PROJET
SYSTEME EMBARQUE

Livrable 3

Fevre Thomas
Pavageau Victor

Parisien Barnabé
Sanahuges Baptiste

SOMMAIRE

Table des matières

Table des illustrations

I - Analyse fonctionnelle

A) Besoins

L'Agence Internationale pour la Vigilance Météorologique (AIVM) se lance dans un projet ambitieux : déployer dans les océans des navires de surveillance équipés de stations météo embarquées chargés de mesurer les paramètres influant sur la formation de cyclones ou autres catastrophes naturelles.

L'agence AIVM a besoin d'un prototype d'une station météo facile d'utilisation qui puisse mesurer différents paramètres météorologiques pour pouvoir les traiter par la suite.

Il y a quatre modes d'utilisation : standard, configuration, maintenance et économique.

B) Diagrammes

a) Diagramme des cas d'utilisation

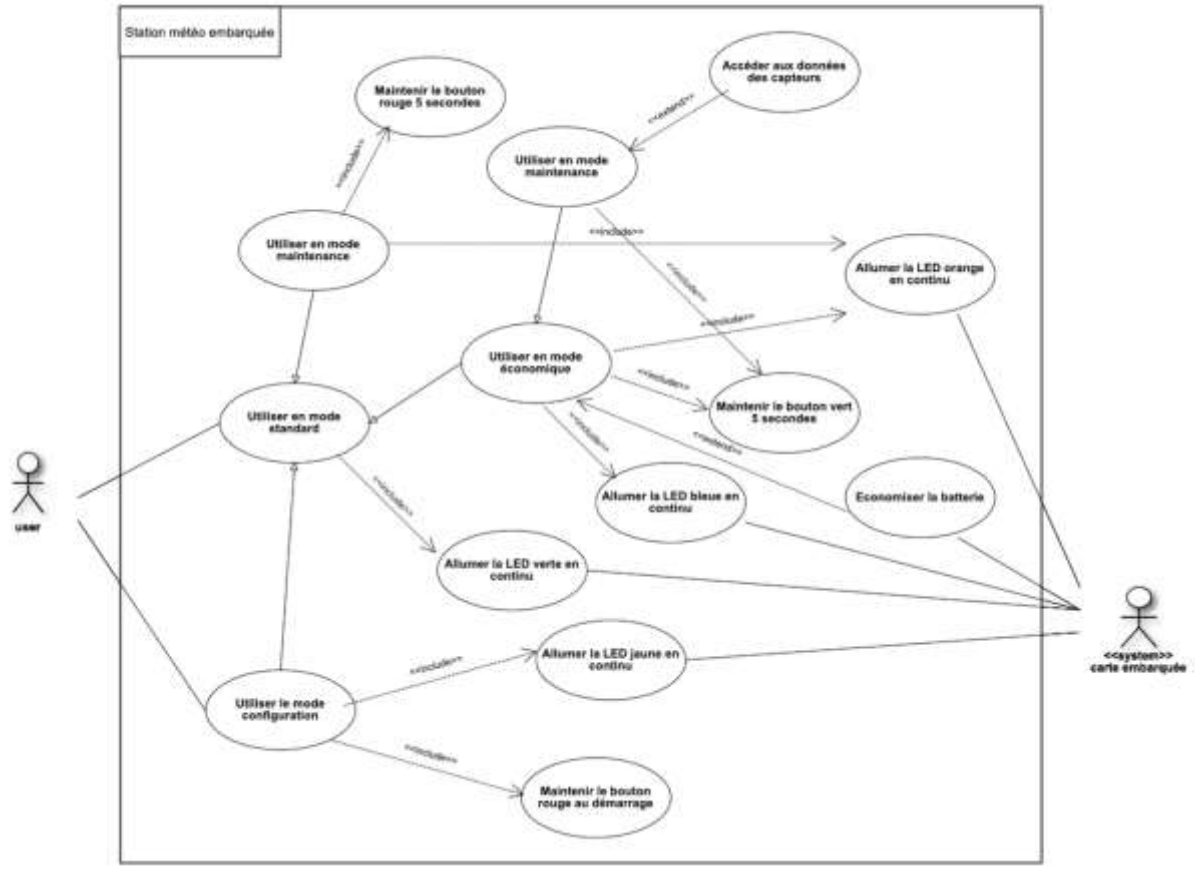


Figure 1 : Diagramme des cas d'utilisation

Dans ce diagramme on peut voir que l'utilisateur a accès à quatre modes d'utilisation :

