PROYECTO SISTEMAS ELECTRÓNICOS DIGITALES

CURSO 2022/23

MICROS: Control domótico de vivienda

APELLIDOS, NOMBRE:

de Martino Garza, Pedro (55207)

Collado González, Rosa M (50168)

Feijoo Rico, Víctor (55228)

Indice

Contenido

A. Introducción

Breve resumen del enunciado.

El siguiente proyecto consiste en la realización de una aplicación domótica con el microcontrolador, concretamente el control de luces, de temperatura y la activación de una alarma para una vivienda, que permita la opción de un control manual, así como uno automático.

B. Estrategia

Descripción de la estrategia y algoritmos desarrollados para realizar el trabajo, justificando las soluciones adoptadas.

Para analizar la estrategia, se hace necesario agrupar en bloques de funcionamiento los distintos servicios que ofrece el programa.

1. CONTROL DE LUCES
   El control de luces tiene 3 partes distintas atendiendo a 3 casuísticas diferentes.
   1.1. Control según luz ambiente
      Se trata de regular un led con ayuda de un LDR, de manera que a partir de cierta oscuridad, este se encienda con distintos valores según la intensidad lumínica necesaria dependiendo de la oscuridad del exterior. Esto se conseguirá con un control tipo PWM del LED.
   1.2. Control mediante presencia
      Para ello se incluye un pulsador capaz de apagar y encender el modo automático de la luz para cuando el usuario quiera o no usar la detección automática. Si el modo automático se encuentra desactivado, la luz se enciende y apaga según el comportamiento de un pulsador. Si el modo automático está activado, se enciende un led para avisar de este evento y el led se encenderá cuando detecte movimiento. Se apagará pasados dos minutos después de no detectar movimiento.
   1.3. Control mediante regulador
      Se regula mediante PWM la intensidad lumínica de un led utilizando un potenciómetro como regulador.

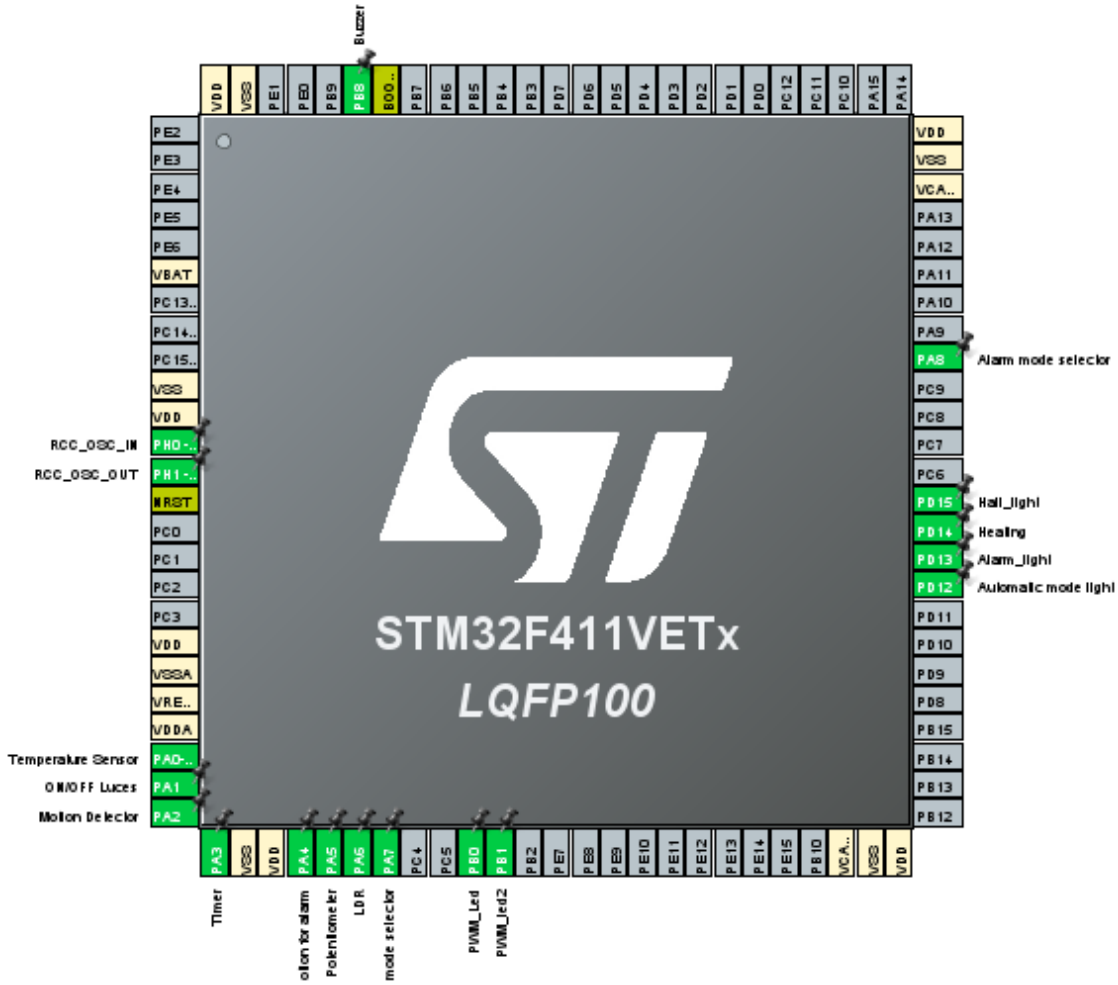
2. CONTROL DE TEMPERATURA
   Se usa el sensor interno de la placa STM para registrar la temperatura y, cuando esta caiga sobre los 25 grados, se activará un led a modo de resistencia de calor.

