MANUEL DÉVELOPPEUR LOGICIEL

ROULE MA POULE SANS NID-DE-POULE

3 mars 2019

Bastian Bouchardon
Polytech Tours
Département informatique industrielle

Table des matières

1.1 INTRODUCTION

Ce document donne des précisions sur la façon dont le programme a été rédigé, les classes, le fonctionnement de la méthode principale, les règles de codage utilisées. Des détails seront apportés sur les outils utilisés et des tutoriels pour les mettre en oeuvre. Enfin, des précisions sont également apportées concernant le futur du code source, des conseils sur la structure.

Tout est disponible sur le gitlab du projet à l'adresse suivante : GitLab

1.2 STRUCTURE LOGICIELLE

1.2.1 Fonctionnalités

Le programme est embarqué dans la cible ESP32. Il est écrit en C++ et permet à l'ESP de réaliser ces tâches :

- Initialisation d'un port série et ou d'un point d'accès wifi avec un serveur telnet
- Création d'un système de fichiers SPIFFS [1] dans la mémoire flash de l'ESP32
- Création d'un fichier CSV dans le système de fichiers
- Écriture d'une ligne dans le fichier CSV avec l'heure, la position GPS, les valeurs X, Y, Z de l'accéléromètre et la valeur de la pulsation dû au capteur piézoélectrique toutes les 200 millisecondes.
- Récupération des accélérations
- Interruption sur les fronts montants et descendants du capteur piézoélectrique et calcul de la durée de cette pulsation.
- Création d'un shell sur le port série :
  - h : aide avec la liste des commandes (help)
  - r : lecture du fichier CSV présent dans le SPIFFS (read)
  - e : effacement du fichier et réécriture avec l'entète CSV (erase)
  - l : liste des fichiers et dossiers (list)
  - o : démarrage / arrêt de l'écriture des données dans le fichier toutes les 200ms (on / off)
  - s : taille restante dans la mémoire flash et pourcentage d'utilisation (size)

1. Serial Peripheral Interface Flash File System

1.2.2 Classes

Voici les classes présentes dans le programme :

