Rapport de projet – ELEAR6EM

Projet Ruche Connectée

Projet réalisé par :

BAYLE Laetitia
NIRO Enzo

Sous la direction de Périssé Thierry

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Licence Électronique Énergie électrique Automatique
parcours Réorientation vers les Études Longues

Année 2020-2021

Table des matières

Résumé / Abstract

• Le projet de la ruche connecté consiste à embarquer un Arduino dans une ruche pour lire une température, une humidité et autres capteurs, pour l'envoyer finalement à travers une transmission longue distance sans fil vers un serveur qui lira la trame et sauvegardera les valeurs dans une base de données.

• L'utilisateur peut modifier les paramètres et lire les valeurs sur la ruche même grâce à une IHM locale, une IHM intégrée au serveur embarqué par une Raspberry PI, ou/et une page web hébergée par le serveur. L'utilisateur doit être capable de regarder les valeur depuis son réseau local, et si possible à travers Internet !

• L'objectif principal de ce projet, est d'être capable de lire les valeurs des capteurs et de les transmettres à l'aide de la modulation LoRa, afin que le serveur puisse lire la trame et traçer les résultats sur un graphe. Il doit être capable de les interpréter dans un noeud de réseau radio.

• Connected hive project consist to embed an Arduino into a Hive to read temperature, humidity, and other sensors, to finally send it through a long range wireless connectivity to a server which read and store these values into a database.

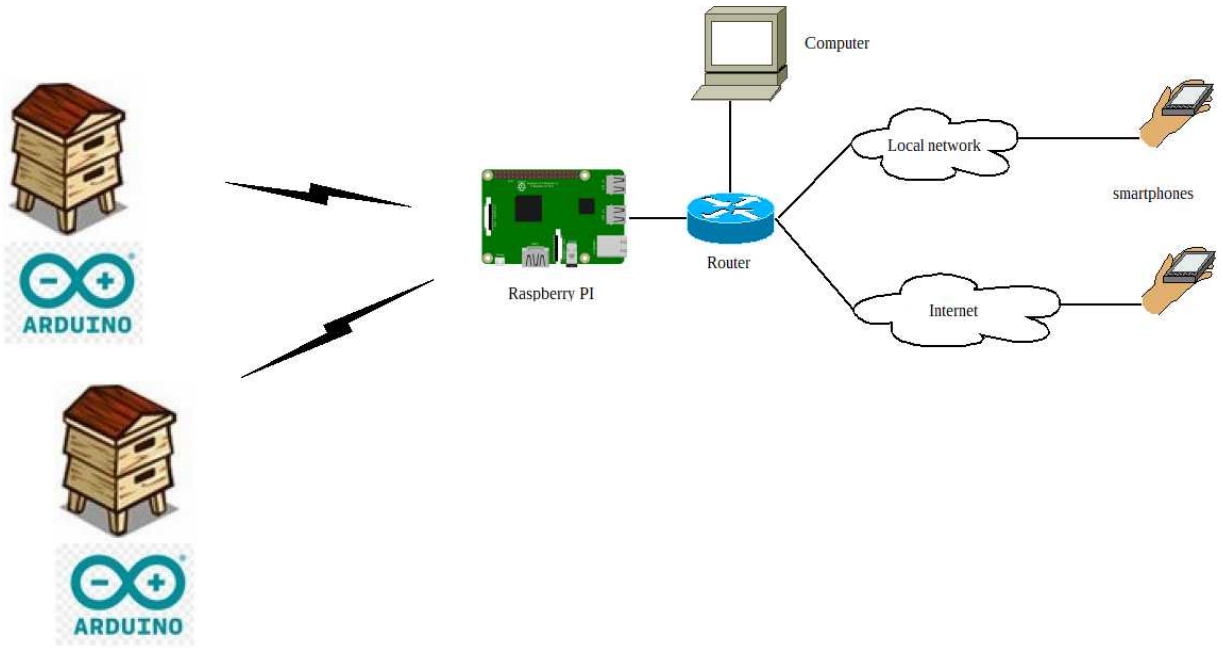
• User can interact with parameters, and read values directly on hives themselves with the local HMI, or check hive's values on the server itself with it integrated HMI, which consist by a touch-capacitive display, or by going to the server web page host by a Raspberry PI. Of course user must be able to check hives' values in his local network, maybe through Internet !

• The main goal of this project is to read data by the embedded hive, and plot values on the web page of the server. And the second step, is the same thing, but with more hives ! The objective is to create a node network that the main server must be capable to read all hives !

Cahier des charges

Objectif du projet :

Réalisation d'une ruche connectée, qui permet de transmettre des relevés effectués tels que : sa température, son humidité... La transmission se fera via une liaison sans fil, à longue portée, directement sur un serveur qui sauvegardera les informations dans une base de données qui pourra être visualisée à l'aide d'une IHM embarquée et/ou une page Web dédiée.



Le travail demandé :

• Les principales raisons d'instrumenter une ruche.

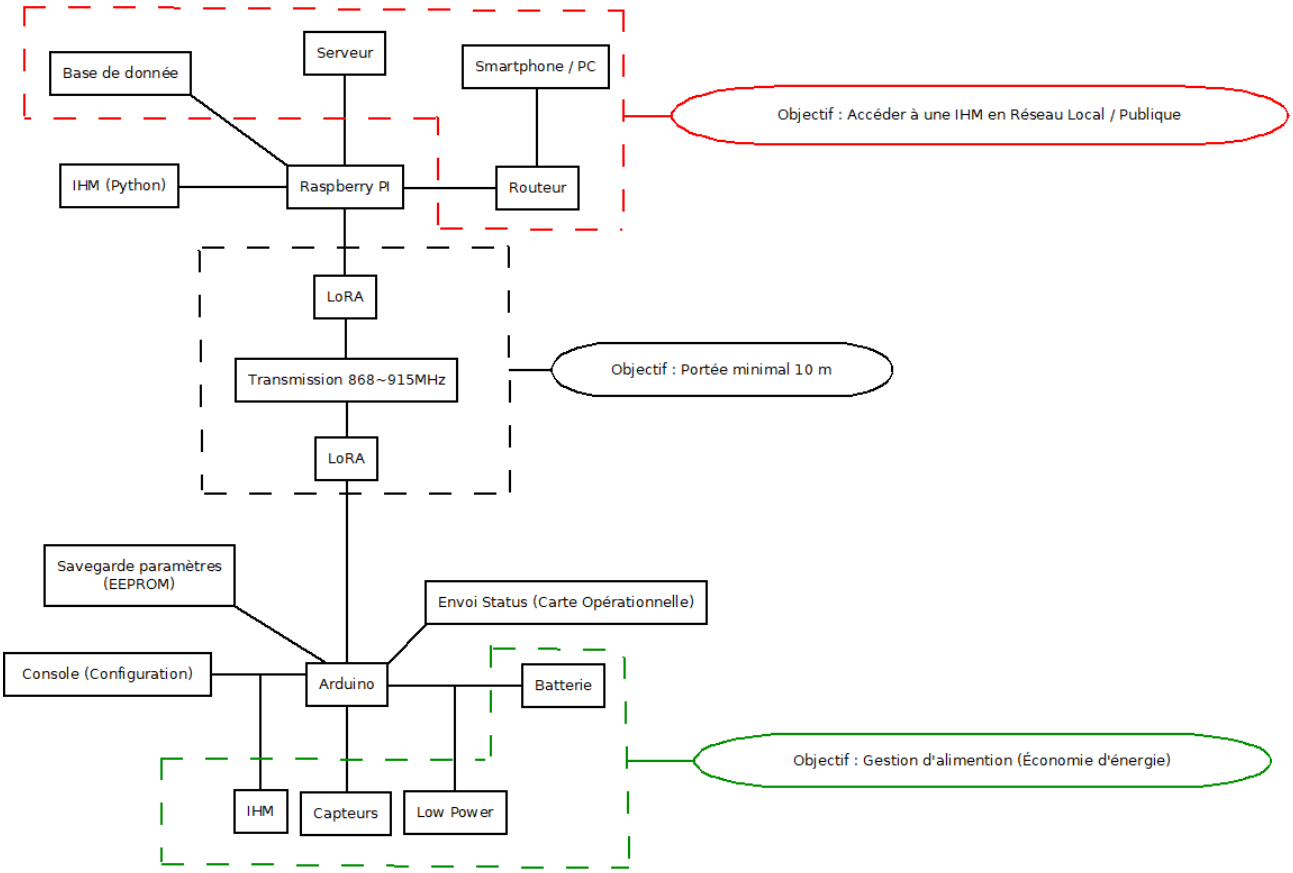
• Gestion de projet :
→Analyse fonctionnelle
→Cahier des charges
→Diagramme de Gantt (disponible en **annexe 14**)

• Choix des capteurs à utiliser (en tenant compte des contraintes environnementales)

• Partie Arduino :
→Prise en main des capteurs et affichage en local (IHM)
→Création d'une console pour modifier et sauvegarder les paramètres d'une ruche
→Gestion d'alimentation pour l'économie d'énergie
→Élaborer une transmission sans fil vers le serveur

• Partie Raspberry Pi :
→Prise en main du système UNIX approprié et intégration d'un affichage en local (IHM)
→Prise en main et configuration d'une plateforme en Python
→Gestion d'un serveur, d'une base de données et intégration d'une page Web
→Transmission et réception des paramètres d'une ruche

Analyse fonctionnelle :



Quelles sont les principales raisons d'instrumenter une ruche ?

Instrumenter une ruche à l'aide de capteurs de température, d'humidité, de masse, de présence et bien d'autres, permet d'enregistrer des données physiques. L'apiculteur pourra donc suivre l'évolution environnementale autour de la ruche concernée. En effet, grâce à cette instrumentation, il pourra agir en fonction des résultats obtenus. Par exemple, si une valeur importante est relevée à l'aide du capteur de masse, l'apiculteur aura l'information qu'il pourra récolter assez de miel dans cette ruche. De même, si le capteur de présence détècte peu de passages d'abeilles, il sera en mesure de remettre en question son emplacement, et donc son environnement, ou éventuellement savoir la cause de cet incident, plus rapidement.

Choix du matériel et estimation des coûts

Suite au cahier des charges établi, nous avons fait le choix des composants en prenant en compte nos objectifs à atteindre à moindre coût.

Composant	Description / Caractéristiques	Quantité	Prix Unitaire TTC (€)	Prix total TTC (€)
Grove DHT22	3,3V – 6V DC -40°C ~ 80°C 0% ~ 100%	1	10,39€	10,39€
Arduino UNO	7V ~ 12V Microcontrôleur ATmega328 32 Ko de Flash Mémoire 2 Ko de Ram 1 Ko de EEPROM	1	19,94€	19,94€
Raspberry PI 4 (2GB version)	Processeur Broadcom BCM2711, Quad core Cortex-A72 1,5 Ghz 2 Go de Ram	1	56,00€	56,00
Dragino LoRa Shield	868 MHz ~ 915 Mhz	2	21,34€	42,68€
PITFT (3.5")	Résolution 480x320	1	46,39€	46,39€
Grove RGB LCD 16 x 2	3,3V ~ 5V	1	14,94€	14,94€
Grove Encodeur		1	5,02€	5,02€
Grove Bouton		1	2,00€	2,00€
Relai	5V	1	4,80€	4,80€
Grove Shield	Connectique pour le développement	1	4,80€	4,80€
			TOTAL TTC :	206,96€

On suppose qu'il serait préférable de créer notre propre carte afin d'optimiser au maximum le coût de notre projet.

Étude et fonctionnement des composants

Le capteur DHT22

Description générale :

Le rôle de ce capteur est de relever la température et l'humidité. Il doit s'alimenter entre 3,3V et 6V pour une consommation 1,5 mA. Il peut relever une température se situant de -40°C à 80°C avec une précision de ±0,5°C, et une humidité entre 5% à 99% d'une précision de ±2%. La résolution du capteur est 0,1 pour la température ou l'humidité.

Description technique :

Pour communiquer avec le capteur, il faut utiliser le bus 1-Wire, qui est une transmission asynchrone sur une connection. Nous pouvons émettre et recevoir une donnée, sachant que chaque communication dure 25 ms au total pour une séquence de start et la transmission de la trame.

Voici la représentation d'une transmission complète :

La transmission de la trame ne dure que 4 ms.

Si nous faisons un zoom sur la trame, nous pouvons observer qu'elle est composée :

- **d'un bit de start**
- **d'un bit de réponse**
- **de 16 bits de données pour l'humidité**
- **de 16 bits de données pour la température**
- **de 8 bits de checksum**
- **d'un bit de stop**

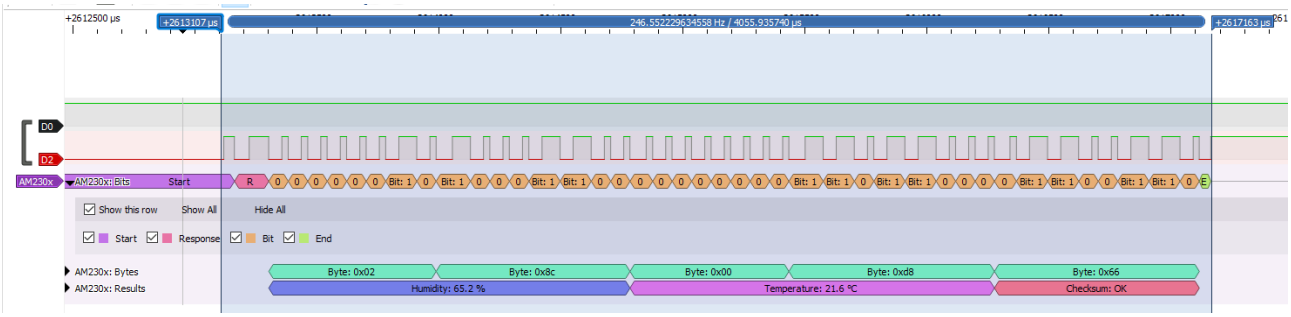


Illustration 1: Trame du capteur DHT22 pour la lecture d'une température de 21,6°C et d'une humidité de 65,2% à l'aide du logiciel PulseView

C'est le cas comme le démontre la datasheet (**p.3**) du DHT22 / AM2302

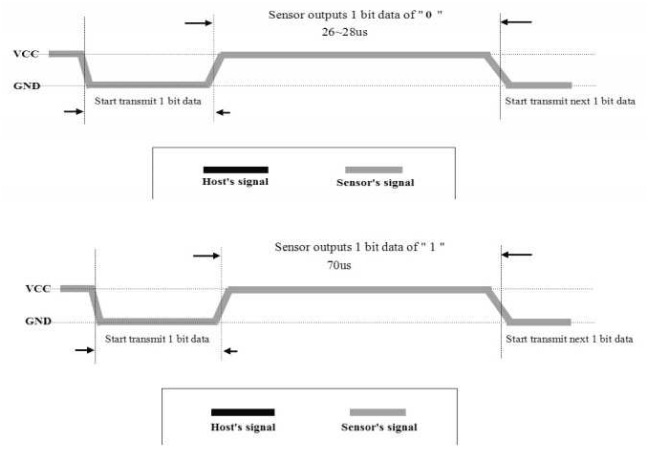
DATA=16 bits RH data+16 bits Temperature data+8 bits check-sum

Example: MCU has received 40 bits data from AM2302 as

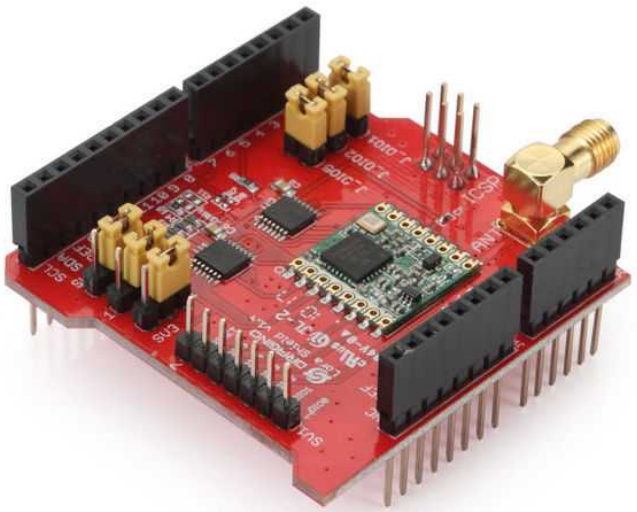
0000 0010 1000 1100	0000 0001 0101 1111	1110 1110
16 bits RH data	16 bits T data	check sum

Le checksum est le résultat du poids faible de la somme des 4 octets précédents de la trame, pour vérifier si les valeurs sont correctes.

D'après la documentation (**p.4**), nous pouvons constater que le codage des bits est défini sur la durée du signal à l'état haut précédé par un niveau bas pour informer l'envoi d'un bit.