3. ALARMA
   Para el control de esta función, se ha dispuesto de un pulsador, una luz, un detector de movimiento y un zumbador (PWM). Cuando se activa el pulsador, se enciende el modo alarma y se enciende un led indicador. Si después de activarlo el sistema detecta un movimiento, saltará la señal sonora del buzzer. Si se pulsa el botón de modo alarma de nuevo, se desactiva el sonido del buzzer. Sino, pasado un tiempo también se desactiva para no generar una contaminación acústica sostenida en el tiempo.

Por facilitar la comprensión del lector, la explicación de cada servicio del programa se estructurará en bloques.

C. Distribución y ajuste de pines

Se utilizaron pines de interrupción, timers (PWM y de tipo básico), conversores analógico-digital y digitales



1. Pines de interrupción (GPIO_EXTI)
   - Se implementa el interruptor de encendido/apagado de luces manual en PA1
   - Se implementa el detector de movimiento para encendido de luces en PA2
   - Se implementa el botón que enciende y apaga el modo de alarma en PA8
   - Se implementa el detector de presencia que una vez activado el modo alarma la hace saltar en PA4
   - Se implementa el botón que enciende el modo de funcionamiento automático en el PA7

2. Pines de temporización (TIM)
   Se usan tres temporizadores distintos. Para el control de las luces se usa el TIM3, para el zumbador el TIM10 y para el tiempo de encendido de la luz asociada al detector de presencia (que funciona así en modo automático) se usó el TIM2. Se estructuró de la siguiente manera:
   - El tiempo de espera para posterior apagado de la luz interna asociada al detector de movimiento se regula con un temporizador en el canal 4 del TIM2 en el pin PA3
   - La regulación de la luz externa según los valores del LDR, se realiza a través del canal 3 del TIM3 en el pin PB0
   - La luz interior que varía intensidad según los valores de un potenciómetro, se realiza a través del canal 4 del TIM3 en el pin PB1
   - La regulación PWM del Buzzer se ha asociado al pin PB8, en el canal 1 del TIM10
3. Pines para conversión analógico-digital (ADC_IN)
   - El pin PA0 hace seguimiento del sensor de temperatura interno de la placa
   - El potenciómetro que regula la intensidad de la luz se encuentra asociado al PA5
   - El LDR que devuelve los diferentes valores según la luz que reciba se asoció a PA6
4. Pines digitales (GPIOD)
   - Para el led que indica el modo automático se usó PD12
   - Para la luz de alarma, PD13
   - Para el calefactor, PD14
   - Para la luz de la entrada que se activa por movimiento en modo automático o con el pulsador en modo manual, se usó PD15.

