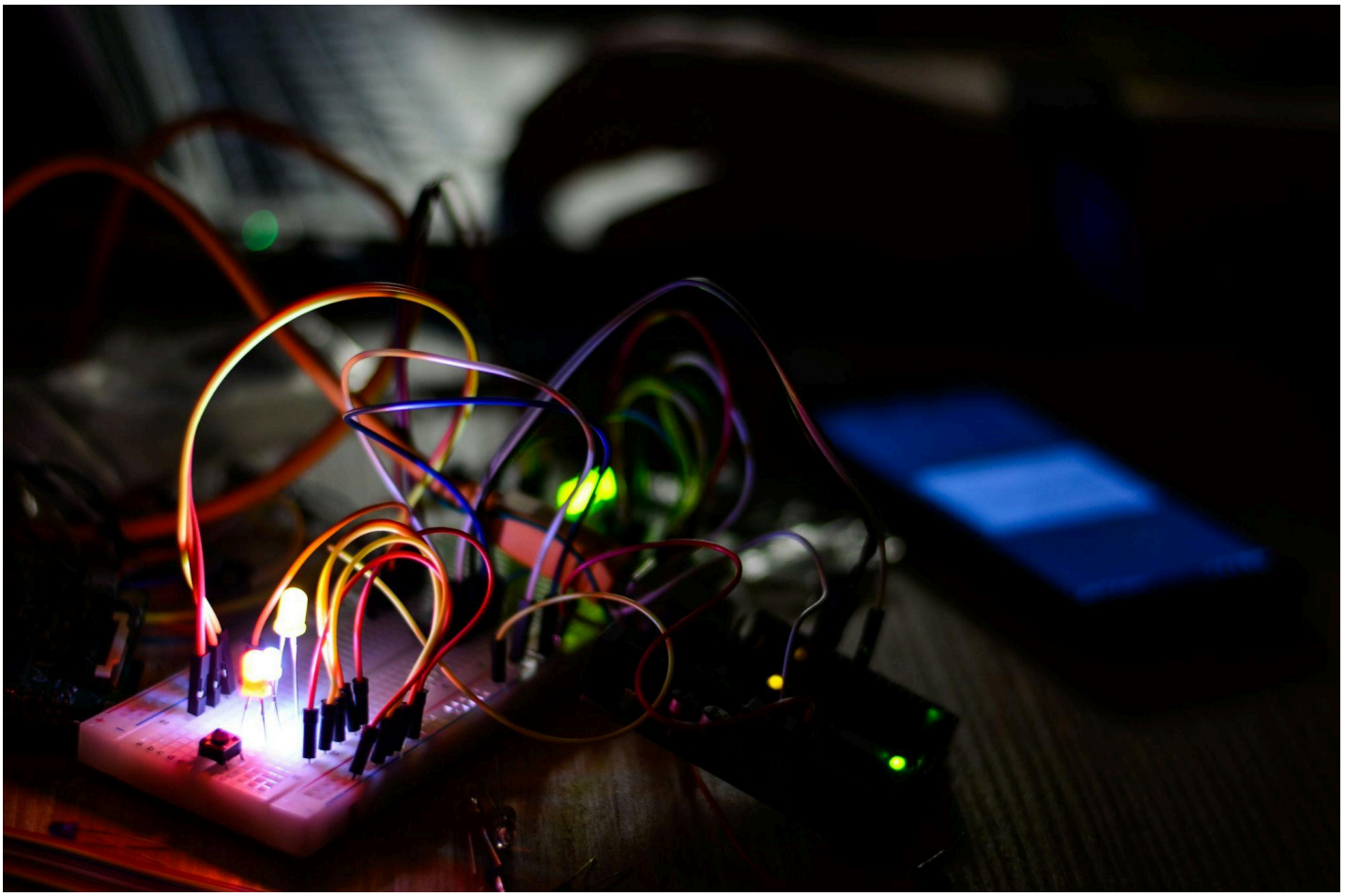
TRACEUR GPS

Conception et mise en œuvre d’un système de géolocalisation basé sur l'ESP32



Conditions préalables

Ce tutoriel se concentre sur la programmation de l'ESP32 à l'aide du noyau Arduino. Avant de commencer , il faut d'abord installer le noyau Arduino ESP32 dans l'IDE Arduino. Voici les étapes :

Avant de continuer, assurez-vous que l'IDE Arduino 2 est installé sur votre ordinateur :

https://docs.arduino.cc/software/ide/#ide-v2

Rendez-vous sur le site Web d'Arduino et téléchargez la version adaptée à votre système d'exploitation.

https://www.arduino.cc/en/software/#experimental-software

Linux : extrayez le fichier téléchargé et ouvrez le fichier arduino-ide qui lancera l'IDE.

