PROYECTO 1

ENVIRONMENT/WEATHER STATION

FECHA:

31/05/2022

AUTORES:

Miguel de las Heras Fuentes

Diego Dorado Galán

Contenido

1. OBJETIVO DEL PROYECTO

El objetivo de este proyecto es representar una estación meteorológica y monitorizarla mediante un servidor de grafana. Para ello, primero mediremos algunas variables ambientales en la placa STM32 y estos datos recogidos los enviaremos mediante bluetooth a la placa ESP32 que a su vez los publicará en un broker mqtt.

2. ESQUEMA DEL CIRCUITO EN FRITZING

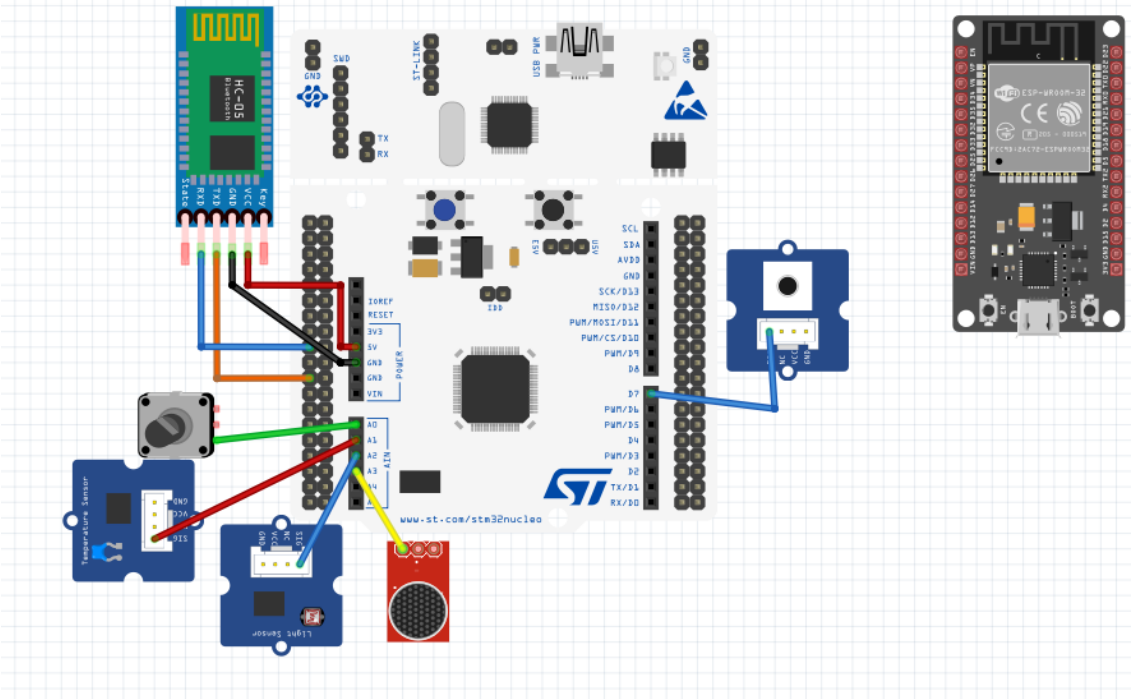

Ilustración 1: Circuito implementado en la plataforma Fritzing

3. COMPONENTES

- **Placa STM32 F411RE**

Placa encargada de la principal funcionalidad de la estación. Se encarga de calcular los valores percibidos por los sensores (utilizando algunas de las técnicas vistas en clase, como interrupciones o ADC) y enviarlos a la placa ESP32.

- **Placa ESP32**

Se encarga de la lectura de los datos enviados por la STM32 y su posterior publicación mediante mqtt.

- **Placa Grove**

- **Receptor Bluetooth HC-05**

Facilita la conexión entre ambas placas.

- **Sensor de temperatura**

Se encarga de medir la temperatura ambiente en grados centígrados.

- **Sensor de luminosidad**

Mide la intensidad lumínica.

- **Sensor de ruido/sonido**

Mide la contaminación acústica.