D. FUNCIONES DEL PROGRAMA E IMPLEMENTACIÓN

a. Posterior a *int main*

Antes del *int main* del programa y para su correcto funcionamiento, es necesario incluir tres librerías: la propia que regula las funciones y palabras reservadas de la placa usada, STM32F411VE para el caso; la que da acceso a las funciones tipo *hal* y la que dan acceso a la librería asociada al *main.c* que es donde desarrollamos el programa.

```
/* Includes ------------------
#include "main.h"
#include "stm32f4xx.h"
#include "stm32f4xx_hal.h"
```

También, es necesario iniciar las funciones asociadas a la configuración del reloj del sistema, el de los pines GPIO, ADC y los dos timers que se usarán, TIM3 y TIM10.

```
/* Private function prototypes --
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM10_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
```

Por otro lado, y para hacer más intuitiva la lectura del programa, reducir el margen de fallos y poder modificar más fácilmente el diseño, se definieron los pines y palabras claves que usaremos durante su desarrollo en la librería del *main*, *main.h*, usando varios defines:

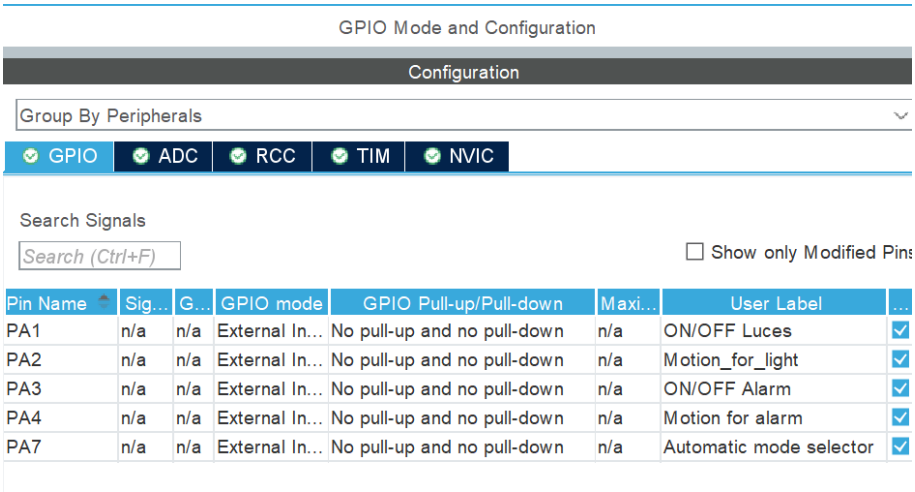
```
#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------
#include "stm32f4xx_hal.h"

/* Private includes ----------------------
/* USER CODE BEGIN Includes */

#define Switch GPIO_PIN_1
#define Motion_detector GPIO_PIN_2
#define Alarm_mode_selector GPIO_PIN_8
#define Alarm_detector GPIO_PIN_4
#define Automatic_mode_selector GPIO_PIN_7
#define Automatic_mode_light GPIO_PIN_12
#define Alarm_light GPIO_PIN_13
#define Heating GPIO_PIN_14
#define Hall GPIO_PIN_15
#define Potentiometer hadc3
#define Temp_reader hadc1
#define LDR hadc2
#define Lights htim3
#define Buzzer htim10
#define Buzzer_Channel TIM_CHANNEL_1
#define Tempor htim2
#define Tempor_Channel TIM_CHANNEL_4
#define Tempor_hall_light TIM_CHANNEL_1
#define Outside_light TIM_CHANNEL_3
#define Room_light TIM_CHANNEL_4
```

Volviendo al desarrollo de *main.c* y teniendo en cuenta la asignación de nombres en la librería, se declararon también las variables que harán posible la conversión analógico-digital de tipo *ADC_HandleTypeDef*. Se procedió de manera análoga para las dos variables de tipo TIM, teniendo en cuenta que en el TIM3 se aúna la gestión de las luces aunque en distintos canales.

```
/* Private variables ----------------
ADC_HandleTypeDef hadc1;
//ADC_HandleTypeDef LDR;
//ADC_HandleTypeDef Potentiometer;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef Tempor;
TIM_HandleTypeDef Lights;
TIM_HandleTypeDef Buzzer;
```

1. GESTIÓN DE INTERRUPCIONES

Las interrupciones son lideradas por 5 entradas distintas: La selección de modo automático o manual, el pulsador encendido/apagado de luces, el pulsador de encendido/apagado de la alarma y los dos detectores de movimiento: el que enciende la luz interior y el que hace saltar la alarma.



