Project 1: Environment/Weather Station

Alberto Vázquez Martínez

Alberto.Vazquez1@alu.uclm.es

Curso 2021/22 – Diseño de Sistemas Basado en Microprocesador

Universidad de Castilla-La Mancha

Escuela Superior de Informática

Contenido

1. Configuración STM32CubeMX

Primero se ha llevado a cabo una configuración de los diferentes pines de la placa STM32F411RE con el software STM32CubeMX. Las configuraciones de los pines son las siguientes:

- Configuración de pines para la señal analógica ADC
- Configuración de pines para comunicación UART
- Configuración de pin como entrada GPIO con interrupciones

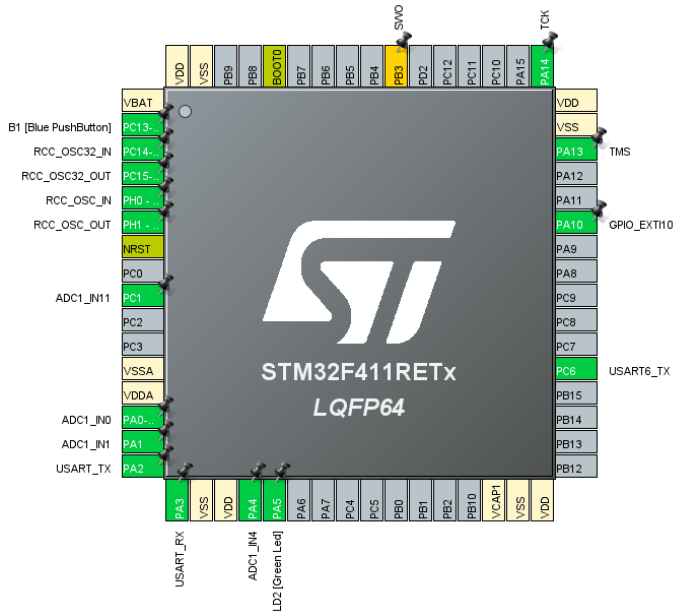

Fig. 1 – Configuración general de los pines

La configuración de los pines para mediciones analógicas es la que se muestra en la imagen. Los pines configurados son el PA0, PA1, PA4 y PC1.

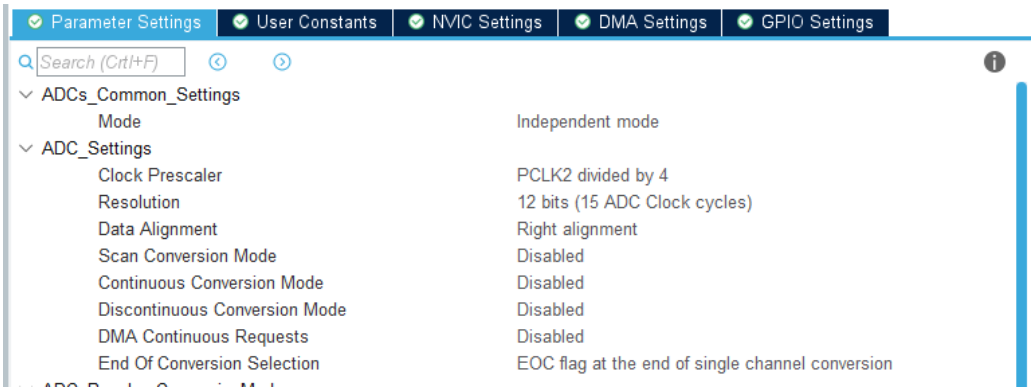

Fig. 2 – Configuración ADC1

A continuación, realizo la configuración para realizar la comunicación UART entre la placa ESP32 y la STM32. Habilito el UART6 y lo pongo en modo *Single Wire (Half-Duplex)* para utilizar un único cable para establecer la comunicación, ya que en este caso solo queremos enviar información desde la STM32 hacia la ESP32.

Establezco el *baud rate* a 115200 bits/s, por lo que, en la configuración de la ESP32, también estableceré ese mismo *baud rate*.