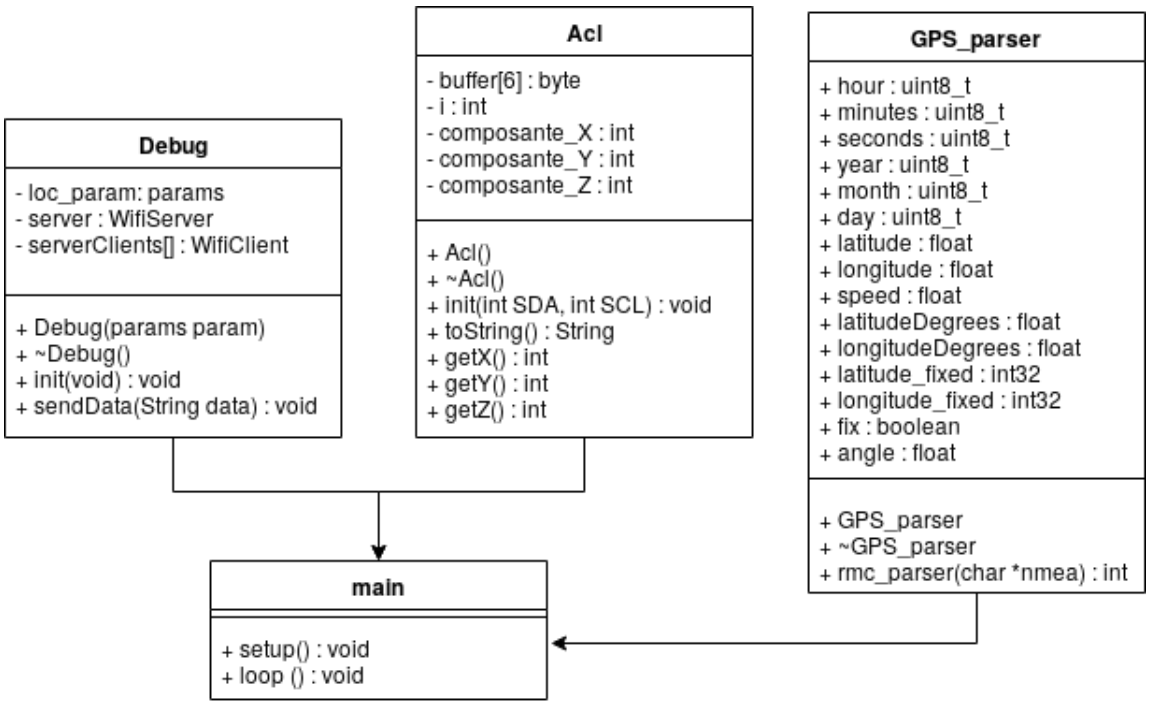

FIGURE 1.1 – Diagramme de classes

1.2.2.1 Classe Debug

Cette classe permet de faire le lien entre le microcontrôleur et l'extérieur. En paramètre de son constructeur, on peut indiquer si on veut que les données soient envoyées par WiFi, par liaison série ou les deux. Cela conditionne les initialisations et le canal sur lequel envoyer les données provenant des capteurs.

Ensuite la méthode init() permet d'initialiser la liaison série et/ou de créer un réseau WiFi et un serveur telnet permettant ensuite de pouvoir se connecter à celui-ci. Le hotspot est protégé par un mot de passe.

Une dernière méthode "sendData(String data)" permet d'envoyer une chaîne de caractères sur le port série, le WiFi ou les deux suivants le choix effectué auparavant.

On peut sur cette capture comment les paramètres sont définis comme le choix du média pour le débogage fait par le biais d'une énumération.

```
#define WIFI_SSID           "test_nidDePoule"
#define WIFI_PWD            "NidDePoule"

#define MAX_SRV_CLIENTS     2

typedef enum params {
    NOTHING           = 0
   ,WIFI_DEBUG        = 1
   ,SERIAL_DEBUG      = 2
   ,SERIAL_WIFI_DEBUG = 3
} params;
```

FIGURE 1.2 – Extrait debug.h

1.2.2.2 Classe Acl

Cette classe fait office de pilote pour le composant ADXL345. Il permet d'initialiser ce dernier et de récupérer grâce à la méthode "getValues()" les accélérations en X, Y et Z sous forme d'une chaîne de caractères. Lors de l'initialisation, nous pouvons choisir quelle sensibilité doit avoir l'accéléromètre (2, 4, 8, 16g).

1.2.2.3 Classe main

C'est la classe principale, avec la boucle principale. Elle permet d'initialiser les objets, de récupérer les informations des capteurs et de les envoyer soit au port série pour le débogage soit en WiFi pour les tests.

Cette classe collabore avec global.h, un fichier qui définit les classes à utiliser, les initialisations à faire. Tout le code est packagé dans des #if #endif permettant d'inhiber certaines parties du code.

```
#ifdef DEBUG_SOFT
  #include <Debug.h>
#else
  #define OFFLINE

  #include <SPIFFS.h>
  #include <FS.h>

  #define PATH_FILE           "/travel.csv"
  #define PATH                "/"
  #define HEADER_CSV          "DATE, LATITUDE, LONGITUDE, ACCELX, ACCELY, ACCELZ, PIEZO\n"
  #define HEADER_CSV_GPS_ONLY "LATITUDE, LONGITUDE\r\n"

  #define TIMEOUT             200
#endif

#ifdef ACCELERO
  #include <Acl.h>
#endif

#ifdef PIEZZO
#endif

#ifdef GPS
  #define LED 25
  #include <GPS_parser.h>
#endif
```

FIGURE 1.3 – Extrait global.h

1.3 RÈGLES D'ÉCRITURE / LICENCES

Le code a été écrit avec les règles d'écriture de code source de Karnigan et Richie. Cela permet d'avoir un code facile à lire donc à maintenir. les classes, les conditions, les calculs sont clairement lisibles et identifiables.

Concernant la licence, j'ai choisi la licence Créative Commons CC BY-NC-SA 4.0 avec ces droits :

FIGURE 1.4 – Creative Commons

— SHARE : copy and redistribute the material in any medium or format

— ADAPT : remix, transform, and build upon the material

On ne peut pas utiliser le code à des fins commerciales et on doit le redistribuer avec la même licence si on modifie le code source.

