SISTEMAS ELECTRÓNICOS DIGITALES

MICROCONTROLADORES

CONTROL DE LUZ EN GARAJE

Trabajo realizado por:

Iñigo Castells Castro – 55790

Juan Ignacio Martín Moreno – 55975

Álvaro Ribalda Hernández - 56555

ÍNDICE

1. INTRODUCCIÓN

En este proyecto se desarrolla un sistema de **domótica simulado** aplicado a un **parking**, con el objetivo de mejorar la eficiencia energética y la comodidad del usuario. El sistema está diseñado para gestionar de manera inteligente la iluminación en función de las condiciones del entorno y las actividades que ocurren dentro del parking. A continuación, se describen las principales características del proyecto:

Espacio principal: Una habitación diseñada para permitir la entrada de un coche.

Modos de funcionamiento:

- **Modo manual**:

  Al pulsar un interruptor, se enciende la luz principal del parking.

  Automáticamente, se apaga la luz roja de emergencia para optimizar la iluminación.

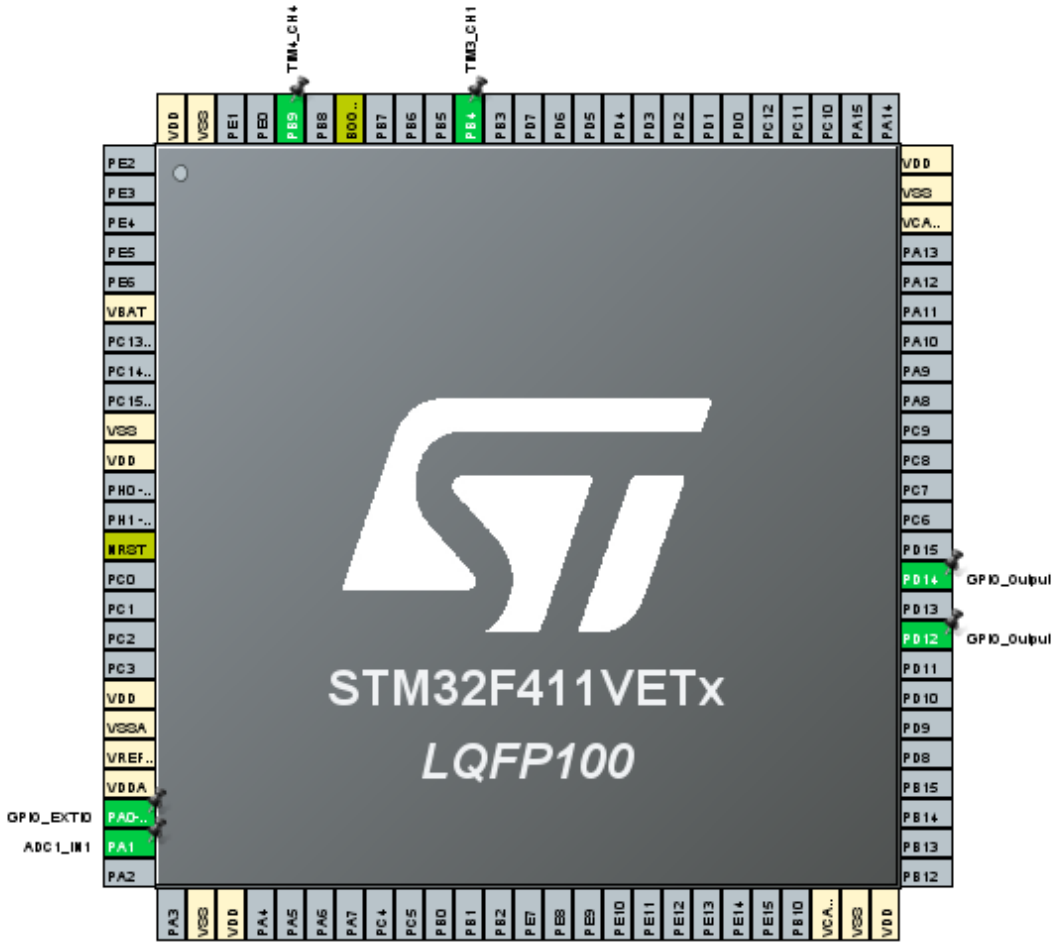
- **Modo automático**:

  El sistema se activa de manera inteligente mediante dos sensores. Un sensor de luz ambiental que detecta condiciones de baja luminosidad, y un sensor de movimiento que identifica la entrada de un coche al parking.

