Académie Aix-Marseille

B.T.S SN 2016

Projet :

MARI-CHAU

Lycée Alphonse Benoit

L'Isle sur Sorgue

Etudiants :

Deligny Dylan

Laly Maxime

Lambert Benjamin

Bernard Alex

Partie Commune :

PROJET MARI-CHAU

Présentation Générale

L'entreprise

Mariton est une entreprise spécialisée dans la fabrication de stores et de moustiquaires sur mesure. Elle est localisée à Saint Chamas dans les Bouches du Rhône (13).

Depuis plus de 60 ans, Mariton habille aussi bien les fenêtres des maisons individuelles que celles des entreprises, des administrations et des collectivités.

Innovante, performante et forte d'un savoir-faire maîtrisé, Mariton propose des produits de qualité, bien pensés et adaptés à chaque configuration : stores vénitiens, stores à bandes verticales, stores enrouleurs intérieurs ou extérieurs, stores plissés, panneaux japonais, vélums de véranda, moustiquaires pour fenêtre, moustiquaires pour porte, Sécurit'Air …

Le projet

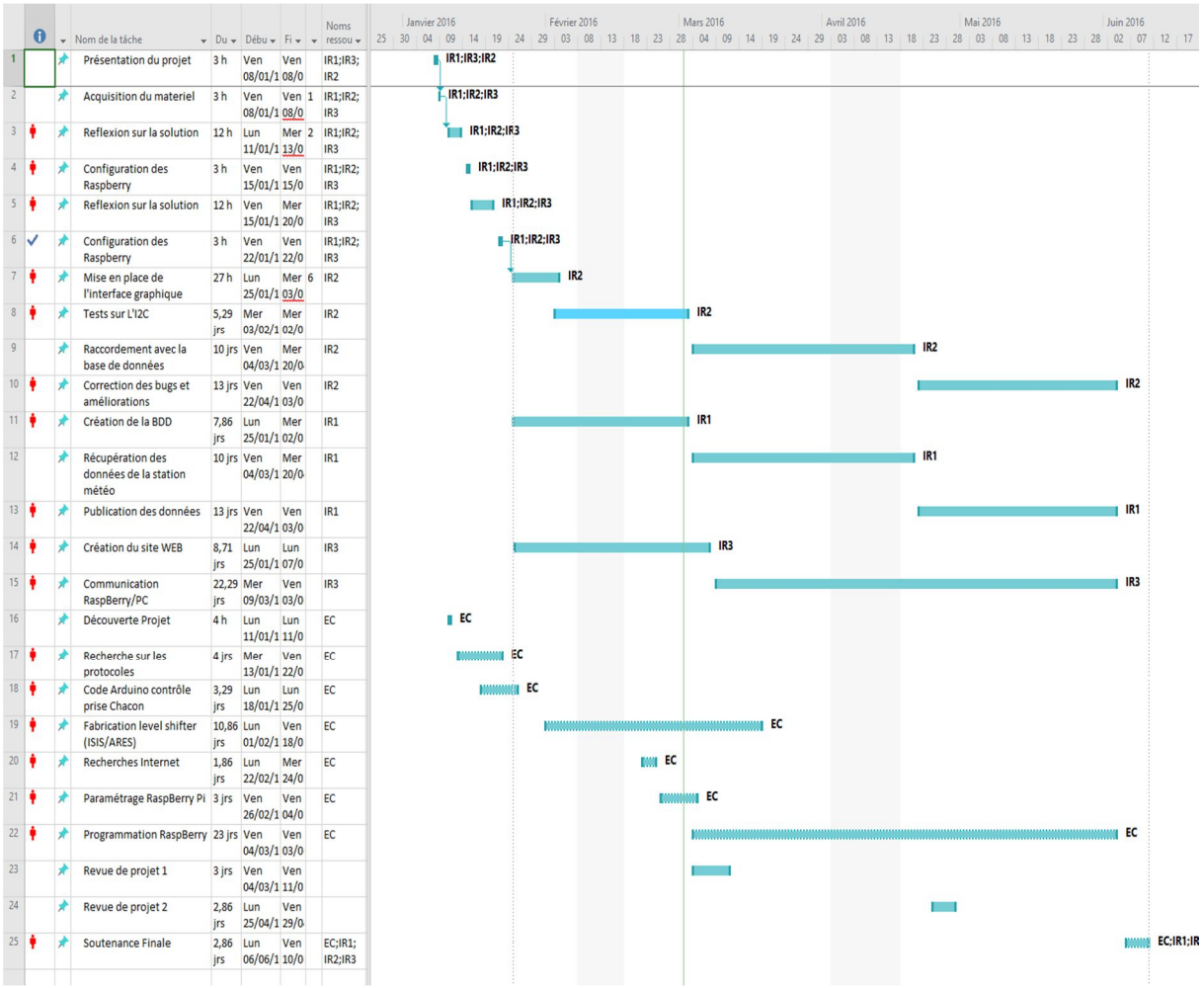
Dans la perspective de prolonger ses activités dans le domaine de la domotique, l'entreprise Mariton a proposé la réalisation d'un système de contrôle à distance de chauffage électrique au lycée Alphonse Benoit.

Le système a pour but de réduire la consommation électrique grâce a une programmation intelligente qui permet à l'utilisateur de contrôler son chauffage à distance et de programmer des plages horaires de fonctionnement. Une régulation automatique de la température doit être réalisée afin de préserver le confort des personnes. Une programmation des radiateurs sera possible par les occupants grâce à un serveur permettant la prise en main du logiciel via n'importe quel ordinateur connecté au réseau. Le logiciel permettra une récupération des donnés météo et une régulation automatique en fonction du désir de l'occupant et de la météo. Il pourra par exemple régler le chauffage pour qu'a 18h, quand il rentrera cher lui, la température soit de 20°C. Le chauffage se mettra donc automatiquement en route pour qu'a 18h la température atteigne 20 °C.

Une base de données des consignes de températures permettra à la régulation de s'adapter à la présence ou non des occupants.

Ce projet comprend des parties logicielles et matériels qui seront mise en place par quatre étudiants en BTS Système Numérique, dont trois en option Informatique et Réseau, et un en option Électronique et Communication.

Planification



	Nom de la tâche	Du	Début	Fin	Noms ressou
1	Présentation du projet	3 h	Ven 08/01/1	Ven 08/0	IR1;IR3; IR2
2	Acquisition du materiel	3 h	Ven 08/01/1	Ven 08/0	1 IR1;IR2; IR3
3	Reflexion sur la solution	12 h	Lun 11/01/1	Mer 13/0	2 IR1;IR2; IR3
4	Configuration des Raspberry	3 h	Ven 15/01/1	Ven 15/0	IR1;IR2; IR3
5	Reflexion sur la solution	12 h	Ven 15/01/1	Mer 20/0	IR1;IR2; IR3
6	Configuration des Raspberry	3 h	Ven 22/01/1	Ven 22/0	IR1;IR2; IR3
7	Mise en place de l'interface graphique	27 h	Lun 25/01/1	Mer 03/0	6 IR2
8	Tests sur L'I2C	5,29 jrs	Mer 03/02/1	Mer 02/0	IR2
9	Raccordement avec la base de données	10 jrs	Ven 04/03/1	Mer 20/0	IR2
10	Correction des bugs et améliorations	13 jrs	Ven 22/04/1	Ven 03/0	IR2
11	Création de la BDD	7,86 jrs	Lun 25/01/1	Mer 02/0	IR1
12	Récupération des données de la station météo	10 jrs	Ven 04/03/1	Mer 20/0	IR1
13	Publication des données	13 jrs	Ven 22/04/1	Ven 03/0	IR1
14	Création du site WEB	8,71 jrs	Lun 25/01/1	Lun 07/0	IR3
15	Communication RaspBerry/PC	22,29 jrs	Mer 09/03/1	Ven 03/0	IR3
16	Découverte Projet	4 h	Lun 11/01/1	Lun 11/0	EC
17	Recherche sur les protocoles	4 jrs	Mer 13/01/1	Ven 22/0	EC
18	Code Arduino contrôle prise Chacon	3,29 jrs	Lun 18/01/1	Lun 25/0	EC
19	Fabrication level shifter (ISIS/ARES)	10,86 jrs	Lun 01/02/1	Ven 18/0	EC
20	Recherches Internet	1,86 jrs	Lun 22/02/1	Mer 24/0	EC
21	Paramétrage RaspBerry Pi	3 jrs	Ven 26/02/1	Ven 04/0	EC
22	Programmation RaspBerry	23 jrs	Ven 04/03/1	Ven 03/0	EC
23	Revue de projet 1	3 jrs	Ven 04/03/1	Ven 11/0	
24	Revue de projet 2	2,86 jrs	Lun 25/04/1	Ven 29/0	
25	Soutenance Finale	2,86 jrs	Lun 06/06/1	Ven 10/0	EC;IR1; IR2;IR3

Partie de l'IR 1 :

DELIGNY Dylan

Récupération des données de la station météo et stockage dans la base de données

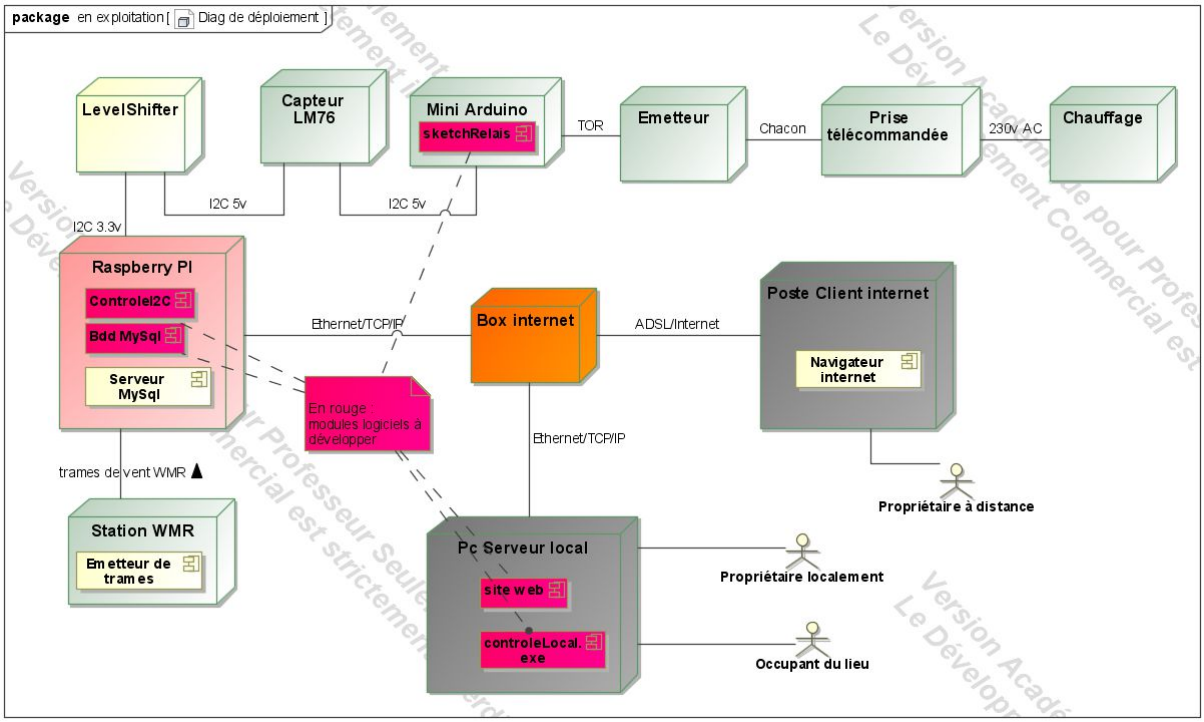
Introduction

1. Présentation du projet :

Le projet Mariton Chauffage consiste à aider l'entreprise à acquérir de l'autonomie dans la fourniture de matériels et logiciels qui serviront à améliorer leurs produits déjà existants.

Dans notre projet notre travail consiste à gérer le système de chauffage d'une maison ou d'un local. Cela permettra de réduire la consommation tout en conservant le confort de ces occupants. Une base de données est intégrée dans ce projet afin de paramétrer des consignes de température. Ces consignes servent à réguler la température afin de s'adapter à la présence ou non des occupants.

Diagramme de déploiement :



Présentation de ma partie personnelle

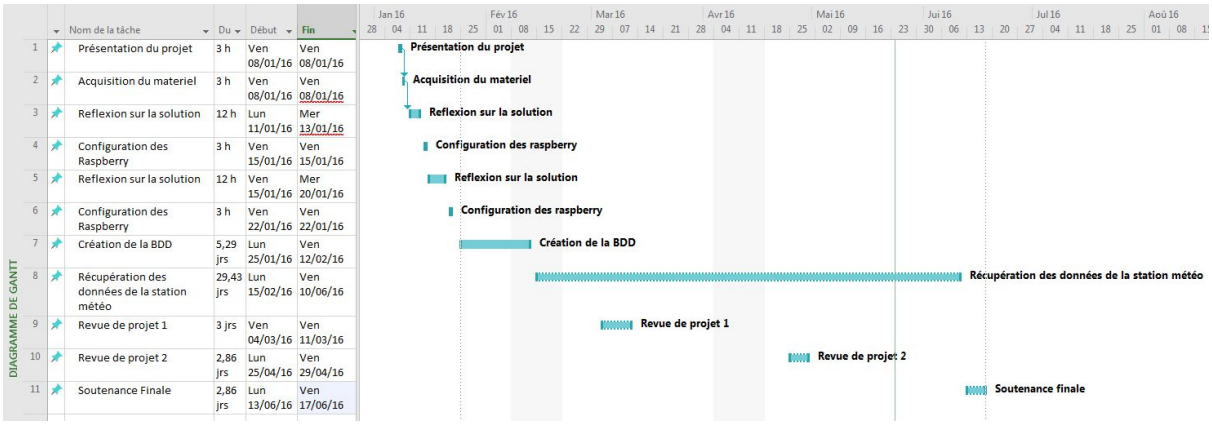
1.1 - Travail à effectuer et matériel mis à disposition :

Étudiant 1 IR 1	*Liste des fonctions assurées par l'étudiant* Récupération des informations de la station météo, stockage dans la Base de données et publication.	Installation : Qt, MySql, Apache, i2c-dev. Mise en œuvre : MySql, Apache, i2c-dev. Configuration : MySql, Apache. Réalisation : Communication avec la station météo, stockage et publication des données. Documentation : Installation, mise en service, dossier de développement.

Afin de réaliser ces tâches j'ai à disposition :

- Une RaspBerry PI
- Une station météo Oregon WMR928NX
- Adaptateur RS232/USB afin de pouvoir connecter la station météo à la RaspBerry.