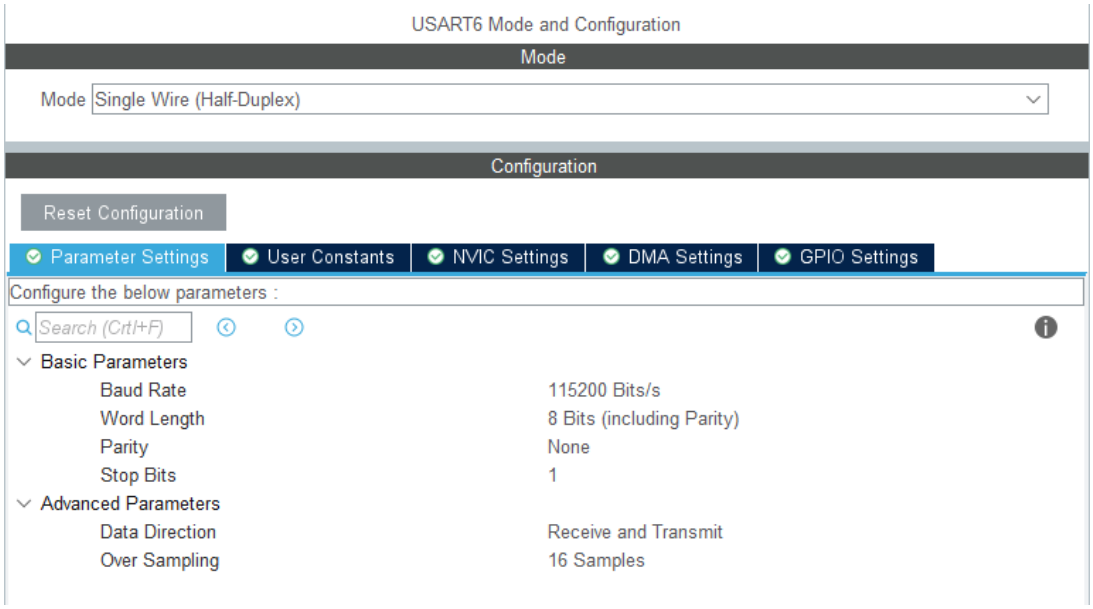

Fig. 3 – Configuración UART6

Por último, configuro el PA10 como GPIO_EXT10, ya que este pin se utilizará para simular el sensor de precipitación y para ello se utilizará una interrupción. Además, se ha configurado el NVIC para que se habiliten las interrupciones por los pines del 10 al 15.

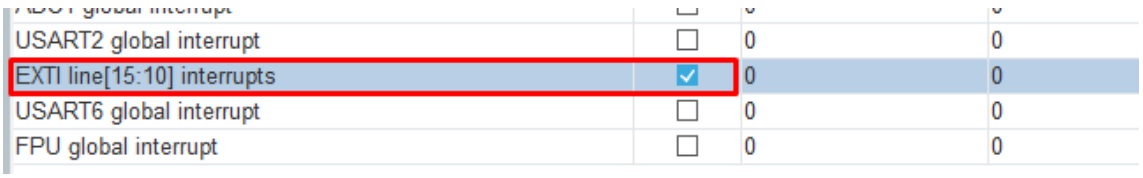

Fig. 4 – Habilitar las interrupciones

2. Circuito diseñado

El circuito diseño para este proyecto es el siguiente. (El *buzzer* representa al sensor de sonidos).