  Si ambas condiciones se cumplen, la luz principal se enciende automáticamente.

- **Objetivos**:
  - Optimizar el uso de energía mediante la activación automática solo cuando sea necesario.
  - Brindar flexibilidad al usuario con la opción de alternar entre modos manual y automático.

2. ELEMENTOS



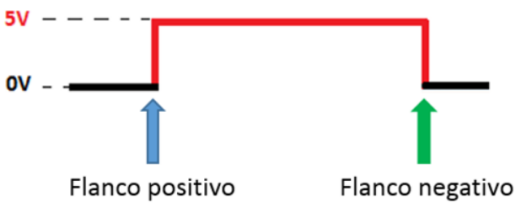
En la imagen se pueden observar los pines de entrada y salida que usaremos en nuestro proyecto.

2.1. ENTRADAS

2.1.1. INTERRUPTOR

El interruptor es necesario para el modo manual, este permitirá apagar o encender la luz manualmente según lo requiera el usuario. Para ello, se usará el siguiente interruptor:

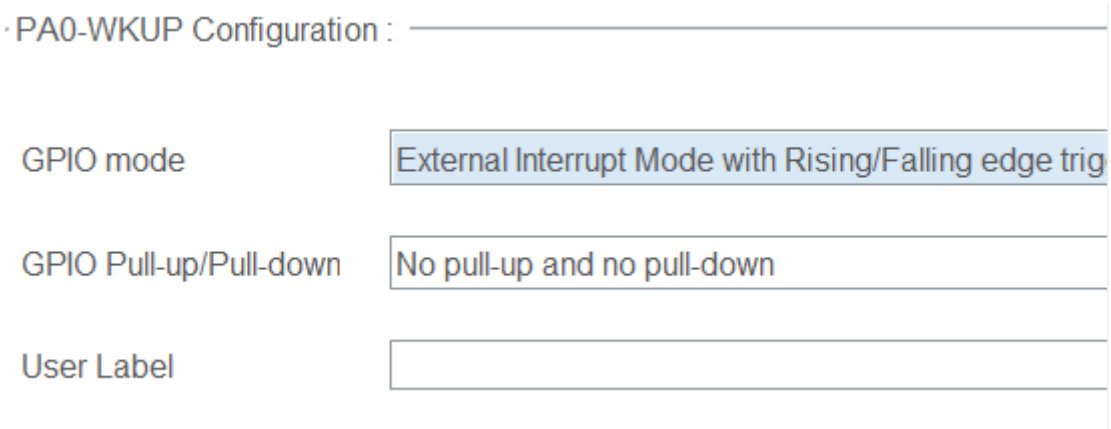
A continuación, se muestra el funcionamiento de un interruptor:



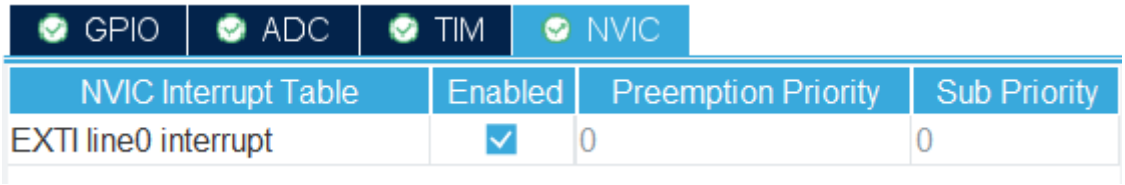
Cuando se detecta el flanco de subida, el interruptor pasará a estar en ON hasta que se detecte el siguiente flanco de bajada, pasando a estar en OFF.

PROGRAMACIÓN VISTA PINES

Para llevar esto a la placa, se configura el pin PA0 como una interrupción, y se configura el GPIO mode para que detecte tanto el flanco de subida como el de bajada, ya que debe de haber una interrupción tanto en la subida como en la bajada de flanco.



Una vez configurado, se habilita la interrupción marcando la casilla en NVIC.



Una vez terminada la configuración, se procederá a realizar el código. Para esta interrupción, necesitaremos una función callback, esta función le dará un valor a la variable global "SW" en función del estado del interruptor.