1.2 - Diagramme de GANTT réel :

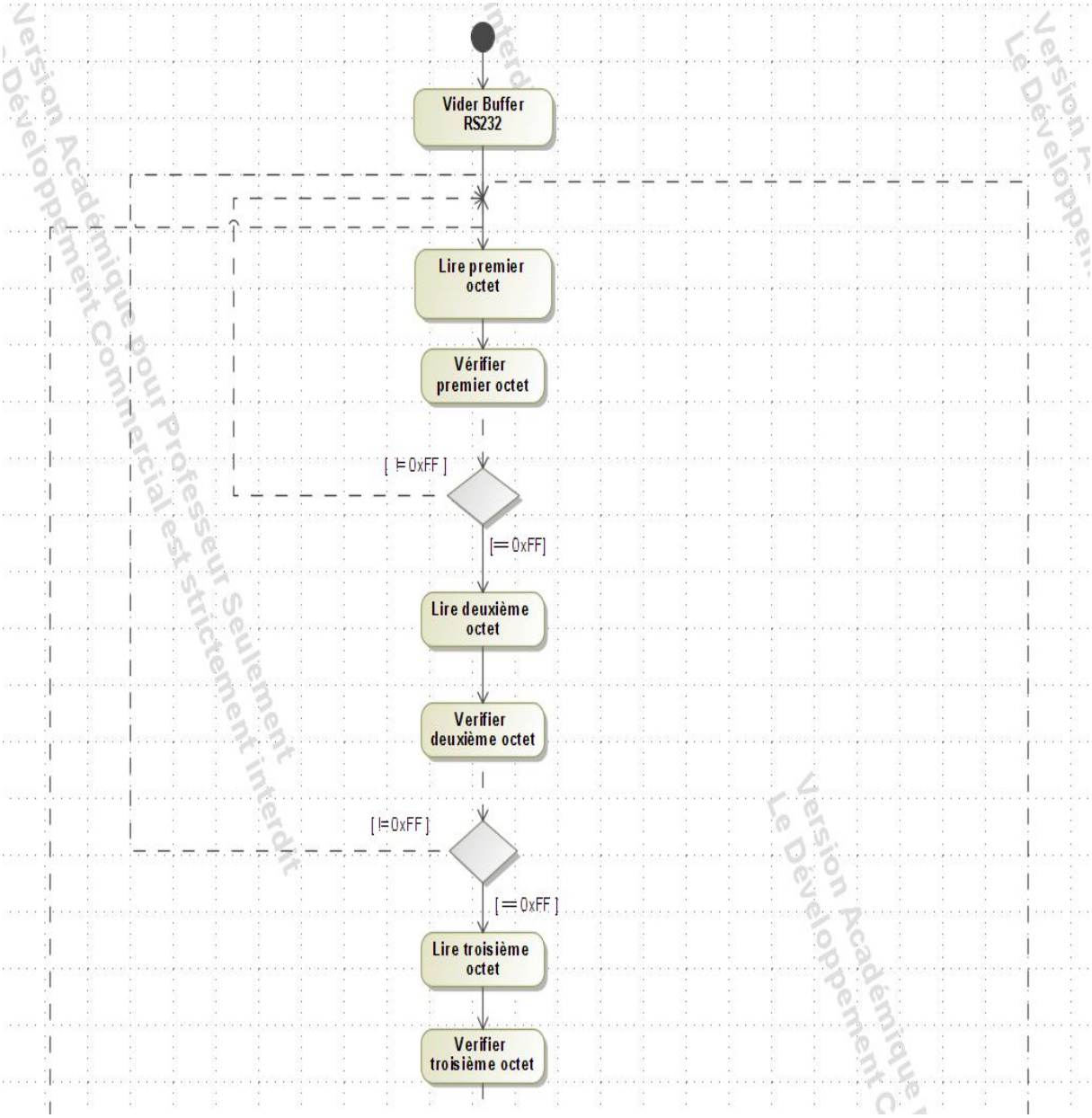


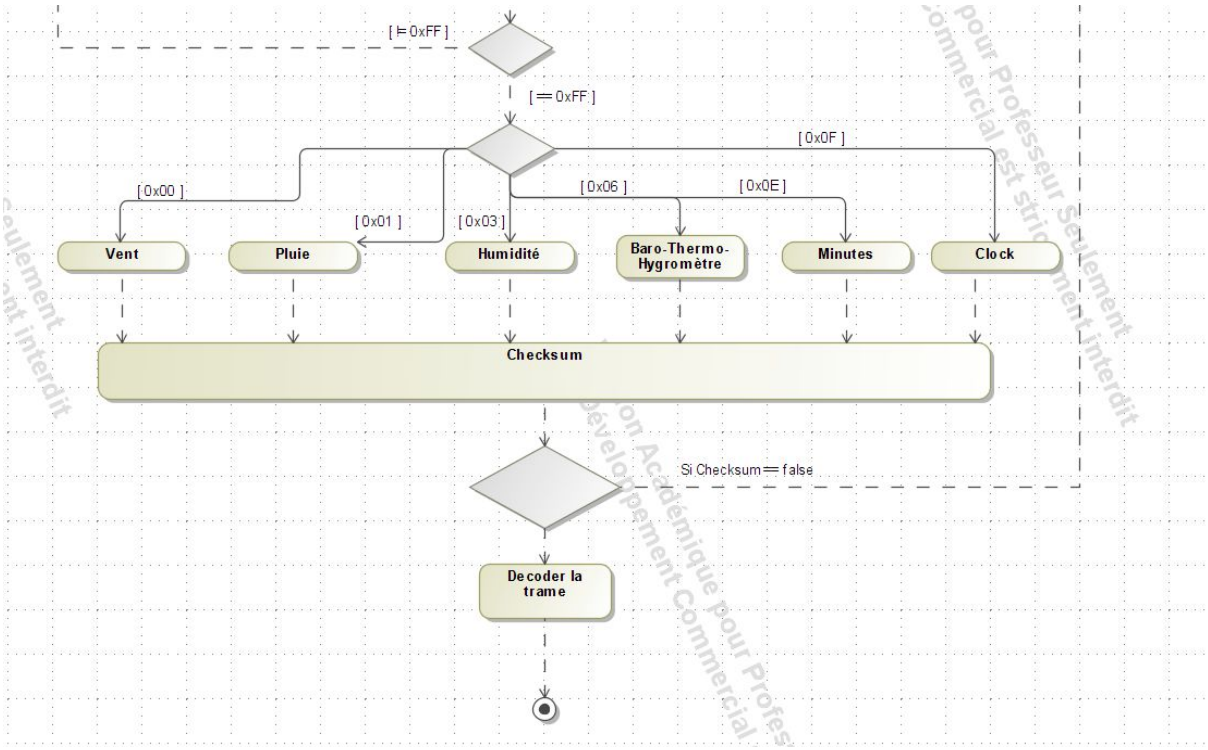
Nous pouvons constater qu'il y a des différences avec le diagramme de GANTT prévisionnel.

En effet, j'ai pris du retard sur la tâche « Récupération des données de la station météo ». J'ai eu des difficultés à concevoir le code permettant de recevoir et décoder les trames émises par la station météo.

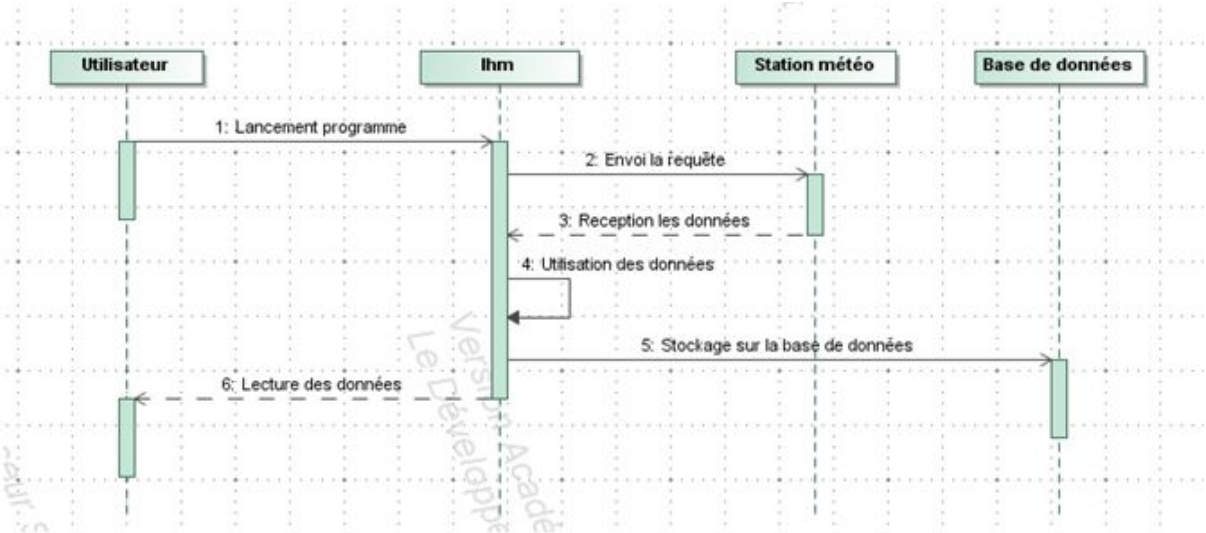
Partie SysML/UML

1.1 - Diagramme d'activité

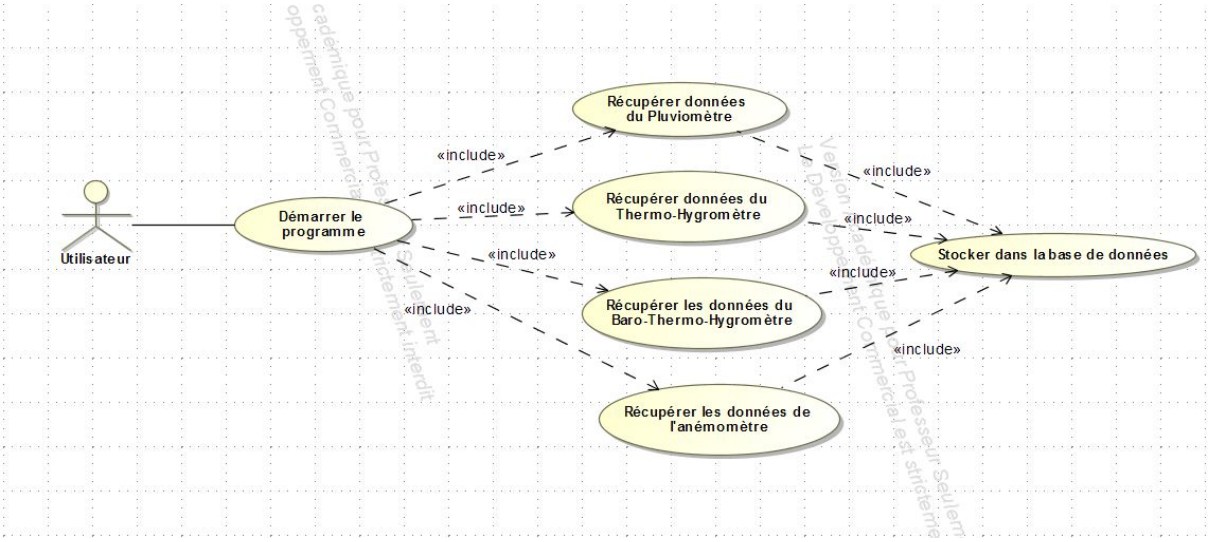




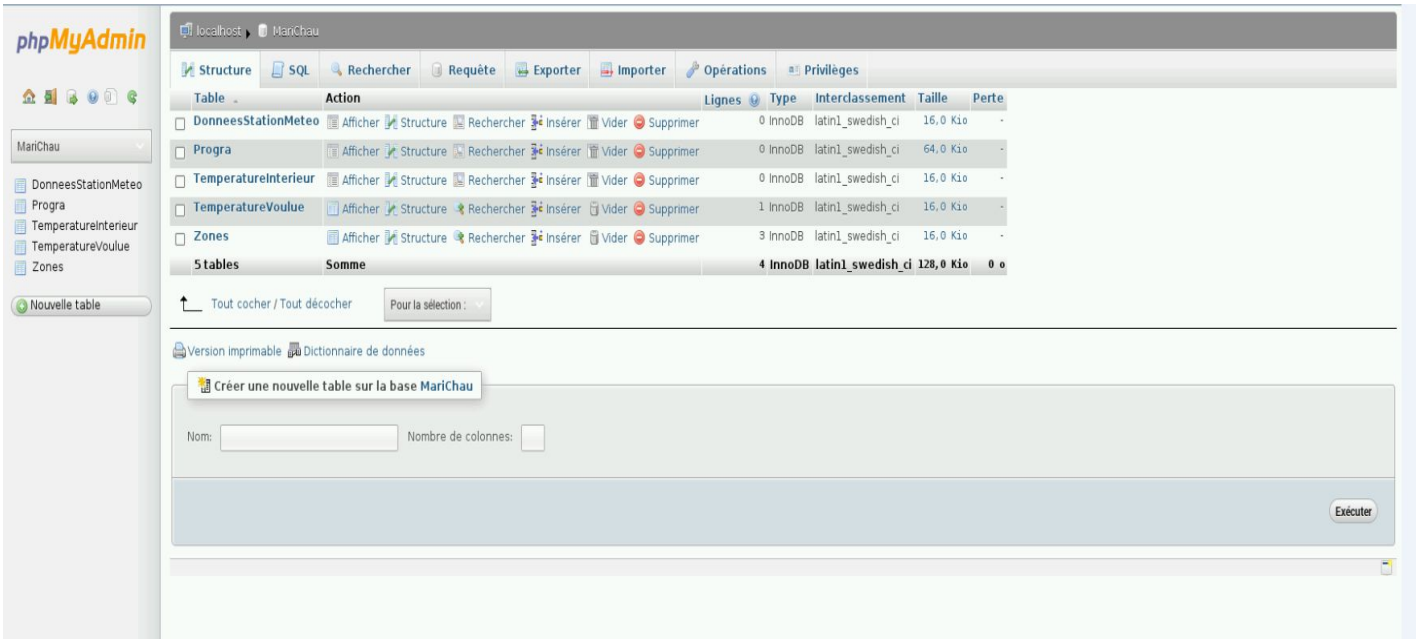
1.2 - Diagramme de séquence :



