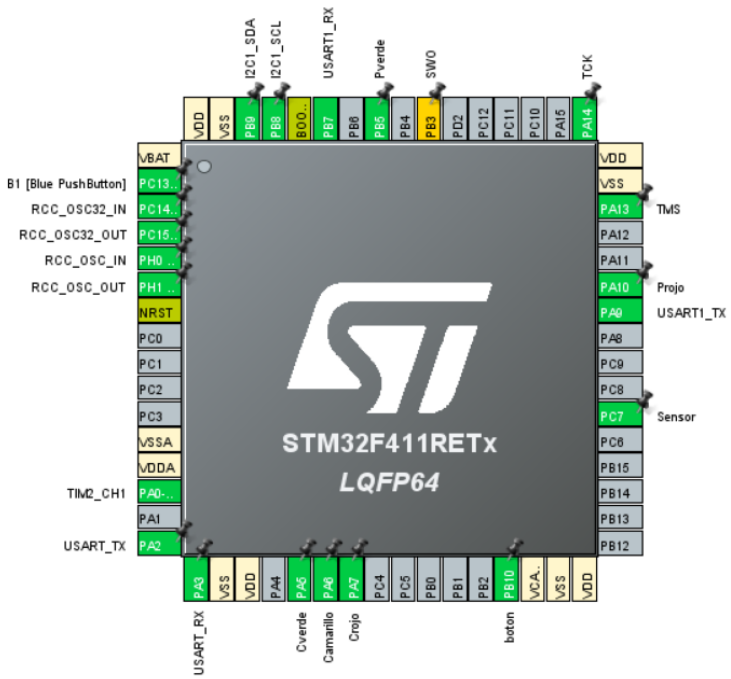
– Lab 7: MQTT Remote State & Integration –

Objetivo:

En este caso nuestro cliente desea añadir al semáforo desarrollado hasta el momento una monitorización remota que permita conocer el estado de cada semáforo mediante MQTT. Por lo tanto, se deberá incorporar la plataforma de desarrollo ESP32 a modo de módulo WiFi con el fin de proporcionar una comunicación inalámbrica a la plataforma STM32-F411RE. La comunicación entre el ESP32 y el F411RE se realizará mediante una conexión punto a punto UART como muestra la figura (dependiendo de cada diseño, es posible que los pines utilizados sean diferentes), es decir, la placa F411RE enviará la información del estado del semáforo a través de un puerto TX a la placa ESP32, cuya información será recogida por un pin asociado a un módulo UART, donde sólo se hará uso del puerto RX*. A su vez, el ESP32 estará conectado vía WiFi y publicará cada cambio de estado mediante MQTT.

Esquema STM32CubeMX:

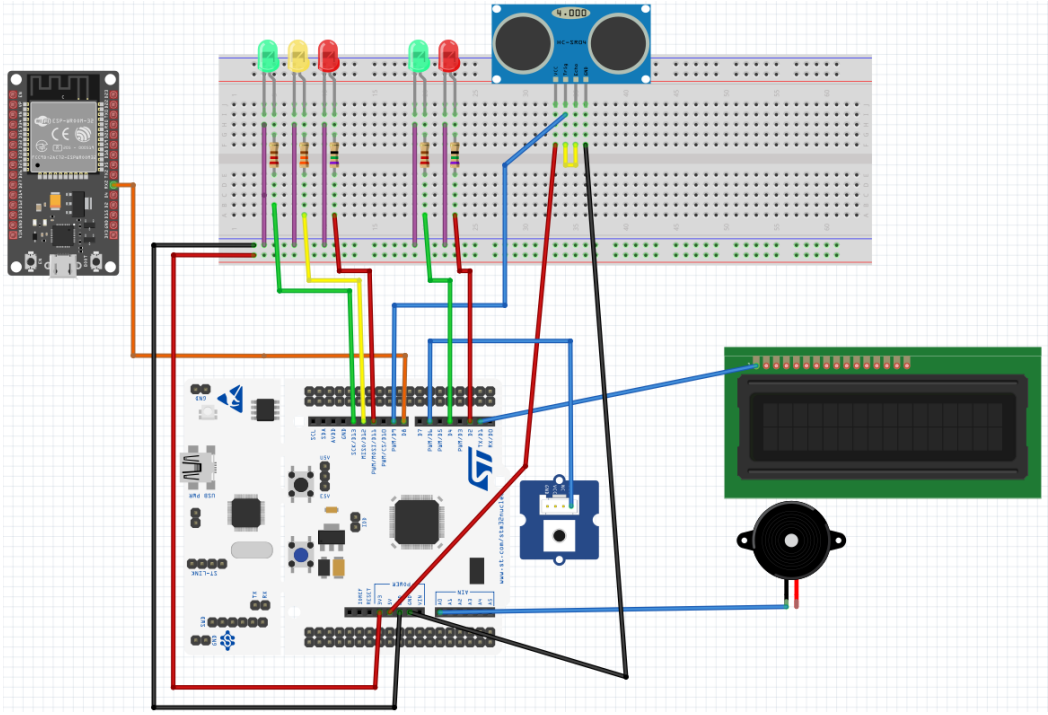


Pin Name	Signal on Pin	GPIO outpu...	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PA5	n/a	Low	Output Pus...	No pull-up a...	Low	Cverde	☑
PA6	n/a	Low	Output Pus...	No pull-up a...	Low	Camarillo	☑
PA7	n/a	Low	Output Pus...	No pull-up a...	Low	Crojo	☑
PA10	n/a	Low	Output Pus...	No pull-up a...	Low	Projo	☑
PB5	n/a	Low	Output Pus...	No pull-up a...	Low	Pverde	☑
PB10	n/a	n/a	External Int...	No pull-up a...	n/a	boton	☑
PC7	n/a	Low	Output Pus...	No pull-up a...	Low	Sensor	☑
PC13-ANTI...	n/a	n/a	External Int...	No pull-up a...	n/a	B1 [Blue P...	☑

Pin Name	Signal on Pin	GPIO outpu...	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PA2	USART2_TX	n/a	Alternate F...	No pull-up a...	Very High	USART_TX	☑
PA3	USART2_RX	n/a	Alternate F...	No pull-up a...	Very High	USART_RX	☑
PA9	USART1_TX	n/a	Alternate F...	No pull-up a...	Very High		☐
PB7	USART1_RX	n/a	Alternate F...	No pull-up a...	Very High		☐

Tal y como nos pide el enunciado, hemos añadido una conexión UART para conectar ambas placas. La placa F411RE usa un puerto TX y la ESP32 un puerto RX por lo que el puerto RX de la placa F411RE y el puerto TX de la placa ESP32 no se utilizarán en ningún momento.

Esquema:



Componentes:

- Placa STM32 F411RE
- Placa Grove
- Botón para cable de cuatro canales
- Ultrasonidos HC-SR04
- 2 LED rojos, 2 LED verdes, LED Amarillo
- Respectivas resistencias (resistencia explicada en prácticas anteriores)
- Buzzer
- Grove-LCD RGB Backlight
- Placa ESP32

Explicación de la solución:

En primer lugar, debemos mencionar que hemos declarado una variable booleana *send*, que nos servirá como mecanismo de control para el envío de información a la placa ESP32.

```
typedef enum {false, true} bool;
bool send = false;
```

En lo referente a la lógica del programa, respecto a prácticas anteriores ha sido modificada para lograr el funcionamiento deseado. Por ejemplo, en el estado inicial tenemos el siguiente código:

```
case ST_Inicial:
  GPIOA -> ODR = GPIO_ODR_OD5_Msk | GPIO_ODR_OD10_Msk; //coche verde y peaton rojo activos

  setRGB(255,0,0);
  clearLCD();
  HAL_I2C_Master_Transmit(&hi2c1,LCD,lcdgreen,sizeof(lcdgreen)-1,100);

  if(!send) {
    printState(&huart1,"0");
    send=true;
  }

  calcularDistancia();
  distancia = (pulso * 10/58);
  printf("Distancia: %d cm\r\n", distancia);

  if(distancia < 20){
      next_state = ST_CAmarillo;
  }
  break;
```

Hemos añadido una sentencia if que comprobará el valor de la variable send y en caso de que esta sea false enviará el estado del semáforo a la placa ESP32 y cambiará el estado de la variable a true. Se ha utilizado esta variable para evitar que se realice un envío redundante de información a la placa ESP32 que pueda provocar un bloqueo de esta.

En el resto de los estados, las únicas modificaciones que se han realizado han sido la adición de los métodos printState.

El método printState lo utilizaremos para enviar el estado del semáforo a la placa ESP32. Este método consiste en:

```
//metodo para enviar el estado del semaforo a ESP32
void printState(USART_HandleTypeDef *huart, char _out[]){
  HAL_UART_Transmit(huart, (uint8_t *) _out, strlen(_out),HAL_MAX_DELAY);
}
```

Mediante la función HAL_UART_Transmit, transmite al manejador huart el contenido del array (que será el valor 0,1 o 2 que refleja el estado en el que se encuentra el semáforo), indicando además su tamaño y un timeout. De tal forma, enviamos el estado del semáforo a ESP32.

El código correspondiente a la placa ESP32 es el siguiente:

Definimos y declaramos las siguientes variables:

```
#define RXD2 16
#define TXD2 17 //Este no se va a utilizar

const char* ssid = "AndroidApp";
const char* password = "12345678";

#define TOPIC "esi/lab7"
#define BROKER_IP "192.168.43.174"
#define BROKER_PORT 2883

WiFiClient espClient;
PubSubClient client(espClient);
```

Como podemos ver, los pines UART se han declarado en el 16(RX) y 17(TX) aunque este último no se utilizará ya que no tenemos que enviar nada a la placa F411RE.

Se declaran también los métodos wificonnect y mwttConnect para realizar tanto la conexión wifi como la conexión con el broker mqtt.

```
void wifiConnect()
{
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);

  }

}

void mqttConnect() {
  client.setServer(BROKER_IP, BROKER_PORT);
  while (!client.connected()) {


    if (client.connect("ESP32Client1")) {

    } else {

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

    }
  }
}
```

Como se implementaron para la tarea A12 no entraremos en mucho más detalle.

