PROJET : **Worldwide Weather Watcher**

Livrable 2 : **Architecture du programme**



Groupe 1 :   
Matthieu Marconnet Régidor

Romain Hemart

Alban Godier

Hugo Helm

Adam Ben Amar

Rendu le 24/10/2023

Sommaire :

[Introduction : 5](#_Toc149070045)

[Réalisation du livrable 6](#_Toc149070046)

[Logigramme : Qu’est-ce que c’est ? 6](#_Toc149070047)

[Logigrammes 7](#_Toc149070048)

[Pseudocode 14](#_Toc149070049)

[Variables 21](#_Toc149070050)

[Conclusion : 23](#_Toc149070051)

[Bibliographie : 24](#_Toc149070052)

Table des figures :

Figures :

[Figure 1 - Symbole de processus 6](#_Toc149049982)

[Figure 2 - Symbole de début/fin 6](#_Toc149049983)

[Figure 3 - Symbole de décision 6](#_Toc149049984)

[Figure 4 - Symbole de processus prédéfini 6](#_Toc149049985)

Logigrammes :

[Logigramme 1 - Logigramme fonction setup 7](#_Toc149050091)

[Logigramme 2 - Logigramme fonction loop 8](#_Toc149050092)

[Logigramme 3 - Logigramme fonction VerifConfig 9](#_Toc149050093)

[Logigramme 4 - Logigramme fonction Maintenance 9](#_Toc149050094)

[Logigramme 5 - Logigramme fonction standard 10](#_Toc149050095)

[Logigramme 6 - Logigramme fonction eco 11](#_Toc149050096)

[Logigramme 7 - Logigramme fonction Config 11](#_Toc149050097)

[Logigramme 8 - Logigramme fonction InterBoutonR 12](#_Toc149050098)

[Logigramme 9 - Logigramme fonction InterBoutonV 13](#_Toc149050099)

Pseudocodes :

[Pseudocode 1 - Pseudocode de la fonction acquisition 15](#_Toc149070068)

[Pseudocode 2 - Pseudocode de la fonction createTimer 16](#_Toc149070069)

[Pseudocode 3 - Pseudocode de la fonction calculOCR 16](#_Toc149070070)

[Pseudocode 4 - Pseudocode de la fonction ISR 17](#_Toc149070071)

[Pseudocode 5 - Pseudocode de la fonction changerEnModeStandart 18](#_Toc149070072)

[Pseudocode 6 - Pseudocode de la fonction changerEnModeEconomie 18](#_Toc149070073)

[Pseudocode 7 - Pseudocode de la fonction changerEnModeConfiguration 18](#_Toc149070074)

[Pseudocode 8 - Pseudocode de la fonction changerEnModeMaintenance 19](#_Toc149070075)

[Pseudocode 9 - Pseudocode de la fonction changerMode 19](#_Toc149070076)

[Pseudocode 10 - Pseudocode de la fonction BoutonRougePressé 20](#_Toc149070077)

[Pseudocode 11 - Pseudocode de la fonction boutonVertPressé 21](#_Toc149070078)

Glossaire :

Interruption : Mécanisme matériel ou logiciel permettant de dévier temporairement l'exécution normale du programme pour gérer un événement externe ou une condition particulière. Les interruptions sont couramment utilisées dans les systèmes embarqués, y compris les microcontrôleurs Arduino, pour traiter des événements en temps réel sans bloquer l'exécution principale du programme.

Logigramme : Représentation graphique qui permet de visualiser le déroulement d'un processus ou d'un système. Il est largement utilisé en informatique, en ingénierie, en gestion de projet et dans de nombreux autres domaines pour décrire les étapes d'un processus, les relations entre ces étapes et la logique de prise de décision. Il est composé d'une série de symboles graphiques et de lignes de liaison, chacun ayant une signification spécifique. (Plus de détail au cours du livrable)

Timer : Un timer logiciel en Arduino est généralement implémenté en utilisant une variable (généralement de type *unsigned long*) pour stocker le temps écoulé depuis le début de l'exécution du programme. Le compteur est mis à jour périodiquement, par un composant dédié. Nous pouvons ensuite, par exemple, vérifier que le compteur atteint une valeur spécifiée (un seuil) correspondant à un intervalle de temps donné, pour ensuite déclencher une action spécifique.