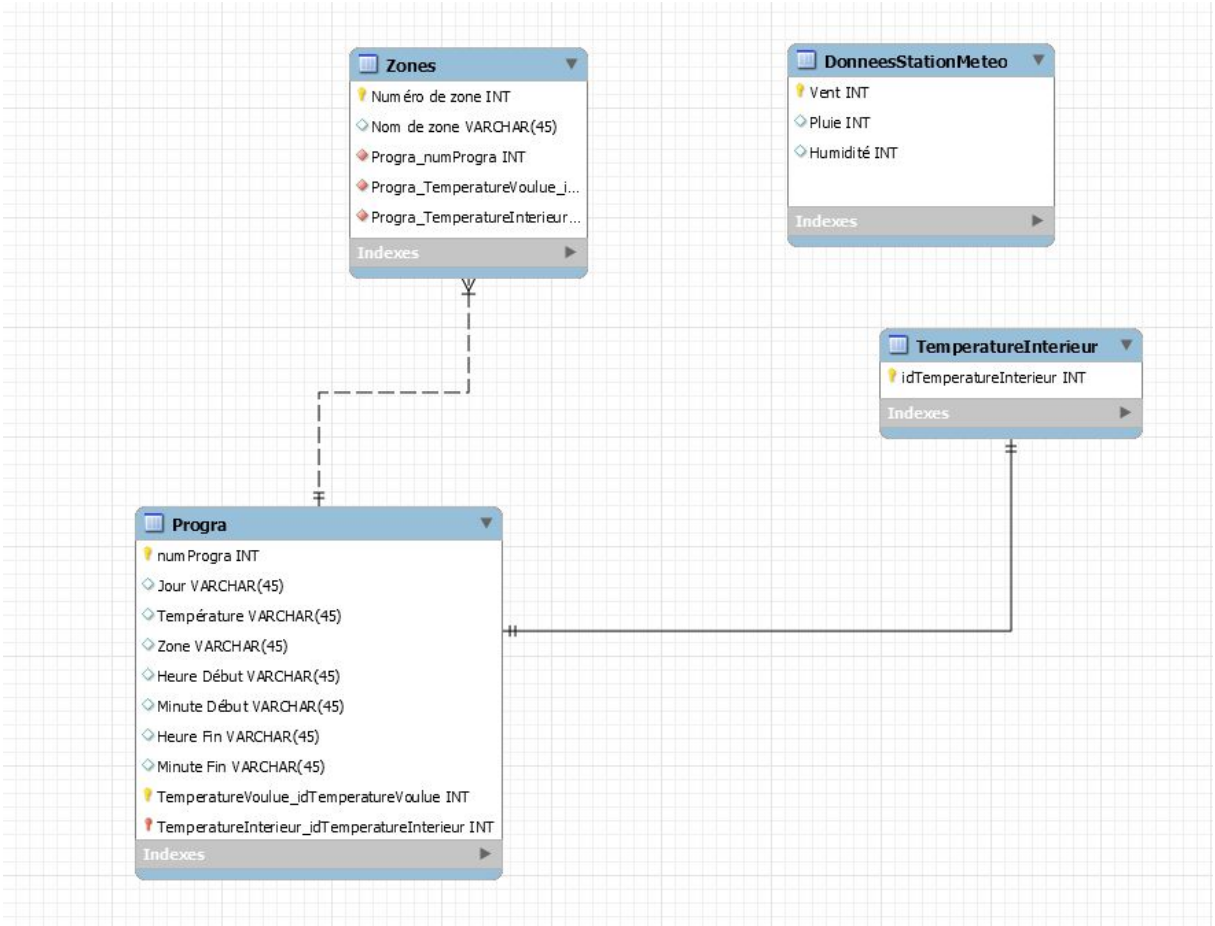
1.3 - Diagramme de cas d'utilisation :



Base de données



1.1 - Modélisation de la base de données :



1.2 - Explications du contenu des tables :

Table Zone : Contient le nom des zones. Un numéro est attribué aux zones.

La zone numéro 1 est le salon.

La zone numéro 2 est la chambre.

La zone numéro 3 est la salle de bain.

Table DonneesStationMeteo : Contient les données envoyées par la station météo.

Cette table est composée de 3 colonnes : Vent, Pluie et Humidité.

Table TemperatureInterieur : Contient la valeur récupérée par le capteur LM76.

Table Progra : Contient les programmations effectuées par l'occupant de la maison.

Elle se remplira à l'aide du programme qui permet de configurer les programmations. Cette table est composée de plusieurs colonnes :

numProgra → C'est le numéro de la programmation. En effet il se peut qu'il y est plusieurs programmations.

Jour → C'est le jour où la programmation démarrera.

Temperature → C'est la température souhaitée par l'occupant de la maison pour la programmation qu'il est entrain de configurer.

Heure Début / Heure Fin → C'est l'heure à laquelle la programmation démarrera et s'arrêtera.

Minute Début / Minute Fin → C'est la minute à laquelle la programmation démarrera et s'arrêtera.

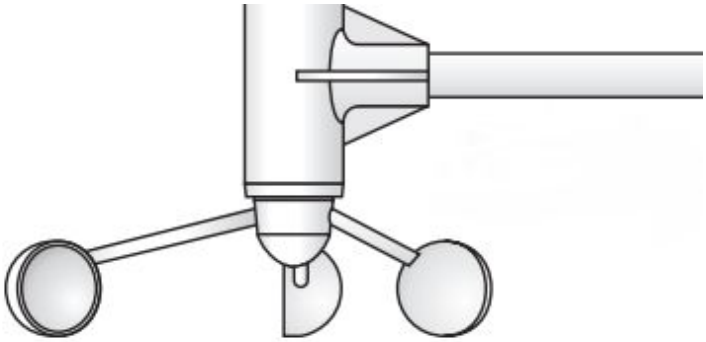
La station météo

1.1 Présentation de la station météo :

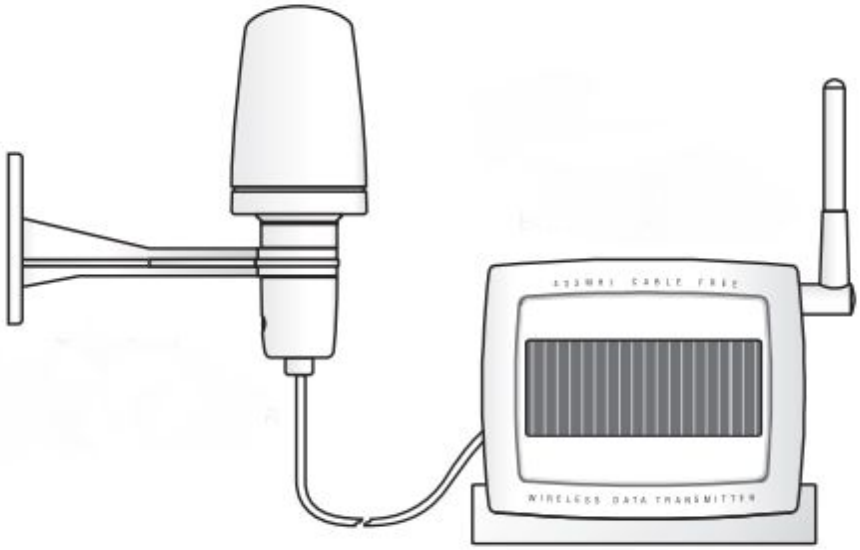
La station météo est une Oregon WMR928NX. Elle est composée de 4 capteurs et d'un appareil principal.

L'appareil principal : il indique tous les relevés et comprends toutes les commandes.

L'anémomètre : Il relève la vitesse et la direction du vent.



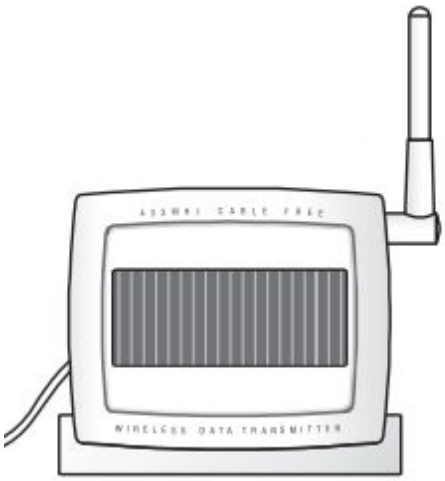
Le thermo-hygromètre : Il relève la température et l'humidité extérieures.



Le Baro-Thermo-Hygromètre : Il relève la pression atmosphérique, la température et l'humidité.

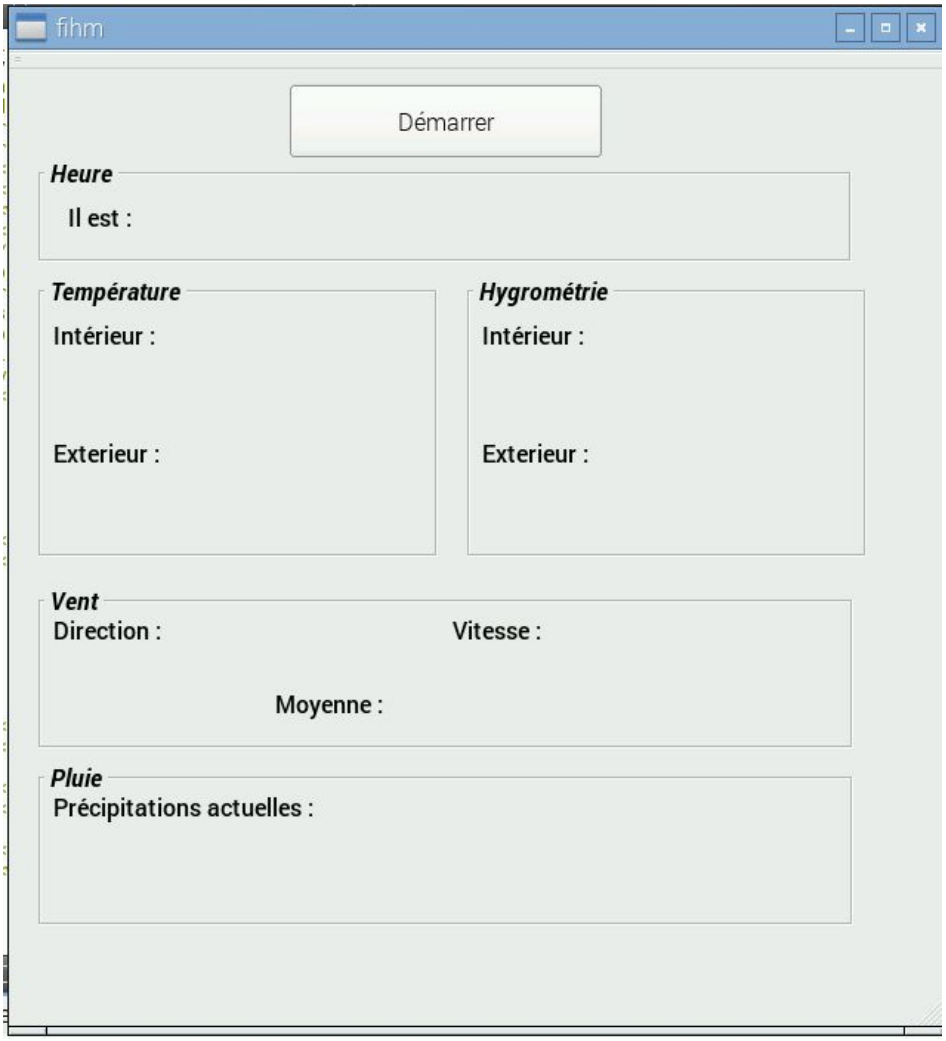
Le Pluviomètre : Il relève la quantité totale et le débit des précipitations.

Le thermo-hygromètre et le pluviomètre sont alimentés par des transmetteurs solaires, ils utilisent l'énergie solaire pour alimenter les instruments auxquels ils sont connectés.



La station météo fonctionne sur une transmission sans fil en 433 MHz et à une portée effective de 100 mètres.

1.2 Codage de la réception des trames de la station météo



Dans l'IHM (Interface Homme Machine) nous trouverons les données que l'on reçoit de la station météo.

```
//---------------------Initialisation port série---------------------//

    void fihm::init()
    {

        serial->setPortName(PORT);
        serial->setBaudRate(QSerialPort::Baud9600);
        serial->setDataBits(QSerialPort::Data8);
        serial->setParity(QSerialPort::Parity);
        serial->setStopBits(QSerialPort::OneStop);
        serial->setFlowControl(QSerialPort::NoFlowControl);
        serial->open(QSerialPort::ReadWrite);

    }
```

Cette partie de code sert à initialiser la communication du port série.

Baud9600 → Indique la vitesse de transmission.

Data 8 → Indique qu'il y a 8 bits de données. La station météo envoie ses valeurs sur 8 bits.

Parity → indique qu'il y a un bit de parité. Ce bit sert à vérifier si la trame est correcte.

OneStop → Indique qu'il y a un bit de stop.

NoFlowControl → Indique qu'il n'y a pas de contrôle de flux. Le contrôle de flux sert à diminuer le débit d'une machine au cas où son débit montant est supérieur au débit descendant de la destination.

ReadWrite → indique qu'on peut lire et écrire.

Cet exemple du code permet de récupérer la trame de pluie.

```
case 1:                                    //Rain
    i+=2;
    rainCurrent += (data[i] & 0x01);
    rainCurrent += (data[i] >> 4)*10;
    i++;
    rainCurrent += (data[i] & 0x01)*100;
    ui->lnedRainCurrent->setText(QString::number(rainCurrent) + " mm/h");
    break;
```

Partie physique

1. Présentation *de la liaison RS232*

Le transfert des données s'effectue par liaison série RS232.

Les modes de transmission de cette liaison sont :

- Simplex : Mode de transmission unidirectionnel
- Semiduplex (ou half-duplex) : Mode de transmission bidirectionnel, mais un seul dispositif peut émettre à la fois.
- Full-duplex : Mode de transmission bidirectionnel. Les deux dispositifs émettent en même temps.

Afin d'établir une communication effective il faut définir le protocole à utiliser, la communication est généralement présentée comme cela :

- 1 bit de départ (niveau bas)
- Le bit de poids faible de l'octet transmis est envoyé en premier
- 7 à 8 bits de données
- 1 bit de parité (optionnel)
- 1 ou de bits d'arrêt (stop)

La vitesse ou débit de transmission s'exprime en Bauds ou bits par seconde.

Plusieurs vitesses sont disponibles : 19200, 9600, 4800 et 2400 bauds. Dans notre cas on utilise une vitesse de 9600 bauds.

Ma Raspberry PI est connecté à la station météo à l'aide d'un adaptateur Port série / USB.

1.2 Antennes et longueur d'onde :

La station WMR928NX fonctionne sur une fréquence de 433 MHz.

La longueur d'onde est la distance parcourue par l'onde pendant une période d'oscillation.

Cela donne une longueur d'onde de 69.3 cm. On utilise une antenne quart d'onde de 17.3 cm.

