

**PROYECTO 1**

**ENVIRONMENT/WEATHER STATION**

**FECHA**:

31/05/2022

**AUTORES:**

Miguel de las Heras Fuentes

Diego Dorado Galán



Contenido

[1. OBJETIVO DEL PROYECTO 3](#_Toc104839935)

[2. ESQUEMA DEL CIRCUITO 4](#_Toc104839936)

[3. COMPONENTES 4](#_Toc104839937)

[- **Placa STM32 F411RE** 4](#_Toc104839938)

[- **Placa ESP32** 4](#_Toc104839939)

[- **Placa Grove** 4](#_Toc104839940)

[- **Receptor Bluetooth HC-05** 4](#_Toc104839941)

[- **Sensor de temperatura** 4](#_Toc104839942)

[- **Sensor de luminosidad** 4](#_Toc104839943)

[- **Sensor de ruido/sonido** 5](#_Toc104839944)

[- **Touchpad** 5](#_Toc104839945)

[- **Ángulo rotatorio** 5](#_Toc104839946)

[4. PROYECTO STM32CubeMX 5](#_Toc104839947)

[5. EXPLICACIÓN DEL CÓDIGO 6](#_Toc104839948)

[CÓDIGO DE LA PLACA ESP32 7](#_Toc104839949)

[6. STACK TIG (DASHBOARD) 10](#_Toc104839950)

# OBJETIVO DEL PROYECTO

Mediante este proyecto buscaremos representar una estación de medida, encargada de medir las distintas variables ambientales del entorno, mientras que la segunda parte de esta tratará de monitorizar los valores captados por dichos sensores.

# ESQUEMA DEL CIRCUITO

Diagrama

Descripción generada automáticamente

# 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) y comunicarlos a la placa ESP32.

# **Placa ESP32**

Se encarga de realizar el protocolo 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:Gráfico

Descripción generada automáticamente con confianza media

# PROYECTO STM32CubeMX

Interfaz de usuario gráfica

Descripción generada automáticamente

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.

(Explicar resto de pines, que no sé cuál es cuál xd)

# 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 "192.168.1.138"
* #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.

* **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 el diagrama de estados finitos adjunto.

Básicamente, mientras el dispositivo esté conectado vía bluetooth, se irá añadiendo los datos a sus respectivas colas de tareas. Lo hemos realizado mediante una serie de if y mediante la función strtok para una mayor claridad y distinción de las tareas.

* **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, para finalmente eliminar dicha 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.

## CÓDIGO DE LA PLACA STM32

Podemos mostrar la realización de las distintas tareas mediante un diagrama de estados finito (aquí podría ir el diagrama):

* **Medición de temperatura**
* **Medición de luz**

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);

//HAL\_Delay(1000);

next\_state = ST\_sound;

break;

En la variable val3 almacenamos el valor obtenido por el manejador de señal, y que utilizamos para calcular la luminosidad.

* **Medición de ruido**
* **Medición de viento**
* **Medición de lluvia**

Para simular esta funcionalidad, comprobamos si el touchpad está siendo pulsado en el momento de la medición. Básicamente, si el botón está pulsado el valor de lluvia incrementa. Posteriormente, se enviará a la placa ESP32 un valor 0 (no lluvia) o 1 (lluvia).

void HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)

{

if (GPIO\_Pin == btn\_lluvia\_Pin)

{

lluvia++;

}

}

# STACK TIG (DASHBOARD)