
GRP8
LIVRABLE3

SOMMAIRE

1/-PRÉSENTATION DU GROUPE 8

///

///

///

2/-INTRODUCTION

Le projet Worldwide Weather Watcher vise à concevoir une station météo embarquée pour les navires. Elle mesurera en continu des données essentielles pression, température, humidité, luminosité et position GPS afin d'améliorer la prévision des phénomènes climatiques extrêmes. Ces informations, disponibles en temps réel et stockées sur carte SD, pourront ensuite être échangées entre bâtiments pour affiner l'anticipation des catastrophes naturelles.

Ce rapport présente l'architecture générale du programme utilisé par la carte Arduino basée sur un microcontrôleur AVR ATmega328. Il détaille la structure du code, les principales fonctions et variables, ainsi que l'organisation logique du système. De plus, il inclut une explication du script de compilation et de téléversement utilisé pour transférer le programme sur la carte. Ce document a pour objectif de fournir une **vue d'ensemble claire du fonctionnement du logiciel**, servant à la fois de guide de développement et de référence pour la compréhension du code avant sa finalisation.

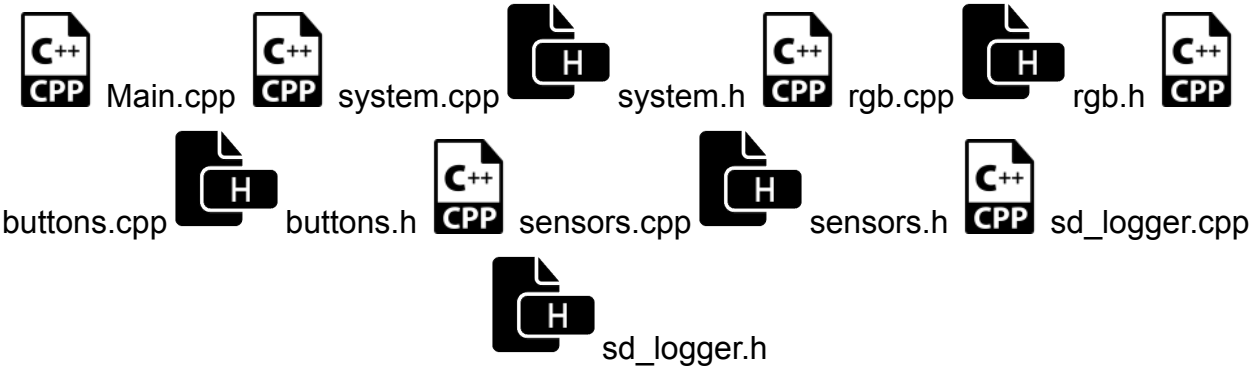
AIVM

Agence Internationale pour la Vigilance Météorologique

3/-ORGANISATION DU CODE SOURCE

Le code source de l'appareil prend en compte de nombreuses fonctionnalités, ce qui lui confère une taille importante. Avoir l'intégralité du code dans un seul fichier (`main.cpp`) compliquerait sa compréhension et pourrait rendre le débogage plus difficile. C'est pourquoi il a été divisé en plusieurs fichiers `.cpp` et en headers `.h`. Chaque fichier contient une partie du code correspondant à une fonctionnalité générale.

VISUALISATION ORGANISATION DU CODE SOURCE



 Main.cpp	**Contient les fonctions de base `setup()` et `loop()`. Il s'occupe de la transition entre les différents modes, de l'initialisation du système au démarrage, ainsi que du système de commandes via UART.**
 system.cpp	**Contient des variables globales assignées, qui peuvent être modifiées pendant l'exécution du code et sont accessibles depuis l'ensemble du système.**
 system.h	**Contient les déclarations de variables ainsi que les constantes `const` (stockées dans la mémoire flash).**
 rgb.cpp	**Contient le code qui contrôle la LED RGB avec différents presets de couleurs et de motifs, appelé lors des changements de mode ou en cas d'erreur.**
 rgb.h	**Initialisation RGB et accès aux valeurs globales via `#include <system.h>`.**
 buttons.cpp	**Contient les fonctions d'interruption associées aux deux boutons poussoirs, appelées lors de leurs changements d'état.**
 buttons.h	**Initialisation des fonctions définies dans `buttons.cpp` et accès aux valeurs globales via `#include <system.h>`**

