projet LoRa Modbus

Lycée Maupertuis
1 rue Pierre de Coubertin
35400 Saint-Malo

Valentin Allouët
Étudiant 2

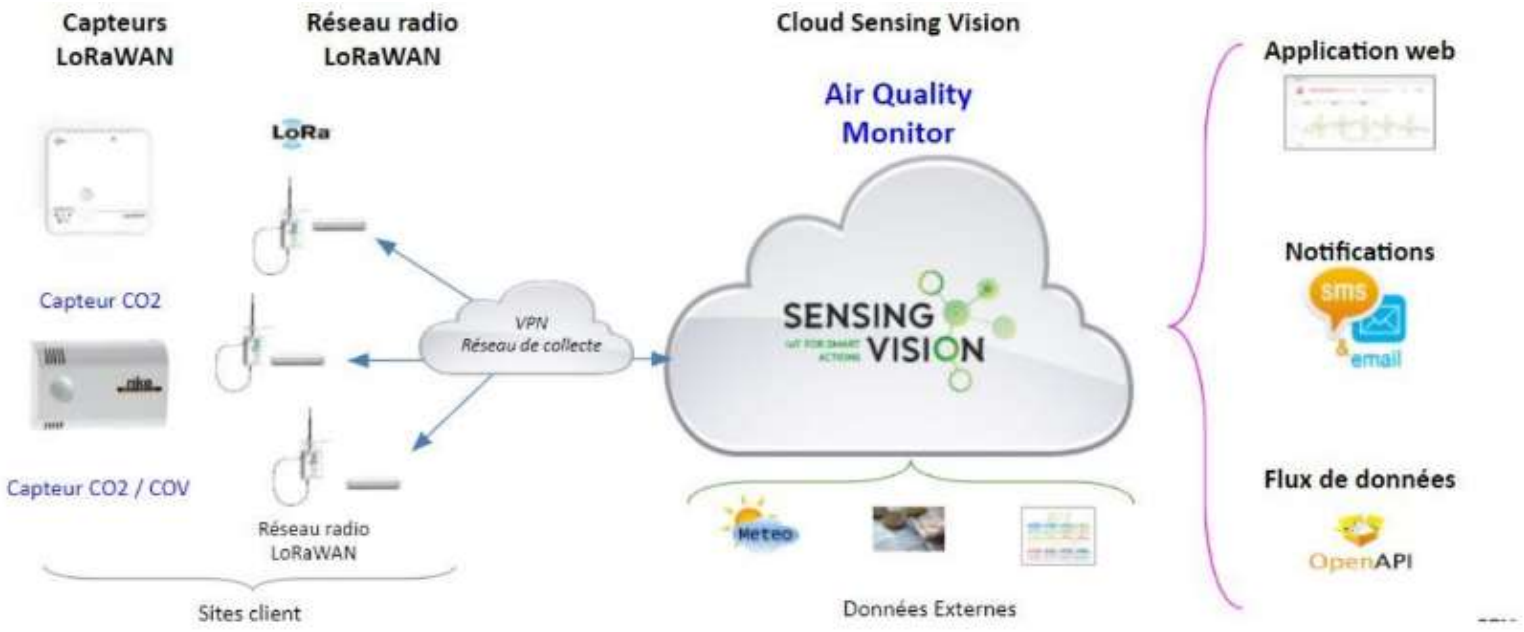
Léo Gueguen
Étudiant 1

Samuel Stekke
Étudiant 3

2023

Sommaire

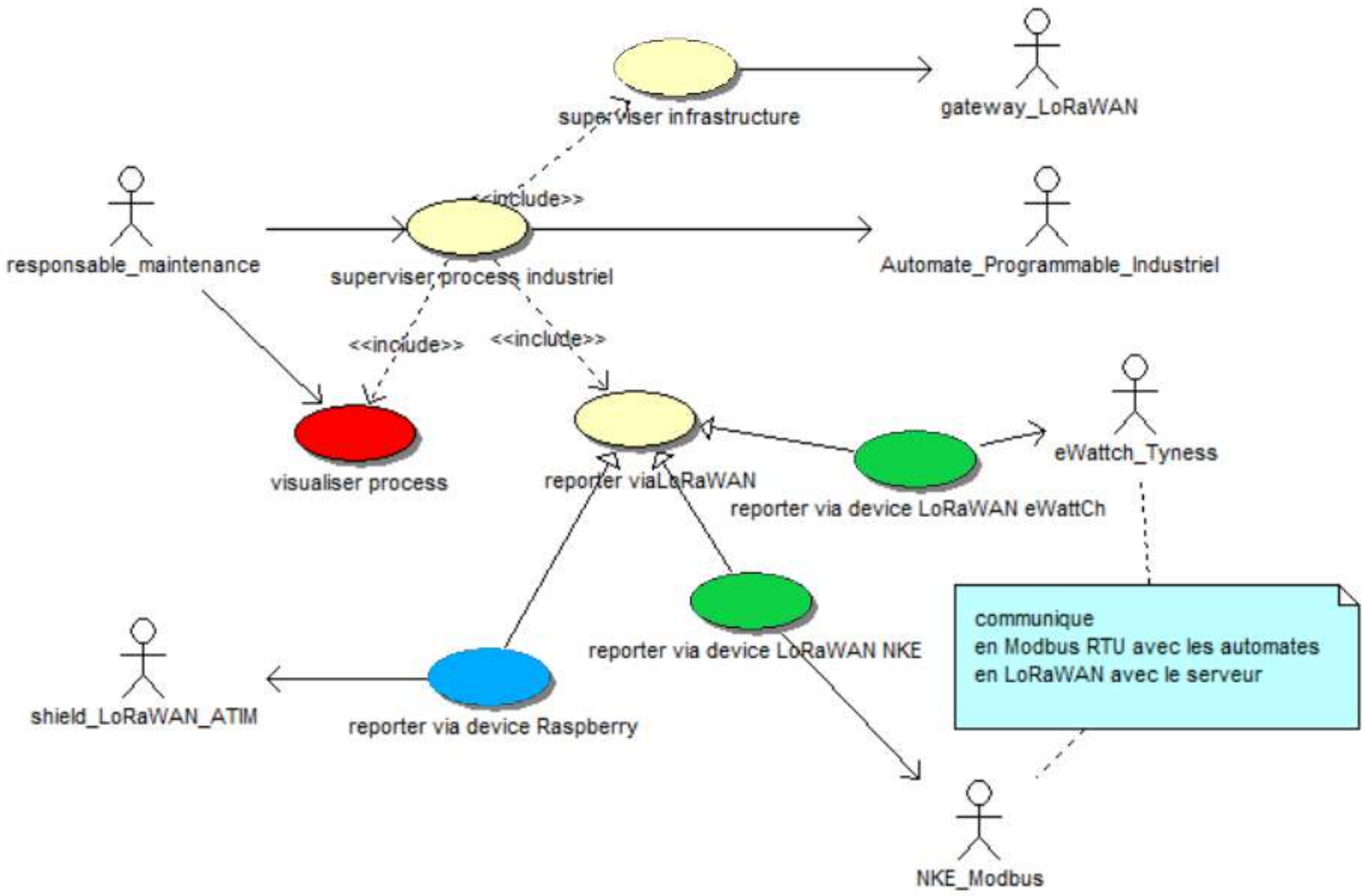
Introduction, et solution existante de notre projet



Sensing Vision est une société spécialisée dans le domaine de l'Internet des objets (IoT). Elle développe et installe des solutions pour surveiller les bâtiments de manière non-intrusive, principalement pour maîtriser la consommation d'énergie. Les capteurs déployés par Sensing Vision sont des capteurs communicants en LoRaWAN qui mesurent l'énergie et l'environnement. Dans ce projet, nous étudierons les solutions pour collecter les informations disponibles via Modbus dans des automates programmables (Twido) utilisés dans un atelier. Les informations collectées seront transmises via LoRaWAN et rendues disponibles sur un site web. Nous utiliserons les automates Twido déjà programmés et les interrogerons en utilisant Modbus RS485 et Modbus TCP. Les étapes comprennent le câblage d'eWattch Tyness et NKE Modbus avec l'automate Twido, la configuration de l'accès Modbus, les tests de relevés Modbus sur TTN, le développement de fonctions de décodage Tyness Modbus et NKE Modbus en Javascript sur TTN, l'intégration avec d'autres étudiants, la documentation de chaque partie et le développement d'une solution ergonomique pour la configuration du device.

