' && rx_buffer[3] == 'N'){

                           HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11,
GPIO_PIN_RESET);

                           memset (rx_buffer, 0, 10);

                           HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Lampara 1 encendida\n"), 500);

                    }


                    else if(rx_buffer[0] == 'L' && rx_buffer[1] == '1' &&
rx_buffer[2] == 'O' && rx_buffer[3] == 'F' && rx_buffer[4] == 'F'){

                           HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11, GPIO_PIN_SET);
```

```
                        memset (rx_buffer, 0, 10);

                        HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Lampara 1 apagada\n"), 500);

                  }


                  //LUZ 2

                  else if(rx_buffer[0] == 'L' && rx_buffer[1] == '2' &&
rx_buffer[2] == 'O' && rx_buffer[3] == 'N'){

                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12,
GPIO_PIN_RESET);

                        memset (rx_buffer, 0, 10);

                        HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Lampara 2 encendida\n"), 500);

                  }


                  else if(rx_buffer[0] == 'L' && rx_buffer[1] == '2' &&
rx_buffer[2] == 'O' && rx_buffer[3] == 'F' && rx_buffer[4] == 'F'){

                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12, GPIO_PIN_SET);

                        memset (rx_buffer, 0, 10);

                        HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Lampara 2 apagada\n"), 500);


                  }


                  //LUZ 3

                  else if(rx_buffer[0] == 'L' && rx_buffer[1] == '3' &&
rx_buffer[2] == 'O' && rx_buffer[3] == 'N'){

                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_13,
GPIO_PIN_RESET);

                        memset (rx_buffer, 0, 10);

                        HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Lampara 3 encendida\n"), 500);


                  }


                  else if(rx_buffer[0] == 'L' && rx_buffer[1] == '3' &&
rx_buffer[2] == 'O' && rx_buffer[3] == 'F' && rx_buffer[4] == 'F'){

                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_13, GPIO_PIN_SET);

                        memset (rx_buffer, 0, 10);

                        HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Lampara 3 apagada\n"), 500);
```

```
                }


            if (rx_buffer[0] == 'S' && rx_buffer[1] == 'I' && rx_buffer[2] ==
'S' && rx_buffer[3] == 'O' && rx_buffer[4] == 'F' && rx_buffer[5] == 'F'){

                Sis_BT = 1; //EL SISTEMA PASA A ESTADO STANDBY POR
BLUETOOTH

            }


            else if (rx_buffer[0] == 'S' && rx_buffer[1] == 'I' &&
rx_buffer[2] == 'S' && rx_buffer[3] == 'O' && rx_buffer[4] == 'N') {

                Sis_BT = 2; //EL SISTEMA PASA A ESTADO DE FUNCIONAMIENTO
POR BLUETOOTH

            }


            if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 1 &&
boton_pulsado_azul == 0){

                last_time = millis(); //Guarda el valor de millis()

                boton_pulsado_azul = 1;

            }


            if (((HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0) == 1) && ((millis() -
last_time) > REF_Debounce)) || Sis_BT == 1 || Sis_BT == 2){

                boton_pulsado_azul = 0;


                if (EnEspera_Funcionando == 0 || Sis_BT == 1){ //APAGA EL
SISTEMA (STANDBY)

                    memset (rx_buffer, 0, 10);

                  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11, GPIO_PIN_SET);

                    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12, GPIO_PIN_SET);

                    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_13, GPIO_PIN_SET);


                    EnEspera_Funcionando = 1;

                    HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Sistema apagado (stanby)\n"), 500);


                }


                else if(EnEspera_Funcionando == 1 || Sis_BT == 2){ //PONE
EL SISTEMA EN FUNCIONAMIENTO

                    memset (rx_buffer, 0, 10);

                  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11, GPIO_PIN_SET);
```

```
                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12, GPIO_PIN_SET);
                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_13, GPIO_PIN_SET);

                        EnEspera_Funcionando = 0;
                        HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Sistema en funcionamiento\n"), 500);