 sensors.cpp	**Contient le code qui assure la lecture des capteurs, de l'horloge Tiny RTC et du GPS. Il vérifie également à chaque lecture si les composants sont accessibles et, en cas de problème, renvoie le code d'erreur approprié.**
 sensors.h	**Initialisation des fonctions définies dans `sensors.cpp` et accès aux valeurs globales via `#include <system.h>`.**
 sd_logger.cpp	**Contient le code chargé de la sauvegarde des données sur la carte SD. Il vérifie également si la carte SD est accessible ou pleine et, en cas de problème, renvoie le code d'erreur approprié.**
 sd_logger.h	**Initialisation des fonctions définies dans `sd_logger.cpp` et accès aux valeurs globales via `#include <system.h>`.**

INFORMATION:
À quoi servent les fichiers headers `.h` ?

Les fichiers headers `.h` servent à **déclarer les fonctions, les variables et les constantes** utilisées dans un programme, afin de pouvoir les partager entre plusieurs fichiers `.cpp`. Ils permettent ainsi d'organiser le code de manière modulaire, d'éviter les répétitions et de faciliter la maintenance et le débogage. En résumé, un fichier header définit **l'interface** d'un module sans en inclure directement l'implémentation.

4/-FONCTIONNEMENT DU CODE

Cette partie explique le fonctionnement **des interactions entre les différentes fonctions et systèmes**.

FONCTIONEMENT GENERALE

Au démarrage du système, la fonction `setup()` initialise des paramètres clés, tels que la communication Serial UART. Elle vérifie également si le bouton rouge est pressé :

- **Si le bouton est pressé**, le système bascule en **Mode Config**, dans lequel les valeurs stockées en EEPROM peuvent être modifiées via des commandes UART.

- **Si le bouton n'est pas pressé**, le système démarre directement en **Mode Standard**, qui consiste à acquérir les données des capteurs à intervalles réguliers et à les sauvegarder sur la carte SD.

Si un capteur ou un composant est retiré pendant le Mode Standard ou le **Mode Économique** (similaire au Mode Standard mais avec un intervalle d'acquisition doublé), une erreur est déclenchée. Cela inclut également la carte SD, qui est utilisée pour le stockage dans ces deux modes.

Le **Mode Maintenance**, quatrième et dernier mode, désactive le stockage sur la carte SD et affiche les données directement sur le port Serial UART. Ce mode n'est pas sensible aux erreurs liées à la carte SD puisqu'elle n'est pas utilisée.

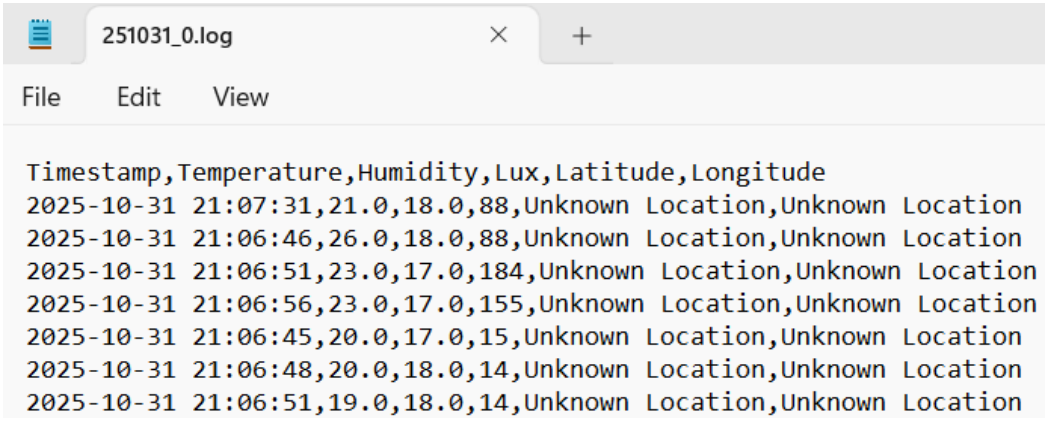
Le passage entre les modes Standard, Économique et Maintenance peut également se faire via les boutons. Un système d'interruption est utilisé pour détecter le moment où un bouton est pressé, en sauvegardant la valeur de `millis()`. Le changement de mode est validé si le bouton reste pressé pendant 5 secondes (`millis() - 5000`).