Será el bucle while(1) del main quien se encargue de activar o desactivar el led en función de estas variables globales.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if(GPIO_Pin == GPIO_PIN_0){
        if(interruptor == 0)
        {
            interruptor = 1;
        }
        else
        {
            interruptor = 0;
        }
    }
}
```

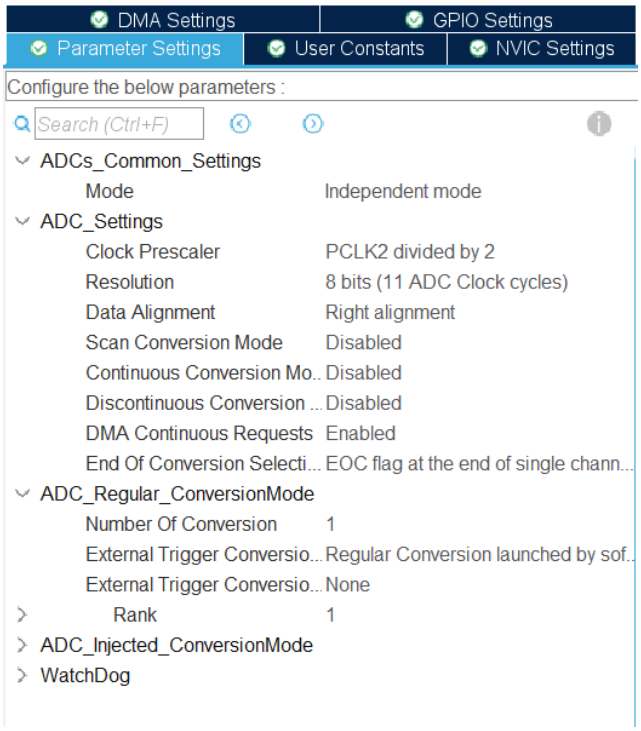
```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    if(interruptor==1)//bucle para encender el led con el interruptor
        {
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
        }else
        {
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        }
      }
```

2.1.2. SENSOR LDR

Este sensor nos permite medir la cantidad de luz que hay en el ambiente, lo que nos permitirá detectar si hace falta que se encienda el led para mayor luminosidad cuando entre algún coche.

Un sensor LDR genera valores analógicos, ya que la variación de su resistencia produce un voltaje continuo que cambia gradualmente según la intensidad de luz. Debido a que la placa funciona con valores digitales, tenemos que usar el convertidor ADC1 que ya tiene la placa integrada.

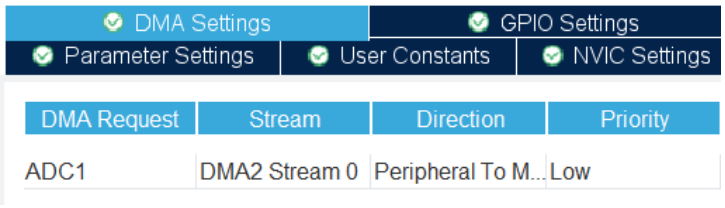
La configuración del convertidor será la siguiente.



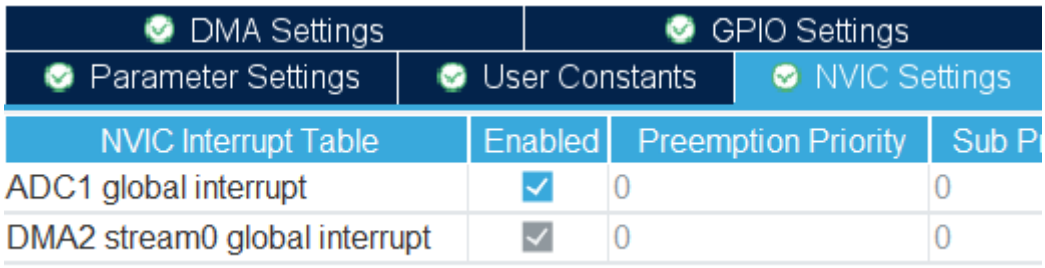
Usaremos una resolución de 8 bits, ya que no necesitamos mucha precisión, haciendo así que el valor obtenido por el LDR esté comprendido entre 0 y 255, siendo lo más cercano al 0 la oscuridad y a 255 el máximo de luz.

Para optimizar la obtención de estos datos, usaremos DMA (Direct Memory Acces). Como se va a necesitar el dato del LDR constantemente, esta es una buena práctica para reducir los tiempos de espera, ya que de esta forma, el ADC1 podrá acceder

directamente a los valores del LDR sin necesidad de un bucle que esté obteniéndolos repetidamente. La configuración para usar DMA será la siguiente.



Ahora, se deberán activar las interrupciones producidas, tanto la del ADC1 para los datos del LDR y la del DMA.



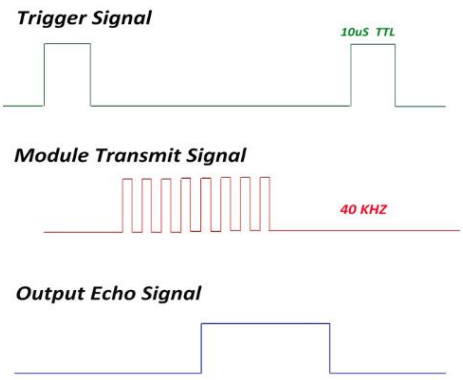
Una vez hecho todo esto, se procederá con el código. De igual forma que con el interruptor, se creará una variable global "sensor_luz" que estará activa o no según las condiciones proporcionadas.

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc){
    if(hadc->Instance == ADC1){
        //como estamos usando DMA, no tenemos que tratar el valor devuelto por el convertidor
        if(luz < 50 && sensor_luz == 0){
            sensor_luz = 1; // si no hay luz ambiente, podemos encender el LED
        }
        else if(luz > 50 && sensor_luz == 1){
            sensor_luz = 0; // si hay luz ambiente, no podemos encender el LED
        }
        HAL_ADC_Start_DMA(&hadc1, &luz, 1); // arrancamos de nuevo el convertidor
    }
}
```

