PROYECTO DE MICROS

CONTROL DE LUCES DE UNA VIVIENDA

Sergio de Paula Moratilla 55209

Carlos Perchín García 55401

Rubén Ruiz Martínez 54935

ÍNDICE

1 OBJETIVO DEL PROYECTO

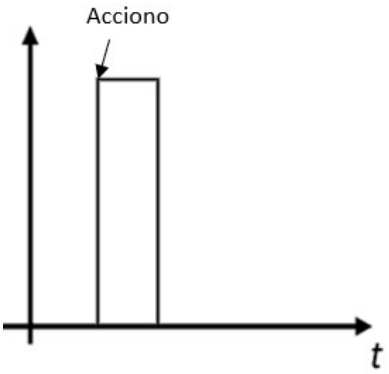
El objetivo del proyecto es el de realizar el control de luces de una vivienda. Van a existir 2 modos de funcionamiento:

- **MANUAL**: Controlado mediante un interruptor. Será nuestro modo predominante, es decir, cuando el pulsador este activo, la luz (representada para la simulación como un LED) siempre permanecerá encendida, independientemente del nivel de luz existente.
- **AUTOMÁTICO**: Controlado mediante una fotorresistencia (nos permitirá determinar el nivel de luminosidad de la habitación) y un ultrasonido (mediante el cual detectaremos la entrada de una persona). En el caso de encenderse de forma automática, permanecerá en ese estado durante 5 segundos para luego apagarse.

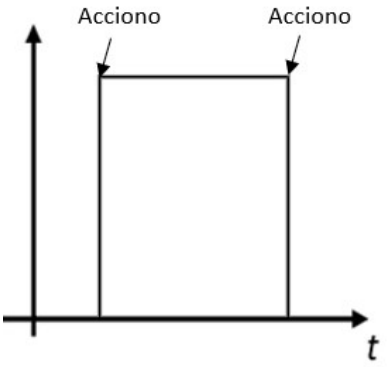
2 INTERRUPTOR MANUAL

Como hemos mencionado, existirá la posibilidad de encender y apagar la luz manualmente. Para ello, haremos uso de un interruptor como el de la imagen:

Antes de continuar, conviene observar la diferencia entre pulsador e interruptor. Dibujamos la forma de onda generada por cada uno de ellos:



PULSADOR



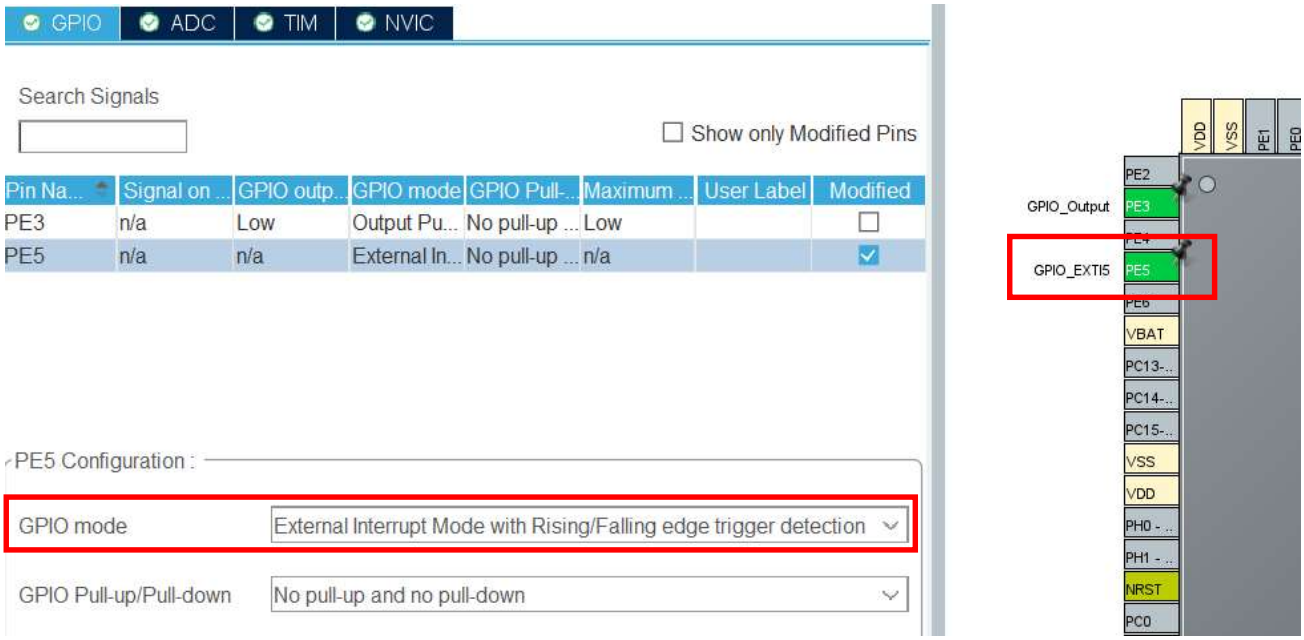
INTERRUPTOR

Por tanto, como veremos más adelante, tendremos que hacer que el micro sea capaz de detectar ambos flancos, tanto el de subida como el de bajada.

2.1 INICIALIZACIÓN

Antes de proceder con la programación, tenemos que inicializar el interruptor. Corresponde con una entrada GPIO, por lo que seguiremos una serie de pasos:

1) Para liberar de tareas el micro, usaremos interrupciones. Activamos la línea de interrupción EXTI5, usando el pin PE5 de la placa.



Como se puede observar, usamos el modo "Rising/Falling edge trigger detection"

2) Habilitamos la interrupción.