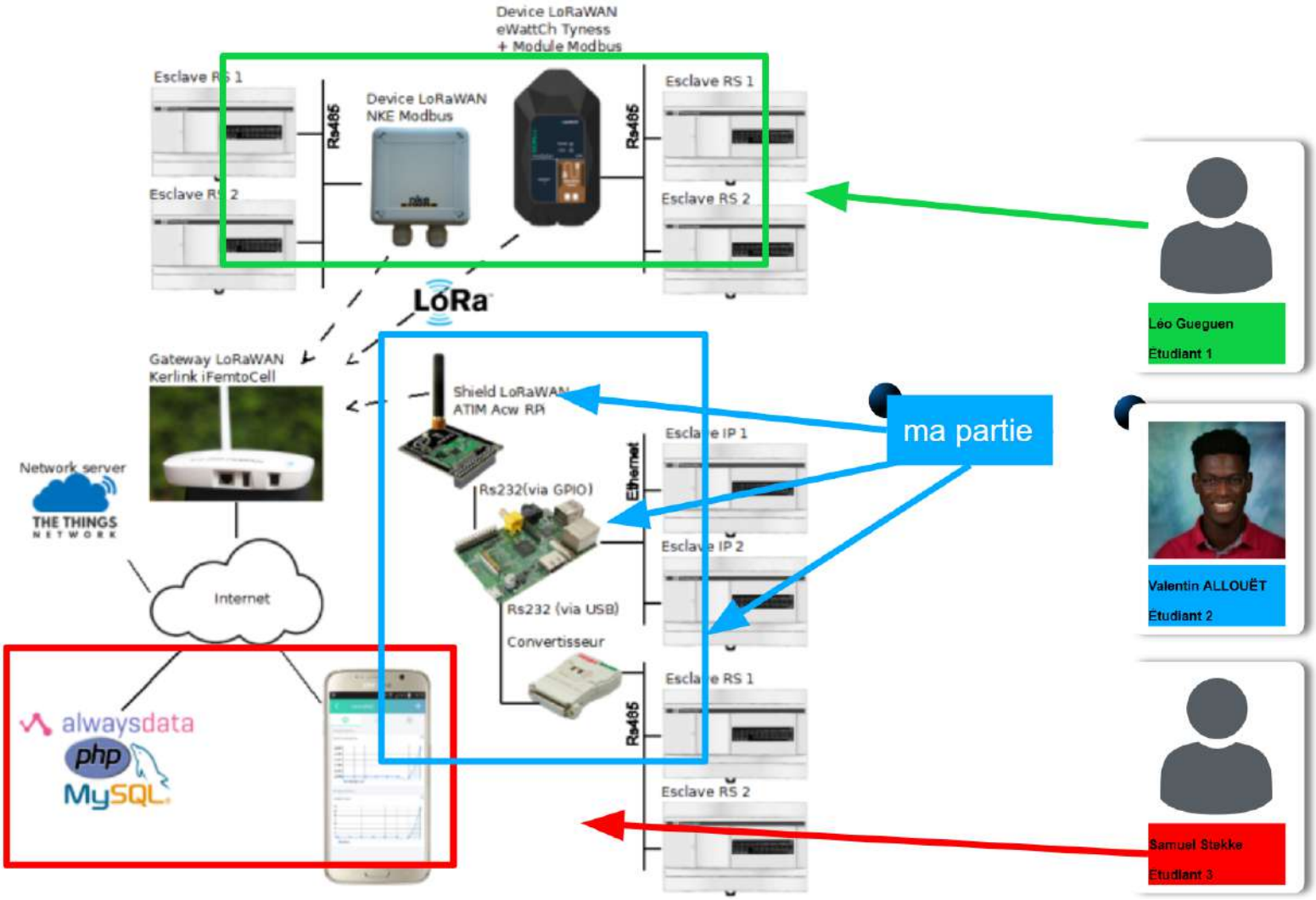
Notre cahier des charges est similaire à Sensing Vision dans le sens où nous avons aussi la même expression du besoin d'étudier des solutions permettant d'acquérir des informations disponibles via Modbus dans des automates programmables à l'échelle d'un atelier, de les faire remonter en LoRaWAN, et de les rendre disponibles sur un site web. Je précise que les automates Twido peuvent être interrogés en Modbus RS485 et Modbus TCP depuis un programme maître.

Analyse UML diagramme de cas d'utilisation :



Léo Gueguen et moi avons des objectifs similaires dans notre projet d'étude, car tous les deux nous devons récupérer des informations à partir d'un automate pour les afficher sur le site web de Samuel Stekke, notre collègue. Cependant, la méthode utilisée par Léo Gueguen pour obtenir ces informations est différente de la mienne, il doit les transmettre également en réseaux LoraWan dans des fréquences 863 MHz à 870 MHz, mais avec un Tyness et un appareil NKE.

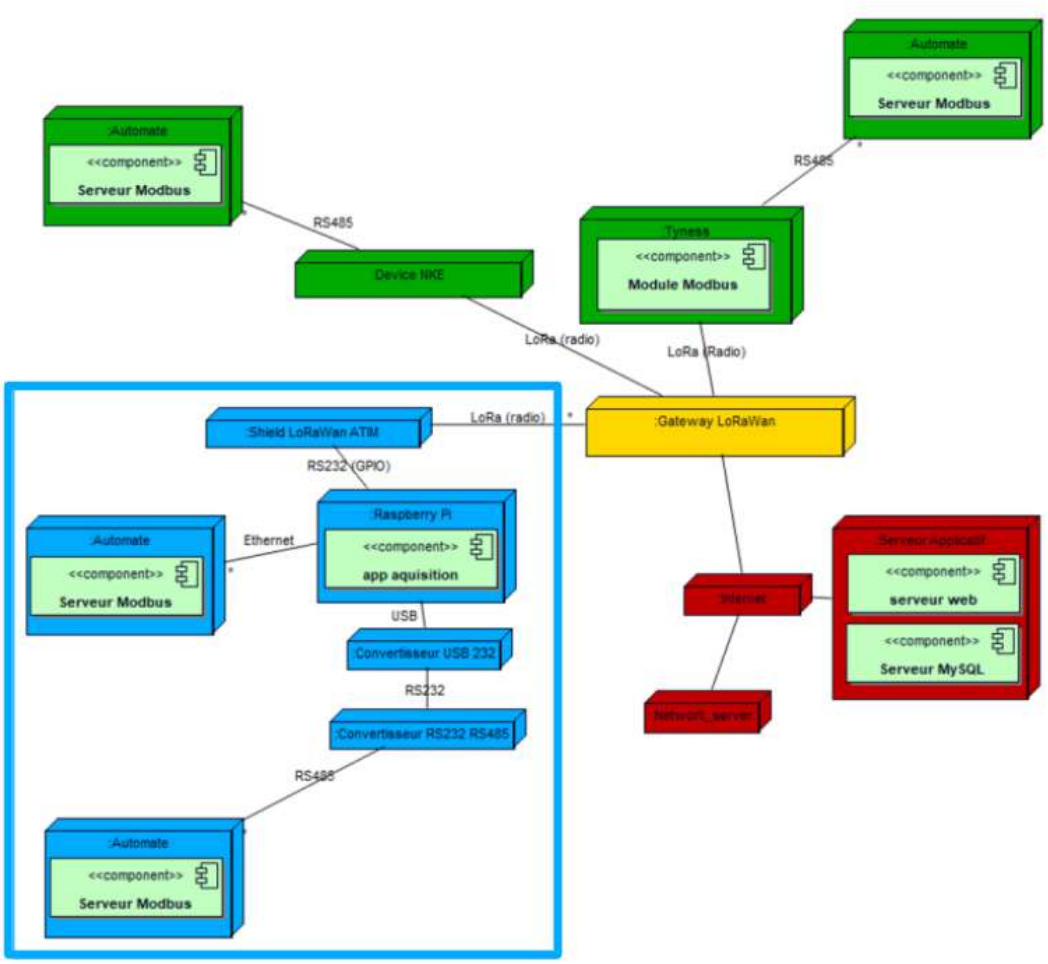
Présentation du Projet LoRaModbus :



Le projet LoRaModbus pourrait être utilisé dans divers contextes de télécommunications, téléphonie et réseaux téléphoniques, informatique, réseaux et infrastructures, mobilité, systèmes embarqués ou de mesure, instrumentation et microsystèmes, automates et robotique. Dans le cadre de ce projet, Samuel Stekke, étudiant 3, est responsable de la création d'un site web et de sa base de données en utilisant Bootstrap et AlwaysData. Les contraintes pour la création de son site sont les suivantes : il doit être responsive design et permettre la visualisation de l'évolution des compteurs de cycle des différents automates, avoir des seuils d'alerte pour les compteurs, un système de notification et une sécurité d'accès aux différentes pages avec un mot de passe et une gestion de session.

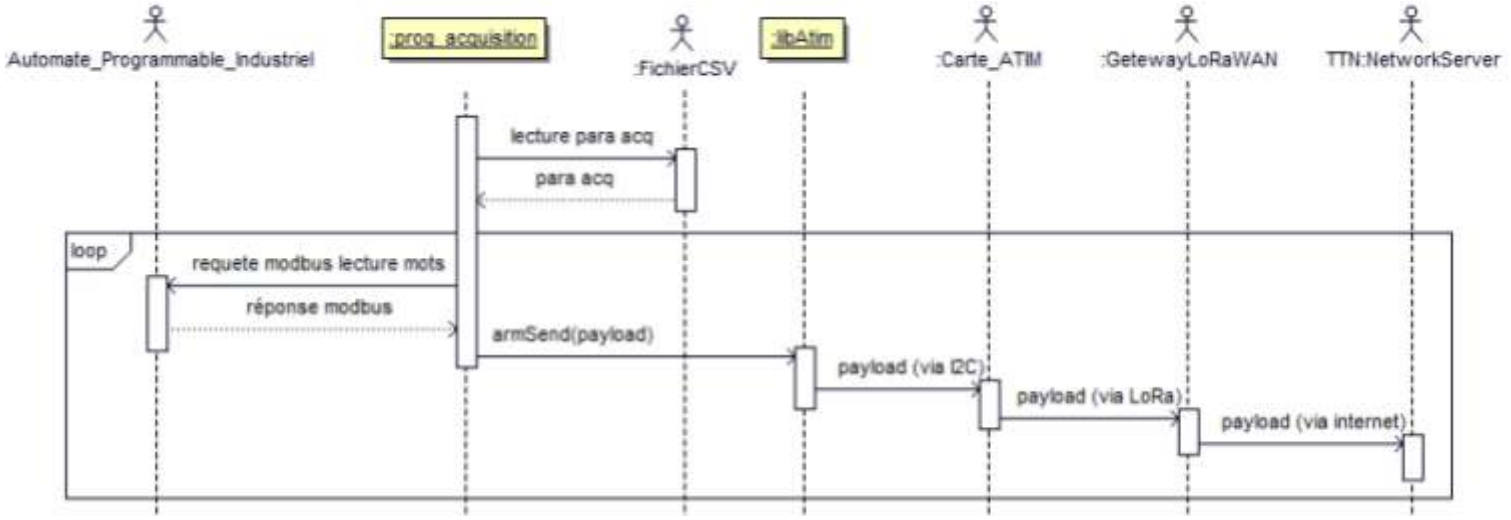
Je devrai donc transmettre à Samuel Stekke les données des automates, qui correspondent au compteur Cycles de l'automate et à l'état des feux de l'automate.

