# Flujo del código

Programa para dispositivo ESP8266 cuya función es realizar la adquisición de datos de temperatura y humedad utilizando sensores y enviar los datos adquiridos a través de ESP-NOW. Vamos a analizarlo por secciones, dividiéndolo en 6 partes:

## Declaración de librerías.

En esta sección se incluyen las librerías necesarias para el programa. Estas librerías proporcionarán las funciones y clases necesarias para la comunicación con los sensores, la conexión a redes WiFi y la actualización del firmware a través de OTA (Over-the-Air).

/\*LIBRERIAS\*/

#include "AUTOpairing.h" //Es una librería creada para realizar el emparejamiento automático de múltiples dispositivos ESP8266 con un ESP32 que actúa como servidor.

#include <ArduinoJson.h> //Librería para la manipulación de datos JSON.

#include <ESP8266httpUpdate.h> //Librería para realizar la actualización de firmware vía OTA.

#include <ESP8266HTTPClient.h> //Librería utilizada para realizar solicitudes HTTP.

#include <ESP8266WiFi.h> //Librería para realizar conexiones a redes wifi.

#include <OneWire.h> // Librería para comunicación con sensores de temperaturas DS18B20 multiplexados por un solo cable.

#include <DallasTemperature.h> //Librería para la obtención de valores de temperaturas.

#include "funciones.h" //Librería con funciones personalizadas para procesar las medidas de los sensores, creada por el alumno.

## Declaración de pines de alimentación de los sensores, variables, estructura y objeto.

En primer lugar, se definen los pines para controlar la alimentación de los sensores de temperatura y humedad. A continuación, se declaran variables para almacenar los valores de humedad y de temperatura (las variables S1\_T, S2\_T y S3\_T representan las temperaturas individuales de cada sensor, mientras que el array temp se utiliza para almacenar los valores de temperatura de los tres sensores).

Se instancia un objeto de la clase AUTOpairing\_t llamado clienteAP para manejar la comunicación a través de ESP-NOW.

Se define una estructura llamada configuración que tiene dos campos sleep y timeout, ambos de tipo uint32\_t. La estructura se utiliza para almacenar la configuración del programa, definida por el tiempo de sueño profundo y el tiempo de espera de conexión.

/\*PINES\*/

const int Power\_s\_temp = 12; // Pin GPIO para alimentar los sensores de temperatura

const int Power\_s\_hum = 13; // Pin GPIO para alimentar sensor de humedad

/\*VARIABLES\*/

float S1\_T=0, S2\_T=0, S3\_T=0;

float temp[3];

float humedad;

AUTOpairing\_t clienteAP; // objeto de la clase AUTOpairing\_t para la comunicación por ESP-NOW.

// Estructura utilizada para almacenar la configuración del programa, (tiempo de sueño profundo y el tiempo de espera de conexión).

struct configuracion

{

  uint32\_t sleep;

  uint32\_t timeout;

} mi\_configuracion;

## Declaración de variables para la actualización vía FLOTA (Firmware Over-the-Air).

Aquí se definen variables relacionadas con la actualización del firmware a través de FOTA. Se define el nombre de la red WiFi (ssid) y la contraseña (password) para la conexión, la URL para la actualización del firmware (OTA\_URL) y la versión del firmware (HTTP\_OTA\_VERSION). Además, se crea un objeto WiFiClientSecure llamado WClient para la conexión segura con el servidor.

//--------------------- ACTUALIZACION FOTA ----------------------

const char\* ssid = "huerticawifi";

const char\* password = "4cc3sshu3rt1c4";

#define OTA\_URL    "https://huertociencias.uma.es/esp8266-ota-update"

#define HTTP\_OTA\_VERSION String(\_\_FILE\_\_).substring(String(\_\_FILE\_\_).lastIndexOf('\\')+1) + ".d1\_mini\_pro"

WiFiClientSecure WClient;

## Declaración de la función procesa mensaje.

Se declara la función procesa\_mensajes es llamada cuando se reciben mensajes a través de ESP-NOW. dependiendo del tema del mensaje, se realiza la actualización del firmware o se configuran parámetros. Se encarga de procesar los mensajes recibidos a través de ESP-NOW. Toma dos argumentos: topic, que representa el tema del mensaje y payload, que contiene los datos del mensaje.