- Standard
- Configuration
- Économique
- Maintenance

b) Diagramme de séquence

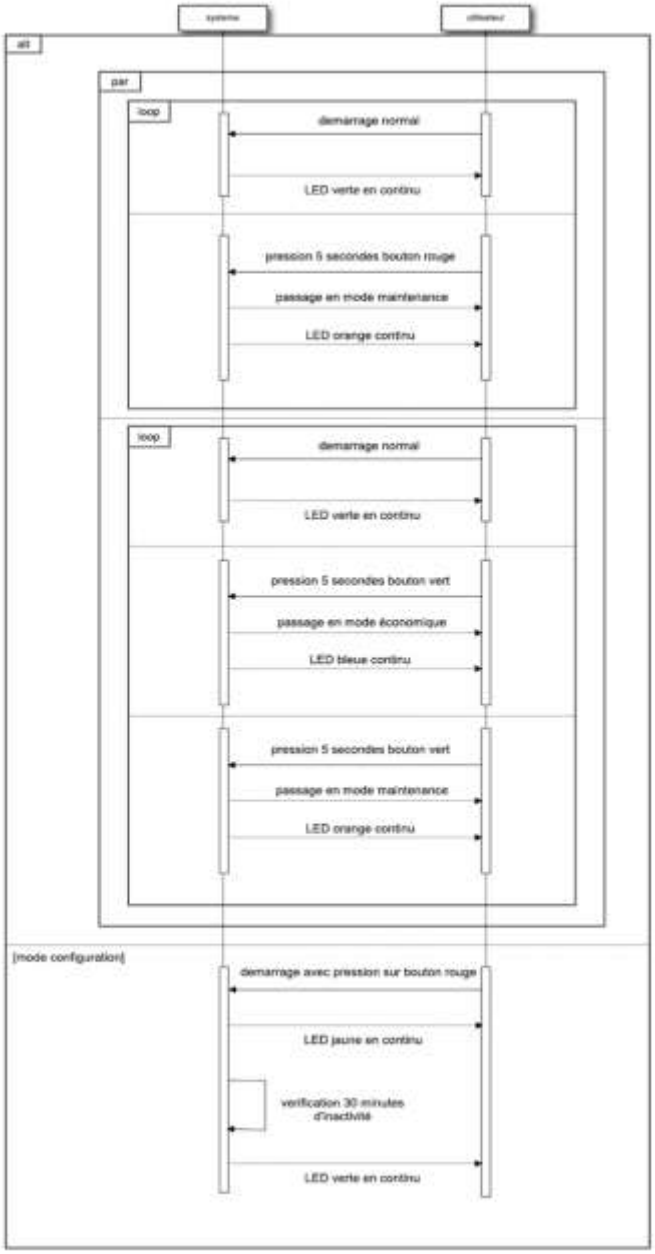


Figure 2 : Diagramme de séquence

Ce diagramme différencie deux types de démarrage :

Le premier, le démarrage normal, permet d'accéder aux autres modes de fonctionnements (maintenance et économique) il s'agit du mode standard. Lorsque le système est dans ce mode la LED vert est allumée en continu. En mode maintenance c'est la LED orange et bleue pour le mode économique. On accède à ces deux modes en maintenant cinq secondes soit le bouton vert soit le bouton rouge pour, respectivement, les modes économique et maintenance.

Le second, démarrage en appuyant sur le bouton rouge, le système est en mode configuration à ce moment la LED jaune est allumée. Après 30 secondes d'inactivité le système rebascule en mode standard.

Le système dispose de 4 modes de fonctionnement préprogrammés accessibles grâce à une interaction avec les boutons poussoirs :

- *Mode "standard" : Le système est démarré normalement (sans bouton pressé) pour faire l'acquisition des données.*
- *Mode "configuration" : Le système est démarré avec le bouton rouge pressé. Il permet de configurer les paramètres du système, l'acquisition des capteurs est désactivée et le système bascule n mode standard au bout de 30 minutes sans activité.*
- *Mode "maintenance" : Accessible depuis le mode standard ou économique, il permet d'avoir accès aux données des capteurs directement depuis une interface série et permet de changer en toute sécurité la carte SD sans risque de corrompre les données. On y accède en appuyant pendant 5 secondes sur le bouton rouge. En appuyant sur le bouton rouge pendant 5 secondes, le système rebascule dans le mode précédent.*
- *Mode "économique" : Accessible uniquement depuis le mode standard, il permet d'économiser de la batterie en désactivant certains capteurs et traitements. On y accède en appuyant pendant 5 secondes sur le bouton vert. En appuyant 5 secondes sur le bouton rouge, le système rebascule en mode standard.*

(cf. A2_-_Projet_Systeme_embarque_-_Modes_de_fonctionnement)