Lors du stockage des données sur la carte SD, celles-ci sont écrites ligne par ligne dans des fichiers `.log` au format CSV, pour permettre une ouverture facile sous forme de tableau dans Excel. Les noms des fichiers suivent le format : **"AAJJMM_R.LOG"**, où :

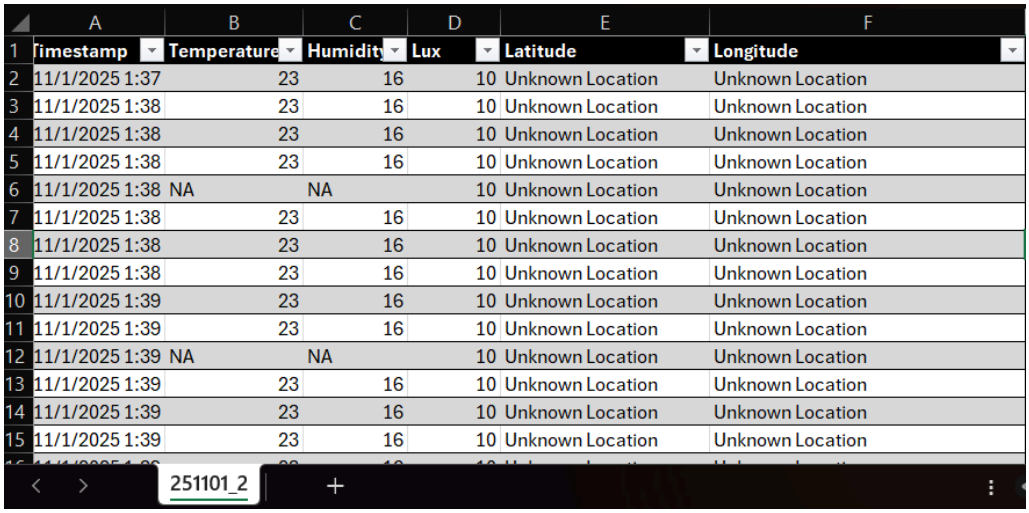
- `AA` = deux derniers chiffres de l'année
- `JJ` = jour
- `MM` = mois
- `R` = numéro de révision

Les données sont ajoutées au fichier `.log` courant jusqu'à ce que sa taille dépasse la limite définie par la constante `FILE_MAX_SIZE`, après quoi un nouveau fichier est créé.

251031_0.log



251101_2.csv



	A	B	C	D	E	F
1	Timestamp	Temperature	Humidity	Lux	Latitude	Longitude
2	11/1/2025 1:37	23	16	10	Unknown Location	Unknown Location
3	11/1/2025 1:38	23	16	10	Unknown Location	Unknown Location
4	11/1/2025 1:38	23	16	10	Unknown Location	Unknown Location
5	11/1/2025 1:38	23	16	10	Unknown Location	Unknown Location
6	11/1/2025 1:38	NA	NA	10	Unknown Location	Unknown Location
7	11/1/2025 1:38	23	16	10	Unknown Location	Unknown Location
8	11/1/2025 1:38	23	16	10	Unknown Location	Unknown Location
9	11/1/2025 1:38	23	16	10	Unknown Location	Unknown Location
10	11/1/2025 1:39	23	16	10	Unknown Location	Unknown Location
11	11/1/2025 1:39	23	16	10	Unknown Location	Unknown Location
12	11/1/2025 1:39	NA	NA	10	Unknown Location	Unknown Location
13	11/1/2025 1:39	23	16	10	Unknown Location	Unknown Location
14	11/1/2025 1:39	23	16	10	Unknown Location	Unknown Location
15	11/1/2025 1:39	23	16	10	Unknown Location	Unknown Location

251101_2

5/-LIBRAIRIES UTILISÉE

Adafruit BusIO, https://github.com/adafruit/Adafruit_BusIO
Cette bibliothèque fournit une couche d'abstraction pour la communication I²C et SPI et pour la gestion des registres de périphériques. Elle facilite l'écriture de drivers de capteurs ou d'interface utilisateur. L'empreinte Flash est modérée mais non négligeable sur un microcontrôleur limité comme l'ATmega328P.
BH1750, https://github.com/claws/BH1750
Bibliothèque pour le capteur de luminosité numérique BH1750 via I²C. Elle permet de lire les valeurs de lux en différents modes de mesure. La bibliothèque est relativement légère mais il est recommandé de mesurer son impact mémoire dans un contexte critique.
DHTNEW, https://github.com/RobTillaart/DHTNew
Bibliothèque pour les capteurs DHT11 et DHT22. Elle gère la lecture de la température et de l'humidité automatiquement, simplifiant l'utilisation des capteurs. L'empreinte mémoire reste modérée.