Es una función compleja que se explicará paso a paso:

* Imprime información sobre el mensaje recibido, como el tema (topic) y la carga útil (payload).
* Comprueba el tema del mensaje utilizando una estructura de control if-else. Hay dos posibles temas:
  + Si el tema es "fota", significa que se recibió un mensaje para realizar una actualización de firmware. En este caso, corre el código de actualización FOTA.
    - Realiza la conexión wifi utilizando la librería <ESP8266WiFi.h>. Si no conecta transcurrido le tiempo establecido, el nodo pasa a estado de sueño profundo, si conecta, a través de la clase WClient realiza la actualización llamando a la función ESPhttpUpdate.update(WClient,OTA\_URL, HTTP\_OTA\_VERSION).
    - Si no realiza la conexión el nodo entra en sueño profundo.
  + Si el tema es "config", significa que se recibió un mensaje para actualizar la configuración del dispositivo. En este caso, ejecuta el código correspondiente para realizar la configuración.
    - Se crea un StaticJsonDocument con un tamaño adecuado para cargar el mensaje JSON recibido.
    - Se deserializa el mensaje JSON usando deserializeJson para convertir la carga útil en un documento JSON manejable.
    - Se comprueba si la deserialización fue exitosa.
    - Se verifica si el documento JSON contiene los campos "sleep" y "timeout". Si están presentes, significa que se recibió una configuración válida.
    - Se extraen los valores de "sleep" y "timeout" del documento JSON y se almacenan en la estructura de configuración mi\_configuracion.
    - Finalmente, se llama al método set\_config del objeto clienteAP para actualizar la configuración con los nuevos valores.

void procesa\_mensajes (String topic, String payload)

{

  Serial.println("Mensaje recibido...");

  Serial.print("Topic: ");

  Serial.println(topic);

  Serial.print("Payload: ");

  Serial.println(payload);

  if(topic=="fota")

  {

    delay(100);

    Serial.println();

    Serial.print("Conectando a WiFi con SSID: ");

    Serial.println(ssid);

    //wifi\_promiscuous\_enable(0);

    WiFi.disconnect();

    WiFi.mode(WIFI\_STA);

    WiFi.begin(ssid, password);

    unsigned long ahora= millis();

    while (WiFi.status() != WL\_CONNECTED && millis()-ahora<15000) {

      delay(100);

      Serial.print(".");

    }

    if(WiFi.status() != WL\_CONNECTED)

    {

      Serial.println("WiFi NO conectado");

      clienteAP.gotoSleep();

    }

    Serial.println("");

    Serial.println("WiFi conectado");

    Serial.println("Direccion IP: ");

    Serial.println(WiFi.localIP());

    WClient.setTimeout(12); // timeout argument is defined in seconds for setTimeout

    WClient.setInsecure();  // no chequea certificado...

    Serial.println( "Intentando la actualización FOTA..." );

    Serial.println(OTA\_URL);

    Serial.println(HTTP\_OTA\_VERSION);

    switch(ESPhttpUpdate.update(WClient,OTA\_URL, HTTP\_OTA\_VERSION)) {

    case HTTP\_UPDATE\_FAILED:

      Serial.printf("Error en actualización HTTP: Error #(%d): %s\n", ESPhttpUpdate.getLastError(), ESPhttpUpdate.getLastErrorString().c\_str());

      break;

    case HTTP\_UPDATE\_NO\_UPDATES:

      Serial.println(F("No hay nueva actualiación"));

      break;

    case HTTP\_UPDATE\_OK:

      Serial.println(F("Actualiación OK"));

      break;

    }

    clienteAP.gotoSleep();

  } // end FOTA

  if(topic=="config") // el mesaje debe tener topic = config y payload = {"sleep": # , "timeout": # }

  {

    StaticJsonDocument<512> doc; // el tamaño tiene que ser adecuado para el mensaje

    // Deserialize the JSON document

    DeserializationError error = deserializeJson(doc, payload.c\_str(),payload.length());

    // Compruebo si no hubo error

    if (error) {

      Serial.print("Error fallo en deserializeJson(): ");

      Serial.println(error.c\_str());

    }

    else

    if(doc.containsKey("sleep") && doc.containsKey("timeout") )  // comprobar si existe el campo/clave que estamos buscando

    {

     int valor = doc["sleep"];

     Serial.print("JSON sleep = ");

     Serial.println(valor);

     mi\_configuracion.sleep=valor;

     valor = doc["timeout"];

     Serial.print("JSON timeout = ");

     Serial.println(valor);

     mi\_configuracion.timeout=valor;

     clienteAP.set\_config((uint8\_t\*)&mi\_configuracion);

    }

    else

    {

      Serial.println("ERROR: claves \"sleep\" & \"timeout\" no aparecen en JSON");

    }

  } // end config

}

## Entrada en el bucle de Setup.

El código de este bucle se encarga de configurar los pines, iniciar la comunicación, establecer valores de configuración y realiza la inicialización de la comunicación ESP-NOW, y realizar otros ajustes iniciales necesarios para el funcionamiento del dispositivo. Paso por paso:

* Se configura la alimentación de los sensores de temperatura y humedad mediante los pines Power\_s\_temp y Power\_s\_hum. Se establece su modo como salida y se encienden inicialmente.
* Se inicia la comunicación serie (puerto serie) con una velocidad de transmisión de 115200 baudios.
* Se imprimen mensajes en la consola para indicar que la configuración está comenzando.
* Se inicializa el tamaño de la configuración del objeto clienteAP para que coincida con el tamaño de la estructura mi\_configuracion.
* Se verifica si hay una configuración previamente guardada utilizando el método get\_config de la clase clienteAP. Si no hay configuración guardada, se establecen valores por defecto para sleep y timeout en la estructura mi\_configuracion.
* Se imprimen en la consola los valores de sleep y timeout obtenidos o establecidos.
* Se configuran parámetros de la clase clienteAP utilizando métodos:
  + set\_timeOut, establece el tiempo máximo de conexión.
  + set\_deepSleep, establece el tiempo en sueño profundo (en segundos).
  + set\_channel, set\_debug, establece el canal donde empieza el escaneo de dispositivos en la red wifi para el autoemparejamiento.
  + set\_callback, llama a la función procesa\_mensajes.
* Se inicia la comunicación a través de ESP-NOW llamando al método begin de clienteAP. El método begin se explica al final del documento, ya que es una de las funciones principales del proyecto para autoemparejar a los dispositivos.
* Se realiza una adquisición de las direcciones de los sensores de temperatura llamando a la función adquisicion\_direcciones\_temp (función perteneciente a la librería funciones.cpp).

//-----------------------   SETUP   ---------------------------------

void setup() {

  // Iniciar alimentación de sensores

  pinMode(Power\_s\_temp, OUTPUT);

  pinMode(Power\_s\_hum, OUTPUT);

  digitalWrite(Power\_s\_temp, HIGH);

  digitalWrite(Power\_s\_hum, HIGH);

  // Iniciar puerto serie

  Serial.begin(115200);

  Serial.println();

  Serial.println("SETUP...");

  clienteAP.init\_config\_size(sizeof(mi\_configuracion));

// Lee configuración de la memoria.

  if (clienteAP.get\_config((uint8\_t\*)&mi\_configuracion)==false)