2.2 PROGRAMACIÓN

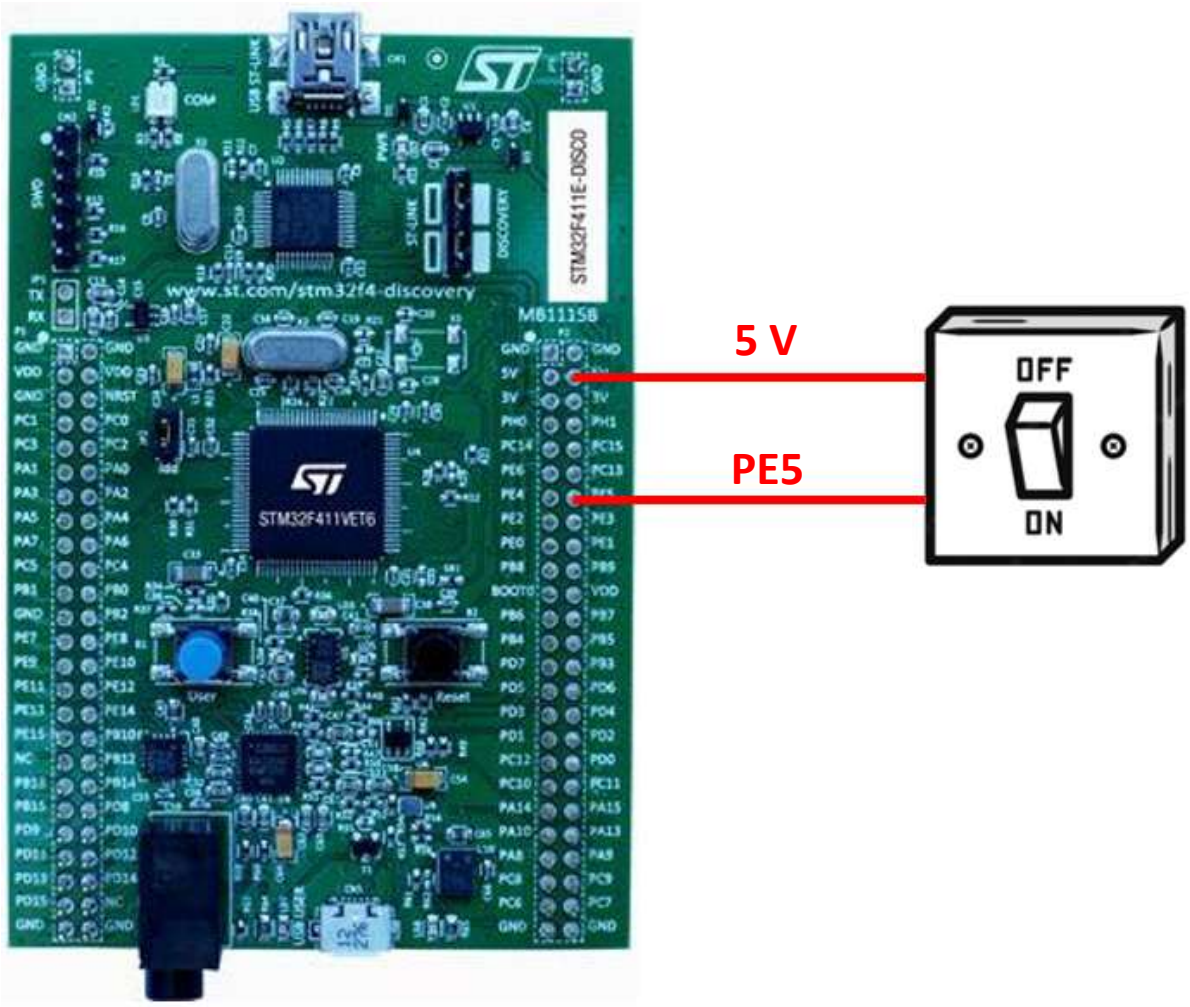
Una vez inicializado, el código introducido ha sido:

```
int interruptor = 0; // para controlar el estado del interruptor

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if(GPIO_Pin == GPIO_PIN_5){
        if(interruptor == 0){
            interruptor = 1;
            sensor_luz = 0; // forzamos que la luz no se pueda encender por presencia
            HAL_ADC_Stop_DMA(&hadc1); // paramos la medida de la luminosidad
            HAL_TIM_IC_Stop_IT(&htim3, TIM_CHANNEL_1); // paramos la medida de la presencia
        }
        else{
            interruptor = 0;
            HAL_ADC_Start_DMA(&hadc1, &luminosidad, 1); // arrancamos la medida de la luminosidad
            HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1); // arrancamos la medida de la presencia
        }
    }
}
```

Como se puede observar, cada vez que pulsemos el interruptor, la variable correspondiente cambiará su valor, alternando entre 0 y 1. Esto cuadra con el funcionamiento explicado anteriormente.

2.3 CONEXIÓN EN LA PLACA

Para poder simular su comportamiento en la placa, se han realizado las siguientes conexiones:



3 DETECCIÓN DEL NIVEL DE LUZ

El encendido automático de la luz solamente estará habilitado cuando el nivel de luminosidad existente en la habitación sea inferior a un cierto umbral. Para poder medir la luminosidad, hemos utilizado una LDR, cuya resistencia varía con la radiación luminosa.

Como la magnitud a medir es la tensión en la resistencia, y es una variable analógica, hemos tenido que utilizar un convertidor A/D.

3.1 INICIALIZACIÓN

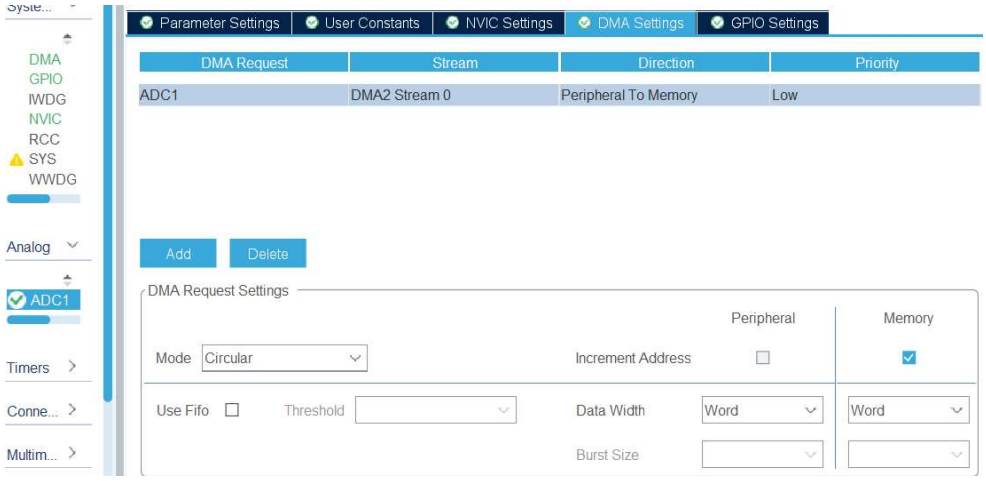
Antes de proceder con la programación, tenemos que inicializar el convertidor. Para ello, seguimos una serie de pasos:

1) Activamos la línea de conversión IN1, correspondiente al pin PA1 de la placa.

2) Configuramos los parámetros del convertidor. Para mayor rapidez, hemos hecho uso del DMA. También hemos reducido el *Sampling Time*, pues no necesitamos que el convertidor actúe tan rápido.

ADCs_Common_Settings	
Mode	Independent mode
ADC_Settings	
Clock Prescaler	PCLK2 divided by 2
Resolution	8 bits (11 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Disabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Selection	EOC flag at the end of single channel conversion
ADC_Regular_ConversionMode	
Number Of Conversion	1
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 1
Sampling Time	480 Cycles

3) Configuramos los parámetros del DMA.