- **Touchpad**

Sensor que simula la presencia de lluvia (true/false)

- **Ángulo rotatorio**

Sensor que simula la señal de un anemómetro. Esto se realiza de la siguiente forma:

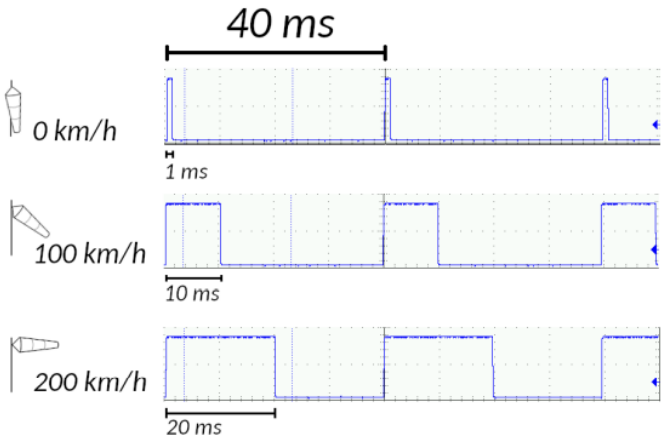

Ilustración 2: Lógica del ángulo rotatorio utilizado

4. PROYECTO STM32CubeMX

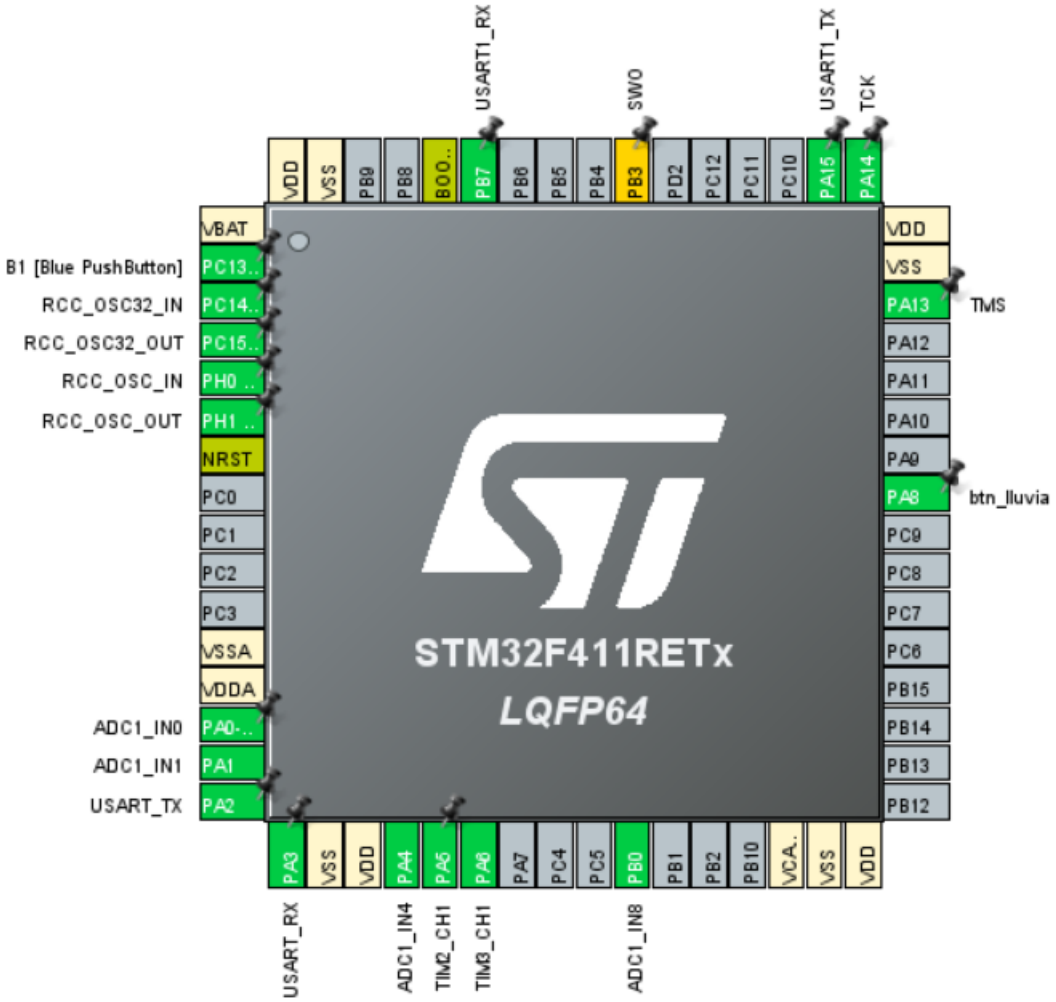

Ilustración 3: Configuración de los pines en STM32Cube

Como podemos observar en el esquema STM32Cube MX, hemos configurado el pin PA8 como la entrada de un botón (touchpad) que se corresponde con el sensor de presencia de lluvias. Este botón está configurado como interrupción EXTI8 de manera que cada vez que se pulse se lanzará una interrupción que incrementará un contador de gotas y así podremos saber si está lloviendo o no.