Registres : Zone de mémoire défini dans le matériel d'un microcontrôleur ou d'un processeur. Les registres sont utilisés pour stocker des données, des paramètres de configuration et des informations de contrôle.

Prescaler : Variable permettant de définir le nombre de tick avant d’incrémenter le timer.

Introduction :

Rappel du contexte :

Le projet World Weather Watcher a été entrepris par l'Agence Internationale pour la Vigilance Météorologique (AIVM).

Il consiste à concevoir et déployer dans les océans des navires de surveillance équipés de stations météo embarquées chargées de mesurer les paramètres influant sur la formation de cyclones ou autres catastrophes naturelles.

Notre startup a été embauchée pour la création d’un prototype.

La conception de ce dispositif doit prendre en compte divers besoins et contraintes, comme la nécessité de rendre l’utilisation accessible à tous, et de résister à son environnement. Ce projet étant de grande envergure, notre solution se doit d’être irréprochable.

Nous avons réalisé un premier livrable qui consistait à présenter les besoins et spécificités de la station météo. Nous avons présenté sous forme de diagrammes, schémas et algorithmes les différentes fonctionnalités prévues.

Nous allons maintenant représenter de manière détaillée le fonctionnement logiciel de notre projet à l’aide de différents logigrammes.

Cette étape est importante pour la réussite de la suite de notre travail, car elle permet de définir précisément ce que l’on va devoir implémenter et développer lors de la prochaine étape.

Réalisation du livrable

## Logigramme : Qu’est-ce que c’est ?

Les logigrammes sont des graphiques permettant de définir le fonctionnement d’un programme, par exemple. Nous allons donc les utiliser dans ce rapport afin de représenter de manière visuelle, cohérente et intuitive notre futur programme.

Afin de créer ces logigrammes, de nombreux logiciels sont disponibles ; Nous avons choisis draw.io, qui nous laisse une grande liberté de création.

Ces logigrammes, afin d’être universellement compréhensibles, sont écrit de façon normalisée. Nous avons donc une liste d’éléments que nous pouvons utiliser pour créer notre graphique :

* Symbole de processus/d’action : Réalisation d’une action, d’un processus.

Symbole de processus d'un logigramme

Figure 1 - Symbole de processus

* Symbole de début/fin : Point de départ ou d’arrivée, ou déclaration de fonction.

Symbole de début/de fin d'un logigramme

Figure 2 - Symbole de début/fin

* Symbole de décision : Indique une question nécessitant une réponse de type vrai/faux.

Symbole de décision d'un logigramme

Figure 3 - Symbole de décision

* Symbole de processus prédéfini : Appelle une fonction.

symbole de processus prédéfini

Figure 4 - Symbole de processus prédéfini

## Logigrammes

Maintenant que nous avons défini comment nous allons créer les logigrammes, nous allons les présenter et décrire.

Nous en avons un pour chaque fonction de notre programme :

Une image contenant texte, diagramme, capture d’écran, reçu

Description générée automatiquement

Logigramme 1 - Logigramme fonction setup

La première fonction qui s’exécutera est la fonction **setup** qui se lance uniquement lors du démarrage du système ou d’un reset de la carte Arduino.

Dans cette fonction, nous allons dans un premier temps venir faire appel à des sous programmes qui vont initialiser les différents capteurs à notre disposition ainsi que les autres initialisations en fonctionnement logiciel.

Par la suite, nous allons venir regarder si le bouton est enfoncé lors de cette procédure grâce à la fonction **verif config.** Dans un dernier temps, nous initialiseront les interruptions du système.

Une image contenant diagramme, Dessin technique, croquis, Plan

Description générée automatiquement

Logigramme 2 - Logigramme fonction loop

La fonction suivante est la fonction **loop**, c’est une fonction qui va tourner en boucles et répéter les instructions contenues dans cette dernière.

