Projet Worldwide Weather Watcher

3. Documentation

DATE : Jeudi 22 Octobre 2020

EMETTEUR(s) : Eswann GIREAUD, Mattéo COFFIN, Théo SEROUSSI, Julie GILS

DESTINATAIRE(s) : Agence Internationale pour la Vigilance Météorologique (AIVM)

OBJET : Documentation technique et documentation utilisateur

INFORMATION GENERALE

Vous trouverez dans ce document le fonctionnement global du système avec flux d'information et l'architecture générale du programme, le but étant de vous présenter les choix opérés ainsi que de vous informer des performances du système. En plus de ce document, vous avez le document utilisateur (*Manuel Utilisateur WWW.pdf*) destiné aux membres de l'équipage qui seront amenés, nous l'espérons, à utiliser la station météo.

L'équipe CESI

SOMMAIRE

I) Schémas explicatifs

1. Diagramme d’état

Nous pouvons voir ci-dessus le diagramme d’état. Il illustre les différents états du système, ses transitions et ses éléments en action. Par rapports aux autres, ce diagramme apporte en plus le système de LED et leurs conditions d’activations. Il est important de rappeler que la sortie du mode maintenance s’effectue en direction du mode par lequel on y a accédé.

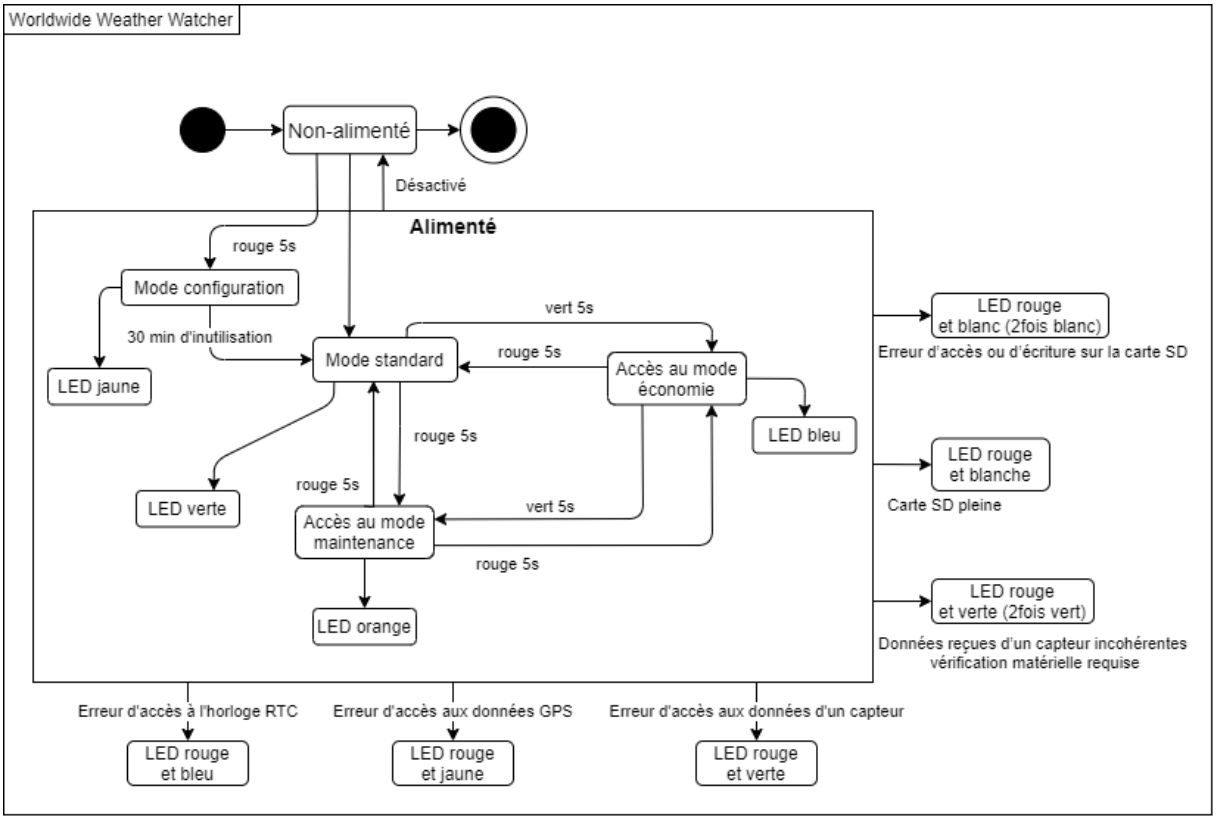

Fig 1 : diagramme d’état

2. Schéma de la structure globale simplifiée

Ce schéma nous offre une vue globale de la structure de notre programme. Il a pour but de faciliter l'approche et la compréhension du programme. Nous pouvons y voir les divers modes ainsi que les variables clefs permettant la communication entre les fonctions.

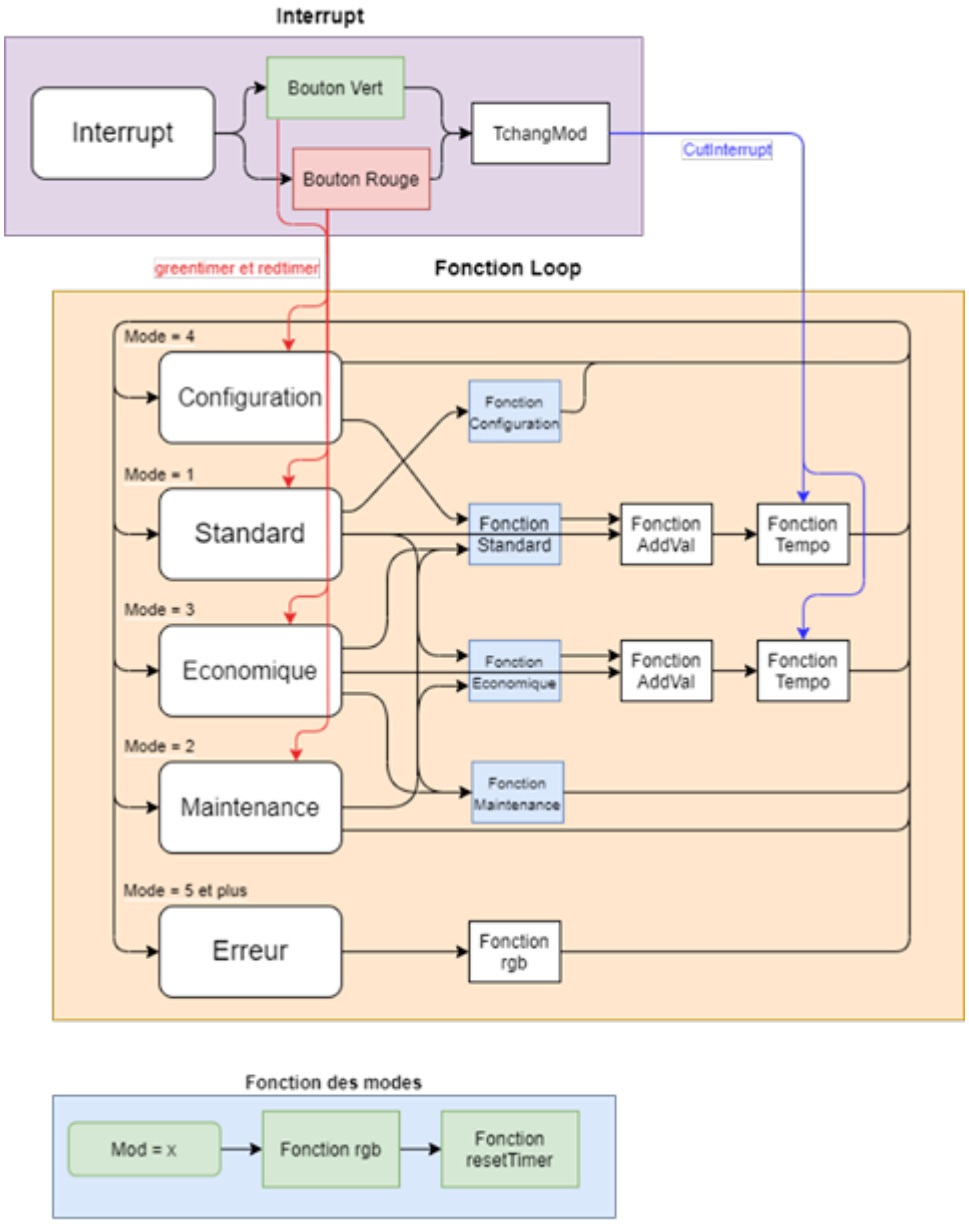

Fig 2 : Schéma de la structure globale simplifiée

3. Schéma des flux d'information

Voici le schéma des flux et des composants du système. Nous pouvons y observer les unités des données transmises.

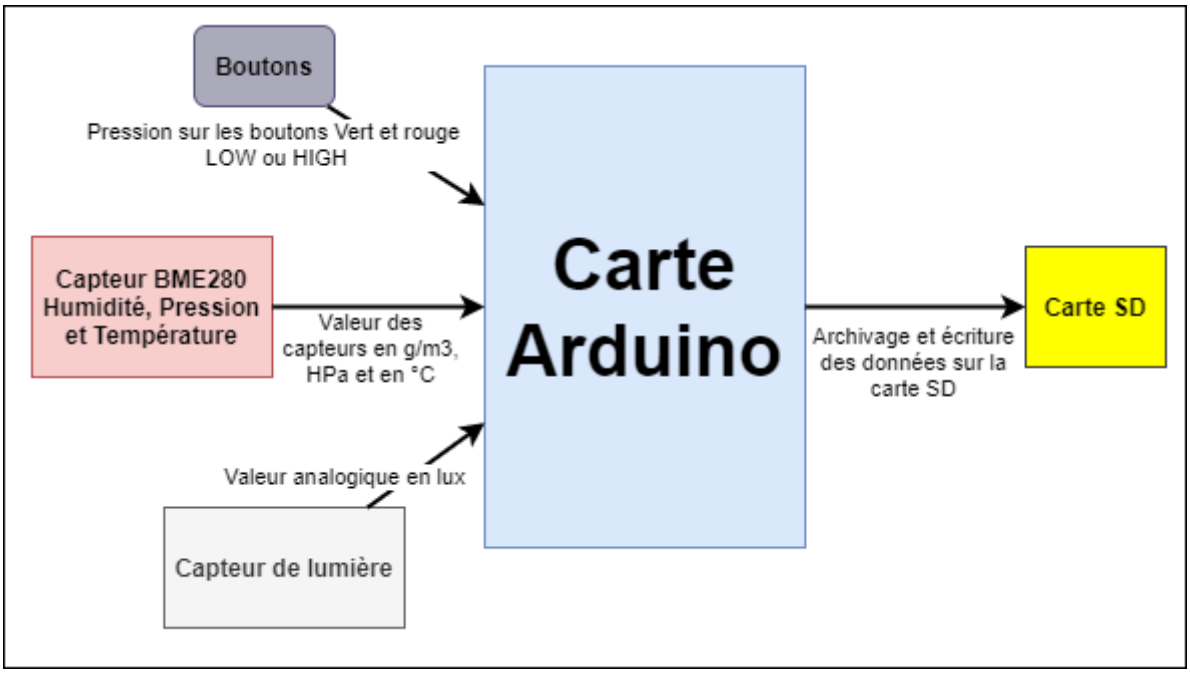

Fig 3 : Schéma de la structure globale simplifiée

II) Explication des fonctions et algorithmes

Afin de faire fonctionner les différentes fonctionnalités du système, nous avons écrit un programme Arduino composé de nombreuses fonctions. Dans cette partie, vous trouverez des explications claires pour chacune des fonctions ainsi que l'algorithme correspondant.

La fonction `interruption()` :

Cette fonction est appelée lorsqu'un évènement extérieur imprévisible a lieu, c'est-à-dire lorsque qu'un des deux boutons est pressé. Elle est constituée du numéro des broches, de 2 attachInterrupt qui servent à définir les fonctions à appeler, et ce qui va provoquer l'interruption : ici, le mot clef CHANGE signifie que l'interruption aura lieu lorsque la broche concernée va changer d'état (correspond à un appui sur le bouton poussoir ou un relâchement).

Traitement :

Appel de la fonction attachInterrupt avec comme paramètre :

Interrutpion = digitalPinToInterrupt(buttonPin1),

Fonction = boutonRouge

Mode = CHANGE

Appel de la fonction attachInterrupt avec comme paramètre :

Interrutpion = digitalPinToInterrupt(buttonPin2),

Fonction = boutonVert

Mode = CHANGE

La fonction `BoutonRouge (BoutonVert)` :

Cette fonction sert à connaître le temps depuis lequel le bouton rouge **(vert)** est pressé : en effet, lorsque l'évènement a lieu, la variable startred **(startgreen)** prend le temps t à partir duquel le bouton vient d'être pressé. Lorsqu'il est relâché, la variable redtimer **(greentimer)** prend la durée pendant laquelle le bouton est resté pressé en utilisant la valeur de startred **(startgreen)** et la fonction millis qui correspond au temps depuis lequel le programme a commencé son exécution. De plus, lorsque le bouton est relaché, la fonction appelle Tchangmod en prenant en paramètre redtimer **(greentimer)** et 1 **(0)** (autrement dit la valeur associée au bouton rouge **(vert)** pour valider le changement de mode.

boutonRouge

Initialisation :

Utilisation de « buttonPin1 » en pinMode « INPUT_PULLUP »

Utilisation des variables « redtimer », « greentimer » et « startred » de type « unsigned long »

Traitement :

SI buttonPin1 est pressé **ALORS**

| Affecter millis() à startred

SINON

| SI startred est différent de 0 **ALORS**

| **|**Affecter millis()-startred à redtimer

|FIN

FIN

boutonVert

Initialisation :

Utilisation de « buttonPin2 » en pinMode « INPUT_PULLUP »

Utilisation des variables « redtimer »,« greentimer » et « startgreen » de type « unsigned long »

Traitement :

SI buttonPin1 est pressé **ALORS**

| Affecter millis() à startgreen

SINON

| SI startgreen est différent de 0 **ALORS**

| |Affecter millis()-startgreen à greentimer

|FIN

FIN

La fonction `resetTimer()` :

Les deux variables redtimer et greentimer sont remis à 0. Ces deux variables correspondent à la durée depuis laquelle le bouton rouge ou le bouton vert est pressé. Les réinitialiser sert à pouvoir les utiliser de nouveau lors d'un nouveau comptage temporel.

Initialisation :

Utilisation des variables « redtimer » et « greentimer » de type int

Traitement :

Affecter 0 à redtimer

Affecter 0 à greentimer

La fonction Tempo() :

Cette fonction est équivalent à un delay() à la différence près qu'elle est interrompable pour passer à la suite.

Son fonctionnement est simple : c'est une boucle vide qui tourne tant qu'elle n'a pas atteint un certain temps ou qu'elle n'a pas été interrompue. L'interruption est mise en place par une variable (CutInterrupt) qui annule le temps d'attente quand elle est changée. CutInterrupt est changé quand un changement de mode est détecté grâce à la fonction TchangMod.

Initialisation :

Déclarer la variable MillisEnreg de type unsigned long.

MillisEnreg prend comme valeur le temps écoulé depuis le lancement du programme.

Récupération de la variable Attente qui était en paramètre de fonction.

Utilisation de la variable CutInterrupt de type bollean.

Utilisation de la variable Millis de type long.

Traitement :

TANT QUE (Millis **<** MillisEnreg + Attente*CutInterrupt **)**
FAIRE
|Rien
FIN
CutInterrupt prend la valeur 1 ;

La fonction `TchangMod()` :

Cette fonction s'active avec l'interrupt, lors du relâchement d'un bouton. Elle prend en paramètre le temps de pression et la couleur du bouton (1=Rouge, 0 =Vert).

Elle teste par la suite si les conditions sont réunies pour changer de mode. Les conditions sont : bouton appuyé plus de 5 sec, ou bien mode = 1 (Standard) plus Couleur = 1 (Rouge).

Si les conditions sont réunies, alors on change CutInterrupt à 0, cela dans le but de couper la fonction Tempo, et donc de changer de mode sans attente.

Initialisation :

Récupération de la variable Timer qui était en paramètre de fonction.

Récupération de la variable Rouge qui était en paramètre de fonction.

Utilisation de la variable CutInterrupt de type bollean.

Utilisation de la variable tps de type int.

Traitement :

SI (Timer > tps **OU (**mode = 1 **ET** Rouge = 1**)) ALORS**
| CutInterrupt prend la valeur 0 ;
FIN

L'Horloge `RTC()` :

Nous utilisons une horloge RTC DS1307 définit par la librairie DS1307.h. Cette horloge est connectée au bus I2C comprenant les lignes SDA (Serial Data) et SCL (Serial Clock) :

- SDA permet le transport de données.
- SCL permet la synchronisation.

L'horloge et ses paramètres sont définis dans le setup() permettant de mettre en marche l'horloge au début du programme.

La procédure d'erreur et simplement une définition de variable d'heure identique.

Bibliothèque : Librairie Wire.h (Cette bibliothèque vous permet de communiquer avec l'horloge RTC) et DS1307.h (le type d'horloge RTC).

Entrée :

ErreurHorloge en booléen

Initialisation :

Affecter ErreurHorloge à 0

Traitement :

Définir la date du jour et l'heure

Intégrer la date à l'horloge RTC

Si erreur horloge

| Affecter ErreurHorloge à 1

FIN

La fonction `valeur_maintenance()` **:**

Cette fonction est lancée lors de l'activation du mode maintenance. Elle sert à afficher proprement la valeur actuelle de chacun des capteurs.

Bibliothèque **:**

- Adafruit_Sensor.h permettant l'utilisation de capteur de la gamme Adafruit.
- Adafruit_BME280.h permettant l'utilisation du capteur BME280.

Entrées **:**

Toutes nos variables globales :

Initialisation **:**

Ecrit les valeurs du capteur BME280.

Ecrit la valeur du capteur light_sensor.

La fonction `mode_erreur(Erreur *actuel_erreur)` **:**

Cette fonction permettra à la Loop de sélectionner le mode d'erreur.

Entrées :

Erreur *actuel_erreur.

Initialisation :

Si erreur_hygrometrie ou erreur_temp ou erreur_ pression est égale à 1 **alors**

| la valeur mode est égale à 5

Sinon Si Erreur_Horloge est égale à 1 **alors**

| la valeur mode est égale à 6

Sinon Si Erreur_stockage est égale à 1 **alors**

| la valeur mode est égale à 7

Sinon Si Erreur_ecriture est égale à 1 **alors**

| la valeur mode est égale à 8

Sinon Si Erreur_acces est égale à 1 **alors**

| la valeur mode est égale à 9

La fonction `Carte_SD()`**:**

Nous avons une structure contenant la valeur de la Pression atmosphérique, la température de l'air, l'hygrométrie et la luminosité. Grâce à elles, nous pourrons écrire sur la carte SD les valeurs récupérées par les capteurs.

Elle permet aussi de déclarer des erreurs par rapport aux données des capteurs. La partie carte_SD procède par une conversion des variables jour/mois/année en char grâce à la fonction sprintf(Nom du fichier, « emplacement de variable.txt » , les différentes variables)

La carte SD dans le port shield utilise le chipSelect (pin) 4 permettant d'écrire ou d'accéder à la carte SD.

Nous utiliserons ce caractère pour nommer notre fichier.txt. Ce fichier.txt comprendra toutes nos valeurs du jour.

L'écriture des valeurs sera confrontée à diverse paramètre externe (si le capteur est désactivé ou bien s'il y a une erreur dans les valeurs).

Si une erreur est signalée la valeur écrite sera « na » et renverra une variable erreur dans la fonction erreur.

Les valeurs du capteur BME280 utilise deux librairies Adafruit_Sensor.h et Adafruit_BME280.h. La présence du capteur est définie par Adafruit_BME280 nom du capteur.

Pour lire ses valeurs nous avons juste à utiliser les fonctions des librairies (nom du capteur.readHumidity()).

Une exception est à signaler pour la partie Humidité lors de l'utilisation en mode économique une variable « eco » en booléen qui switch d'état à chaque passage ainsi une fois sur deux l'acquisition des données se fera.

Déclarer la variable data de type Terreur

Structure Terreur :

| Contient Erreur_pression

| Contient Erreur_temp

| Contient Erreur_hygrométrie

| Contient erreur_accés

| Contient erreur_Horloge

| Contient erreur_Stockage

| Contient erreur_Ecriture

Bibliothèque :

- DS1307.h pour l'horloge RTC.
- SD.h librairie pour l'usage du SD avec le shield.
- SPI.h librairie permettant d'utiliser le protocole de données série synchrone(SPI).
- structure.h notre structure de données (Erreur).
- Adafruit_Sensor.h permettant l'utilisation de capteur de la gamme Adafruit.
- Adafruit_BME280.h permettant l'utilisation du capteur BME280.

Traitement **:**

Entrées :

Toutes nos variables globales :

Initialisation :

Définir jour / mois et année les valeurs de l'horloge RTC.
Convertir les valeurs entier (jour/mois/année) en char avec .txt.
Définir myFile comme fichier avec pour nom **(**jour/mois/année).

Traitement :

Si myFile à était correctement créer **alors**
| **Si** LUMIN est égale à 1 **alors**
| | Si LUMIN_HIGH est plus petit que la valeur lue **alors**
| | | écrire les valeurs sur la carte SD
| | | Spécifié que la valeur est forte
| | Sinon Si LUMIN_LOW est plus grand que la valeur lue **alors**
| | | écrire les valeurs sur la carte SD
| | | Spécifier que la valeur est faible

| **Si** TEMP_AIR est égale à 1 **alors**
| | Si MAX_TEMP_AIR est plus petit que la valeur lue **ou** MIN_TEMP_AIR est plus grand que la valeur lue **alors**
| | | Ecrire « na » sur le SD .

| | | Envoyer 1 sur la structure d'erreur.
| | Sinon
| | | Ecrire les valeurs sur le SD.

| Si PRESSURE est égale à 1 **alors**
| | Si PRESSURE_MAX est plus petit que la valeur lue **ou** PRESSURE_MIN est plus grand que la valeur lue **alors**
| | | Ecrire « na » sur le SD .
| | | Envoyer 1 sur la structure d'erreur.
| | Sinon
| | | Ecrire les valeurs sur le SD.

