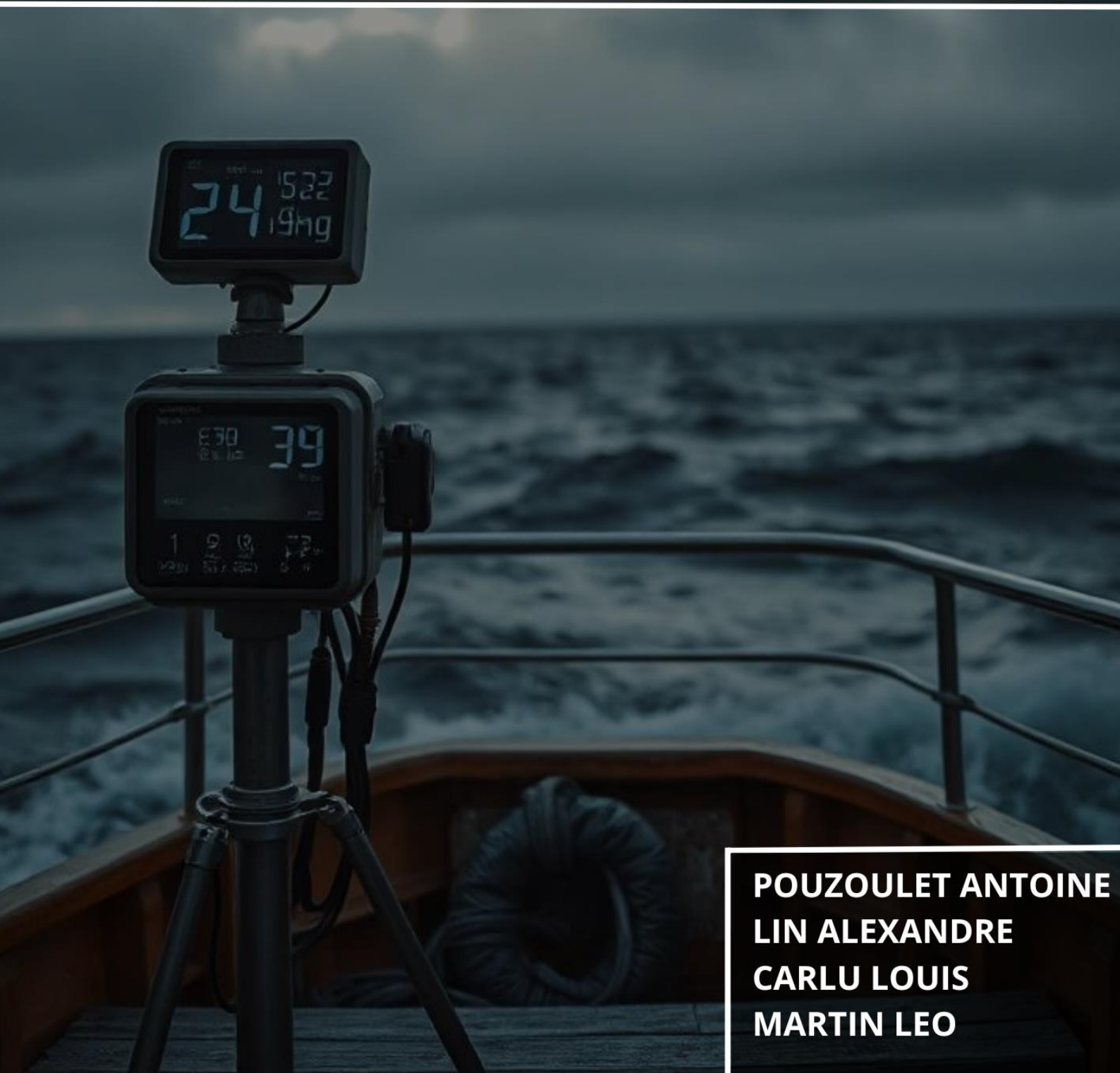
Livrable 4 : Worldwide Weather Watcher

Documentation



POUZOULET ANTOINE
LIN ALEXANDRE
CARLU LOUIS
MARTIN LEO

Table des matières

I. Introduction

L'Agence Internationale pour la Vigilance Météorologique (AIVM) se lance un projet ambitieux, il consiste à déployer dans les océans, des navires de surveillance équipés de stations météo embarquées. Ces stations météo ont pour rôle de mesurer les paramètres influant sur la formation de catastrophes naturelles. De nombreuses société utilisant des transports navals ont alors acceptées d'équiper leurs bateaux des stations météo embarquées. Celles-ci devront être simples d'utilisation, efficaces et pilotables par un membre de l'équipage. La réalisation du prototype pour ce projet a été confié par un des dirigeant de l'agence à une startup dont nous faisons partie.

Le projet est sur le point de se terminer, les objectifs du système ont été validés par l'agence, le montage et les différentes fonctions du système ont également été présentés lors d'une démonstration. Pour finaliser le projet et le faire approuver par l'agence, il ne reste qu'à fournir une documentation solide sur le fonctionnement et l'utilisation de notre station météo. Vous retrouverez donc dans ce document une partie documentation technique (fonctionnement global du système et architecture générale du système) expliquant comment la station météo a été réalisée, puis une partie documentation utilisateur (superviseur final) permettant de comprendre comment utiliser notre station météo.

II. Fonctionnement global du système

- Objectifs

La finalité de ce projet est de prévoir les catastrophes naturelles grâce à l'échange de données entre les navires. Les stations doivent donc être capables de recueillir instantanément les données des différents capteurs et de les sauvegarder sur une carte SD.

- Composants

Notre système se compose de d'une carte Arduino, de deux boutons, d'une LED rgb, d'un lecteur de carte SD et sa carte, d'une horloge RTC et de deux capteurs.

1. Carte Arduino :
   Comme microcontrôleur pour notre projet nous avons choisi la carte Arduino Uno qui est basé sur le microcontrôleur ATmega328P.
   Nous avons choisi ce microcontrôleur pour sa facilité d'utilisation et sa flexibilité, en effet il s'intègre très bien dans beaucoup de projets don le projet Worldwide Weather Watcher. Nous avons aussi choisi ce microcontrôleur pour sa compatibilité avec beaucoup de capteurs et de modules.
   Dans notre système, la carte Arduino va être la pièce maitresse. C'est elle qui va exécuter notre programme et donc interagir avec tous les composants de notre système.

2. Boutons :
   Pour interagir physiquement avec notre système, nous avons décidé d'utiliser deux boutons, un bouton vert et un bouton rouge. Techniquement, ces boutons vont laisser passer un signal à la carte Arduino lorsque ils seront pressés. C'est grâce à ces boutons que l'utilisateur pourra changer de modes par exemple.

3. LED rgb :
   Dans notre système, la LED rgb va permettre la communication de la carte Arduino vers l'utilisateur. En effet en fonction de la couleur qu'elle affichera, l'utilisateur sera si il y aune erreur ou dans quel mode il se situe. Pour gérer cette LED dans notre programme, nous utilisons la bibliothèque Arduino « ChainableLED.h ».

4. Lecteur de carte SD et sa carte :
   Le lecteur de carte SD va nous permettre d'enregistrer sur la carte SD les données captés. Nous utiliserons la carte SD en écriture pour pouvoir enregistrer les données, mais aussi en lecture lors du mode maintenance par exemple où nous affichons les données qui y sont enregistrés. Pour gérer l'enregistrement des données sur la carte SD, nous utilisons la bibliothèque Arduino « SD.h ».

5. Horloge RTC :

   L'horloge RTC (Real Time Clock) va être le composant nous permettant d'horodater nos données. En effet comme son nom l'indique, ce composant renvoi la date et la date et l'heure précisément. De plus, même en cas de coupure de courant, cette horloge conserve la bonne heure et la bonne date car elle est équipé d'une pile lui permettant de continuer à fonctionner même sans alimentation de la part de la carte Arduino. Pour gérer l'horloge dans notre programme, nous utilisons la bibliotheque Arduino « RTClib.h ».

6. Capteurs :

   Notre système est équipé de deux capteurs, un capteur de température intégrant aussi l'humidité et la pression atmosphérique. Et un capteur de luminosité.

   Le capteur de température est le BME280 et va nous permettre de recueillir avec une précision au 0,1 degrés près la température. En plus de la température, il va capter le taux d'humidité en pourcentage et la pression atmosphérique. Pour le gérer dans notre programme, nous utilisons la bibliothèque Arduino « Adafruit_BME280.h ».

   Le capteur de luminosité quant à lui va se lire sur un port analogique de la carte Arduino. Le signal analogique reçu par la carte sera ensuite converti en une valeur numérique. Il faut ensuite mettre cette valeur numérique sur une échelle de 0 à 1023 afin qu'elle offre une résolution suffisante pour capturer des nuances de luminosité tout en étant rapide à calculer.

- Différents modes

Dans notre système de station météo, nous avons 4 modes :

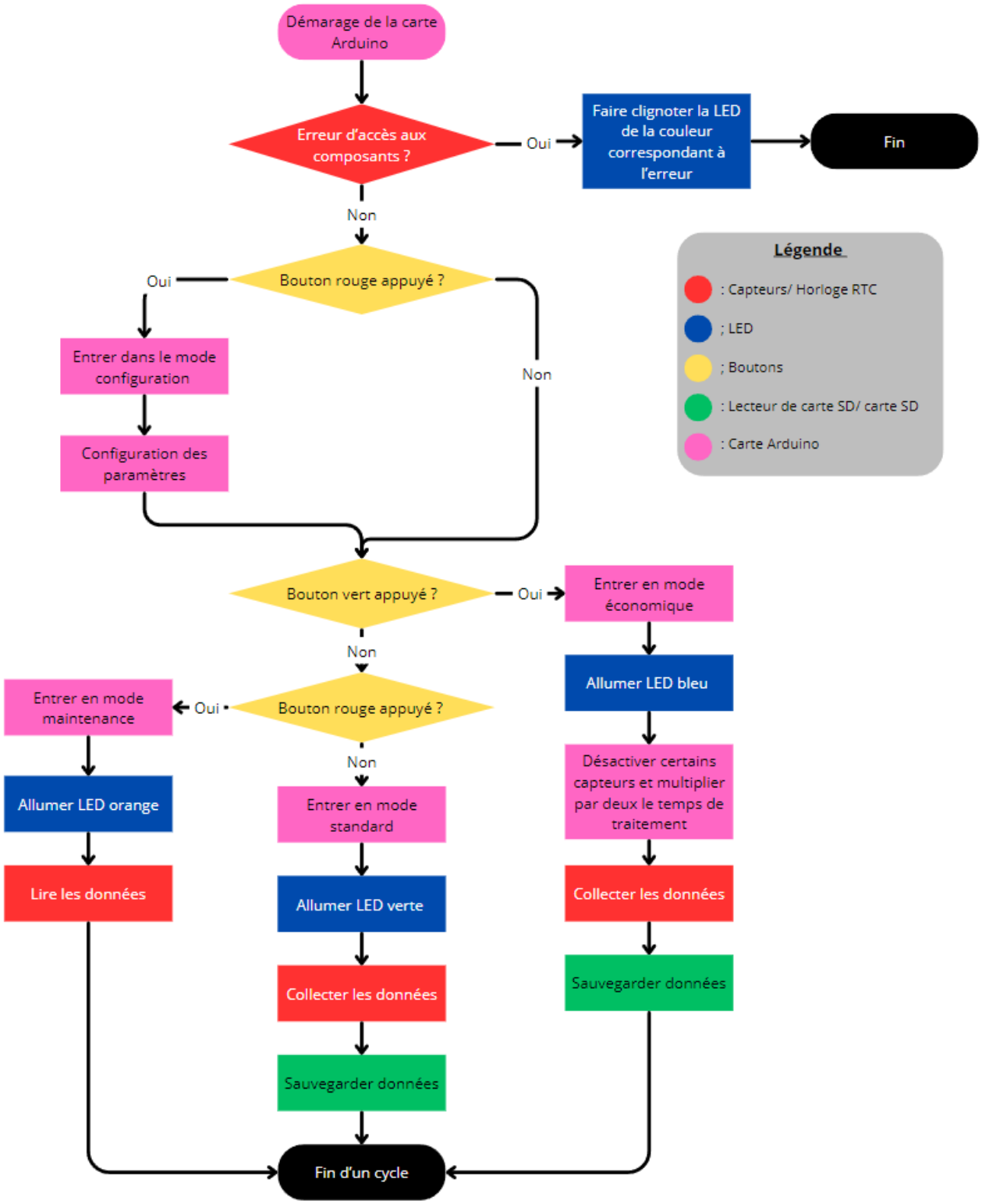
- Le mode standard : Ce mode permet de faire l'acquisition des données ; dans ce mode, le système collecte des valeurs/données des capteurs avec un intervalle par défaut de 10 minutes, configurable par le biais du paramètre « LOG_INTERVAL ». Toutes les mesures sont enregistrées sur une seule ligne horodatée ; si la donnée associée à un capteur ne répond pas au bout du temps défini par le paramètre « TIMEOUT », qui est par défaut à 30s, elle vaudra « NA ». Toutes les données acquises seront enregistrées dans un fichier de 2 Ko défini par le paramètre « FILE_MAX_SIZE » et prendront comme nom « 200531_0.LOG » avec « 20 » pour l'année, « 05 » le mois, « 31 » le jour et « 0 » le numéro de révision. Lorsque le fichier est plein, il crée une copie avec un nouveau numéro de révision et reprend l'écriture des données sur ce nouveau fichier. Ce mode peut être activé sans appuyer sur un bouton.

- Le mode configuration : Ce mode permet à l'utilisateur de configurer les paramètres du système et désactive l'acquisition des données des capteurs. Cette interaction se fait sur l'interface série. Par exemple, on peut utiliser « PRESSURE_MAX = 1030 », qui va fixer le seuil maximal de notre capteur de pression à 1030 HPa et a comme domaine de définition {300-1100}. L'utilisateur pourra donc modifier les paramètres maximaux et minimaux de chaque capteur et ajuster divers paramètres tels que l'horloge, la date ou encore le jour. Il peut être activé au début en appuyant sur le bouton rouge pendant 5 secondes. Si l'utilisateur reste inactif pendant un temps de 30 minutes, le système basculera automatiquement en mode standard.

- Le mode économique : Ce mode a pour objectif d'économiser de la batterie en désactivant certains capteurs et traitements. De plus, l'acquisition des données GPS se fait une fois sur deux, et l'intervalle entre deux mesures défini par le paramètre « LOG_INTERVAL » est doublé tant que le système reste dans ce mode. Ce mode peut être activé uniquement à partir du mode standard en maintenant le bouton vert enfoncé durant 5 secondes.

- Le mode maintenance : Ce mode a pour but de désactiver l'écriture, d'afficher et d'accéder aux données des capteurs enregistrées sur la carte SD via l'interface série, et permet de changer et de retirer la carte SD en toute sécurité sans corrompre les données enregistrées. Ce mode est activable uniquement dans le mode standard en maintenant le bouton rouge enfoncé pendant 5 secondes.

- Schéma des flux d'information



III. Architecture générale du programme

- Programme principal

Le programme principal est le corps du programme, c'est lui qui va s'occuper d'appeler les modes, de gérer les boutons et de définir les variables globales.

Notre programme commence par importer toutes les bibliothèques permettant de gérer la LED, l'horloge RTC, le capteur de température, la carte SD… Par exemple, la commande « #include <ChainableLED.h> » permet d'importer la bibliothèque qui gère la LED RGB.

Ensuite, il nous faut préciser sur quelle adresse/pin sont branchés nos composants afin de pouvoir les utiliser. Par exemple, « #define adresseI2CBME 0x76 » nous permet de définir que le capteur de température est branché sur l'adresse 0x76, ou encore avec « #define lightsensor A1 », on sait que le capteur de luminosité est branché sur le port analogique 1.

L'étape suivante est de définir toutes nos variables globales qui seront susceptibles d'être appelées ou modifiées dans l'ensemble du programme. Par exemple, « static bool GPS_Actif = true; » nous permet de déclarer une variable globale booléenne nous permettant de savoir si le GPS est actif ou non. Le fait d'avoir défini une variable globale pour savoir si le GPS est actif ou non nous permet de pouvoir la consulter ou la modifier dans l'ensemble du programme, donc dans chacun des 4 modes.

Une fois que toutes les variables et les paramètres de base sont définis, le programme peut passer à la fonction setup() qui, en Arduino, est la fonction exécutée au lancement de la carte. Dans notre programme, la fonction setup() ouvre une communication série de 9600 bauds permettant d'interagir avec le moniteur série dans la suite du programme.

Ensuite, elle initialise les entrées et les sorties de notre carte Arduino, soit elle initialise la LED avec « leds.init() », elle initialise les boutons en entrée avec « pinMode(RED_B, INPUT_PULLUP); » et elle initialise le lecteur de carte SD avec « pinMode(SS, OUTPUT); ».

Sachant que, en fonction de si le bouton rouge est appuyé pendant 5 secondes au démarrage ou non, le programme entrera en mode configuration ou non, la fonction setup doit savoir si le bouton rouge est appuyé pendant son exécution. C'est alors ce que fait le programme avec la condition « digitalRead(RED_B) == LOW ». Si cette condition est vraie, le mode configuration est lancé.

Ensuite, dans l'initialisation de la carte, nous devons vérifier s'il n'y a pas d'erreurs de lecture des composants. Pour cela, la fonction setup appelle une par une les fonctions permettant de savoir s'il y a une erreur pour chaque composant. S'il y a en effet des erreurs, ce sont les fonctions d'erreur qui les gèrent (voir la partie décrivant la gestion des erreurs).

La dernière étape de l'initialisation de la carte consiste à définir les interruptions. En effet, pour changer de mode avec les boutons, nous avons décidé d'associer les appuis des boutons à des interruptions. Elles permettent alors que, s'il y a un appui sur l'un des boutons, le programme arrête ce qu'il était en train de faire et exécute la fonction associée à l'interruption. Dans notre programme, dès le moment où on appuie sur un bouton, une fonction enregistre le moment où l'appui du bouton a commencé, et si le temps écoulé depuis l'appui du bouton est supérieur à 5 secondes et que le bouton n'a pas été relâché, alors la variable globale du mode à activer devient True et celle du mode à désactiver devient False.

Une fois l'initialisation terminée, le programme écrit dans le moniteur série « Initialisation terminée » afin d'avertir, puis passe à la fonction loop() qui sera exécutée en boucle.

Cette fonction loop() permet simplement de rediriger le programme dans les différents modes. En effet, cette fonction est une énumération de conditions et, si l'une des conditions est validée, alors la fonction du mode actif est exécutée. Concrètement, la fonction loop() vérifie quelle est la variable globale des modes qui vaut True, et celle qui vaut True a alors sa fonction associée exécutée.

- Mode standard

Le mode standard est composé de plusieurs fonctions, la première étant la fonction globale du mode standard. Cette fonction permet qu'une fois le mode standard activé, nous allumons la LED en vert afin que l'utilisateur sache dans quel mode il se situe. Ensuite, le programme capture le moment depuis lequel il a été lancé à l'aide de la fonction « millis() ». Une fois capturé, il le compare à « LOG_INTERVAL », qui représente le temps entre chaque capture de données. Si la valeur renvoyée par « millis() » est supérieure ou égale à « LOG_INTERVAL », alors le processus de traitement des données se lance ; sinon, il ne fait rien.

Pour optimiser le programme, nous avons décidé de stocker les variables globales dans l'EEPROM, ce qui permet de libérer de la place dans la SRAM.

Une fois le processus lancé, nous définissons la variable locale « previousLogTime » en lui assignant la valeur de « millis() ». Cela permet que dans la condition pour entrer dans le processus de traitement des données, nous puissions utiliser « millis() – previousLogTime ». Le résultat de cette soustraction représente alors le temps écoulé depuis le dernier traitement des données.

Une fois la gestion du temps effectuée, nous pouvons commencer à collecter les données. Cela commence par récupérer la date et l'heure afin d'avoir des données horodatées. Pour ce faire, nous définissons d'abord une variable locale de type « char » (caractère) d'une longueur de 40. Cette variable, qui va stocker la date et l'heure, est de type « char » car la fonction appelée pour récupérer l'heure renverra une chaîne de caractères contenant les informations de date et d'heure (jour, mois, année, heures, minutes et secondes).

Ensuite, nous appelons la fonction « lire_date_heure() », qui interroge l'horloge RTC et renvoie la chaîne de caractères contenant les éléments de la date et de l'heure.

Pour stocker les données capturées en attendant de les enregistrer sur la carte SD, nous créons des variables locales pour chaque donnée à enregistrer.

Ensuite, la fonction standard appelle les fonctions de lecture des données :

- « lumiere(int* raw_light, int* light) » : Elle prend en paramètres deux pointeurs, « raw_light » qui sera la valeur brute analogique captée par la carte Arduino et « light » qui va stocker la valeur de la lumière après la conversion. Pour ces deux variables, nous utilisons des pointeurs afin de ne pas les dupliquer lors de l'appel des fonctions et donc de gagner en espace mémoire. Cette fonction lit la valeur analogique du port de la carte où est branché le capteur de lumière, puis convertit cette valeur sur un intervalle de 0 à 1023 à l'aide de la fonction « map() ».
- « lire_temperature(float* temp, float* humi, float* press, float* alt) » : Sur le même principe que la fonction de capture de la lumière, elle prend en paramètres les pointeurs des variables locales définies précédemment et leur assigne les valeurs capturées par le capteur BME.

En ce qui concerne la pression, afin que la valeur soit en hectopascals, et sachant que la valeur brute captée est en pascals, nous divisons le résultat par 100. Pour que l'altitude approximative soit la plus précise possible, nous indiquons la pression atmosphérique moyenne, qui est de 1029 pascals, en paramètre de la fonction permettant de lire la donnée.

Pour terminer, la dernière étape du mode standard est d'enregistrer les données sur la carte SD. Pour ce faire, nous appelons la fonction « sauvegarde » en précisant en paramètres les valeurs stockées dans nos variables locales et correspondant aux données capturées par les différents capteurs.

La fonction « sauvegarde » commence par vérifier quel jour nous sommes afin de déterminer s'il faut remettre l'indice de révision à zéro. En effet, si la date a changé entre le dernier enregistrement et celui-ci, il faut remettre l'indice de révision à zéro car le nom de fichier change également.

Une fois cela fait, le nom du fichier est créé en appelant la fonction « generateFileName(revision) », prenant en paramètre le bon indice de révision. Cette fonction demande à l'horloge RTC la date afin de renvoyer le nom du fichier au format chaîne de caractères, sous la forme : « jour+mois+deux derniers chiffres de l'année+indice de révision.log ».

Ensuite, le programme revient dans la fonction « sauvegarde » pour ouvrir en écriture ou créer un fichier sous le nom précédemment généré. Les données sont alors toutes imprimées sur une ligne avec la date et l'heure.

Afin de s'assurer que le fichier ne dépasse jamais 2 Ko, nous vérifions sa taille après l'écriture. Si le fichier dépasse cette taille, l'indice de révision est incrémenté. Enfin, le programme ferme le fichier, et une boucle de la fonction standard est terminée. Tant qu'aucune interruption, comme un appui de bouton, ne se produit, le programme répète ces étapes en boucle.

- Mode économique

Le mode économique permet de gérer la collecte de données à des intervalles deux fois plus longs que le mode standard afin d'économiser de l'énergie. Voici comment il fonctionne :

On active le mode économique en appuyant au moins 5 secondes sur le bouton vert depuis le mode standard. Une fois l'appui réalisé, le système passe en mode économique, la LED RGB s'allume alors en bleu afin d'indiquer visuellement ce changement de mode.

La particularité majeure du mode économique réside dans l'intervalle de collecte des données, défini par LOG_INTERVAL, qui est doublé par rapport à celui du mode standard.

Pour gérer cela, le programme utilise la fonction « millis() » afin de suivre le temps écoulé depuis le dernier enregistrement. Si le temps écoulé dépasse le double de LOG_INTERVAL, le système procède à la collecte des données.

Pour collecter les données, le système commence par récupérer la date et l'heure actuelles en appelant la fonction « lire_date_heure() ». Cette fonction interroge le module RTC (Real-Time Clock) pour obtenir des informations précises sur le moment de la collecte, assurant ainsi que chaque enregistrement soit correctement horodaté.

Ensuite, il interroge le module GPS, mais la particularité du mode économique est que ce module, qui consomme beaucoup d'énergie, n'est activé qu'un enregistrement sur deux. Une variable booléenne « GPS_Actif » est alors utilisée pour alterner l'activation du GPS à chaque collecte de données. À chaque boucle, le programme inverse la variable « GPS_Actif » : si elle valait True, elle passe à False, et inversement. Le programme ne capture les données GPS que lorsque « GPS_Actif » vaut True, ce qui permet d'économiser de l'énergie en interrogeant le capteur GPS une fois sur deux.

Afin d’économiser encore plus d’énergie, les capteurs de température, d'humidité, de pression atmosphérique et de luminosité sont activés uniquement lorsque cela est nécessaire. La fonction « lire_temperature() » récupère les données du capteur BME280, tandis que la fonction « lumiere() » mesure la luminosité ambiante. Nous utilisons des pointeurs pour les paramètres de ces fonctions afin de minimiser l’espace mémoire occupé par le programme sur la carte Arduino.

Après la collecte des données, le système vérifie la cohérence des valeurs obtenues à l'aide de la fonction « donneeIncoherente() ». Cette vérification permet de s'assurer que les données capturées sont cohérentes par rapport à la plage de données attendue. Par exemple, une valeur de luminosité supérieure à 1023 ou inférieure à 0 serait considérée comme incohérente, déclenchant un message d'erreur via le moniteur série.

Une fois les données collectées et vérifiées, la fonction « sauvegarde() » est appelée pour enregistrer les informations sur la carte SD. Cette fonction ouvre un fichier ayant un nom de la forme « jour+mois+deux derniers chiffres de l’année+indice de révision.log » où sont enregistrées la date, l’heure, la température, l'humidité, la pression atmosphérique, l'altitude estimée et la luminosité. Avant l'écriture, la fonction erreur_SD() s'assure que la carte SD est correctement initialisée et accessible. Après l'enregistrement, le fichier est fermé pour préserver son intégrité.

- Mode configuration

On entre dans le mode configuration en appuyant sur le bouton rouge lors de l’allumage du système. Une fois dans ce mode, plusieurs étapes sont exécutées.

La première instruction réalisée dans ce mode est l’allumage de la LED RGB en jaune continu, pour indiquer visuellement l'entrée en mode configuration.

Ensuite, les capteurs sont désactivés afin d’éviter l’acquisition de données, permettant ainsi à l’utilisateur de configurer les paramètres sans interférences.

On passe ensuite dans la fonction « configurer_parametres(Serial.readString()) », qui détecte la commande à exécuter. Cette fonction prend en paramètre « Serial.readString() », qui permet de lire les instructions saisies par l’utilisateur dans le terminal. Si la commande est valide, les paramètres demandés seront modifiés dans l'EEPROM ou dans l'horloge RTC en fonction de l'instruction. Si la commande est invalide, aucun changement ne sera effectué et la fonction se terminera. Par exemple, si l’on écrit la commande « LOG_INTERVALL=10 » dans le moniteur serie, alors la valeur de LOG_INTERVALL stocké dans l’EEPROM vaudra 10.

Ensuite, le programme entre dans la fonction de détection de l'inactivité. Au début de celle-ci, un minuteur est démarré pour surveiller la durée d'inactivité. Si aucune commande n'est saisie, on reste dans une boucle pendant un maximum de 30 minutes, en vérifiant si une instruction est entrée dans le moniteur série. Si une commande est saisie, on retourne à la fonction « configurer_parametres(Serial.readString()) » et le minuteur est réinitialisé à 0.

Si le minuteur atteint 30 minutes sans activité, le programme quitte le mode configuration, réactive les capteurs et revient en mode standard.

- Mode maintenance

Le mode maintenance s'active depuis le mode standard en appuyant pendant 5 secondes sur le bouton rouge. Une LED orange s'allume alors en continu pour indiquer que le mode maintenance est actif. Ce mode permet de gérer de façon sécurisée les données et la carte SD. Voici les étapes de fonctionnement de ce mode.

Tout d'abord, le programme vérifie si la carte SD est correctement initialisée avec la commande « SD.begin(chipSelect) ». Si l'initialisation réussit, l'écriture sur la carte SD est désactivée en ouvrant le fichier en lecture seule grâce à l'option « FILE_READ ».

Ensuite, le programme ouvre la carte SD avec « SD.open() », puis affiche les données enregistrées sur celle-ci. Pour cela, il utilise une boucle « while » qui s'exécute tant que « dataFile.available() » est vrai. Cette commande vérifie la disponibilité des données. Dans la boucle, à chaque itération, les données enregistrées sur la carte SD sont lues avec « dataFile.read », puis affichées dans le moniteur série avec « Serial.write ». Pour lire les données de tous les fichiers présents sur la carte SD, le programme ouvre le répertoire racine avec « SD.open("/") ».

Une fois toutes les données lues, tous les fichiers sont fermés et la carte SD est désactivée pour pouvoir être retirée ou remplacée. Le programme ferme le répertoire et le fichier avec « root.close() » et « dataFile.close() », puis désactive la carte SD via « SD.end() ». Avant de retirer la carte SD, le programme met le Slave Select (ou parfois Chip Select) à « HIGH » pour le déconnecter du montage Arduino.

Si la carte SD rencontre un problème, la LED clignotera alternativement en rouge et en vert pour signaler l'erreur.

- Gestion des erreurs

Pour gérer les erreurs dans notre programme, nous avons décidé de créer une fonction pour chaque erreur qui pouvait apparaître.

Nous avons alors créé une fonction pour les erreurs :

- Erreur d'accès à l'horloge
- Erreur d'accès au capteur de température/pression atmosphérique/hygrométrie
- Erreur d'accès au capteur GPS
- Erreur d'accès à la carte SD
- Données reçues incohérentes

Dans chacune des fonctions gérant ces erreurs, le principe est le même. Nous regardons si l'initialisation du capteur renvoie 0 avec la condition « if (!SD.begin(chipSelect)) »; si c'est le cas, alors le capteur n'est pas reconnu et donc il y a une erreur. Dans ce cas, le programme allume la LED d'une couleur, attend un certain temps avec « delay(1500) » (1,5 seconde le plus souvent), allume la LED d'une autre couleur et ré-attend le même temps ou le temps *2 en fonction de l'erreur.

Les couleurs de la LED et les temps entre chaque couleur sont définis dans le manuel d'utilisation plus bas.

La gestion de l'erreur « données reçues incohérentes » est un peu différente ; au lieu d'interroger le capteur, la comparaison va être faite entre les données reçues des capteurs et les valeurs seuils de référence configurées par l'utilisateur. Si la valeur reçue du capteur est en dehors des valeurs de seuil, alors il affiche l'erreur via la LED, sinon il ne fait rien.

Les fonctions des différentes erreurs sont appelées dans l'initialisation et avant chaque utilisation des composants. Par exemple, avant d'interroger l'horloge RTC, le programme va appeler la fonction erreur RTC pour savoir si elle fonctionne bien.

L'erreur des données incohérentes est appelée après chaque réception de données.

IV. Superviseur final

- ## Démarrage du système

Pour démarrer notre système météo, deux options s'offrent à l'utilisateur :

- Démarrage en mode configuration
  Pour démarrer en mode configuration, il suffit de brancher notre système à une source de courant et d'appuyer dès le branchement sur le bouton rouge. Une fois la LED jaune allumée, restez appuyé pendant 5 secondes puis relâchez. Vous êtes alors en mode configuration.

- Démarrage en mode standard
  Pour démarrer en mode standard, il suffit de brancher notre système à une source de courant et d'attendre que l'initialisation soit terminée. Une fois l'initialisation terminée, le message « Initialisation terminée » apparaît dans le moniteur série et la LED s'allume en vert.

- ## Utilisation des différents modes

- Mode standard :
  Une fois en mode standard, l'utilisateur n'a pas grand-chose à faire. Les données vont être collectées puis enregistrées à un intervalle régulier défini par l'utilisateur grâce à « LOG_INTERVAL ». Pendant le mode standard, seul un message prouvant que les données ont bien été enregistrées sera écrit. Lorsqu'un nouveau fichier est créé, un message le prouvant est également écrit dans le moniteur série.

- Mode configuration :
  Le mode configuration permet de gérer les différents paramètres pour le bon fonctionnement de notre station météo. Une fois entré en mode configuration, vous avez 30 minutes pour configurer le système. Dans ce mode, vous pouvez écrire des commandes prédéfinies ci-dessous pour configurer la station météo. Ces commandes sont à entrer sous forme d'instructions et seront sauvegardées dans le système (dans l'EEPROM) pour les conserver lors des prochains redémarrages de la station météo.

Les différentes commandes que vous pouvez saisir sont :

- « LOG_INTERVAL=10 » → permet de définir l'intervalle entre deux mesures (par défaut la valeur est de 10 minutes).
- « FILE_MAX_SIZE=4096 » → permet de définir la taille maximale des fichiers.
- « RESET » → réinitialise l'ensemble des paramètres à leur valeur par défaut.
- « VERSION » → affiche la version du programme et un numéro de lot.
- « TIMEOUT=30 » → durée au bout de laquelle l'acquisition des données d'un capteur est abandonnée.
- « LUMIN=1 » → définit l'activation (1) ou la désactivation (0) du capteur de luminosité.
- « LUMIN_LOW=255 » → définit la valeur en dessous de laquelle la luminosité est considérée comme "faible" (par défaut, la valeur est 255).
- « LUMIN_HIGH=768 » → définit la valeur au-dessus de laquelle la luminosité est considérée comme "forte" (par défaut, la valeur est 768).
- « TEMP_AIR=1 » → définit l'activation (1) ou la désactivation (0) du capteur de température de l'air.
- « MIN_TEMP_AIR=-10 » → définit le seuil de température (en °C) en dessous duquel le capteur se mettra en erreur (par défaut, la valeur est -10).
- « MAX_TEMP_AIR=60 » → définit le seuil de température (en °C) au-dessus duquel le capteur se mettra en erreur (par défaut, la valeur est 60).
- « HYGR=1 » → définit l'activation (1) ou la désactivation (0) du capteur d'hygrométrie.
- « HYGR_MINT=0 » → définit la température en dessous de laquelle les mesures d'hygrométrie ne seront pas prises en compte (par défaut, la valeur est 0).
- « HYGR_MAXT=50 » → définit la température au-dessus de laquelle les mesures d'hygrométrie ne seront pas prises en compte (par défaut, la valeur est 50).
- « PRESSURE=1 » → définit l'activation (1) ou la désactivation (0) du capteur de pression atmosphérique.
- « PRESSURE_MIN=850 » → définit le seuil de pression atmosphérique (en HPa) en dessous duquel le capteur se mettra en erreur (par défaut, la valeur est 850).
- « PRESSURE_MAX=1080 » → définit le seuil de pression atmosphérique (en HPa) au-dessus duquel le capteur se mettra en erreur (par défaut, la valeur est 1080).
- « CLOCK=hh:mm:ss » → configure l'heure du jour dans l'horloge RTC (au format heure:minute:seconde).
- « DATE=MM/JJ/AAAA » → configure la date du jour dans l'horloge RTC (au format MOIS/JOUR/ANNÉE).

