

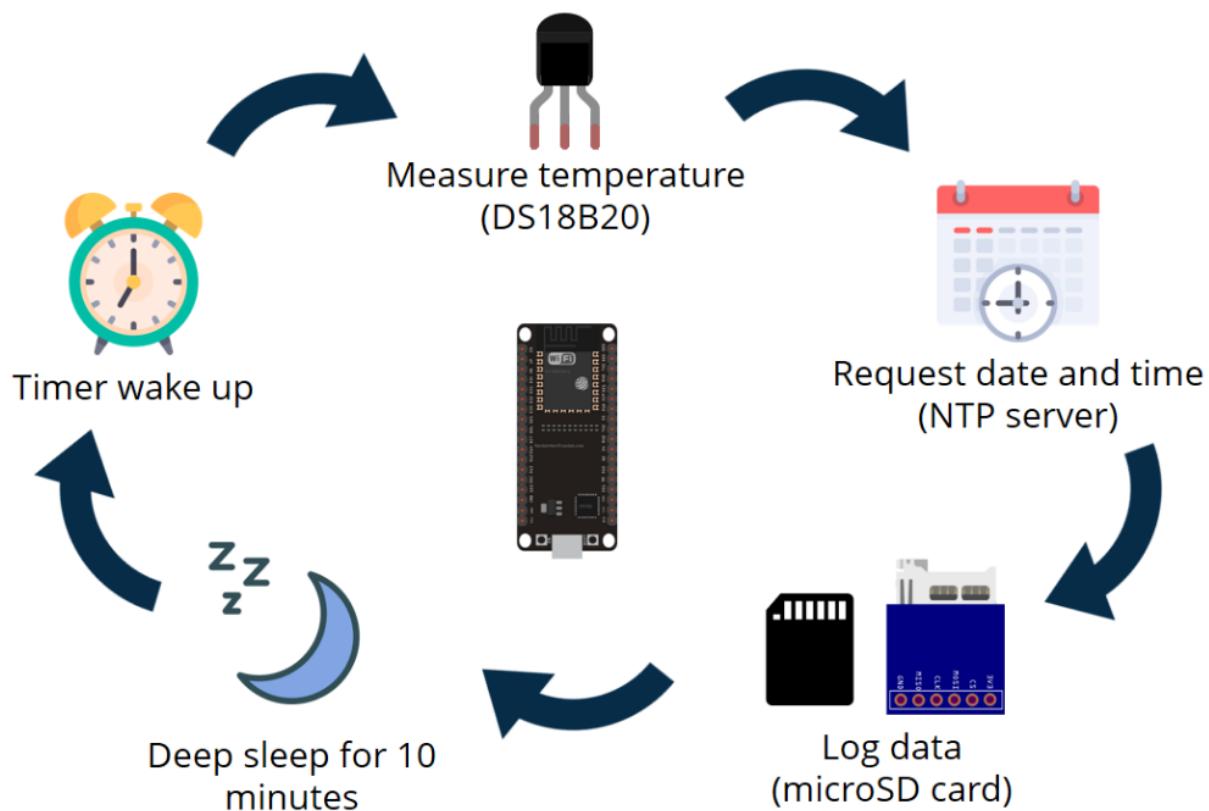
PROJECTE PROCESSADORS DIGITALS:

Carregar les dades del sensor de temperatura dins la targeta MicroSD mitjançant l'ESP32

1. INTRODUCCIÓ

En Gerard Vila i jo, Juan C. De Los Ríos, hem escollit fer aquest projecte on hem mostrat com registrar dades amb marques de temps, en una targeta microSD mitjançant l'ESP32. Hem registrat les lectures de temperatura del sensor DS18B20 cada 10 minuts. L'ESP32 s'ha estat en mode de repòs profund entre cada lectura i ha sol·licitat la data i l'hora mitjançant Network Time Protocol (NTP).

Primer de tot, farem una descripció general del projecte, on hi destaquem les **principals característiques**:



- Primer de tot, l'ESP32 llegeix la temperatura mitjançant el sensor de temperatura DS18B20.
- Després d'obtenir la temperatura, fa una sol·licitud a un servidor NTP (Network Time Protocol) per obtenir la data i l'hora. Per tant, l'ESP32 necessita una connexió Wi-Fi.
- Les dades (temperatura i marca de temps) se'n registren a una targeta microSD.
- Per registrar dades a la targeta microSD, hem fet servir un mòdul de targeta microSD.
- Després de completar aquestes tasques anteriors, l'ESP32 dorm durant 10 minuts i a continuació es desperta i repeteix el procés.

Aquí llistem les parts necessàries per construir aquest projecte:

- ESP32 DOIT DEVKIT V1



- Mòdul de targeta MicroSD



- Targeta MicroSD



- Sensor de temperatura DS18B20



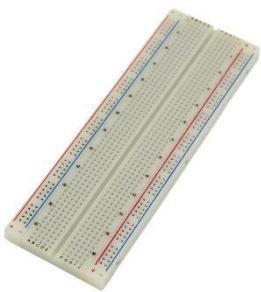
- Resistència de 10 kOhm



- Filferros de pont



- “Protoboard”

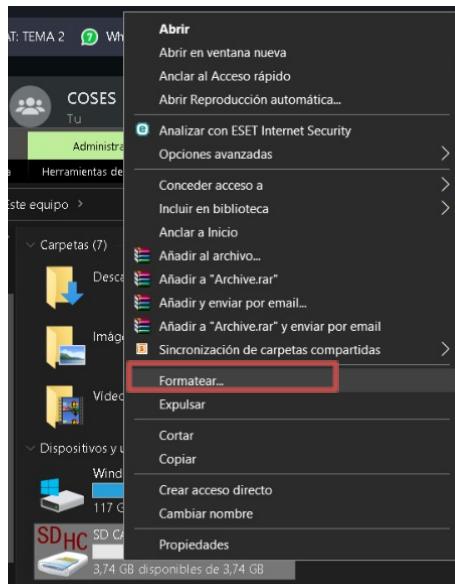


Per desar dades a la targeta microSD amb l'ESP32, hem fet servir el mòdul de targeta microSD que es comunica amb l'ESP32 mitjançant el protocol de comunicació SPI.

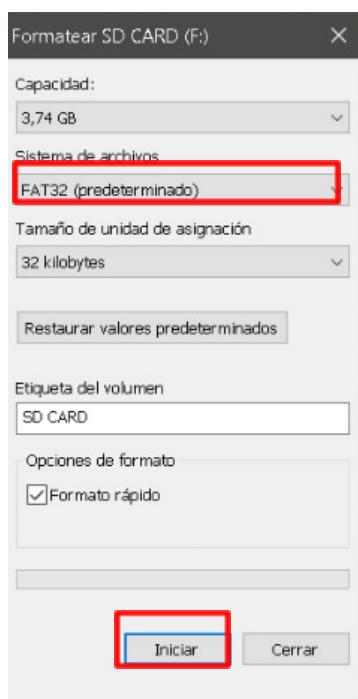
Formatejar la targeta microSD

Quan s'utilitza una targeta microSD amb l'ESP32, primer s'ha de formatejar. A continuació fiquem les instruccions per dur-ho a terme:

1. Primer s'introduceix la targeta microSD a l'ordinador. Dins “**El meu equip**” fem clic amb el botó dret sobre la targeta SD i seleccionem “**Formatear...**”.