Dragino LoRa Shield



Description générale :

Les cartes (ou shields) LoRa sont basées sur des modules **RFM95** capables de gérer des transmissions sans fil de très longue portée, capables d'envoyer une trame à **35 Km en champ libre**, tout en opérant aux fréquences **868 MHz ~ 915 MHz**. Ils sont capables de gérer plusieurs modulations (comme la **FSK** ou **OOK**), dont la modulation qui nous intéresse ici, la modulation **LoRa**.
Nous avons effectué un test d'envoi avec des obstacles, le résultat de la portée est disponible à l'**annexe 12** et un relevé spectral disponible à l'**annexe 13**.

Un avantage que peut également présenter les RFM95, ces sont des transmetteurs **Low Power**. Leurs dimensions physiques sont de 16mm*16mm, leurs permettant une intégration facile à embarquer dans le routage d'une carte électronique. Le RFM95 dispose de 5 broches DIO utilisées pour communiquer diverses informations(Réception/émission terminée, Mode opérationnel, Timeout réception, ...)

La modulation LoRa ?

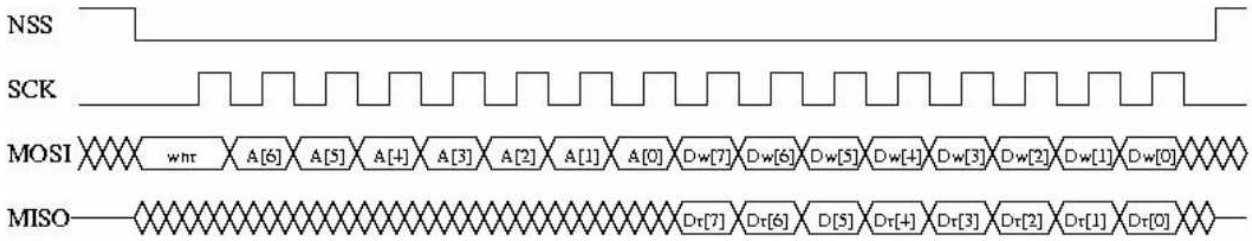
C'est une modulation de fréquence, similaire à la FSK. L'avantage de cette modulation c'est, qu'elle se base sur des sauts de fréquence pour envoyer directement des mots binaires, contrairement à la FSK qui se base sur 2 fréquences, pour envoyer un bit à l'état 0 ou 1.

Description technique :

Le module RFM95 doit être alimenté idéalement en 3.3V, mais il peut être aussi alimenté entre 1.8V ~ 3.7V. Les signaux entrants ne doivent en aucun cas dépasser la tension d'alimentation maximale, alors que les signaux de sorties dépendent de l'alimentation. Le shield de dragino propose de réajuster les 5V en 3.3V à l'aide de deux **74VHC125**.

Pour piloter les modules, les RFM95 intègrent **une liste de registres**, pour effectuer des actions selon leurs paramètres (paramétrage fréquence, modulation, puissance d'émisssion,...).

Toute commade à envoyer au RFM95, se fait par le bus SPI.
Chaque instruction est envoyé au format 16 bits, le premier octet indique l'adresse du registre auquel nous voulions accéder, ainsi que le paramètre que l'on souhaite écrire.



D'après la datasheet (**p.75**), il faut noter que le premier octet est composé de 7 bits d'adresse et d'un bit de commande (wnr), signifiant : le souhait d'écrire dans un registre, ou tout simplement lire ce registre (**respectivement 1 ou 0**).

Mise en situation :

Nous allons vous présenter 2 séries d'instructions qui porteront sur le **paramétrage de la puissance d'émission**, ainsi que **l'envoi d'une trame** (à savoir la chaine de caractère suivante : **eea**)

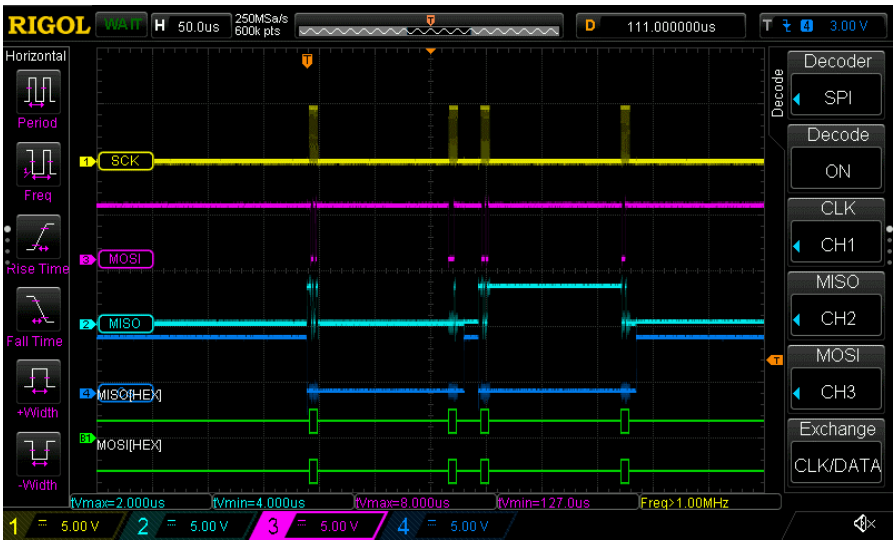
Paramétrage puissance d'émission

Si nous envoyons l'instruction suivante à l'arduino, nous émettrons une puissance d'émission de **+20dBm** :

```
rf95.setTxPower(20, false ; // puissance d'émission
```

Le relevé à l'oscilloscope nous donne donc la trame complète représentée ci-dessous.

Il est à noter que la trame SPI de l'arduino est configurée en **Mode 0** (La **polarité de l'horloge à 0** ainsi que **la phase à 0**, comme indiqué dans la datasheet **p.75**) avec une fréquence de **1 Mhz**.



Il faut noter que nous nous intéressons seulement aux octets envoyés, donc aux octets sur le channel 3 étiquettée **MOSI (Master Out Slave In)**.

En faisant un zoom sur l'échelle temporelle, nous pouvons observer les différentes consignes envoyées (Référées au annexes 1 2 et 3) que nous allons expliciter en détails.

La trame envoyée et lue à l'oscilloscope est la suivante :

Adresse	Donnée	Adresse	Donnée
0xCD	**0x07**	**0x89**	**0x8F**

0xCD correspond à l'écriture du registre **0x4D** (**0x4D OU 0x80**, avec **0x80** l'octet pour un bit wnr à 1), qui est le paramètre du registre **RegPaDac** permettant d'activer l'option**+20dBm** à la broche **PA_BOOST** liée à l'antenne, à condition que la donnée écrite à l'adresse de ce registre (soit **0x4D**), soit à la valeur de **0x07** et que le paramètre **OutputPower soit 0x0f** .

0x07 est la donnée écrite au registre **0x4D**, donc sa valeur initiale est 0x04 par défaut d'après la datasheet (**p.99**).

	3-0	reserved	rw	0x09	Reserved. Retain default value.
RegPaDac (0x4d)	7-3	reserved	rw	0x10	reserved. Retain default value
	2-0	PaDac	rw	0x04	Enables the +20dBm option on PA_BOOST pin 0x04 → Default value 0x07 → +20dBm on PA_BOOST when OutputPower=1111
RegFormerTemp	7-0	FormerTemp	rw		Temperature saved during the latest IQ (RSSI and Image)

0x89 correspond à l'écriture du registre **0x09** (**0x09 OU 0x80**), qui est le paramètre du registre **RegPaConfig** permettant d'attribuer la puissance d'émission sur l'antenne.

0x8F est la donnée écrite au registre **0x09**, permettant d'activer la broche **PA_BOOST** (soit **0x80**), ainsi que d'attribuer la valeur **0x0f** au paramètre **OutputPower**.

Nous avons une puissance d'émission de +20dBm (qui est la puissance maximale)

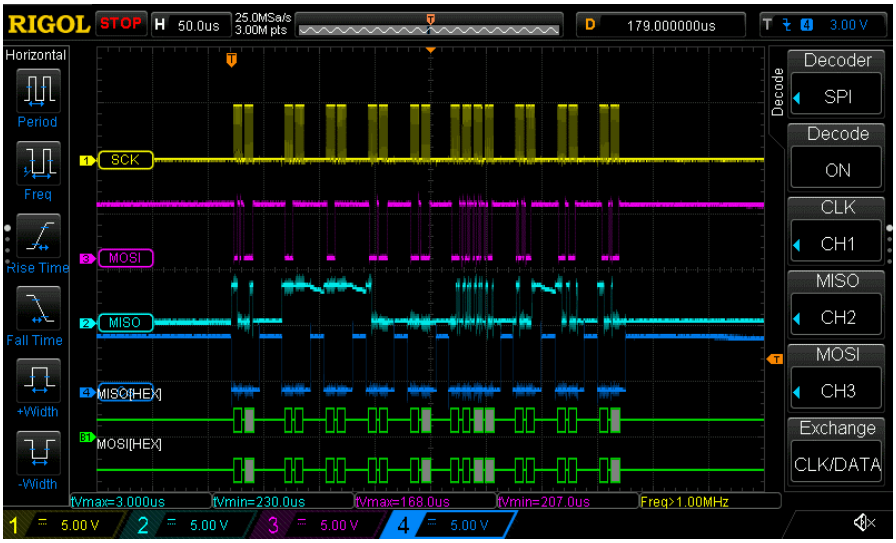
Registers for the Transmitter					
RegPaConfig (0x09)	7	PaSelect	rw	0x00	Selects PA output pin 0 → RFO pin. Maximum power of +14 dBm 1 → PA_BOOST pin. Maximum power of +20 dBm
	6-4	MaxPower	rw	0x04	Select max output power: Pmax=10.8+0.6*MaxPower [dBm]
	3-0	OutputPower	rw	0x0f	Pout=Pmax-(15-OutputPower) if PaSelect = 0 (RFO pins) Pout=17-(15-OutputPower) if PaSelect = 1 (PA_BOOST pin)

Envoi d'une trame (chaîne de caractère : **eea**)

A l'aide des instructions suivantes, nous émettons le message "**eea**" :

```
char message[3] = “eea”;
rf95.send((uint8_t *)message, 3);
```

Nous pouvons obtenir le relevé suivant à l'oscilloscope :



Si nous faisons un zoom sur l'échelle temporelle vers le milieu de la trame, nous pourrons retrouver les codes ASCII du message "**eea**", soit **0x65**, **0x65** et **0x61,** se finissant par une commande de validation d'envoi de la trame (référencée à l'annexe 4).

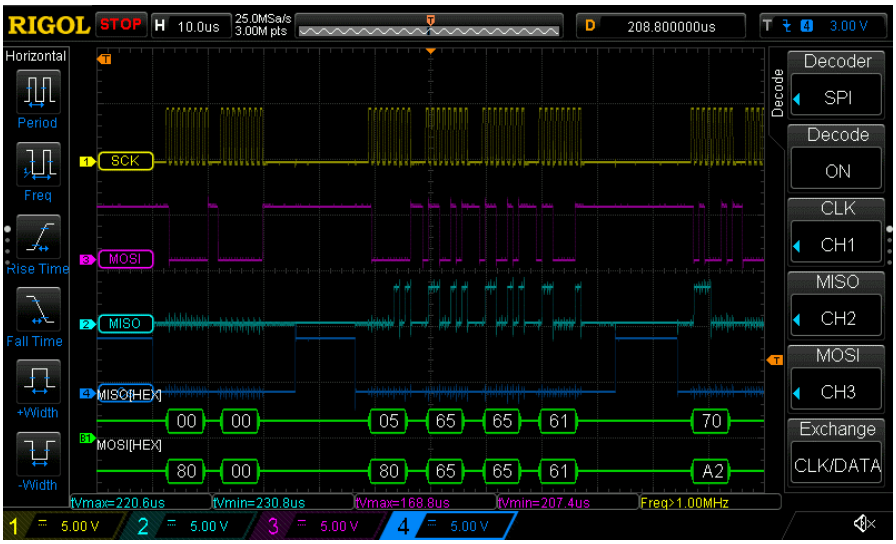


Illustration 2: Représentation de la trame du module LoRA pour l'envoi du message "eea"

Nous allons donner quelques détails des octets envoyés au RFM95 de la trame complète :

Adresse registre	Donnée associé	Description
0x8D	0x00	Écriture dans le registre **0x0D** → **RegFIFOAddrPtr** Donnée tout le temps à **0x00** → **Adressage des valeurs envoyées par SPI dans le buffer FIFO**
0x80	0xFF	Écriture dans le registre 0x00 →Écriture de **0xFF** au buffer **FIFO**
0x80	0xFF	Écriture dans le registre 0x00 →Écriture de **0xFF** au buffer **FIFO**
0x80	0x00	Écriture dans le registre 0x00 →Écriture de **0x00** au buffer **FIFO**
0x80	0x00	Écriture dans le registre 0x00 →Écriture de **0x00** au buffer **FIFO**
0x80	0x65 0x65 0x61	Écriture dans le registre 0x00 →Écriture de "**eea**" au buffer **FIFO**
0xA2	0x07	Écriture dans le registre **0x22** → **RegPayloadLength** **7 octets** à envoyer
0x81	0x03	Écriture dans le registre **0x01** → **RegOpMode (mode opérationnel)** **0x03 → envoi du buffer FIFO**
0xC0	0x40	Écriture dans le registre **0x40** → **RegDioMapping1** **0x40 → Mise à 1 de la broche DIO0**

Note: • FIFO → Acronyme de First In First Out, est une mémoire de type file d'attente.

• La broche DIO0 est utilisée en guise d'interruption sur l'arduino pour permettre une bonne réception et décodage de la trame.

Arduino Uno

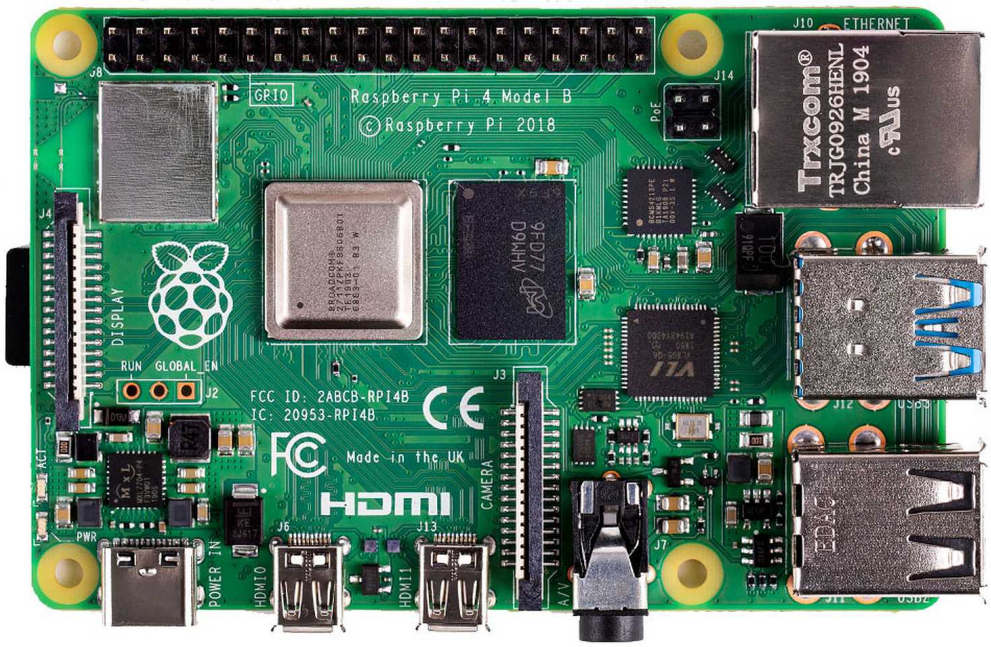
Description générale:

Arduino Uno est une carte de développement, basé d'un microcontrôleur 8 bits Atmega328. Il a une capacité de mémoire flash de 32 Ko, une Ram de 2 Ko et une EEPROM de 1Ko.

L'Arduino Uno est composé de 20 entrées/sorties, dont 6 dédiées à la conversion analogique numérique. Il est capable de gérer des bus hardwares(SPI, I2C, UART), des interruptions externes, lire des tensions à l'aide de convertisseurs analogiques numériques et applications diverses à l'aide d'entrées sorties.

Ce matériel permet de réaliser des applications ne nécessitant pas énormément de performances (Lecture d'une trame, envoyer un octet, contrôler un écran basse résolution, générer un signal).