Windows : exécutez le fichier téléchargé et suivez les instructions du guide d'installation (pour Windows, je vous recommandons la première option).

Mac OS X : copiez le fichier téléchargé dans votre dossier Applications.

Installer le module complémentaire ESP32 dans l'IDE Arduino

Pour installer la carte ESP32 dans votre IDE Arduino, suivez les instructions suivantes :

Ouvrez le gestionnaire de cartes. Vous pouvez aller dans Outils > Carte > Gestionnaire de cartes… ou simplement cliquer sur l'icône du gestionnaire de cartes dans le coin gauche.

Recherchez ESP32 et appuyez sur le bouton d'installation pour esp32 par Espressif Systems version 3.X.

Test de l'installation

Pour tester l'installation du module complémentaire ESP32, nous allons télécharger un code simple qui fait clignoter la LED intégrée (GPIO 2).

Copiez le code suivant dans votre IDE Arduino :

```
/*********

  Nathan Martin

*********/

#include <Arduino.h>

#define LED 2

void setup() {

  // put your setup code here, to run once:

  Serial.begin(115200);

  pinMode(LED, OUTPUT);

}

void loop() {

  // put your main code here, to run repeatedly:

  digitalWrite(LED, HIGH);

  Serial.println("LED is on");

  delay(1000);

  digitalWrite(LED, LOW);

  Serial.println("LED is off");

     delay(1000);
```

Téléchargement du croquis

Sélectionnez votre carte avant de télécharger le code. Dans le menu déroulant du haut, cliquez sur « Sélectionner une autre carte et un autre port… »

Une nouvelle fenêtre, comme illustré ci-dessous, s'ouvrira. Recherchez le modèle de votre carte ESP32.

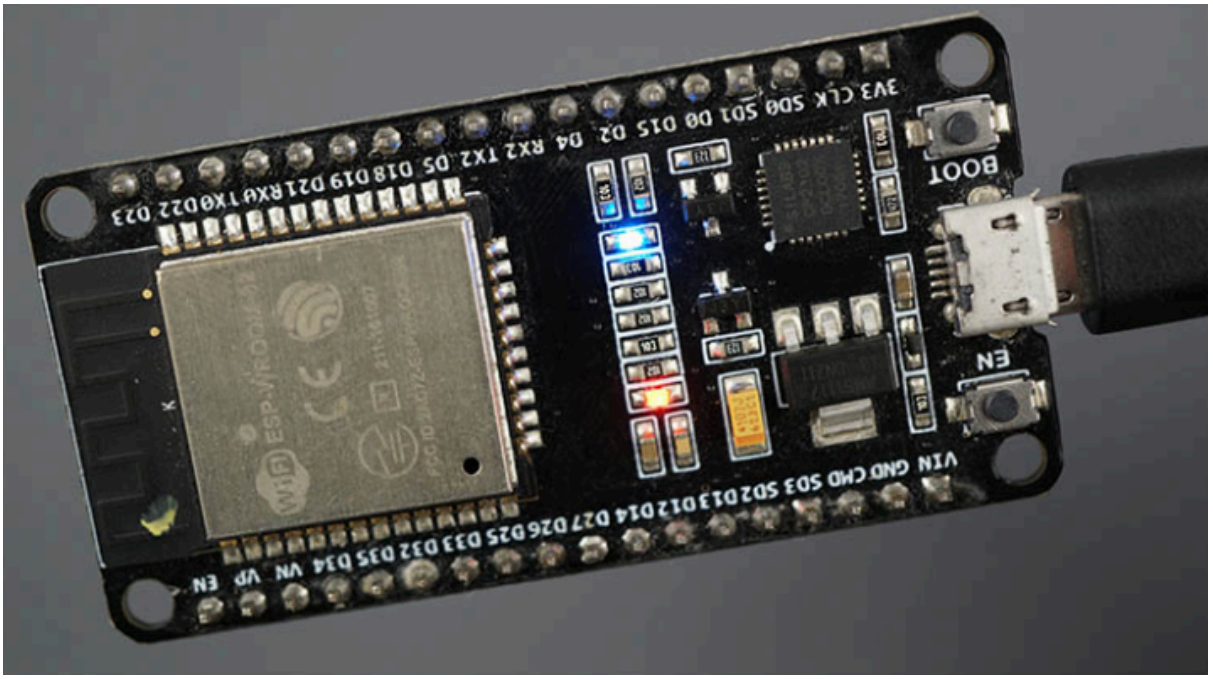
Sélectionnez le modèle de carte ESP32 que vous utilisez, ainsi que le port COM. Dans mon cas, j’utilise le DOIT ESP32 DEVKIT V1. Cliquez sur OK lorsque vous avez terminé.