  { // si no hay configuración guardada la pongo por defecto

    mi\_configuracion.sleep=10;

    mi\_configuracion.timeout=3000;

  }

  Serial.printf(" > DeepSleep : %d\n",mi\_configuracion.sleep );

  Serial.printf(" > TimeOut   : %d\n",mi\_configuracion.timeout );

  clienteAP.set\_timeOut(mi\_configuracion.timeout,true); // tiempo máximo

  clienteAP.set\_deepSleep(mi\_configuracion.sleep);  //tiempo dormido en segundos

  clienteAP.set\_channel(6);  // canal donde empieza el scaneo

  clienteAP.set\_debug(true);   // depuración, inicializar Serial antes

  clienteAP.set\_callback(procesa\_mensajes);  //Revisa si tiene algún mensaje de configuración o actualización en cola

    clienteAP.begin();

  adquisicion\_direcciones\_temp();

}

## Entrada en el bucle principal.

En el bucle, se mantiene la conexión con AUTOpairing y se comprueba si hay mensajes disponibles de actualización o configuración del dispositivo. A continuación, obtienen los datos de temperaturas y humedad de los sensores, una vez recibidos deja de alimentar a los sensores y se crea un mensaje con los datos en formato JSON. Luego se utiliza en método espnow\_send\_check() de la clase clienteAP para enviar el mensaje a través de ESP-NOW. Después de enviar los datos, el dispositivo entra en modo sueño profundo (deepsleep) hasta que se active nuevamente transcurrido en el tiempo fijado por configuración.

//-----------------------   LOOP    ---------------------------------

void loop() {

  //digitalWrite(Power\_s\_temp, HIGH);

  //digitalWrite(Power\_s\_hum, HIGH);

  clienteAP.mantener\_conexion();

  if (clienteAP.envio\_disponible()) {

      char mensaje[256];

      pedir\_temperaturas(temp);

      digitalWrite(Power\_s\_temp, LOW); // apagamos los sensores de temperatura

      humedad=pedir\_humedad();

      digitalWrite(Power\_s\_hum, LOW);  // apagamos sensor de humedad

      sprintf(mensaje, "{\"topic\":\"datos\",\"T1\":%4.2f,\"T2\":%4.2f,\"T3\":%4.2f, \"hum\":%4.2f }", temp[0], temp[1], temp[2], humedad);

      clienteAP.espnow\_send\_check(mensaje); // hará deepsleep por defecto

  }

}

# Función begin() perteneciente a la clase AUTOpairing\_t de la librería AUTOpairing.h .

Esta función se encarga de iniciar el proceso de emparejamiento automático (AUTOpairing) utilizando el protocolo ESP-NOW. La explicación del flujo del código es el siguiente:

* Se verifica si se ha habilitado el modo de depuración (debug) y se imprime un mensaje indicando que comienza el proceso de emparejamiento automático (AUTOpairing) y se inicializan variables relacionadas con el tiempo (previousMillis\_scanChannel y start\_time).
* Se realiza la inicialización de la memoria EEPROM o RTC MEM según la configuración definida en usar\_FLASH, en nuestro caso se trabaja con RTC, luego, se utiliza RTC MEM, se lee directamente desde la memoria RTC.
* Se verifica si el valor code1 en la estructura rtcData coincide con el valor MAGIC\_CODE1. Si coinciden, significa que hay datos de emparejamiento almacenados en la memoria y se procede a recuperar esos datos.
* Se copian los datos de emparejamiento (pairingData) desde la memoria a la estructura local.
* Se imprimen mensajes informativos sobre la recuperación de datos de emparejamiento, incluyendo la dirección MAC del dispositivo emparejado y el canal utilizado.
* Se configura el dispositivo como una estación Wi-Fi (WiFi.mode(WIFI\_STA)).
* Se habilita el modo promiscuo (wifi\_promiscuous\_enable(1)) para capturar paquetes.
* Se establece el canal Wi-Fi en el valor especificado en pairingData.channel utilizando wifi\_set\_channel().
* Se deshabilita el modo promiscuo después de configurar el canal (wifi\_promiscuous\_enable(0)).
* Se desconecta cualquier conexión Wi-Fi existente utilizando WiFi.disconnect().
* Se inicializa el protocolo ESP-NOW mediante esp\_now\_init(). Si hay un error en la inicialización, se imprime un mensaje de error y la función termina.
* Se establece el rol de ESP-NOW como "combo" (ESP\_NOW\_ROLE\_COMBO) mediante esp\_now\_set\_self\_role().
* Se registran las funciones de devolución de llamada para recibir datos (OnDataRecv) y para enviar datos (OnDataSent) A través de ESP-NOW.
* Se agrega el dispositivo emparejado como peer en la lista de peers de ESP-NOW utilizando esp\_now\_add\_peer().
* Se establece el estado de emparejamiento como "emparejado" mediante la variable pairingStatus.