La longueur se calcule : lambda = Vitesse de l'onde / la fréquence.

= 9600 / 433 MHz

= 69.3 cm

Une antenne quart d'onde est constituée d'un élément de longueur égale au quart de la longueur d'onde.

69.3 cm / 4

= 17.3 cm

Nous avons donc une antenne quart d'onde de 17.3 cm.

Une antenne quart d'onde est une antenne qui ne comporte qu'un seul brin rayonnant associé à un plan de masse. Cette antenne se comporte aussi comme un circuit R,L,C série.

A la résonance elle a une impédance moitié de celle du dipôle soit environ 36 ohms.

Conclusion

Ayant quelques lacunes en programmation j'ai rencontré quelques difficultés à réaliser ce projet.

J'ai démarré ce projet par la conception de ma base de données, après quelques formations m'apportant des rappels sur des connaissances que j'avais sur ce sujet, j'ai pu bien concevoir ma base de données.

Arrivé à la partie programmation je me suis retrouvé un peu perdu dans ma démarche pour mener mon projet à son terme. Un professeur référant un peu absent ne m'a pas forcément aidé à progresser et avancer plus vite dans mon projet. C'est pour cela que j'ai pris un peu de retard.

Au sein de mon groupe de projet l'ambiance de travail était très agréable, on a su s'aider lorsqu'une personne avait besoin d'aide et lors de diverses réunions nous arrivions très vite à se mettre d'accord.

Partie de l'IR 2 :

LALY Maxime

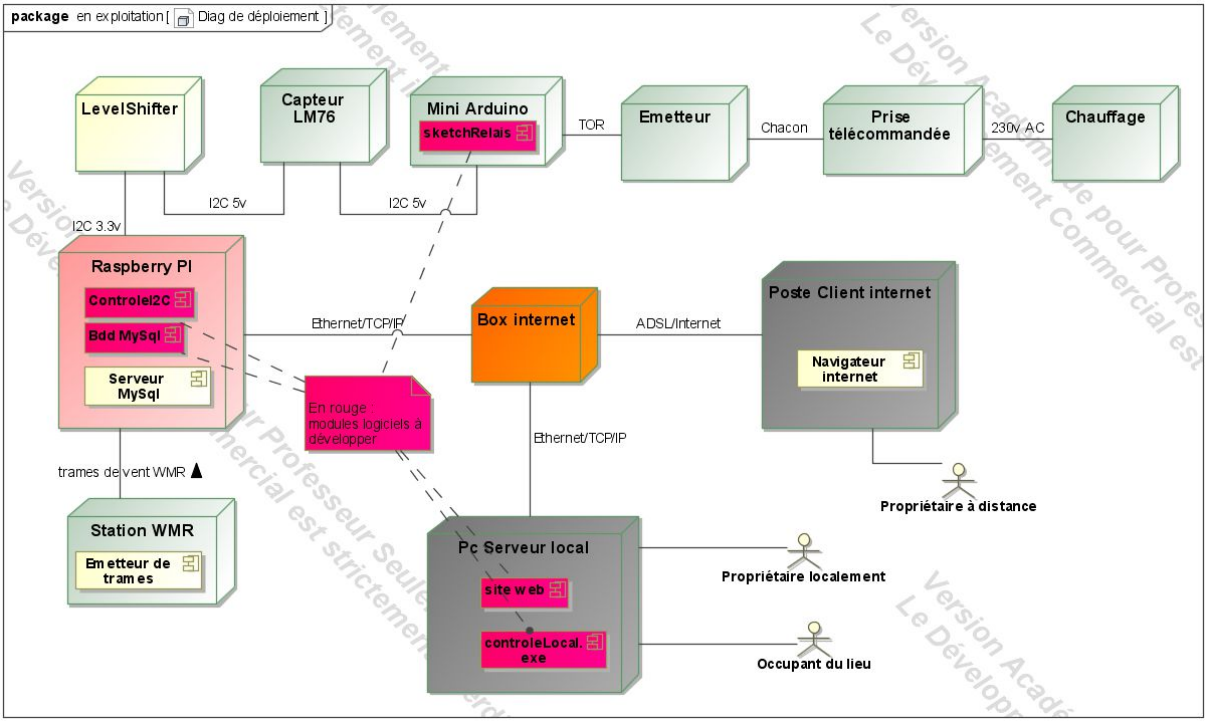
Gestion des programmations de température

Régulation de la température

Gestion du capteur LM76

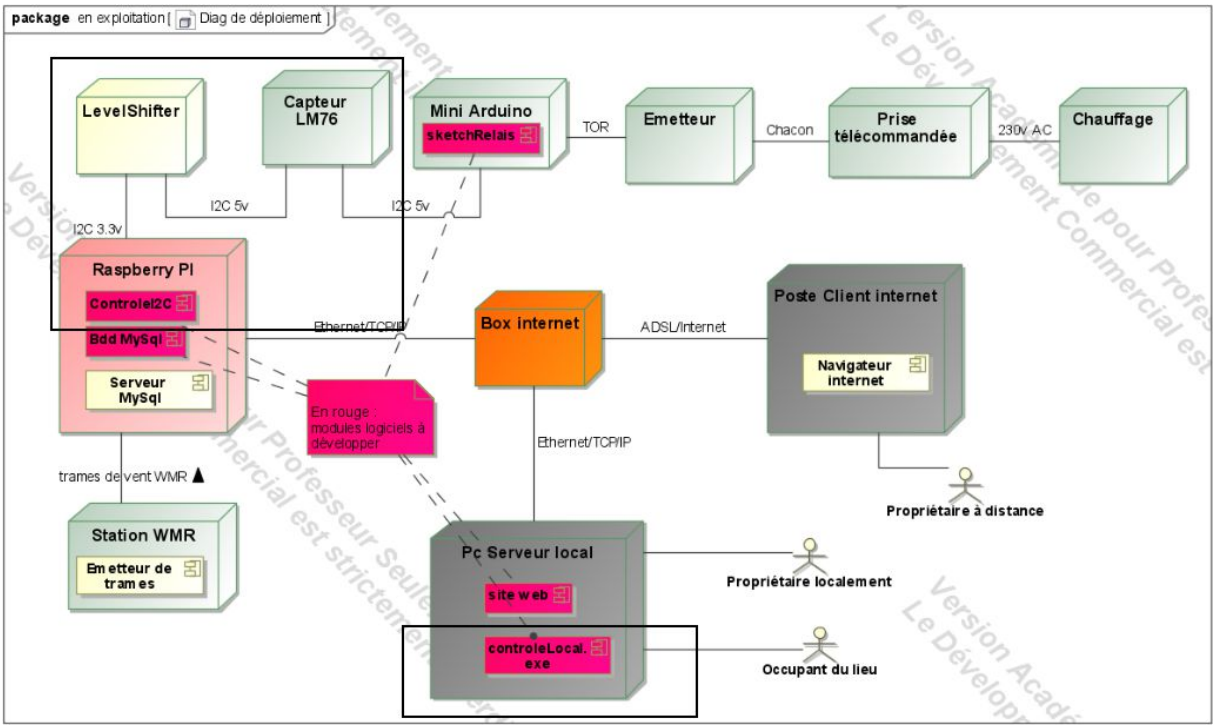
3.1. Présentation du projet

L'entreprise Mariton veut prolonger son activité dans le domaine de la domotique. Pour ce faire, elle veut pouvoir gérer le chauffage électrique d'une maison ou d'un local quelconque afin de réduire la consommation tout en préservant le confort des occupants lorsque ceux-ci sont présents. Une base de données des consignes de températures permettra à la régulation de s'adapter à la présence ou non des occupants.



3.2. Travail de l'étudiant

La partie du projet dont je m'occupe consiste en la gestion des programmations, la gestion des chauffages et la récupération des données du capteur de température.

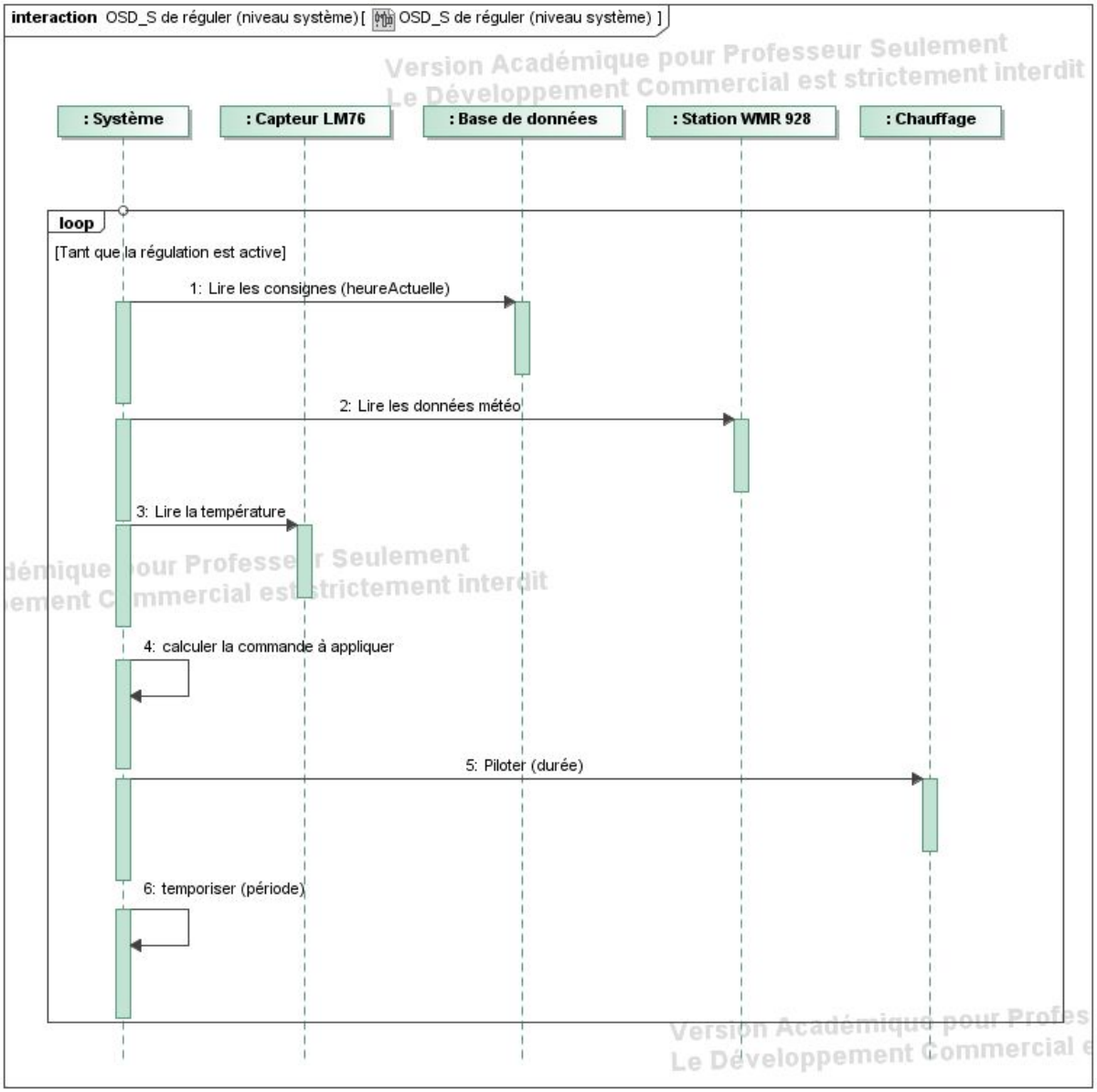


3.2.1. Planning Réel

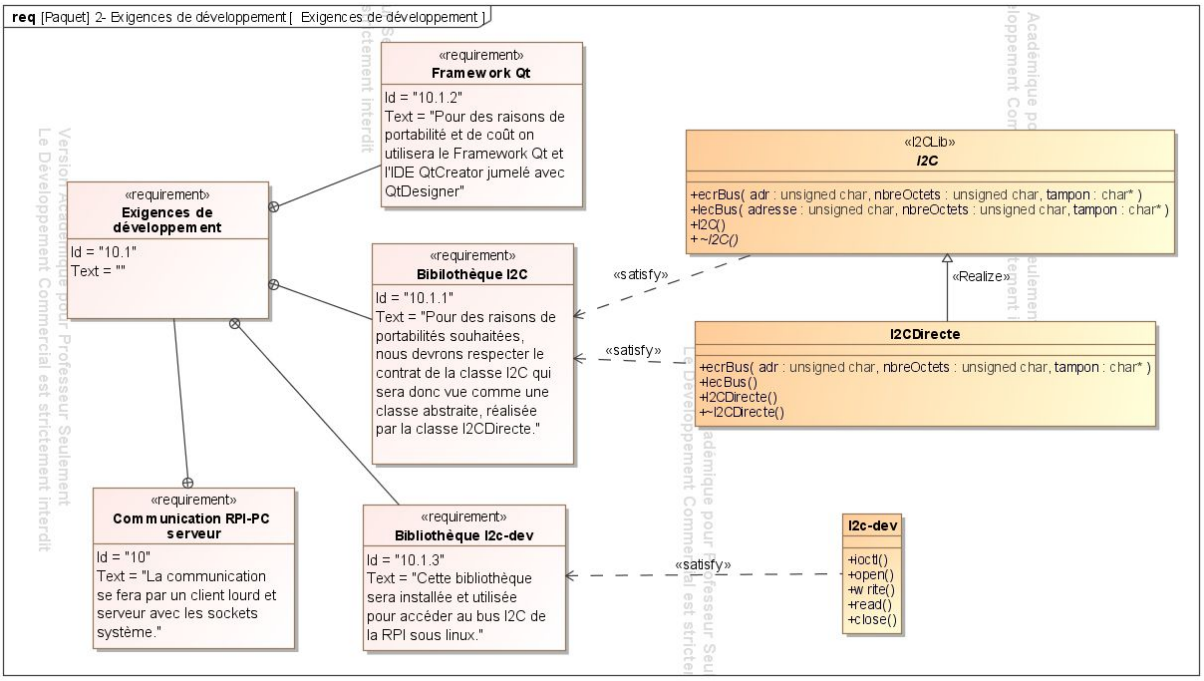
			Nom de la tâche	Du	Débu	Fin	Prédéce	Noms ressources
1			Présentation du projet	3 h	Ven 08/01/1	Ven 08/01/16		IR1;IR3;IR2
2			Acquisition du materiel	3 h	Ven 08/01/1	Ven 08/01/16	1	IR1;IR2;IR3
3			Reflexion sur la solution	12 h	Lun 11/01/1	Mer 13/01/16	2	IR1;IR2;IR3
4			Configuration des Raspberry	3 h	Ven 15/01/1	Ven 15/01/16		IR1;IR2;IR3
5			Reflexion sur la solution	12 h	Ven 15/01/1	Mer 20/01/16		IR1;IR2;IR3
6			Configuration des Raspberry	3 h	Ven 22/01/1	Ven 22/01/16		IR1;IR2;IR3
7			Mise en place de l'interface graphique	27 h	Lun 25/01/1	Mer 03/02/16	6	IR2
8			Développement du programme de gestion	8,29 jrs	Jeu 04/02/1	Ven 11/03/16		
9			Raccordement avec la base de données	5,86 jrs	Lun 14/03/1	Mer 30/03/16		
10			Tests sur L'I2C	2,71 jrs	Jeu 31/03/1	Lun 04/04/16		IR2
11			Développement du test du jour programmé	7 jrs	Mer 06/04/1	Jeu 05/05/16		IR2
12			Développement de l'algorithme de chauffe	11 jrs	Ven 06/05/1	Ven 10/06/16		IR2
13			Revue de projet 1	3 jrs	Ven 04/03/1	Ven 11/03/16		
14			Revue de projet 2	2,86 j	Lun 25/	Ven 29/04/16		EC;IR1;IR2;IR3
15			Revue de projet finale	1,29 j	Lun 13/	Lun 13/06/16		EC;IR1;IR2;IR3