2.1.3. SENSOR ULTRASONIDO

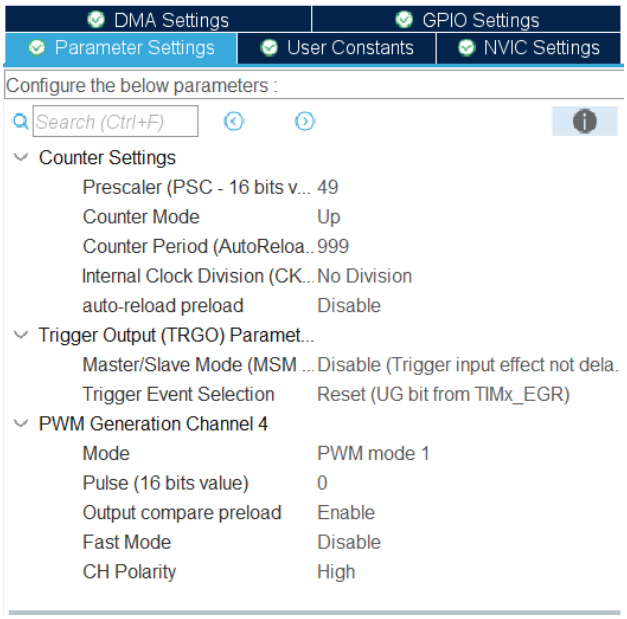
Se hará uso de este sensor para detectar la presencia de algún coche o persona que entre al parking, encendiendo la luz cuando detecte esta presencia junto con un nivel de luminosidad bajo.

El sensor ultrasónico HC-SR04, mide distancias utilizando ondas de sonido a frecuencias ultrasónicas (40 kHz). Este sensor, como se puede apreciar en la imagen, consta de 4 patillas.



- Vcc – Alimentación del sensor, irá conectada a la tensión de 5V de la placa.
- Trigger – Es un pin de entrada. El microcontrolador envía una señal de pulso corto (de 10 microsegundos) para iniciar la medición.
- Echo – Recibe la señal de retorno, es un pin de salida. El sensor genera un pulso cuya duración corresponde al tiempo que tarda la onda ultrasónica en viajar hasta el objeto y regresar.
- Gnd – Tierra o referencia del sensor.

Para la señal trigger, se necesita generar una entrada de 10µs y un periodo de 1ms. Para ello, se usará un temporizador PWM, ya que este permite generar la entrada de forma eficiente, precisa y automatizada. Para ello, se habilitará en el pin PB9 el temporizador TIM4 en modo PWM con la siguiente configuración.



Los valores del prescaler y del periodo se han obtenido con los siguientes cálculos.

$$\text{Prescaler} = \frac{TIMx_{CLK}}{f_{tim}} - 1 \quad = \frac{50\,\text{MHz}}{1\,\text{MHz}} - 1 = 50 - 1 = 49$$

$$ARR = (\text{Período deseado en µs}) \cdot f_{tim} - 1 \quad = 1000 \cdot 1 - 1 = 1000 - 1 = 999$$

El valor de TIMx_CLK se ha obtenido de la datasheet de la placa.