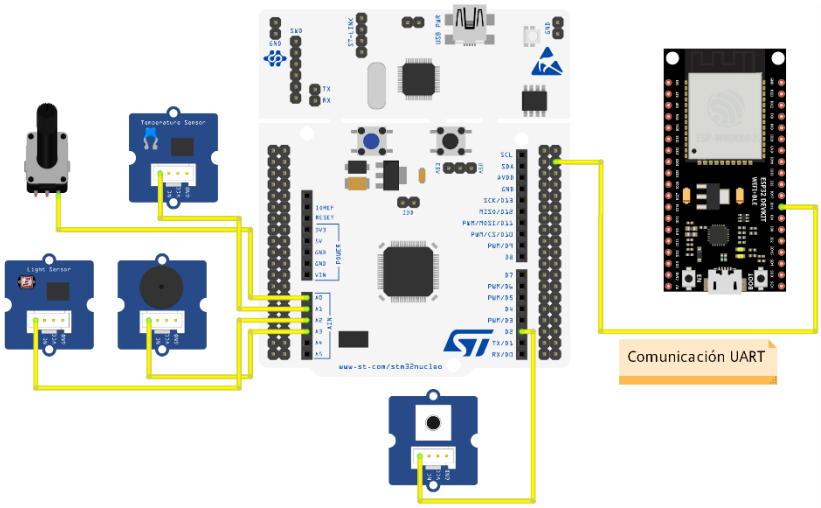

Fig. 5 – Circuito diseñado

3. Código desarrollado para la placa STM32

Se ha utilizado una maquina finita de estados (FSM) para ir midiendo cada uno de los sensores. Los estados que se han creado ha sido los siguientes:

- ST_TEMPERATURE: Estado para medir la temperatura.
- ST_BRIGHTNESS: Estado para medir la luminosidad.
- ST_NOISE: Estado para medir el ruido.
- ST_PRECIPITATION: Estado para medir la precipitación.
- ST_WIND: Estado para medir el viento.

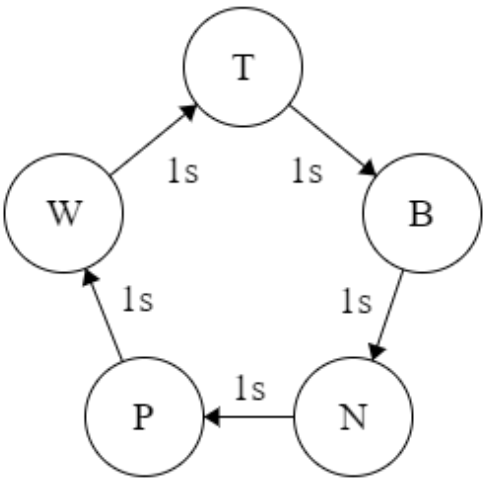

Fig. 6 – FSM diseñada

En cada estado se realizarán las siguientes tres operaciones:

1. Cambiar el canal de ADC1 al correspondiente
2. Medir con el sensor correspondiente
3. Enviar la información a la ESP32

Para cambiar el canal de ADC1 dependiendo del sensor que se esté utilizando, se ha definido una función llamada change_adc_channel(), que dependiendo de un índice que se le pasa como parámetro para indicar el canal a que se tiene que cambiar. Para ello se utiliza la función *HAL_ADC_ConfigChannel* pasándole una estructura de tipo *ADC_ChannelConfTypeDef* con la correspondiente configuración.

```c
/* Change channel of ADC1 */
void change_adc_channel(int index) {
  ADC_ChannelConfTypeDef sConfig;
  switch (index) {
    case 0:
      sConfig.Channel = ADC_CHANNEL_1;
      break;
    case 1:
      sConfig.Channel = ADC_CHANNEL_4;
      break;
    case 2:
      sConfig.Channel = ADC_CHANNEL_11;
      break;
    case 3:
      sConfig.Channel = ADC_CHANNEL_0;
      break;
  }

  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  sConfig.Offset = 0;

  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
    Error_Handler();
  }
}
```

Fig. 7 – Función para cambiar canal de ADC1

Después, se realiza la medición del sensor correspondiente. La implementación de le medición de dichas señales se realiza de diferente manera dependiendo del tipo de sensor. Por ejemplo, para los sensores de viento, sonido y luminosidad se lee el valor del sensor utilizando ADC1 y se convierte dicho valor a un porcentaje entre 0 y 100.

```c
/* Meausure analog brightness and convert */
uint32_t measure_brightness() {
  HAL_ADC_Start(&hadc1);
  HAL_ADC_PollForConversion(&hadc1, 100);
  uint32_t analogValue = HAL_ADC_GetValue(&hadc1);

  int lightPercentage = map(analogValue, 0, 2500, 0, 100);
  printf("(B) Value = %d\r\n", lightPercentage);
  return lightPercentage;
}
```