Il ne vous reste plus qu'à cliquer sur le bouton « Upload ».

Après quelques secondes, le téléchargement devrait être terminé.

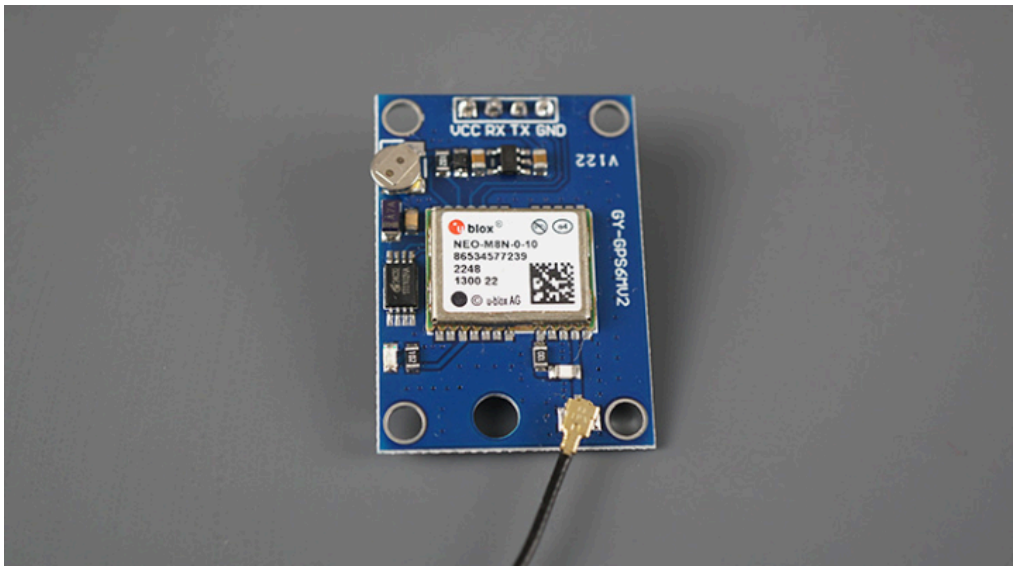
Remarque : certaines cartes de développement ESP32 ne passent pas automatiquement en mode flashage/téléchargement lors du téléchargement d’un nouveau code et vous verrez alors apparaître de nombreux points dans la fenêtre de débogage, suivis d’un message d’erreur. Si tel est le cas, vous devez appuyer sur le bouton ESP32 BOOT dès que les points apparaissent dans la fenêtre de débogage.

La LED intégrée à l’ESP32 devrait clignoter toutes les secondes.


BOOT
EN

Moniteur série
Vous pouvez cliquer sur l'icône du moniteur série pour ouvrir l'onglet « Serial Monitor ».
Veillez à sélectionner un débit de 115 200 bauds.
Et voilà ! Vous avez installé avec succès les cartes ESP32 dans l'IDE Arduino 2.