Pin Name	Sig...	G...	GPIO mode	GPIO Pull-up/Pull-down	Maxi...	User Label	...
PA1	n/a	n/a	External In...	No pull-up and no pull-down	n/a	ON/OFF Luces	☑
PA2	n/a	n/a	External In...	No pull-up and no pull-down	n/a	Motion_for_light	☑
PA3	n/a	n/a	External In...	No pull-up and no pull-down	n/a	ON/OFF Alarm	☑
PA4	n/a	n/a	External In...	No pull-up and no pull-down	n/a	Motion for alarm	☑
PA7	n/a	n/a	External In...	No pull-up and no pull-down	n/a	Automatic mode selector	☑

Se usó la función predefinida *HAL_GPIO_EXTI_Callback*, cambiando el valor de variables volátiles asociadas a la lectura de las mismas, por ejemplo se observa la variable *automatic_mode* inicializada a nivel bajo que se pone en nivel alto cuando la interrupción viene del pin asociado a *Automatic_mode_selector.* Cuando se activa el modo alarma, se enciende un led para hacer seguimiento de este estado. Como se trata de interrupciones, se escriben también antes del *int main.*

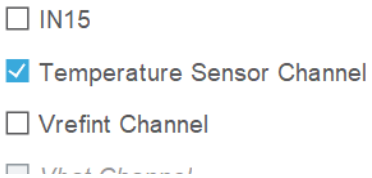
```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
volatile int motion = 0;
volatile int automatic_mode = 0;
volatile int sw = 0;
volatile int alarm_mode = 0;
volatile int alarm = 0;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    if (GPIO_Pin == Automatic_mode_selector) //mode selector
    {
        if(automatic_mode) {automatic_mode = 0;}
        else automatic_mode = 1;
    }
    if (GPIO_Pin == Switch) //interruptor de apagado/encendido de luces
    {
        if (sw) {sw = 0;}
        else sw = 1;
    }
    if (GPIO_Pin == Motion_detector) //detección de movimiento en la entrada
     {
        if (motion == 0) {motion = 1;}
        else motion = 0;
     }
```

```
if (GPIO_Pin == Alarm_mode_selector) //ON/OFF de alarma
     {
        if (alarm_mode) {alarm_mode = 0;}
        else
            {
            //HAL_DELAY(120); //2 minutos hasta que pueda saltar la alarma;
            alarm_mode = 1; HAL_GPIO_WritePin(GPIOD,Alarm_light,GPIO_PIN_SET);
            }
     }
if (GPIO_Pin == Alarm_detector) //activación de alarma si hay movimiento en la entrada
         {
            if (alarm == 0) {
                    //HAL_DELAY(120); //2 minutospara poder desactivar la alarma;
                    alarm = 1;
            }
         }
```

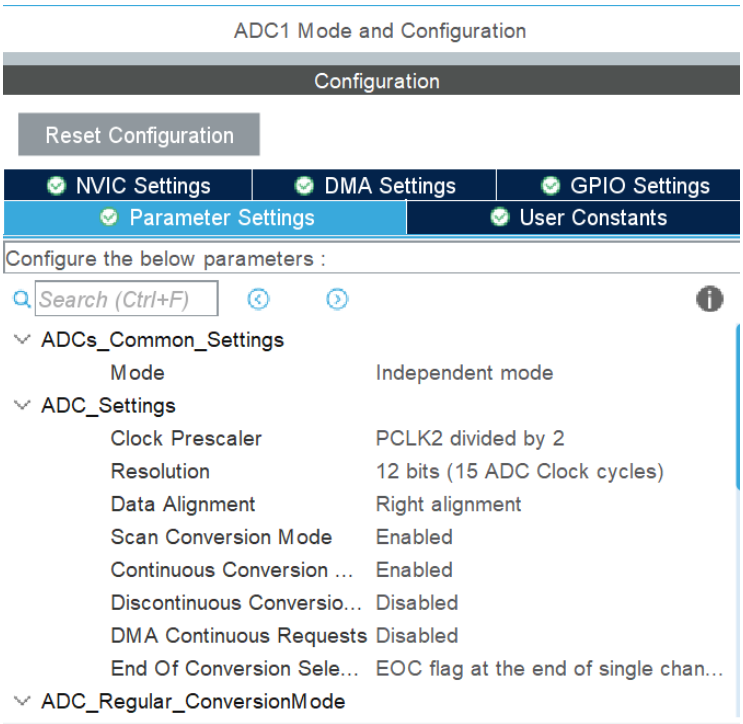
2. LECTURA DE ENTRADAS ANALÓGICAS

   Al usar varias entradas analógicas, se necesita asociar cada una de ellas con los distintos valores de *hadc*

   Es importante recalcar que IN0 es la entrada asociada a la lectura del sensor de temperatura interno y para poder habilitar este modo, es importante seleccionarlo también:

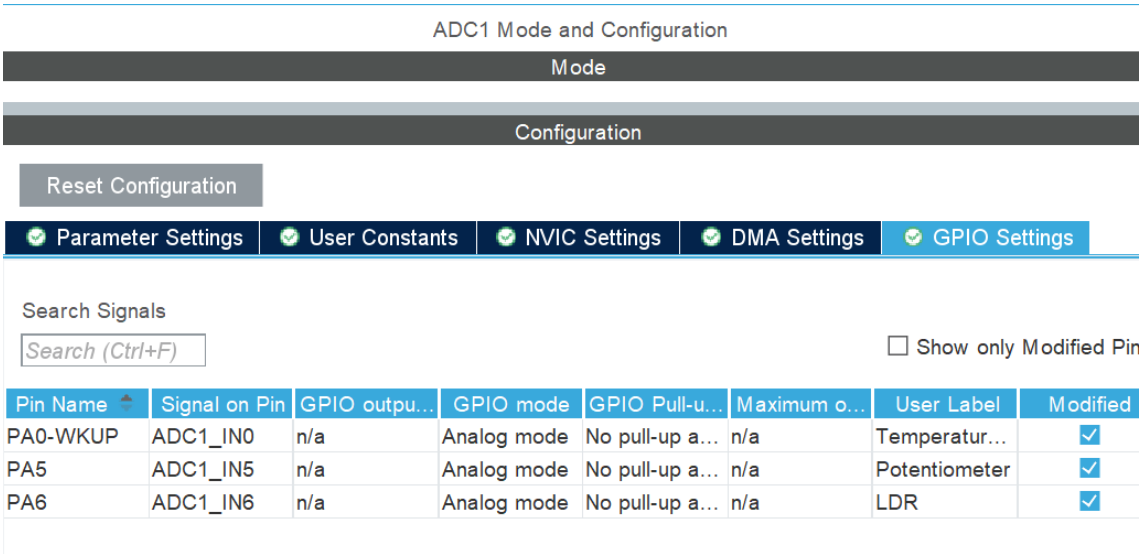


   También, como se usarán varios *hadc,* en *parameters setting* del ADC1 hay que habilitar el modo de funcionamiento continuo, especificar que se harán 3 conversiones y ajustar cada *hadc* para que concuerden *rank* y canal. Para el diseño *hadc1* corresponde al Sensor de Temperatura, *hadc2* a PA5 (el potenciómetro) y *hadc3* a PA6 (el LDR).



Number Of Conversion	3
External Trigger Convers...	Regular Conversion launched by so...
External Trigger Convers...	None
Rank	1
Channel	Channel Temperature Sensor
Sampling Time	480 Cycles
Rank	2
Channel	Channel 6
Sampling Time	3 Cycles
Rank	3
Channel	Channel 5
Sampling Time	3 Cycles

Finalmente, se observa que obtenemos un GPIO de los ADC1 satisfactorio:



Pin Name	Signal on Pin	GPIO outpu...	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PA0-WKUP	ADC1_IN0	n/a	Analog mode	No pull-up a...	n/a	Temperatur...	☑
PA5	ADC1_IN5	n/a	Analog mode	No pull-up a...	n/a	Potentiometer	☑
PA6	ADC1_IN6	n/a	Analog mode	No pull-up a...	n/a	LDR	☑

Como se necesitará recurrir a la lectura de valores analógicos varias veces a lo largo del programa, se declaran también en esta parte, para poderlas llamar a posteriori, las variables volátiles asociadas en el *int main* a la lectura de la función tipo asociada a la extracción de datos analógicos, una para la lectura del LDR asociada a la luz exterior, otra para la lectura del potenciómetro que variará la luz interior y una última para el valor de la temperatura.

```
volatile uint16_t room_light_adjust;
volatile uint16_t outside_light_adjust;
volatile uint32_t adc_temp;
```