Analyse UML diagramme de déploiement :

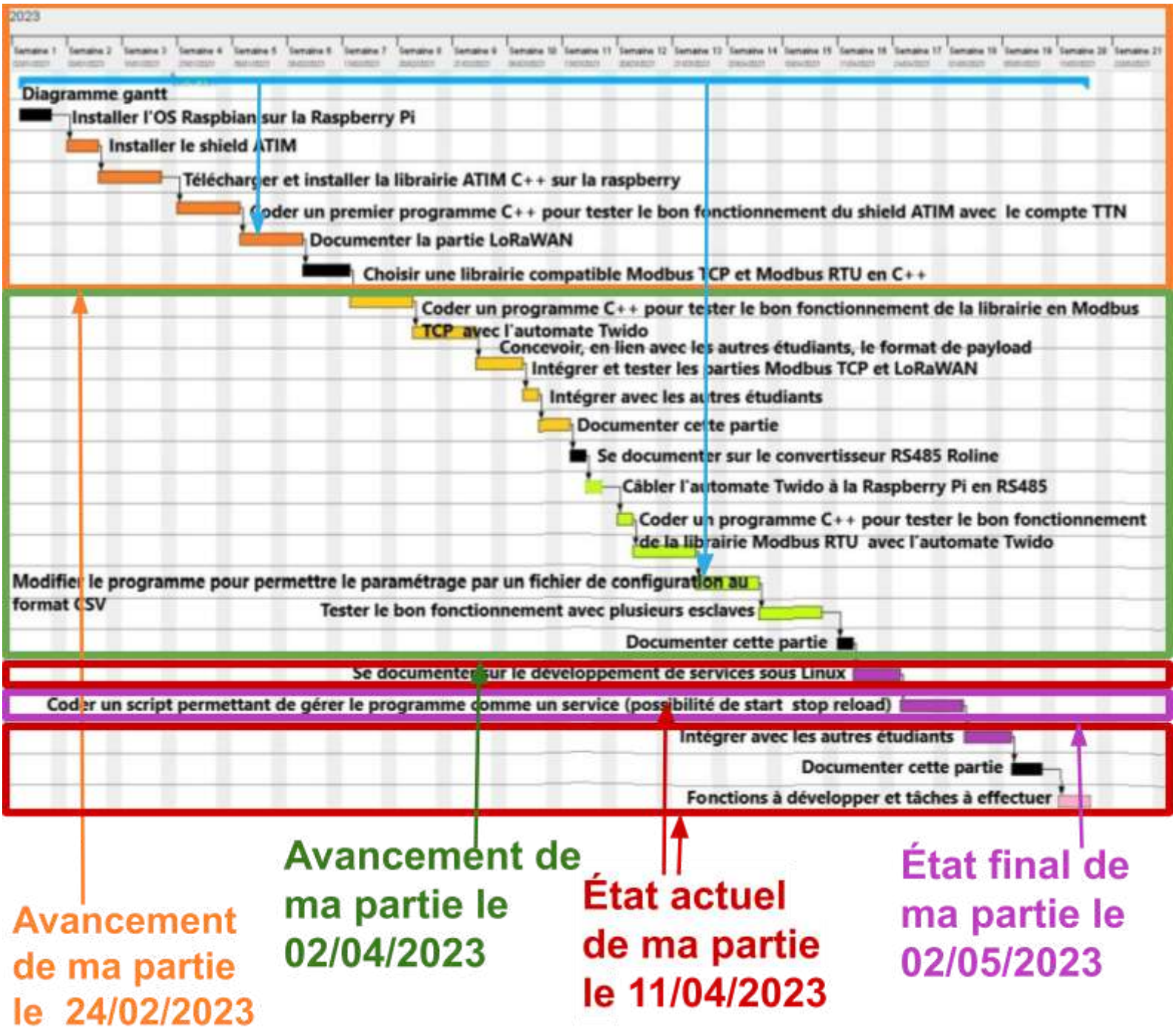


Mon travail consiste à transmettre des informations comme le compteur Cycles de l'automate et l'état des feux de l'automate, de l'automate Twido via un réseau Lora, dans une fréquence européen comprise entre 863 MHz et 870 MHz. Bien qu'un projet similaire ait déjà été réalisé en utilisant Arduino, j'ai utilisé une Raspberry Pi. Le programme en C utilise deux bibliothèques une librairie Modbus RTU, pour envoyer des requêtes et récupérer des réponses de l'automate et une librairie ATIM officiel, pour tout ce qui est la gestion de la carte Atim, qui envoie des données au comte TTN (The Things Network), qui permet d'interroger l'automate, est installé sur la Raspberry Pi et il peut quand même fonctionner sans cette dernière. Selon moi, il est important de noter que l'automate ne peut pas être connecté directement à la Raspberry Pi. Comme indiqué sur le diagramme de déploiement, les connexions entre l'automate et la Raspberry Pi sont réalisées via Ethernet ou par l'intermédiaire d'un convertisseur RS232 RS485, ainsi que via un convertisseur USB RS232. Le programme en C prend aussi en compte un fichier CSV qui permet l'ajout d'automates sans toucher au code du programme. Il y à un script sur la raspberry permet de lancer le programme et de le stopper.

le cahier des charges et le diagramme de gantt :

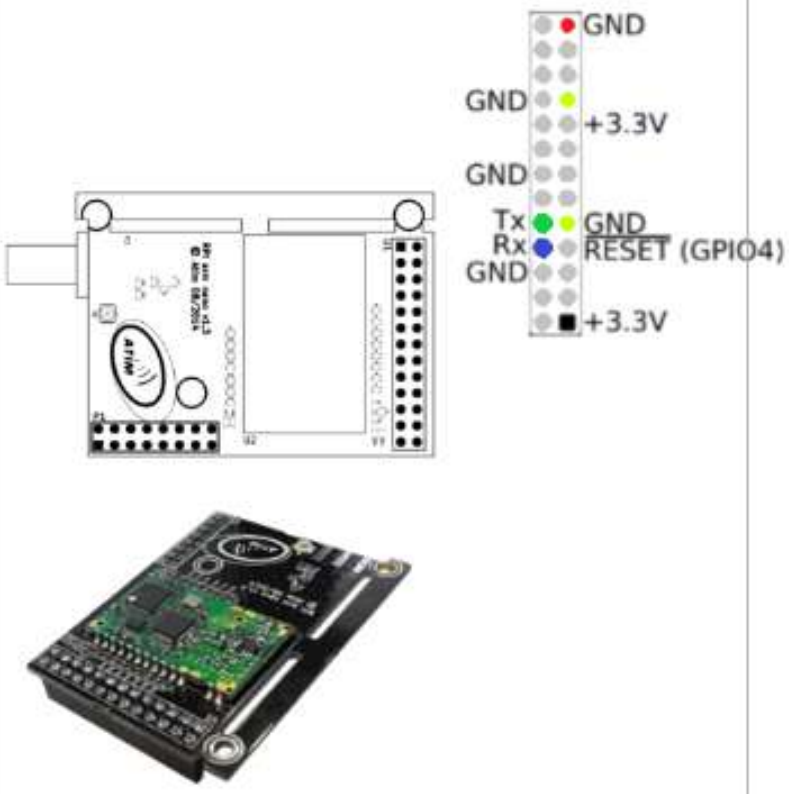


Ici, chaque couleur correspond à un des quatre blocs de ma partie du projet LoRaModbus.



Matériels utilisés :

j'aurais bien sûr pu connecter la carte ACW-RPI directement sur la Raspberry. Mais pour l'instant, je préfère connecter uniquement ce qui est essentiel. Chaque point coloré correspond à une broche de la carte ACW-RPI qui doit être connectée à la Rapberry par un capable comme ceci :