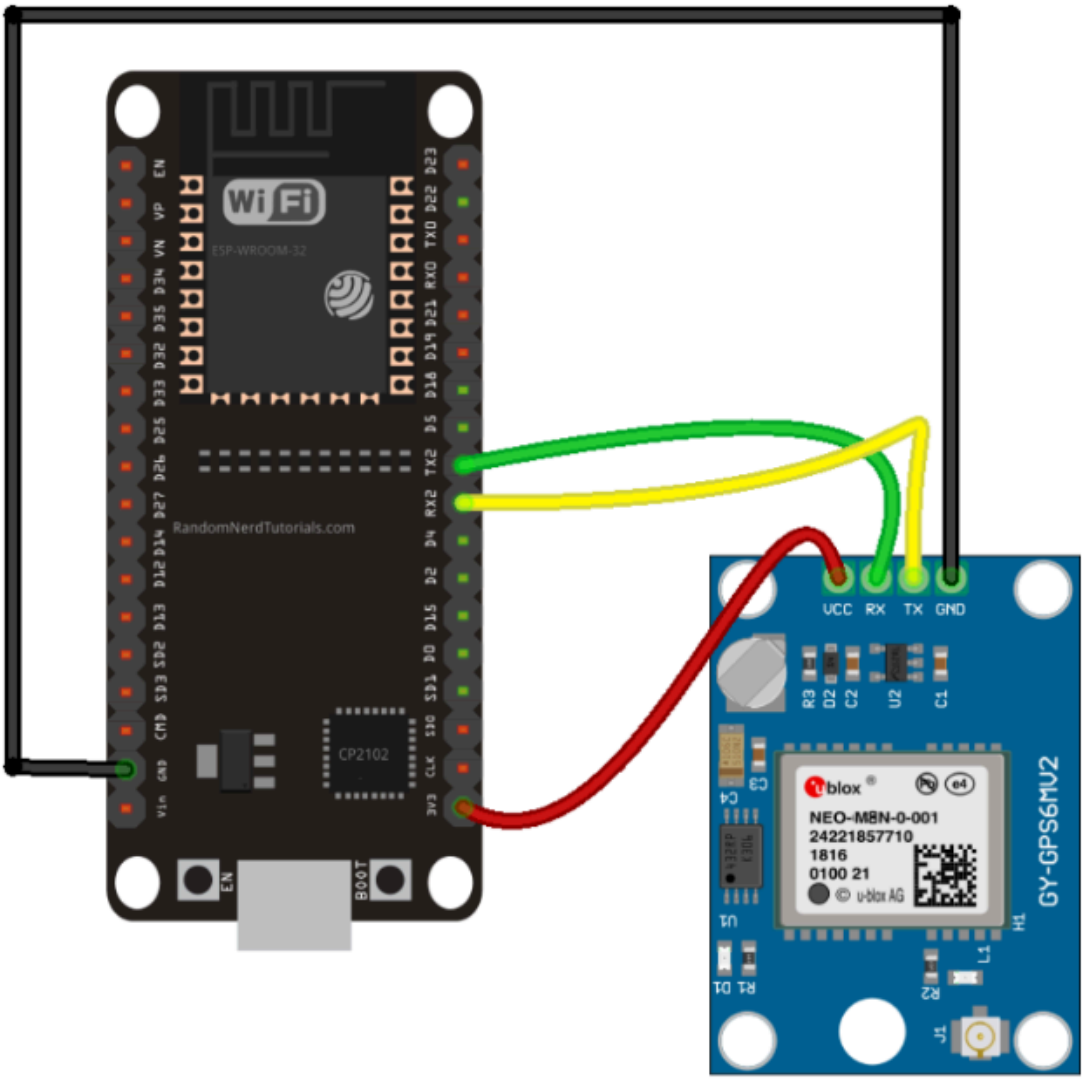
Présentation du module GPS NEO-M8N
Le module GPS NEO-M8N est l'un des récepteurs GPS les plus populaires utilisés avec les microcontrôleurs dans les projets de navigation et de suivi. Il permet d'obtenir des données sur la latitude, la longitude, l'altitude et l'heure.
Il prend en charge plusieurs systèmes satellitaires, notamment GPS, Galileo, GLONASS et BeiDou. Il offre un meilleur suivi satellite que le NEO-6M, ce qui le rend plus fiable dans des conditions difficiles.



Selon la fiche technique, il offre une précision de positionnement horizontal comprise entre 2,5 et 4 mètres et des temps de démarrage rapides (1 seconde pour un démarrage à chaud, 26 à 57 secondes pour un démarrage à froid — prévoyez des délais plus longs si vous vous trouvez à proximité de bâtiments).
Le module comprend une batterie de secours, une mémoire EEPROM intégrée et un voyant LED qui clignote lorsqu'une position est déterminée.
Ce module est généralement fourni avec une antenne GPS en céramique. Cependant, vous pouvez la remplacer par n'importe quelle autre antenne compatible qui conviendrait mieux à votre projet. Par exemple, j'aime utiliser celle qui se trouve à droite sur la photo ci-dessous, car elle est étanche et est fournie avec un long câble qui offre plus de flexibilité.

Le module GPS NEO-M8N communique avec un microcontrôleur via un protocole de communication série et utilise des phrases NMEA standard. NMEA signifie « National Marine Electronics Association » ; dans le domaine du GPS, il s'agit d'un format de données standard pris en charge par les fabricants de GPS.