Como se hace necesario la toma de valores de distintas fuentes de tipo analógico para su posterior conversión, se declaran varias funciones que, en cada llamada a ellas, cambian el canal al que se necesita activar, es por esta razón que sólo se usa una variable *hacd1* y no tres, porque siempre le pasaremos a las funciones de la familia *HAL_ADC* el mismo puntero. Se desarrollan 3 funciones tipo *void*, una por cada elemento analógico. Se observa que el canal de el sensor interno de temperatura es propio del sistema.

```
void ADC_Select_LDR (void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_6;
	sConfig.Rank = 1;
	sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
	  Error_Handler();
	}
}
void ADC_Select_Potentiometer (void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_5;
	sConfig.Rank = 1;
	sConfig.SamplingTime = ADC_SAMPLETIME_84CYCLES;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
	  Error_Handler();
	}
}
void ADC_Select_CHTemp (void)
{
	ADC_ChannelConfTypeDef sConfig = {0};
	sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
	sConfig.Rank = 1;
	sConfig.SamplingTime = ADC_SAMPLETIME_112CYCLES;
	if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
	{
	  Error_Handler();
	}
}
```

3. LECTURA DEL VALOR DE LA TEMPERATURA

   Se realizó una función para la lectura de la temperatura. Para ello se declara variable temperatura de tipo *float*. Esta función ajusta los valores de cambios de voltaje que arroja el sensor de temperatura y los convierte en un valor en grados. La variable *real_temp* es la que usaremos de modo final en el *int main*:

```
float temp;
#define k 0.76
#define slope 0.0025
#define voltage_conv 3.3/4096
float real_temp;
float temp_value (uint32_t adc_temp_value)
{
        adc_temp_value = adc_temp_value - 130;
        temp = (float)adc_temp_value * voltage_conv;
        temp = (temp-k);
        temp= temp/slope;
        temp = temp + 25;
        return temp;
}
```

4. ACCIÓN DEL TIMER

La configuración del timer, alojada en TIM2, se ajustó para que con un *prescaler* de 1600, el periodo sean milisegundos naturales, por lo que nuestra luz durará encendida 2 segundos.

Se usa la función *HAL_TIM_PeriodElapsedCallback* para esperar a que el timer llegue a el tiempo de espera para el que está diseñado y después apague el Led

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    UNUSED(htim);
    HAL_GPIO_WritePin(GPIOD,Hall,GPIO_PIN_RESET);
}
```

b. En int main

Primeramente, se inicializan todas las configuraciones de periféricos que se usarán así como se dejan disponibles para su utilización todas las entradas analógicas que luego se pretenden leer y convertir. Haremos lo mismo con el timer que regula las luces de la habitación controladas por el potenciómetro puesto que esta acción (la de regulación de luz) no tiene ningún evento propio que la dispare.

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_ADC1_Init();
MX_TIM3_Init();
MX_TIM10_Init();
MX_TIM1_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */
HAL_ADC_Start(&Temp_reader);
// HAL_ADC_Start(&LDR);
// HAL_ADC_Start(&Potentiometer);
HAL_TIM_PWM_Start(&Lights,Room_light);
```

5. CONTROL DE LUCES Y TEMPERATURA (Modo automático)

En el interior del bucle *while* es donde se desarrolla el resto del programa. Primeramente ,se resetea la luz del led de la alarma para evitar desajustes. Ajuste en modo automático activado/desactivado

Cuando este modo se encuentra activado, el *Callback* de la interrupción devuelve la variable *automatic_mode*. Cuando el programa la detecta lo primero activa el piloto (led) indicador que el modo se encuentra activado.

```
while (1)
{
  /* USER CODE END WHILE */
    HAL_GPIO_WritePin(GPIOD,Alarm_light,GPIO_PIN_RESET);
        if(automatic_mode) //si estoy en modo automático
            {
                HAL_GPIO_WritePin(GPIOD,Automatic_mode_light,GPIO_PIN_SET); //luz que indica modo automático
```

Cuando si el modo automático está desactivado, simplemente se apaga el led indicador de modo.

```
else //si estoy en modo manual
{
    HAL_GPIO_WritePin(GPIOD,Automatic_mode_light,GPIO_PIN_RESET); //apago luz que indica modo automático
    HAL_GPIO_WritePin(GPIOD,Hall,sw); //copio en la luz el estado del pulsador
}
```

Siguiendo con las acciones dentro del modo automático:

- Si detecta movimiento en el recibidor (interior de la vivienda) *Callback* pone a 1 la variable *motion*. En ese caso, se enciende la luz del recibidor y se dispara el temporizador que es el gobierna el apagado de la luz con la función *HAL_TIM_Base_Start_IT*.

```
if(automatic_mode) //si estoy en modo automático
    {
        HAL_GPIO_WritePin(GPIOD,Automatic_mode_light,GPIO_PIN_SET); //luz que indica modo automático
        if(motion)//si detecto movimiento en la entrada
        {
            HAL_GPIO_WritePin(GPIOD,Hall,GPIO_PIN_SET);
            HAL_TIM_Base_Start_IT(&Tempor);
        }
```