                }

                user_mode = 0;
                Sis_BT = 0;
            }

            if (EnEspera_Funcionando == 0){ //ESTADO EN FUNCIONAMIENTO

                HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
//Led rojo se apaga

                //Se pulsa el boton de cambio de modo
                if ((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_1) == 1) &&
boton_pulsado_modo == 0){
                        last_time = millis(); //Guarda el valor de
millis()
                        boton_pulsado_modo = 1;
                }

                //Se confirma que se ha pulsado y cambia a modo automático
                if(((user_mode == 0 && HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_1)
== 0) && ((millis() - last_time) > REF_Debounce) && (boton_pulsado_modo == 1))
|| (rx_buffer[0] == 'M' && rx_buffer[1] == 'O' && rx_buffer[2] == 'D' &&
rx_buffer[3] == 'A')){
                        memset (rx_buffer, 0, 10);
                        NVIC_EnableIRQ(TIM3_IRQn);
                        HAL_GPIO_WritePin (GPIOD, GPIO_PIN_12,
GPIO_PIN_SET); //Led verde encendido

                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11,
GPIO_PIN_SET);
                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12,
GPIO_PIN_SET);
                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_13,
GPIO_PIN_SET);
                        user_mode = 1;
```

```
                        HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Cambio a modo automatico\n"), 500);


                    }


                    //Se confirma que se ha pulsado y cambia a modo manual

                    else if((user_mode == 1 && (HAL_GPIO_ReadPin(GPIOC,
GPIO_PIN_1) == 0) && ((millis() - last_time) > REF_Debounce) &&
(boton_pulsado_modo == 1)) || (rx_buffer[0] == 'M' && rx_buffer[1] == 'O' &&
rx_buffer[2] == 'D' && rx_buffer[3] == 'M')){

                            memset (rx_buffer, 0, 10);

                            NVIC_DisableIRQ(TIM3_IRQn);


                            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11,
GPIO_PIN_SET);

                            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12,
GPIO_PIN_SET);

                            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_13,
GPIO_PIN_SET);

                            user_mode = 0;

                        HAL_UART_Transmit(&huart6, (uint8_t*)tx_buffer,
sprintf (tx_buffer, "Cambio a modo manual\n"), 500);


                            boton_pulsado_modo = 0;


                    }


                    //USUARIO EN MODO MANUAL

                    if (user_mode == 0){


                        HAL_GPIO_WritePin (GPIOD, GPIO_PIN_12,
GPIO_PIN_RESET); //Led verde apagado


                        //Si se pulsa el primer pulsador...

                        if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == 0 &&
boton_pulsado_L1 == 0){

                            last_time = millis(); //Guarda el valor de
millis()

                            boton_pulsado_L1 = 1;

                        }


                        if ((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == 0) &&
(millis() - last_time) > REF_Debounce && (boton_pulsado_L1 == 1)){
```

```
                                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11);
                                        boton_pulsado_L1 = 0;
                                }

                                //Si se detecta una temperatura superior a la
ambiente en tres grados o se pulsa el segundo pulsador...

                                if (temperature <= (temp_ambiente + DIF_Temp) &&
sensor_activa == 0){
                                        if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14) == 0
&& boton_pulsado_L2 == 0){
                                                last_time = millis(); //Guarda el valor
de millis()
                                                boton_pulsado_L2 = 1;
                                        }

                                        if ((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_14) ==
0) && (millis() - last_time) > REF_Debounce && (boton_pulsado_L2 == 1)){
                                                HAL_GPIO_TogglePin(GPIOE,
GPIO_PIN_12);
                                                boton_pulsado_L2 = 0;
                                        }
                                }

                                if (sensor_activa == 0 && temperature >
(temp_ambiente + DIF_Temp)){
                                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12,
GPIO_PIN_RESET);
                                        sensor_activa = 1;
                                }

                                else if (sensor_activa == 1 && (temperature <=
(temp_ambiente + DIF_Temp))){
                                        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12,
GPIO_PIN_SET);
                                        sensor_activa = 0;
                                }