Fig. 8 – Ejemplo lectura de sensor de luminosidad

Sin embargo, para el sensor de temperatura es diferente, ya que hay que convertir lo que lee el sensor a grados centígrados. Para ello hay que utilizar una fórmula que ha sido extraída de la documentación oficial de Grove.

```c
/* Meausure analog temperature and convert */
uint32_t measure_temperature() {
  HAL_ADC_Start(&hadc1);
  HAL_ADC_PollForConversion(&hadc1, 100);
  int val = HAL_ADC_GetValue(&hadc1);

  float R = 4095.0/val-1.0;
  R = 100000*R;
  int temperature = 1.0/(log(R/100000)/B+1/298.15)-273.15;

  printf("(T) Value = %dºC\r\n", temperature);
  return temperature;
}
```

Fig. 9 – Lectura de sensor de temperatura y conversión

Para el sensor de precipitación simplemente se lee si hay un 0 o un 1 en ese pin, 0 significa que no hay precipitación y 1 que hay precipitación. Además, aparte de leer este sensor en la máquina de estados, también se ejecuta en una rutina de interrupción cuando se pulsa sobre el botón.

```
case ST_PRECIPITATION:
  if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_10)) {
    printf("(P) Value = true\r\n");
    send_info(1, "precipitation");
  } else {
    printf("(P) Value = false\r\n");
    send_info(0, "precipitation");
  }
  next_state = ST_WIND;
```

Fig. 10 – Lectura sensor de precipitación

```
/* ISR change FSM to precipitation state */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_10)) {
    next_state = ST_PRECIPITATION;
  }
}
```

Fig. 11 – Rutina de interrupción del sensor de precipitación

Por último, se envía los datos obtenidos, utilizando una comunicación UART, a la placa ESP32.

```
/* Send info to ESP32 with UART communication */
void send_info(uint32_t val, char * index) {
  char buffer[32];
  sprintf(buffer, "%s:%d", index, val);
  HAL_UART_Transmit(&huart6, buffer, sizeof(buffer), 1000);
  HAL_Delay(1000);
}
```

Fig. 12 – Envío de información por UART a ESP32

4. Código desarrollado para la placa ESP32

La placa ESP32 se encargará en recibir los datos enviados por la STM32 a través de una comunicación UART y enviar esos datos a los diferentes *topics* de MQTT que se han definido.

He definido dos *topics* donde la ESP32 publicará los diferentes datos recibidos:

- Topic "esi/room1/sensors": se publicará los datos referentes a la temperatura y a la luminosidad.
- Topic "esi/room2/sensors": se publicará los datos referentes a la precipitación, sonido y viento.

Primero se establece una configuración para que funcione la comunicación UART en la función *setup()*. Además, esta función también se llama a las funciones de *wifiConnect()* y *mqttConnect()* para establecer la conexión Wifi con el punto de acceso y establecer la conexión con *broker* MQTT.

En mi caso utilizo los pines IO16 y IO17 para recibir y enviar datos, aunque solo se utilizará el IO16, ya que la ESP32 no enviará datos por UART, solo recibirá datos.

```
void setup() {
    Serial.begin(115200);
    const uart_port_t uart_num = UART_NUM_2;
    uart_config_t uart_config = {
        .baud_rate = 115200,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_CTS_RTS,
        .rx_flow_ctrl_thresh = 122,
    };
    // Configure UART parameters
    ESP_ERROR_CHECK(uart_param_config(uart_num, &uart_config));
    ESP_ERROR_CHECK(uart_set_pin(UART_NUM_2, TXD_PIN, RXD_PIN, 18, 19));
    const int uart_buffer_size = (1024 * 2);
    QueueHandle_t uart_queue;
    // Install UART driver using an event queue here
    ESP_ERROR_CHECK(uart_driver_install(UART_NUM_2, uart_buffer_size, \
                                        uart_buffer_size, 10, &uart_queue, 0));
    delay(4000);
    wifiConnect();
    mqttConnect();
}
```