c) Diagramme d'activité

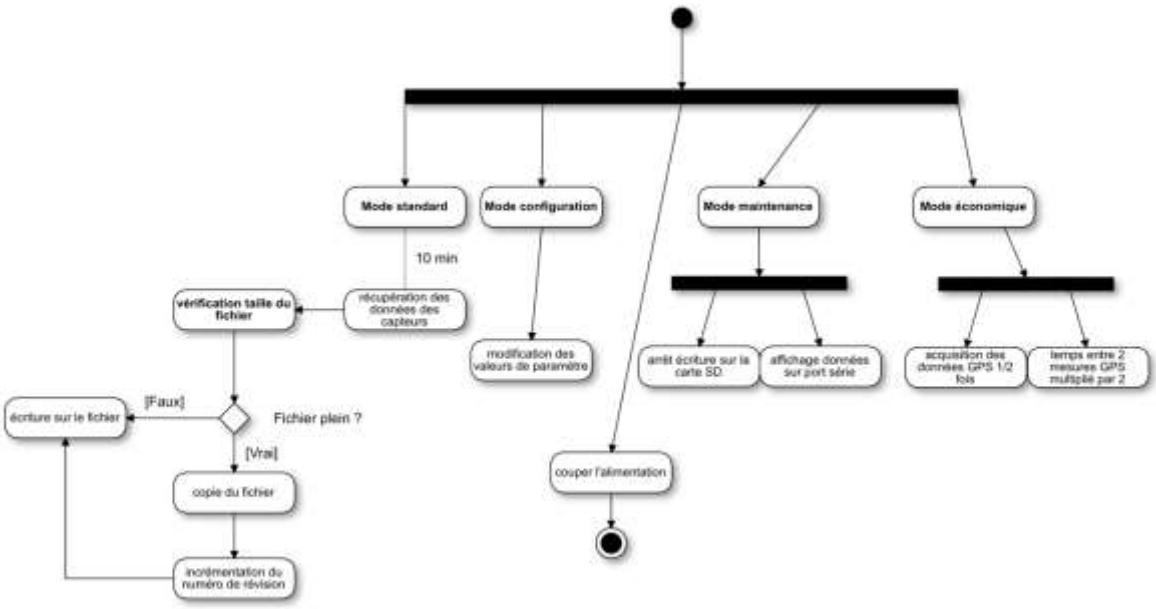


Figure 3 : Diagramme d'activité

Ce diagramme permet de représenter le déclenchement des évènements en fonction des états de notre système et de modéliser les différents comportements qui en découlera. Il ressemble fortement à un algorithme. Il nous permet d'avoir une vue d'ensemble de notre station météo et ainsi la réaliser au mieux en prenant en compte tous les enchainements d'états et les différentes possibilités en fonction de l'états actuel.

d) Diagramme de blocks

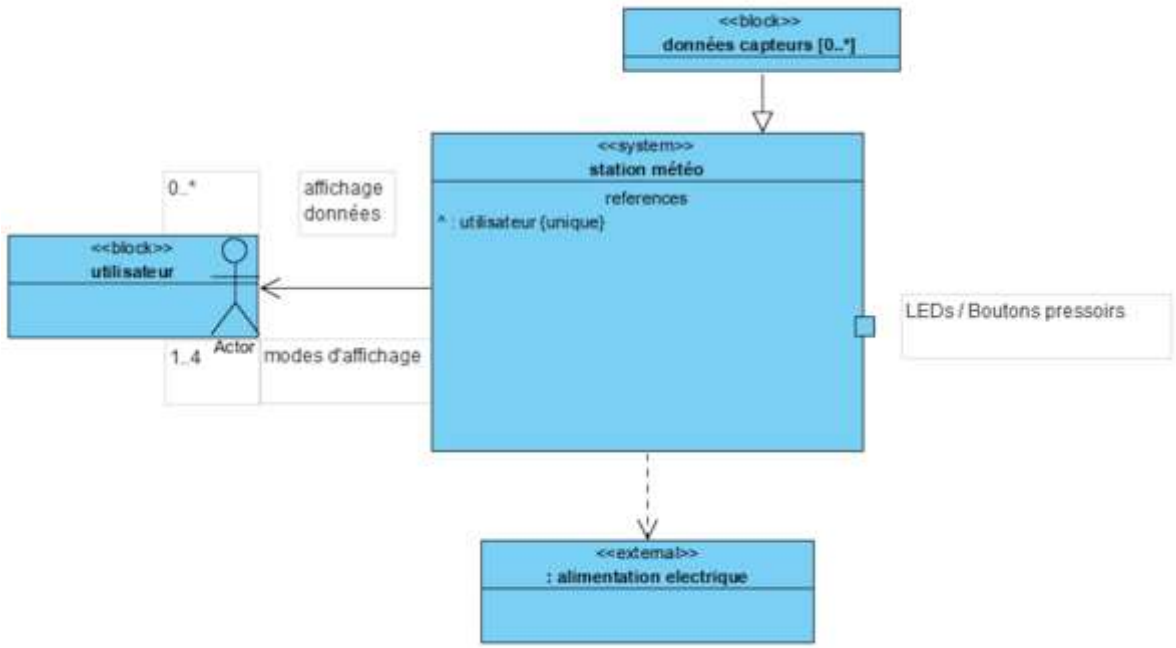


Figure 4 : Diagramme de blocks

Ce diagramme de blocks permet de voir les différents périphériques nécessaires afin que notre système, ici la station météo fonction (capteur, alimentation). Il nous montre également le flux des informations, les capteurs captent les données qui vont dans la station météo que l'utilisateur peut consulter à tout moment.