Câblage du module GPS NEO-M8N sur l'ESP32
Nous allons connecter le module GPS NEO-M8N. Vous pouvez vous référer à l'image et au tableau suivants.



NEO-M8N GPS Module	ESP32
VCC	3V3
RX	TX2 (GPIO 17)
TX	RX2 (GPIO 16)
GND	GND

Obtenir des données GPS brutes – Tester le module GPS NEO-M8N avec l'ESP32

Pour obtenir des données GPS brutes, vous devez établir une communication série avec le module GPS et lire les données disponibles.

Le code suivant établit une communication série avec le module GPS et lit les données disponibles.

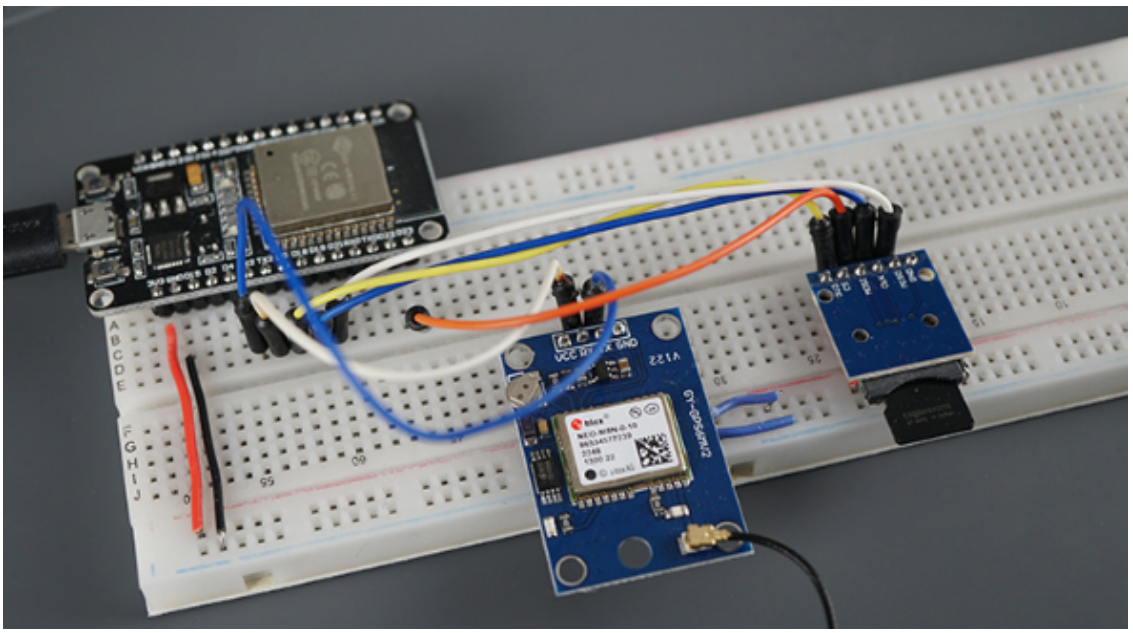
```
/*********

  Nathan Martin



 // Define the RX and TX pins for Serial 2

#define RXD2 16

#define TXD2 17


#define GPS_BAUD 9600


// Create an instance of the HardwareSerial class for Serial 2

HardwareSerial gpsSerial(2);


void setup(){

  // Serial Monitor

  Serial.begin(115200);



  // Start Serial 2 with the defined RX and TX pins and a baud rate of 9600

  gpsSerial.begin(GPS_BAUD, SERIAL_8N1, RXD2, TXD2);

  Serial.println("Serial 2 started at 9600 baud rate");

}


void loop(){

  while (gpsSerial.available() > 0){

    // get the byte data from the GPS

    char gpsData = gpsSerial.read();

    Serial.print(gpsData);

  }

  delay(1000);

  Serial.println("-------------------------------");

   }
```

Tester le code
Téléchargez le code sur votre carte ESP32. Ouvrez le moniteur série avec un débit de 115 200 bauds. Assurez-vous que votre module GPS est placé à l'extérieur ou près d'une fenêtre afin de pouvoir capter les données des satellites.