Dans ce cas, la fonction à pour utilité de regarder le mode dans lequel est la carte est appliquer la fonction qui convient.

Nous avons ici choisi de faire correspondre les valeurs suivantes aux différents modes :

* 0 : mode configuration
* 1 : mode standard
* 2 : mode économique
* 3 : mode maintenance

Une image contenant diagramme, texte, ligne

Description générée automatiquement

Logigramme 3 - Logigramme fonction VerifConfig

Ici la fonction **VerifConfig** va vérifier si l’utilisateur active ou non le mode « configuration ».

Pour cela, on va d’abord mettre dans la variable boutonRouge 🡪 « digitalRead(pinRouge) » qui va permettre de lire l’entrée dans le bouton rouge.

Ensuite, on fait une boucle if qui permet de vérifier s’il y’a appui ou non sur le bouton rouge, si oui on passe en mode « configuration », sinon on reste en mode « standard ».

Une image contenant texte, diagramme, capture d’écran, cercle

Description générée automatiquement

Logigramme 4 - Logigramme fonction Maintenance

La fonction **maintenance** permet de passer en mode “maintenance” qui va permettre à l’utilisateur d’avoir accès aux données depuis le port série et de changer la carte SD sans perdre les données.

Dans cette fonction, on met la fonction “Acquisition des capteurs” avec le paramètre “MaintenanceDelay”.

Une image contenant texte, diagramme, cercle, capture d’écran

Description générée automatiquement

Logigramme 5 - Logigramme fonction standard

La fonction **standard** permet de passer en mode “standard” qui va permettre l’acquisition des données.

Dans cette fonction, on met la fonction “Acquisition des capteurs” avec le paramètre “LogIntervalle” qui va permettre de créer un intervalle de temps entre 2 données.

Une image contenant diagramme, cercle, texte, croquis

Description générée automatiquement

Logigramme 6 - Logigramme fonction eco

La fonction **eco** permet de passer en mode “économique” qui va permet d’économiser de la batterie en désactivant certains capteurs.

Dans cette fonction, on met la fonction “Acquisition des capteurs” avec le paramètre “LogIntervalle” qui va être multiplier par 2 pour que l’intervalle de temps entre 2 données soit deux fois plus longue.

Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement

Logigramme 7 - Logigramme fonction Config

La fonction **Config**, est la fonction qui est appelé lorsque le système rentre dans le mode configuration.

Dans cette dernière, le système vient vérifier si le temps depuis la dernière activité est supérieur à 30 minutes et nous pourrons aussi effectuer des commandes dans ce dernier.

Une image contenant diagramme, texte, croquis, dessin

Description générée automatiquement

Logigramme 8 - Logigramme fonction InterBoutonR

La fonction **InterBoutonR** va permettre de vérifier si le bouton rouge est activé ou non.

On retrouve dans la fonction une boucle if qui va vérifier si le bouton est activé avec « DigitalRead », si oui on active le timer sinon on le désactive.

Une image contenant texte, capture d’écran, Police, diagramme

Description générée automatiquement

Logigramme 9 - Logigramme fonction InterBoutonV

La fonction **InterBoutonV** va permettre de vérifier si le bouton vert est activé ou non.

On retrouve dans la fonction une boucle if qui va vérifier si le bouton est activé avec « DigitalRead », si oui on active le timer sinon on le désactive.

## Pseudocode

Certaines parties de notre futur programme serait complexe à représenter via des logigrammes, nous avons donc décidé d’utiliser du pseudocode pour ces fonctions spécifiques complexes ou possédant un grand nombre d’instructions :

Une image contenant texte, capture d’écran

Description générée automatiquement

Pseudocode - Pseudocode de la fonction acquisition

La fonction acquisition permet de parcourir les capteurs et d’acquérir les données pour les stocker dans un tableau. Ces données ainsi acquises sont ensuite sauvegardées dans la carte SD si aucune erreur n’est détectée. Sinon, si le temps limite pour acquérir les données est dépassé, on passe la station en mode d’erreur.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pseudocode - Pseudocode de la fonction createTimer

La fonction **createTimer** permettra de réinitialiser un timer pour un délai désiré.