Carte Atime et Antenne	Raspberry PI 1
	

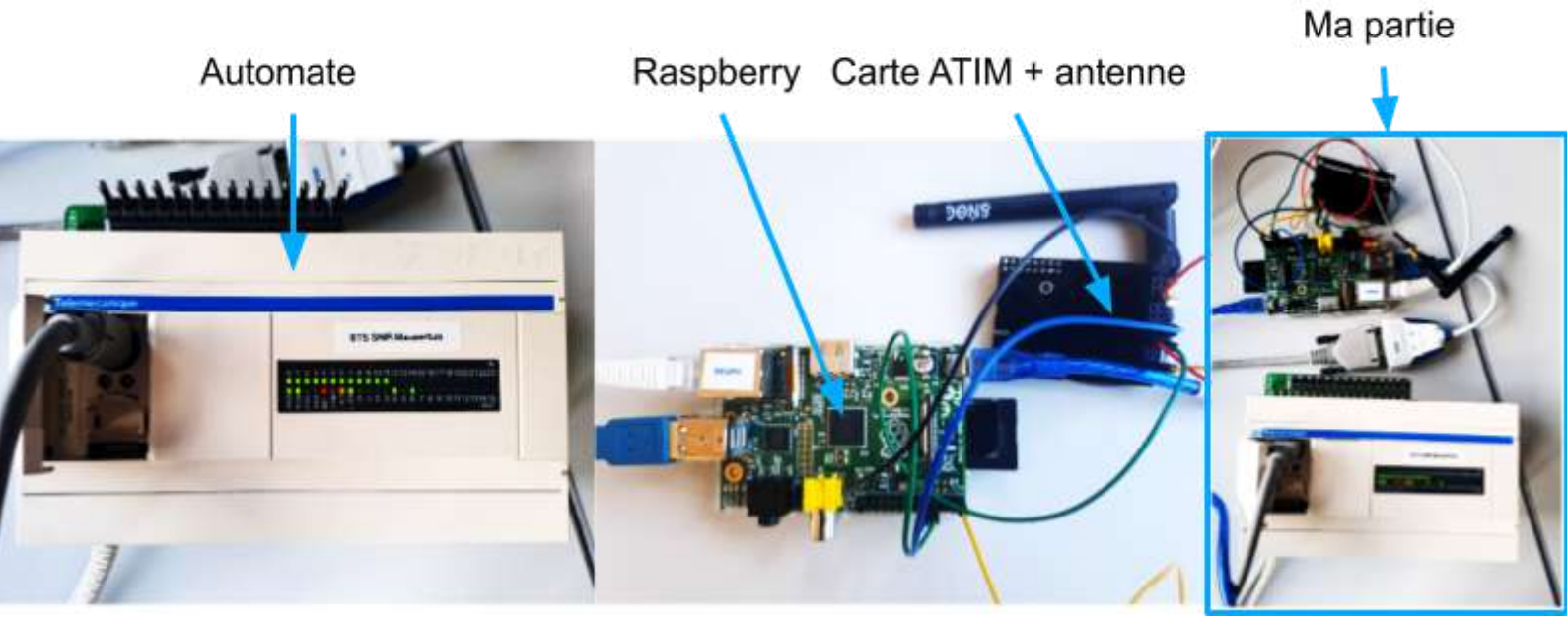


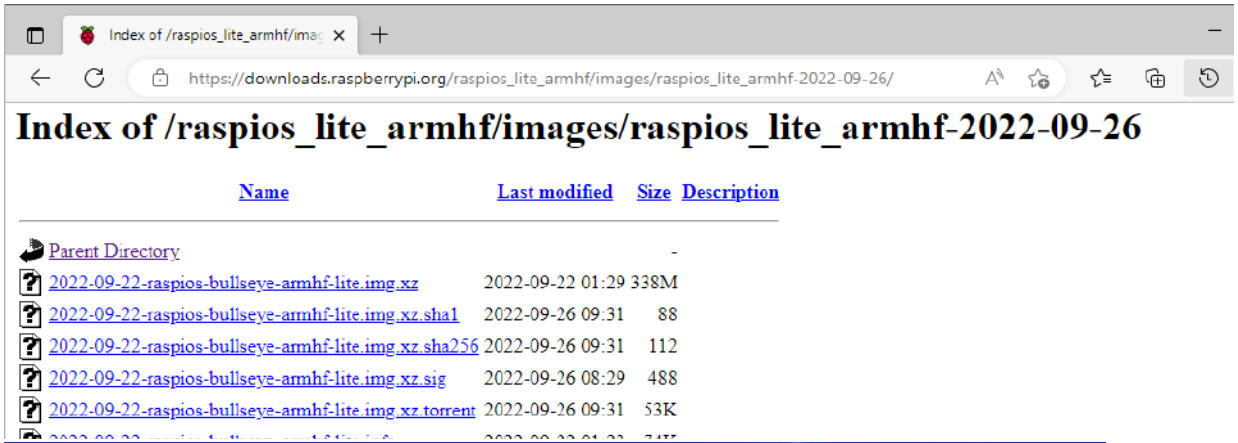
Tableau comparatif des solutions existantes :

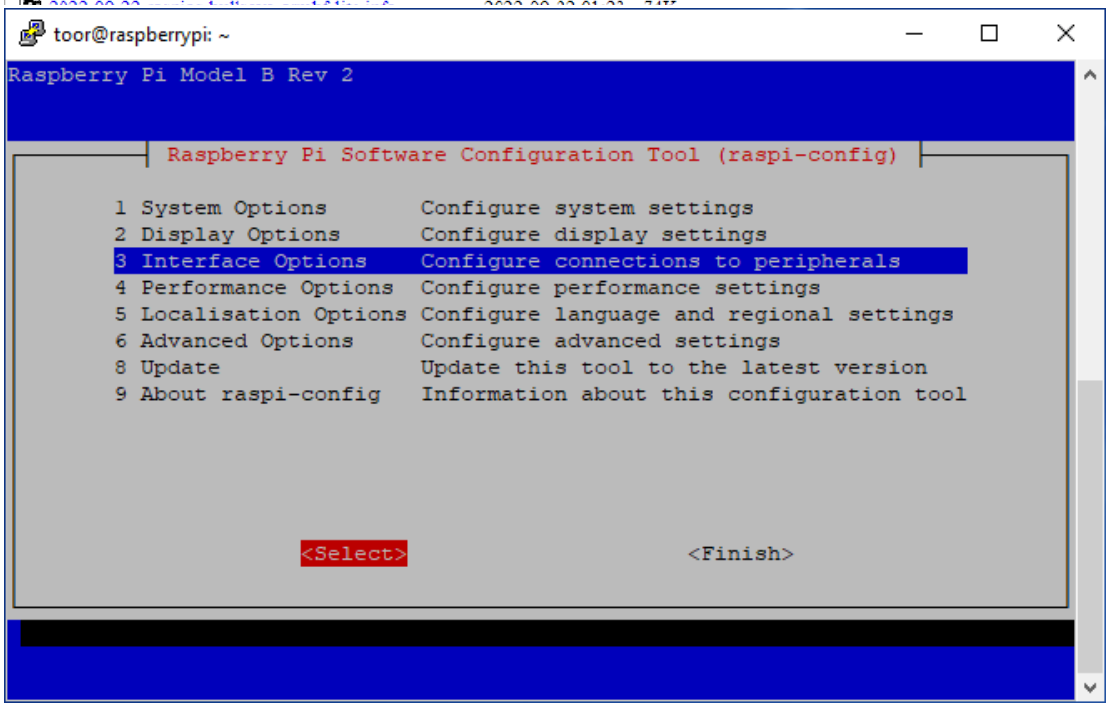
Vu qu'un projet similaire existe déjà et qu'il a été réalisé en utilisant Arduino, il serait intéressant de le comparer avec le projet LoRaModbus. La Raspberry Pi est plus performante et offre plus de possibilités de connectivité que l'Arduino, mais elle est également plus chère et consomme plus d'énergie. C'est pourquoi, pour des projets utilisant quelques dizaines d'automates, il est recommandé d'utiliser 2 ou 3 Arduino plutôt qu'une Raspberry, ce qui serait un gâchis pour le low tech. En revanche, pour des projets comportant plusieurs dizaines voire quelques centaines d'automates, il sera plus avantageux d'utiliser une dizaine de Raspberry que des centaines d'Arduino.

	Raspberry Pi	Arduino
Le coût plus élevée	✔	
+ de capacités de traitement	✔	
+ capacités de connectivité (Wi-Fi, Bluetooth)	✔	
- de consommation d'énergie		✔

Intégration de la carte ATIM et installation l'OS Raspbian de sur la Raspberry PI

J'ai téléchargé l'image de Raspbian depuis le site web de Raspberry Pi. Ensuite, j'ai gravé l'image sur une carte SD à l'aide d'un logiciel de gravure de disque, comme Etcher, ou j'ai utilisé Raspberry Pi Imager. Ensuite, j'ai inséré la carte SD dans la Raspberry Pi et je l'ai allumée. J'ai configuré les paramètres de la Raspberry Pi à l'aide de l'outil de configuration raspi-config qui est dans la console de Raspberry. Ensuite, j'ai mis à jour et installé les logiciels nécessaires en utilisant les commandes sudo apt update. Pour que les modifications prennent effet, j'ai redémarré la Raspberry Pi. Enfin, je me suis connecté en SSH à la Raspberry Pi à l'aide d'un client SSH tel que Putty.





Pour que le shield ATIM fonctionne avec la Raspberry Pi 1, les ports doivent être activés avec le terminal. Il faut aller dans le terminal exécuter la commande suivante “sudo raspi-config”, puis choisir l’option 3 et valider tous pour redémarrer la raspberry.

Réalisation d' un premier programme C++ pour tester le fonctionnement du shield ATIM avec le compte TTN

```cpp
//Include API
#include <arm.h>
//Instance of ARM class
Arm myArm;
//The message to sand
uint8_t msg[] = "Hello world";

int main()
{
//Initialize of API and check the error code.
if (armInit(&myArm, "/dev/ttyAMA0") != ARM_ERR_NONE)
return 1;

//If we want use the LoraWan network:
armSetMode(&myArm, ARM_MODE_LORAWAN);
//If we want use the Sigfox network:
//armSetMode(&myArm, ARM_MODE_SFX);
//If we want use the local mode:
//armSetMode(&myArm, ARM_MODE_SFK);

//After to send the message, we need to send the configuration to the radio modem.
armUpdateConfig(&myArm);

//Send of message.
armSend(&myArm, msg, sizeof(msg)-1);

return 0;
}
```

Source : http://atim-radiocommunications.github.io/armapi/doc/html/index.html

Le programme exemple de la documentation est écrit en C++ et utilise une API appelée "arm.h". Il envoie un message "Hello world" à un réseau de communication sans fil, soit Sigfox, soit LoraWan ou en mode local.

- La directive "#include" permet d'inclure l'API "arm.h".
- La classe "arm_t" représente une instance de la classe "arm".
- La variable "myArm" est une instance de la classe "arm_t".
- La variable "msg" est un tableau d'octets qui contient le message à envoyer.
- La fonction "main" est la fonction principale dans tout programme C++ et contient le code à exécuter.
- La fonction "armInit" initialise l'API en utilisant le périphérique "/dev/ttyAMA0". Si l'initialisation échoue, la fonction retourne 1.
- La fonction "armSetMode" définit le mode de communication sans fil à utiliser (Sigfox, LoraWan ou local).
- La fonction "armUpdateConfig" met à jour la configuration pour correspondre au mode de communication choisi.
- La fonction "armSend" envoie le message en utilisant l'instance de la classe "myArm".
- Enfin, la fonction "main" retourne la valeur 0 pour indiquer que le programme s'est terminé avec succès.