Para la medición de los datos hemos utilizado el ADC1 de manera que podremos convertir los valores analógicos obtenidos de los sensores a valores digitales. Como estamos utilizando varios sensores y cada uno está situado en un pin distinto, es necesarios usar varios canales. En este caso hemos utilizado los canales 0,1,4 y 8 que corresponderán a los pines PA0,PA1,PA4 y PB0. De esta manera la disposición de sensores será la siguiente:

- Ángulo rotatorio: PA0
- Sensor de temperatura: PA1
- Sensor de luminosidad: PA4
- Sensor de ruido: PB0
- Touchpad: PA8

En cuanto a conectividad hemos habilitado el UART1 con los pines PA15(TX) y PB7(RX) para realizar el envío de datos al receptor bluetooth. En este caso como solo vamos a enviar datos; no usaremos el pin RX.

En la sección de Timers; además del timer 2(habilitado por el profesor); hemos habilitado el canal 1 del timer 3 en modo input capture para realizar la medición del viento.

Por último, en la sección NVIC hemos habilitado las interrupciones globales del ADC1, las interrupciones EXTI line[9:5] para el botón, las interrupciones del timer 3 y las interrupciones del UART1.

5. EXPLICACIÓN DEL CÓDIGO

CÓDIGO DE LA PLACA ESP32

- **Configuración del Wi-Fi y del mqtt**

```
- const char* ssid = "AndroidApp";
- const char* password = "12345678";
-
- #define TOPICTEMP "esi/room1/temp"
- #define TOPICLUM "esi/room1/lum"
- #define TOPICWIND "esi/room1/wind"
- #define TOPICSOUND "esi/room1/sound"
- #define TOPICRAIN "esi/room1/rain"
- #define BROKER_IP "192.168.43.174"
- #define BROKER_PORT 2883
-
- WiFiClient espClient;
- PubSubClient client(espClient);
-
- //configuración de bluetooth
-
- uint8_t address[6]  = {0xFF, 0xDB, 0x01, 0x00, 0x81, 0x82};
- String name = "BT07";
- char *pin = "1234";
- bool connected;
-
- BluetoothSerial SerialBT;
-
- //Configuración freeRTOS
-
```

```
- QueueHandle_t xQueueWind;
- QueueHandle_t xQueueTemp;
- QueueHandle_t xQueueLight;
- QueueHandle_t xQueueSound;
- QueueHandle_t xQueueRain;
```

En este fragmento de código, observamos la definición de variables como el ssid, password, los distintos topic que usaremos para la conexión mqtt, la sección encargada de la configuración bluetooth, y la creación de colas para freeRTOS.

- Métodos enfocados en la conexión Wi-Fi y mqtt

```
//Método para la conexión WI-FI
void wifiConnect()
{
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  Serial.println("Connected to the WiFi network");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}

//Método para la conexión mqtt
void mqttConnect() {
  client.setServer(BROKER_IP, BROKER_PORT);
  while (!client.connected()) {
    Serial.print("MQTT connecting ...");

    if (client.connect("ESP32Client1")) {
      Serial.println("connected");
    } else {
      Serial.print("failed, status code =");
      Serial.print(client.state());
      Serial.println("try again in 5 seconds");

      delay(5000);  //* Wait 5 seconds before retrying
    }
  }
}
```

Este código ha sido reutilizado de prácticas anteriores, dónde ya fueron explicados, por lo que no entraremos en mayor detalle.

- **Método de conexión Bluetooth**

```
- void BTConnect(){
-   SerialBT.begin("ESP32",true);
-   SerialBT.setPin(pin);
```

```
-     Serial.println("The device started in master mode, make sure
      remote BT device is on!");
-
-     SerialBT.enableSSP();
-
-     do {
-         Serial.println("Trying to connect...");
-         delay(500);
-         connected = SerialBT.connect(address);
-       } while (SerialBT.hasClient() == 0 && !connected);
-
-     Serial.println("Connected Succesfully!");
-   }
```

En este código establecemos la conexión bluetooth para el dispositivo BT07 mediante un bucle do-while, ya que habitualmente no se conecta al primer intento. Como podemos apreciar, utilizamos el argumento true para configurar esta placa como maestro en la comunicación bluetooth.