NeoSWSerial, https://github.com/SlashDevin/NeoSWSerial
Bibliothèque pour communication série logicielle optimisée. Permet de créer des liaisons RX/TX sur des broches digitales avec une consommation CPU réduite par rapport à la SoftwareSerial standard. L'empreinte Flash est faible et adaptée aux microcontrôleurs limités.
RTClib, https://github.com/adafruit/RTClib
Bibliothèque pour horloges temps réel comme DS3231 ou DS1307. Elle gère la date, l'heure et les alarmes. L'empreinte mémoire est modérée et adaptée aux microcontrôleurs avec peu de Flash.
SdFat, https://github.com/greiman/SdFat
Bibliothèque avancée pour la gestion des cartes SD avec FAT16 ou FAT32. C'est un des modules les plus consommateurs de Flash dans le projet. La bibliothèque a été configurée pour réduire son poids sur la Flash en limitant le support aux systèmes FAT nécessaires et en désactivant les fonctionnalités non utilisées, Elle a été choisie pour permettre de stocker des fichiers avec des noms dépassant la limite classique de 8 caractères pour le nom et 3 caractères pour l'extension, propre au format 8.3 des systèmes FAT.

6/-OPTIMISATION DU CODE

Le microcontrôleur de la carte Arduino Uno R3, l'**ATmega328P**, est très limité en performances et en espace mémoire. Il dispose de **32 Ko de mémoire Flash** pour le programme (dont 0,5 Ko pour le bootloader), de **2 Ko de SRAM** pour l'exécution et de **1 Ko d'EEPROM** pour le stockage de données non volatiles. Le code et les librairies utilisées sont relativement volumineux par rapport à ces ressources, c'est pourquoi il est impératif d'**optimiser le code au maximum** afin d'éviter un **stack overflow** ou une mémoire Flash insuffisante pour stocker le programme lui-même.

COMMENT OPTIMISER LE CODE?

1. Minimiser l'utilisation de variables globales

Limiter le nombre de variables globales permet de réduire la consommation de SRAM et d'éviter un dépassement de la pile. Préférer les variables locales lorsque possible, et passer les valeurs par paramètres de fonction.

Exemple dans notre cas: utilisation de variables globales uniquement lorsqu' il est nécessaire de sauvegarder leur valeurs entre plusieurs appels de fonctions

2. Utiliser des types de données adaptés

Choisir des types de données correspondant strictement aux besoins. Par exemple, utiliser `uint8_t` au lieu de `int` si la valeur ne dépasse pas 255. Cela réduit l'espace mémoire et la taille du programme.

Exemple dans notre cas: utilisé lorsque possible comme pour les constantes numéro de pins

3. Exploiter la mémoire Flash pour les constantes

Stocker les chaînes de caractères et constantes en Flash avec le mot-clé `const` et `PROGMEM` afin de libérer de la SRAM.

Exemple dans notre cas: Serial.Print(**F(**"TEXT"**)**) stocke le "TEXT" en Flash et non pas en SRAM

4. Réduire la taille des bibliothèques utilisées

Inclure uniquement les librairies nécessaires et utiliser des versions légères ou personnalisées pour éviter de charger des fonctions inutiles en Flash.

Exemple dans notre cas: désactivation de fonctionnalités non nécessaires dans la librairie SdFAT

5. Optimiser les structures de données
Utiliser des tableaux de taille minimale, éviter les structures imbriquées lourdes et préférer les tableaux simples lorsque possible.

Exemple dans notre cas: utilisé uniquement pour stocker les données lors d'un cycle d'acquisition

6. Réduire la profondeur des appels de fonctions
Limiter les appels récursifs et la profondeur des fonctions imbriquées afin d'éviter d'épuiser la pile SRAM.

7. Libérer la mémoire dynamique après usage
Si l'allocation dynamique est utilisée (`malloc`, `new`), s'assurer de libérer la mémoire avec `free` ou `delete` dès qu'elle n'est plus nécessaire.

8. Éviter les chaînes de caractères temporaires
Limiter la création de `String` dynamiques et préférer les tableaux de caractères fixes pour stocker des chaînes.