Réalisation d’une alternative fonctionnelle du programme en C++

Voici une alternative qui ne nécessite pas de librairie, mais qui est au moins aussi fonctionnelle. À noter que pour un test le fonctionnement de la carte ACW-RP ce code est largement suffisant, puisqu'il revient à exécuter la commande “`echo test > /dev/ttyAMA0`” sur la Raspberry. Mais pour piloter ce projet, je recommande la compilation d’un code et d'utilisation de la librairie ATIM.

```cpp
#include <iostream>
#include <stdlib.h>
#include <cstdlib>
using namespace std;
int main() {
  std:: system("echo test > /dev/ttyAMA0");
    return 0;
}
```

Le programme en C++ écrit la chaîne de caractères "test" dans le périphérique "/dev/ttyAMA0".

- La directive "#include" permet d'inclure les bibliothèques nécessaires dans le programme.
- La bibliothèque "iostream" fournit les entrées/sorties standard pour les entrées et les sorties de base en C++.
- La bibliothèque "stdlib.h" fournit des fonctions générales pour le langage C.
- La bibliothèque "cstdlib" fournit des fonctions générales pour le langage C++.
- La directive "using namespace std" permet d'utiliser les fonctions et les objets du namespace "std".
- La fonction "main" est la fonction principale dans tout programme C++ et contient le code à exécuter.
- La fonction "system" de la bibliothèque "stdlib.h" permet d'exécuter une commande shell dans un terminal.
- Dans ce cas, la commande "echo test > /dev/ttyAMA0" écrit la chaîne de caractères "test" dans le périphérique "/dev/ttyAMA0".
- Enfin, la fonction "main" retourne la valeur 0 pour indiquer que le programme s'est terminé avec succès.

Utilisation des commandes importantes sur Raspberry, pour tester le fonctionnement du shield ATIM avec le compte TTN

`echo 1 > /sys/class/gpio/gpio4/value` → echo 1 est une commande qui permet d'écrire la valeur "1" dans le fichier "value" dans le chemin "`/sys/class/gpio/gpio4/`" dans la raspberry. Elle réinitialise puis éteint la carte ATIM ACW-RPI.

`echo 0 > /sys/class/gpio/gpio4/value` → echo 0 est similaire à echo 1, mais elle réinitialise puis allume la carte ATIM ACW-RPI.

`stty -F /dev/ttyAMA0` → permet de connaître le débit de mot en baud.

`stty -F /dev/ttyAMA0 19200` → change le débit de mot en baud.

`gcc -o ATIM_TTN.bin ATIM_TTN.o arm.o armport/armport_unix.o` → Cette commande utilise le compilateur GCC pour compiler et lier des fichiers pour produire un exécutable nommé "ATIM_TTN.bin".

- "gcc" (GNU Compiler Collection), qui est un ensemble de compilateurs pour les différents langages de programmation, y compris C et C++.
- "-" signifie que l'option suivante est une option de ligne de commande pour le compilateur.
- "o ATIM_TTN.bin" indique que le nom de l'exécutable produit doit être "ATIM_TTN.bin".
- "ATIM_TTN.o" est un fichier objet compilé à partir du code source "ATIM_TTN.c".
- "arm.o" est un fichier objet compilé à partir du code source "arm.c".
- "armport/armport_unix.o" est un fichier objet compilé à partir du code source "armport_unix.c" situé dans le répertoire "armport".

g++ -o ATIM_TTN.c.bin ATIM_TTN.c → Exécute et compile le programme ATIM_TTN.

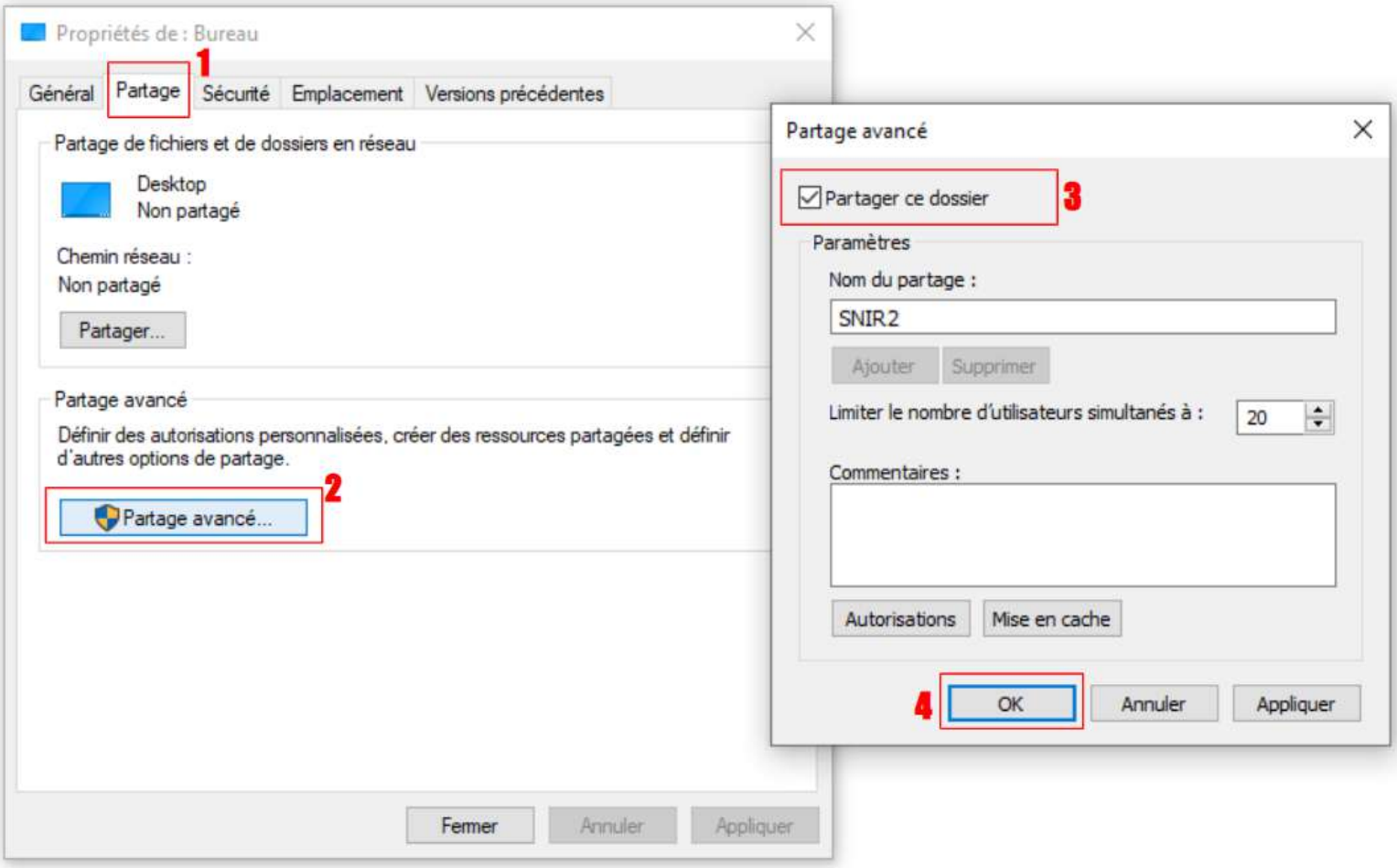
Intégration de la cross compilation avec partage de dossiers Côté Windows :

Pourquoi faire de la cross compilation ?

Dans notre cas, nous utilisons une Raspberry Pi 1, et nous voulons compiler sur un ordinateur plus puissant pour améliorer le temps de compilation un meilleur outil de développement comme Codeblocks et optimiser le projet.

La cross-compilation n'est en soit pas obligatoire, mais j'ai préféré l'utiliser, car elle permet de compiler un programme sur une architecture de développement différente comme windows, mais sera exécutée via un partage de dossier entre un ordinateur windows et une raspberry. Cela m'a permis de gagner du temps en évitant de devoir installer un environnement de développement sur la raspberry qui est dix fois plus lent qu'un ordinateur, ainsi que de réduire la taille du binaire en supprimant les parties inutiles de la rasberry. La cross-compilation est particulièrement utilisée dans les environnements embarqués, où les ressources sont limitées, comme le projet LoRaModbus . La cross compilation permet donc de compiler des programmes pour des plateformes différentes, par exemple pour des systèmes d'exploitation ou des processeurs différents. La cross-compilation à des avantages comme la portabilité, la rapidité et l'optimisation de la taille de l'exécution du code.

Pour commencer il suffit de choisir un dossier et de suivre ces étapes :



Côté Raspberry :

première technique :

sudo apt install smbclient → Cette commande install un smbclient sur la raspberry, cet outil open source à permis à la raspberry de se connecter à des partage Samba sur un ordinateur du projet LoRaModbus

sudo mount pc43 → Cette commande monte le dossier pc43 dans la raspberry, elle crée un dossier dans la raspberry qui fera le lien entre le dossier SNIR2 dans l'ordinateur et le dossier pc43 dans la raspberry.

sudo mount -overs=2.1,username=[redacted],password=[redacted] //192.168.12.43/partagePi pc43 → Cette commande donne une vision de la machine windows sur la raspberry, je précise qu'il faut saisir le nom d'utilisateur et le mot de passe de la machine qui partage le dossier SNIR2.