Raspberry PI 4 (2 GB version)



Description générale :

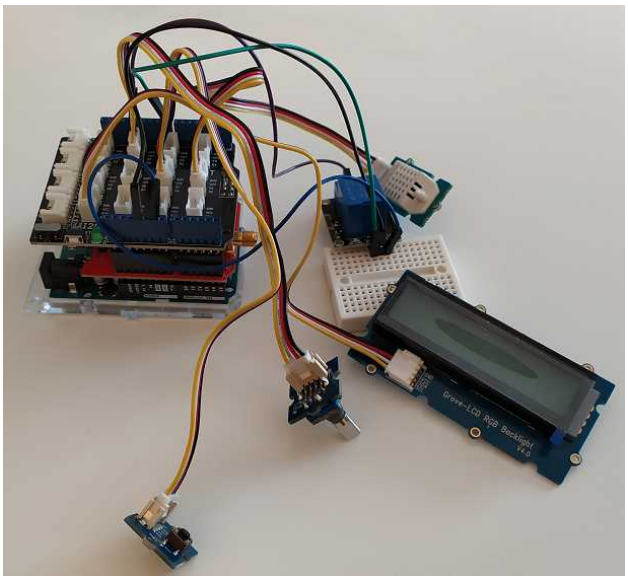
Raspberry PI est un micro-ordinateur, doté d'un processeur Processeur Broadcom BCM2711, Quad core Cortex-A72 cadencés à 1,5 Ghz par défaut. Il dispose de 2 Go de Ram, d'autres versions proposent soit 4 Go de Ram ou bien 8 Go. Contrairement à l'Arduino Uno, il ne dispose en aucun cas d'une mémoire flash interne, nécessitant une mémoire externe, qui est dans ce cas présent, une carte µSD.

La Raspberry PI est composée de 28 entrées/sorties. Elle est capable de gérer des bus hardwares (SPI0, SPI1, I2C, UART). Contrairement à l'Arduino, elle ne peut pas lire des tensions analogiques, car elle ne dispose d'aucun convertisseur analogique/numérique.

Ce matériel permet de réaliser des tâches nécessitants un minimum de ressources (embarquer un système UNIX, dans notre cas Raspberry PI OS basé sur la distribution Debian Linux). Il est à noter que pour le fonctionnement de ce projet, une Raspberry PI 4 modèle 2 Go de Ram, est le minimum requis pour lancer l'application que nous avons développé. Une démonstration d'un benchmark entre une Raspberry PI 4 modèle B 4 Go de Ram et une Raspberry PI 3 modèle B est disponible à l'**Annexe 5**.

Remarque : Le développement a été fait sous une Raspberry PI 3 modèle B ! La Raspberry PI 4 a été utilisée dans le cadre des tests et performances.

Ruche embarquée



Préface : Dans cette partie nous allons utiliser un Arduino Uno, capteur DHT22 pour lire la température et l'humidité, un écran LCD, un bouton et un encodeur pour gérer et prendre en main l'IHM, ainsi qu'un relai pour la gestion d'alimentation. Bien entendu, un module LoRa est implémenté pour la transmission sans fil. Nous avons réaliser le schéma structurel disponible à l'annexe 7.

Gestion de l'IHM

Encodeur et bouton

L'utilisateur de l'encodeur et du bouton permet de pouvoir naviguer d'un menu à l'autre, plus facilement.

Le fonctionnement de l'encodeur est défini par 2 sens de rotations (1 cran à gauche respectivement 1 à droite), qui emmetra 2 signaux identiques mais décalés sur 2 broches différentes :

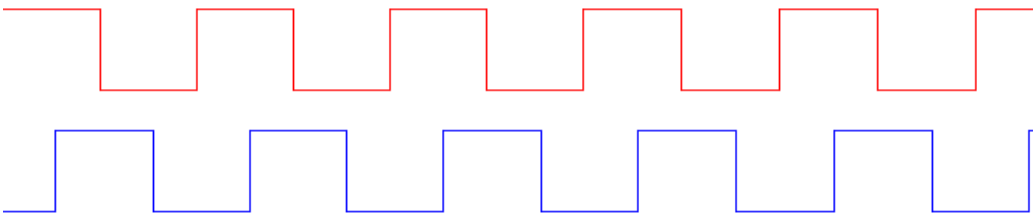


Illustration 3: Représentation des signaux du premier sens de rotation, broche SIGA → signal rouge, SIGB → signal bleu

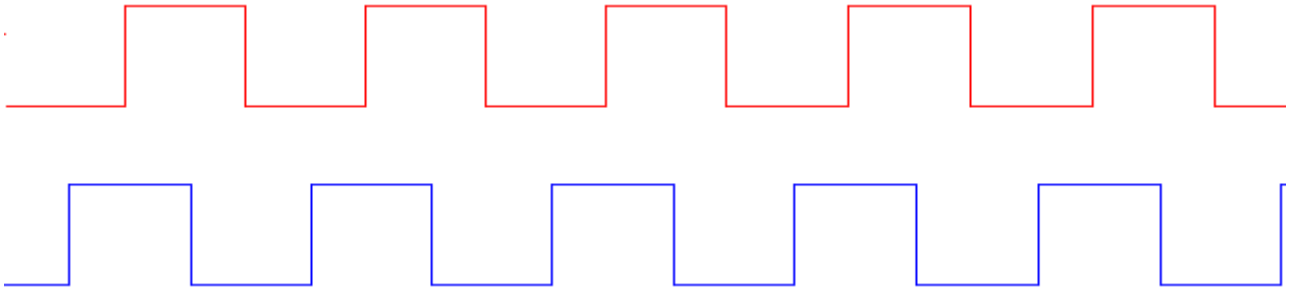
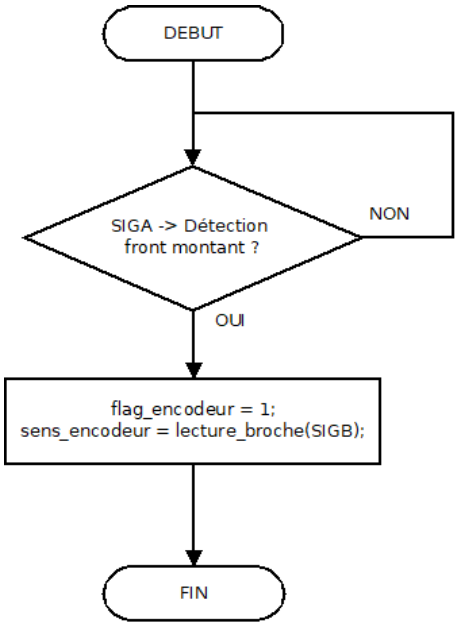


Illustration 4: Représentation des signaux du second sens de rotation, broche SIGA → signal rouge, SIGB → signal bleu

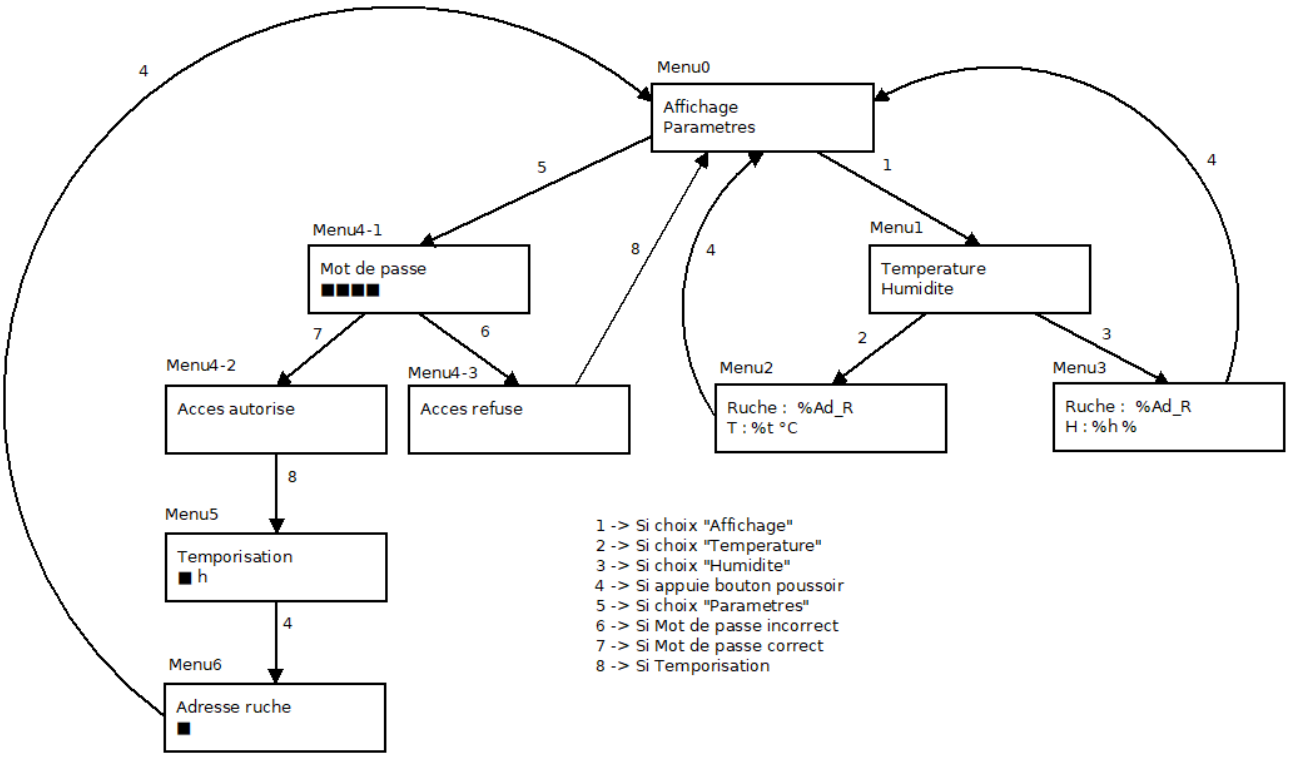
En prenant en compte ces différentes représentations, nous avons établi la structure algorithmique suivante :



Lorsque **SIGA** émet un front montant suite à l'utilisation de l'encodeur, on écrit la valeur 1 à la variable **flag_encodeur** pour avertir qu'il y a eu manipulation du composant. Et donc on vérifie l'état de la broche **SIGB**, pour connaitre le sens de rotation. Cela nous permettra d'interagir avec la navigation des menus qui sera explicitée par la suite.

La sélection des menus et des paramètres se fera à l'aide d'un bouton poussoir.

Architecture des menus



A l'aide du diagramme réalisé ci-dessus, nous avons pu développer une navigation sur un écran LCD.

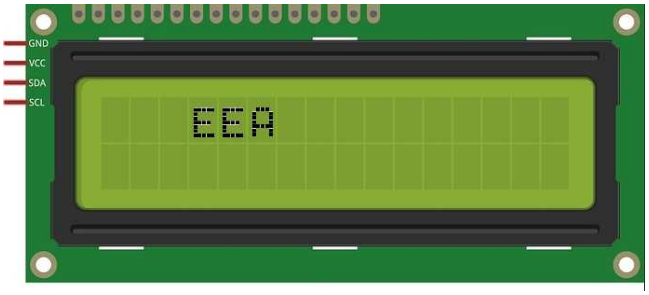
Fonctionnement du LCD

Pour écrire un message sur l'écran LCD, il faut prendre en compte les coordonnées du curseur et la chaine de caractères. En effet, l'écran peut afficher 2 lignes de 16 caractères, d'où le format **16x2**.

Par exemple pour écrire le message "**EEA**" sur la première ligne en commençant à la 4ème case, nous devons utiliser les lignes de code suivantes :

```
lcd.setCursor(3,0) ; //placement du curseur en 1ere ligne 4eme case
lcd.print(‘’EEA’’) ; //écriture du message « EEA »
```

Nous aurons donc à l'écran l'affichage suivant :

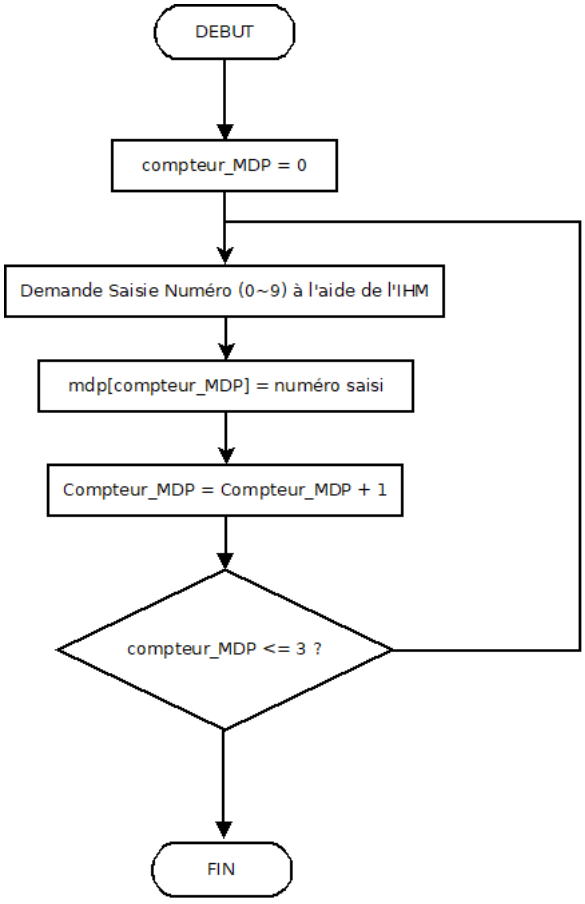


Sécurité des paramètres

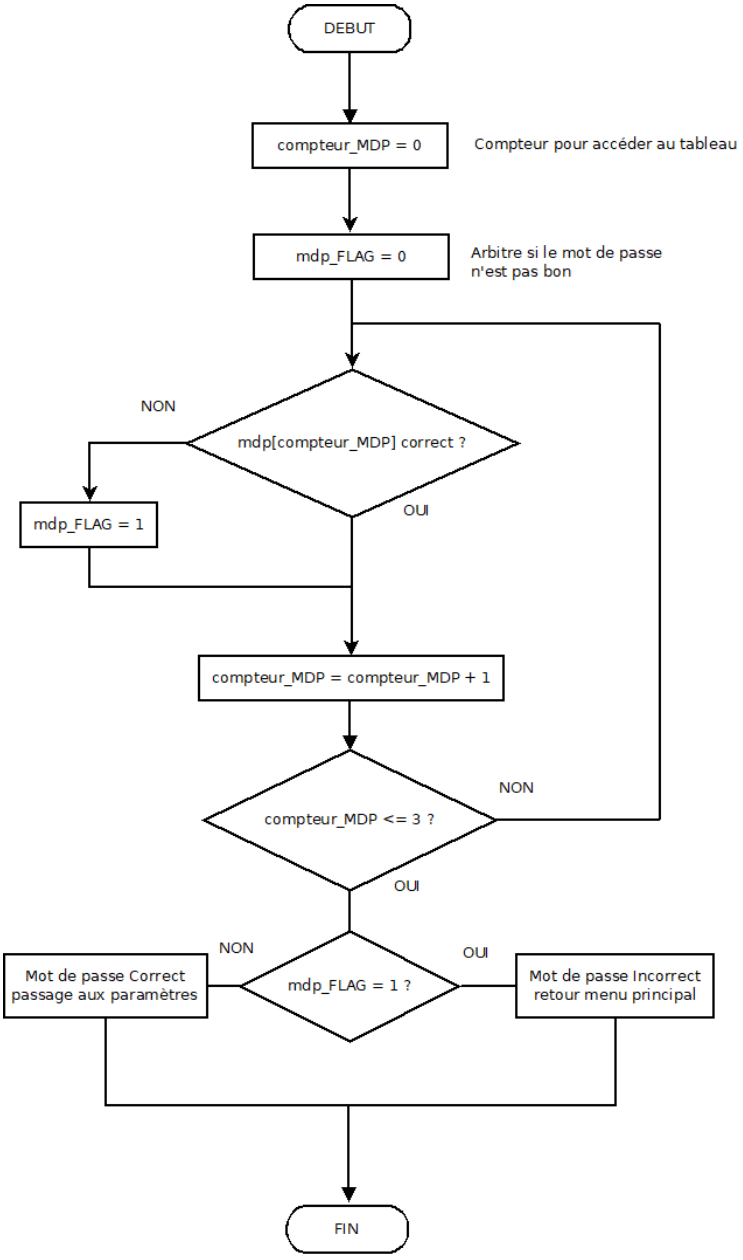
Afin que les paramètres ne soient changés malencontreusement par autrui, ils sont protégés à l'aide d'un mot de passe qui laissera accès si seulement le mot de passe n'est pas erroné. Dans le cas contraire, le système renvoi l'utilisateur au menu principal. Le mot de passe est défini dans un code pin à 4 chiffres.