4) Habilitamos la interrupción

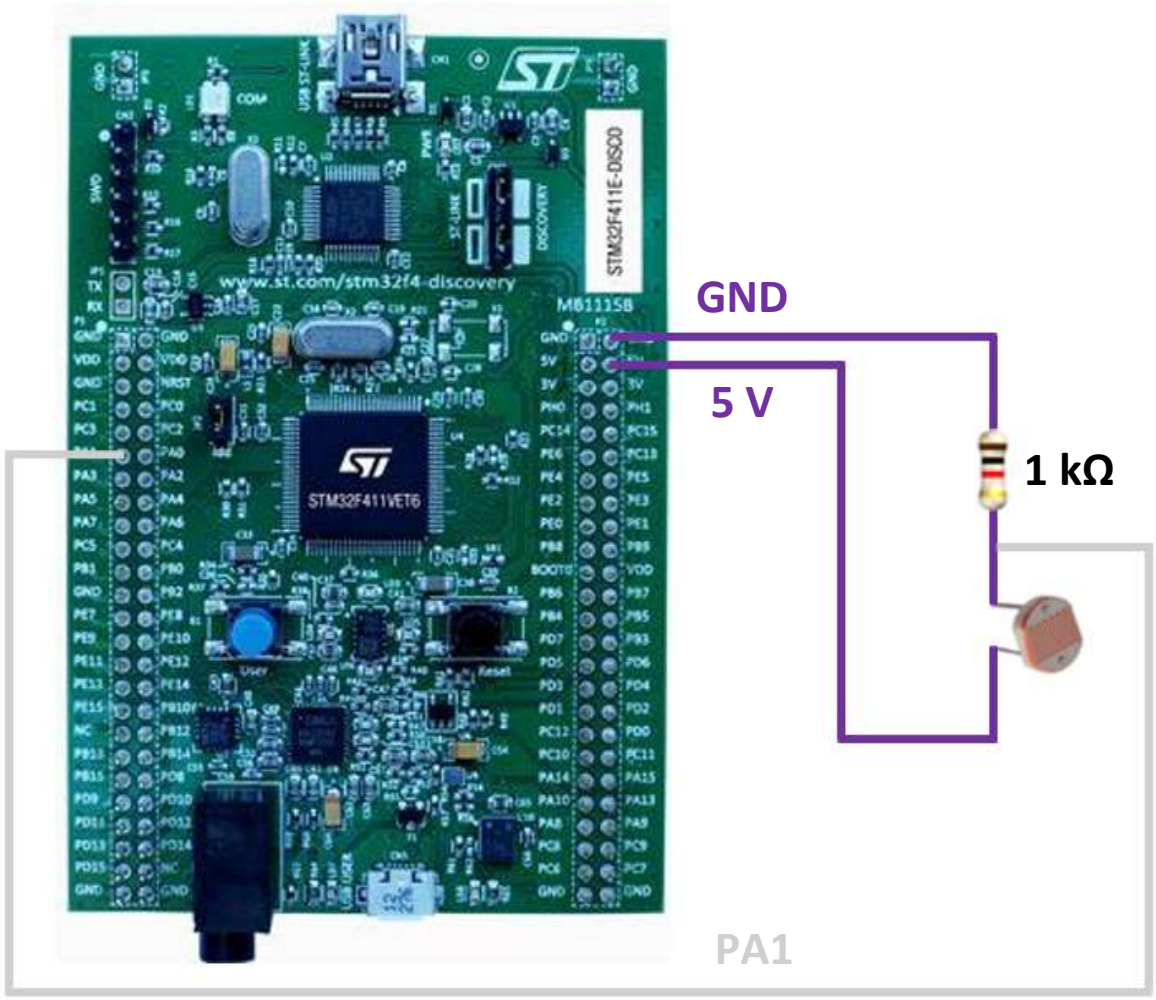
3.2 PROGRAMACIÓN

Una vez inicializado, el código introducido ha sido:

```
uint32_t luminosidad = 0; // luminosidad detectada por la LDR
int sensor_luz = 0; // la luz solo puede encenderse por presencia cuando haya poca luz ambiente

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc){
    if(hadc->Instance == ADC1){
        // Como el DMA lo hace por nosotros, no hace falta coger el valor devuelto por el convertidor
        if(luminosidad < 50 && sensor_luz == 0){
            sensor_luz = 1; // si no hay luz ambiente, podemos encender el LED
            presencia_detectada = 0;
        }
        else if(luminosidad > 50 && sensor_luz == 1){
            sensor_luz = 0; // si hay luz ambiente, no podemos encender el LED
            presencia_detectada = 0; // evitamos que la luz se encienda
        }
        HAL_ADC_Start_DMA(&hadc1, &luminosidad, 1); // arrancamos de nuevo el convertidor
    }
}

int main(void)
{
    // Programas de inicialización
    HAL_ADC_Start_DMA(&hadc1, &luminosidad, 1);

    while (1)
    {
        // Código del while, se explicará mas adelante
    }
}
```

3.3 CONEXIÓN EN LA PLACA

Para poder simular su comportamiento en la placa, se han realizado las siguientes conexiones:



4 DETECCIÓN DE PRESENCIA

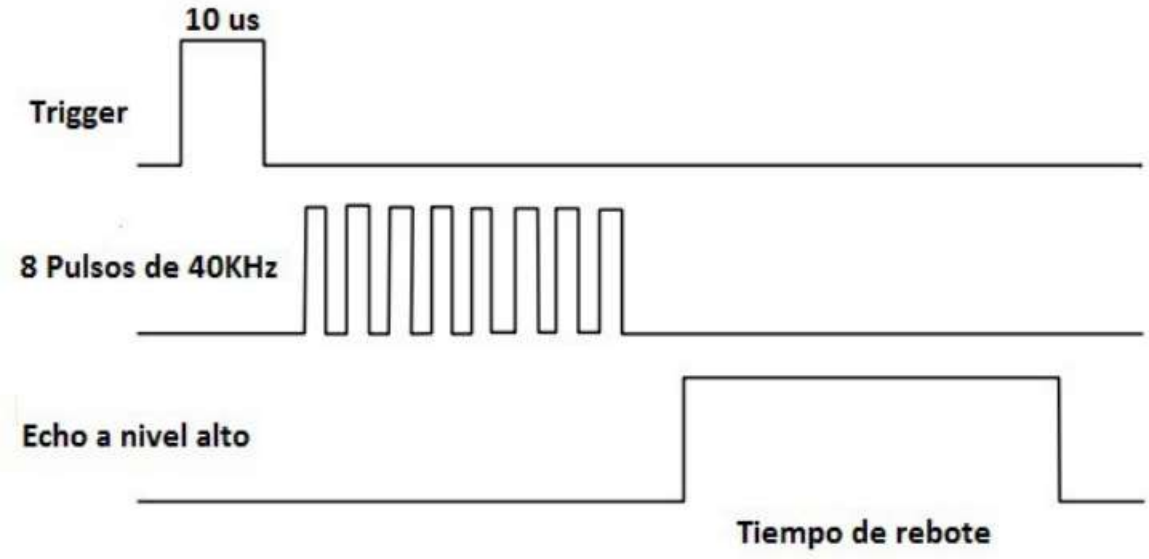
Para que la luz se encienda de forma automática, no solo se debe cumplir que haya baja luminosidad, sino también debe detectarse la entrada de una persona en la habitación. Para ello hemos usado un ultrasonido colocado encima de la puerta de entrada a la sala.