Session Adresse IP dossier

```
toor@raspberrypi:~ $ smbclient -U adm //192.168.12.43/snir2
Enter WORKGROUP\adm's password:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Wed Mar 15 07:27:06 2023
  ..                                  D        0  Wed Mar 15 07:27:06 2023

                62172671 blocks of size 4096. 18157315 blocks available
smb: \> ls
  .                                   D        0  Wed Mar 15 07:38:42 2023
  ..                                  D        0  Wed Mar 15 07:38:42 2023
  protoCrossCompilation01.bin         A     5584  Wed Mar 15 07:26:03 2023

                62172671 blocks of size 4096. 18157241 blocks available
smb: \> quit
```

Deuxième technique similaire à la première :

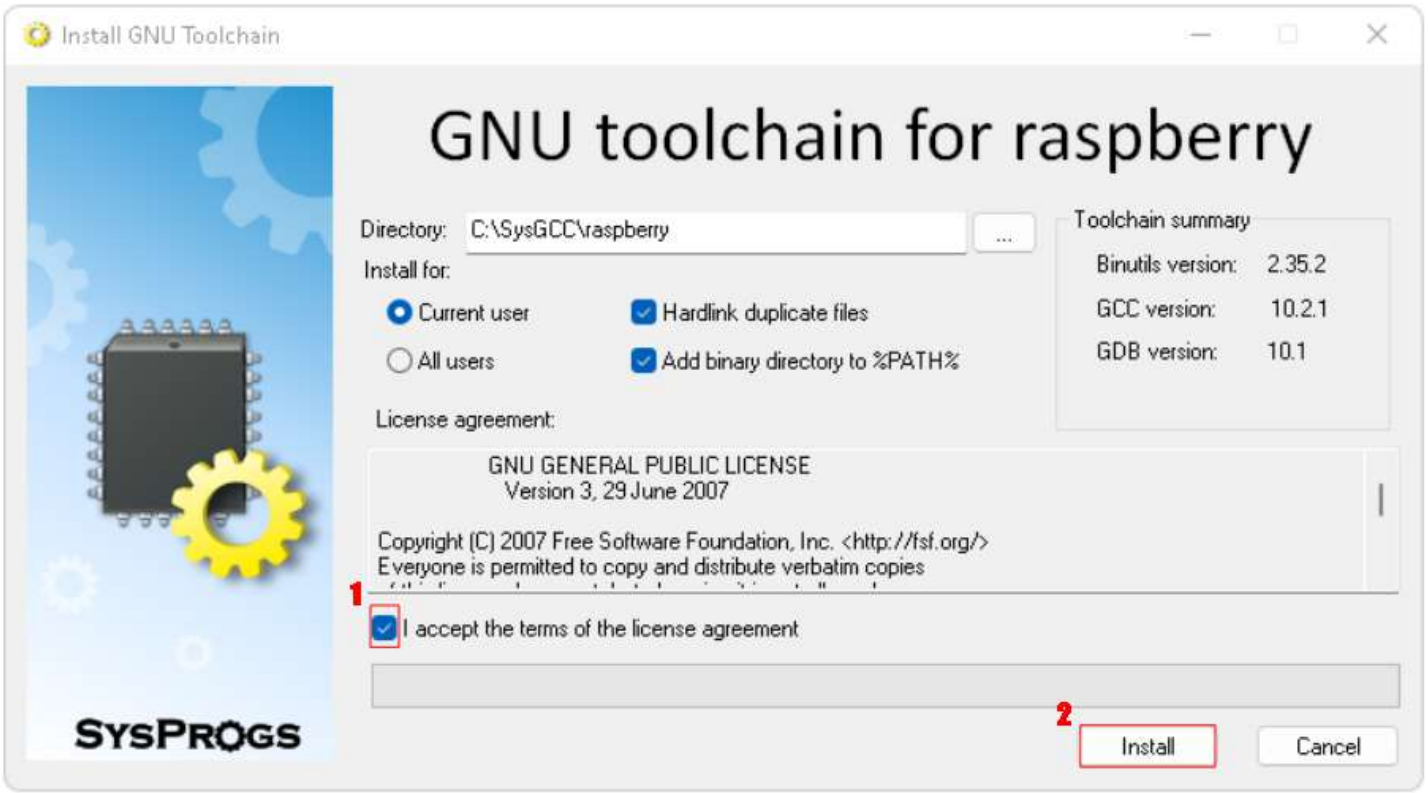
```
toor@raspberrypi:~ $ cd pc43
toor@raspberrypi:~/pc43 $ ls
toor@raspberrypi:~/pc43 $ cd
toor@raspberrypi:~ $ sudo mount -overs=2.1,username=adm,password=[redacted]//192.168.12.43/partagePi pc43
toor@raspberrypi:~ $ cd pc43
toor@raspberrypi:~/pc43 $ ls
acquisitionAutomateViaRS485EtEnvoiAtimv00.bin     paraModbusv00.csv
CSVacquisitionAutomateViaRS485EtEnvoiAtimv00.bin  save
toor@raspberrypi:~/pc43 $
```

Intégration de la cross compilation avec GNU toolchains :

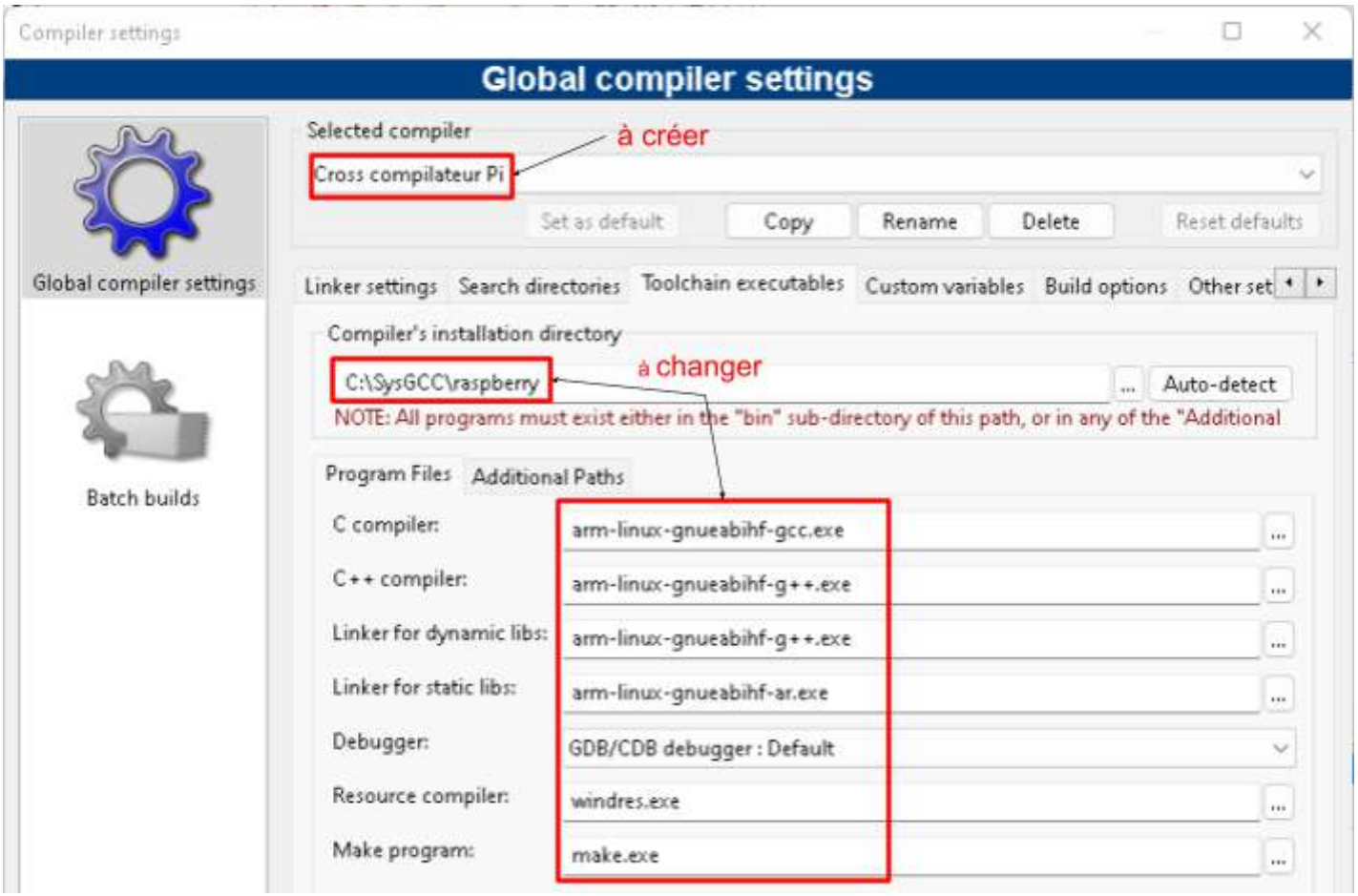
GCC	Compatible SD card image	Download link
10.2.1	2021-10-30-raspios-bullseye	raspberry-gcc10.2.1.exe (588 MB)
8.3.0	2021-05-07-raspios-buster	raspberry-gcc8.3.0-r4.exe (761 MB)
8.3.0	2020-12-02-raspios-buster-armhf	raspberry-gcc8.3.0-r3.exe (749 MB)
8.3.0	2019-07-10-raspbian-buster-full	raspberry-gcc8.3.0.exe (288 MB)
6.3.0	2019-04-08-raspbian-stretch-full	raspberry-gcc6.3.0-r5.exe (448 MB)