Le déroulement de la session se fait en 2 étapes, la première par la saisi du code pin, la deuxième par la vérification si le code correspond au code attendu. Le fonctionnement est décrit par les 2 algorigrammes ci-dessous et un visuel est disponible à l'**annexe 6** :

Algorigramme de la saisie du mot de passe



Algorigramme de la vérification du mot de passe



Sauvegarde et chargement des paramètres

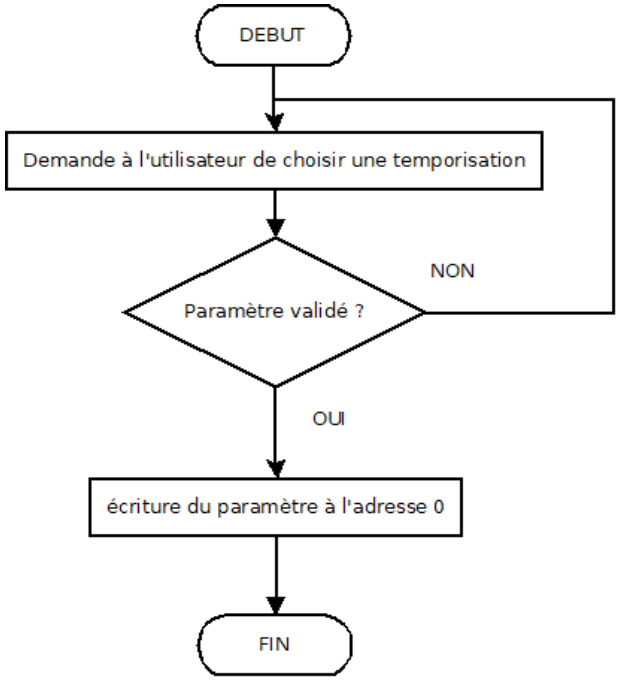
La savegarde et le chargement des paramètres se fait à l'aide de l'**EEPROM** (Electrically Erasable Programmable Read-only Memory) du microcontrôleur. Elle est capable de garder en mémoire des données lorsque le système est allumé ou éteint, d'où le terme technique "**mémoire morte**".

Dans le cadre de notre programme, nous allons utiliser deux fonctions internes à la core d'Arduino, telles que **EEPROM.write(adresse, valeur)** et **EEPROM.read(adresse)**.

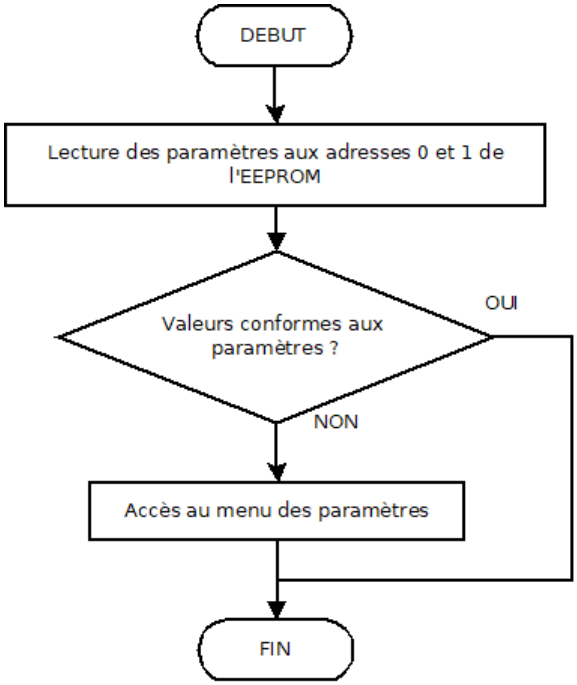
La fonction **write()**, permet d'écrire un octet à l'adresse souhaitée dans l'EEPROM.
La fonction **read()**, permet de lire un octet à l'adresse souhaitée dans l'EEPROM.

Dans le cadre de notre programme, deux exemples seront présentés dans les algorigrammes ci-dessous :

Écriture d'un octet dans l'EEPROM



Lecture des paramètres dans l'EEPROM



Remarque : "Valeurs conformes aux paramètres ?" est une sécurité pour vérifier si le paramètre de la temporisation est compris dans l'intervalle **[1 ~ 24]**, et verifier si la valeur de l'adresse de la ruche est comprise dans l'intervalle **[1 ~ 99]**.

Gestion Alimentation

Une gestion d'alimentation est nécessaire afin d'avoir l'autonomie la plus optimale.
En effet, il faut réduire la consommation de courant sur notre système.

Il est à noter que chaque composant est alimenté en 5V

Composant	Consommation (en A)
Arduino Uno	19 mA
Dragino Shield LoRa	0,190 µA si le module ne communique pas 11,3 mA si le module reçoit 117 mA si le module transmet
Grove RGB LCD	→ 22 mA si la backlight est allumée (couleur Rouge) → 2 mA si la backlight est éteinte
Relai	Environ 40 mA si le relai est activé Sinon environ 0 A
DHT22	→ 50µA au repos → 1,5mA Si transmission
Encodeur	550 µA si aucune rotation
Bouton	550 µA si appui Sinon 0 A

Dans le cadre du circuit optimisé, la consommation totale serait de **21.5 mA** lorsque l'alimentation du capteur DHT22 est gérée par le relai, et lorsque la backlight de l'écran est éteinte. Il est à noter également que le système ne communique pas avec le serveur Raspberry PI, et que l'IHM (Encodeur, LCD et Bouton) n'est pas manipulée.

Si nous faisions notre propre carte, nous pourrions économiser environ 19 mA, car la carte Arduino Uno présente des leds allumées en permanence, qui sont source de consommation. Nous pourrions dans ce cas optimiser le système de façon à ce qu'il ait une consommation d'environ **2 mA** au repos.

Si nous voulions avec cette nouvelle carte, avoir une autonomie de 2 mois, il faudrait avoir une batterie ayant une capacité de **3 Ah**.

$$Autonomie = 2\,mois \simeq 62\,jours = 1488\,heures = 1488\,h * 2\,mA = 2976\,mAh \simeq 3\,Ah$$

Remarque : Il est préférable d'approfondir cette gestion d'énergie, car les observations faites sont strictement théoriques. De plus, la charge par panneau solaire n'est en aucun cas présentée dans ces conditions.

Communication avec le serveur Raspberry PI

Pour communiquer avec la Raspberry PI, nous utilisons les modules LoRa (explicité **p.7 ~ 11**). Nous avions défini une trame finale stable et sécurisée (baptisée **IOB protocol v1.0**). La trame est définie comme ci-dessous :

Identifiant	Commande	Données
5 octets	1 octet	X octets

Identifiant : 5 octets définis suivant la forme : **73.79.66.X.Y**
→ avec X l'adresse du récepteur, ici 0 qui est l'adresse assignée au serveur.
→ avec Y l'adresse de l'émetteur, ici l'adresse de la ruche qui est comprise en **1 et 99**.

Commande : Permet d'envoyer soit :
→ un accusé de réception
→ une requête
→ une commande

Données : Données envoyées selon l'octet "**Commande**".

Cependant, nous avons opté pour une solution plus rapide et simplifiée définie par cette trame prototype :

Identifiant	Données température	Données Humidité
2 octets	2 octet	2 octets

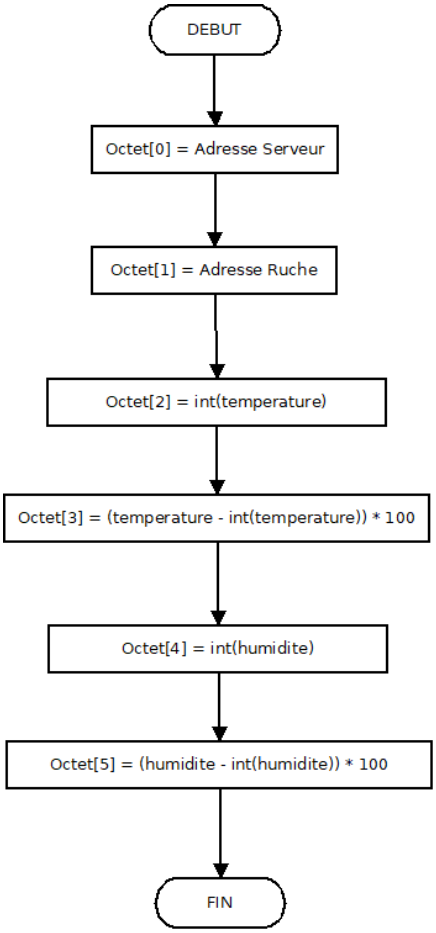
Identifiant : 2 octets définis suivant la forme : X.Y
→ avec X l'adresse du récepteur, ici 0 qui est l'adresse assignée au serveur.
→ avec Y l'adresse de l'émetteur, ici l'adresse de la ruche qui est comprise en 1 et 99.

Données de la température :
→1 octet pour la partie entière de la température
→1 octet pour la partie décimale de la température

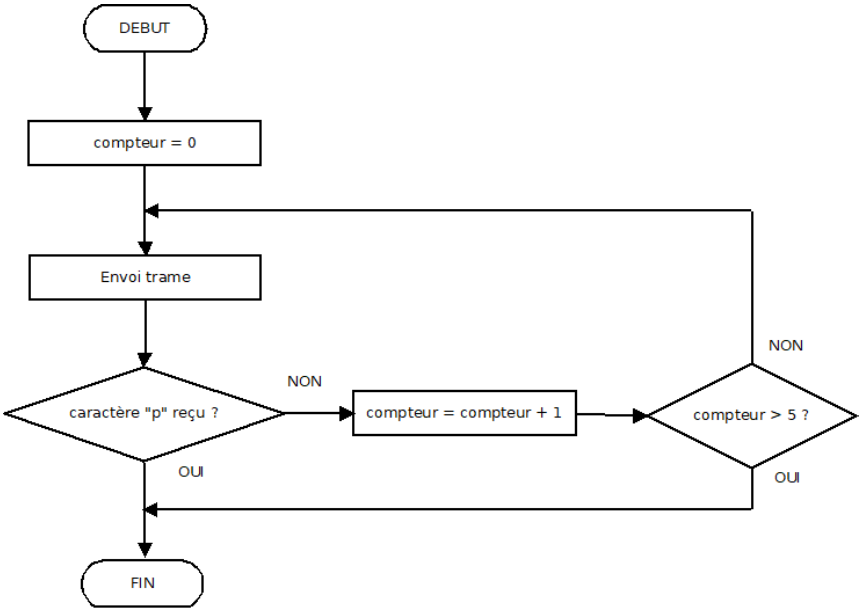
Données de l'humidité :
→1 octet pour la partie entière de l'humidité
→1 octet pour la partie décimale de l'humidité

Remarque : L'accusé de réception sera caractérisé par l'envoi d'un octet à la valeur 112.

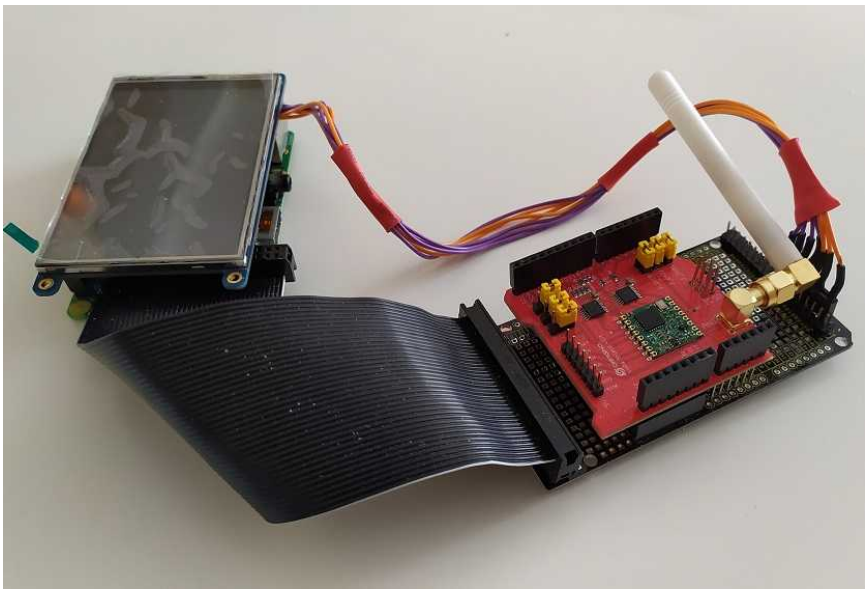
L'algorigramme de la fonction d'envoi est défini ci-dessous suivant la trame prototype :



Pour la réception, nous avons défini l'accusé de réception par le caractère "p", ce qui permet d'avoir une sécurité pour éviter d'envoyer constamment la trame. Voici l'algorigramme expliquant ce principe :



Serveur OpenIOB



Préface : Dans cette partie nous allons utiliser une Raspberry PI 4, un écran PITFT 3,5" pour la gestion d'un serveur et d'une IHM tactile et un module LoRa pour la transmission radio sans fil. Le schéma structurel de ce circuit est disponible à l'**annexe 8**.

Qu'est-ce qu'OpenIOB ?

Illustration 5: Logo d'OpenIOB v1.0

OpenIOB est un logiciel embarqué et développé sous **Python** et **Javascript**. Il est compatible avec Windows et linux (pour l'affichage), d'où l'intérêt de l'embarquer sur une Raspberry PI. Ce logiciel est composé de 3 parties :

→Une partie de **gestion d'une IHM** permettant d'avoir une interface pour visualiser en temps réel les valeurs obtenues, et paramétrer l'environnement du système (Combien de ruches peuvent communiquer avec le serveur ? L'adresse LoRa du serveur, synchroniser les ruches).

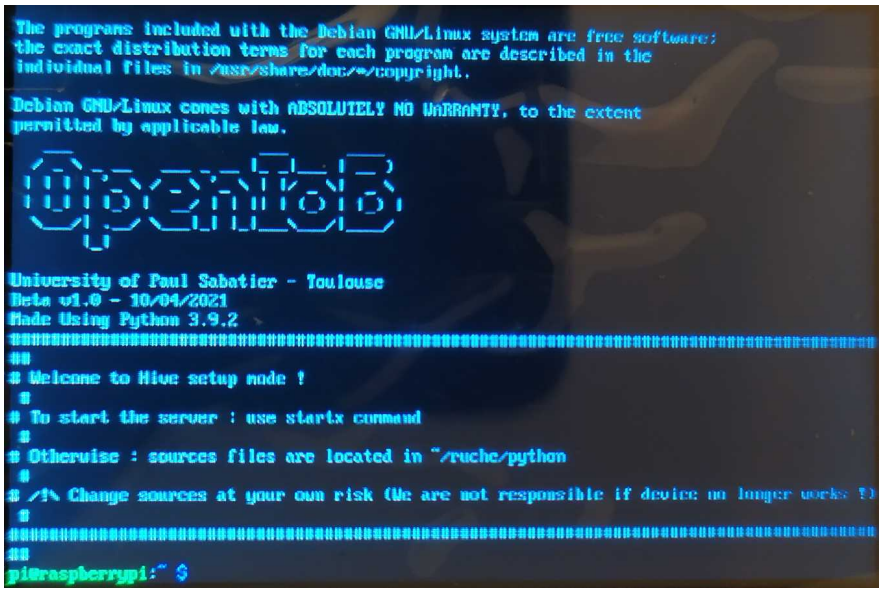
→Une partie de **gestion d'une page Web**, permettant seulement de consulter l'historique des valeurs obtenues sur différentes ruches, depuis un autre poste, ou smartphone connecté au même réseau que le serveur.

→Une partie de **gestion d'une communication LoRa** avec les ruches embarquées. Actuellement cette partie n'est seulement compatible qu'avec la Raspberry PI.

Intégration et fonctionnement dans la Raspberry PI

OpenIOB est installé dans le système d'exploitation **Raspberry PI OS**, qui est un système d'exploitation basé sur la distribution **Debian Linux**. Le système entier est contenu dans une carte µSD de **16 Go**.

Lorsque la Raspberry PI démarre, le logiciel OpenIOB n'est pas executé automatiquement, en effet le serveur reste dans la console ou l'utilisateur peut éventuellement tenter de réparer le système, dans le cas ou OpenIOB aurait un problème technique.

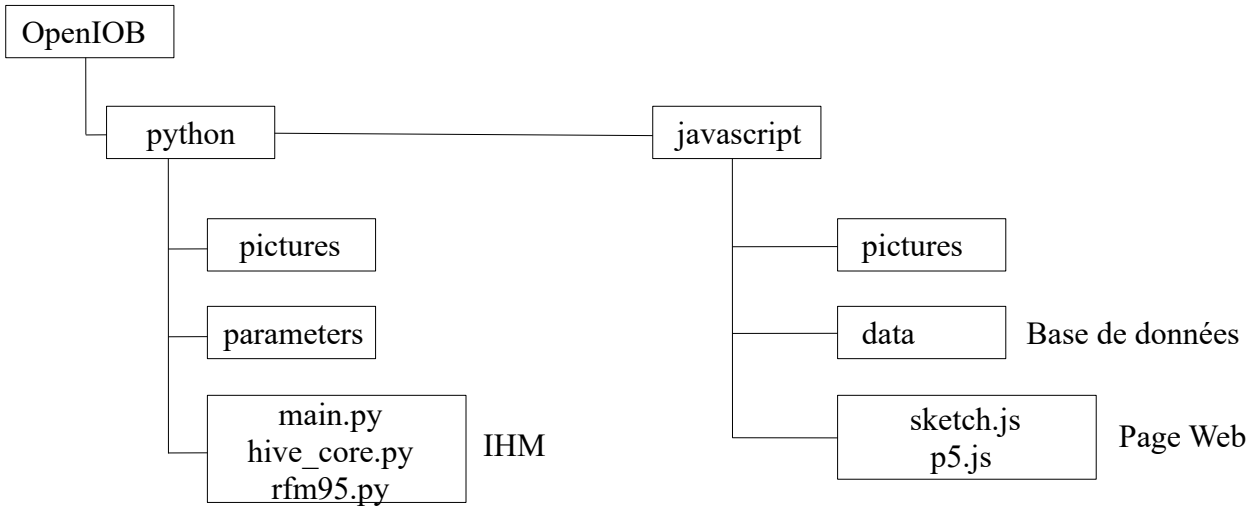


Pour lancer OpenIOB, il suffit d'entrer la commande **startx**. Elle permettra de passer le système d'exploitation en mode graphique, et de lancer les commandes suivantes :

→ **sudo python3 main.py** #Lance l'IHM graphique
→ **sudo python3 rfm95.py** #Lance le serveur Radio LoRa
→ **sudo python3 -m http.server --directory js** #Lance le serveur Web

Remarque : L'utilisateur prend connaissance de l'adresse IP du serveur à l'aide de la commande **ifconfig**.

L'arboresence des éléments essentiels d'OpenIOB est décrite ci-dessous :



Gestion de l'IHM

Présentation générale

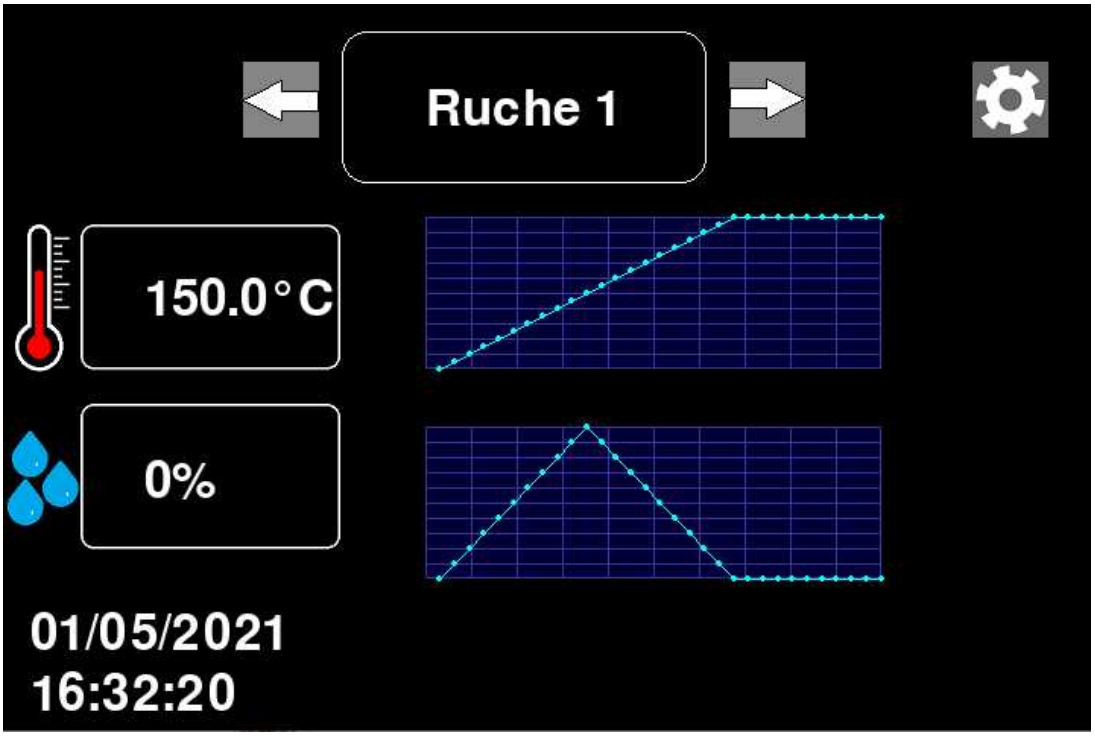


Illustration 6: IHM graphique généré par le fichier main.py

Lorsque le logiciel est lancé, nous pouvons observer l'affichage ci-dessus présent sur l'écran tactile. Sur cette page, nous remarquons l'historique des valeurs obtenues de la ruche et le mois concerné. Il est à noter que les valeurs affichées dans les encadrements blancs à gauche, sont les dernières valeurs de leurs historiques respectifs. Ainsi, le logiciel propose d'afficher la date et l'heure en temps réel.

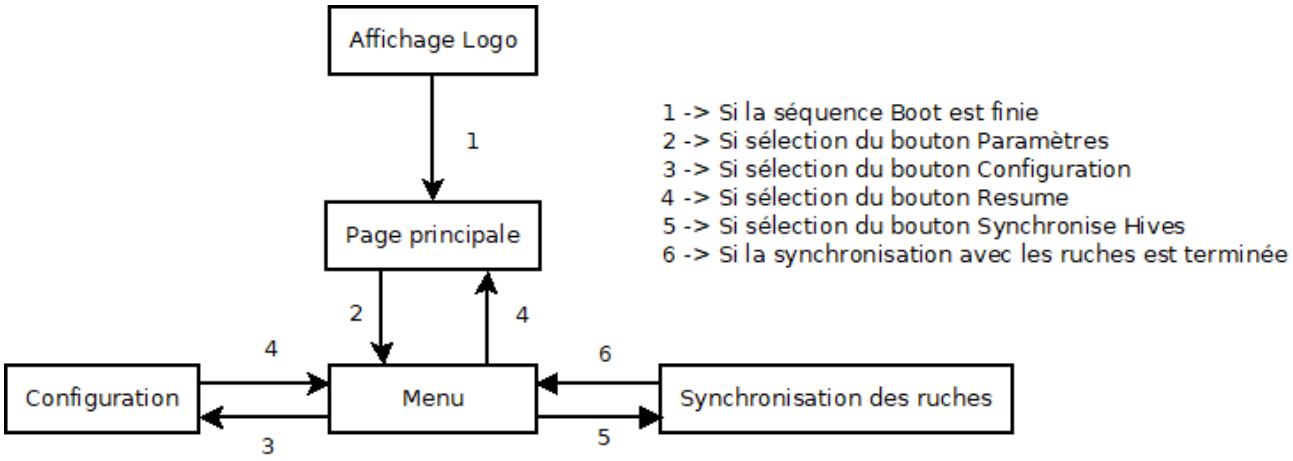
A l'aide des boutons gauche et droite, l'utilisateur peut naviguer sur les différents historiques de ses différentes ruches. Le bouton de la roue crantée à droite permet d'accéder au menu disponible à l'**annexe 9**.

Ainsi, le menu propose :

→ De retourner à la page principale
→ D'accéder aux paramètres (permettant de modifier le nombre de ruches souhaitées et l'adresse du serveur)
→ De synchroniser les ruches (**Fonctionnalitée non implémentée**)
→ De quitter le programme

Navigation des pages

Afin de définir les différentes pages, nous avons défini le diagramme suivant :



Éléments graphiques

Tout les éléments visuels sont fait de routines graphiques, ou d'images faites, disponibles dans le dossier **python/pic**. Il est à noter que le placement de ces élements dans le fenêtrage, relève du développement d'une **interface graphique**. Cela consiste à placer un objet dans les coordonnées X et Y défini comme l'exemple ci-dessous :

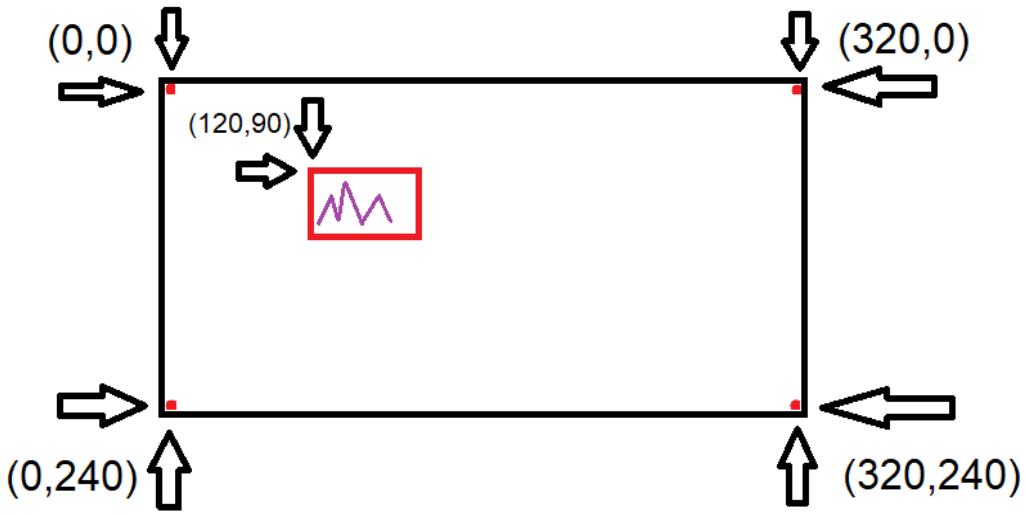


Illustration 7: Principe d'un placement d'élément dans une fenêtre

Principe d'un bouton graphique (Back-end)

Il est à noter que chaque élément a ses spécificités, comme la gestion d'un bouton pour interagir avec l'interface.

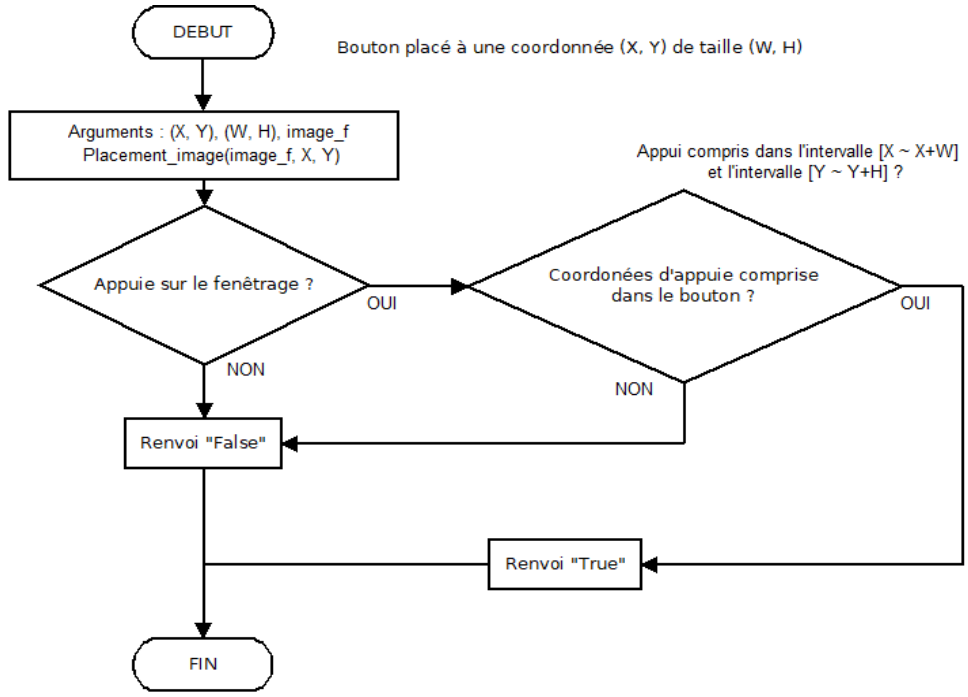


Illustration 8: Algorigramme du fonctionnement des boutons dans l'application python

Gestion de la base de données

Aspect général

Pour lire et interagir avec l'historique, le système a besoin de gérer une base de données commune à **l'application python et la page Web**. Elle est lue par **l'IHM** (**main.py**) et la **page Web**, et écrite par le **serveur Radio LoRa** (**rfm95.py**)

L'architecture des fichiers est défini par une liste de fichiers textes nommés "**YaaaaMmm.txt**", avec **aaaa** l'année correspondante et **mm** le mois correspondant (Exemple : Relevé du mois d'avril en 2021 → **Y2021M04.txt**). Chacun de ces fichiers comportent une suite de trames contenant :

→ La ruche concernée au format entier
→ La date à laquelle la trame a été enregistrée, au format flottant → **dd , hhhhhhhhhh**
avec **d** le jour du mois et **h** l'heure
→ La température au format flottant
→ L'humidité au format flottant

Voici un exemple de trame de la base de données au fichier "**Y2021M03.txt**" :

R02 D1.2390907 T24.2 H71.5

Ruche : 2
Date : Jour 1 à 5h44
Donées : 24,20 °C - 71,5%

Il est à noter que pour obtenir la date et l'heure depuis la trame, il faut effectuer la procédure suivante :

→ La partie entière sera toujours le jour actuel, il faut donc mettre la partie décimale de coté.

→ La partie décimale est une simple conversion de l'intervalle [0h ~ 24h] en [0,0 ~ 1,0].

Nous pouvons donc démontrer par les calculs suivants :

$$Heure = 0{,}2390907 * 24 = 5{,}7381768 \rightarrow 5\,h$$
$$Minutes = 0{,}7381768 * 60 = 44{,}290608 \rightarrow 44\,m$$

Donc la date complète est **01/03/2021 - 5 : 44**

Écriture de la base de données

Comme nous l'avions explicité (**p.26**), seul le sous programme **rfm95.py** intervient dans l'écriture de la base de données s'il reçoit une trame correcte. La communication avec la ruche embarquée est différente sur le serveur, car contrairement à l'Arduino Uno (**voir le fonctionnement de la communication p.20**), la Raspberry PI ne renvoie que l'octet **112** en guise d'accusé de réception, comme nous l'avons défini dans la trame prototype.
Le fonctionnement explicité, est décrit dans l'**annexe 10**.

Lecture de la base de données

Afin que l'IHM ou la page Web puissent lire les valeurs qui sont stockées dans la base de données, les trames intégrées dans les fichiers textes sont interprétées à l'aide d'un outil développé dans l'application python ainsi que dans la page Web, appelé **parseur**.

Le rôle du parseur dans cette application, est de prendre les valeurs correspondantes sous forme de chaîne de caractères, afin qu'elles soient converties en valeurs numériques et attribuées à leurs variables respectives. Le traitement se fait caractère par caractère.
Reprenons l'exemple de la trame étudiée afin de démontrer son fonctionnement :

R02 D1.2390907 T24.2 H71.5

Lecture du caractère "R" → Lecture de la chaîne "02" convertion en entier → Attribution à la variable ruche → ruche = 2
Lecture du caractère "D" → Lecture de la chaîne "1,2390907" convertion en flottant → Attribution à la variable days → days = 1,2390907
Lecture du caractère "T" → Lecture de la chaîne "24,2" convertion en flottant → Attribution à la variable temperature → temperature = 24,20
Lecture du caractère "H" → Lecture de la chaîne "71,5" convertion en flottant → Attribution à la variable humidity → humidity = 71,50

Le fonctionnement du parseur est décrit à l'algorigramme disponible à l'**annexe 11**.