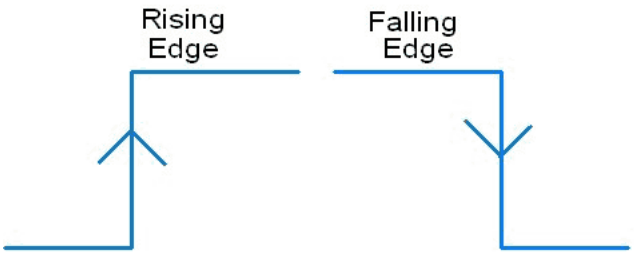
Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complemen- tary output	Max. interface clock (MHz)	Max. timer clock (MHz)
Advanced -control	TIM1	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	100	100
General	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	50	100
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	50	100

Una vez configurado, en el código se añadirá, en la función main, las siguientes líneas para activar el temporizador.

```
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);//temporizador para el trigger del sensor ultrasonido
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 10);
```

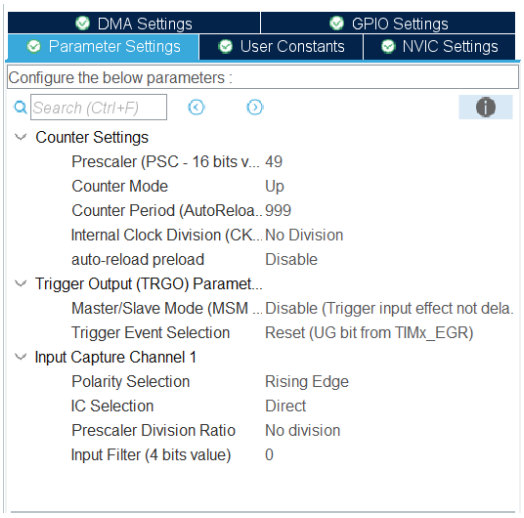
Cuando ya esté el temporizador para el trigger configurado, se procederá con el temporizador Input Capture (IC) para el ECHO.

Se utilizará este tipo de temporizador ya que así se puede obtener con precisión el tiempo entre dos eventos.

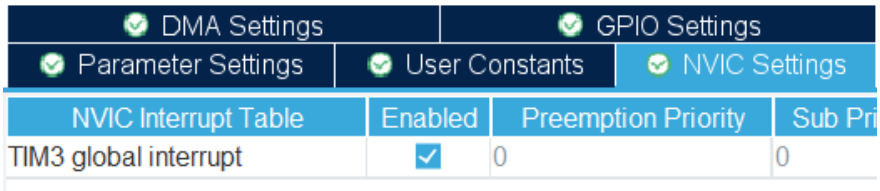


Este temporizador asignará un valor de tiempo nada más recibir el flanco de subida, y esperará al flanco de bajada para asignar el siguiente valor. Una vez obtenidos el valor en tiempo del flanco de subida y de bajada, los compara y obtiene una distancia.

Para llevarlo al proyecto, deberá tener la siguiente configuración.



Se usan los mismos parámetros del prescaler y periodo que para el temporizador anterior.



Se habilitan las interrupciones y se procede a hacer el código. Para ello, se usará una función callback, que activará la variable global “sensor_presencia” según las condiciones que se den. Al igual que anteriormente, será en el main donde se habilite o no el led.

```c
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);//leemos el valor del tiempo cuando ocurre el flanc
            // 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);//leemos el valor del tiempo cuando ocurre el flanc

            __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 < 200){ // presencia detectada
                sensor_presencia = 1;
            }
            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);
        }
    }
}
```

2.2. SALIDAS

2.2.1. LED

En este proyecto, las salidas utilizas son dos leds, uno rojo, que será la luz de emergencia, y uno verde, que simula la luz que ilumina toda la habitación. Los pines utilizados son los siguientes.