Ainsi, ces diagrammes nous ont permis d'identifier clairement ce que L'Agence Internationale pour la Vigilance Météorologique (AIVM) attendait de notre produit. A l'aide de ces diagrammes nous pouvons clairement identifier les différents modes d'utilisation du prototype et donc les différentes tâches qu'il pourra accomplir, nous avons fait en sorte que cela corresponde au mieux avec les exigences de l'AIVM.

Ensuite nous nous sommes penchés sur le développement de la station météo avec toutes les exigences identifier dans les différents diagrammes.

II – Réalisation

Dans cette partie nous retrouverons toute la documentation liée au code de notre station météo, c'est-à-dire les différentes variables, fonctions, bibliothèque ainsi que la présentation des différents mode (Configuration, Standard, Economique, Maintenance).

A) Dictionnaire de données

B) Bibliothèque

Afin de réaliser notre projet nous avons importé différentes bibliothèques qui sont les suivantes :

- #Include <SPI.h> : Cette bibliothèque permet de communiquer rapidement avec un ou plusieurs périphériques sur de courtes distances.
- #Include <avr/interrupt.h> : Celle-ci nous permet d'utiliser les interruptions, on met *avr/* afin de l'installer car elle ne l'est pas dans le logiciel Arduino.
- #Include <avr/sleep.h> : permet de mettre une mise en veille de la carte Arduino.
- #Include <avr/power.h> : permet de désactiver ou réactiver les différents ports de la carte Arduino.
- #Include <MsTimer2.h> : sert à générer une interruption à intervalle régulier.
- #Include <SD.h> : sert à utiliser une carte mémoire SD.
- #Include <stdlib.h> : bibliothèque pour les fonctionnalités de base.
- #Include <stdio.h> : bibliothèque pour les fonctionnalités de base.
- #include <SdFat.h> : permet de format de stockage FAT dans la carte SD.
- #include <Wire.h> : permet d'envoyer et de recevoir des données sur un réseau de capteurs.
- #include ''DS1307.h'' : permet d'utiliser l'horloge en temps réelle.
- #include <SoftwareSerial.h> : permet une communication série logicielle sur n'importe quelles broches de la carte Arduino.
- #include ''SparkFunBME280.h'' : permet de lire les capteurs de température, pression et humidité.
- #include <ChainableLED.h> : permet de contrôler la LED RVB.
- #include <Sting.h> : elle contient les définitions des macros, des constantes et des déclarations de fonctions ainsi que pour la manipulation de la mémoire.
- #include <avr/pgmspace.h> : permet l'accès à la mémoire flash de la carte Arduino.

C) Les changements de modes

Nous avons décidé d'utiliser les interruptions afin de passer d'un mode à un autre. Nous ne pouvions pas faire autrement car nous ne savons pas quand l'utilisateur appuiera sur le bouton et les interruptions est la solution idéale car elle s'exécute lorsque qu'il y a un évènement pas prévu par le programme comme l'appuie d'un bouton dans notre cas.

```
void doRedButton() {
  if (digitalRead(RBUTTON) == LOW) {
    rStart = millis();
    redTimer = 0;
  } else {
    if (rStart != 0) {
      redTimer = millis() - rStart;
    }
  }
}
```

Ci-dessus, la routine « doRedButton » qu'on utilisera dans l'interruption du bouton rouge. Elle nous permet de calculer le temps d'appui sur un bouton. La variable « rStart » sera initialisée au moment où l'on déclenche l'interruption par le temps à ce moment-là. Puis on déduit « rStart » à « redTimer » qui lui sera le moment auquel on lâche le bouton. Ainsi nous pourrons déterminer si le bouton est pressé plus ou moins de 5 secondes. Le même est refait pour l'interruption lorsqu'on appuie sur le bouton vert.

```
if (redTimer >= DEFAULT_LONGPRESS_LEN && economyMode == 0) {
  restartTimer();
  maintenanceMode = !maintenanceMode;
  if (maintenanceMode) {
    detachInterrupt(digitalPinToInterrupt(GBUTTON));
  } else {
    attachInterrupt(digitalPinToInterrupt(GBUTTON), doGreenButton, CHANGE);
  }
} else if (greenTimer >= DEFAULT_LONGPRESS_LEN) {
  restartTimer();
  economyMode = !economyMode;
  maintenanceMode = 0;
  if (economyMode) {
    detachInterrupt(digitalPinToInterrupt(GBUTTON));
  } else {
    attachInterrupt(digitalPinToInterrupt(GBUTTON), doGreenButton, CHANGE);
  }
} else if (redTimer >= DEFAULT_LONGPRESS_LEN && maintenanceMode == 1 || redTimer >= DEFAULT_LONGPRESS_LEN && economyMode == 1 ) {
  restartTimer();
  maintenanceMode = 0;
  economyMode = 0;
}
```