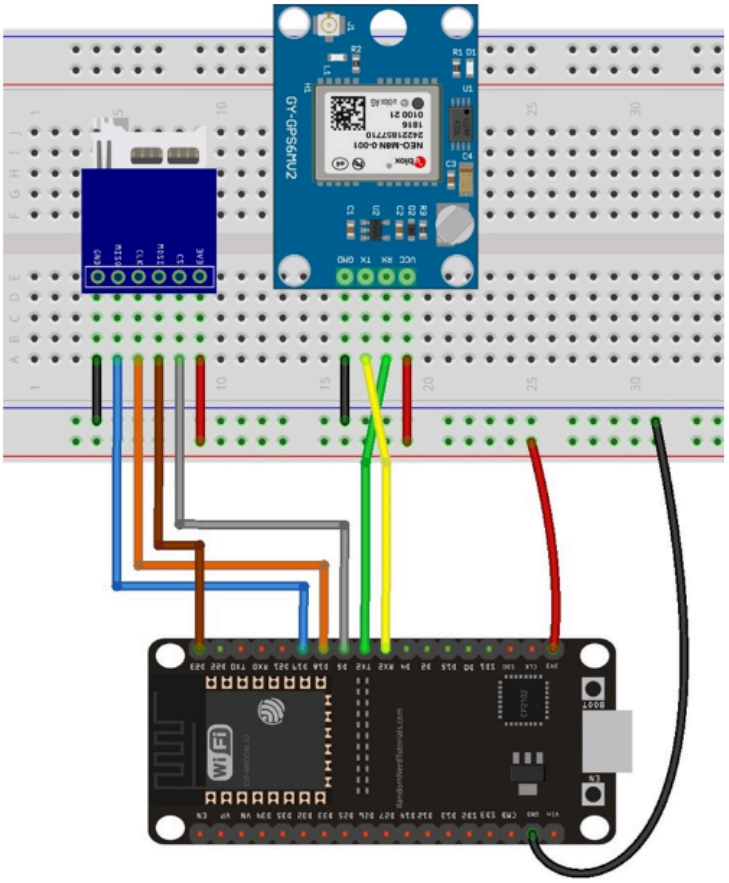
Le moniteur série vous fournira des données sur votre position actuelle, votre vitesse, votre altitude, le nombre de satellites visibles, le HDOP et l'heure.
HDOP signifie « Horizontal Dilution of Precision » (dilution horizontale de la précision). Il s'agit d'une mesure de la précision du calcul de position. Plus la valeur HDOP est élevée, moins le calcul de position sera précis. Idéalement, vous devriez obtenir une valeur inférieure à 2. Une valeur plus faible signifie une meilleure précision.

Enregistreur GPS et affichage du tracé sur Google Earth

Maintenant que vous maîtrisez mieux l'utilisation du module GPS NEO-M8N avec l'ESP32, créons un enregistreur GPS qui consigne votre position au fil du temps dans un fichier sur une carte microSD. Vous pourrez ensuite modifier et utiliser ce fichier sur Google Earth pour visualiser l'évolution de votre position dans le temps (tracé).



Câblage du circuit
Raccordez le module GPS aux broches UART2 par défaut de l'ESP32 (comme nous l'avons fait précédemment) et le module de carte microSD aux broches SPI par défaut de l'ESP32. Vous pouvez vous référer au schéma suivant ou aux tableaux ci-dessous.



You may also like reading: ESP32: Guide for MicroSD Card Module using Arduino IDE.

NEO-M8N GPS Module	ESP32
VCC	3V3
RX	TX2 (GPIO 17)
TX	RX2 (GPIO 16)
GND	GND

MicroSD Card Module	ESP32
3V3*	3V3
CS	GPIO 5
MOSI	GPIO 23
CLK	GPIO 18
MISO	GPIO 19
GND	GND

Code

The following code initializes the microSD card and the GPS module and saves GPS data as soon as the data points are available and the new point is at least 1 meter from the previous one.

```
/*********

  Nathan Martin

*********/


#include <TinyGPS++.h>

#include <FS.h>

#include <SD.h>

#include <SPI.h>


// Define the RX and TX pins for Serial 2

#define RXD2 16

#define TXD2 17


#define GPS_BAUD 9600


// The TinyGPS++ object

TinyGPSPlus gps;
```

```
// Create an instance of the HardwareSerial class
for Serial 2
HardwareSerial gpsSerial(2);

String GPSdataToAppend;

// Previous GPS coordinates
double lastLat = 0.0;
double lastLng = 0.0;

// Initialize SD card
void initSDCard() {
  if (!SD.begin()) {
    Serial.println("Card Mount Failed");
    return;
  }
  uint8_t cardType = SD.cardType();

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

```
  }

  Serial.print("SD Card Type: ");

  if (cardType == CARD_MMC) {

    Serial.println("MMC");