3.3. Diagrammes SysML

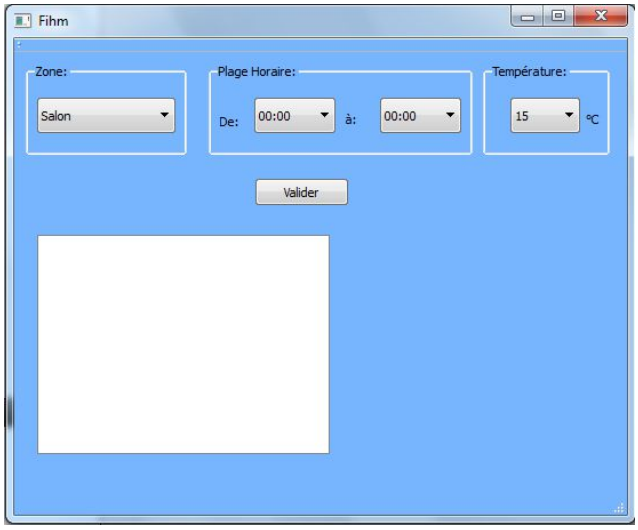
Diagramme de séquence de la régulation



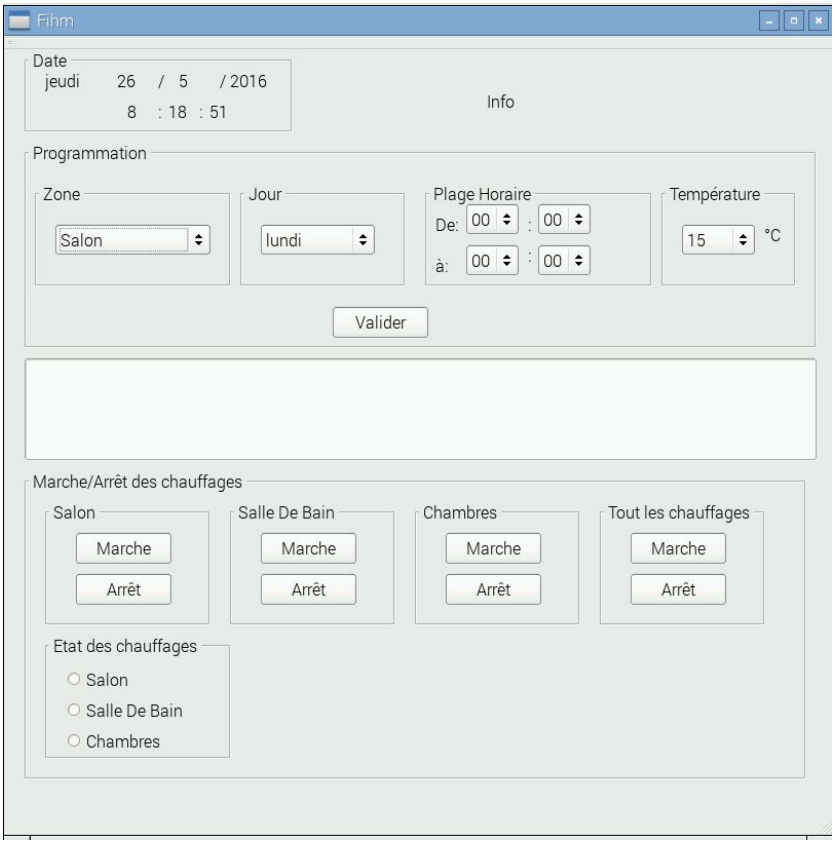
Contraintes de développement



3.4. Programme



Cette interface graphique est la première interface que j'ai créée. Suite à quelques modifications, je l'ai modifiée pour arriver à celle-ci :



Cette interface graphique permet de rentrer une température en fonction des zones, du jour de la semaine et d'une plage horaire programmable à la minute près. Pour plus de détail se référer au mode d'emploi page 15. Elle permet aussi de se renseigner sur la date et l'heure actuelle.

3.4.1. Code

Au lancement du programme, la connexion avec la base de données se fait.

```
17
18      /*____________________Connexion BDD_____________*/
19
20      bdd = QSqlDatabase::addDatabase("QMYSQL");
21      bdd.setHostName("localhost");
22      bdd.setDatabaseName("test");
23      bdd.setUserName("root");
24      bdd.setPassword("pipod");
25
26      /*____________________________________________________*/
27
28
```

Quand l'utilisateur appuie sur le bouton « Valider », le contenu des comboBox se stocke dans des QString :

```
47
48
49      /*_____________Stockage des ComboBox___________*/
50
51      zone = ui->cbZone->currentText();
52      jour = ui->cbJour->currentText();
53      heureDebut = ui->cbHeure1->currentText();
54      heureFin = ui->cbHeure2->currentText();
55      minuteDebut = ui->cbMin1->currentText();
56      minuteFin = ui->cbMin2->currentText();
57      temperature = ui->cbTemp->currentText();
58
59      /*____________________________________________________*/
60
```

Ces QString sont ensuite envoyées dans la base de données grâce à une requête SQL.

```
63
64  /*_____________________________________Requete SQL pour stocker la programmation dans la bdd_____________________
65      bdd.open();
66      QSqlQuery requeteur;
67
68
69      if( requeteur.prepare("INSERT INTO Progra VALUES (:numProgra, :jour, :temperature, :zone, :heureDebut, :mi
70      {
71          requeteur.bindValue(":numProgra", numProgra);
72          requeteur.bindValue(":jour", jour);
73          requeteur.bindValue(":temperature", temperature);
74          requeteur.bindValue(":zone", zone);
75          requeteur.bindValue(":heureDebut", heureDebut);
76          requeteur.bindValue(":minuteDebut", minuteDebut);
77          requeteur.bindValue(":heureFin", heureFin);
78          requeteur.bindValue(":minuteFin", minuteFin);
79          requeteur.exec();
80
81          ui->leUtilisateur->setText("Programmation enregistree dans la base de donnees.");
82      }
83      else
84      {
85          ui->leUtilisateur->setText("Erreur dans la programation..");
86      }
87
88  /*____________________________________________________________________________________________________________
89
90
```

L'IHM permettant également de se renseigner sur l'heure et la date actuelle, j'ai codé cela grâce aux classes QDate et QString.

```
 99
100  /*____________________Mise à jour de la date et de l'heure____________________*/
101
102
103      QDate a = QDate();
104      a = QDate::currentDate();
105      int nbj = a.dayOfWeek();
106      jourReel = QDate::longDayName(nbj);
107      ui->lbJour->setText(jourReel);
108
109      QDate e = QDate();
110      e = QDate::currentDate();
111      int nbd = e.day();
112      ui->lbDate->setText(QString::number(nbd));
113
114      QDate f = QDate();
115      f = QDate::currentDate();
116      int nbMo = f.month();
117      ui->lbMois->setText(QString::number(nbMo));
118
119      QDate g = QDate();
120      g = QDate::currentDate();
121      int nby = g.year();
122      ui->lbAnnee->setText(QString::number(nby));
123
124      QTime b = QTime();
125      b = QTime::currentTime();
126      int nbh = b.hour();
127      ui->lbHeure->setText(QString::number(nbh));
128
129      QTime c = QTime();
130      c = QTime::currentTime();
131      int nbm = c.minute();
132      ui->lbMinute->setText(QString::number(nbm));
133
134      QTime d = QTime();
135      d = QTime::currentTime();
136      int nbs = d.second();
137      ui->lbSeconde->setText(QString::number(nbs));
138
139
```

3.4.2 I2C

J'ai commencé à travailler sur ce capteur en créant la bibliothèque I2C-dev, mais ce capteur a malheureusement été endommagé donc je n'ai pas pu continuer mon travail et je suis passé sur une autre tâche.

3.5. Base de données

Cette base de données est la base de données de test. Je l'ai créée pour tester la connexion de mon programme QT à cette base de données, et pour tester le stockage des programmations.

#	Column	Type	Collation	Attributes	Null	Default	Extra	Action
1	numéro de programmation	int(5)			No	None		Change Drop More
2	Température	varchar(5)	latin1_swedish_ci		No	None		Change Drop More
3	Zone	varchar(15)	latin1_swedish_ci		No	None		Change Drop More
4	Horaire Début	varchar(10)	latin1_swedish_ci		No	None		Change Drop More
5	Horaire Fin	varchar(10)	latin1_swedish_ci		No	None		Change Drop More

J'ai par la suite modifié cette base de données de test pour prendre en compte le fait de pouvoir prendre en compte la programmation à la minute près. J'ai pour cela remplacer les colonnes « horaire début » et « horaire fin » par les colonnes « heure début », « minute début », « horaire fin » et « minute fin ».

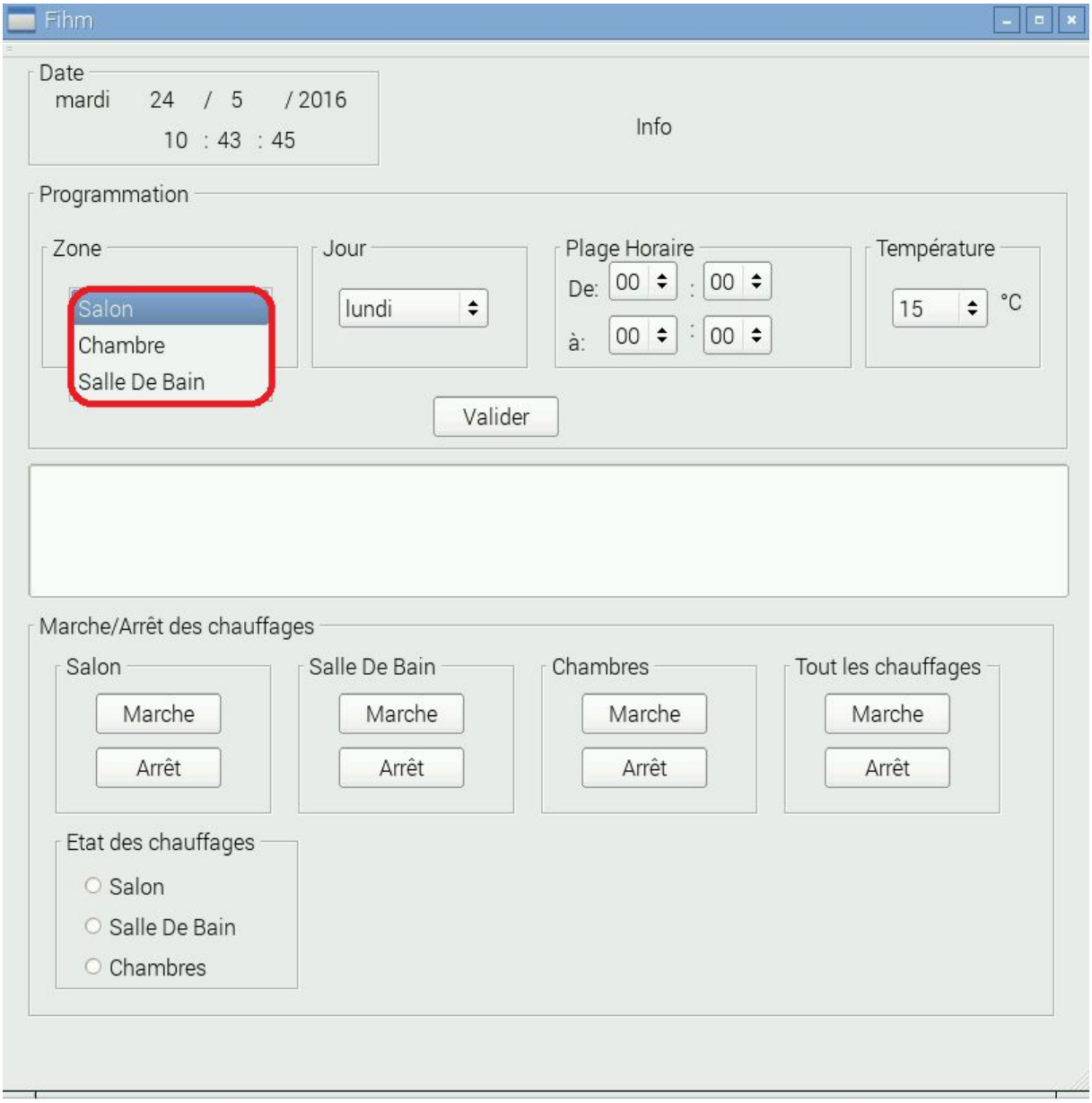
#	Column	Type	Collation	Attributes	Null	Default	Extra	Action
1	numéro de programmation	int(5)			No	None		Change Drop More
2	Jour	varchar(20)	latin1_swedish_ci		No	None		Change Drop More
3	Température	varchar(10)	latin1_swedish_ci		No	None		Change Drop More
4	Zone	varchar(15)	latin1_swedish_ci		No	None		Change Drop More
5	Heure Début	int(10)			No	None		Change Drop More
6	Minute Début	int(5)			No	None		Change Drop More
7	Heure Fin	int(10)			No	None		Change Drop More
8	Minute Fin	int(11)			No	None		Change Drop More

J'ai également créé une table « Température » pour stocker les données du capteur de température LM76. Cette table ne comprend que 2 colonnes : date et température, ce qui permet de faire un historique des températures.