- **Tarea recibir**

```
 //Tarea para recibir los datos de la STM32 mediante Bluetooth
void vTaskReceive(void *parameters){

  portBASE_TYPE xStatus;
  const portTickType xTicksToWait = 100 / portTICK_RATE_MS;
  String recibido = "";
  for(;;){

    while (SerialBT.available())
    {
      Serial.println("Recibiendo nuevos datos...");
      recibido = SerialBT.readString();
      Serial.println(recibido);

      char datos[20];
      strcpy(datos, recibido.c_str());

      char *token = strtok(datos,","); //dato viento
      xStatus = xQueueSendToBack(xQueueWind,token,xTicksToWait);
      if(xStatus !=pdPASS) Serial.println("No se ha podido añadir
elemento a la cola wind");

      token = strtok(NULL,","); //dato temperatura
      xStatus = xQueueSendToBack(xQueueTemp,token,xTicksToWait);
      if(xStatus !=pdPASS) Serial.println("No se ha podido añadir
      elemento a la cola sound");

        … //RESTO DE TAREAS

      token = strtok(NULL,","); //dato lluvia
      char llueve = '0';
      if(token[0] == '0') xStatus =
xQueueSendToBack(xQueueRain,&llueve,0);
      else {
```

```
        llueve = '1';
        xStatus = xQueueSendToBack(xQueueRain,&llueve,xTicksToWait);
      }
      if(xStatus !=pdPASS) Serial.println("No se ha podido añadir
elemento a la cola rain");
      vTaskDelay(5000/portTICK_RATE_MS);

    }
  }
}
```

Esta tarea es la de mayor prioridad, la encargada de recepción de datos desde la placa STM32. Es decir, priorizamos la recepción de datos frente a la realización del resto de tareas, tal y como observamos en la ilustración 4.

Mientras el dispositivo esté conectado vía bluetooth, se irán añadiendo los datos a sus respectivas colas de tareas. Lo hemos realizado mediante un conjunto de sentencias if y mediante la función strtok para fragmentar el mensaje recibido de la STM32 y obtener el dato de cada sensor. A continuación introducimos el dato en su cola correspondiente con un formato definido mediante sprintf.

- **Resto de tareas**

```
- void vTaskWind(void *parameters){
-
-   portBASE_TYPE xStatus;
-   const portTickType xTicksToWait = 500 / portTICK_RATE_MS;
-   char buffer[8];
-   char envio[30];
-   for(;;){
-     xStatus = xQueueReceive(xQueueWind, buffer,xTicksToWait);
-     if (xStatus == pdPASS)
-     {
-       Serial.println(buffer);
-       sprintf(envio,"{\"wind\":%s}",buffer);
-       client.publish(TOPICWIND,envio);
-       vTaskDelay(4000/portTICK_RATE_MS);
-     }
-   }
-   vTaskDelete(NULL);
- }
```

Para la publicación de tareas utilizamos un bucle infinito que, en función del estado de la lectura de mensajes de la cola, imprimirá los valores almacenados en el buffer y publicará en el respectivo topic el mensaje. En caso de que salga de este bucle por cualquier error se eliminará la tarea.

De forma idéntica se realiza la publicación del resto de tareas. La lectura de la presencia de lluvia se realiza distinguiendo entre 0 (no lluvia) y 1 (lluvia).

- **Creación de tareas**

```
-     Serial.println("Intentando crear tareas");
-
-     if (xQueueWind !=NULL && xQueueTemp !=NULL && xQueueLight
    !=NULL && xQueueSound !=NULL && xQueueRain !=NULL )
-     {
-       Serial.println("CREANDO TAREAS");
-       xTaskCreatePinnedToCore(vTaskReceive,"Task receive", 2000,
    NULL, 1, NULL, 0);
-       xTaskCreatePinnedToCore(vTaskWind,"Task wind", 2000, NULL,
    2, NULL, 0);
-       xTaskCreatePinnedToCore(vTaskTemp,"Task temp", 2000, NULL,
    2, NULL, 0);
-       xTaskCreatePinnedToCore(vTaskLight,"Task light", 2000, NULL,
    2, NULL, 0);
-       xTaskCreatePinnedToCore(vTaskSound,"Task sound", 2000, NULL,
    2, NULL, 0);
-       xTaskCreatePinnedToCore(vTaskRain,"Task rain", 2000, NULL,
    2, NULL, 0);
-
-     }
```

Observamos como la tarea con mayor prioridad es la de recepción. Además, a todas las tareas se les asigna el core 0 ya que según las fuentes que hemos consultado; este core es el encargado de realizar las funciones relacionadas con WI-FI y bluetooth.