On utilise les registres **TCCR1B** (le registre qui permet notamment de régler le timer)**,** **TCNT1** (qui contient la valeur du timer), **TIMSK1** (qui permet de choisir le mode du timer). Pour régler le prescaler à 1024 et pour activer le mode CTC, on agit dans le registre TCCR1B.

On décide d’utiliser le timer en mode compare match car cela nous permet de ne pas interrompre de manière excessive le programme. en effet puisque la valeur maximum du registre de comparaison est de 65535, ce mode nous permet d’attendre que le timer atteigne une valeur plus grande avant d’effectuer une interruption.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pseudocode - Pseudocode de la fonction calculOCR

La fonction **calculOCR** servira à calculer la valeur du registre de comparaison (car on utilise le mode de timer « compare match ») pour un délai cible. On prend d’abord la fréquence de tick du microcontrôleur (16 MHz), que l’on divise par la valeur du prescaler. Puis on inverse cette valeur pour obtenir le temps entre deux ticks. On divise finalement la valeur de délai cible par cette valeur de temps pour obtenir la valeur optimale à mettre dans le registre de comparaison (OCRA1). On a ainsi le calcul suivant :

Comme la taille du registre de comparaison est de 16 bits, la valeur maximum que l’on peut mettre dedans est de 65535. Ainsi, dans le programme, lorsque cette valeur est dépassée, on stocke le reste dans la valeur retenue. On calcul ensuite la valeur du compteur (qui définit le nombre d’interruptions faites par le timer logiciel) en prenant en compte notre division. Puis on réinjecte le reste précédemment obtenu pour recréer un timer qui sera exactement de la durée pour compléter le timer.

La valeur de delaiCible doit être entrée en microsecondes.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pseudocode - Pseudocode de la fonction ISR

La fonction **ISR** (Interupt Service Routine) permettra d’interrompre le programme afin de faire le décompte de changement de mode. Elle prend en compte la retenue, c’est-à-dire que si le nombre de ticks dépasse 65535, elle répète l’interruption un nombre n de fois avec comme valeur de comparaison 65535, puis elle le réexécute une dernière fois avec une valeur plus petite pour atteindre le délai exacte demandé.

Lorsque le décompte est fini, elle appelle la fonction dont le pointeur est fourni en argument.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Pseudocode - Pseudocode de la fonction changerEnModeStandart

La fonction **changer en mode standard** est une fonction qui s’utilise dans un des cas de l’interruption.

Cette dernière vient sauvegarder le mode précédent, passer la variable mode sur la valeur standard (la valeur de MODESTANDARD est un entier qui est vérifier dans le switch du loop pour choisir quel mode activer, les fonctions suivantes suivent le même principe) et passe la couleur de la led en vert.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Pseudocode - Pseudocode de la fonction changerEnModeEconomie

La fonction **changer en mode Economie** est une fonction qui s’utilise dans un des cas de l’interruption.

Cette dernière vient sauvegarder le mode précédent, passer la variable mode sur la valeur économie et passe la couleur de la led en bleu.

Une image contenant texte, Police, capture d’écran

Description générée automatiquement

Pseudocode - Pseudocode de la fonction changerEnModeConfiguration

La fonction **changer en mode configuration** est une fonction qui s’utilise dans un des cas de l’interruption.

Cette dernière vient sauvegarder le mode précédent, passer la variable mode sur la valeur configuration et passe la couleur de la led en orange.

Une image contenant texte, Police, capture d’écran, Graphique

Description générée automatiquement

Pseudocode - Pseudocode de la fonction changerEnModeMaintenance

La fonction **changer en mode maintenance** est une fonction qui s’utilise dans un des cas de l’interruption.

