Livrable 3

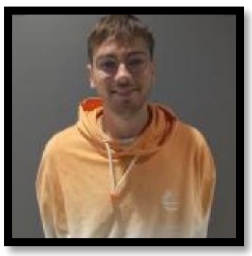
PROJET WORLDWIDE WEATHER WATCHER

Réalisé par :

Phan Cong Laurent

Le Nozahic Corentin

Perez Le Tiec Tom



Lapert Marceau

CESI
ÉCOLE D'INGÉNIEURS

Table des matières

Contexte global :

Le projet "Worldwide Weather Watcher" représente une initiative visant à développer une station météorologique avancée, capable d'examiner diverses situations météorologiques pour anticiper les conditions atmosphériques sur un plan mondial. Cette mission nous a été confiée à notre agence, qui jouera un rôle fondamental dans la réalisation de ce projet.

Contexte de ce livrable :

Le but de ce livrable va être de réaliser les branchements Arduino ainsi que de programmer celle-ci, ce qui comporte :

- Un programme complet de la carte.
- Un script de compilation/téléversement.
- Un montage minimal centré sur la carte Arduino pour une démonstration.
- Un scénario de démonstration permettant de prouver l'efficacité de la réalisation.

Programme de la carte Arduino :

Étant donné la taille de notre programme (), nous avons décidé de le diviser en plusieurs parties afin d'expliquer au mieux son fonctionnement, étape par étape.

Partie #include fondamentale :

Tout d'abord nous allons importer quelques bibliothèques qui sont nécessaires au fonctionnement de notre station météo :

```
#include <Arduino.h>
#include <Wire.h>
#include <RTClib.h>
#include <SPI.h>
#include <SD.h>
#include <ChainableLED.h>
#include <BME280I2C.h>
#include <avr/pgmspace.h>
#include <SoftwareSerial.h>
```

Sert à utiliser les fonctions du capteur

Va nous permettre d'utiliser l'horloge RTC dans notre système

Sert à avoir une relation de maitre/esclaves entre les différents composants.

Sert à utiliser les fonctions utiles à la programmation d'une carte SD.

Sert à utiliser la LED RGB.

Sert à utiliser le capteur de température, de pression et d'humidité.

Permet de déclarer des variables et des constantes dans la mémoire flash.

Permet d'utiliser les broches numérique Arduino comme port série

Partie variables globales :

```
uint8_t ereurLum = 0;
uint8_t ereurtemp = 0;
uint8_t ereurhum = 0;
uint8_t ereurpres = 0;
uint8_t ereurSD = 0;
```

Nous déclarons ici les variables que nous allons utiliser dans la partie « Erreur » de notre programme que nous verrons par la suite. Notons que « uint8_t » est un alias pour « unsigned char », il est souvent utilisé dans les systèmes embarqués à mémoire limitée et permet de stocker les données dans la mémoire flash.

Pour la suite du code, nous avons décidé d'ajouter des commentaires pour des raisons de lisibilités.

```
int8_t LUMIN_LOW = 200; // Définit les valeurs minimum pour la luminosité
int LUMIN_HIGH = 1023; // Définit les valeurs maximum pour la luminosité
int8_t MIN_TEMP_AIR = -5; // Même chose pour la température
int8_t MAX_TEMP_AIR = 30;
int8_t HYGR_MINT = 0; // Même chose pour l'humidité
int8_t HYGR_MAXT = 50;
int PRESSURE_MIN = 450; // Même chose pour la pression
int PRESSURE_MAX = 1030;
uint8_t LOG_INTERVALL = 1; // Définit l'intervalle de journalisation en minutes
uint16_t FILE_MAX_SIZE = 2048; // Définit la taille max de notre fichier à 2 ko.
int8_t LUMIN = 1;
int8_t TEMP_AIR = 1;
int8_t HYGR = 1;
int8_t PRESSURE = 1;

unsigned long previousMillis = 0; // Déclare les variables de temps
unsigned long interval = LOG_INTERVALL * 60UL * 1000UL;
```

« Unsigned » permet d'éviter d'aller dans les négatifs et donc d'aller de 0 à une certaine valeur.

Une constante est une fonction qui permet à notre valeur de ne pas changer.