4.1 FUNCIONAMIENTO DEL ULTRASONIDO

En primer lugar, vamos a explicar como funciona el ultrasonido seleccionado. Consta de 4 patillas:

- ***Vcc***: Pin de alimentación (5V).
- ***Trigger***: Pin de disparo. Este pin es una entrada.
- ***Echo***: Pin de salida del sensor.
- ***Gnd***: Pin negativo de alimentación.

El diagrama de temporización del ultrasonido es:



Se aprecia como solo es necesario aplicar un pulso, de 10 µs, en el *Trigger* para comenzar con la medición.

Para implementar este pulso, haremos uso de un temporizador PWM de periodo 1 ms y ciclo 10 µs

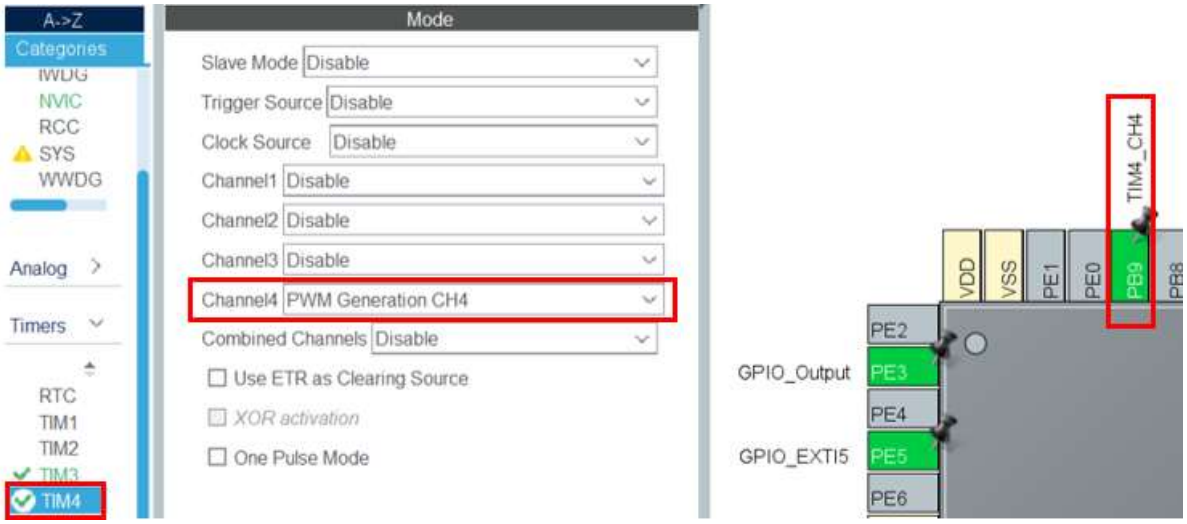
A continuación, el sensor envía una serie de 8 pulsos de 40KHz y pone *Echo* a nivel alto. Permanecerá a nivel alto hasta que se reciba el eco de los pulsos de 40KHz. Por lo tanto, para saber a la distancia a la que se encuentra el objeto, solo hay que medir el tiempo durante el cual *Echo* está a nivel alto.

Para medir este tiempo, utilizaremos un temporizador Input Capture

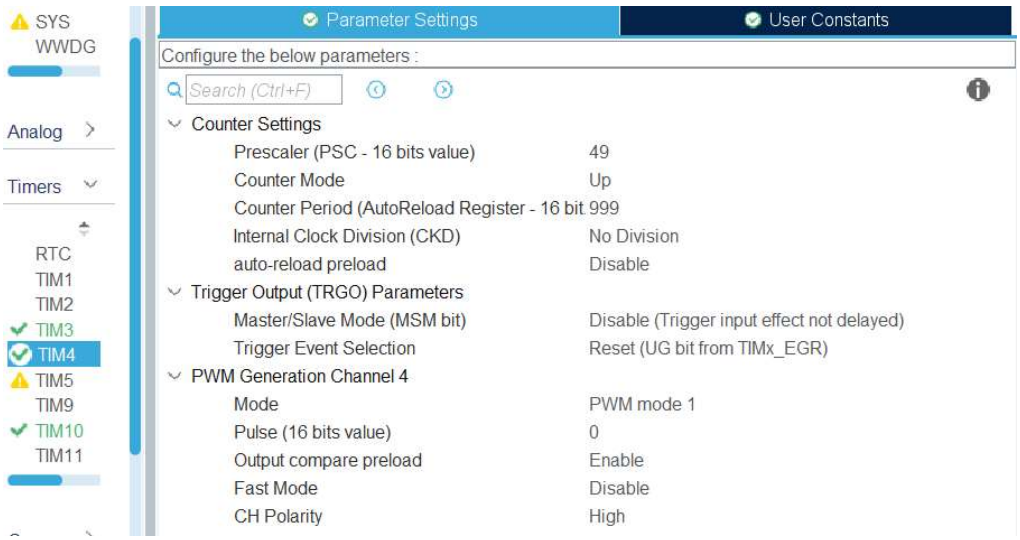
4.2 TEMPORIZADOR PWM

Como hemos dicho, para implementar la entrada *Trigger*, vamos a utilizar un temporizador PWM. Lo primero que debemos que hacer es inicializarlo, para lo que seguimos una serie de pasos:

1) Activamos el canal 4 del temporizador TIM4 en modo PWM, correspondiente al pin PB9 de la placa.



2) Configuramos los parámetros del temporizador. Como queremos conseguir un periodo de 1 ms, hemos escogido unos valores para el *Prescaler* y *Counter* de:



$$Periodo = \frac{1}{\frac{TIMx\_CLK}{(Prescaler + 1)(Counter + 1)}} = \frac{1}{\frac{50 * 10^6}{(49 + 1)(999 + 1)}} = 0.001\ s = 1\ ms$$

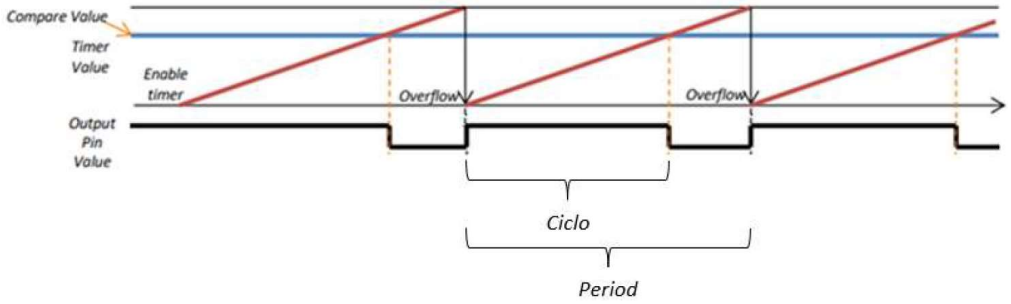
El valor de $TIMx\_CLK$ ha sido obtenido del *datasheet* de la placa:

Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complemen-tary output	Max. interface clock (MHz)	Max. timer clock (MHz)
TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	50	100