Fig. 11 – Configuración de la placa ESP32

Para recibir los datos enviados por parte de la placa STM32 defino una función llamada *receive_data()* que está constantemente ejecutándose como una tarea. Utilizando la función *uart_get_buffered_data_len()* y la función *uart_read_bytes()*, compruebo y obtengo los datos.

```
/* Receive data from STM32 */
void receive_data(void *pvParameters) {
  for(;;) {
    String data_string;
    const uart_port_t uart_num = UART_NUM_2;
    uint8_t data[128];
    int length = 0;
    ESP_ERROR_CHECK(uart_get_buffered_data_len(uart_num, (size_t*)&length));
    length = uart_read_bytes(uart_num, data, length, 100);
    if (length > 0) {
        data_string = (char*) data;
        Serial.println(data_string);
        send_info_mqtt(data_string);
    }
    vTaskDelay(150/portTICK_PERIOD_MS);
  }
  vTaskDelete(NULL);
}
```

Fig. 12 – Recepción de datos por comunicación UART

En el caso de que se existan datos, entonces, se llama a la función *send_info_mqtt()* para que convierta los datos recibidos a un objeto JSON y lo envíe a *topic* MQTT correspondiente.

```
/* Convert string to json object and send info to mqtt topic */
void send_info_mqtt(String data_char) {
  std::string segment;
  std::stringstream stream(data_char.c_str());
  std::vector<std::string> seglist;

  /* Split the receive string data */
  while(std::getline(stream, segment, ':'))
  {
    seglist.push_back(segment);
  }

  StaticJsonBuffer<300> JSONbuffer;
  JsonObject& JSONencoder = JSONbuffer.createObject();
  JSONencoder[seglist.at(0).c_str()] = atoi(seglist.at(1).c_str());
  char JSONmessageBuffer[100];
  JSONencoder.printTo(JSONmessageBuffer, sizeof(JSONmessageBuffer));

  JSONencoder.printTo(JSONmessageBuffer, sizeof(JSONmessageBuffer));
  if (JSONencoder.containsKey("temperature") || JSONencoder.containsKey("brightness")) {
    Serial.println("Send to topic 1");
    client.publish(TOPIC1, JSONmessageBuffer);
  } else {
    Serial.println("Send to topic 2");
    client.publish(TOPIC2, JSONmessageBuffer);
  }
}
```

Fig. 13 – Convertir datos a JSON y enviar al topic MQTT

5. Configuración de dashboard en Grafana

En el *dashboard* de *Grafana* se ha configurado los diferentes medidores para los 5 sensores. En este caso se ha utilizado 4 medidores *gauge* para la temperatura, viento, luminosidad y ruido. Para la precipitación se ha utilizado un medidor de tipo *state-timeline* para mostrar si hay precipitaciones o no.

Fig. 12 – Dashboard con los diferentes medidores

La estación meteorológica quedaría tal que así:

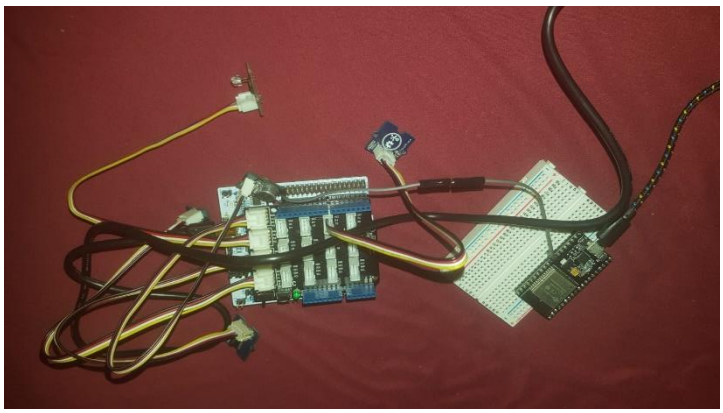

Fig. 13 – Imagen del circuito