« Byte » est l'équivalent d'un « int », il permet de déclarer une valeur entière.

SD :

```
const byte PROGMEM chipSelect = 4;
// Déclare la variable chipSelect comme une constante entière égale à 4.
File fichier; // Permet d'accéder et de manipuler un fichier sur la carte SD

void printSD(); // On appelle ici la fonction qui va nous permettre d'initialiser
// la carte SD
```

Boutons :

La fonction volatile va nous permettre de changer l'état d'une variable quand on appui sur le bouton dans notre cas c'est un booléen donc état 0 ou 1.

```
SoftwareSerial softSerial(2, 3); // Permet d'utiliser le pin 2 et 3
const byte PROGMEM redBtn = 3; // Le bouton rouge est sur le pin 3
const byte PROGMEM greenBtn = 2; // Le bouton vert est sur le pin 2

volatile bool redBtnState = false; //L'état du bouton rouge est "non-appuyé"
unsigned int redBtnTime = 0;

volatile bool greenBtnState = false; //L'état du bouton vert est "non-appuyé"
unsigned int greenBtnTime = 0;
void buttonRedInterrupt(); // Appel de la fonction d'interruption du bouton red
void buttonGreenInterrupt(); // Même chose pour le bouton vert
```

Horloge :

```
RTC_DS1307 rtc; // Créé une instance de la classe RTC_DS1307
// pour avoir horloge/calendrier
String date(); // Ecrit la date
```

LED :

```
ChainableLED leds(7, 8, 1); // Initialisation de la LED
```

Capteur de luminosité :

```
void CLumiere(int8_t n); // Appel de la fonction
int lightSensor = A3; // Capteur de luminosité sur le port A3
```

Capteur BME (température, pression et humidité) :

```
BME280I2C bme; Créé une instance de la classe BME280I2C

void BMETemp(Stream* client, int8_t n); // Appel des fonctions pour BME
void BMEhum(Stream* client, int8_t n);
void BMEpres(Stream* client, int8_t n);
```

Mode :

```
uint8_t mode = 0; // Déclare le mode à 0
uint8_t previous_mode = 0; // Déclare le mode précédennt à 0
```

Partie setup() :

```
void setup() {
  Serial.begin(9600); // initialise le port série
  pinMode(redBtn, INPUT_PULLUP); // Configure la broche redBtn comme entrée
  // avec résistance de tirage activée
  pinMode(greenBtn, INPUT_PULLUP); // Même chose mais pour la broche greenBtn
  Wire.begin(); // Initialise la communication I2C

  if (!rtc.begin()) {
    Serial.println(F("Could not find RTC!")); // Affichage de l'erreur
    while (1) {
      leds.setColorRGB(0, 0, 0, 255);
      delay(500);
      leds.setColorRGB(0, 255, 0, 0);
      delay(500);
    }
  }

  // Si l'initialisation du RTC réussit,
  // ajuste l'heure à la date et à l'heure de compilation
  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

  pinMode(chipSelect, OUTPUT); // Configure la broche chipSelect comme sortie

  if (SD.begin(4)) { // Initialisation de la carte SD
    Serial.println(F("Carte SD initialisée."));
  } else {
    Serial.println(F("Erreur lors de l'initialisation de la carte SD."));
    while (1)
    {
      leds.setColorRGB(0, 255, 0, 0);
      delay(500);
      leds.setColorRGB(0, 255, 255, 255);
      delay(1000);
    }
    return;
  }
```

```
  attachInterrupt(digitalPinToInterrupt(redBtn), buttonRedInterrupt, FALLING);
  // // Attache une interruption à la broche redBtn
  attachInterrupt(digitalPinToInterrupt(greenBtn), buttonGreenInterrupt, FALLING);

  while(!Serial) {} // Attendre

  while(!bme.begin())
  {
    Serial.println(F("Not find BME280"));
    delay(1000);
    while (1)
    {
      leds.setColorRGB(0, 255, 0, 0);
      delay(500);
      leds.setColorRGB(0, 0, 255, 0);
      delay(500);
    }
  }

  switch(bme.chipModel()) // Initialisation du capteur BME
  {
     case BME280::ChipModel_BME280:
       Serial.println(F("BME280 success."));
       break;
     case BME280::ChipModel_BMP280:
       Serial.println(F("No Humidity available."));
       break;
     default:
       Serial.println(F("UNKNOWN sensor"));
  }
  delay(7000); // Attente de 7 secondes
    if(digitalRead(redBtn) == LOW){ // Si le bouton est appuyé au début du
// démarrage, le système passe en mode configuration
      mode = 2;
      Serial.println("Mode configuration");
    }
    else{
      mode = 0;
      Serial.println("Mode standard");
  }
}
```

Nous avons globalement dans ce setup() :

- L'initialisation du port série.
- L'initialisation de la communication SPI.
- L'initialisation de l'horloge RTC.
- L'initialisation de la carte SD.
- L'initialisation du capteur BME.
- Un code qui permet, à la pression du bouton rouge, le passage du mode configuration au démarrage du système.

Fonctionnement du programme :

Nous allons maintenant nous pencher sur le fonctionnement de notre programme. Nous avons décidé de fractionner notre programme de telle manière à ce que l'on comprenne comment chaque partie fonctionne. Nous allons ici revoir sur certaines parties les initialisations afin de comprendre du début à la fin le déroulé de chaque partie.

LED :

Fonction initialisation :

```
ChainableLED leds(7, 8, 1);
```

Utilisation dans le code :

```
leds.setColorRGB(0,R,G,B);
```

Le « R » représente le rouge, le « G » le vert et le « B » le bleu.

Capteur de luminosité :

Fonction globale et prototype relatif au capteur de luminosité

```
void CLumiere(int8_t n);
int lightSensor = A3;
```

La variable « lightSensor » garde en mémoire sur que pin le capteur est brancher

Fonction relative au capteur de luminosité

Le « void CLumiere(int8_t n) ; » appelle la fonction ci-dessous :

```
void CLumiere(int8_t n) {
  if (LUMIN_LOW <= analogRead(lightSensor) && analogRead(lightSensor) <= LUMIN_HIGH) {
    if (n == 1) {
      fichier.print(" luminosité : ");
      fichier.print(map(analogRead(lightSensor), 0, 1023, 0, 100));
      fichier.print(", ");
    } else {
      Serial.print(F(" luminosité : "));
      Serial.print(map(analogRead(lightSensor), 0, 1023, 0, 100));
      Serial.print(F(", "));
    }
  } else {
    if (n == 1) {
      fichier.print(" luminosité : erreur, ");
    } else {
      Serial.print(F(" luminosité : erreur, "));
    }
    if (ereurLum==0){
      leds.setColorRGB(0,255,0,0);
      delay(500);
    }else{
      leds.setColorRGB(0,0,255,0);
      delay(500);
    }
    ereurLum = !ereurLum;
  }
}
```

Cette fonction est représentative de la structure utilisée pour chaque capteur.

Elle fait d'abord un test pour savoir si les valeurs sont cohérentes avec les valeurs maximales et minimales que l'on a définie en amont sinon renvoie une erreur d'incohérence.

Nous avons aussi écrit un code qui permet de dissocier le « print » du moniteur série du « print » sur la carte SD. Si c = 1, le code « print » sur la carte SD sinon sur le moniteur série.

Capteur BME :

Variable globale et prototype relatif au capteur de luminosité :

```
BME280I2C bme;
#define SERIAL_BAUD 9600

void BMETemp(Stream* client, int8_t n);
void BMEhum(Stream* client, int8_t n);
void BMEpres(Stream* client, int8_t n);
```

Initialisation du capteur dans le setup() :

```
while(!bme.begin())
{
  Serial.println(F("Not find BME280"));
  delay(1000);
  while (1)
  {
    leds.setColorRGB(0, 255, 0, 0);
    delay(500);
    leds.setColorRGB(0, 0, 255, 0);
    delay(500);
  }
}

switch(bme.chipModel())
{
   case BME280::ChipModel_BME280:
     Serial.println(F("BME280 success."));
     break;
   case BME280::ChipModel_BMP280:
     Serial.println(F("No Humidity available."));
     break;
   default:
     Serial.println(F("UNKNOWN sensor"));
}
```

Permet de vérifier si les capteurs sont présents.