| Si HYGR est égale à 1 **alors**
| | SI le mode sélectionner est 3 **alors**
| | | Si eco est égale 1 **alors**
| | | | Si HYGR_MAXT est plus petit que la valeur lue **ou** HYGR_MINT est plus grand que la valeur lue **alors**
| | | | | Ecrire « na » sur le SD .
| | | | | Envoyer 1 sur la structure d'erreur.
| | | | Sinon
| | | | | Ecrire les valeurs sur le SD.
| | Sinon Si HYGR_MAXT est plus petit que la valeur lue **ou** HYGR_MINT est plus grand que la valeur lue **alors**
| | | Ecrire « na » sur le SD .
| | | Envoyer 1 sur la structure d'erreur.
| | Sinon
| | | Ecrire les valeurs sur le SD.
| eco prend la valeur de son opposé.
| Fermer myFile
Sinon
| Ecrire message d'erreur
| Envoyer 1 sur la structure d'erreur.
Fin

Les modes : fonctions `Standard(), Configuration(), Maintenance(), Economique():`

Lorsqu'une fonction mode est appelée, la variable mode change et prend la valeur correspondante. La LED s'allume en utilisant la fonction leds.setColorRGB() et en lui donnant en paramètre les intensités de bleu, de vert et de rouge requises afin que la lumière soit de la couleur demandée. Pour finir, la fonction resetTimer est appelée afin de réinitialiser les timer des boutons rouge et vert pour un prochain changement de mode.

Nom de mode	Valeur de la variable mode	Couleur	Paramètres de leds.setColorRGB
Standard	1	Vert	0.255.0
Maintenance	2	Orange	255.40.0
Economique	3	Bleu	0.0.255
Configuration	4	Jaune	255.125.0

Bibliothèque pour toutes les fonctions mode : chainableLED.h

La fonction `Standard()` **:**

Initialisation :

Utilisation de la variable « mode » de type int

Traitement :

Affecter 1 à mode
Afficher la led en vert.
Afficher « mode standard »
Appeler fonction resetTimer()
Appeler fonction erreur()

La fonction `Maintenance()` **:**

Initialisation :

Utilisation de la variable « mode » de type int

Traitement :

Affecter 2 à mode
Afficher la led en orange.
Afficher « mode maintenance »
Appeler fonction resetTimer()
Appeler fonction erreur()

La fonction `Economique()` **:**

Initialisation :

Utilisation de la variable « mode » de type int

Traitement :

Affecter 3 à mode

Afficher la led en bleue.

Afficher « mode economique »

Appeler fonction resetTimer()

Appeler fonction erreur()

La fonction `Configuration()` :

Entrées :

Toutes les valeurs globales.

Initialisation :

la valeur mode est égale à 4.

Allume la LED en jaune.

Définit une variable locale (path_config) , elle permettant de sauvegarder nos choix.

Définit une variable locale (etape_config) , elle permettant de voyager entre les config.

Ecrire un texte interrogant le souhait de l'utilisateur (« Définir vos paramètres par défaut ? »).

Tant que etape_config est égale à zéro **alors**

| Si vous entrez un caractère sur le moniteur **alors**

| | path_config prend la valeur de ce caractère.

| | etape_config prend la valeur 1.

Si la valeur est égale à « oui » **alors**

| Définit les paramètres en défaut.

Sinon

| Modifie les paramètres.

LE LOOP

Il s'agit de la fonction principale d'un programme Arduino, celle qui s'exécutera en boucle (comme son nom l'indique).

Dans notre cas, elle servira à communiquer à la carte les instructions requises selon le mode choisi. Pour cela, nous avons utilisé l'instruction `switch … case` qui sert à ordonner des instructions précises en fonction de la variable mode.

Le loop est commencé avec le `case 0 :` le démarrage. Lorsque l'on met en route le système, il passe directement en mode standard.

Ensuite, pour chaque mode (`case 1, case 2, case 3, case 4`), des boucles `if` sont utilisées pour tester les valeurs de redtimer et greentimer selon les possibilités de changement de mode. Une fois rentré dans ces boucles if, les fonctions de modes sont alors appelées pour effectuer la transition.

Le `case 4` (mode configuration) est un peu spécial. Il y a deux moyens d'en sortir : soit en appuyant 5 secondes ou plus sur le bouton vert. Soit en attendant 30 minutes. Dans ce cas-là, il y a un timer (TimerConfig) qui fonctionne comme le green et redtimer. Il s'initialise en enregistrant le millis() une première fois, puis test en boucle si le temps passé depuis est supérieur aux 30 minutes. Pour que ces « 30 minutes d'attentes » se transforment en « 30 minutes d'inactivité », il suffit de réenregistré le millis() de départ juste après une modification manuel dans le mode configuration.

Pour finir, les `case 5, 6, 7, 8 et 9` servent pour les différentes erreurs possibles. Dans chaque `case,` on retrouve la fonction leds.setColorRGB afin d'éclairer la LED selon le code couleur fourni pour les erreurs.

Initialisation :

Utilisation des variables « mode » et « etat_precedent » de type « int »

Utilisation des variables « redtimer », « greentimer » et « startred » de type « unsigned long »

Utilisation de la constante « tps » égale à 5000

Traitement :

SI mode est égal à 0 **ALORS**
| Appeler fonction Standard()
FIN
SI mode est égal à 1 **ALORS**
|**SI** redtimer est supérieur ou égal à tps **ALORS**
| |Appeler fonction Maintenance()
| |Affecter 1 à etat_precedent
|**FIN**
|**SINON SI** greentimer est supérieur ou égal à tps **ALORS**
| |Appeler fonction Economique()
|**FIN**
|**SINON SI** redtimer est compris entre 1 et tps **ALORS**
| |Appeler fonction Configuration()
|**FIN**
|**SINON** Appeler fonction AddVal()
|Appeler fonction Tempo() avec le temps à attendre en paramètre
|**FIN**

FIN
SI mode est égal à 2 **ALORS**
|**SI** redtimer est supérieur ou égal à tps **ALORS**
| |**SI** etat_precedent est égal à 1 **ALORS**
| | |Appeler la fonction Standard()
| |**FIN**

```
|       |SINON SI etat_precedent est égal à 3 ALORS
|       |       |Appeler la fonction Economique()
|       |FIN
|FIN
FIN
SI mode est égal à 3 ALORS
| SI greentimer est supérieur ou égal à tps ALORS
|       |Appeler la fonction Standard()
|FIN
|SINON  SI redtimer est supérieur ou égal à tps ALORS
|       |Appeler la fonction Maintenance()
|       |Affecter 3 à etat_precedent
|FIN
|SINON Appeler fonction AddVal()
       |Appeler fonction Tempo() avec le temps à attendre en paramètre
|FIN
FIN
SI mode est égal à 4 ALORS
|SI greentimer > tps ALORS
|       | Appeler la fonction Standard()
|FIN
|SINON SI TimerConfig = 0 ALORS
|       | TimerConfig = Millis
|FIN
|SINON SI Millis > TimerConfig + 10sec  ALORS
|       | Appeler la fonction Standard()
|       | TimerConfig prend la valeur 0
|FIN
FIN

SI mode est égal à 5 / 6 / 7 / 8 / 9 ALORS
     |Allume la led correspondant à l'erreur.

FIN
```