- En el modo automático, se activa también la regulación del led exterior. Para ello se usa la función antes definida para cambiar de canal al del LDR (*ADC_Select_LDR*), que es el elemento que supedita el sistema, y se habilita para su uso mediante *HAL_ADC_Start.* También se habilita para usarse el led asociado a los cambios de lumnosidad que es de tipo PWM con la función *HAL_TIM_PWM_Start.*
  Para la conversión de datos se procederá ahora y en el resto del cógido de la misma manera (incluyendo la selección del canal y la habilitación mediante *HAL_ADC_Start)*, por lo que se desarrollará esta única vez y se dará por supuesta su funcionamiento en el resto de esta memoria. La Para la conversión, se hace necesaria una variable de tipo *hadc,* esperando mediante el uso de un condicional a que haga la conversión, recogiendo el valor en una variable que se llamó *outside_light_adjust* en este caso, que será la que devuelva este proceso.
  La sentencia.*HAL_ADC_PollForConversion(&hadc,HAL_MAX_DELAY)* lo que hace es esperar durante un tiempo máximo (*HAL_MAX_DELAY)* a que, como ya se adelantaba, la lectura del valor analógico de *hadc* se cumpla, cuando esto pasa, devuelve *HAL_OK*. De ahí el desarrollo del if. *HAL_ADC_GetValue* toma el valor del *hadc.* Después, se detiene el uso del ADC mediante *HAL_ADC_Stop* para poder habilitarla de nuevo con distintos periféricos analógicos previo cambio de canal.

  Si este valor obtenido se encuentra por debajo de un umbral (30 para el caso), implementamos en el PWM el valor obtenido (mediante la función *__HAL_TIM_SET_COMPARE*) restando 50 que es el valor máximo que devolvía el LDR usado cuando incidía en él el flash de un móvil. Esto es porque la relación entre luminosidad del led y luminosidad ambiente es inversa, es la forma de conseguir que a menos luz, se encienda el led de forma más intensa.
  Si el valor está por encima del umbral, apagaremos el led PWM mediante la función *__HAL_TIM_SET_COMPARE* pasándole como valor un cero.

```
//Ajuste automático de la luz exterior
ADC_Select_LDR();
HAL_ADC_Start(&hadc1);
HAL_TIM_PWM_Start(&Lights,Outside_light);
//HAL_ADCEx_Calibration_Start(&hadc2);
  if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY)==HAL_OK)
              {
              outside_light_adjust = HAL_ADC_GetValue(&hadc1);
              }
HAL_ADC_Stop(&hadc1);
if(outside_light_adjust<=30)
{
  __HAL_TIM_SET_COMPARE(&Lights, Outside_light, outside_light_adjust-50);
    //TIM3->CCR1=outside_light_adjust-255;
   //HAL_DELAY(1);
}
else __HAL_TIM_SET_COMPARE(&Lights, Outside_light, 0);
```

- El control de temperatura también se activa mientras se esté usando el modo automático. Para ello, se cambia al canal de temperatura, habilitando el ADC después y se obtiene el valor tras la conversión analógico digital que en código se nombra como *adc_temp.* Después se convierte este valor obtenido de voltios a grados con la ya explicada función *temp_value* y se desactiva el ADC. Este último valor obtenido almacenado en *real_temp,* será el valor real final. Si la temperatura está por debajo de 25 grados, se activa la calefacción mediante *HAL_GPIO_WritePin*. En caso contrario, se desactiva

```
// CONTROL DE TEMPERATURA
ADC_Select_CHTemp();
HAL_ADC_Start(&hadc1);
if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY)==HAL_OK)
  {
  adc_temp = HAL_ADC_GetValue(&hadc1);
  }
//adc_temp = adc_value(&Temp_reader);
real_temp = temp_value(adc_temp);
HAL_ADC_Stop(&hadc1);

if(real_temp<25)
    {
        HAL_GPIO_WritePin(GPIOD,Heating,GPIO_PIN_SET);
    }
else{ HAL_GPIO_WritePin(GPIOD,Heating,GPIO_PIN_RESET);} ///if temperatura debajo umbral, enciendo led
```