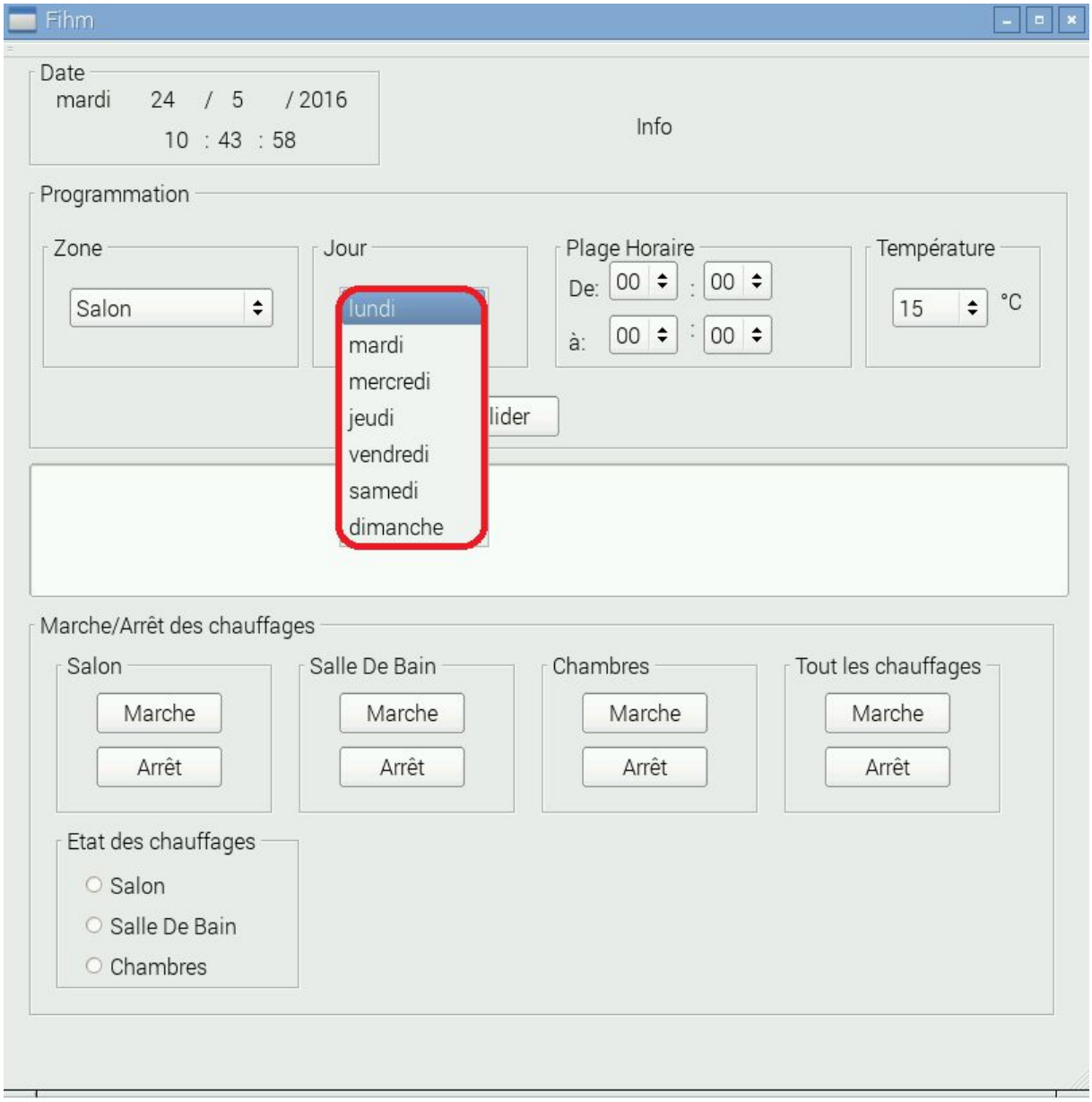
3.6. Mode d'emploi

Application

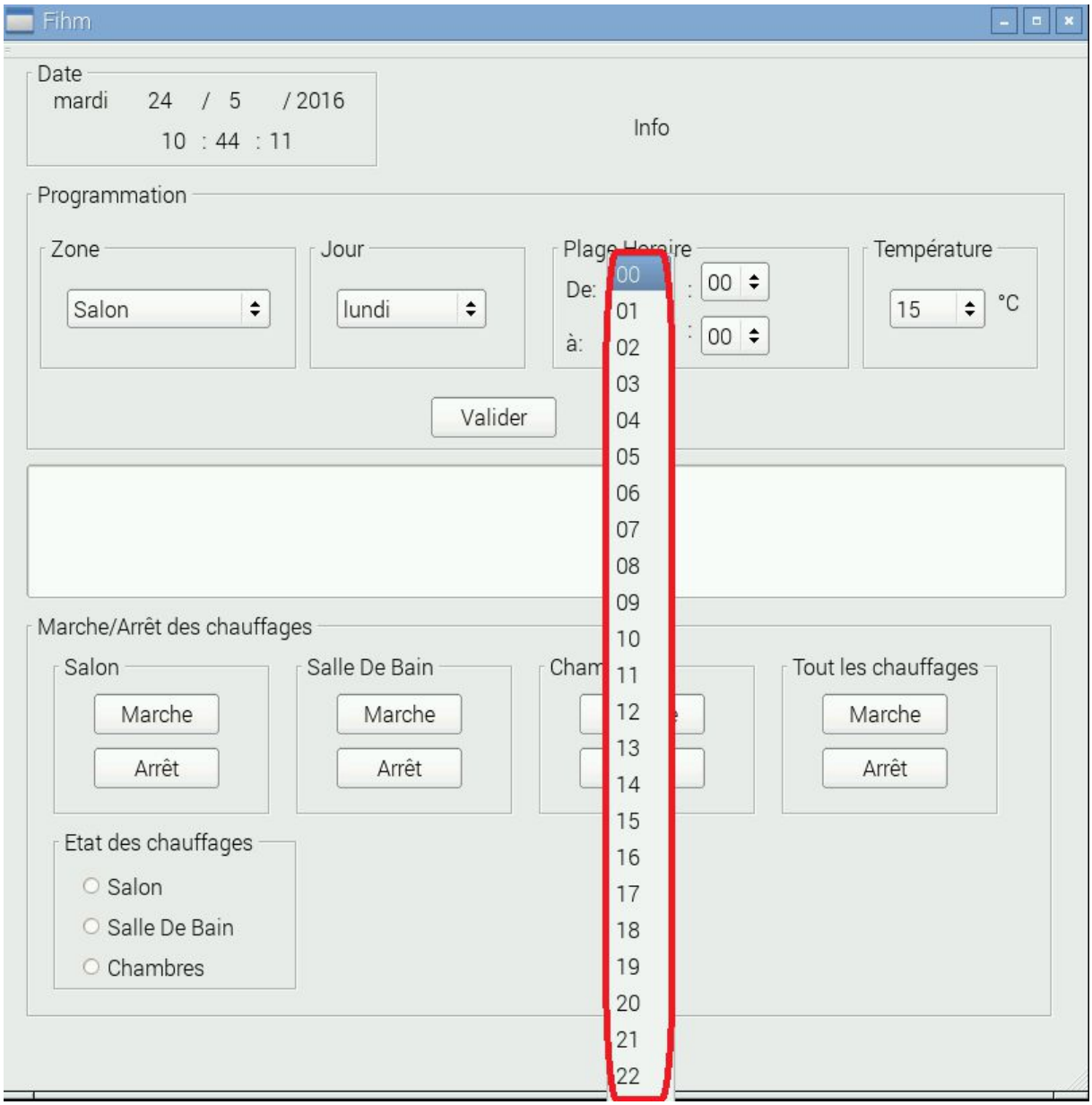
1. Double-cliquer sur l'icône « Mariton » sur le bureau.
   Cela vous ouvre le programme permettant de gérer votre chauffage.
2. Cliquer sur la boite de dialogue « zones » et sélectionner la zone voulue.



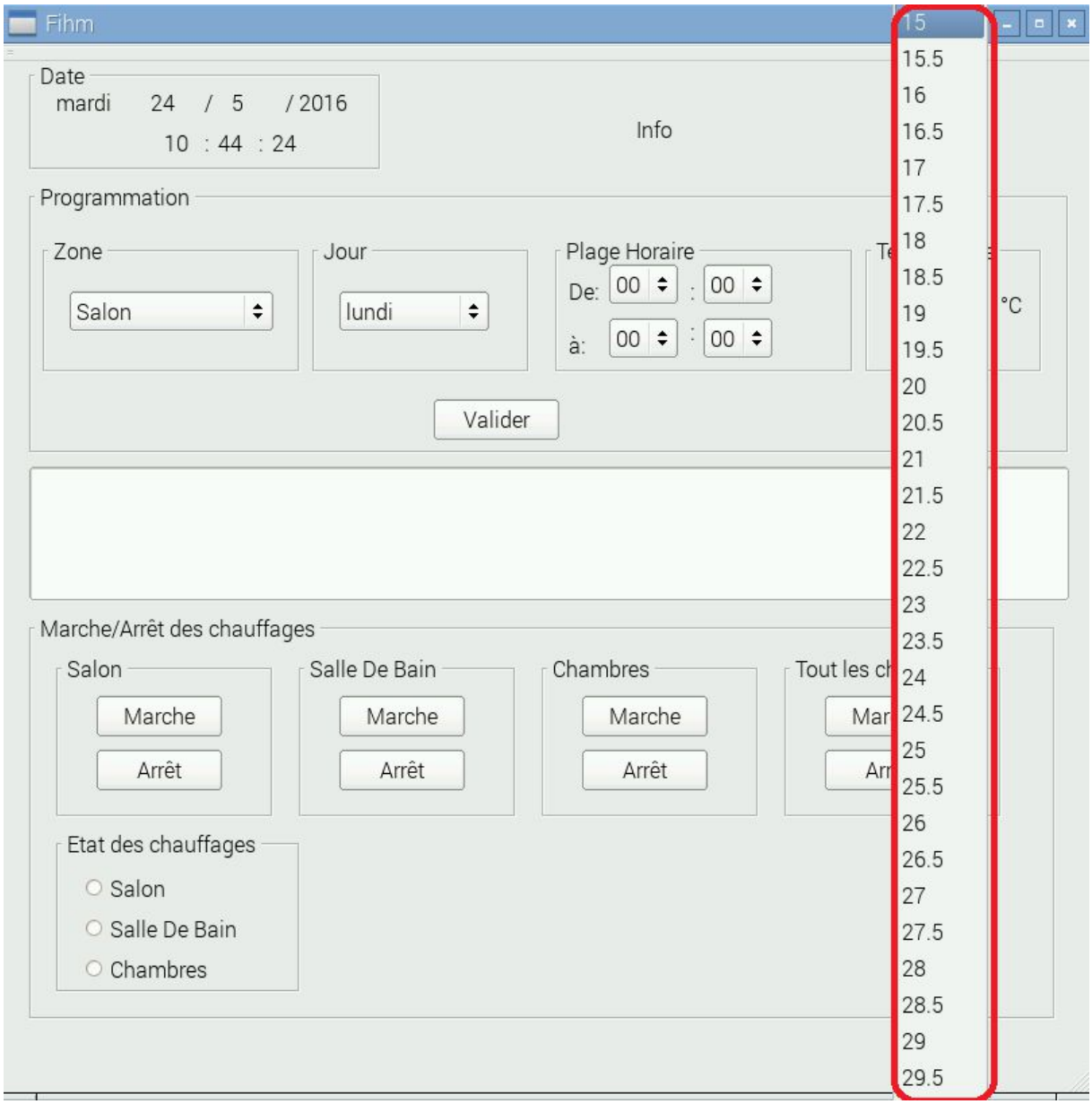
3. Cliquer sur la boite de dialogue « jour » et sélectionner le jour voulu.



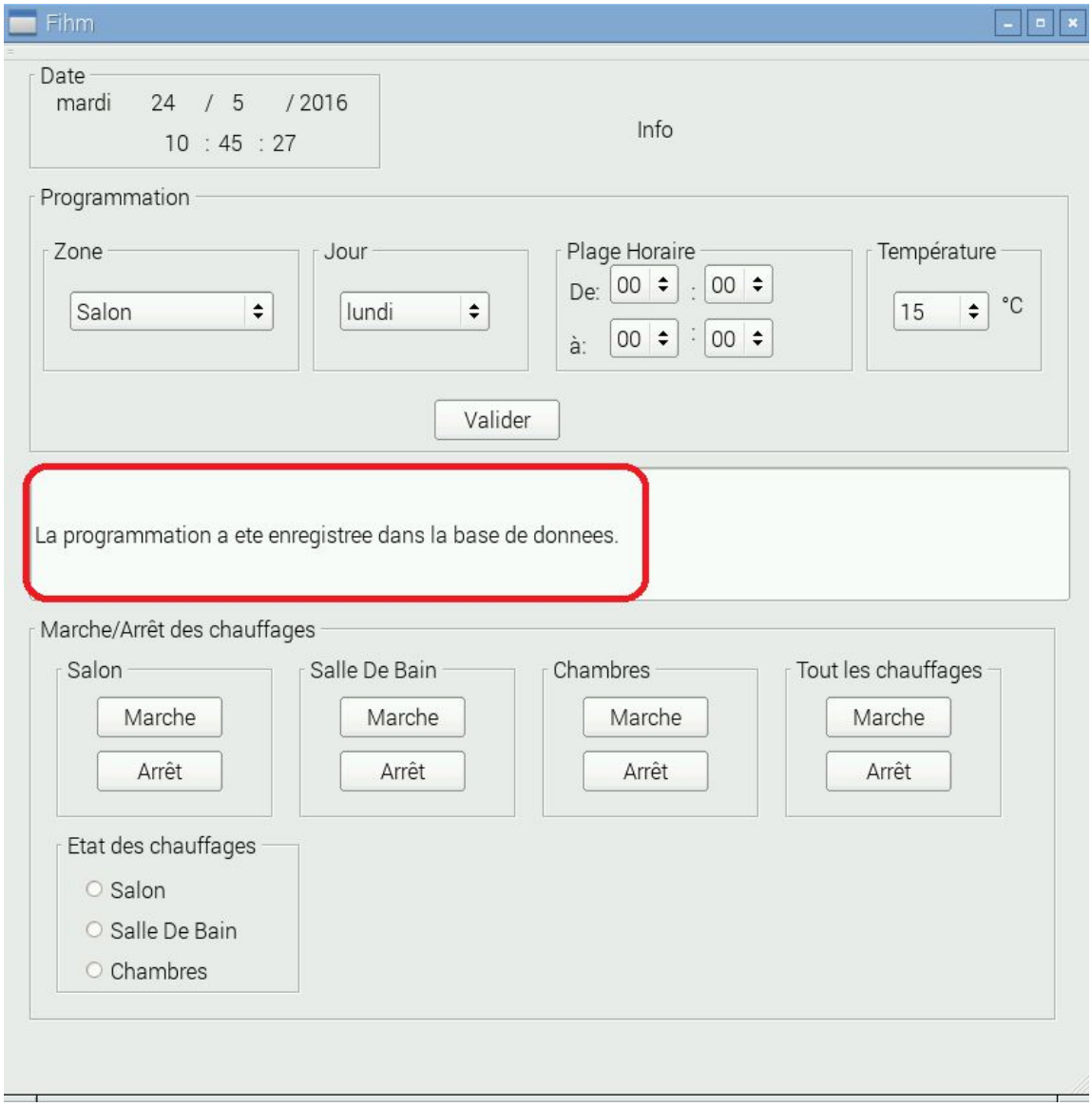
4. Cliquer sur les boites de dialogues « Plage Horaire » et sélectionner la plage horaire.