1.4 OUTILS

Le code a été développé en C++ avec des librairies Arduino sur l'IDE platformIO. Cela permet d'être adaptable facilement sur tout type de microcontrôleur ou SoC.

1.4.1 PlatformIO

PlatformIO est un IDE sous forme d'un plugin pour les éditeurs de texte Atom et VS Code. Il permet de combiner un interpréteur, un compilateur, un débugueur, un outil de test unitaire, un terminal série pour 598 cartes de différents constructeurs. Il remplace 30 plates-formes de développement avec 17 frameworks et 6186 librairies.
Pour compiler le code vous pouvez utiliser l'IDE d'arduino ou ce plugin en l'installant dans Atom ou VS Code avec leur installateur de plugins.

FIGURE 1.5 – plugins installés dans VS Code

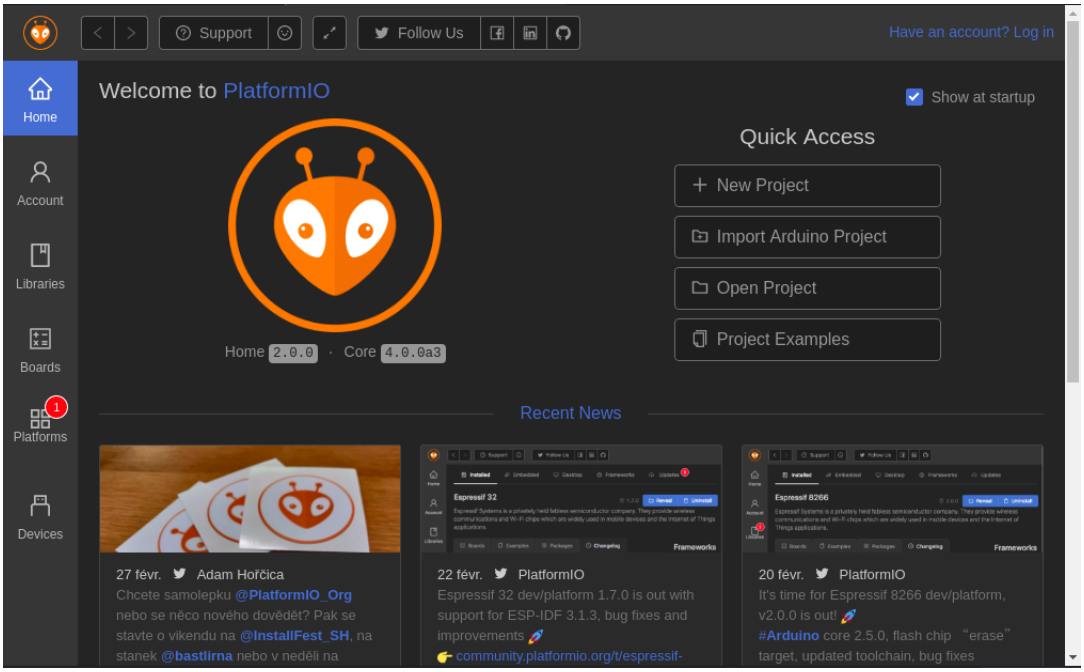

FIGURE 1.6 – page PlatformIO

1.5 CONSEILS

Pour les prochains développeurs sur ce projet, je conseille de passer par une machine à états pour gérer la partie embarquée dans le l'ESP32. Cela permettra de gérer tous les événements, de capter les vibrations, de donner une note, de détecter des pavés sur la route, d'intégrer la position GPS et les événements provenant du bluetooth.
Pour la partie communication bluetooth ou wifi, je passerai par des données transmises en binaire ou en BSON. Cela permettra d'envoyer les données utiles avec très peu d'octets. Un format de trame pourrait être : éventuel identifiant, position GPS, note de l'état de la route. Cette note pourrait être sur une échelle de 0 à 5 avec de 0 à 4 l'état de la route (0 très mauvaise et 4 très bonne) et enfin 5 correspondrait à une route pavée.

Enfin, l'équipe de RIOT, un système d'exploitation libre destiné aux cibles de l'IOT comme les microcontrôleurs (32, 16 et 8bits), porte son OS sur l'ESP32. A l'heure où j'écris ces lignes, certaines fonctionnalités de l'ESP ne sont pas encore prises en charge.

Table des figures