  } else if (cardType == CARD_SD) {

    Serial.println("SDSC");

  } else if (cardType == CARD_SDHC) {

    Serial.println("SDHC");

  } else {

    Serial.println("UNKNOWN");

  }

  uint64_t cardSize = SD.cardSize() / (1024 * 1024);

  Serial.printf("SD Card Size: %lluMB\n", cardSize);

}


// Append data to the SD card

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 Monitor
  Serial.begin(115200);

  // Initialize the microSD card
  initSDCard();

  // Start Serial 2 with the defined RX and TX pins
and a baud rate of 9600
```

```
  gpsSerial.begin(GPS_BAUD, SERIAL_8N1, RXD2, TXD2);
  Serial.println("Serial 2 started at 9600 baud
rate");
}

void loop() {
  unsigned long start = millis();

  while (millis() - start < 1000) {
    while (gpsSerial.available() > 0) {
      gps.encode(gpsSerial.read());
    }
    if (gps.location.isUpdated()) {
      double currentLat = gps.location.lat();
      double currentLng = gps.location.lng();

      // Check if the distance from the last point
is at least 1 meter
      double distance =
TinyGPSPlus::distanceBetween(lastLat, lastLng,
currentLat, currentLng);

      if (distance >= 1.0) {
```

```
            lastLat = currentLat;
            lastLng = currentLng;

            // Prepare data to append
            GPSdataToAppend = String(currentLng, 6) +
"," + String(currentLat, 6) + "," +
String(gps.altitude.meters());
            Serial.println(GPSdataToAppend);

            // Append to file
            appendFile(SD, "/data.txt",
GPSdataToAppend.c_str());
        }
    }
  }
 }
```

Pour enregistrer les données dans un format lisible par Google Earth, nous devons enregistrer la longitude, la latitude et l'altitude dans cet ordre, séparées par des virgules. C'est ce que nous faisons dans cet exemple. Par la suite, nous devrons convertir ces informations en un fichier .kml.

Test du projet

Téléchargez le code sur votre carte. La carte microSD devrait s'initialiser et, après quelques secondes, commencer à enregistrer des données sur la carte microSD (assurez-vous d'être à l'extérieur ou près d'une fenêtre).

Si tout semble fonctionner comme prévu, déconnectez l'ESP32 de votre ordinateur et alimentez-le à l'aide d'un chargeur portable ou d'une batterie. Partez en promenade avec votre circuit afin qu'il commence à enregistrer vos coordonnées au fil du temps et que vous puissiez obtenir un nombre suffisant de points pour tracer un itinéraire.

Après avoir enregistré quelques données, déconnectez la carte microSD du module et connectez-la à votre ordinateur. La carte microSD devrait contenir un fichier data.txt avec les coordonnées de votre parcours.

Pour que vous puissiez télécharger ces données sur Google Earth et tracer un itinéraire, nous devons convertir ce fichier au format .kml.

Fichiers KML

KML est un format de fichier utilisé pour afficher des données géographiques dans un navigateur de type Google Earth. Le KML utilise une structure basée sur des balises avec des éléments et des attributs imbriqués, et repose sur la norme XML.

Conversion du fichier data.txt au format .kml

1) Ouvrez le fichier data.txt contenant les données GPS recueillies par le module GPS.

2) Modifiez votre fichier de manière à ce que vos coordonnées se trouvent entre les balises <coordinates></coordinates>, comme suit :

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Style id="yellowPoly">
<LineStyle>
<color>7f00ffff</color>
<width>4</width>
</LineStyle>
```

```
<PolyStyle>

<color>7f00ff00</color>

</PolyStyle>

</Style>

<Placemark><styleUrl>#yellowPoly</styleUrl>

<LineString>

<extrude>1</extrude>

<tesselate>1</tesselate>

<altitudeMode>absolute</altitudeMode>

<coordinates>
```

YOUR COORDINATES GO HERE

```
</coordinates>

</LineString></Placemark>

</Document></kml>
```

3) Enregistrez ce fichier.

4) Renommez-le en remplaçant « data.txt » par « data.kml ». Un message d'avertissement s'affichera pour vous signaler que le format du fichier va changer. Acceptez-le. Le fichier devrait désormais pouvoir être importé dans Google Earth.