Gestion de la page Web

Présentation générale

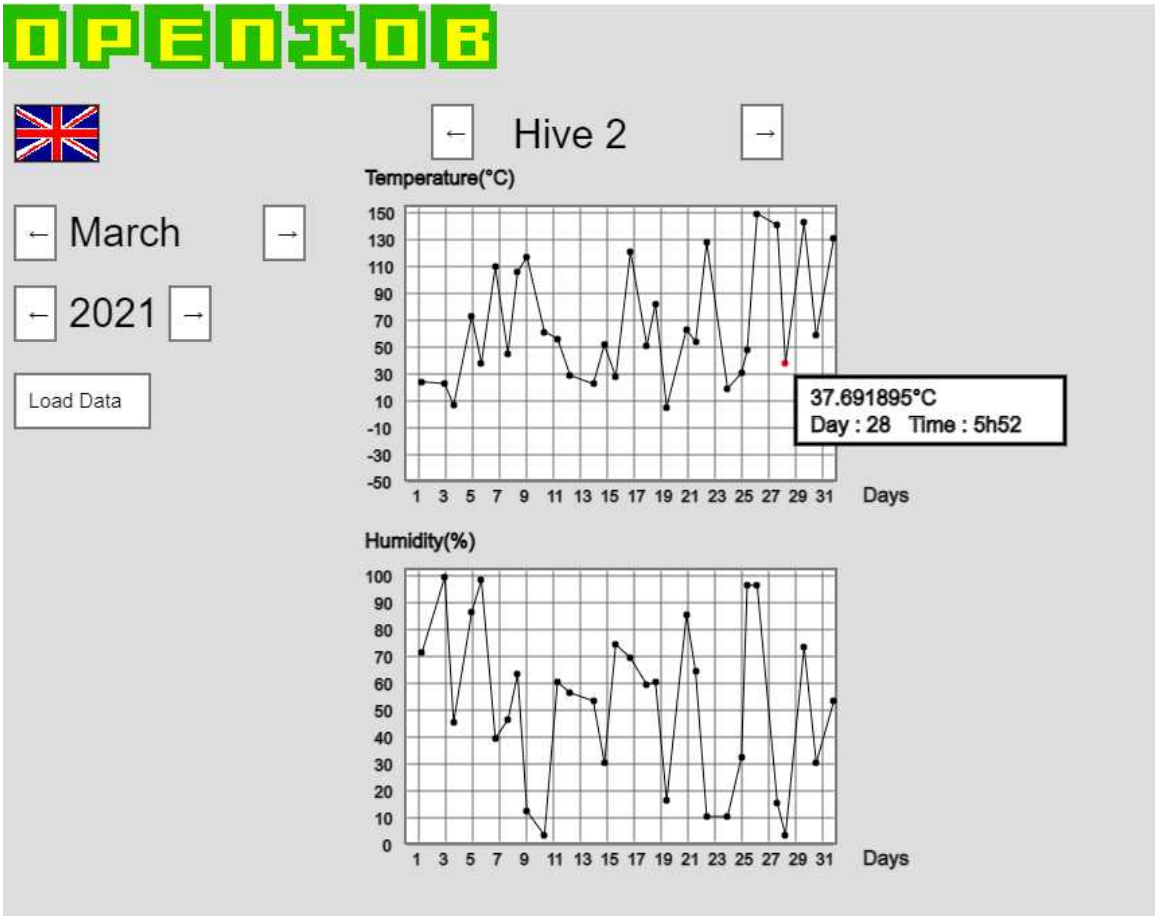


Illustration 9: Page Web du programme OpenIOB

OpenIOB dispose également d'une page Web accessible depuis un poste ou un smartphone. L'accès se fait sur navigateur, où l'adresse IP du serveur et le port **8000** doivent être renseignés. Par exemple, si le serveur avait l'adresse IP **192.168.1.76** d'assignée, il faudrait renseigner au navigateur : **192.168.1.76:8000.**

La page Web ne permet que de consulter les valeurs des historiques de chaque ruches avec plus de précision. En effet, le paramétrage du serveur ne se fait que sur l'application python.

Il est à noter que le développement de la page Web a été fait en **Javascript**, à l'aide de **Processing.js**, ou plutôt **p5.js.** Il relève de la même technique de programmation que l'interface graphique de l'IHM (**p.25**).

Remarque : 7 langues sont disponibles sur la page Web, à savoir : Anglais, Français, Espagnol, Italien, Chinois, Japonais, Russe.

Développement du graphe (Back-end)

Le graphe a pour but de récupérer les chaînes de données acquises à l'aide du parseur (**explicité p.27**), et de les afficher en respectant l'échelle.

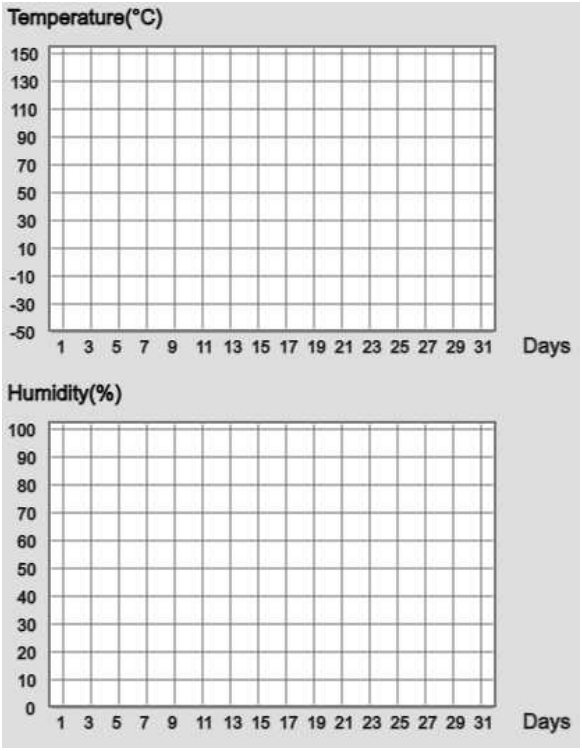


Illustration 10: Graphes de température et d'humidité

L'affichage d'un graphe se fait en 5 étapes :

→Il affiche d'abord le fond du graphe (Rectangle blanc ici)
→Il affiche le cadriage
→Il regarde qu'elle chaîne de données doit être affichée (Température ou Humidité), afin de définir son échelle
→Il affiche les échelles
→Il trace les lignes et les points en fonction de l'échelle définie

Une autre fonctionnalité a été ajoutée sur la page Web, que l'IHM ne possède pas, c'est le pointage de valeur.

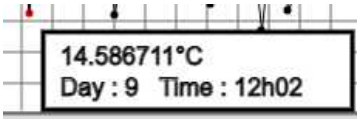


Illustration 11: Pointage d'une valeur de l'historique de température

Si la coordonnée de la souris du poste de travail, ou la coordonnée d'appuie sur le smartphone, est comprise dans l'intervalle ajustée dans le programme par rapport au point d'un relevé de l'historique, la page Web affichera sa valeur avec sa date associée.

L'algorigramme du graphe est disponible à l'**annexe 11**.

Architecture réseau

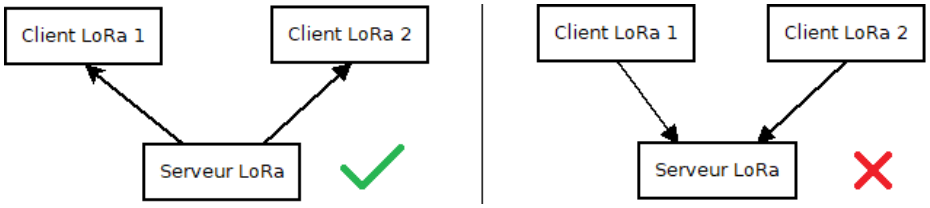
Nous avons démontré que OpenIOB propose une connectivité entre la ruche embarquée et le serveur à l'aide des modules LoRa et un accès d'une page Web à travers le réseau IP.

Réseau radio LoRa

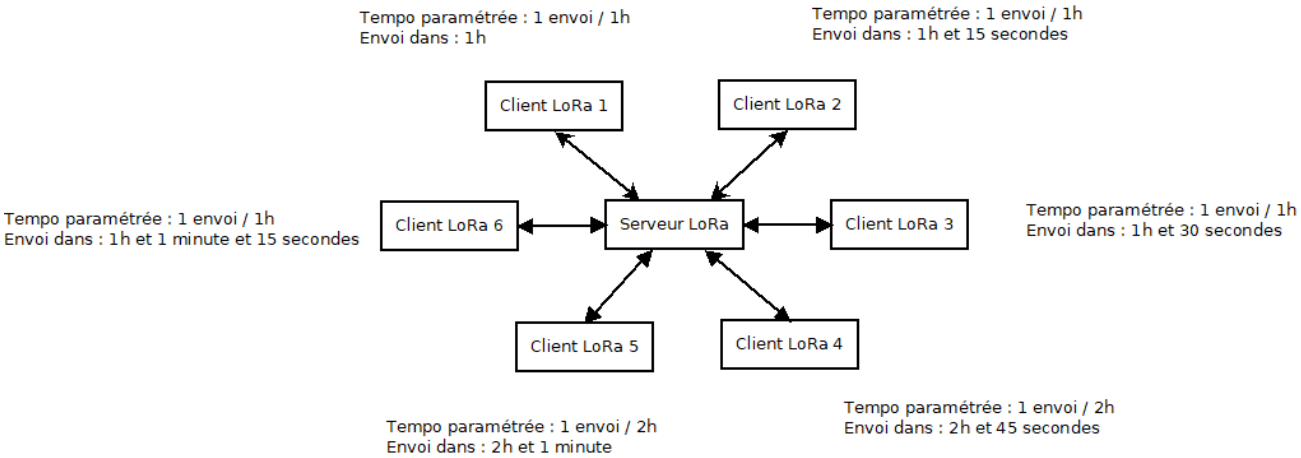
Le sous programme **rfm95.py** dans la Raspberry PI, permet de gérer les communications entre le serveur et les ruches. Comme nous n'avions pas eu le temps d'implémenter la synchronisation des ruches, nous allons expliquer le fonctionnement et son utilitée.

Pendant l'étude des modules RFM95, nous avions constatés que le serveur pouvait émettre à 2 clients en même temps, mais ne pouvait reçevoir les 2 messages émis par ces clients s'ils venaient à émettre en même temps.

En effet, le serveur ne peut traiter que le premier message qu'il reçoit et négliger le second.



C'est la raison pour laquelle nous voulions implémenter une synchronisation des ruches embarquées. Cela demande au serveur d'envoyer une commande spécifique au client, dans une intervalle de 15 ~ 20 secondes, qui ordonnera un redémarrage du client, de sorte que quand la temporisation est atteinte, il enverra les données au serveur.



Il est à noter que le principe explicité est théorique, un approfondissement est alors nécessaire.

Réseau IP (Local et Internet)

Nous avons explicité (**p.28**) que l'utilisateur peut avoir accès à une page Web grâce à l'adresse IP du serveur au port **8000**. Il est à noter que cette technique cette fait uniquement dans un réseau local. C'est à dire qu'à partir du moment l'utilisateur est connecté à son propre réseau IP, il peut avoir accès au serveur. Dans le cas où l'utilisateur voudrait se connecter à distance, depuis un autre réseau faisant appel à internet, il est question cette fois-ci de faire appel à une **redirection de port** depuis le routeur où est connectée la Raspberry PI.

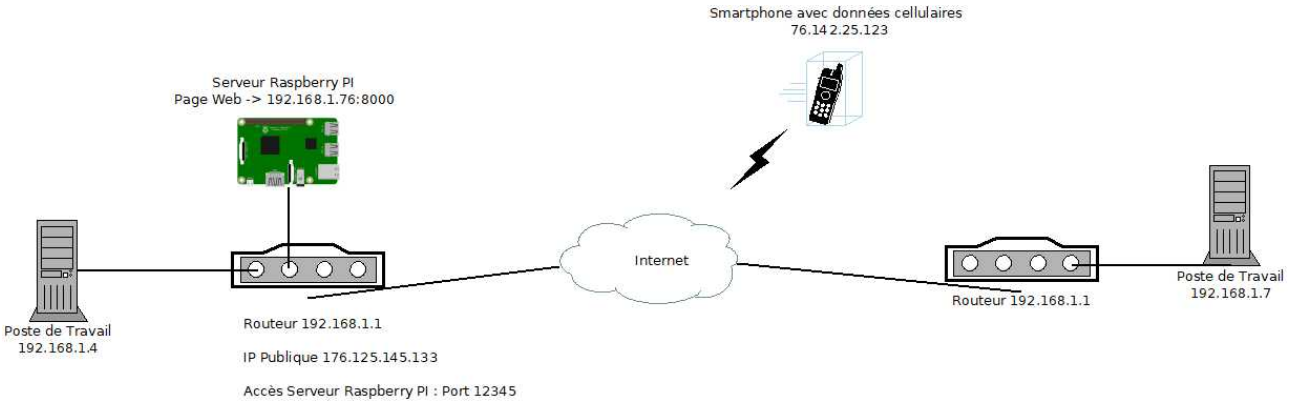


Illustration 12: Exemple d'architecture réseau pour accéder au serveur depuis le réseau local ou depuis internet

En effet, les routeurs disposent de fonctionnalités qui permettent de réaliser des accès depuis d'autres réseaux. Le plus utilisé est le protocole NAT, qui permet de faire la correspondance des **adresses IP privées** aux **IP publiques** sur le routeur même.

D'après l'exemple ci-dessus, nous pouvons avoir accès au serveur depuis un autre réseau en utilisant l'IP publique du routeur et en utilisant le **port externe configuré** pour accéder à la Raspberry PI (ici le port 12345).

Pour résumer : Si l'utilisateur est connecté au même réseau local que le serveur Raspberry PI, il devra se connecter à l'adresse **192.168.1.76:8000**. En revanche, si l'utilisateur est hors de ce réseau local, il devra se connecter à l'adresse **176.125.145.133:12345**.

Conclusion

L'objectif de ce projet était de réaliser un système capable de lire et traiter des données depuis un système embarqué, et de les communiquer à un serveur à l'aide d'une transmission sans fil longue portée. Nous devions avoir une communication possible dans une portée minimale de 10m. Les modules LoRa nous ont démontrés que nous pouvions avoir une portée de **300~400m** dans un champ rempli d'obstacles (Batiments, arbres, ...). Tous nos objectifs principaux ont étés remplis, cependant, nous n'avions pas pu intégrer la fonctionnalité de la synchronisation des ruches, mais il n'en est pas moins que le serveur est capable de communiquer avec plusieurs ruches, à condition qu'elle n'émettent pas une trame en même temps.

Il est à noter que ce projet peut être amélioré et contribué à de nouvelles opportunités dans le développement de nouveaux projets.

Remarque des bînomes :

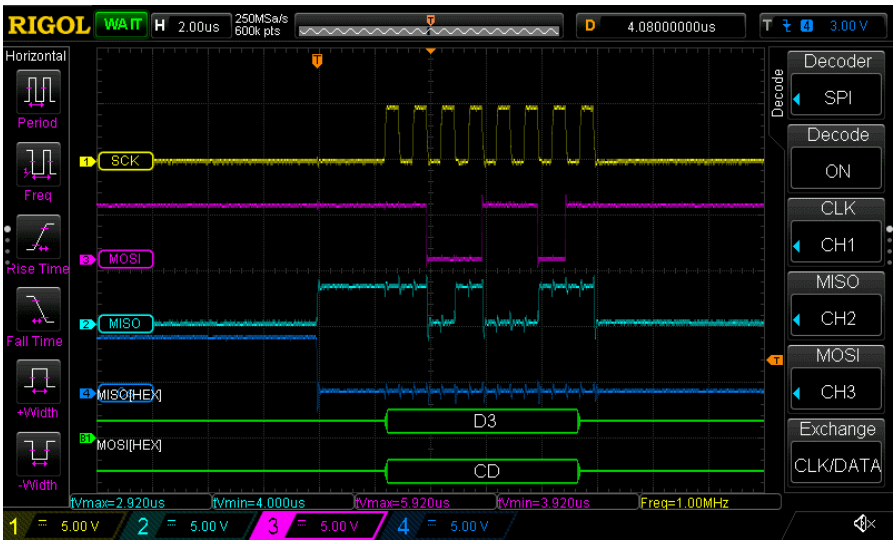
Je suis satisfait du travail que j'ai apporté pour ce projet, ainsi que le travail de ma camarade, car ça m'a permis de démarrer le javascript, d'approfondir le python et l'environnement Debian Linux. Le développement d'un réseau radio LoRa et d'une page Web m'a ouvert les yeux pour développer certains projets personnels. Je repars avec de nouvelles connaisances.

Enzo Niro

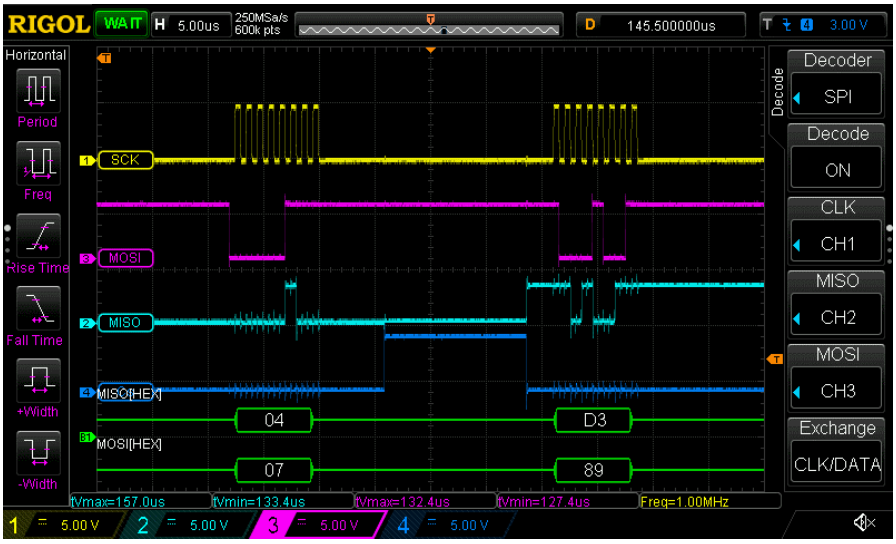
N'ayant jamais fais, voir peu d'électronique et de programmation sur une carte Arduino, ce projet m'a donc permis d'acquérir de nouvelles compétences. En effet, aujourd'hui je suis capable de programmer un écran LCD, qui permet de visualiser différentes données, d'utiliser une EEPROM, de réaliser une sécurité d'accès aux paramètres, mais également connaître la transmission LoRa. Je tiens tout particulièrement à remercier mon binôme, qui a su me donner des conseils et des explications sur des tâches qui m'étaient difficiles de comprendre ou de réaliser.
Ce projet a été enrichissant en tous points !