5. Cliquer sur la boite de dialogue « Température » et sélectionner la température voulue.



6. Vérifier les informations et cliquer sur le bouton « Valider » pour valider la programmation. Le message « Programmation enregistrée dans la base de données » devrait apparaître. Si ce message apparaît, c'est que votre programmation a fonctionnée.



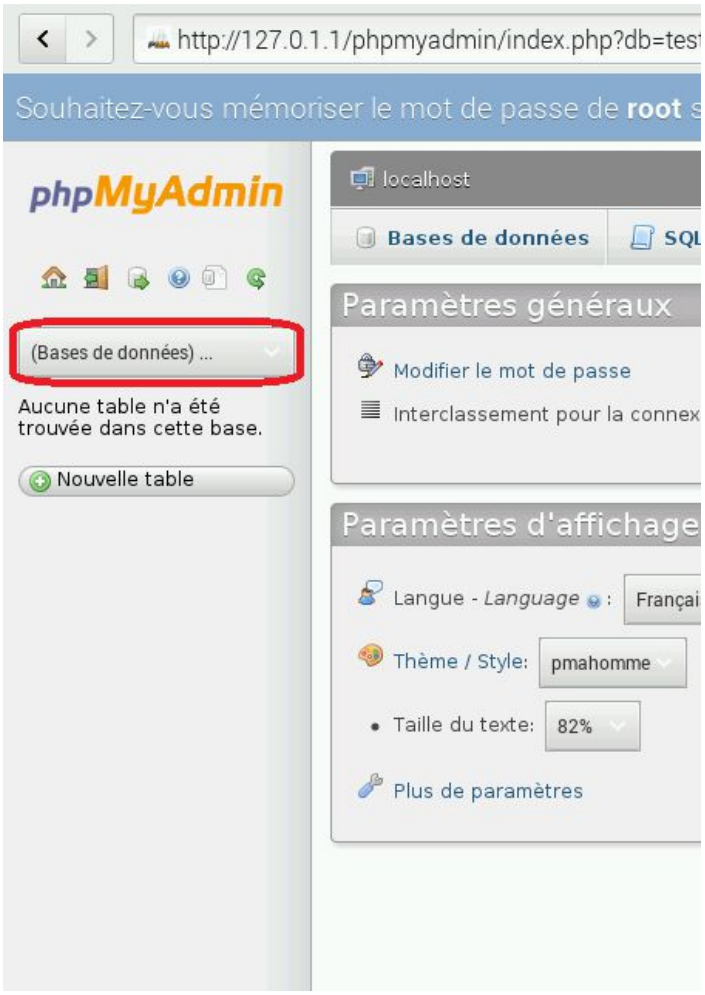
Base de données

1. Double-cliquer sur l'icône « Base de données Mariton » sur le bureau.
2. Rentrer les identifiants de connexion :
   Administrateur :
   Identifiant : root
   Mot de passe : admin

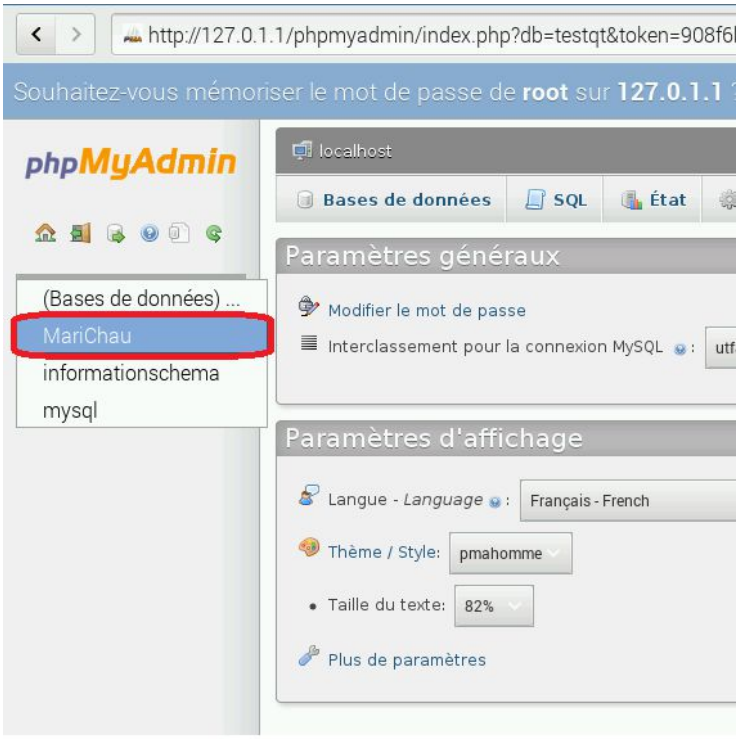
3.7. Maintenance curative

Si vous remarquez que votre chauffage ne chauffe pas comme il faudrait ou que la température ne correspond pas à ce que vous avez programmé :

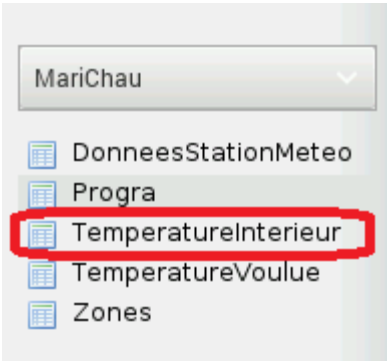
- Connectez-vous à la base de données en utilisant les données de connexion du mode d'emploi.
- Cliquez sur la sélection des bases de données.



- Sélectionnez la base de données de votre chauffage.



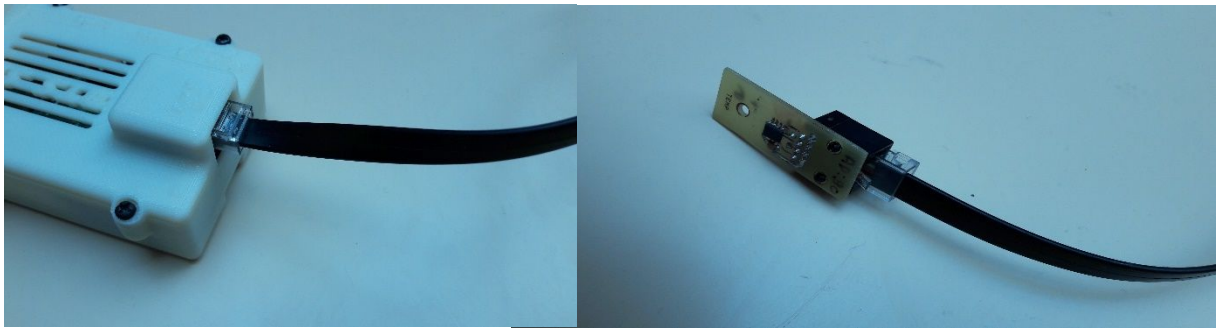
- Sélectionnez la table « TempératureInterieur ».



- Vérifiez que la dernière température enregistrée soit récente.

Si ce n'est pas le cas :

- Vérifiez que le capteur de température n'est pas débranché de la RaspBerry, des deux côtés du câble.



3.8. Conclusion

Pour conclure, ce projet m'a aidé à mieux travaillé en groupe et m'a apporté beaucoup de connaissances et d'expérience. Il m'a permis de mieux me rendre compte de ce que représente un projet en entreprise et du travail à accomplir autant au niveau du projet en lui-même qu'au niveau de la documentation à fournir. J'ai rencontré plusieurs problèmes mais j'ai toujours réussi à les contourner ou à les surmonter avec l'aide de mes collègues de projet. Au moment où je rédige ces lignes le projet n'est pas fini mais je prévois de finir le plus de tâches d'ici à la revue finale.

Partie de l'IR 3 :