A continuación podemos ver un esquema lógico del funcionamiento de esta placa:

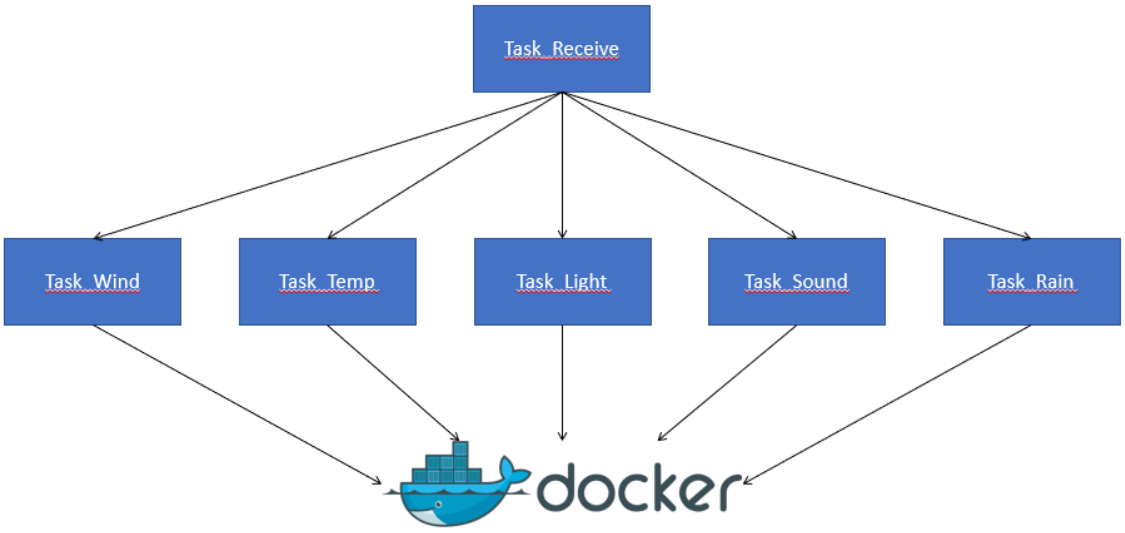

Ilustración 4: Esquema lógico del funcionamiento de la placa ESP32

CÓDIGO DE LA PLACA STM32

A continuación se muestra una imagen de la maquina de estados finita utilizada para la implementación del código.