En el void setup inicializamos dichos métodos y declaramos un serial que será el que utlizaremos para la conexión UART con la placa F411RE. Le daremos una velocidad de 115200 bits/s para que coincida con la velocidad de la F411RE.

```
void setup() {
  // put your setup code here, to run once:
  Serial2.begin(115200, SERIAL_8N1, RXD2, TXD2);
  wifiConnect();
  mqttConnect();
}
```

En el void loop, se leerá el valor del estado (a través del pin 16) que ha sido enviado por la placa F411RE y dependiendo del valor recibido se publicará un mensaje u otro en el topic definido al inicio del código.

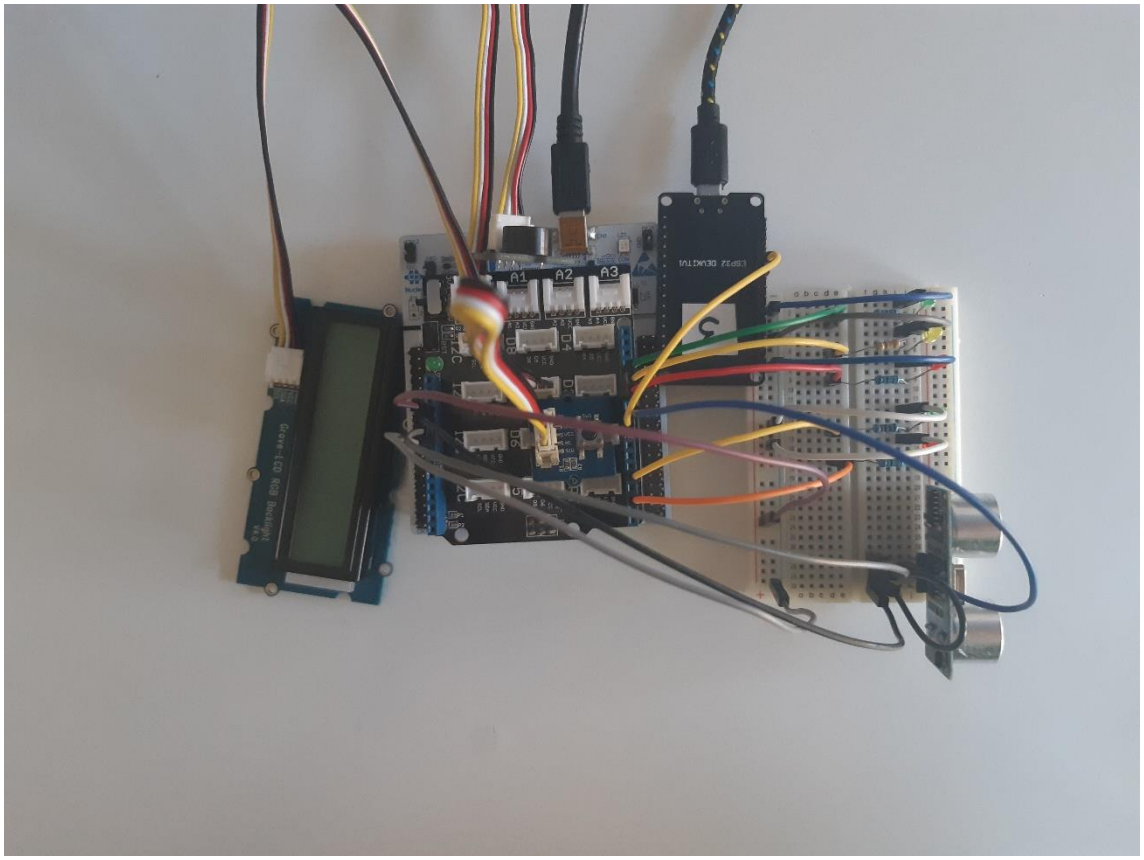
```
void loop() {
  // put your main code here, to run repeatedly:

  char state=Serial2.read();

    if (state=='0')
    {
      client.publish(TOPIC,"Estado del semaforo: GREEN");
      //Serial.println("Green");
    }else if (state=='1')
    {
      client.publish(TOPIC,"Estado del semaforo: YELLOW");
      //Serial.println("Yellow");
    }else if(state=='2'){
      client.publish(TOPIC,"Estado del semaforo: RED");
      //Serial.println("Red");
    }


}
```

Imagen del circuito:



Modo de ejecución(Windows):

Para la ejecución del programa primero deberemos iniciar el broker. Para ello utilizaremos el comando:

"C:\Program Files\mosquitto\mosquitto.exe" -c broker.conf

A continuación iniciaremos el suscriptor que será quien reciba todos los mensajes enviados por los nodos publicadores. Para ello el comando utilizado será:

"C:\Program Files\mosquitto\mosquitto_sub.exe" -t esi/lab7 -h 192.168.43.174 -p 2883

Como podemos ver; el topic utilizado para el suscriptor es esi/lab7. Esto le permitirá recibir los mensajes publicados por la placa ESP32.

La salida del programa en el suscriptor deberá ser algo así: