

Data Logger étanche

Informations :

- Participants : Daoud Lasmar-saïd Kim Lao
- Mails : lasmarsaiddaoud@gmail.com, kimvaldeslao@gmail.com
- Début du projet - Fin du projet(présumé) : 17 juin 2024 - (28 juin 2024) -> fin du stage mais projet non fini

Contexte :

Dans certains environnements, la récupération de données peut-être assez compliqué et les appareils électroniques ne sont pas toujours résistants sur le long terme, notamment au sein d'environnements avec des conditions climatiques extrêmes. En effet, du fait de ces conditions certains appareils tels que les data logger se détériorent extrêmement vite et engendre donc des dépenses supplémentaires.

Objectifs :

Le but du projet est de créer un data logger pouvant supporter une immersion longue sous l'eau et en milieu avec des conditions climatiques extrêmes (100% d'humidité dans notre cas). Mais en raison du manque de temps nous nous sommes pour l'instant plus concentrés sur le développement du data logger que sur la partie étanchéité même si elle a été traitée.

Matériel :

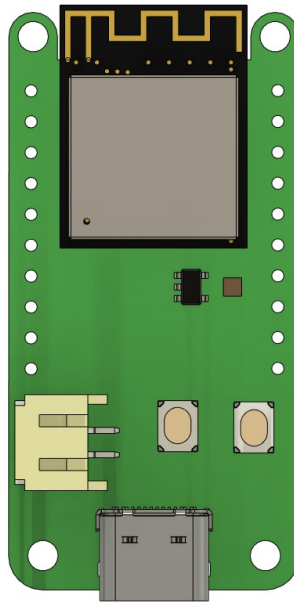
- 1 bread board
- Un capteur de température DS18B20
- fils de connexion
- M5Core2 (avec un esp32)

Machines utilisées :

Aucune

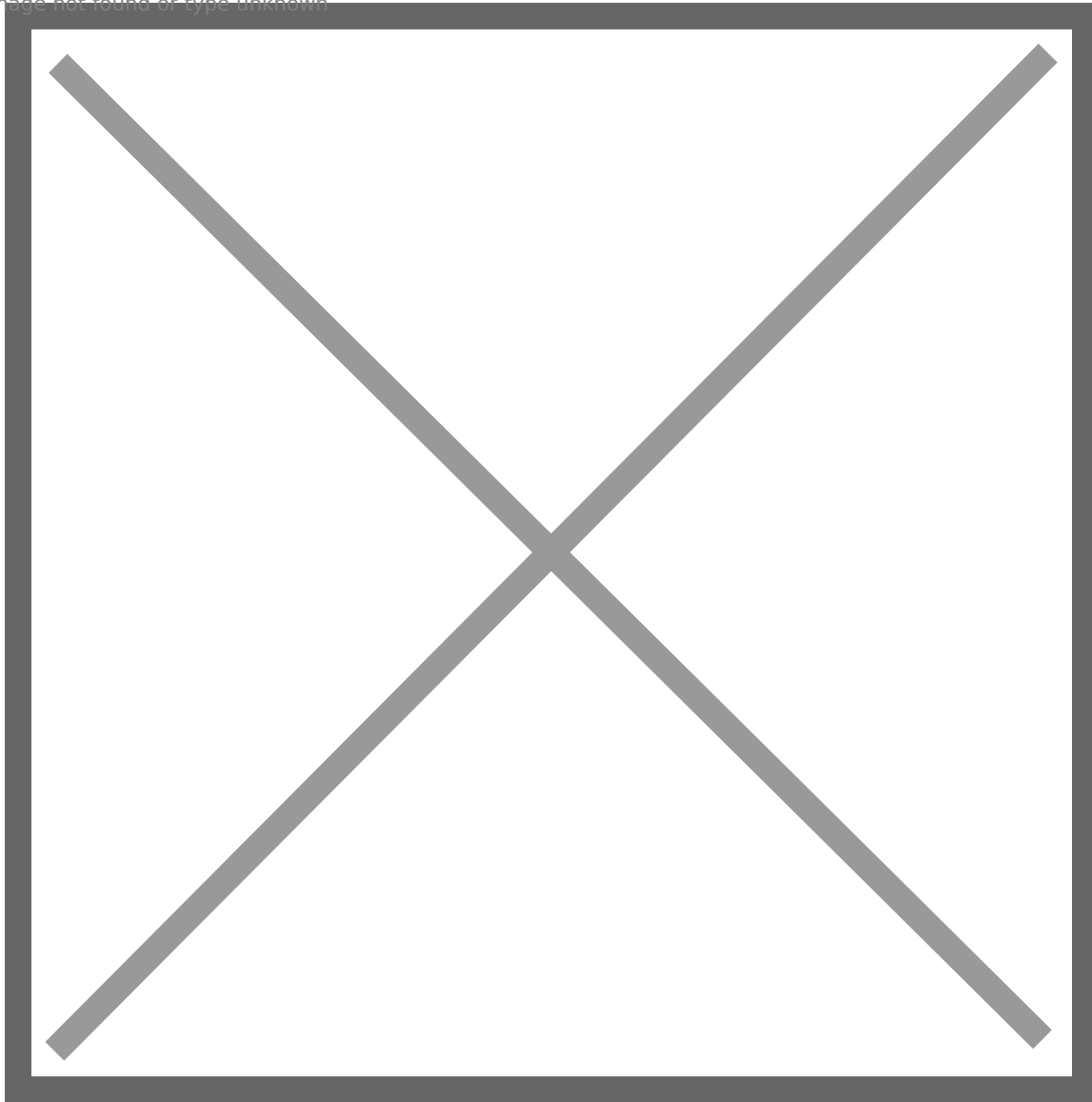
Construction :

Shéma de l'ESP 32 bee



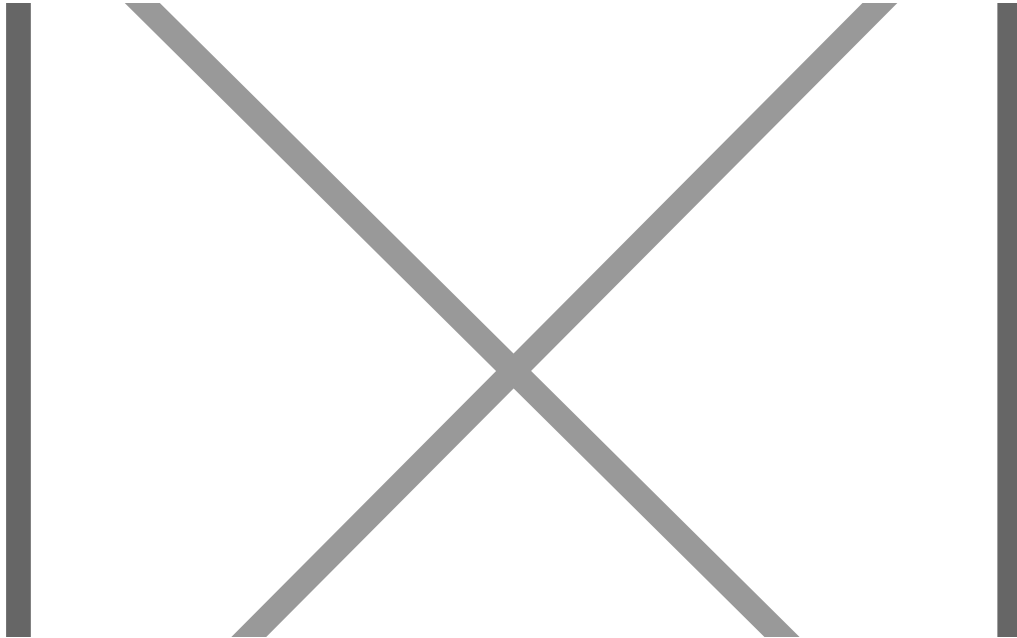
Pin map de l'ESP 32 bee :

Image not found or type unknown



Étape 1

- Réunion avec l'équipe du projet pour trouver des solutions et des informations en fonction du cahier des charges.
- recherches et documentation sur les normes d'étanchéité.



+ IP 69K (L'indice de protection IP69K est adapté aux puissants jets d'eau à haute température. Il protège contre les projections à haute pression et à haute température que l'on va privilégier lors de nos recherches sur des boîtiers étanches).

- recherches de boîtiers électriques étanches immersibles mais sans succès.
- nouvelle option d'une boîte étanche et de presses étoupes (percer la boîte et y installer des presse étoupes).
- recherches de boîtes étanches :

<ul style="list-style-type: none">• https://fr.souriau.com/fr-fr/products/connectors/waterproof-connectors/m-series
<ul style="list-style-type: none">• https://www.polycase.com/an-19f
<ul style="list-style-type: none">• https://bluerobotics.com/store/watertight-enclosures/wte-vp/

Étape 2

- Début du travail sur un Arduino Uno (montage électrique modélisé sur Tinkercad).

- remplacement de l'Arduino Uno par un M5 Stack composé d'un ESP 32 bee (moins couteux que l'ESP 32 classique) muni d'un module Bluetooth et wifi ainsi qu'une carte SD afin de simplifier le montage.
- montage du capteur de température DS18B20 sur le M5 Stack + écriture du code en C++.
- basculement sur un M5Core2 avec un RTC (real time clock).

Étape 3

- tentative de récupération des données de température via Bluetooth entre le M5Core2 et un appareil client (A finir).

Journal de bord :

17/06/2024

Recherche de boîtier étanches avec des caractéristiques répondant au cahier des charges établi lors de la réunion et prise de décision concernant la direction prise par le projet.

19/06/2024

Sélection de certains boîtier en particuliers qui seront par la suite envisagés comme étant des solutions au problème d'étanchéité que l'on retrouvait au début. Commencement du montage électrique sur Tinkercad en utilisant tout d'abord une carte Arduino Uno et un capteur de température DS18B20.

21/06/2024

Remplacement de la carte Arduino Uno par la M5 stack puis par le M5Core2 pour faciliter l'avancement du projet et écriture du code pour le data logger, afin de récupérer les données de températures du capteur et les écrire dans un fichier texte sur une carte SD.

code :

```
#include <M5Core2.h> // Inclusion of the M5Core2 library
#include <OneWire.h> // Inclusion of the OneWire library for the temperature sensor
#include <DallasTemperature.h> // DallasTemperature library
#include <SD.h> // library to write and read the SD card
#include <SPI.h> // library to communicate with SPI devices
#define ONE_WIRE_BUS 27 // We assign port 27 for the DS18B20 temperature sensor
#define SD_CS_PIN 4 // We assign port 4 for the SD card
OneWire oneWire(ONE_WIRE_BUS); // Configure a oneWire instance to communicate with all OneWire devices
DallasTemperature DS18B20(&oneWire);
void setup() // initialization of certain constants and declaration of variables
```

```

{
  M5.begin();
  M5.Lcd.setTextColor(TFT_WHITE, TFT_BLACK);
  M5.Lcd.setTextSize(2);
  if (!SD.begin(SD_CS_PIN)) // Checking for the presence of an SD card in port 4
  {
    M5.Lcd.println("Card Mount Failed");
    return;
  }
  // Initializing the internal clock of the M5Core2 (Real Time Clock)
  M5.Rtc.begin();
  // Set the RTC time if needed
  // RTC_TimeTypeDef TimeStruct;
  // TimeStruct.Hours = 10;
  // TimeStruct.Minutes = 30;
  // TimeStruct.Seconds = 0;
  // M5.Rtc.SetTime(&TimeStruct);
  // RTC_DateTypeDef DateStruct;
  // DateStruct.WeekDay = 3;
  // DateStruct.Month = 6;
  // DateStruct.Date = 19;
  // DateStruct.Year = 23;
  // M5.Rtc.SetDate(&DateStruct);
}

void loop() { // Recovery of data sent by the sensor and writing them to the SD card
  float celsius;
  float fahrenheit;
  DS18B20.begin(); // Starting the sensor
  int count = DS18B20.getDS18Count(); // Checking the number of connected sensors
  M5.Lcd.setCursor(0, 0);
  M5.Lcd.print("Devices found: ");
  M5.Lcd.println(count);
  if (count > 0) {
    DS18B20.requestTemperatures();
    File dataFile = SD.open("/temperature_data.txt", FILE_APPEND); // Writing data to a pre-existing document or
    creating the latter if it did not exist
    if (!dataFile) {
      M5.Lcd.println("Failed to open file");
      return;
    }
  }
}

```

```

// Retrieving the date and time
RTC_TimeTypeDef TimeStruct;
RTC_DateTypeDef DateStruct;
M5.Rtc.GetTime(&TimeStruct);
M5.Rtc.GetDate(&DateStruct);

String dateTime = String(DateStruct.Date) + "/" + String(DateStruct.Month) + "/" + String(2000 +
DateStruct.Year) + " " +
                String(TimeStruct.Hours) + ":" + String(TimeStruct.Minutes) + ":" + String(TimeStruct.Seconds);
for (int i = 0; i < count; i++) {
    celsius = DS18B20.getTempCByIndex(i);
    fahrenheit = celsius * 1.8 + 32.0;
    celsius = round(celsius);
    fahrenheit = round(fahrenheit);
    M5.Lcd.print("Device ");
    M5.Lcd.print(i);
    M5.Lcd.print(": ");
    M5.Lcd.print(celsius, 0);
    M5.Lcd.print("C / ");
    M5.Lcd.print(fahrenheit, 0);
    M5.Lcd.println("F");
    // Writing data to the SD card
    dataFile.print(dateTime);
    dataFile.print(" - Device ");
    dataFile.print(i);
    dataFile.print(": ");
    dataFile.print(celsius, 0);
    dataFile.print("C / ");
    dataFile.print(fahrenheit, 0);
    dataFile.println("F");
}
dataFile.close();
}
delay(2000); // Delay between measurements of 2 seconds
}

```

28/06/2024

Modifications du code pour permettre une connexion Bluetooth entre le M5Core2 et un client afin de communiquer les données du capteur sans pour autant aller chercher la carte SD, ce qui dans le

cadre de l'utilisation des capteurs est à privilégier.

code (à finir) :

```
#include <M5Core2.h> // Inclusion of the M5Core2 library
#include <OneWire.h> // Inclusion of the OneWire library for the temperature sensor
#include <DallasTemperature.h> // DallasTemperature library
#include <SD.h> // library to write and read the SD card
#include <SPI.h> // library to communicate with SPI devices
#include <ArduinoBLE.h> // ArduinoBLE library

#define ONE_WIRE_BUS 27 // We assign port 27 for the DS18B20 temperature sensor
#define SD_CS_PIN 4 // We assign port 4 for the SD card

OneWire oneWire(ONE_WIRE_BUS); // Configure a oneWire instance to communicate with all OneWire devices
DallasTemperature DS18B20(&oneWire);

// BLE Service and Characteristic
BLEService temperatureService("180D"); // Custom service
BLECharacteristic temperatureCharacteristic("2A37", BLERead | BLENotify, 512); // Custom characteristic

void setup()
{
    M5.begin();
    M5.Lcd.setTextColor(TFT_WHITE, TFT_BLACK);
    M5.Lcd.setTextSize(2);

    if (!SD.begin(SD_CS_PIN))
    {
        M5.Lcd.println("Card Mount Failed");
        return;
    }

    // Initializing the internal clock of the M5Core2 (Real Time Clock)
```



```
M5.Rtc.begin();
```

```
Serial.begin(9600);
```

```
while (!Serial);
```

```
// Begin initialization of BLE
```

```
if (!BLE.begin())
```

```
{
```

```
    M5.Lcd.println("Starting Bluetooth® Low Energy module failed!");
```

```
    while (1);
```

```
}
```

```
BLE.setLocalName("M5Core2");
```

```
BLE.setAdvertisedService(temperatureService);
```

```
temperatureService.addCharacteristic(temperatureCharacteristic);
```

```
BLE.addService(temperatureService);
```

```
BLE.advertise();
```

```
M5.Lcd.println("\nBluetooth device active, \nwaiting for connections...");
```

```
}
```

```
void loop()
```

```
{
```

```
    float celsius;
```

```
    float fahrenheit;
```

```
    DS18B20.begin(); // Starting the sensor
```

```
    int count = DS18B20.getDS18Count(); // Checking the number of connected sensors
```

```
M5.Lcd.setCursor(0, 0);
```

```
M5.Lcd.print("Devices found: ");
```

```
M5.Lcd.println(count);
```

```

if (count > 0)
{
    DS18B20.requestTemperatures();

    File dataFile = SD.open("/temperature_data.txt", FILE_APPEND); // Writing data to a pre-existing document or
creating the latter if it did not exist
    if (!dataFile)
    {
        M5.Lcd.println("Failed to open file");
        return;
    }

    // Retrieving the date and time
    RTC_TimeTypeDef TimeStruct;
    RTC_DateTypeDef DateStruct;
    M5.Rtc.GetTime(&TimeStruct);
    M5.Rtc.GetDate(&DateStruct);

    String dateTime = String(DateStruct.Date) + "/" + String(DateStruct.Month) + "/" + String(2000 +
DateStruct.Year) + " " +
        String(TimeStruct.Hours) + ":" + String(TimeStruct.Minutes) + ":" + String(TimeStruct.Seconds);

    for (int i = 0; i < count; i++)
    {
        celsius = DS18B20.getTempCByIndex(i);
        fahrenheit = celsius * 1.8 + 32.0;
        celsius = round(celsius);
        fahrenheit = round(fahrenheit);

        M5.Lcd.print("Device ");
        M5.Lcd.print(i);
        M5.Lcd.print(": ");
        M5.Lcd.print(celsius, 0);
        M5.Lcd.print("C / ");
        M5.Lcd.print(fahrenheit, 0);
        M5.Lcd.println("F");
    }
}

```

```

// Writing data to the SD card
dataFile.print(dateTime);
dataFile.print(" - Device ");
dataFile.print(i);
dataFile.print(": ");
dataFile.print(celsius, 0);
dataFile.print("C / ");
dataFile.print(fahrenheit, 0);
dataFile.println("F");
}

dataFile.close();
}

delay(2000); // Delay between measurements of 2 seconds

BLEDevice central = BLE.central();
if (central)
{
    M5.Lcd.println("Connected to central: ");
    M5.Lcd.println(central.address());

    // While the central is still connected to peripheral:
    while (central.connected())
    {
        File dataFile = SD.open("/temperature_data.txt");
        if (dataFile) {
            M5.Lcd.println("File opened successfully.");
            // Read the file and prepare data to send via BLE
            while (dataFile.available())
            {
                String data = "";
                while (dataFile.available() && data.length() < 512)
                {
                    data += (char)dataFile.read();
                }
            }
        }
    }
}

```

```
}  
temperatureCharacteristic.writeValue((const unsigned char*)data.c_str(), data.length());  
Serial.println(data); // For debugging, print to Serial Monitor  
delay(100); // Delay to ensure the data is sent properly  
}  
dataFile.close();  
Serial.println("\nFile sent successfully.");  
} else  
{  
    Serial.println("Error opening file.");  
}  
}  
// The central has disconnected  
M5.Lcd.println("Disconnected from central: ");  
}  
}
```

Revision #12

Created 28 June 2024 09:51:14 by Visinoni Etienne

Updated 21 October 2024 13:01:08 by Visinoni Etienne