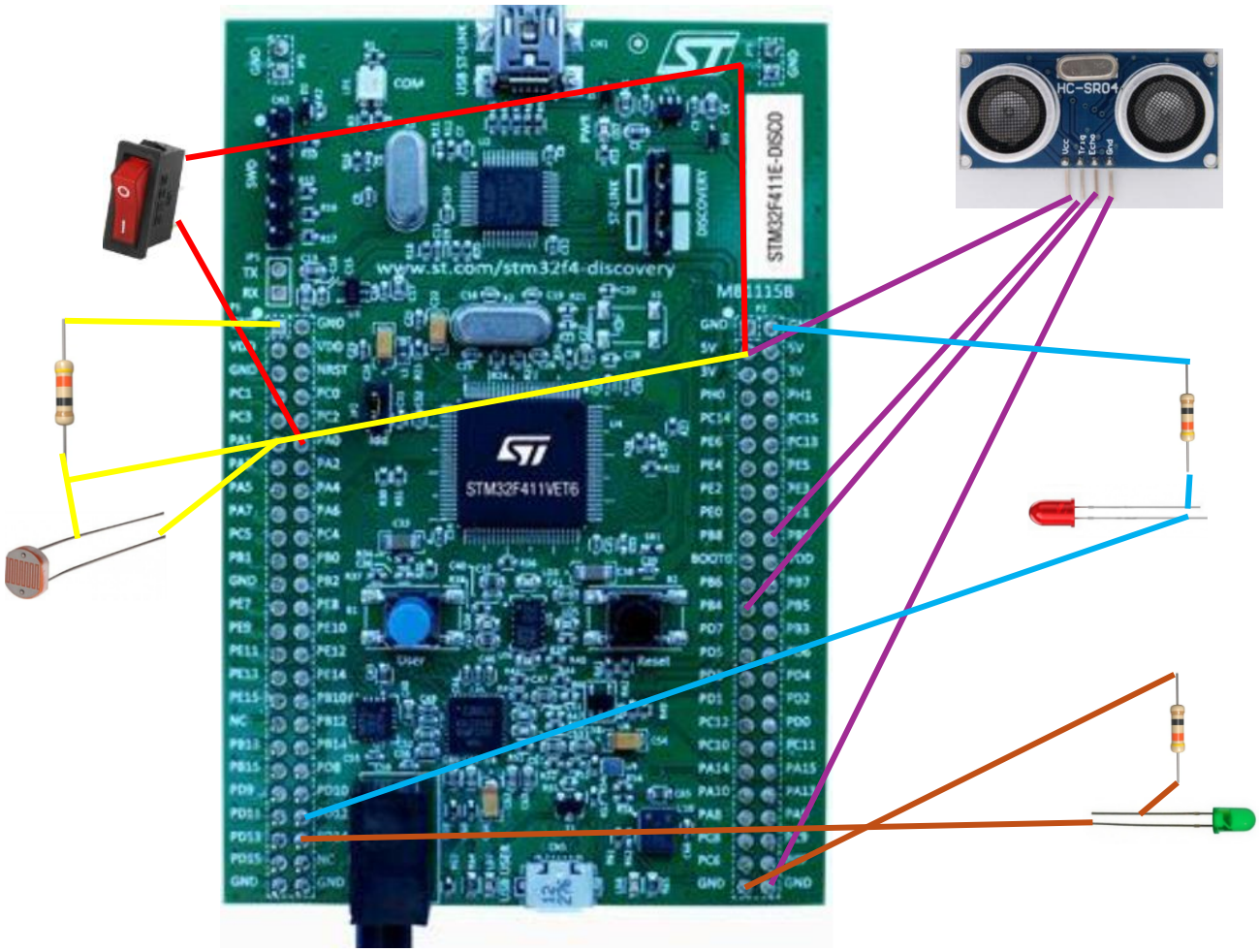
El pin PD12 será el led verde, y el pin PD14 será el pin rojo.

Como ya se ha mencionado anteriormente, el control de los pines viene determinado por variables globales, y en función de estas, los leds tomaran distintos estados. Es por esto que el código correspondiente se encuentra en el bucle while(1) de la función main.

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

  /* USER CODE BEGIN 3 */
    // Si el LED verde está encendido (PD12), apaga el LED rojo (PD14)
    if (HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_12) == GPIO_PIN_SET)
    {
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET); // Apaga LED rojo
    }
    else
    {
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET); // Enciende LED rojo
    }