Fonctions relatives au capteur BME :

Température

```
void BMETemp(Stream* client, int8_t n) {
  bme.read(pres, temp, hum);
  if (MIN_TEMP_AIR <= temp && temp <= MAX_TEMP_AIR) {
    if (n == 1) {
      fichier.print("température : ");
      fichier.print(temp);
      fichier.print(", ");
    } else {
      Serial.print(F("température : "));
      Serial.print(temp);
      Serial.print(F(", "));
    }
  } else {
    if (n == 1) {
      fichier.print("température : erreur, ");
    } else {
      Serial.print(F("température : erreur, "));
    }
    if (ereurtemp==0){
      leds.setColorRGB(0,255,0,0);
      delay(500);
    }else{
      leds.setColorRGB(0,0,255,0);
      delay(500);
    }
    ereurtemp = !ereurtemp;
  }
}
```

Humidité

```
void BMEhum(Stream* client, int8_t n) {
  bme.read(pres, temp, hum);
  if (HYGR_MINT <= hum && hum <= HYGR_MAXT) {
    if (n == 1) {
      fichier.print("humidité : ");
      fichier.print(hum);
      fichier.print(", ");
    } else {
      Serial.print(F("humidité : "));
      Serial.print(hum);
      Serial.print(F(", "));
    }
  } else {
    if (n == 1) {
      fichier.print("humidité : erreur, ");
    } else {
      Serial.print(F("humidité : erreur, "));
    }
    if (ereurhum==0){
      leds.setColorRGB(0,255,0,0);
      delay(500);
    }else{
      leds.setColorRGB(0,0,255,0);
      delay(500);
    }
    ereurhum = !ereurhum;
  }
}
```

Pression

```
void BMEpres(Stream* client, int8_t n) {
  bme.read(pres, temp, hum);
  if (PRESSURE_MIN <= pres && pres <= PRESSURE_MAX) {
    if (n == 1) {
      fichier.print("pression : ");
      fichier.print(pres);
      fichier.print(", ");
    } else {
      Serial.print(F("pression : "));
      Serial.print(pres);
      Serial.print(F(", "));
    }
  } else {
    if (n == 1) {
      fichier.print("pression : erreur, ");
    } else {
      Serial.print(F("pression : erreur, "));
    }
    if (ereurpres==0){
      leds.setColorRGB(0,255,0,0);
      delay(500);
    }else{
      leds.setColorRGB(0,0,255,0);
      delay(500);
    }
    ereurpres = !ereurpres;
  }
}
```

Comme nous l'avons précisé plus tôt, la structure est très proche du capteur de luminosité, sauf que nous utilisons le « bme.read » pour récupérer les valeurs retournées par les capteurs.

Horloge :

Variable globale et prototype relatif au capteur de luminosité :

```
RTC_DS1307 rtc;
String date();
```

Horloge dans le setup() :

```
if (!rtc.begin()) {
  Serial.println(F("Could not find RTC!"));
  leds.setColorRGB(0, 0, 0, 255);
  delay(500);
  leds.setColorRGB(0, 255, 0, 0);
  delay(500);
  while (1);  // Boucle infinie si le RTC n'est pas trouvé
}
```

Permet de vérifier la présence de l'horloge sinon une erreur est affichée sur le moniteur.

Fonction :

```
String date(){
  DateTime now = rtc.now();
  return String(now.day()) + "/" + now.month() + "/"+now.year()+" "+ now.hour()+":"+now.minute();
}
```

La fonction nous renvoie une chaîne de caractères contentent la date, le mois, l'année, l'heure et les minutes sous cette forme :

```
4/11/2023 11:45
```

GPS :

Variable globale et prototype relatif au GPS :

```
void gps(int8_t n);
SoftwareSerial gpsSerial(6, 7);
```

Initialisation de l'horloge dans le setup :

```
gpsSerial.begin(9600);
// Vérifie si l'initialisation du module GPS a échoué
if (!gpsSerial) {
  Serial.println("Erreur lors de l'initialisation du module GPS.");
  while (1)
  {
    leds.setColorRGB(0, 255, 0, 0);
    delay(500);
    leds.setColorRGB(0, 255, 255, 0);
    delay(500);
  }
}

Serial.println("Module GPS initialisé avec succès !");
```

Fonction retournant la longitude et la latitude :

```
void gps(int8_t n){
  if (gpsSerial.available()) {
    String data = gpsSerial.readStringUntil('\n');
    if (n == 1){
      fichier.print(data);
    }else{
      Serial.print(data);
    }
  }
}
```

Écriture sur la carte SD.