void begin()

  {

    if(debug) Serial.println("\nComienza AUTOpairing...");

    previousMillis\_scanChannel=0;

    start\_time=millis();

    // Inicialización de la memoria EEPROM o RTC MEM según la configuración

    // usamos rtc memory

    if(usar\_FLASH)

    {

      if(!EEPROM\_init) { EEPROM.begin(sizeof(rtcData)); EEPROM\_init=true; }

      EEPROM.get(0, rtcData);

      if(debug) Serial.print("Magic code en FLASH: ");

    }

    else

    {

      ESP.rtcUserMemoryRead(0,(uint32\_t \*)&rtcData, sizeof(rtcData));

      if(debug) Serial.print("Magic code en RTC MEM: ");

    }

    if(debug) Serial.print(rtcData.code1,HEX);

    if(debug) Serial.print(" - esperando: ");

    if(debug) Serial.println((unsigned long)MAGIC\_CODE1,HEX);

    if(rtcData.code1==MAGIC\_CODE1) //Si coinciden, significa que hay datos de emparejamiento almacenados en la memoria y se procede a recuperar esos datos.

    {

      memcpy(&pairingData, &(rtcData.data), sizeof(pairingData));

      if(debug) Serial.print("Emparejamiento recuperado de la memoria RTC del usuario ");

      printMAC(pairingData.macAddr);

      if(debug) Serial.print(" en el canal " );

      if(debug) Serial.print(pairingData.channel);    // channel used by the server

      // Imprime el tiempo transcurrido desde el inicio del proceso de emparejamiento

      if(debug) Serial.print(" en ");

      if(debug) Serial.print(millis()-start\_time);

      if(debug) Serial.println("ms");

    // Set device as a Wi-Fi Station

      WiFi.mode(WIFI\_STA);

      if(debug) Serial.print(" MI DIRECCIÓN MAC: ");

      if(debug) Serial.println(WiFi.macAddress());

      wifi\_promiscuous\_enable(1);

      wifi\_set\_channel(pairingData.channel);

      wifi\_promiscuous\_enable(0);

      WiFi.disconnect();

      // Init ESP-NOW

      if (esp\_now\_init() != 0) {

        if(debug) Serial.println("Error al inicializar ESP-NOW");

        return;

      }

      // Set ESP-NOW Role

      esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_COMBO);

      // Register for a callback function that will be called when data is received

      esp\_now\_register\_recv\_cb(AUTOpairing\_t::OnDataRecv);

      esp\_now\_register\_send\_cb(AUTOpairing\_t::OnDataSent);

      esp\_now\_add\_peer(pairingData.macAddr, ESP\_NOW\_ROLE\_COMBO, pairingData.channel, NULL, 0); // add the server to the peer list

      pairingStatus = PAIR\_PAIRED ;            // set the pairing status

    }

}