                                //Si se pulsa el tercer pulsador...
                                if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0 &&
boton_pulsado_L3 == 0){
```

```
                                last_time = millis(); //Guarda el valor de
millis()
                                boton_pulsado_L3 = 1;
                          }

                          if ((HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0) &&
(millis() - last_time) > REF_Debounce && (boton_pulsado_L3 == 1)){
                                HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_13);
                                boton_pulsado_L3 = 0;
                          }
                    }
              }

              else if (EnEspera_Funcionando == 1){ //ESTADO STANDBY
                    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET); //Led
rojo
                    HAL_GPIO_WritePin (GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
//Led verde

                    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11, GPIO_PIN_SET);
                    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_12, GPIO_PIN_SET);
                    HAL_GPIO_WritePin(GPIOE, GPIO_PIN_13, GPIO_PIN_SET);
              }
  }
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
```

```
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_NVIC_Init(void)
{
  HAL_NVIC_SetPriority(TIM3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(TIM3_IRQn);

  HAL_NVIC_SetPriority(ADC_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(ADC_IRQn);

  HAL_NVIC_SetPriority(USART6_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(USART6_IRQn);
}

static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};

  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc1.Init.Resolution = ADC_RESOLUTION_10B;
  hadc1.Init.ScanConvMode = DISABLE;
```

```
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

static void MX_TIM3_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  htim3.Instance = TIM3;
  htim3.Init.Prescaler = 35999;
  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim3.Init.Period = 1600;
  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_USART6_UART_Init(void)
{
  huart6.Instance = USART6;
  huart6.Init.BaudRate = 9600;
  huart6.Init.WordLength = UART_WORDLENGTH_8B;
  huart6.Init.StopBits = UART_STOPBITS_1;
  huart6.Init.Parity = UART_PARITY_NONE;
  huart6.Init.Mode = UART_MODE_TX_RX;
  huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart6.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart6) != HAL_OK)
  {
    Error_Handler();
  }
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOC_CLK_ENABLE();
```

```
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13, GPIO_PIN_SET);

  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12|GPIO_PIN_14|GPIO_PIN_15,
GPIO_PIN_RESET);

  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15|GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_14|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
       if (htim->Instance == TIM3){     //Si la fuente es TIM3
```

```
            if (user_mode == 1){

                  if (paso == 0){
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11);
                        paso++;
                  }

                  else if (paso == 1){
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_12);
                        paso++;
                  }

                  else if (paso == 2){
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11);
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_13);
                        paso++;
                  }

                  else if (paso == 3){
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11);
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_12);
                        paso++;
                  }

                  else if (paso == 4){
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_11);
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_12);
                        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_13);
                        paso = 0;
                  }
            }
      }
}

uint32_t millis() {
  return counter_ms;
}
```

```
void Error_Handler(void)
{
}

#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line)
{
}
#endif
```

**Stm32f4xx_it.c**

```
#include "main.h"
#include "stm32f4xx_it.h"

uint32_t last_time = 0;
uint32_t counter_ms = 0;

extern ADC_HandleTypeDef hadc1;
extern TIM_HandleTypeDef htim3;
extern UART_HandleTypeDef huart6;

void NMI_Handler(void){}

void HardFault_Handler(void){
  while (1){
  }
}

void MemManage_Handler(void){
  while (1) {
  }
}

void BusFault_Handler(void){
  while (1){
  }
```

```
}

void UsageFault_Handler(void){
  while (1){
  }
}

void SVC_Handler(void){}

void DebugMon_Handler(void){}

void PendSV_Handler(void){}

void SysTick_Handler(void){
  HAL_IncTick();
  counter_ms++;
}

void ADC_IRQHandler(void){
  HAL_ADC_IRQHandler(&hadc1);
}

void TIM3_IRQHandler(void){
  HAL_TIM_IRQHandler(&htim3);
}

void USART6_IRQHandler(void){
  HAL_UART_IRQHandler(&huart6);
}
```