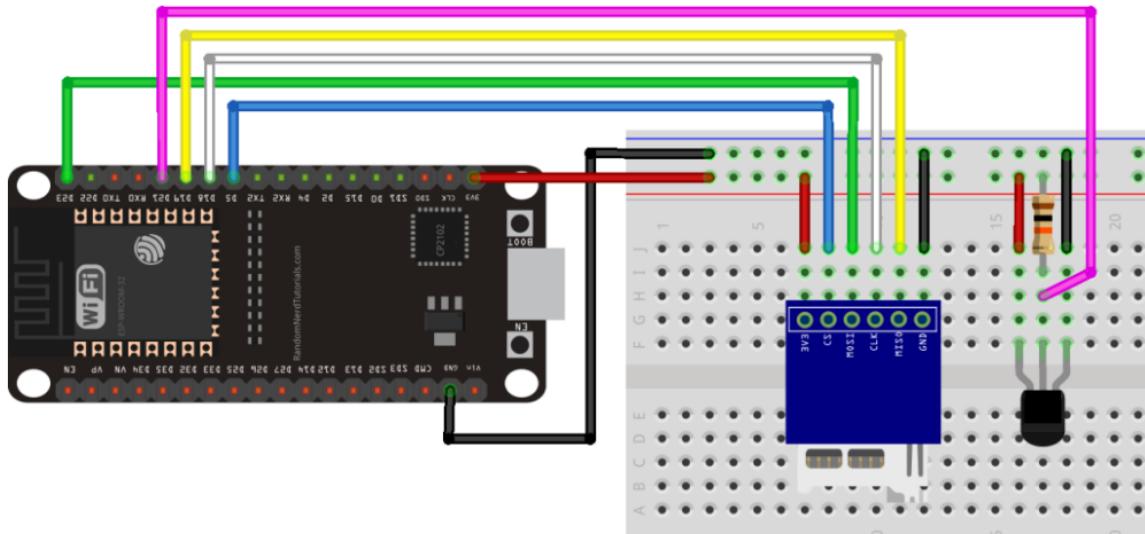


2. Apareix una finestra nova. Seleccionem **FAT32**, premem **Iniciar** per inicialitzar el procés de format i seguim les instruccions en pantalla.



2. ESQUEMA DE MUNTATGE

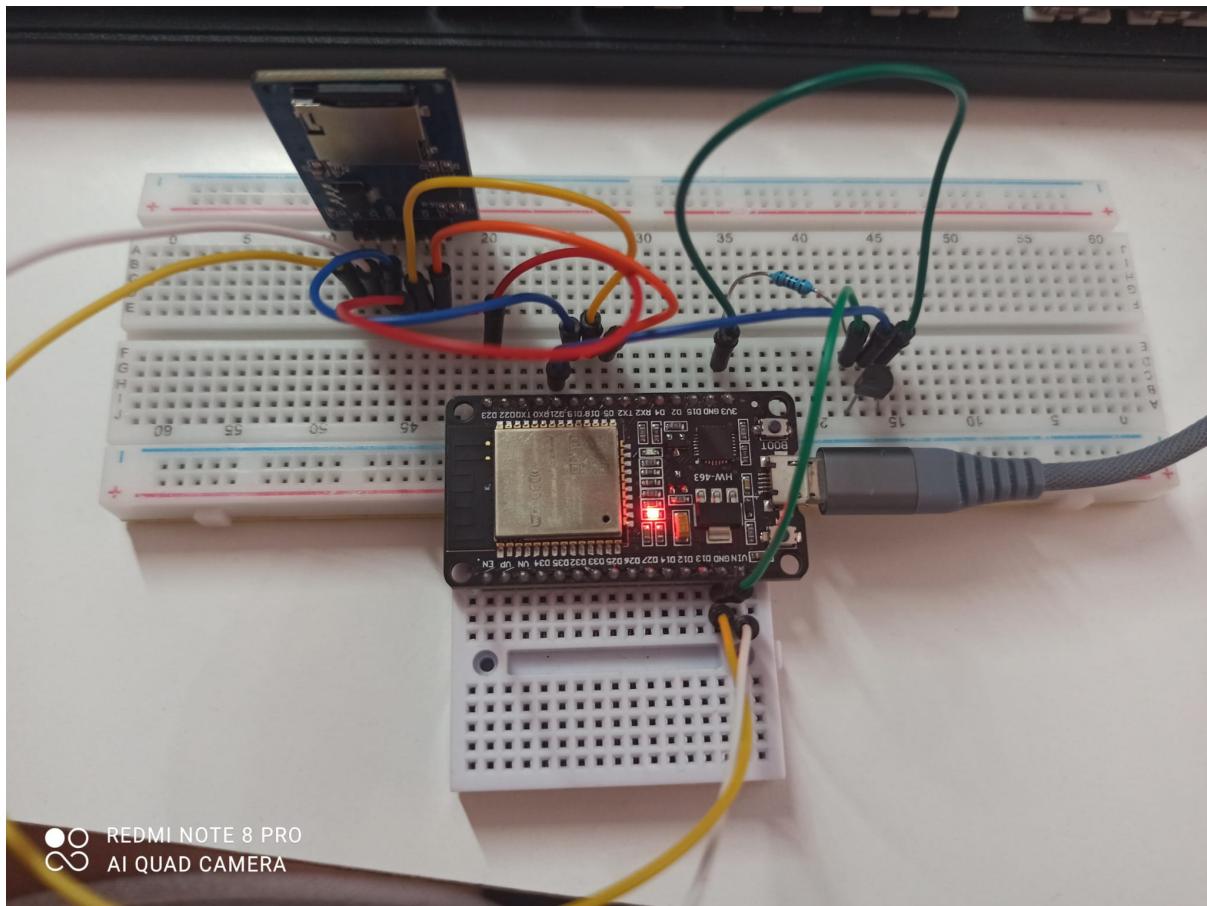
Hem seguit el següent diagrama esquemàtic per muntar el circuit d'aquest projecte. Tot i que l'hem hagut de modificar una mica ja que el nostre adaptador de targeta microSD funciona amb l'alimentació Vin(5V) i no amb la de 3V3. Pel que fa la resta tot queda igual.