6. ALARMA

Si el modo alarma se encuentra activada y salta la alarma, se enciende un led y el valor del Buzzer empieza a variar para variar su frecuencia. Pasado un lapso de tiempo (cuando termina el bucle for que cambia el valor que se imprime en el Buzzer), se apagará para evitar posibles molestias a vecinos. También, si se presiona el botón de selección de modo alarma, el buzzer se apaga.

```
if(alarm_mode && alarm)
    {
        HAL_GPIO_WritePin(GPIOD,Alarm_light,GPIO_PIN_SET); //enciendo la luz de alarma
        int i = 0;
        HAL_TIM_PWM_Start(&Buzzer,Buzzer_Channel);
        while(i<255)
        {
          __HAL_TIM_SET_COMPARE(&Buzzer, Buzzer_Channel, i);
            i += 20;
            //HAL_DELAY(500);
        }
      __HAL_TIM_SET_COMPARE(&Buzzer, Buzzer_Channel, 0);
     }

if(alarm_mode==0)
{
    HAL_TIM_PWM_Start(&Buzzer,Buzzer_Channel);
    __HAL_TIM_SET_COMPARE(&Buzzer, Buzzer_Channel, 0);
}
```

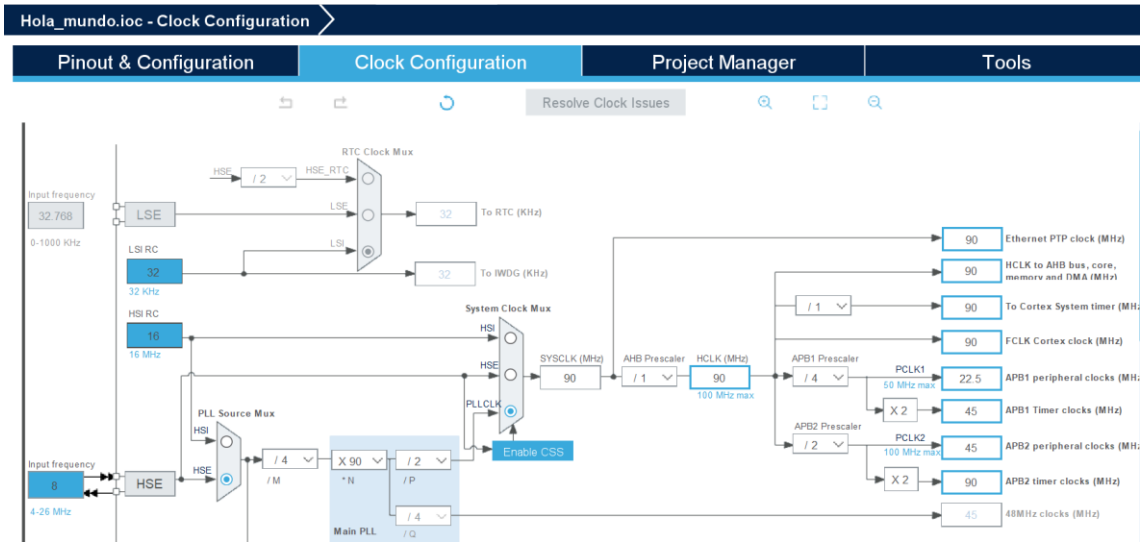
7. AJUSTE DEL REGULADOR DE LUZ

Se selecciona el canal del potenciómetro y se habilita su lectura, extrayendo los valores de la conversión analógico-digital almacenándolos en una variable llamada *room_light_adjust.* No es necesario inicializar el TIM asociada a la luz porque como es un sistema siempre activo, se hizo anteriormente del bucle *while* como se explicó anteriormente. Se implementa el valor obtenido en el led con la función *__HAL_TIM_SET_COMPARE* y se detiene el ADC.:

```
//Ajuste de luz del regulador de luz del salón
ADC_Select_Potentiometer();
HAL_ADC_Start(&hadc1);
if (HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY)==HAL_OK)
          {
          room_light_adjust = HAL_ADC_GetValue(&hadc1);
          }
//room_light_adjust = adc_value(&hadc1);
//TIM3->CCR1=room_light_adjust;
__HAL_TIM_SET_COMPARE(&Lights, Room_light, room_light_adjust+10);
HAL_ADC_Stop(&hadc1);
```

E. Diagrama de tiempos

La configuración de tiempos usada se muestra en la figura



F. Bibliografía

Se usó como base del programa el material alojado en Moodle (tanto de transparencias como el repositorio disponible de bitbuck)