Laetitia Bayle

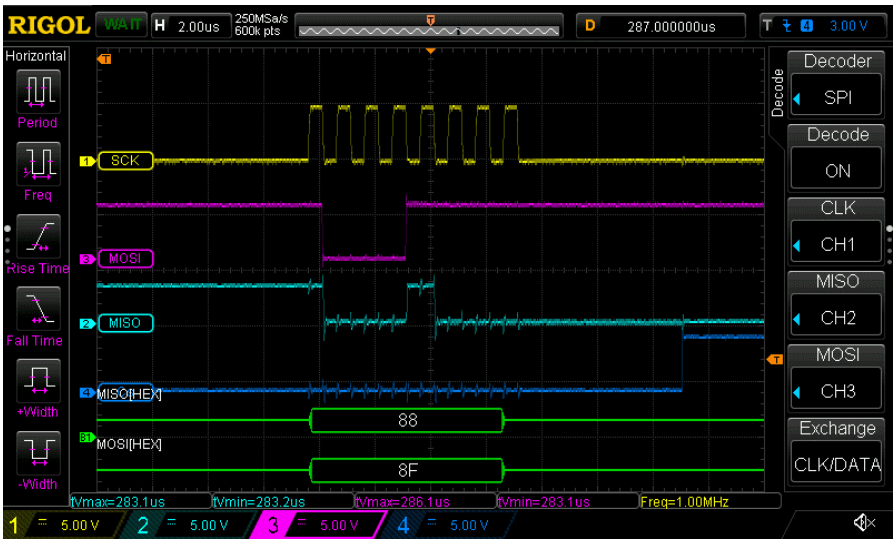
Annexes



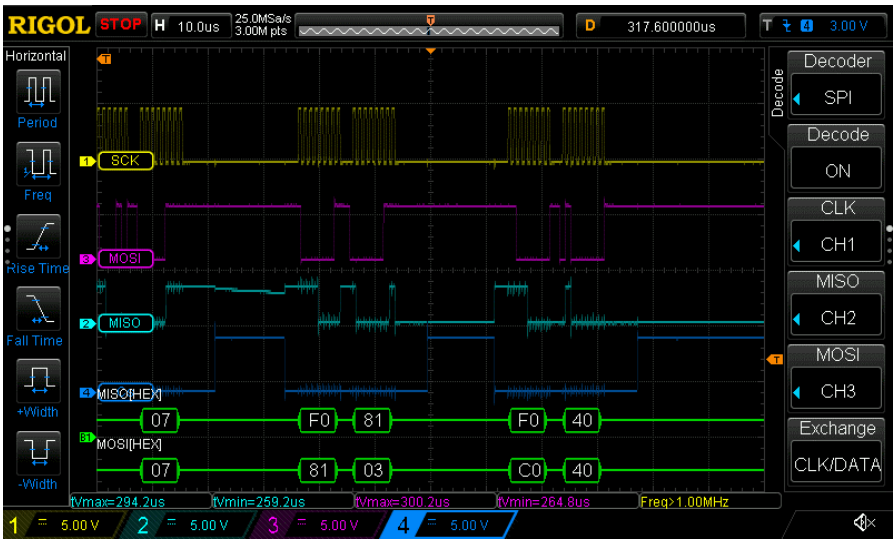
Annexe 1: 1er octet de la 1ère consigne : 0xCD



Annexe 2: 2ème octet de la 1ère consigne : 0x07
1er octet de la seconde consigne : 0x89



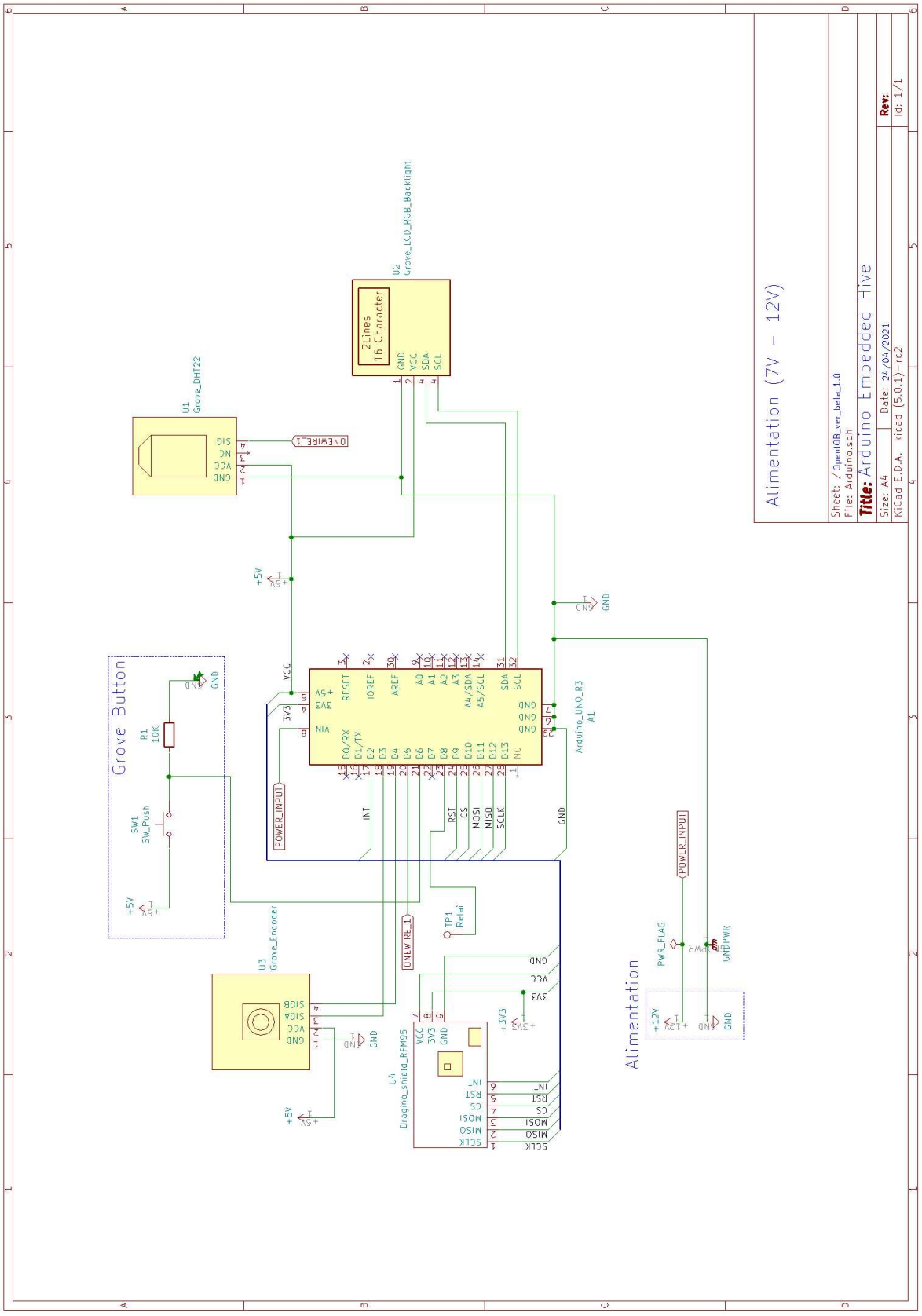
Annexe 3: 2ème octet de la 2ème consigne 0x8F

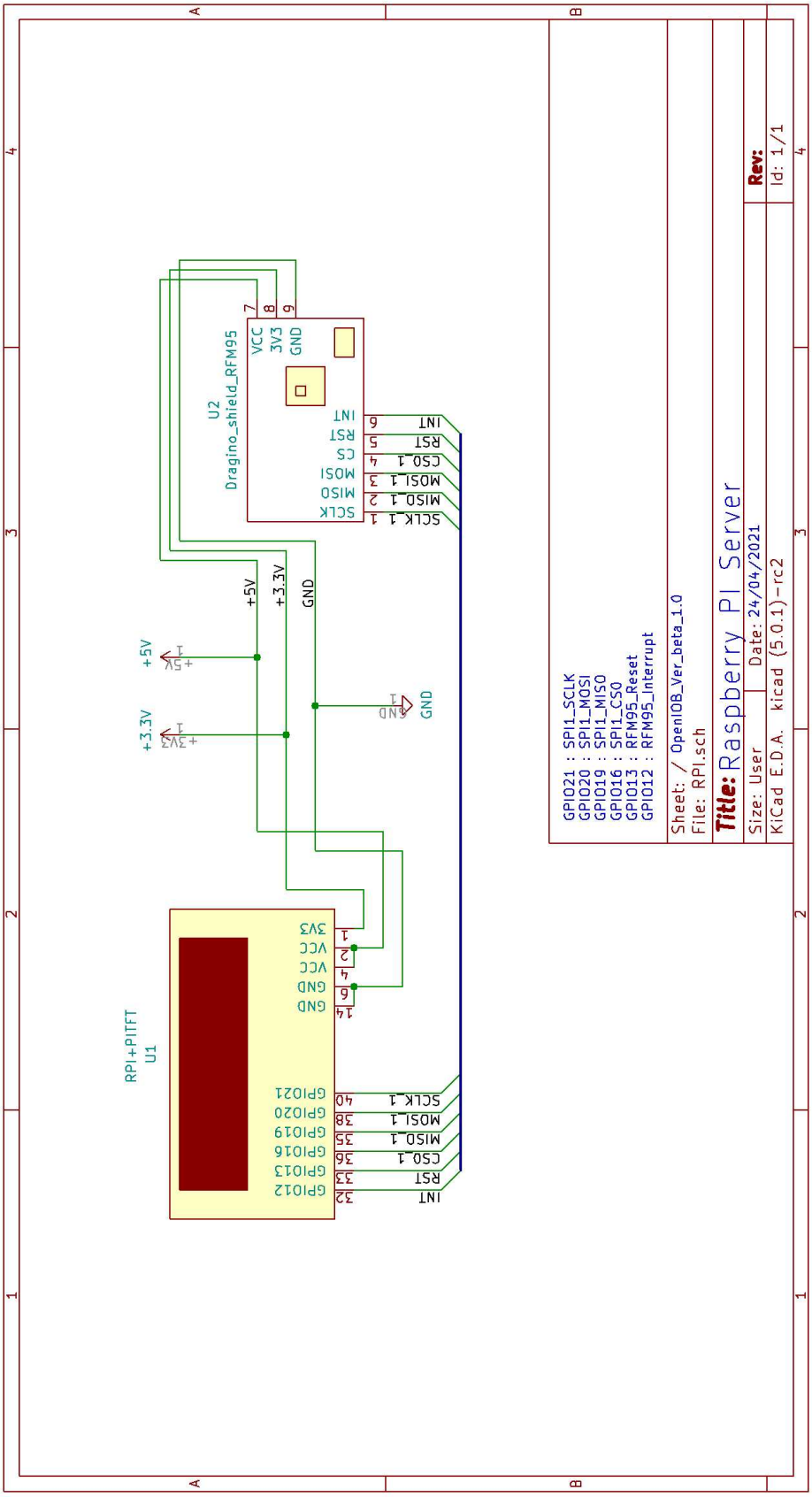


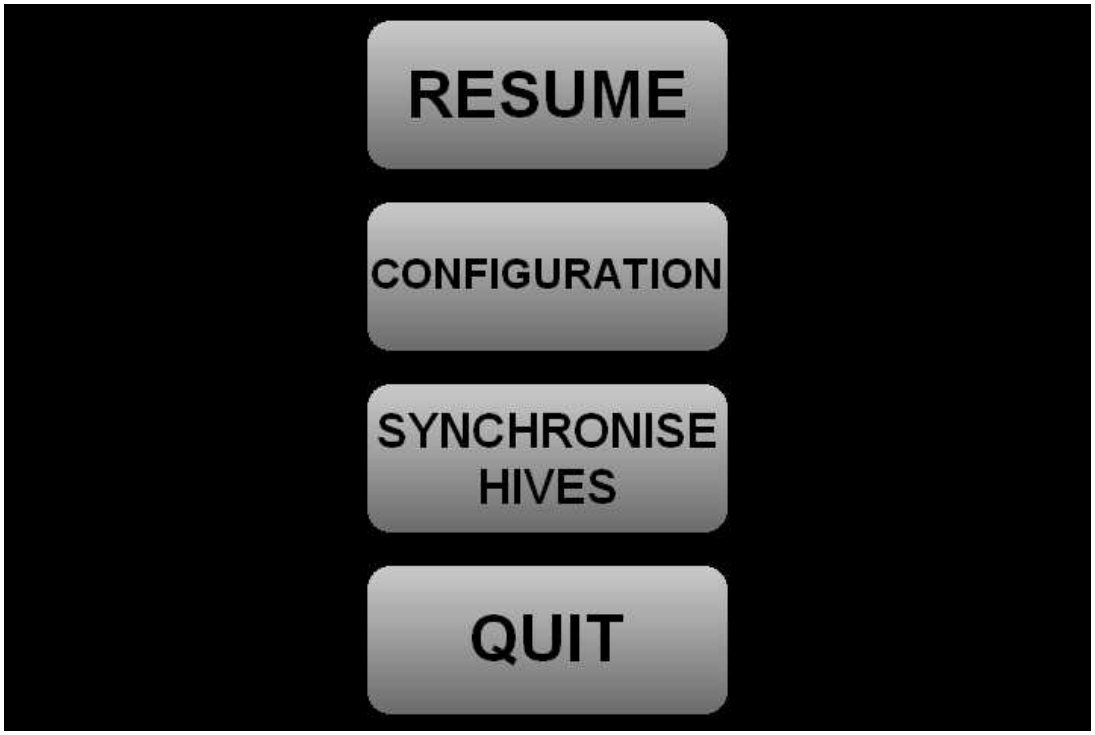
Annexe 4: Commande de validation de l'envoi de la trame "eea", suivie d'une consigne d'un signal d'interruption à la broche DIO0

Annexe 5 : Démonstration et benchmark entre une Raspberry PI 4 modèle B 4 Go Ram et une Raspberry PI 3 modèle B, sur l'application OpenIOB sous le système d'exploitation Raspberry PI OS.
https://www.youtube.com/watch?v=dn1IfX4r9YU

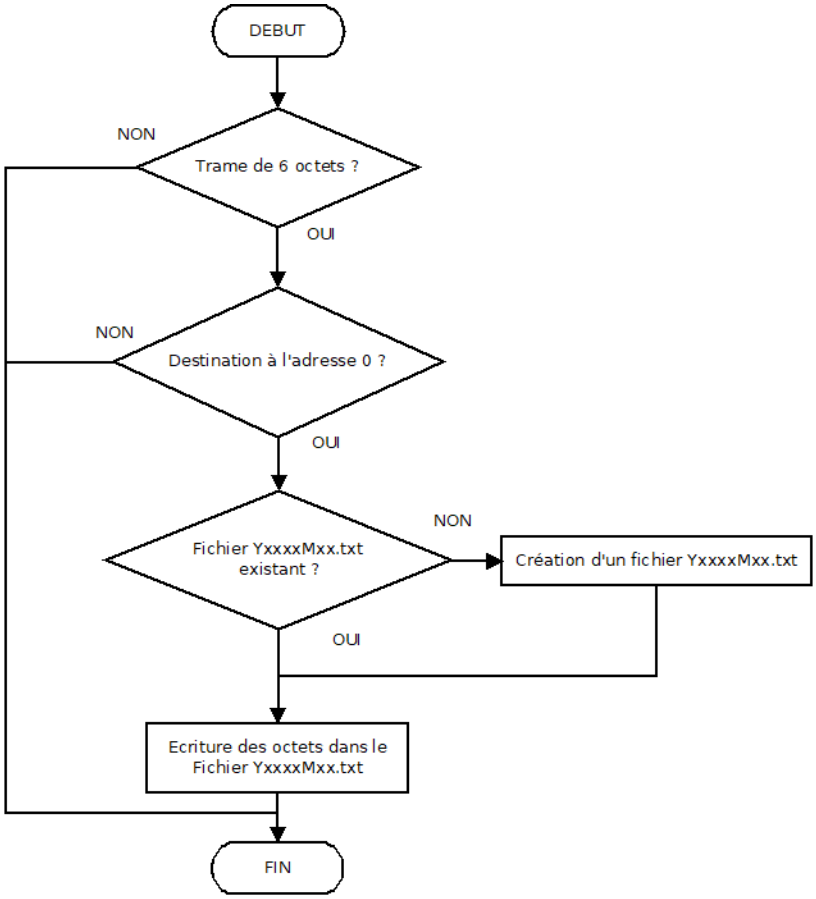
Annexe 6: Fonctionnement de la session (Demande de mot de passe et présentation du cas si le mot de passe est bon ou mauvais)