Aquesta taula mostra la manera que hem connectat els pins:

| ADAPTADOR MICROSD | ESP32 |
|---------------------|---------|
| VCC | VIN |
| CS | GPIO 5 |
| MOSI | GPIO 23 |
| SCK | GPIO 18 |
| MISO | GPIO 19 |
| GND | GND |
| SENSOR TEMP DS18B20 | ESP32 |
| VIN | 3V3 |
| DATA WIRE | GPIO 21 |
| GND | GND |

La següent foto mostra com ha quedat el nostre circuit:



Preparació de l'IDE Arduino

Hi ha un complement per a l'IDE Arduino que ens permet programar l'ESP32 mitjançant l'IDE Arduino i el seu llenguatge de programació.

Instal·lació de biblioteques

Abans de penjar el codi, hem d'instal·lar algunes biblioteques al nostre IDE Arduino. La biblioteca OneWire de Paul Stoffregen i la biblioteca Dallas Temperature, de manera que podem utilitzar el sensor DS18B20. També instal·larem la biblioteca NTPClient bifurcada per Taranais per fer una sol·licitud a un servidor NTP.

Hem utilitzat aquests links per instal·lar les biblioteques al nostre IDE Arduino:

Biblioteca OneWire:

<https://github.com/PaulStoffregen/OneWire/archive/master.zip>

Dallas Temperature library:

<https://github.com/milesburton/Arduino-Temperature-Control-Library/archive/master.zip>

NTPClient library:

<https://github.com/taranais/NTPClient/archive/master.zip>

3. CODI I FUNCIONAMENT

Aquí està el codi original que hem utilitzat per a la programació del nostre projecte.

```
#include "FS.h"
#include "SD.h"
#include <SPI.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <WiFi.h>
#include <NTPClient.h>
#include <WiFiUdp.h>

uint64_t uS_TO_S_FACTOR = 1000000;
uint64_t TIME_TO_SLEEP = 600;

const char* ssid      = "NOM WIFI";
const char* password = "CONTRASENYA WIFI";

#define SD_CS 5

RTC_DATA_ATTR int readingID = 0;

String dataMessage;

#define ONE_WIRE_BUS 21
OneWire oneWire(ONE_WIRE_BUS);

DallasTemperature sensors(&oneWire);
float temperature;
```

```
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP);

String formattedDate;
String dayStamp;
String timeStamp;

void getReadings() {
    sensors.requestTemperatures();
    temperature = sensors.getTempCByIndex(0);
    Serial.print("Temperatura: ");
    Serial.println(temperature);
}

void getTimeStamp() {
    while(!timeClient.update()) {
        timeClient.forceUpdate();
    }
    formattedDate = timeClient.getFormattedDate();
    Serial.println(formattedDate);

    int splitT = formattedDate.indexOf("T");
    dayStamp = formattedDate.substring(0, splitT);
    Serial.println(dayStamp);
    timeStamp=formattedDate.substring(splitT+1, formattedDate.length()-1);
    Serial.println(timeStamp);
}

void logSDCard() {
    dataMessage=String(readingID) + "," + String(dayStamp) + "," +
String(timeStamp) + "," +
        String(temperature) + "\r\n";
    Serial.print("Save data: ");
    Serial.println(dataMessage);
    appendFile(SD, "/data.txt", dataMessage.c_str());
}

void writeFile(fs::FS &fs, const char * path, const char * message) {
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file) {
        Serial.println("Failed to open file for writing");
        return;
    }
    if(file.print(message)) {
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
}
```

```
        }
    file.close();
}

void appendFile(fs::FS &fs, const char * path, const char * message) {
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file) {
        Serial.println("Failed to open file for appending");
        return;
    }
    if(file.print(message)) {
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
    file.close();
}

void setup() {
    Serial.begin(115200);

    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");

    Serial.println("WiFi connected.");

    timeClient.begin();
    // Set offset time in seconds to adjust for your timezone, for
example:
    // GMT +1 = 3600
    // GMT +8 = 28800
    // GMT -1 = -3600
    // GMT 0 = 0
    timeClient.setTimeOffset(3600);

    SD.begin(SD_CS);
    if(!SD.begin(SD_CS)) {
        Serial.println("Card Mount Failed");
        return;
    }
```

```
uint8_t cardType = SD.cardType();
if(cardType == CARD_NONE) {
    Serial.println("No SD card attached");
    return;
}
Serial.println("Initializing SD card...");
if (!SD.begin(SD_CS)) {
    Serial.println("ERROR - SD card initialization failed!");
    return; // init failed
}

File file = SD.open("/data.txt");
if(!file) {
    Serial.println("File doesn't exist");
    Serial.println("Creating file...");
    writeFile(SD, "/data.txt", "Reading ID, Date, Hour, Temperature
\r\n");
}
else {
    Serial.println("File already exists");
}
file.close();

esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);

sensors.begin();

getReadings();
getTimeStamp();
logSDCard();

readingID++;

Serial.println("DONE! Going to sleep now.");
esp_deep_sleep_start();
}

void loop() {
```

