**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:

* Declaración de librerías.

En esta sección se incluyen las librerías necesarias para el programa, como AUTOpairing.h, ArduinoJson.h, ESP8266httpUpdate.h, ESP8266HTTPClient.h, ESP8266WiFi.h, OneWire.h, DallasTemperature.hy funciones.h. 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 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.

* Declaración de pines de alimentación de los sensores.

/\*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

* Declaración de variables.

Aquí se declaran variables para almacenar los valores de humedad y de temperatura de tres sensores. 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.

/\*VARIABLES\*/

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

float temp[3];

float humedad;

* Declaración de estructura de datos.

Se define una estructura llamada configuración que tiene dos campos sleepy 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.

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\_mensajeses 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:

void procesa\_mensajes (String topic, String payload)

{

//Inicialmente imprime el topic y el mensaje recibido a través de la pasarela ESP-NOW

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

  Serial.print("Topic: ");

  Serial.println(topic);

  Serial.print("Payload: ");

  Serial.println(payload);

//Si el topic es fota, se procede a la actualización del Firmware.

  if(topic=="fota")

  { //Inicia la conexión wifi, si no conecta activa el sueño profundo.

    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");

      AUTOpairing::gotoSleep();

    }

//Si tiene éxito y conecta, intenta la actualización.

    Serial.println("");

    Serial.println("WiFi conectado");

    Serial.println("Direccion IP: ");

    Serial.println(WiFi.localIP());

    WClient.setTimeout(12); // tiempo de espera se define en segundos para setTimeout

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

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

    Serial.println(OTA\_URL);

    Serial.println(HTTP\_OTA\_VERSION);

    //Chequea las 3 opciones, fallo de actualización, no hay nueva versión y actualizado. Tras analizar el estado activa el sueño profundo.

    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;

    }

    AUTOpairing::gotoSleep();

  } // end FOTA

// Si el topic es config, intenta deserializar el contenido del mensaje JSON (payload) utilizando la biblioteca ArduinoJson. Si la deserialización es exitosa y el JSON contiene las claves "sleep" y "timeout", se extraen sus valores correspondientes y se almacenan en la estructura mi\_configuracion. Finalmente, se llama a la función AUTOpairing::set\_configpara establecer la configuración con los nuevos valores.

  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;

     AUTOpairing::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.

Aquí es donde empieza el programa y se configuran los parámetros necesarios. En resumen, se inicializan los pines, el puerto serie y se configuran los parámetros de AUTOpairing (Es una librería creada para establezer la configuración y emparejamiento de dispositivos ESP8266 a través de ESP-NOW). Se obtiene la configuración guardada (si existe) o se establecen valores por defecto.

//-------------------------------------------------------------------

//-----------------------   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...");

  adquisicion\_direcciones\_temp();

  AUTOpairing::init\_config\_size(sizeof(mi\_configuracion));

  if (AUTOpairing::get\_config((uint8\_t\*)&mi\_configuracion)==false)

  {

// si no hay configuración guardada se pone por defecto

    mi\_configuracion.sleep=10; //Establece la duración de la suspensión profunda en la clase AUTOpairing en 10 segundos. Esto significa que el dispositivo se mantendrá en modo de bajo consumo durante ese período de tiempo antes de reanudar la ejecución normal.

    mi\_configuracion.timeout=3000; //Establece el tiempo máximo para establecer conexión o enviar un mensaje.

  }

//Se imprimen los valores de configuración por pantalla.

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

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

  AUTOpairing::set\_timeOut(mi\_configuracion.timeout,true); //Configura el tiempo máximo para establecer conexión o enviar un mensaje.

  AUTOpairing::set\_deepSleep(mi\_configuracion.sleep);  //Configura tiempo dormido en segundos

  AUTOpairing::set\_channel(6);  // canal donde empieza el escaneo

  AUTOpairing::set\_debug(true);   // depuración, inicializar Serial antes

  AUTOpairing::set\_callback(procesa\_mensajes);  //Llamamos a la función procesa mensajes para saber cómo actuar.

//Finalmente, se llama a la función "begin()" de la clase "AUTOpairing" para iniciar la comunicación ESP-NOW en el arranque.

  AUTOpairing::begin();

}

* Entrada en el bucle principal.

Aquí se mantiene la conexión con AUTOpairing y se comprueba si hay datos disponibles para ser enviados. Si hay datos disponibles, se obtienen las temperaturas y la humedad de los sensores, se apagan los sensores y se crea un mensaje con los datos en formato JSON. Luego se utiliza la función espnow\_send\_check() de AUTOpairing para enviar el mensaje a través de ESP-NOW. Después de enviar los datos, el dispositivo entra en modo de bajo consumo (deepsleep) hasta que se active nuevamente transcurrido en el tiempo fijado por configuración.