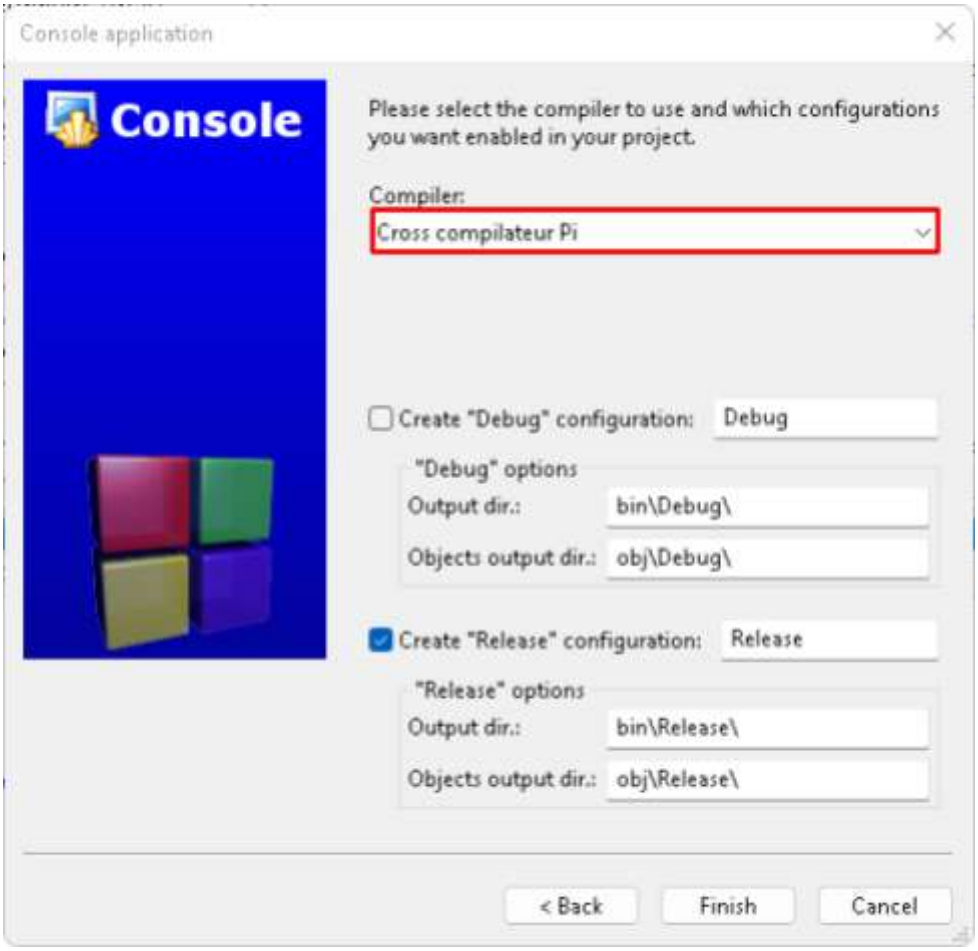
source : https://gnutoolchains.com/raspberry/

L'installation de GNU toolchain for raspberry est un outil de compilation qui sera dans le projet LoRaModbus, l'outil de compilation employé par Codeblocks qui permet la compilation pour une architecture de raspberry.



Intégration de la cross compilation dans Codeblocks :





J'ai aussi dû modifier dans Codeblocks les compilateurs C, C++ et les librairies dynamiques, statiques pour l'édition des liens.

Lors de la création du programme du projet LoraModbus, j'ai choisi le bon compilateur pour raspberry.

```c
#include <stdio.h>      // Standard Input/Output library
#include <stdlib.h>     // Standard library
#include <string.h>     // String handling functions
#include <errno.h>      // Error-handling macros
#include <modbus.h>     // Modbus communication protocol library
#include <stdint.h>     // Standard integer types
#include "arm.h"        // Custom header file
```

```c
struct ModbusConfig {
    char type[MAX_LINE_LEN];
    char adresse[MAX_LINE_LEN];
    char esclave[MAX_LINE_LEN];
    char adrDebut[MAX_LINE_LEN];
    char nbMots[MAX_LINE_LEN];
};
```

La structure ModbusConfig déclare des noms de tableaux, définis leur taille maximum.

La fonction ctx_rtu_tcp() utilise la bibliothèque Modbus, elle peut lire les données de registre à partir d'un périphérique Modbus :

```c
if (ctx == NULL)
{
    fprintf(stderr, "Erreur lors de la création du contexte Modbus RTU/TCP\n");
    return -1;
}
if(modbus_set_slave(ctx,1)!=0)
{
    fprintf(stderr, "err set slave\n");
    return -1;
}

if (modbus_connect(ctx) == -1)
{
    fprintf(stderr, "Connexion Modbus RTU/TCP impossible : %s\n", modbus_strerror(errno));
    modbus_free(ctx);
    return -1;
}
```

Cette condition vérifie si le contexte Modbus a été créé avec succès et si le périphérique esclave de l'automate est défini comme 1.

Sinon, ils affichent une erreur et retournent -1.

modbus_connect() est une fonction qui connecte l'appareil Modbus spécifié par le contexte. Si la connexion échoue, elle affiche une erreur et retourne -1.

```c
int adresseDebut = 0;
int nbMots = 3;
int rc0 = modbus_read_registers(ctx, adresseDebut, nbMots, tab_reg);

if (rc0 == -1)
{
    fprintf(stderr, "Erreur lors de la lecture des registres : %s\n", modbus_strerror(errno));
    modbus_close(ctx);
    modbus_free(ctx);
    return -1;
}
modbus_close(ctx);
modbus_free(ctx);

for(int i = 0; i<3; i++)
    printf("La valeur du registre %d est : %d\n", i, tab_reg[i]);
printf("fin !\n");
return 0;
```

Ici c'est la définition des variables pour l'adresse de début, le nombre de mots et le tableau de registres.

Là il y a un stockage des données lues à partir de l'appareil, avec le tableau de registres, qui est passé à la fonction modbus_read_registers()

modbus_close() et modbus_free() ferment la connexion et libèrent la mémoire allouée pour le contexte Modbus.

Une boucle for est utilisée pour parcourir le tableau de registres et afficher les valeurs lues à des fins de débogage.

Réalisation d'un programme C LoRaModbus, fonctionnant avec shield ATIM avec le compte TTN et le fichier CSV

```c
FILE* fp;
char buffer[MAX_LINE_LEN];
char* field;
int field_count = 0;
struct ModbusConfig config_arr[10];
int row_count = 0;
fp = fopen("paraModbusv00.csv", "r");
if (!fp)
{
    printf("Failed to open paraModbusv00.csv\n");
    exit(1);
}
```

La fonction CSV_file() est une fonction qui lit les données du fichier CSV nommé paraModbusv00.csv. Le pointeur de champ char* et la variable int field_count sont permettent de garder une trace du nombre des colonnes dans chaque ligne du fichier en cours de lecture.

La structure ModbusConfig config_arr[10] est un tableau de 10 objets de structure de type ModbusConfig, qui est défini plus haut dans le code, les champs de la structure correspondant aux colonnes de la ligne.

La variable int row_count est utilisée pour garder une trace du nombre de lignes qui ont été lues depuis le paraModbusv00.csv, le fichier CSV qui ressemble à ceci :

```c
while (fgets(buffer, MAX_LINE_LEN, fp))
{
    field_count = 0;
    struct ModbusConfig config;
    field = strtok(buffer, ";");

    while (field)
    {
        switch (field_count)
        {
        case 0:
            strcpy(config.type, field);
            break;
        case 1:
            strcpy(config.adresse, field);
            break;
        case 2:
            strcpy(config.esclave, field);
            break;
        case 3:
            strcpy(config.adrDebut, field);
            break;
        case 4:
            strcpy(config.nbMots, field);
            break;
        }

        field = strtok(NULL, ";");
        field_count++;
    }

    config_arr[row_count] = config;
    row_count++;
}
```

type	adresse	esclave	adrDebut	nbMots
RS	/dev/ttyUSB0	1	0	3
IP	192.168.12.43	1	0	3
RS	/dev/ttyUSB0	2	0	10
IP	192.168.12.182	1	0	3

Si le fichier ne s'ouvre pas, un message d'erreur est affiché et le programme se termine.

field = strtok(NULL, " ;") utilise la fonction strtok pour faire la séparation d'une ligne, c'est un repaire pour nous. L'argument NULL indique que strtok doit reprendre l'analyse là où elle s'est arrêtée.

La boucle while est utilisée temp que le champ n'est pas null. L'instruction switch vérifiant la valeur des champs de field_count, si field_count est égal à 0, et copie les valeurs des champs avant de s'arrêter.

```c
fclose(fp);
```

fclose(fp) est une fonction qui ferme le fichier CSV.

Réalisation d' un programme C LoRaModbus, fonctionnant avec shield ATIM avec le compte TTN et le fichier CSV

```
for (int i = 0; i < row_count; i++)
{
```

Dans la boucle for printf sert à afficher le contenu du CSV.

```
    printf("%s %s %s %s %s\n", config_arr[i].type, config_arr[i].adresse, config_arr[i].esclave,
config_arr[i].adrDebut, config_arr[i].nbMots);

    char IP[] = "IP";
    char RS[] = "RS";
```

Nous créons ici deux tableaux IP et RS.

```
    if(config_arr[i+1].type != RS)
    {
        ctx = modbus_new_tcp(config_arr[i].adresse, 502);
        ctx_rtu_tcp();
    }
    else
    {
        ctx = modbus_new_rtu(config_arr[i].adresse, 19200, 'N', 8, 1);
        ctx_rtu_tcp();
    }
```

Et là, nous testons si le type dans la première colonne du fichier CSV est RS ou IP.

Là c'est la création du contexte modbus TCP et l'appel de la fonction ctx_rtu_tcp().