    if(interruptor==1)//bucle para encender el led con el interruptor
    {
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
    }else if(interruptor==0 && sensor_luz==1 && sensor_presencia==1)//bucle para encender el led con presencia
    {
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
        HAL_Delay(5000);
    }else
    {
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
    }
}
/* USER CODE END 3 */
```

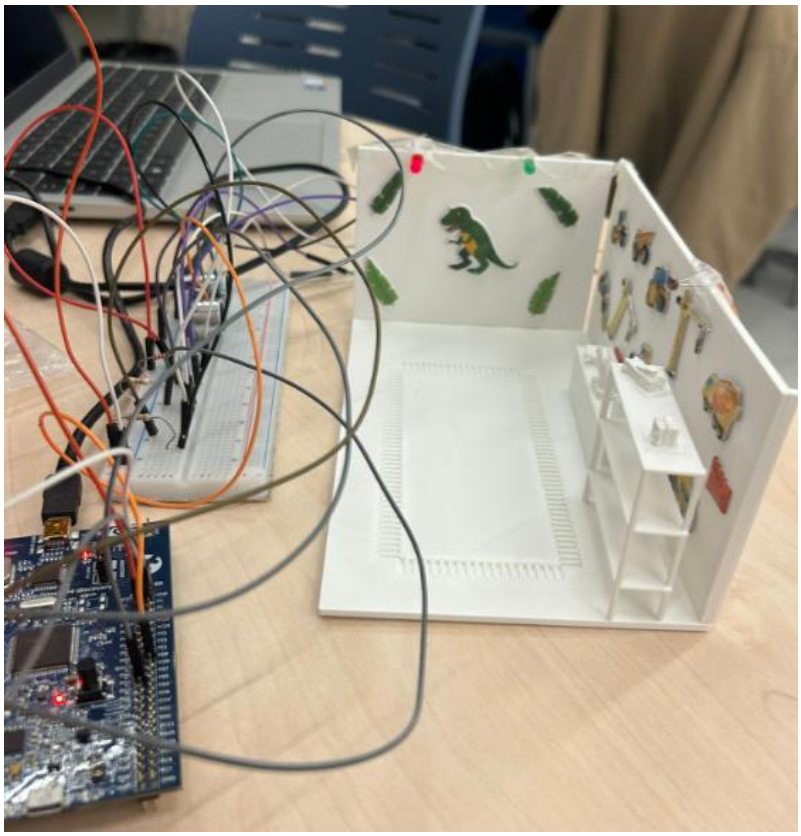
3. ESQUEMA CONEXIONES

A continuación, se muestran todas las conexiones que tendrán los elementos con la placa.

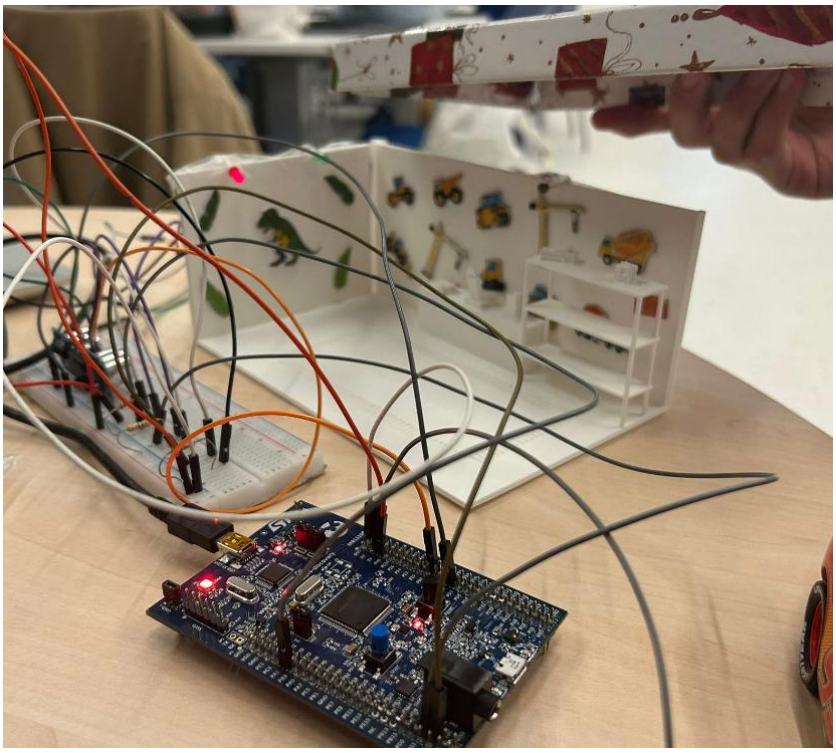


4. MAQUETA

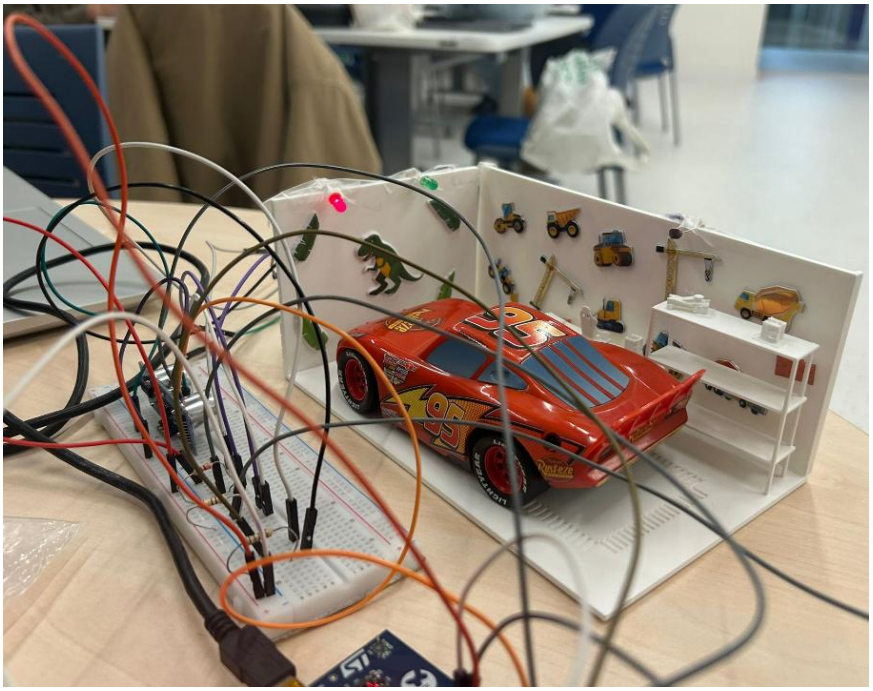
En las siguientes imágenes, se muestran el funcionamiento del código en el modo automático en la maqueta.