Una vez inicializado, el código introducido ha sido:

```
int main(void)
{
    // Funciones de inicialización
    HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 10);

    while (1)
    {
        // Código del while, se explicará más adelante
    }
}
```

Recordando la forma de las señales PWM:

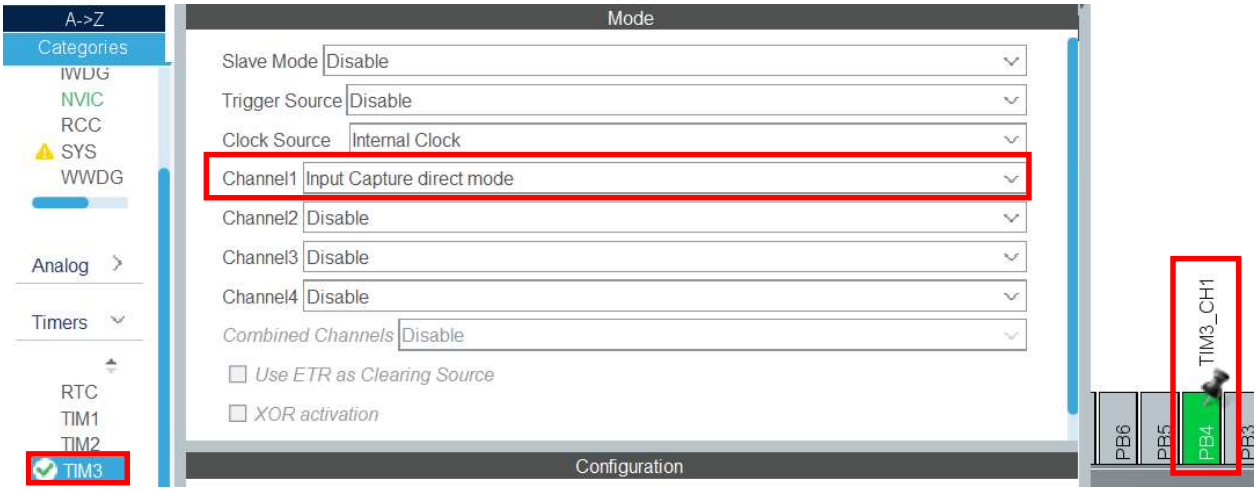


El valor de 10 introducido en la función *__HAL_TIM_SET_COMPARE* se debe a la duración requerida del pulso. Conociendo el valor usado en la inicialización en *Counter*, y sabiendo que cada ms contiene 1000 µs, queremos que la señal valga 1 durante los 10 primeros valojes de contaje.

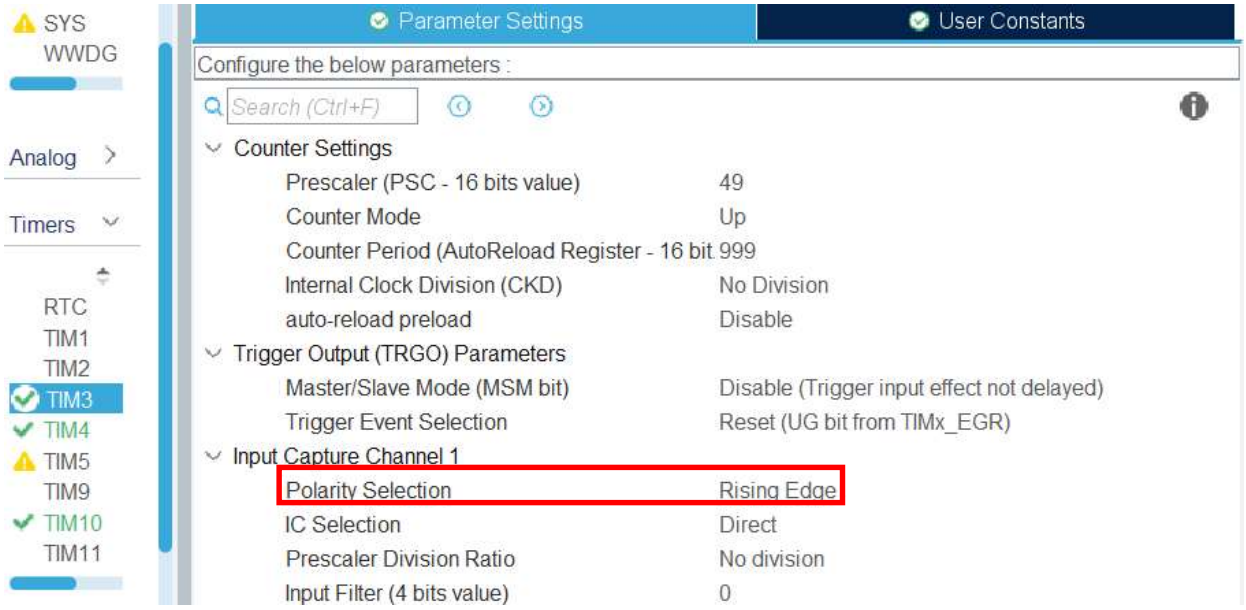
4.3 TEMPORIZADOR INPUT CAPTURE

Como hemos dicho, usaremos este temporizador para medir la duración del pulso originado en la salida *Echo*. Lo primero que debemos que hacer es inicializarlo, para lo que seguimos una serie de pasos:

1) Activamos el canal 1 del temporizador TIM3 en modo *Input Capture*, correspondiente al pin PB4 de la placa.

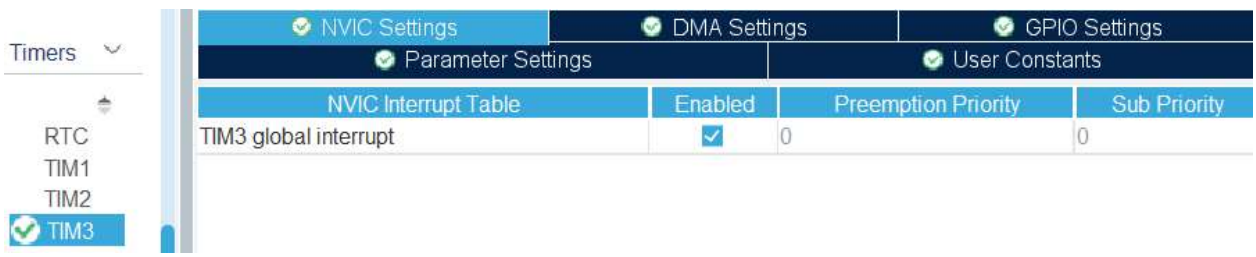


2) Vamos a programarlo mediante interrupciones. En un temporizador *Input Capture* las interrupciones saltan cada vez que detecten un flanco. Inicialmente lo configuraremos para detectar el flanco de subida. Los valores para el *Prescaler* y *Counter* son los mismos que los anteriores:



Como luego tenemos que detectar el flanco de bajada, ese cambio lo realizaremos en el propio código