Mode Standard :

```
leds.setColorRGB(0, 0, 255, 0);
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    printSD();
  }
```

Le mode standard attend le temps de pause entre chaque mesure avant de faire appel à la fonction « printSD() » qui permet d'enregistrer les données des capteurs sur la carte SD.

Mode Maintenance :

```
leds.setColorRGB(0, 255, 128, 0);
Serial.print(date());
if (LUMIN == 1){
  CLumiere(0);
}
if (TEMP_AIR == 1){
  BMETemp(&Serial, 0);
}
if (PRESSURE == 1){
  BMEpres(&Serial, 0);
}
if (HYGR == 1){
  BMEhum(&Serial, 0);
}
fichier.println();

Serial.println();
```

Affiche les valeurs des capteurs sur le moniteur série s'ils sont demandés.

Mode Configuration :

227-405

Ce mode nous permet de changer les paramètres par défaut des capteurs et si on veut les capteurs

```
leds.setColorRGB(0, 255, 255, 0);
Serial.println(F("comande disponible"));
Serial.println(F("1 : LOG_INTERVALL"));
Serial.println(F("2 : FILE_MAX_SIZE"));
Serial.println(F("3 : RESET"));
Serial.println(F("4 : LUMIN"));
Serial.println(F("5 : LUMIN_LOW"));
Serial.println(F("6 : LUMIN_HIGH"));
Serial.println(F("7 : TEMP_AIR"));
Serial.println(F("8 : MIN_TEMP_AIR"));
Serial.println(F("9 : MAX_TEMP_AIR"));
Serial.println(F("10 : HYGR"));
Serial.println(F("11 : HYGR_MINT"));
Serial.println(F("12 : HYGR_MAXT"));
Serial.println(F("13 : PRESSURE"));
Serial.println(F("14 : PRESSURE_MIN"));
Serial.println(F("15 : PRESSURE_MAX"));
Serial.println(F("16 : recap"));
Serial.println(F("17 : exit"));
Serial.println(F("entre le numéro de la commande souhaitée"));
while (Serial.available() == 0) {
}
Ncomande = Serial.parseInt();
```

Les différents « Serial.print » permettent d'afficher les commandes auxquelles nous avons accès dans ce mode. Dans un objectif d'optimisation, nous avons choisi de numéroter les commandes pour que cela prenne moins de place dans la mémoire. Comme ça nous ne devrons pas tester des chaines de caractère entre elle, mais juste des entiers, c'est plus rapide et cela évite beaucoup d'erreurs de frappe.

« Serial.parseInt » permet de récupérer un entier tapé au clavier.

```
if (Ncomande == 2){
  Serial.print(F("FILE_MAX_SIZE = "));
  while (Serial.available() == 0) {
  }
  FILE_MAX_SIZE = Serial.parseInt();
  Serial.println();
}

if (Ncomande == 3){
  Serial.println(F("RESET"));
  LUMIN_LOW = 200;
  LUMIN_HIGH = 700;
  MIN_TEMP_AIR = -5;
  MAX_TEMP_AIR = 30;
  HYGR_MINT = 0;
  HYGR_MAXT = 50;
  PRESSURE_MIN = 450;
  PRESSURE_MAX = 1030;
  LOG_INTERVALL = 10;
  FILE_MAX_SIZE = 2048;
  LUMIN = 1;
  TEMP_AIR = 1;
  HYGR = 1;
  PRESSURE = 1;
}
```

Ensuite la variable **Ncomande** est testée dans une série de **if,** pour connaître la commande que l'opérateur a tapé. Puis, on redemande une saisie au clavier, pour modifier une valeur ou réinitialiser les variables à leurs valeurs. Nous vous invitons à parcourir le mode d'emploi pour connaître les différentes utilisations des commandes.

Mode Économique :

```
leds.setColorRGB(0, 0, 0, 255);
  if (currentMillis - previousMillis >= 2 * interval) {
    previousMillis = currentMillis;
    printSD();
  }
```

Possède le même fonctionnement que le mode standard mais le temps entre 2 mesures est 2 fois plus long.