COM FUNCIONA EL CODI

En aquest cas, l'ESP32 es troba en mode de son profund entre cada lectura. En mode de repòs profund, tot el nostre codi hauria d'anar a la funció setup (), perquè l'ESP32 mai arriba al loop().

Cal remarcar que l'ordre d'algunes de les funcions, no són iguals, per temes de compilació dins el Platformio.

Importació de biblioteques

En primer lloc, importem les biblioteques necessàries per al mòdul de la targeta microSD:

```
#include "FS.h"  
#include "SD.h"  
#include <SPI.h>
```

Importem aquestes biblioteques per treballar amb el sensor de temperatura DS18B20.

```
#include <OneWire.h>  
#include <DallasTemperature.h>
```

Les biblioteques següents ens permeten sol·licitar la data i l'hora d'un servidor NTP.

```
#include <WiFi.h>  
#include <NTPClient.h>  
#include <WiFiUdp.h>
```

Establir un temps de repòs profund

El codi utilitza un factor de conversió de microsegons a segons, de manera que podem establir el temps de repòs a la variable TIME_TO_SLEEP en segons.

En aquest cas, configurem l'ESP32 perquè vagi a dormir durant 10 minuts (600 segons). Si volem que l'ESP32 reposi durant un període de temps diferent, només cal que introduïm el nombre de segons per dormir profundament a la variable TIME_TO_SLEEP.

```
// Define deep sleep options  
uint64_t uS_TO_S_FACTOR = 1000000;  
// Conversion factor for microseconds to seconds  
// Sleep for 10 minutes = 600 seconds  
uint64_t TIME_TO_SLEEP = 600;
```

Configuració de les credencials de xarxa

Escrivim les nostres credencials de xarxa a les variables següents, de manera que l'ESP32 pugui connectar-se a la nostra xarxa local.

```
// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Inicialització de sensors i variables

A continuació, definim el pin SD de la targeta microSD. En aquest cas, s'estableix en GPIO5.

```
#define SD_CS 5
```

Creem una variable anomenada readingID per contenir l'identificador de lectura. Aquesta és una manera d'organitzar les nostres lectures. Per desar un valor variable durant el son profund, el podem desar a la memòria RTC. Per desar dades a la memòria RTC, només cal afegir RTC_DATA_ATTR abans de la definició de variable.

```
// Save reading number on RTC memory
RTC_DATA_ATTR int readingID = 0;
```

Creem una variable String per contenir les dades que es desaran a la targeta microSD.

```
String dataMessage;
```

A continuació, creem les instàncies necessàries per al sensor de temperatura. El sensor de temperatura està connectat a GPIO 21.

```
// Data wire is connected to ESP32 GPIO21
#define ONE_WIRE_BUS 21
// Setup a OneWire instance to communicate with a OneWire device
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature sensor
DallasTemperature sensors(&oneWire);
```

A continuació, creem una variable flotant per mantenir la temperatura recuperada pel sensor DS18B20.