Alimentation (7V - 12V)
Sheet: /OpenIOB_ver_beta_1.0
File: Arduino.sch
Title: Arduino Embedded Hive
Size: A4
Date: 24/04/2021
KiCad E.D.A. kicad (5.0.1)-rc2
Rev:
Id: 1/1
U2 Grove_LCD_RGB_Backlight
2Lines 16 Character
GND VCC SDA SCL
U1 Grove_DHT22
SIG NC VCC GND
ONEWIRE_1
+5V
GND
VCC
Grove Button
R1 10K
SW1 SW_Push
Arduino_UNO_R3
A1
POWER_INPUT
INT
RST
CS
MOSI
MISO
SCLK
TP1 Relai
U3 Grove_Encoder
SIGB SIGA VCC GND
U4 Dragino_shield_RFM95
3V3
Alimentation
PWR_FLAG
+12V

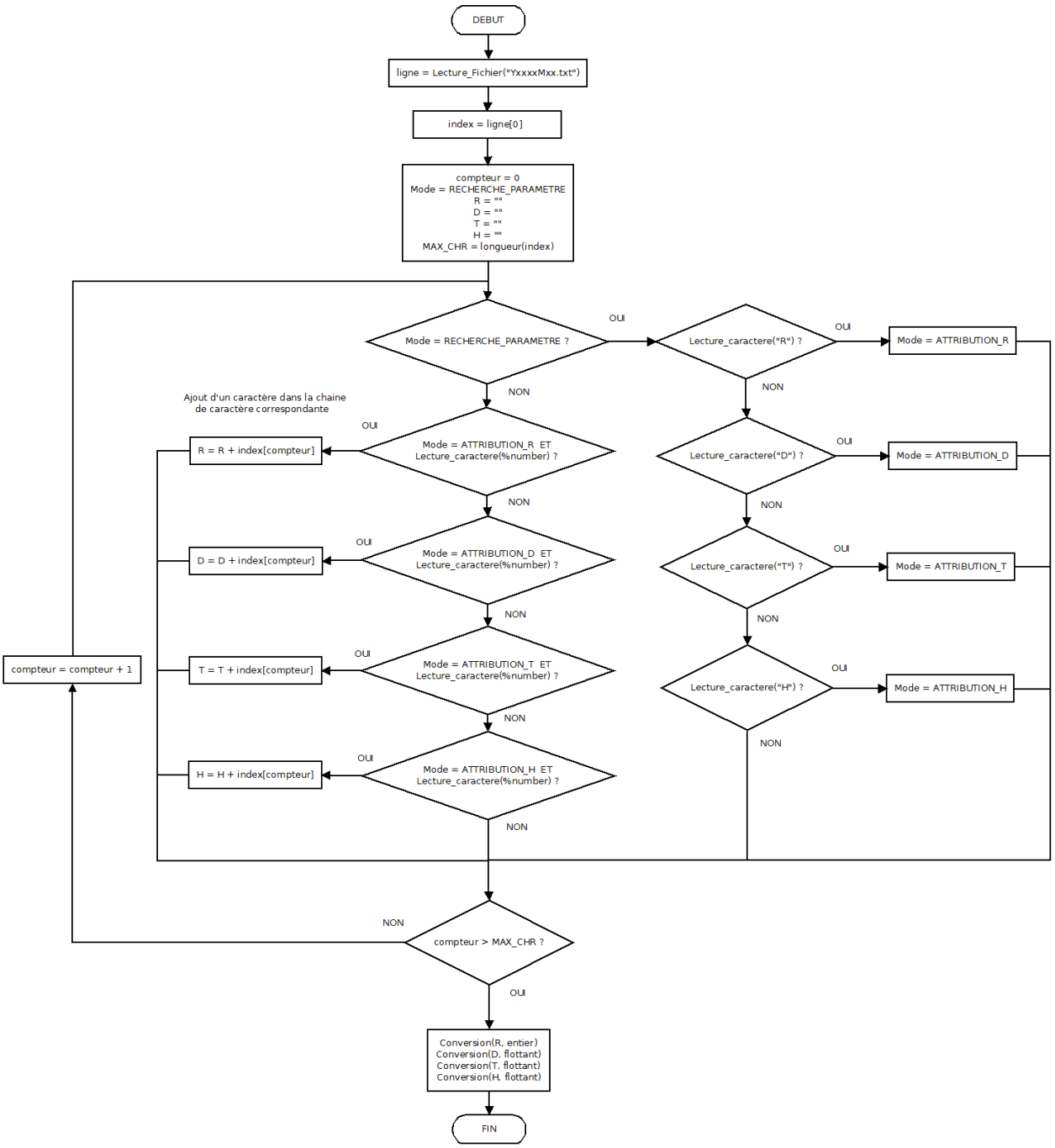

U2
Dragino_shield_RFM95
VCC
3V3
GND
INT
RST
CS
MOSI
MISO
SCLK
RPI+PITFT
U1
3V3
GND
GPIO21
GPIO20
GPIO19
GPIO16
GPIO13
GPIO12
+5V
+3.3V
GND
GPIO21 : SPI1_SCLK
GPIO20 : SPI1_MOSI
GPIO19 : SPI1_MISO
GPIO16 : SPI1_CS0
GPIO13 : RFM95_Reset
GPIO12 : RFM95_Interrupt
Sheet: / OpenIOB_Ver_beta_1.0
File: RPI.sch
Title: Raspberry PI Server
Size: User
Date: 24/04/2021
KiCad E.D.A. kicad (5.0.1)-rc2
Rev:
Id: 1/1



Annexe 9: Menu du logiciel OpenIOB

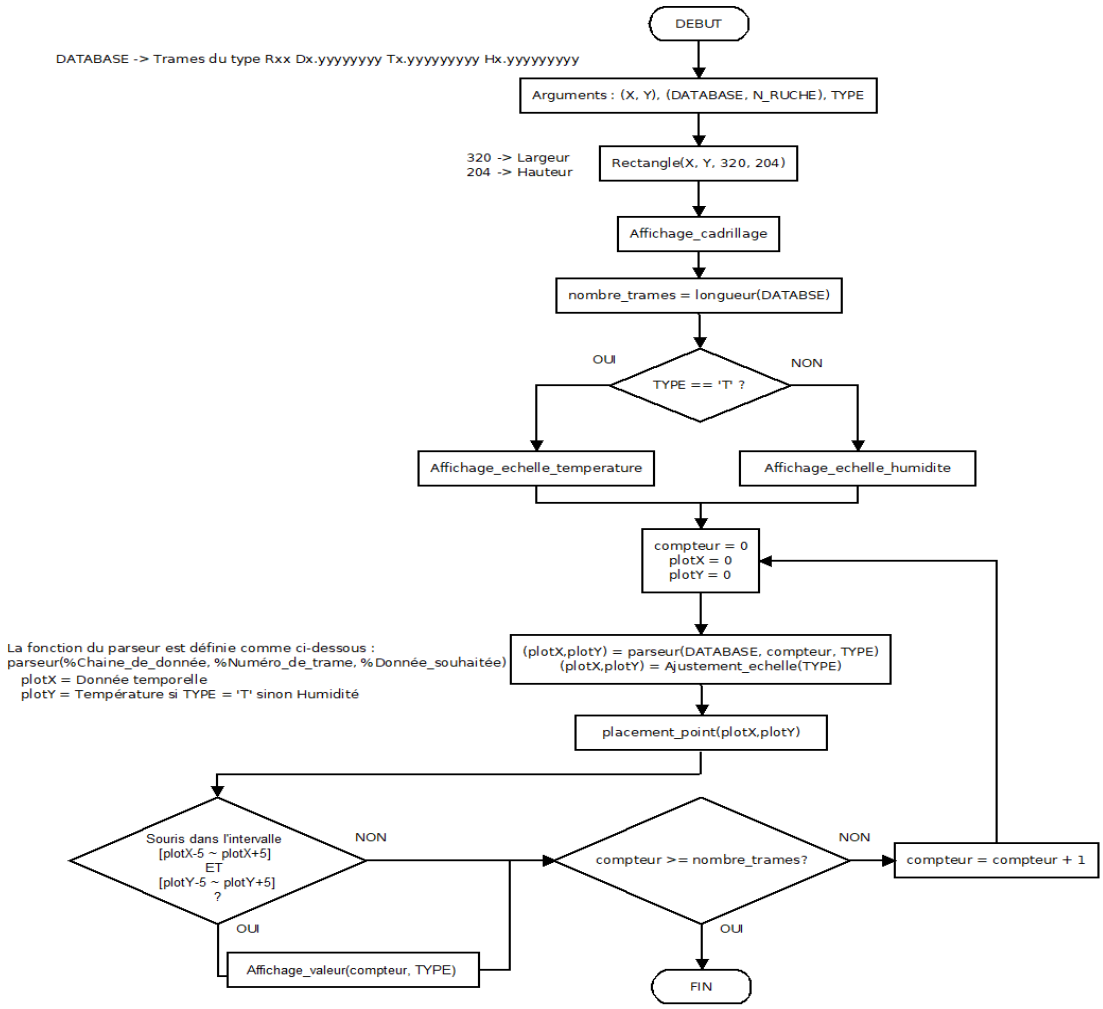


Annexe 10: Algorigramme de l'écriture dans la base de donnée



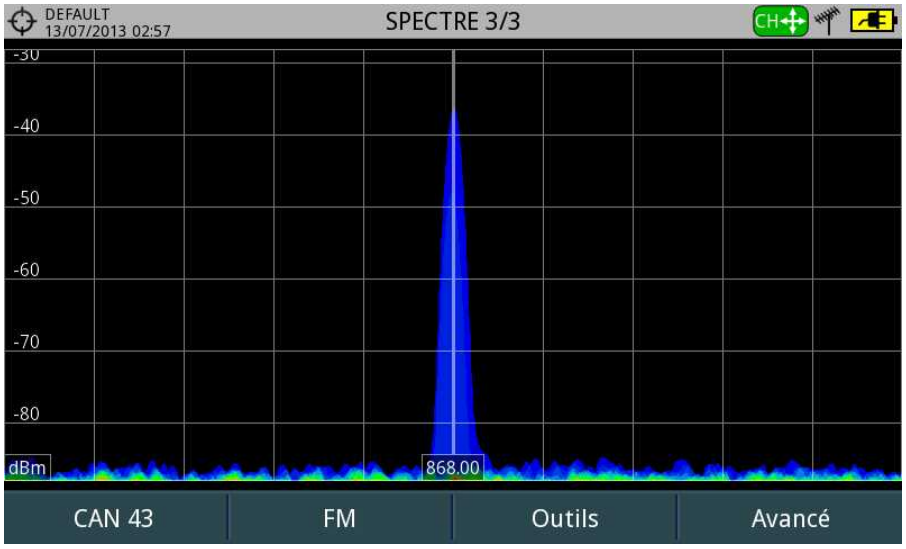
Annexe 11: Algorigramme du parseur

Remarque : "**%number**" vérifie si le caractère pointé correspond à un de ces caractères :
"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "."



Annexe 11: Algorigramme d'un graphe de la page Web

Annexe 12: Test d'une portée de communication entre 2 modules LoRa à l'Université Paul Sabatier. (1 situé à la Salle G45 du batiment 3A - 1 situé à coté du Batiment des Gardiens)



Annexe 13: Relevé spectral d'une transmission d'un module LoRa vers un autre

N°		Nom de la tâche	Durée	Début	Fin	Noms ressources
1		Test du matériel	26 heures	Lun 15/02/21	Mar 16/02/21	BAYLE Laetitia;NIRO Enzo
2	✓	test de l'Arduino	1 heure	Lun 15/02/21	Lun 15/02/21	BAYLE Laetitia
3	✓	test du DHT22	7 heures	Lun 15/02/21	Lun 15/02/21	BAYLE Laetitia
4	✓	écran LCD	7 heures	Mar 16/02/21	Mar 16/02/21	BAYLE Laetitia
5	✓	test module LoRa	10 heures	Lun 15/02/21	Mar 16/02/21	NIRO Enzo
6	✓	test de la Raspberry + écran	1 heure	Mar 16/02/21	Mar 16/02/21	NIRO Enzo
7	✓	Gestion de projet	10 heures	Lun 22/02/21	Dim 07/03/21	BAYLE Laetitia;NIRO Enzo
8	✓	planning	7 heures	Lun 22/02/21	Lun 22/02/21	BAYLE Laetitia;NIRO Enzo
9	✓	cahier des charges	1 heure	Dim 07/03/21	Dim 07/03/21	BAYLE Laetitia;NIRO Enzo
10	✓	analyse fonctionnelle	2 heures	Dim 07/03/21	Dim 07/03/21	BAYLE Laetitia;NIRO Enzo
11		Réalisation du projet	62 heures	Lun 08/03/21	Lun 12/04/21	BAYLE Laetitia;NIRO Enzo
12		Développement de la Ruche embarquée	62 heures	Lun 08/03/21	Lun 12/04/21	BAYLE Laetitia
13	✓	Programmation pour lire une trame du DHT22 +LCD	7 heures	Lun 08/03/21	Lun 08/03/21	BAYLE Laetitia
14	✓	Programmation pour prendre en main l'IHM	26 heures	Lun 08/03/21	Lun 12/04/21	BAYLE Laetitia
15	✓	Sécurité à l'accès des paramètres de la ruche	14 heures	Lun 15/03/21	Lun 22/03/21	BAYLE Laetitia
16	✓	Sauvegarde des paramètres à l'aide de l'EEPROM	10 heures	Lun 29/03/21	Lun 05/04/21	BAYLE Laetitia
17	✓	Programmation module LoRa	5 heures	Lun 05/04/21	Lun 12/04/21	BAYLE Laetitia
18		Développement du serveur OpenIOB	58 heures	Sam 06/03/21	Mer 14/04/21	NIRO Enzo
19	✓	Développement de l'IHM	24 heures	Sam 06/03/21	Lun 15/03/21	NIRO Enzo
20	✓	Gestion affichage	20 heures	Sam 06/03/21	Lun 15/03/21	NIRO Enzo
21	✓	développement du parseur	4 heures	Sam 06/03/21	Sam 06/03/21	NIRO Enzo
22	✓	Développement de la page Web	20 heures	Dim 14/03/21	Lun 22/03/21	NIRO Enzo
23	✓	Gestion d'affichage	16 heures	Dim 14/03/21	Lun 22/03/21	NIRO Enzo
24	✓	développement du parseur	4 heures	Dim 14/03/21	Dim 14/03/21	NIRO Enzo
25		Gestion et développement du serveur	14 heures	Dim 14/03/21	Mer 14/04/21	NIRO Enzo
26	✓	Développement serveur radio LoRa	13 heures	Lun 12/04/21	Mer 14/04/21	NIRO Enzo
27	✓	Mise en place d'un serveur Linux	1 heure	Dim 14/03/21	Dim 14/03/21	NIRO Enzo
28		Mise en oeuvre du projet	20 heures	Lun 12/04/21	Mer 14/04/21	BAYLE Laetitia;NIRO Enzo
29		Vérification de la communication entre l'Arduino et la Raspberry	7 heures	Lun 12/04/21	Lun 12/04/21	BAYLE Laetitia;NIRO Enzo
30		Vérification de l'intégration de la trame dans une base de données	13 heures	Lun 12/04/21	Mer 14/04/21	BAYLE Laetitia;NIRO Enzo

Annexe 14: Diagramme de Gantt du projet

Annexe 15 : Lien du projet complet → https://github.com/protongamer/L3REL_RUCHE

Bibliographie :

Modulation LoRa :

http://genelaix.free.fr/IMG/pdf/presentation_lora_lorawan.pdf

https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf

Capteur de température DHT22 :

https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf

Caractéristiques Raspberry PI :
https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/

Outils / IDEs :

Arduino : https://www.arduino.cc/

Processing : https://processing.org/

p5.js : https://p5js.org/

Python : https://www.python.org/