3) Habilitamos la interrupción

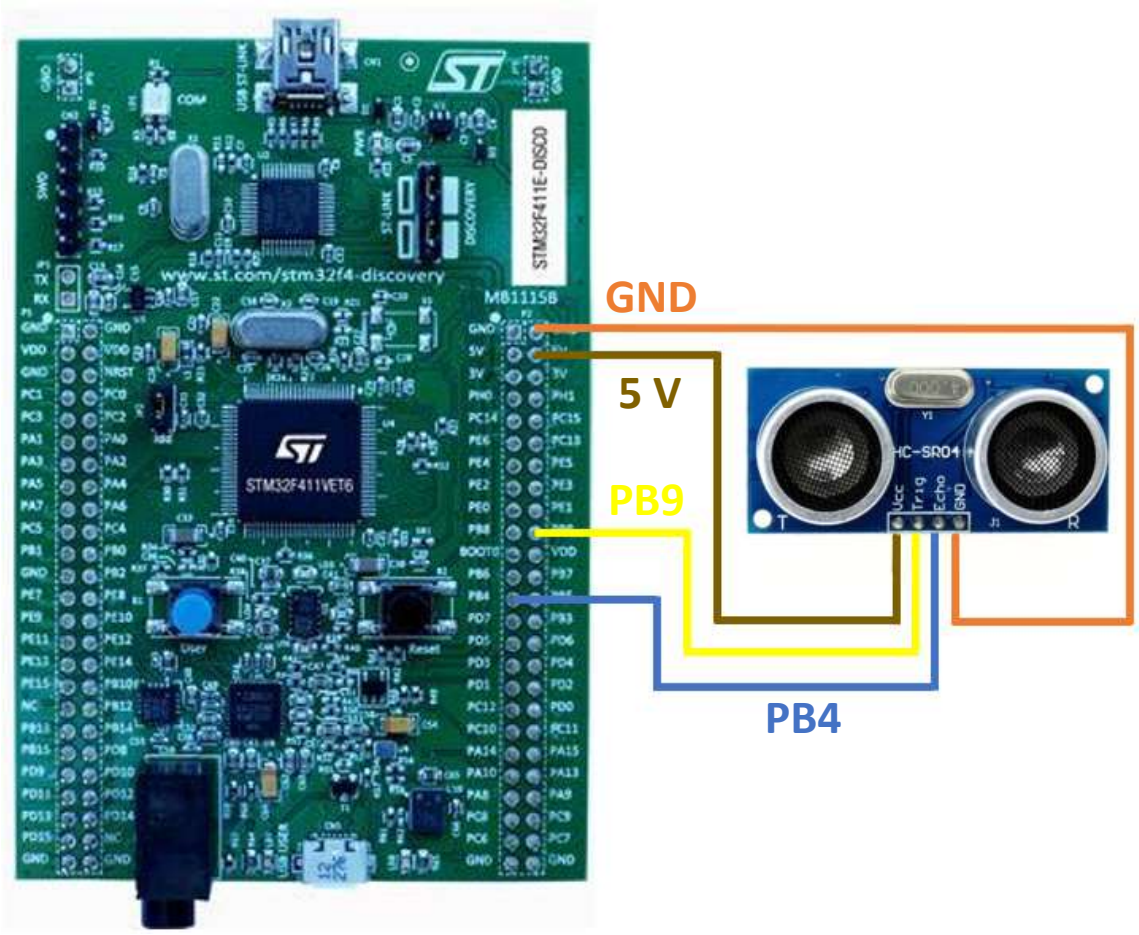


Una vez inicializado, el código introducido ha sido:

```
uint32_t IC_valor_1 = 0; // valor del contador en el flanco de subida
uint32_t IC_valor_2 = 0; // valor del contador en el flanco de bajada
uint32_t distancia = 0; // distancia detectada por el ultrasonido
int presencia_detectada = 0; // la luz se ha encendido por presencia

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance == TIM3){
        if (IC_valor_1 == 0){ // si todavia no se ha producido el flanco de subida
            IC_valor_1 = HAL_TIM_ReadCapturedValue(&htim3, TIM_CHANNEL_1);
            // Cambiamos la sensibilidad de la interrupción al flanco de bajada
            __HAL_TIM_SET_CAPTUREPOLARITY(&htim3, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);
        }
        else if (IC_valor_1 != 0){ // si ya se ha producido el flanco de subida
            IC_valor_2 = HAL_TIM_ReadCapturedValue(&htim3, TIM_CHANNEL_1);

            __HAL_TIM_SET_COUNTER(htim, 0);
            if (IC_valor_2 > IC_valor_1){
                distancia = IC_valor_2 - IC_valor_1;
            }
            else if (IC_valor_1 > IC_valor_2){
                distancia = (0xffff - IC_valor_1) + IC_valor_2;
            }
            if(distancia < 500){
                presencia_detectada = 1;
                HAL_TIM_Base_Start_IT(&htim10); // iniciamos el temporizador de 5 segundos
            }
            IC_valor_1 = 0;
            // Cambiamos la sensibilidad de la interrupción al flanco de subida
            __HAL_TIM_SET_CAPTUREPOLARITY(&htim3, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);
        }
    }
}

int main(void)
{
    // Funciones de inicialización
    HAL_TIM_IC_Start_IT(&htim3, TIM_CHANNEL_1);

    while (1){
        // Código del while, se explicará mas adelante
    }
}
```

4.4 CONEXIÓN EN LA PLACA

Para poder simular su comportamiento en la placa, se han realizado las siguientes conexiones:



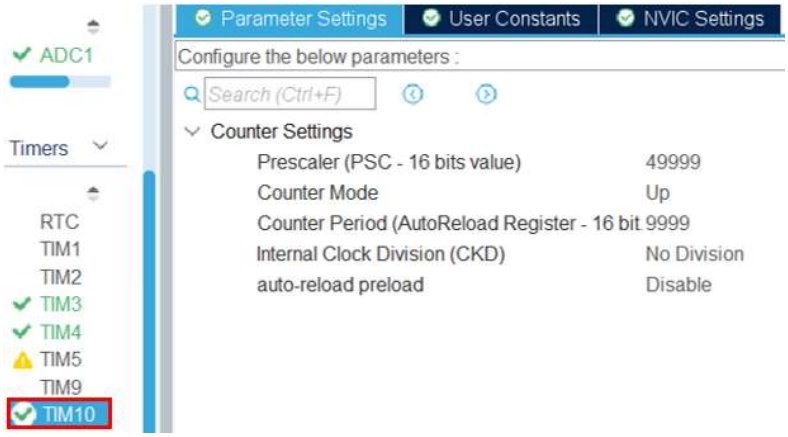
5 ENCENDIDO DE LA LUZ

El encendido de la luz será representado mediante un LED conectado a una salida GPIO de la placa.