Interruption et bouton :

Déclaration des modes :

```
uint8_t mode = 0;
uint8_t previous_mode = 0;
```

Setup() :

```
pinMode(redBtn, INPUT_PULLUP);
pinMode(greenBtn, INPUT_PULLUP);
```

Configure la broche « redBtn » et « greenBtn » comme entrée avec résistance de tirage activée

```
attachInterrupt(digitalPinToInterrupt(redBtn), buttonRedInterrupt, FALLING);
attachInterrupt(digitalPinToInterrupt(greenBtn), buttonGreenInterrupt, FALLING);
```

Attache une interruption à la broche « redBtn ».

```
delay(7000);
  if(digitalRead(redBtn) == LOW){
    mode = 2;
    Serial.println("Mode configuration");
  }
  else{
    mode = 0;
    Serial.println("Mode standard");
}
```

Permet d'accéder au mode configuration.

Dans le loop() :

```
if (redBtnState) {
  unsigned int currentTimeRed = millis();
  if (currentTimeRed - redBtnTime >= 5000) {
    if (mode == 0 || mode == 3){
      mode = 1;
      if (redBtnState) {
        redBtnState = false;
      }
    }
    if(mode == 1 && previous_mode == 1){
      mode = 0;
      if (redBtnState) {
        redBtnState = false;
      }
    }
  }
}
else if (greenBtnState) {
  unsigned int currentTimeGreen = millis();
  if (currentTimeGreen - greenBtnTime >= 5000) {
    if (mode == 0 || mode == 1){
      mode = 3;
      if (greenBtnState) {
        greenBtnState = false;
      }
    }
    if(mode == 3 && previous_mode == 3){
      mode = 0;
      if (greenBtnState) {
        greenBtnState = false;
      }
    }
  }
}
```

Cela vérifie grâce à l'interruption, le temps où on est resté appuyer sur le bouton. Selon le bouton pressé, on passe mode précédent.

Ensuite les modes sont utilisés dans un switch case où :

- Mode = 0 représente le mode standard
- Mode = 1 représente le mode maintenance

- Mode = 2 représente le mode configuration
- Mode = 3 représente le mode économique

Fonction d'interruption :

```
void buttonRedInterrupt() {
  if (!redBtnState) {
    redBtnState= true;
    redBtnTime= millis();
    if (greenBtnState) {
      greenBtnState= false;
    }
  }
}

void buttonGreenInterrupt() {
  if (!greenBtnState) {
    greenBtnState= true;
    greenBtnTime= millis();
    if (redBtnState) {
      redBtnState= false;
    }
  }
}
```

Scénario de démonstration permettant de prouver l'efficacité de la réalisation :

Téléverser le programme fourni sur la carte Arduino

Lors du démarrage, maintenir appuyer le bouton rouge pour rentrer dans le mode configuration, puis appuyer sur le « 1 »de votre clavier pour accéder à la fonction « LOG_INTERVALL ». Ensuite taper « 1 » rapidement pour définir le temps entre deux mesures à une minute

Pour continuer, taper le nombre « 17 » qui correspond à la fonction **exit** permettant de sortir du mode configuration.

Enfin, naviguer entre les modes configuration et standard et si vous allez dans le mode standard vous aurez un message à chaque fois que les valeurs sont écrites sur la carte SD.

Optimisation :

Voici quelques points d'optimisations :

- Utilisation de datatype adapté aux variables pour minimiser la consommation d'espace mémoire.
- Suppression de certains appels de fonctions comme Serial.println().
- Minimisation du texte informatif dans les Serial.println().
- Déplacement de certaines variables globales de la mémoire RAM à Flash grâce au **PROGMEM**.
- Déplacement de certains appels de fonction de la mémoire RAM à Flash grâce au **F()**.
- Eviter les répétitions dans le code avec les fonctions.

La mémoire Flash étant bien plus grande que la RAM et est fixe permettant de stocker les variables globales.

Conclusion :

Nous avons vu dans ce livrable le fonctionnement de chaque composants, modes, fonctionnalités de notre station météo. Le programme présenté ici est bien fonctionnel suite au téléversement de celui-ci dans la carte Arduino.