Ce bout de code nous permet de déterminer si la pression est plus longue que 5 secondes et faire les étapes nécessaires si c'est le cas. Dans un premier temps nous comparons « redTimer » et « DEFEULT_LONGPRESS_LEN » (c'est une macro qui vaut 5000ms) et nous regardons si nous ne sommes pas en mode économique. Si c'est la cas, « redTimer » est supérieur à 5 secondes et que nous ne sommes pas en mode économique, alors on passe en mode maintenance, on remet tous les timers à zéro et on désactive l'interruption sur le bouton vert car on ne s'en sert pas dans ce mode.
Si le « redTimer » ne fait pas 5 secondes alors on vérifie si le « greenTimer » est supérieur à 5 secondes alors on remet les timers à zéro, on passe en mode économique et on désactive l'interruption sur le bouton vert car on ne s'en sert pas dans ce mode. Sinon, si le « redTimer » est supérieur à 5 secondes et qu'on est soit en mode maintenance soit en mode économique alors on remet les timers à zéro et on sort de ces deux modes.

D) Les modes

a- Le mode Configuration

Ce mode permet de configurer le système grâce à une interaction depuis une console sur l'interface série du microcontrôleur. On pourra depuis l'interface série taper des commandes de configuration pour modifier les valeurs des paramètres enregistrés dans l'EEPROM comme la définition de l'intervalle entre deux mesure, définition de la taille maximale d'un fichier qui si celle-ci est atteinte provoquera l'archivage de celui-ci. Il y a aussi la possibilité de réinitialisation de l'ensemble des paramètres à leurs valeurs par défaut, d'afficher la version du programme et un numéro de lot. De plus au bout de de 30 secondes on arrête l'acquisition des données d'un capteur.

```
case Lumin_LOW :
  {
    int p2;
    p2 = verifINT(&part2);
    if (0 < p2 && p2 < 1023) {
      LUMIN_LOW = p2;
      Serial.println("changement effectue !");
    }
    else Serial.println("erreur de valeur de parametre");
  }
  break;
```

On a par exemple le case "lumin_LOW" ci-dessus qui appelle une fonction permettant de vérifier que le paramètre entré est un entier :

```
int verifINT(String *partie) {
  int chiffre = 30000;
  if (int nombre = (*partie).toInt()) {
    return nombre;
  }
  else if (*partie == "0") {
    return 0;
  }
  else {
    return chiffre;
  }
}
```

Ensuite la boucle if permet la vérification de l'encadrement de la valeur du paramètre.
Voici un autre cas ci-dessous, pour réinitialiser les paramètres à leur valeur initiale.

```
case Reset:
  {
    RESET();
    Serial.println("reinitialisation de l'ensemble des parametres à leurs valeurs par defaut.");
  }
  break;
```

Et la fonction qui est appelée contenant les paramètres par défaut :

```
void RESET() {
  LUMIN = 1;
  LUMIN_LOW = 255;
  LUMIN_HIGH = 768;
  TEMP_AIR = 1;
  MIN_TEMP_AIR = -10;
  MAX_TEMP_AIR = 60;
  HYGR = 1;
  HYGR_MINT = 0;
  HYGR_MAXT = 50;
  PRESSURE = 1;
  PRESSURE_MIN = 850;
  PRESSURE_MAX = 1080;
  LOG_INTERVAL = 10; // entre 2 mesures 10min
  FILE_MAX_SIZE = 4096; //4 Ko provoque son archivage
  TIMEOUT = 30; //secondes
}
```

b- Le mode Standard

Ce mode permet la récupération à intervalle régulier la valeur des capteurs, l'ensemble des mesures est enregistré sur une seule ligne horodatée. La donnée associée à un capteur ne répondant pas au bout d'un temps défini dans le mode configuration sera « NA ». Ces données sont enregistrées sur la carte SD dans un fichier dont la taille est définie dans le mode configuration. Les noms des fichiers prendront la forme suivante : 200531_0.LOG (Année=20, mois=05, jour=31, numéro de révision=0). Le système écrit toujours dans le fichier dont le numéro de révision est 0. Quand un fichier est plein, le système crée une copie du fichier avec le numéro de révision adapté puis recommence à enregistrer les données au début du fichier.

```
#define WRITE_IN_SD char fileName[15] = "";\
  sprintf(fileName, "%d%d%d_%d.LOG", clock.year, clock.month, clock.dayOfMonth, fileCounter);\
  Serial.begin(9600);\
  myFile = SD.open(fileName, FILE_WRITE);\
  if (myFile) {\
    Serial.print("Writing...");\
    if (LUMIN == 1){\
      myFile.print(analogRead(LUM_SENSOR));\
      myFile.print(" LUX|");\
    }\
    if (TEMP_AIR == 1) { \
      myFile.print(random(-20,50));\
      myFile.print(" °C|"); \
    }\
    if (PRESSURE == 1) {  \
      myFile.print(random(1,2));\
      myFile.print(" HPa|");\
    }\
    if (HYGR == 1) {  \
      myFile.print(THPsensor.readFloatHumidity());\
      myFile.print(" %|");\
    }\
    myFile.print(random(-40,126));\
    myFile.print(" °C|");\
    myFile.print(random(3,61)/10);\
    myFile.print(" L/min|");\
    myFile.print(random(1,131));\
    myFile.println(" km/h|");\
    myFile.close();\
    Serial.println("done.");\
  } else {\
    Serial.println("error opening");\
  }
```

On doit stocker les données dans la carte SD, pour cela on utilise le code ci-dessus. Dans un premier temps on donne le nom du fichier, ici, l'année, le mois, le jour et le numéro de fichier .LOG. Ensuite on ouvre ce fichier pour écrire toutes les données dessus.

```
} else if (economyMode) {
  displayColor(0, 0, 255);
  WRITE_IN_SD
} else {
  displayColor(0, 255, 0);
  WRITE_IN_SD
}
```

Le bout de code qu'on a vu précédemment est une macro, on l'utilise lorsqu'on est en mode économique ou lorsque l'on n'est pas en mode maintenance.

c- Le mode Maintenance

Les données ne sont plus écrites sur la carte SD mais peuvent être consultées en direct depuis le port série. La carte SD peut être retirée et replacée en toute sécurité.

```
if (maintenanceMode) {
  displayColor(237, 109, 0);
  if (LUMIN == 1) {
    Serial.print(analogRead(LUM_SENSOR));
    Serial.print(" LUX|");
  }
  if (TEMP_AIR == 1) { //Temperature de l'air exterieur
    Serial.print(random(-20, 50));
    Serial.print(" °C|");
  }
  if (PRESSURE == 1) {  //pression atmosphérique
    Serial.print(random(1, 2));
    Serial.print(" HPa|");
  }
  if (HYGR == 1) {  //hygrometrie
    Serial.print(THPsensor.readFloatHumidity());
    Serial.print(" %|");
  }

  Serial.print(random(-40, 126)); //température de l'eau
  Serial.print(" °C|");
  Serial.print(random(3, 61) / 10); //hygrométrie
  Serial.print(" L/min|");
  Serial.print(random(1, 131));
  Serial.println(" km/h|");      //Vitesse du vent
```

Dans le mode maintenance on doit pouvoir enlever la carte SD sans avoir de perte de données. Donc lorsqu'on passe dans ce mode, les données ne s'écrivent plus sur la carte mais sur le moniteur en série.

d- Le mode Économique

L'acquisition des données du GPS n'est plus effectuée qu'une mesure sur deux et le temps entre 2 mesures est multiplié par 2 tant que le système est dans ce mode.

```
if (economyMode) {
  delay(logInterval * 360000);
} else {
  delay(logInterval * 180000);
}
```

En mode économique, nous devons faire qu'une mesure sur deux du coup on multiplie par deux le délai de la boucle.

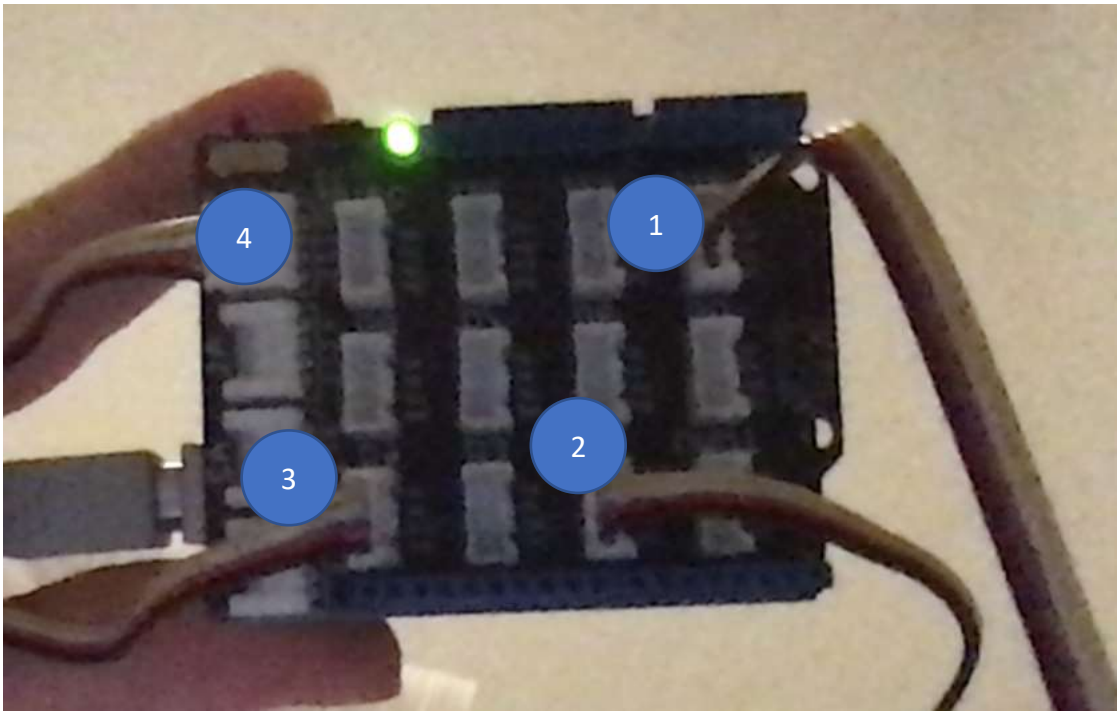
III – Nos estimations

Afin de calculer la complexité d'espace de notre prototype, nous avons estimé une mesure à 150 octets, ensuite nous avons imaginé 3 scénarios, rester en mode standard tout le temps, rester en mode éco (ou en mode maintenance) tout le temps, alterner entre les deux options. Nous avons alors calculé combien de temps il faudrait pour remplir la carte SD à 90% (au-delà de ce stade on la considèrera saturée) dans chacun de ses 3 scénarios.

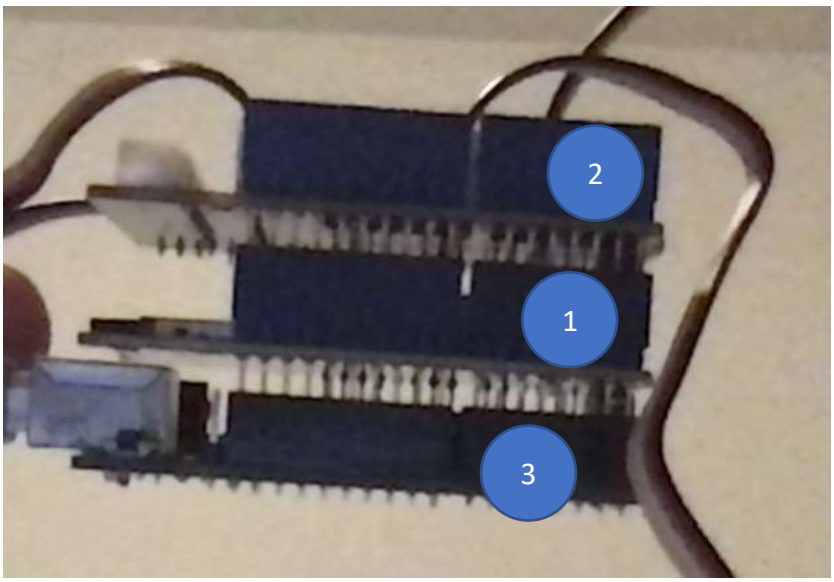
Mémoire	90% de la mémoire	Mesure
8 Go	7,2 Go	150 octets
8,00E+06 octets	7,20E+06 octets	

	Nombre de mesures que l'on peut effectuer sur la carte SD avant saturation	Nombres de jour avant saturation de la carte SD	Nombres d'année(s) avant saturation de la carte SD
100% Mode Éco ou Mode Maintenance	48000	666,67 jours	1,83 ans
50% Mode Éco ou Maintenance et 50% Mode Standard		500 jours	1,37 ans
100% Mode Standard		333,33 jours	0,92 an

IV – Montage



1- RTC
2- Boutons
3- LED rvb
4- Capteur de luminosité



1- SD CARD Shield
2- Base Shield
3- Carte Arduino

V - Conclusion

Ainsi, nous proposons un prototype de station météorologique embarquée correspondant à la demande faite par l'AIVM afin de déployer des navires de surveillance chargés de mesurer les paramètres influant sur la formation de cyclones ou autres catastrophes naturelles.

Le prototype est découpé en 4 modes de fonctionnements distincts qui s'identifient facilement grâce à une LED qui s'allume de couleur différentes, ensembles, ils permettent de régler les paramètres, effectuer des mesures météorologiques grâce à des capteurs et les analyser, écrire les données sur une carte SD ou via une interface série.

Cependant, le prototype reste sujet à des améliorations concernant quelques de ses fonctionnalités, mais également la réparation de certaines autres.

VI - Annexes

Bibliographie
Manuel d'utilisation + (VIDEO)
CDCT

BIBLIOGRAPHIE

https://eduscol.education.fr/sti/sites/eduscol.education.fr.sti/files/ressources/techniques/979/979-179-p100.pdf
https://github.com
https://www.techniques-ingenieur.fr
https://openclassrooms.com/fr/
https://fr.wikipedia.org/wiki/
https://www.arduino.cc
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.Librairies

LOGICIELS

https://cacoo.com/
https://www.tinkercad.com

Annexe 1 – 1er Livrable 2

Nom	Type	Paramètres	Valeur par d efaut	Description
MACROS				
BUTTON1_PIN	x	x	2	Pin du bouton rouge
BUTTON2_PIN	x	x	3	Pin du bouton vert
DEFAULT_LONGPRESS_LEN	x	x	100	Longueur (en temps) d'un appui long sur un bouton
DELAY	x	x	20	Délai d'une « loop »
Lverte	x	x	5	Pin du vert sur le LED rvb
Lrouge	x	x	6	Pin du rouge sur le LED rvb
Lbleue	x	x	7	Pin du bleu sur le LED rvb
VARIABLES GLOBALES				
LOG_INTERVALL	x	x	10	définition de l'intervalle entre 2 mesures
FILE_MAX_SIZE	x	x	4096	définition de la taille maximale (en octets) d'un fichier de log
RESET	x	x	x	réinitialisation de l'ensemble des paramètres à leurs valeurs par défaut
VERSION	x	x	x	affiche la version du programme et un numéro de lot
TIMEOUT	x	x	30	durée (en s) au bout de laquelle l'acquisition des données d'un capteur est abandonnée. Après 2 mesures en timeout, le capteur est signalé en erreur
LUMIN	Entier	{0,1}	1	définition de l'activation (1) / désactivation (0) du capteur de luminosité
LUMIN_LOW	Entier	{0-1023}	255	définition de la valeur en dessous de laquelle la luminosité est considérée comme « faible »