```
float temperature;
```

Les dues línies següents defineixen un client NTP per sol·licitar la data i l'hora d'un servidor NTP.

```
WiFiUDP ntpUDP;  
NTPClient timeClient(ntpUDP);
```

A continuació, inicialitzem les variables de cadena per desar la data i l'hora.

```
String formattedDate;  
String dayStamp;  
String timeStamp;
```

getReadings ()

Vegem la funció getReadings(). Aquesta funció simplement lleixa la temperatura del sensor de temperatura DS18B20.

```
sensors.requestTemperatures();  
temperature=sensors.getTempCByIndex(0); // Temperature in Celsius
```

Per defecte, el codi recupera la temperatura en graus centígrads. Podem fer un comentari a la línia següent i comentar l'anterior per obtenir temperatura a Fahrenheit.

```
//temperature = sensors.getTempFByIndex(0);  
// Temperature in Fahrenheit
```

getTimeStamp()

La funció getTimeStamp() obté la data i l'hora. Les següents línies ens asseguren que obtindrem una data i hora vàlides:

```
while(!timeClient.update()) {  
    timeClient.forceUpdate();  
}
```

De vegades, el NTPClient recupera l'any 1970. Per assegurar-nos que no succeeixi, forçem l'actualització.

A continuació, convertim la data i l'hora a un format lleigible amb el mètode getFormattedDate():

```
formattedDate = timeClient.getFormattedDate();
```

La data i l'hora es tornen en aquest format:

2018-04-30T16:00:13Z

Per tant, hem de dividir aquesta cadena per obtenir la data i l'hora per separat. Això és el que fem aquí:

```
// Extract date
int splitT = formattedDate.indexOf("T");
dayStamp = formattedDate.substring(0, splitT);
Serial.println(dayStamp);
// Extract time
timeStamp=formattedDate.substring(splitT+1,
formattedDate.length()-1);
Serial.println(timeStamp);
```

La data es desa a la variable dayStamp i l'hora a la variable timeStamp.

writeFile() i appendFile()

Les dues funcions: writeFile() i appendFile() s'utilitzen per escriure i afegir dades a la targeta microSD. Inclouen els exemples de la biblioteca de targetes SD i no els hem de modificar.

```
// Write to the SD card (DON'T MODIFY THIS FUNCTION)
void writeFile(fs::FS &fs, const char * path, const char * message) {
    Serial.printf("Writing file: %s\n", path);

    File file = fs.open(path, FILE_WRITE);
    if(!file) {
        Serial.println("Failed to open file for writing");
        return;
    }
    if(file.print(message)) {
        Serial.println("File written");
    } else {
        Serial.println("Write failed");
    }
    file.close();
}

// Append data to the SD card (DON'T MODIFY THIS FUNCTION)
```

```
void appendFile(fs::FS &fs, const char * path, const char * message) {
    Serial.printf("Appending to file: %s\n", path);

    File file = fs.open(path, FILE_APPEND);
    if(!file) {
        Serial.println("Failed to open file for appending");
        return;
    }
    if(file.print(message)) {
        Serial.println("Message appended");
    } else {
        Serial.println("Append failed");
    }
    file.close();
}
```

logSDCard()

La funció logSDCard() concatena tota la informació de la variable de cadena dataMessage. Cada lectura està separada per comes.

```
dataMessage = String(readingID) + "," + String(dayStamp) + "," +
String(timeStamp) + "," + String(temperature) + "\r\n";
```

Notem que, “\ r \ n” al final de la variable dataMessage garanteix que la següent lectura s’escrigui a la següent línia.

A continuació, amb la següent línia, escrivim tota la informació al fitxer data.txt de la targeta microSD.

```
appendFile(SD, "/data.txt", dataMessage.c_str());
```

Notem que la funció appendFile() només accepta variables de tipus const char per al missatge. Per tant, utilitzeu el mètode c_str() per convertir la variable dataMessage.

setup ()

Quan utilitzem el mode de son profund amb l’ESP32, tot el codi hauria d’anar dins de la funció setup(), perquè l’ESP32 mai arriba al loop().

Connexió a Wi-Fi

El següent fragment de codi es connecta a la xarxa Wi-Fi. Cal que ens connectem al wi-fi per sol·licitar data i hora al servidor NTP.