5.1 TEMPORIZADOR BÁSICO

Como hemos dicho anteriormente, cuando el LED se active de forma automática, permanecerá encendido durante 5 segundos y luego se apagará. Para contar el tiempo, hemos recurrido a un temporizador básico. Lo primero que debemos que hacer es inicializarlo, para lo que seguimos una serie de pasos:

1) Si miramos el datasheet de la placa, podemos emplear para programar temporizadores básicos, por ejemplo, el TIM10.
2) Configuramos los parámetros del temporizador. Como queremos conseguir un periodo de 5 s, hemos escogido unos valores para el *Prescaler* y *Counter* de:



$$Periodo = \frac{1}{\frac{TIMx\_CLK}{(Prescaler + 1)(Counter + 1)}} = \frac{1}{\frac{100 * 10^6}{(49999 + 1)(9999 + 1)}} = 5\ s$$

El valor de $TIMx\_CLK$ ha sido obtenido del *datasheet* de la placa:

Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complemen-tary output	Max. interface clock (MHz)	Max. timer clock (MHz)
TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	100	100

3) Habilitamos la interrupción

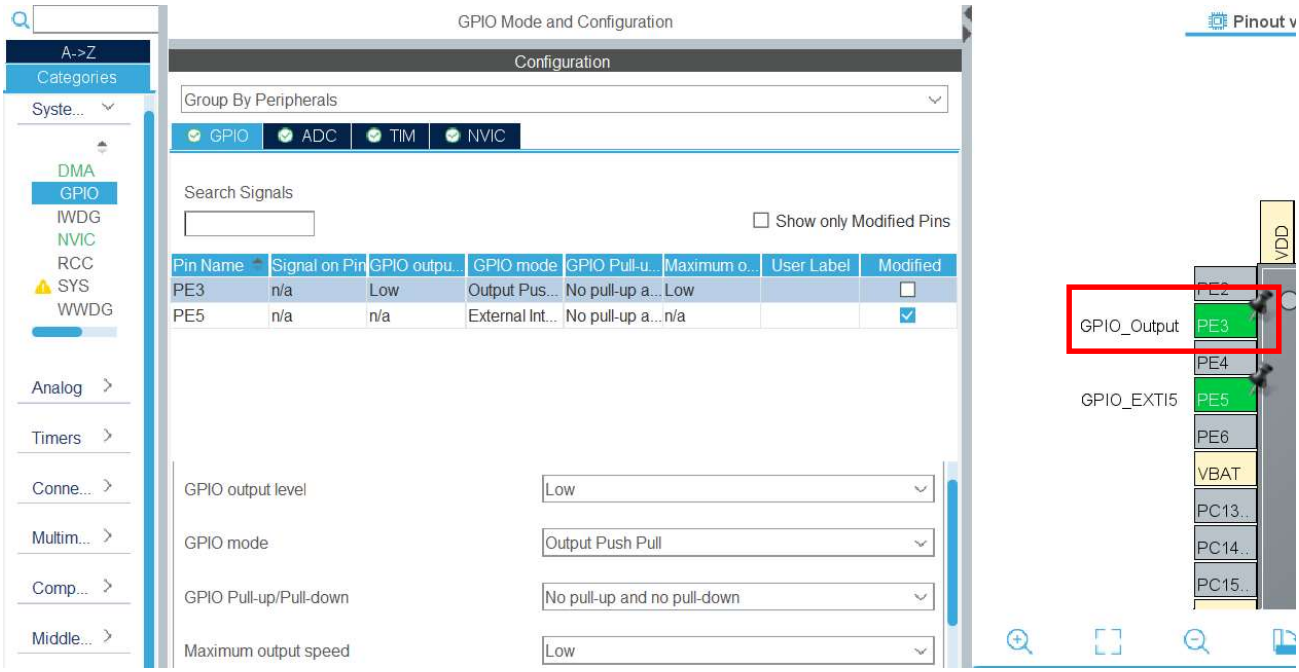
Una vez inicializado, el código introducido ha sido:

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance == TIM3){
        if (IC_valor_1 == 0){ // si todavia no se ha producido el flanco de subida
            IC_valor_1 = HAL_TIM_ReadCapturedValue(&htim3, TIM_CHANNEL_1);
            // Cambiamos la sensibilidad de la interrupción al flanco de bajada
            __HAL_TIM_SET_CAPTUREPOLARITY(&htim3, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);
        }
        else if (IC_valor_1 != 0){ // si ya se ha producido el flanco de subida
            IC_valor_2 = HAL_TIM_ReadCapturedValue(&htim3, TIM_CHANNEL_1);

            __HAL_TIM_SET_COUNTER(htim, 0);
            if (IC_valor_2 > IC_valor_1){
                distancia = IC_valor_2 - IC_valor_1;
            }
            else if (IC_valor_1 > IC_valor_2){
                distancia = (0xffff - IC_valor_1) + IC_valor_2;
            }
            if(distancia < 500){
                presencia_detectada = 1;
                HAL_TIM_Base_Start_IT(&htim10); // iniciamos el temporizador de 5 segundos
            }
            IC_valor_1 = 0;
            // Cambiamos la sensibilidad de la interrupción al flanco de subida
            __HAL_TIM_SET_CAPTUREPOLARITY(&htim3, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);
        }
    }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
    if(htim->Instance == TIM10){
        presencia_detectada = 0;
        HAL_TIM_Base_Stop_IT(&htim10);
    }
}
```

Como se puede observar, el temporizador comienza a contar cuando el ultrasonido detecta presenta. Una vez pasado el tiempo, deja de detectar presencia y el temporizador se desactiva.

5.2 PROGRAMACIÓN DEL LED

Como hemos dicho antes, la luz será representada mediante un LED conectado a una salida GPIO de la placa. Lo primero que debemos que hacer es inicializar el puerto, para lo que hemos usado el pin PE3 de la placa.

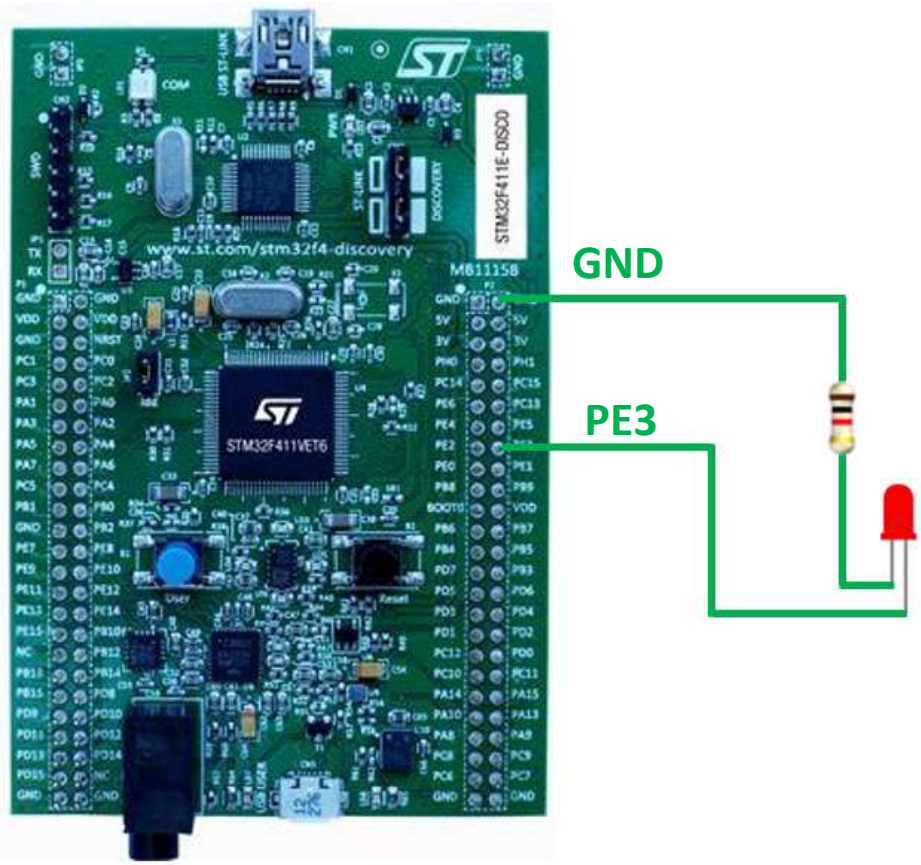


Una vez inicializado, el código introducido ha sido:

```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    // si el interruptor está abierto, encendemos el LED
    if(interruptor == 1)
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
    // si el interruptor está cerrado, no hay luz ambiente y detectamos presencia, encendemos el LED
    else if(interruptor == 0 && sensor_luz == 1 && presencia_detectada == 1)
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
    // en cualquier otro caso, apagamos el LED
    else
        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);
}
```

5.3 CONEXIÓN EN LA PLACA

Para poder simular su comportamiento en la placa, se han realizado las siguientes conexiones:

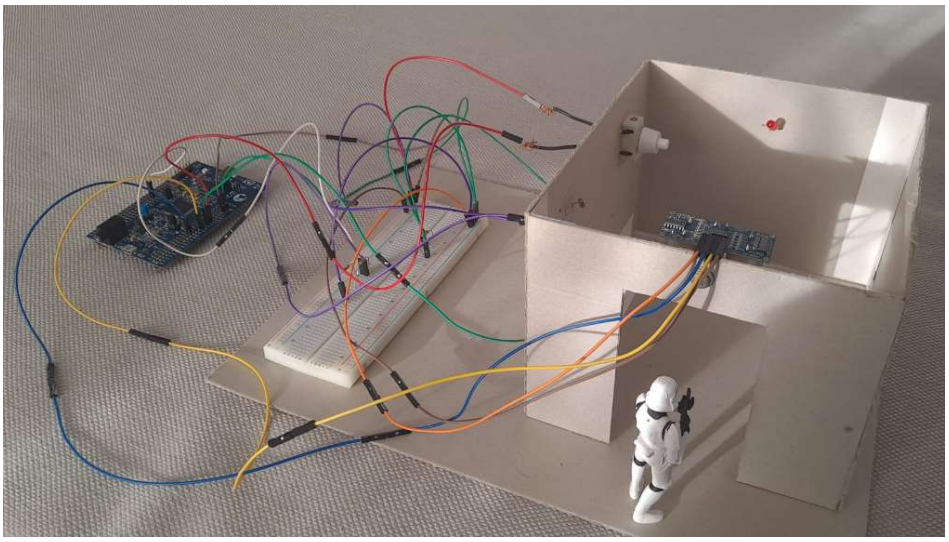


Hay que tener en cuenta que la patilla más larga es el ánodo (+)

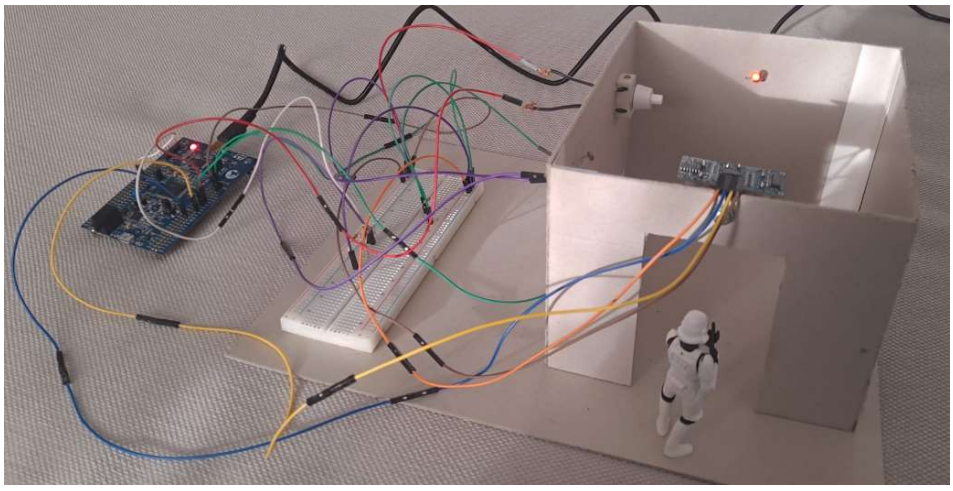
6 COMPROBACIÓN DEL FUNCIONAMIENTO

Para poder comprobar el funcionamiento del sistema, hemos desarrollado una maqueta. Veamos algunas situaciones:

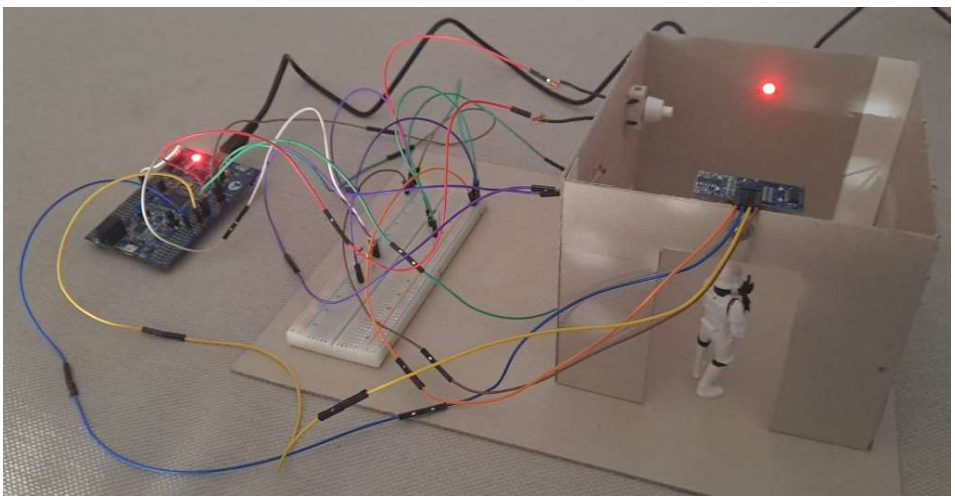
1) Situación inicial



2) Si pulsamos el interruptor, a pesar de no haber nadie en la habitación, se enciende el LED.



3) Si cerramos el interruptor y entramos en la habitación, habiendo baja luminosidad, se enciende el LED.



4) Si, en cambio, entramos en la habitación habiendo un elevado nivel de iluminación, el LED no se enciende.