Cette dernière vient sauvegarder le mode précédent, passer la variable mode sur la valeur maintenance et passe la couleur de la led en orange.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pseudocode - Pseudocode de la fonction changerMode

La fonction « **changerMode** » permet grâce à la boucle switch de changer de mode :

Selon la variable « **nouveauMode** » qui prend en valeur le mode souhaité, la boucle permettra d’activer ce mode.

Par exemple, dans le cas où **nouveauMode** est égal à **MODEECO**, la boucle exécutera la fonction **changerEnModeEconomie** qui permet de valider ce mode.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Pseudocode - Pseudocode de la fonction BoutonRougePressé

La fonction **BoutonRougePressé** permet d’interpréter un pressage du bouton rouge en fonction de la situation au moment de la pression. Elle effectue alors les instructions prévues.

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Pseudocode - Pseudocode de la fonction boutonVertPressé

La fonction **BoutonVertPressé** permet d’interpréter un pressage du bouton vert en fonction de la situation au moment de la pression. Elle effectue alors les instructions prévues :   
à l’aide de l’instruction switch, si le mode est en standard : on passe en mode économie puis vice-versa.

## Variables

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

|  |  |  |  |
| --- | --- | --- | --- |
| Adresse | Nom | Description | Domaine de définition |
| 0 | LOG\_INTERVAL | l’intervalle entre 2 mesures configurable depuis le mode configuration | {0,10} défaut : 10 |
| 1 | FILE\_MAX\_SIZE | La taille maximale d’un fichier de log (en octets) configurable depuis le mode configuration | Défaut 4096 |
| 2 | TIMEOUT | Durée en secondes au bout de laquelle l’acquisition des données des capteurs est abandonnée (au bout de deux mesures en timeout) | Défaut 30 |
| 3 | LUMIN | Variable de contrôle du capteur de luminosité | {0,1} |
| 4 | LUMIN\_LOW | Variable de seuil de luminosité faible | {0-1023} |
| 5 | LUMIN\_HIGH | Variable de seuil de luminosité forte | {0-1023} |
| 6 | TEMP\_AIR | Variable de contrôle du capteur de température ambiante | {0,1} |
| 7 | MIN\_TEMP\_AIR | Variable de seuil de température d’erreur minimale | {-40-85} |
| 8 | MAX\_TEMP\_AIR | Variable de seuil de température d’erreur maximale | {-40-85} |
| 9 | HYGR | Variable d’activation du capteur d’hydrométrie | {0,1} |
| 10 | HYGR\_MINT | Variable de seuil d’hydrométrie d’erreur minimale | {-40-85} |
| 11 | HYGR\_MAXT | Variable de seuil d’hydrométrie d’erreur maximale | {-40-85} |
| 12 | PRESSURE | Variable d’activation du capteur de pression | {0,1} |
| 13 | PRESSURE\_MIN | Variable de seuil de pression d’erreur de minimale | {300-1100} |
| 14 | PRESSURE\_MAX | Variable de seuil de pression d’erreur de maximale | {300-1100} |

# Conclusion :

Durant cette étape importante du projet, nous avons représenté de manière rigoureuse le fonctionnement de notre futur programme permettant le fonctionnement de notre prototype en respectant les contraintes définies lors du livrable précédent.

Nous avons maintenant différents logigrammes facilement compréhensibles permettant d’être sûr de notre but lorsque nous allons implémenter ce fonctionnement en langage C/Arduino.

Cette étape essentielle nous permet d’être prêt à l’implémentation, et nous servira également de support de présentation du fonctionnement du prototype.

# Bibliographie :

<https://moodle.cesi.fr/pluginfile.php/29159/mod_resource/content/7/res/CPI_A2-2021-SystemeEmbarque-WS_Pile_-_ELEVES.pdf>

<https://moodle.cesi.fr/pluginfile.php/29183/mod_resource/content/8/res/WS_ListesChainees_Etudiants.docx>

<https://draw.io>

<https://www.arduino.cc/reference/fr/>

<https://github.com/pjpmarques/ChainableLED>

<https://www.arduino.cc/reference/en/libraries/adafruit-bme280-library/>

<https://www.lucidchart.com/pages/fr/logigramme>