```
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

}
```

Inicialització del client NTP

A continuació, inicialitzem el client NTP per obtenir la data i l'hora d'un servidor NTP.

```
timeClient.begin();
```

Podem utilitzar el mètode setTimeOffset (<time>) per ajustar l'hora de la vostra zona horària.

```
timeClient.setTimeOffset(3600);
```

Aquí tenim alguns exemples de diferents zones horàries:

- GMT +1 = 3600
- GMT +8 = 28800
- GMT -1 = -3600
- GMT 0 = 0

Inicialització del mòdul de la targeta microSD

A continuació, inicialitzem la targeta microSD. A continuació, si els extractes comproven si la targeta microSD està connectada correctament.

```
SD.begin(SD_CS);

if (!SD.begin(SD_CS)) {
    Serial.println("Card Mount Failed");
    return;
}

uint8_t cardType = SD.cardType();
if (cardType == CARD_NONE) {
    Serial.println("No SD card attached");
    return;
}

Serial.println("Initializing SD card...");
if (!SD.begin(SD_CS)) {
    Serial.println("ERROR - SD card initialization failed!");
    return; // init failed
}
```

A continuació, intentem obrir el fitxer data.txt a la targeta microSD.

```
File file = SD.open("/data.txt");
```

Si aquest fitxer no existeix, l'hem de crear i escriure l'encapçalament del fitxer .txt.

```
writeFile(SD, "/data.txt", "Reading ID, Date, Hour, Temperature
\r\n");
```

Si el fitxer ja existeix, el codi continua.

```
else {  
    Serial.println("File already exists");  
}
```

Finalment, tanquem el fitxer.

```
file.close();
```

Activa el despertador del temporitzador

A continuació, activem el despertador del temporitzador amb el temporitzador que hem definit anteriorment a la variable TIME_TO_SLEEP.

```
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
```

Inicialització de la biblioteca per a DS18B20

A continuació, inicialitzem la biblioteca per al sensor de temperatura DS18B20.

```
sensors.begin();
```

Obtenir les lectures i el registre de dades

Després de tenir-ho tot inicialitzat, podem obtenir les lectures, la marca de temps i registrar-ho tot a la targeta microSD.

Per facilitar la comprensió del codi, hem creat les funcions següents:

- **getReadings()**: llegeix la temperatura del sensor de temperatura DS18B20;
- **getTimeStamp()**: obté la data i l'hora del servidor NTP;
- **logSDcard()**: regista les dades anteriors a la targeta microSD.

Després de completar aquestes tasques, incrementem el **readingID**.

```
readingID++;
```

Finalment, l'ESP32 inicia el repòs profund.

```
esp_deep_sleep_start();
```

L'ESP32 mai arriba al bucle () .

```
void loop() {  
    // The ESP32 will be in deep sleep  
    // it never reaches the loop()  
}
```

4. DEMOSTRACIÓ

Si hem seguit tots els passos correctament i no ens ha donat cap missatge d'error, a la sortida del port serie del monitor ens ha d'aparèixer els següents missatges:

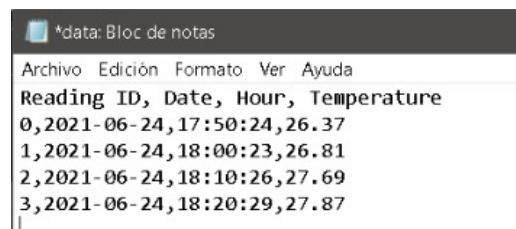
```
Connecting to CASA VIPA
..
WiFi connected.

Initializing SD card...
File already exists
Temperature: 26.37
2021-06-24T17:50:24Z
2021-06-24
17:50:24
Save data: 0,2021-06-24,17:50:24,26.37

Appending to file: /data.txt
Message appended
DONE! Going to sleep now.
```

Com hem dit en l'explicació del codi, si deixem el microprocessador funcionant aquest missatge ens apareix (en el nostre cas) cada 10 minuts que es el temps de repòs que hem utilitzat.

Un cop hem esperat el temps desitjat per tenir suficients mostres de temperatura, aquestes les podem visualitzar a la targeta microSD, per tant si disconnectem el microprocessador i connectem la targeta microSD a l'ordinador veurem un document anomenat "data.txt" on hi podem observar les dades que hem pres anteriorment.



Com es pot veure cada 10 minuts pren el valor de la temperatura.