Afficher l'itinéraire sur Google Earth

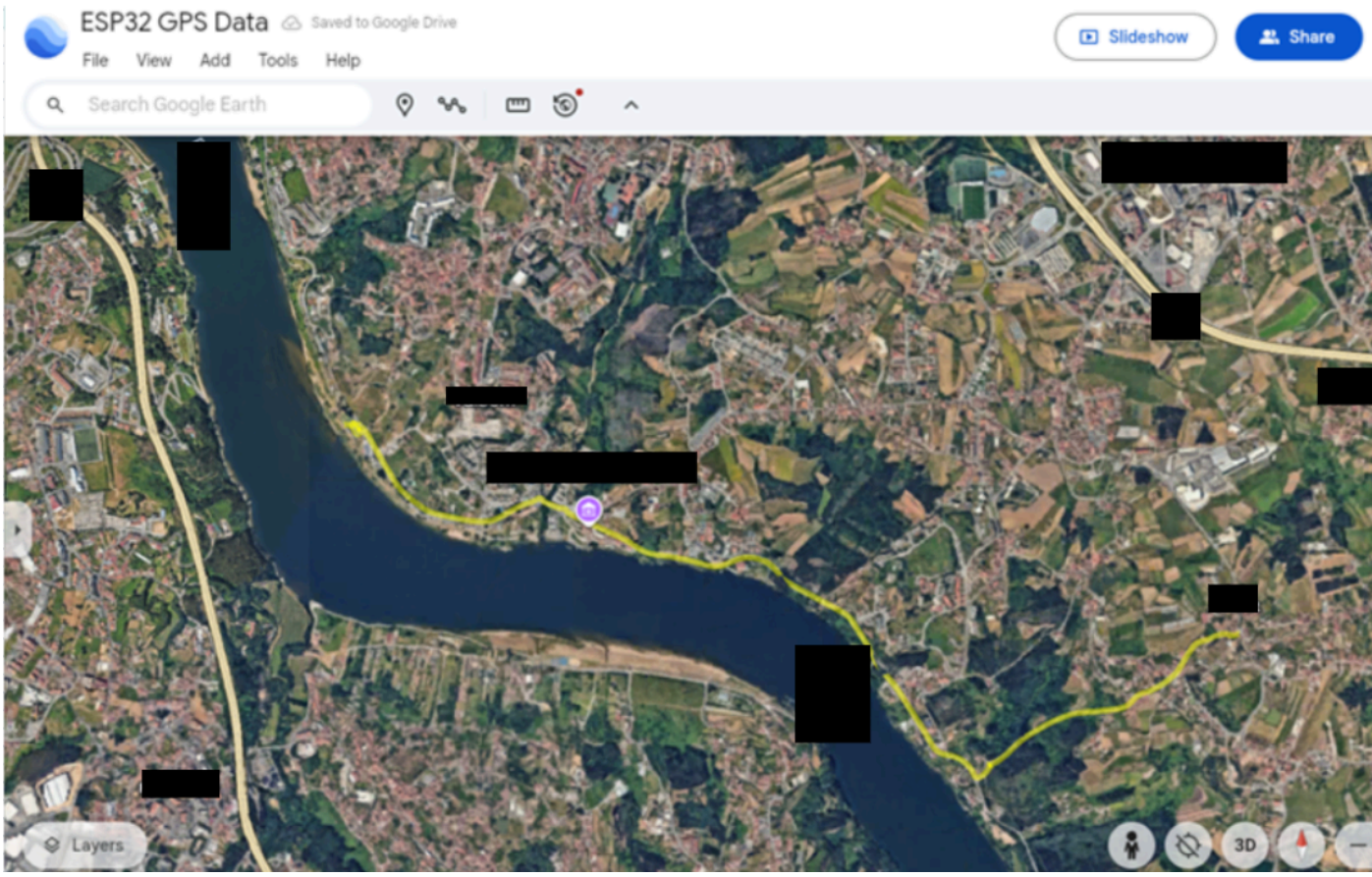
Suivez maintenant les étapes suivantes pour afficher et visualiser votre itinéraire sur Google Earth.

1) Rendez-vous sur le site Web de Google Earth.

2) Créez un nouveau projet et donnez-lui un nom.

3) Allez dans Fichier > Importer un fichier dans le projet > Importer depuis un appareil.

4) Sélectionnez le fichier .kml créé précédemment.
5) Votre itinéraire s'affichera sur Google Earth avec un contour jaune.


ESP32 GPS Data
Saved to Google Drive
File
View
Add
Tools
Help
Slideshow
Share
Search Google Earth
Layers
3D