Après 30 minutes d'inactivité, le système repassera automatiquement en mode standard.

- Mode économique :
  Pour mettre la station météo en mode économique, vous devez être en mode standard et appuyer 5 secondes sur le bouton vert.
  Une fois cela fait, la LED devrait s'allumer en bleu, indiquant que vous êtes en mode économique. La LED peut mettre quelques secondes à s'allumer après l'appui de 5 secondes sur le bouton vert, car la station météo attend de finir son cycle de traitement des données avant de changer de mode.
  En mode économique, tout fonctionne comme en mode standard : vous n'avez rien à faire pour traiter les données. Des messages de confirmation, comme la confirmation de l'enregistrement des données, apparaîtront.
  La différence entre le mode économique et le mode standard est qu'en mode économique, les données sont enregistrées à un intervalle deux fois supérieur à celui du mode standard.
  Pour quitter le mode économique et revenir au mode standard, appuyez simplement 5 secondes sur le bouton rouge.

- Mode maintenance :
  Pour activer le mode maintenance, il faut être en mode standard et appuyer 5 secondes sur le bouton rouge. Une fois cela fait, la LED va s'allumer en orange, et le mode permettra à l'utilisateur d'accéder aux données des capteurs. Le programme affichera alors un message indiquant que l'écriture est désactivée et affichera ensuite sur l'interface série toutes les données des capteurs stockées jusqu'au répertoire racine (fichiers et dossiers). Vous pourrez les consulter directement sur l'interface série.
  Le mode ferme au fur et à mesure les fichiers et le répertoire racine, coupe toute communication entre le système et la carte SD, puis affiche un message pour vous prévenir que vous pouvez changer ou retirer la carte SD en toute sécurité.

- Gestion des erreurs et identification de la couleur de la LED

Les erreurs sont communiquées par le biais de la LED, si la LED de votre système s'allume de façon anormale, veuillez-vous référer au guide ci-dessous :

Couleur et fréquence du signal lumineux	Etat du système
LED verte continue	Mode standard
LED jaune continue	Mode configuration
LED bleue continue	Mode économique
LED orange continue	Mode maintenance
LED intermittente rouge et bleue (fréquence 1 Hz, durée identique pour les 2 couleurs)	Erreur d'accès à l'horloge RTC
LED intermittente rouge et jaune (fréquence 1Hz, durée identique pour les 2 couleurs)	Erreur d'accès aux données du GPS
LED intermittente rouge et verte (fréquence 1Hz, durée identique pour les 2 couleurs)	Erreur accès aux données d'un capteur
LED intermittente rouge et verte (fréquence 1Hz, durée 2 fois plus longue pour le vert)	Données reçues d'un capteur incohérentes - vérification matérielle requise
LED intermittente rouge et blanche (fréquence 1Hz, durée 2 fois plus longue pour le blanc)	Erreur d'accès ou d'écriture sur la carte SD

En cas d'erreur d'accès à l'horloge RTC ou d'accès aux données du GPS, cela signifie qu'ils ne sont plus reconnus par la carte Arduino ; vérifiez alors leur branchement sur l'un des ports I2C de la carte.

En cas d'erreur d'accès aux données d'un capteur, cela signifie aussi qu'ils ne sont pas reconnus par la carte Arduino ; vérifiez alors que le capteur de température, le BME 280, est bien branché sur l'un des ports I2C de la carte et que le capteur de luminosité est bien branché sur le port A1 de la carte.

En cas d'erreur de données reçues d'un capteur incohérentes, cela signifie que les valeurs que l'un des capteurs renvoie ne sont pas cohérentes. Cela est peut-être dû à un capteur mal réglé ou endommagé ; il faut donc se rapprocher du service technique.

En cas d'erreur d'accès ou d'écriture sur la carte SD, cela signifie que la carte SD n'est pas reconnue par la carte Arduino. Vérifiez alors qu'elle est bien insérée dans son lecteur et qu'elle ne soit pas endommagée.

V. Conclusion

Pour conclure, cette documentation technique permet de comprendre les différentes fonctionnalités de notre système. Nous avons 4 modes : le mode standard, le mode configuration, le mode maintenance et le mode économique. Ces différents modes permettent de gérer l'ensemble du système ainsi que de le configurer pour répondre le plus précisément aux besoins du client. De plus, elle permet de comprendre dans sa globalité l'utilisation et l'architecture du système avec ses différents paramètres configurables par le biais du mode configuration, l'acquisition des données par le mode standard, l'économie de la batterie grâce au mode économique et la gestion de la carte SD et des données par le mode maintenance. Son architecture simple permet de comprendre chaque procédure des différents modes et les interactions de toutes les fonctions ainsi que les flux d'informations qui sont échangés durant l'exécution du programme de notre système.