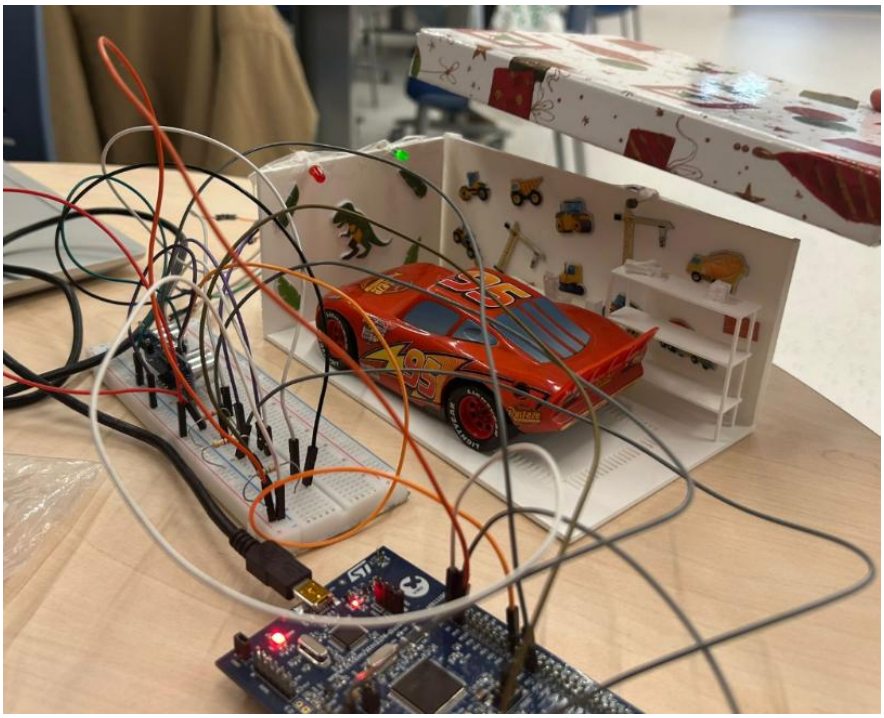
Inicialmente, estará la luz de emergencia encendida. Ya que no se ha detectado ni oscuridad ni presencia.



Al añadir oscuridad al entorno, se observa que no ocurre nada, ya que el requisito para que se encienda la luz es oscuridad y presencia.



De igual forma, cuando detecta presencia si no hay oscuridad, tampoco ocurrirá nada.



Una vez detecta oscuridad y presencia, se encenderá la luz. Permaneciendo encendida durante 5 segundos, hasta que se apagará.

En la entrega se presenta también el video en el que se muestra el funcionamiento completo.

5. GITHUB

https://github.com/juanignacio-martin/TRABAJO_STM32

En el link anterior se incluye el repositorio con el código utilizado, así como el video mencionado anteriormente y esta memoria.