LUMIN_HIGH	Entier	{0-1023}	768	définition de la valeur au-dessus de laquelle la luminosité est considérée comme “forte”
TEMP_AIR	Entier	{0,1}	1	Définition de l’activation (1) / désactivation (0) du capteur de température de l'air
MIN_TEMP_AIR	Réel	{-40-85}	-10	Définition du seuil de température de l'air (en °C) en dessous duquel le capteur se mettra en erreur.
MAX_TEMP_AIR	Réel	{-40-85}	60	Définition du seuil de température de l'air (en °C) au-dessus duquel le capteur se mettra en erreur.
HYGR	Réel	{0,1}	1	Définition de l’activation (1) / désactivation (0) du capteur d’hygrométrie
HYGR_MINT	Réel	{-40-85}	0	définition de la température en dessous de laquelle les mesures d’hygrométrie ne seront pas prises en compte.
HYGR_MAXT	Réel	{-40-85}	50	définition de la température au-dessus de laquelle les mesures d’hygrométrie ne seront pas prises en compte.
PRESSURE	Entier	{0,1}	1	définition de l’activation (1) / désactivation (0) du capteur de pression atmosphérique.
PRESSURE_MIN	Entier	{300-1100}	850	définition du seuil de pression atmosphérique (en HPa) en dessous duquel le capteur se mettra en erreur.
PRESSURE_MAX	Entier	{300-1100}	1080	définition du seuil de pression atmosphérique

				(en HPa) au- dessus duquel le capteur se mettra en erreur.
CLOCK	Entier	HEURE {0-23} : MINUTE {0-59} : SECONDE {0-59}	x	Configuration de l'heure du jour
DATE	Entier	MOIS {1-12}, JOUR {1-31}, ANNEE {2000-2099}	x	Configuration de la date du jour
DAY	Chaine de caractères	{MON, TUE, WED, THU, FRI, SAT, SUN}	x	Configuration du jour de la semaine
PATH	Chaine de caractères	x	x	Chemin d'accès aux fichiers
CLASS ButtonHandler				
Init()	Void	x	x	Fonction d'initialisation de la classe
Handle()	Entier	x	x	Focntion de gestion des bouton (temps de pression)
Was_pressed	Boolean	x	x	Etat du bouton avant la pression
Presser_counter	Entier	x	x	Compteur de temps de pression
Pin	Entier constant	x	x	Pin du bouton à utiliser
Longpress_len	Entier constant	x	x	Longueur (en temps) d'une pression longue
FONCTIONS				
Mode_eco()	void	x	x	Fonction du passage en mode économique
Mode_standard()				Fonction du passage en mode standard
Mode_config()				Fonction du passage en mode configuration
verifType()	void	(int)	x	Fonction qui vérifie le type d'une variable entrée par l'utilisateur
verifEncadre()	void	(int,int,int)	x	Fonction qui vérifie l'encadrement d'une va

				riable entrée par l'utilisateur
Sepration()	void	(char,char)	x	Fonction qui sépare une chaine de caractère en fonction d'un caractère
Mode_maint()	void	x	x	Fonction du passage en mode maintenance
Print_event()	void	(char*,int)	x	Fonction qui permet de passer d'un mode à un autre en fonction des boutons
displayColor()	void	(char,char,char)	x	Fonction qui donne la couleur à la LED rvb

	Pas de résultat
	Grands titres
	Public (class)
	Protected (class)

Annexe – Makefile

```
#!/bin/sh
echo "Quel est le nom de votre fichier de mise à jour ? [sans extension]"
read fic

echo "Compilation de votre programme..."
make


ok="oui"

echo Votre fichier est bien ${fic}.c [oui/non] ?

read rep

if [ ${rep} = ${ok} ]
then
echo "Installation de la mise à jour..."

    avrdude -p com1 -U eeprom:w:~/upload/${fic}.c

echo "Mise à jour installée"

else
    echo Relancez le script
fi
```

Ci-dessus, un script Shell qui nous permet de lancer le makefile (ci-dessous) et de téléverser le fichier dans la carte Arduino.

```
arduino:  main.o
        gcc -o arduino main.o
exemple.o: exemple.c
clean:
mrproper: clean
rm -f *.o core
```

Le makefile nous permet de compiler le programme à mettre dans la carte Arduino.