Sinon on crée un contexte modbus RTU et l'appel de la fonction ctx_rtu_tcp().

```
}
```

```
int main()
{

    CSV_file()
    if (armInit(&myArm,(void*) "/dev/ttyAMA0") != ARM_ERR_NONE)
        return 1;
    armSetMode(&myArm, ARM_MODE_LORAWAN);
    armUpdateConfig(&myArm);

    for(int i = 0; i<3; i++)
    {
        msg[i*2]=tab_reg[i]%256; // poids faible
        msg[i*2+1]=tab_reg[i]/256; // poide fort
    }

    armSend(&myArm, msg, sizeof(msg));

    printf("fin !\n");
    return 0;
}
```

La fonction armUpdateConfig(&myArm), met à jour la configuration de l'interface série, avec de nouvelle options de transmission

Ici la boucle for est utilisée pour créer un tableau d'octets nommé msg

Et ici, la fonction armSend() envoie les données stockées dans le tableau msg

Vérification du fonctionnement du programme LoRaModbus, avec le compte TTN et le fichier CSV

Ici, on peut remarquer que tous les paramètres du fichier CSV sont pris en compte, puisqu'ils sont utilisés pour la connexion entre les automates et la raspberry. Dans cet exemple, il y a un automate et une raspberry capable de communiquer entre eux pour ce faire, la raspberry se connecte à l'automate en RTU, modbus ou avec une adresse ip via un câble ethernet. Et dans le Payload Formatters, il y a une fonction qui décode et reformate le Payload reçu par TTN.

```
function decodeUplink(input) {
  return {
    data: {
      etatFeux : input.bytes[1]*256+input.bytes[0],
      compteurCycles : (input.bytes[5]*256+input.bytes[4])*10000+(input.bytes[3]*256+input.bytes[2]),
    },
    warnings: [],
    errors: []
  };
}
```

Console The Things Network

```
16:20:42  Console: Events cleared        The events list has been cleared
15:17:43  Forward uplink data message    DevAddr: 26 0B 10 46  Payload: {compteurCycles: 7434, etatFeux: 72}
15:17:43  Successfully processed data …  DevAddr: 26 0B 10 46  48 00 0A 1D 00 00
```

Valeurs importantes

Console Raspberry Pi 1

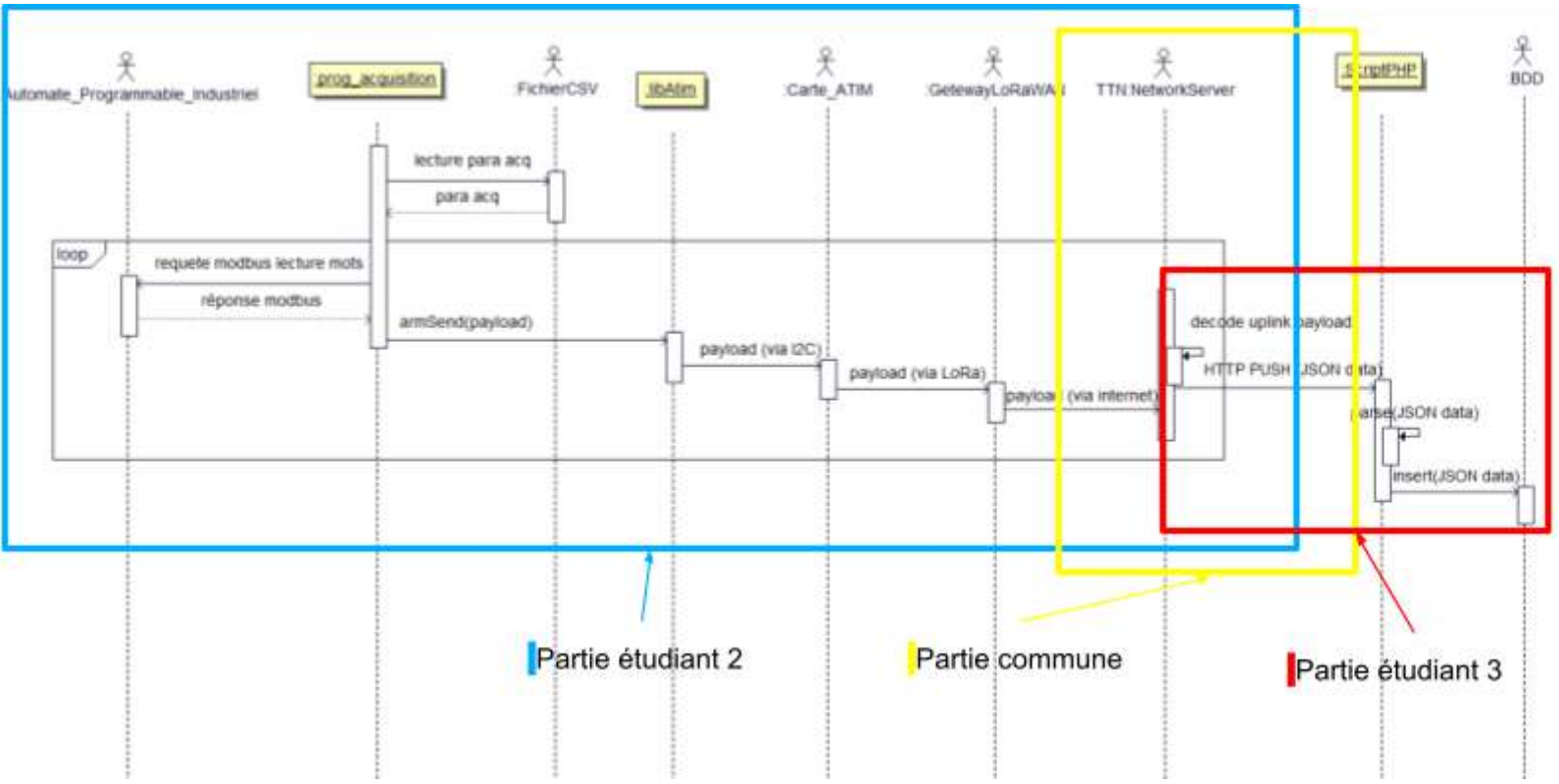
```
toor@raspberrypi:~ $ sudo mount -overs=2.1,username=adm,password=bonjour //192.168.12.43/partagePi pc43
mount error(16): Device or resource busy
Refer to the mount.cifs(8) manual page (e.g. man mount.cifs) and kernel log messages (dmesg)
toor@raspberrypi:~ $ cd pc43
toor@raspberrypi:~/pc43 $ ./CSVacquisitionAutomateViaRS485EtEnvoiAtimv01.bin
type adresse esclave adrDebut nbMots

Connexion Modbus RTU/TCP impossible : Success
RS /dev/ttyUSB0 1 0 3

Connexion Modbus RTU/TCP impossible : Success
IP 192.168.12.43 1 0 3

Connexion Modbus RTU/TCP impossible : Operation now in progress
RS /dev/ttyUSB0 2 0 10

Connexion Modbus RTU/TCP impossible : Operation now in progress
IP 192.168.12.182 1 0 3

La valeur du registre 0 est : 72
La valeur du registre 1 est : 7434
La valeur du registre 2 est : 0
fin !
fin !
```

Valeurs importantes

type	adresse	esclave	adrDebut	nbMots
RS	/dev/ttyUSB0	1	0	3
IP	192.168.12.43	1	0	3
RS	/dev/ttyUSB0	2	0	10
IP	192.168.12.182	1	0	3

paraModbusv00.csv

La Fiche de recette :



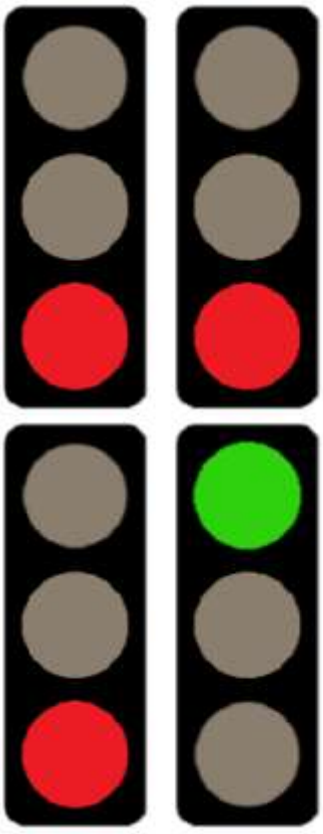
OK	NO	**Fonctionnalitées**
✔		**Fonctionnement de l'OS sur la Raspberry.**
✔		**Fonctionnement de la carte Atim sur la Raspberry.**
✔		**Réception et envoie des données via la carte Atim.**
✔		**Prise en charge du format de payload**
✔		**Intégration des parties Modbus TCP et LoRaWAN**
✔		**Acquisition de l'automate en TCP par la Raspberry via un programme en C**
✔		**Prise en compte des mode de connection avec un fichier CSV**
✔		**Coder un script permettant de gérer le programme comme un service (possibilité de start sto reload)**
✔		**Intégration avec les autres parties du projet**

Conclusion

Selon moi, l'utilisation de la librairie ATIM pour le projet LoRaModbus est optionnelle, mais pourrait être utile pour une réutilisation ultérieure du projet LoRaModbus dans un contexte différent. Pour d'autres projets, la librairie ATIM aurait permis de mieux contrôler la carte ATIM.

Au niveau de ma partie, les fonctionnalités principales sont présentes, et il y en a une, notamment les fonctionnalités qui permettent de pouvoir démarrer, lancer et arrêter mon programme fonctionne. Mais une fois que la Raspberry redémarre, le script n'est plus appelé et il faut donc ré-exécuter le script qui appelle toutes les caractéristiques de programmes du projet. Sur la partie intégration avec les autres membres du groupe, nous pouvons affirmer que tous fonctionne, puisque j'ai travaillé avec Léo l'étendant 1, et il m'a montré comment faire pour convertir des données hexadécimales qui sont en little endian, en donner décimal directement depuis la console TNN. J'ai aussi travaillé avec l'étudiant 3 Samuel, il reçoit les données de la Raspberry Pi, d'où le nom de snirpi01.

Adresse URL actuelle à la rédaction de ce rapport : http://vsl.alwaysdata.net/



nkemodbus01 : etatFeux =68et8440 cycles à 2023-05-10 11:49:13

nkemodbus01 représente les données de l'automate de Léo l'étudiant 1.

snirpi01 : etatFeux =80et3758 cycles à 2023-05-12 11:42:16

snirpi01 représente les données d'un des automate que j'ai utilisé.

Je dirais aussi que LoRaWAN n'est pas un bon choix pour notre projet, car il doit gérer un système en temps réel de communication sans fil. LoRaWAN est fait pour des projets qui nécessitent moins de rapidité. Donc, pour un projet de météorologie qui envoie des données heure par heure, LoRaWAN serait plus adapté.