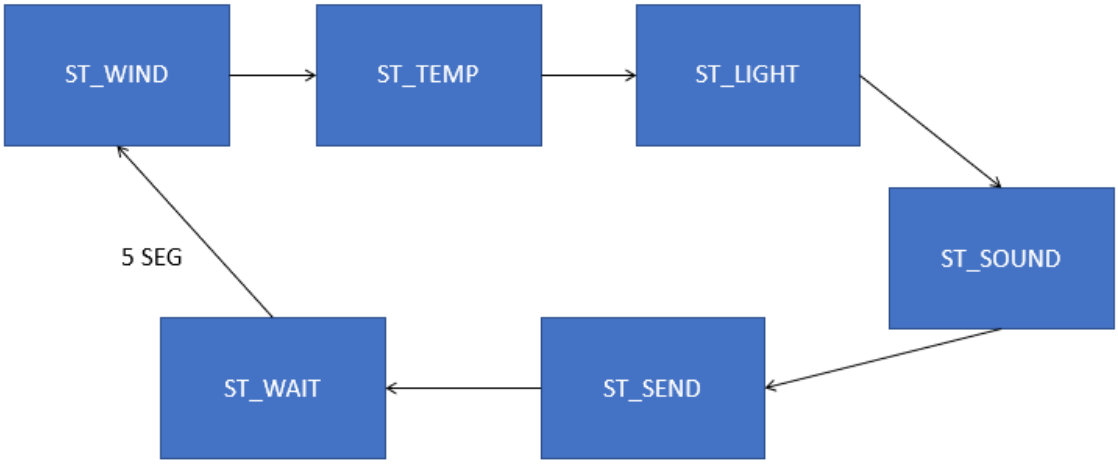

Ilustración 5: FSM utilizada en la placa STM32

Al tener disponible un único módulo ADC; es necesario cambiar de canal cuando queramos medir un sensor distinto. Por ello, asignamos un periodo de tiempo a cada canal. Como podemos ver en la imagen anterior, cada estado corresponde a la medición de un determinado sensor. Todos los estados de medición tienen el siguiente formato salvo el de envío de datos.

```
case ST_light:

    ADC_Select_CH4();
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, 1000);
    int val3 = HAL_ADC_GetValue(&hadc1);
    luminosidad = val3 * 100 / 4095;
    printf("Sensor luminosidad%%: %d\r\n", luminosidad);
    next_state = ST_sound;

    break;
```

Como vemos, en val3 almacenamos el valor obtenido del sensor(0-4095) y a continuación mediante las fórmulas matemáticas correspondientes transformaremos ese valor uno con las unidades de medida deseadas.

- **Medición de lluvia**

Para simular esta funcionalidad, utilizamos una interrupción de manera que cada vez que se pulse el botón, saltará una rutina en la que incrementaremos la variable lluvia que simulará un contador de gotas de agua.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if (GPIO_Pin == btn_lluvia_Pin)
  {
    lluvia++;
  }
}
```

- **Envío de datos**

```
void make_message(){

  char mensaje[20];
  sprintf(mensaje, "%d,%d,%d,%d,%d", viento, temperatura, luminosidad,
sonido, lluvia);
  printf("LLuvia: %d\r\n", lluvia);
  printf("%s\r\n",mensaje);
  HAL_UART_Transmit(&huart1,mensaje,sizeof(mensaje),100);

}
```

6. IMAGEN DEL CIRCUITO

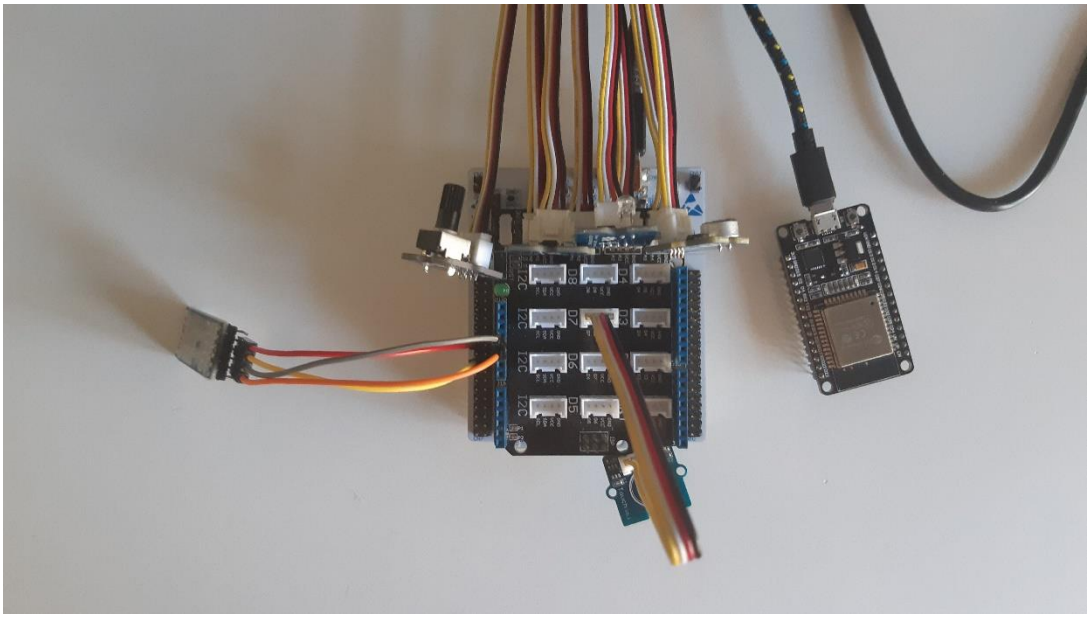

Ilustración 6: Circuito creado para la realización del proyecto