III) Montage en images

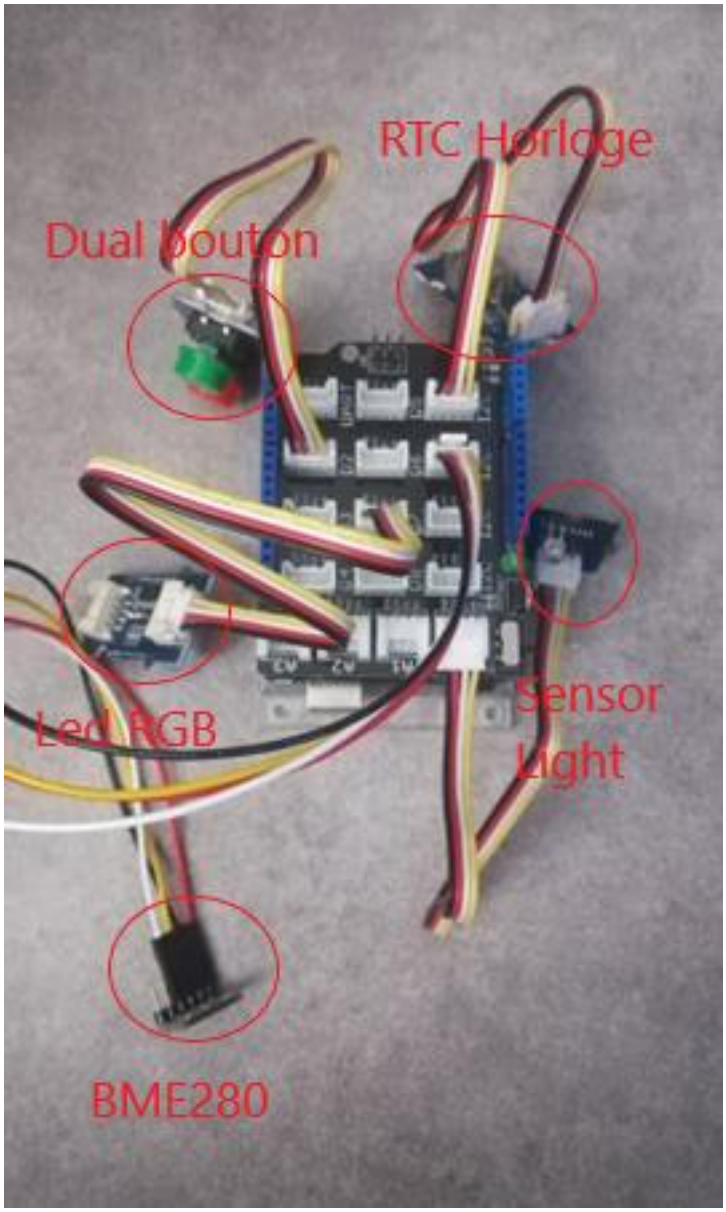

Fig 4 : Photo du montage du système

- Dual bouton : initialisé dans le bus D2 correspondant aux pins digitaux 2 et 3 nécessaires pour la fonction interrupteur.
- RTC Horloge : initialisée dans le bus I2C comprenant les lignes SDA (Serial Data) et SCL (Serial Clock).
- Sensor Light : initialisé dans le bus A0 (la partie analogue).
- BME280 : Initialisé dans le bus I2C comme l'horloge
- Led RGB : initialisé dans le bus D7 correspondant aux pins digitaux 7 et 8.

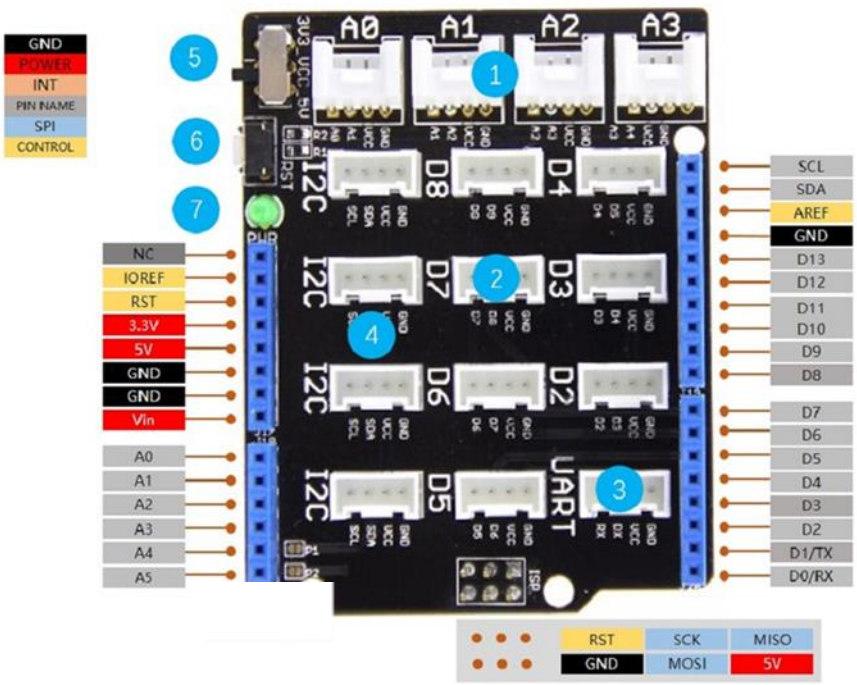

Fig 3 : Image des ports du système

1. **Ports analogiques** : comprennent 4 ports analogiques, A0, A1, A2 et A3.
2. **Ports numériques** : comprennent 7 ports numériques, D2, D3, D4, D5, D6, D7 et D8.
3. **Port UART** : 1 port UART.
4. **Ports I2C** : 4 ports I2C.
5. **Commutateur Power** : lorsque vous utilisez Arduino UNO avec Base Shield v2, veuillez tourner le commutateur en position 5v; Lors de l'utilisation de Seeeduino Arch avec Base Shield v2, veuillez régler le commutateur sur 3.3v.
6. **Reset Buton** : réinitialise la carte arduino.
7. **LED PWR** : La LED verte s'allume lors de la mise sous tension.

CONCLUSION

Voici donc la documentation technique à fournir au technicien en cas de problème du produit. Un diagramme d'état pour comprendre comment fonctionne la station météo, un schéma de la structure, l'explication de l'algorithme utilisé pour faire marcher la station météo et les différents modes de fonctionnement ainsi que les photos du produit avec les capteurs, les LEDs, l'horloge RTC… Vous avez donc tous les éléments pour pouvoir maintenir en état le world weather watcher. Nous vous souhaitons une bonne utilisation du produit et n'hésitez pas à revenir vers nous en cas de questions supplémentaires.

FIN DU DOCUMENT