9. Optimiser les boucles et conditions
Simplifier les boucles et conditions, éviter les calculs redondants à l'intérieur des boucles, et pré-calculer les valeurs constantes.

10. Compacter le code des interruptions
Les routines d'interruption doivent être courtes et rapides, en appelant seulement des fonctions essentielles pour libérer rapidement la pile et le CPU.

Exemple dans notre cas: Les fonctions d'interruption des boutons sont courtes et rapides.

11. Surveiller l'utilisation de la mémoire SRAM
Utiliser des outils ou macros pour vérifier l'utilisation de la SRAM pendant le développement et détecter les zones critiques avant qu'elles provoquent des erreurs.

12. Fractionner le code en modules
Diviser le code en fichiers `.cpp` et `.h` modulaires permet de compiler uniquement les parties nécessaires, de réduire la duplication de code et d'améliorer la lisibilité et la maintenance.

7/-LIENS DE TÉLÉCHARGEMENT

Projet_SE_GRP8.zip Programme complet de la carte (PlatformIO)

Build_Upload_Groupe8.bat Le script de compilation et de téléversement

ETAPE 1

Telecharger PlatformIO pour cela suivez cette video youtube:
Tutorial: Visual Studio Code install PlatformIO IDE extension

ETAPE 2

Télécharger notre projet. **PlatformIO** contient la configuration, le code source et les librairies nécessaires. Ensuite, ouvrir le projet dans **PlatformIO**.

ETAPE 3

Télécharger notre script de téléversement, qui utilise **PlatformIO** pour compiler et téléverser le programme sur la carte Arduino. Il suffit de spécifier au début du script le chemin vers le projet **PlatformIO**, puis de l'exécuter.

```
@echo off
:: ========================================
:: PlatformIO Automation Script for Projet_SE_GRP8
:: ========================================
:: Make sure PlatformIO Core (CLI) is installed:
:: pip install platformio
:: And added to PATH

:: Go to your PlatformIO project folder
cd /d "C:\Users\NITRO\Documents\PlatformIO\Projects\Projet_SE_GRP8"
```

Si l'Arduino est connecté à un port COM, le script compilera ensuite le programme, le téléversera sur la carte, puis ouvrira l'interface série pour communiquer avec l'Arduino.

```
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file ".pio\build\uno\firmware.hex"
avrdude: writing flash (31668 bytes):

Writing | ################################################## | 100% 5.18s

avrdude: 31668 bytes of flash written
avrdude: verifying flash memory against .pio\build\uno\firmware.hex:
avrdude: load data flash data from input file .pio\build\uno\firmware.hex:
avrdude: input file .pio\build\uno\firmware.hex contains 31668 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 4.06s

avrdude: verifying ...
avrdude: 31668 bytes of flash verified

avrdude: safemode: Fuses OK (E:00, H:00, L:00)

avrdude done.  Thank you.

============================================ [SUCCESS] Took 13.54 seconds ============================================
```

```
==========================
Opening serial monitor...
==========================
--- Terminal on COM10 | 9600 8-N-1
--- Available filters and text transformations: colorize, debug, default, direct, hexlify, log2file, nocontrol, printabl
e, send_on_enter, time
--- More details at https://bit.ly/pio-monitor-filters
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
Logged: 251101_2.log
Logged: 251101_2.log
Logged: 251101_2.log
Logged: 251101_2.log
```

5-MÉMOIRE UTILISÉE APRÈS COMPILATION

```
RAM:   [=======   ]  70.8% (used 1450 bytes from 2048 bytes)
Flash: [==========]  98.2% (used 31668 bytes from 32256 bytes)
```

Ce qui est dans les limites de l'acceptable, et la compilation est pré-optimisée pour réduire la taille du code grâce à build_flags = -Os dans le fichier platformio.ini.

8-CONCLUSION

En conclusion, ce rapport a permis de présenter de manière claire et structurée l'architecture et le fonctionnement du programme développé pour la carte Arduino basée sur le microcontrôleur AVR ATmega328. La description du code, des fonctions principales et de l'organisation du système, ainsi que l'explication du script de compilation et de téléversement, offrent une compréhension globale du logiciel. Ce document constitue ainsi un outil de référence précieux pour le développement futur, la maintenance et l'optimisation du code, tout en facilitant la prise en main du projet pour tout développeur souhaitant y contribuer.

AIVM

Agence Internationale pour la Vigilance Météorologique

FIN DU DOCUMENT