LAMBERT Benjamin

Communication TCP

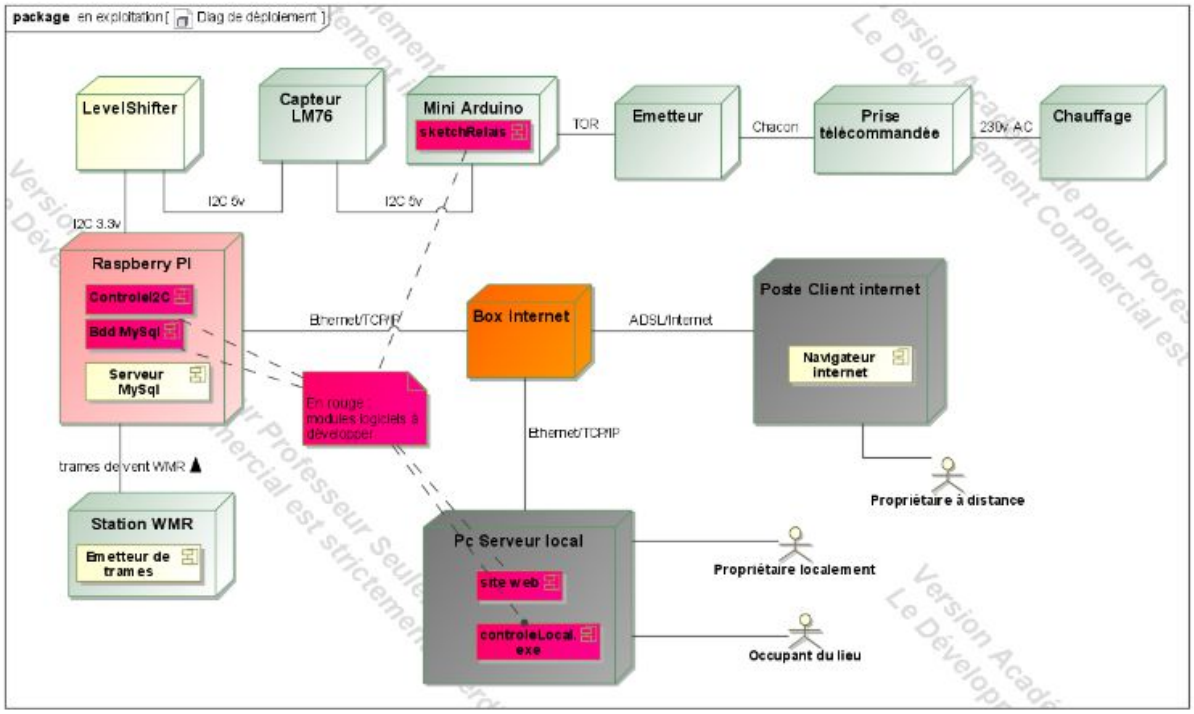
Sommaire

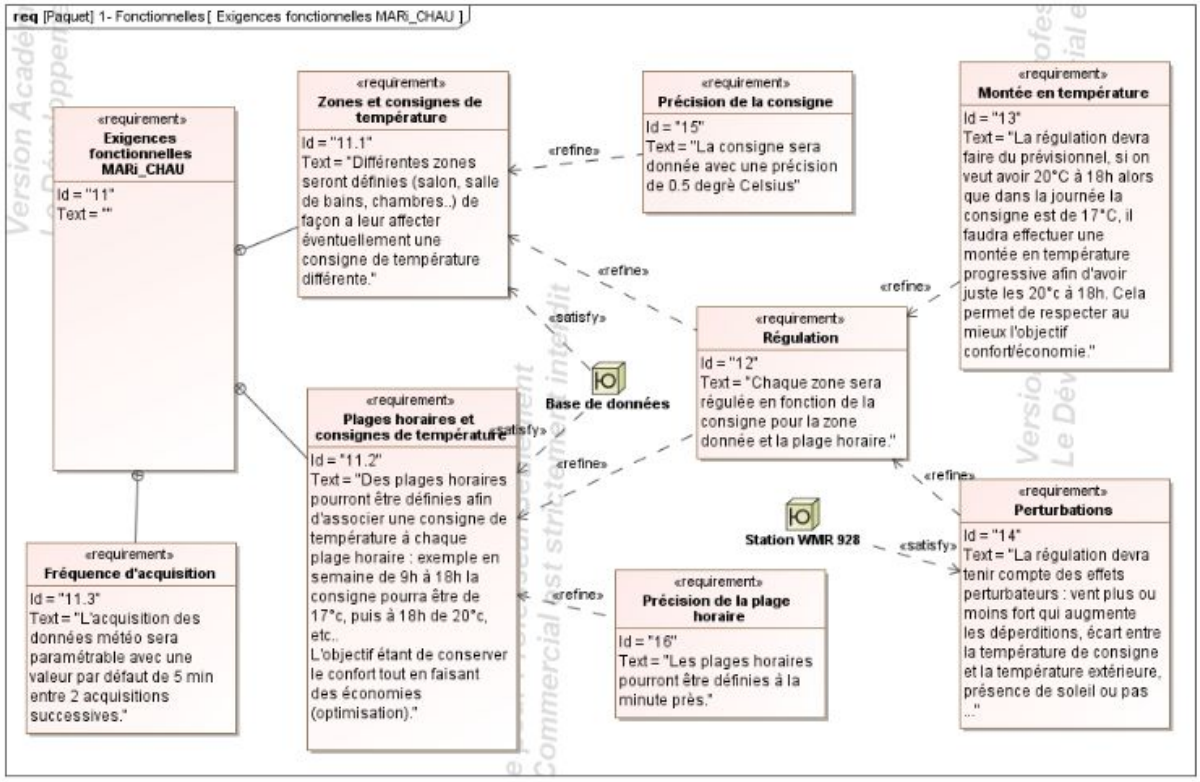
Introduction

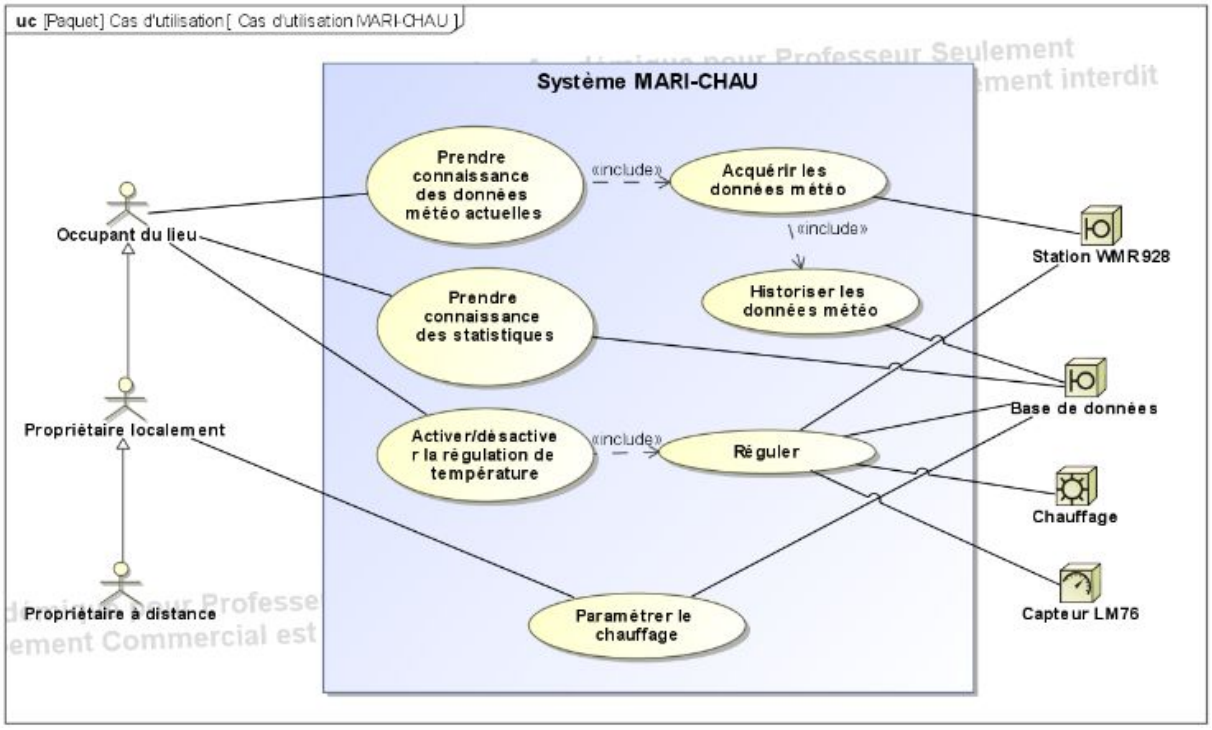
Il s'agit de prolonger les activités de la société dans le domaine de la domotique et d'acquérir une autonomie dans la fourniture de matériels et logiciels nécessaires pour l'intégration aux produits existants. Dans ce projet il s'agit plus précisément de gérer le chauffage électrique d'une maison ou d'un local quelconque afin de réduire la consommation tout en préservant le confort des occupants lorsque ceux-ci sont présents. Une base de données des consignes de températures permettra à la régulation de s'adapter à la présence ou non des occupants.

Diagrammes liés au projet

Diagramme de paquetage du système avec la solution incluse




req [Paquet] 1- Fonctionnelles [Exigences fonctionnelles MARi_CHAU]
«requirement»
Exigences fonctionnelles MARi_CHAU
Id = "11"
Text = ""
«requirement»
Zones et consignes de température
Id = "11.1"
Text = "Différentes zones seront définies (salon, salle de bains, chambres..) de façon a leur affecter éventuellement une consigne de température différente."
«requirement»
Précision de la consigne
Id = "15"
Text = "La consigne sera donnée avec une précision de 0.5 degré Celsius"
«requirement»
Montée en température
Id = "13"
Text = "La régulation devra faire du prévisionnel, si on veut avoir 20°C à 18h alors que dans la journée la consigne est de 17°C, il faudra effectuer une montée en température progressive afin d'avoir juste les 20°c à 18h. Cela permet de respecter au mieux l'objectif confort/économie."
«refine»
«satisfy»
Base de données
«requirement»
Régulation
Id = "12"
Text = "Chaque zone sera régulée en fonction de la consigne pour la zone donnée et la plage horaire."
«requirement»
Plages horaires et consignes de température
Id = "11.2"
Text = "Des plages horaires pourront être définies afin d'associer une consigne de température à chaque plage horaire : exemple en semaine de 9h à 18h la consigne pourra être de 17°c, puis à 18h de 20°c, etc..
L'objectif étant de conserver le confort tout en faisant des économies (optimisation)."
«requirement»
Fréquence d'acquisition
Id = "11.3"
Text = "L'acquisition des données météo sera paramétrable avec une valeur par défaut de 5 min entre 2 acquisitions successives."
Station WMR 928
«requirement»
Perturbations
Id = "14"
Text = "La régulation devra tenir compte des effets perturbateurs : vent plus ou moins fort qui augmente les déperditions, écart entre la température de consigne et la température extérieure, présence de soleil ou pas ..."
«requirement»
Précision de la plage horaire
Id = "16"
Text = "Les plages horaires pourront être définies à la minute près."


uc [Paquet] Cas d'utilisation [Cas d'utilisation MARI-CHAU]
Système MARI-CHAU
Prendre connaissance des données météo actuelles
«include»
Acquérir les données météo
«include»
Prendre connaissance des statistiques
Historiser les données météo
Activer/désactiver la régulation de température
«include»
Réguler
Paramétrer le chauffage
Occupant du lieu
Propriétaire localement
Propriétaire à distance
Station WMR928
Base de données
Chauffage
Capteur LM76

La tâche qui m'est attribués

Étudiant 3 IR 3	*Liste des fonctions assurées par l'étudiant* Communication entre le PC serveur local et la Raspberry PI par client/serveur avec les sockets système.	Installation : Qt. Mise en œuvre : Sockets linux, sockets windows. Configuration : Réalisation : Communication client lourd/serveur avec les sockets système. Documentation : Installation, mise en service, dossier de développement.

Matériel à disposition

- Une carte Raspeberry .
- QT Creator, avec sa documentation.

Communication TCP/IP

Introduction

Tout d'abord, pourquoi TCP et non UDP, TCP est un protocole fonctionnant en mode "connecté" (on devrait dire "associé" car "connecté" est originaire des télécommunications). C'est à dire qu'il y a une connexion logique entre le client et le serveur permettant d'identifier chacune des parties.

TCP offre également l'assurance que les paquets arriveront à destination et permet d'acquitter un transfert.

UDP est un protocole beaucoup plus simple. Il n'y a pas de gestion de connexion ou des données on les envoie en espérant qu'elles arrivent à destination.
Par contre, le récepteur ne peut pas différencier les différents émetteurs.

1. Communication physique

La communication entre le client et le serveur, se fait par onde porteuse à travers un câble Ethernet, ou par ADSL

1.1.Ethernet

Ethernet est un protocole de réseau local à commutation de paquets. (Norme internationale : ISO/IEC 8802-3.)

On utilise très fréquemment Ethernet sur paires torsadées pour la connexion des postes clients, et des versions sur fibre optique pour le cœur du réseau.

L'Ethernet est basé sur le principe de pairs sur le réseau, envoyant des messages dans ce qui était essentiellement un système radio, captif à l'intérieur d'un fil ou d'un canal commun, parfois appelé *l'éther*. Chaque pair est identifié par une clé globalement unique, appelée adresse MAC, pour s'assurer que tous les postes sur un réseau Ethernet aient des adresses distinctes.

Une technologie connue sous le nom de CSMA/CD régit la façon dont les postes accèdent au média.

Lorsqu'un ordinateur veut envoyer de l'information, il obéit à l'algorithme suivant :

1.1.1Procédure principale

1. Trame prête à être transmise.
2. Si le medium n'est pas libre, attendre jusqu'à ce qu'il le devienne puis attendre la durée intertrame (9,6 µspour l'Ethernet 10 Mbit/s) et démarrer la transmission.
3. Si une collision est détectée, lancer la procédure de gestion des collisions. Sinon, la transmission est réussie.

1.1.2Procédure de gestion des collisions

1. Continuer la transmission à hauteur de la durée d'une trame de taille minimale (64 octets) pour s'assurer que toutes les stations détectent la collision.
2. Si le nombre maximal de transmissions (16) est atteint, annuler la transmission.
3. Attendre un temps aléatoire dépendant du nombre de tentatives de transmission.
4. Reprendre la procédure principale.

1.1.3 Découpage de trame en octet

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14 … 1513	1514	1515	1516	1517
Adresse MAC destination						Adresse MAC source						Type de protocole		Données	FCS/CRC			

il existe plusieurs types de trames Ethernet qui possèdent d'autres particularités. Le champ *Type de protocole* peut prendre par exemple les valeurs suivantes :

- 0x0800 : IPv4
- 0x86DD : IPv6
- 0x0806 : ARP
- 0x8035 : RARP
- 0x809B : AppleTalk
- 0x88CD : SERCOS III
- 0x0600 : XNS
- 0x8100 : VLAN

1.2ADSL

L'ADSL (ou *asymmetric digital subscriber line*) est une technique de communication numérique (couche physique) de la famille xDSL. Elle permet d'utiliser une ligne téléphonique, une ligne spécialisée, ou encore une ligne RNIS

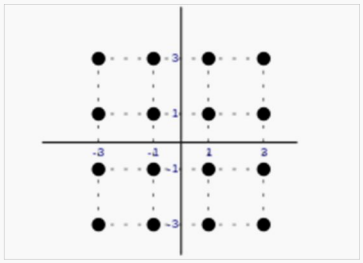
pour transmettre et recevoir des données numériques de manière indépendante du service téléphonique conventionnel (c'est-à-dire analogique) via un filtre ADSL,

L'ADSL fait appel à la notion de sous-porteuses : la bande de fréquences comprise entre 0 Hz et environ 1,1 MHzest divisée en 255 intervalles de 4,3125 kHz. À chaque intervalle est associée une sous-porteuse, qui donne un signal modulé. La n-ième sous-porteuse est donc matérialisée sous la forme d'un signal dont la fréquence de base vaut [n × 4,3125 kHz].

Un modem ADSL peut donc être considéré comme la mise en parallèle d'un grand nombre de modems analogiques, chacun transmettant sur une fréquence différente : 4,3125 kHz pour le premier,8,6250 kHz pour le second, 12,9375 kHz pour le troisième, et ainsi de suite.

La sous-porteuse d'indice 0 n'est pas utilisée, car elle correspond à un signal de fréquence nulle. Les sous-porteuses d'indice 1 à 255 sont théoriquement utilisables pour transmettre des données. (les fréquences 1à 6 ne sont également généralement pas exploitées en raison de présence possible de signaux téléphoniques)

1.2.1Modulation des sous-porteuses



Modulation d'une sous-porteuse pour 4 bits transportés

Chaque sous-porteuse est modulée en amplitude et en phase, au rythme de 4 000 symboles par seconde (4000bauds)

les modulations complexes permettent de transporter un nombre de bits élevé, ce qui favorise le débit, mais ce type de modulation est plus difficile à décoder au niveau du récepteur et est donc plus sensible aux erreurs de transmission provoquées par d'éventuelles perturbations en ligne . Le niveau de modulation de chaque sous-porteuse peut donc être ajusté pour transporter entre 2 et 15 bits d'information par symbole. Le nombre de bits affecté à chaque sous-porteuse est déterminé en début de connexion, après une phase de mesure de qualité de la ligne effectuée par échange de signaux de test entre les deux équipements ADSL qui établissent la communication.

1.2.2Trames ADSL

Les informations transportées par l'ADSL dans chaque sens de communication sont organisées en trames d'une taille égale à la somme des bits véhiculés par l'ensemble des sous-porteuses affectées à ce sens de communication. En supposant par exemple que le sens descendant utilise 40 sous-porteuses et que chaque sous-porteuse transporte 8 bits par symbole, la taille de la trame correspondante est de 320 bits, soit 40 octets. Chaque sous-porteuse étant modulée à raison de 4 000 symboles par seconde, ce sont donc 4 000 trames qui sont envoyées à chaque seconde, et avec les chiffres de notre exemple, le débit brut du sens descendant s'établit à 4 000 × 320 bits, soit 1 280 kbit/s.

Chaque trame contient des informations de service, des données utilisateur, et éventuellement des octets de redondance utilisés pour détecter et si possible corriger les erreurs.

1.2.3Supertrames ADSL

Pour des raisons de synchronisation, les trames ADSL sont regroupées en « trains » de 68

trames consécutives et complétées par une 69e trame de contrôle qui contient des informations de service additionnelles plutôt que des données utilisateur. Ces groupes de 69 trames sont désignés sous le nom de « supertrames »

Cette dernière prenant de la « place » chaque porteuse n'est pas exactement de 4000bauds mais de 4000 x 69/68 bauds ce qui représente environ 4059 bauds de qui permet la transmission d'exactement 4000 symbole « utiles » par seconde.

Exemples de valeurs d'atténuation et de débit en fonction de la longueur de ligne

Longueur totale (m)	Longueur (m) en diamètre 4/10 mm	Longueur (m) en diamètre 6/10 mm	Atténuation (dB)	Débit (Mbit/s) en mode ADSL1 ou ADSL2	Débit (Mbit/s) en mode ADSL2+
170	170		4,1	8,0	19,4
458	458		8,4	8,0	18,7
730	730		12,5	8,0	18,2
1038	698	340	15,5	8,0	16,6
1301	1158	143	20,3	7,3	14,2
2430	679	1751	29,7	6,0	11,3
2540	2540		39,6	5,7	7,4
3909	1240	2669	47,6	4,2	5,4
5004		5004	53.0	3,1	4,3
5755		5755	60,8	2,0	3,0

2.TCP/IP

Le TCP (**Transmission Control Protocol** ou protocole de contrôle de transmissions en français) est un protocole de transport fiable, en mode connecté.

Dans le modèle Internet, aussi appelé modèle TCP/IP, TCP est situé au-dessus de IP. Dans le modèle OSI, il correspond à la couche transport, intermédiaire de la couche réseau et de la couche session. Les applications transmettent des flux de données sur une connexion réseau. TCP découpe le flux d'octets en *segments* dont la taille dépend de la MTU du réseau sous-jacent (couche liaison de données).

2.1Fonctionnement

Une session TCP fonctionne en trois phases :

- l'établissement de la connexion ;
- les transferts de données ;
- la fin de la connexion.

L'établissement de la connexion se fait par un handshaking en trois temps. La rupture de connexion, elle, utilise unhandshaking en quatre temps. Pendant la phase d'établissement de la connexion, des paramètres comme le numéro de séquence sont initialisés afin d'assurer la transmission fiable (sans perte et dans l'ordre) des données.

Le handshaking en trois temps représente les 3 étapes de connexions entre deux hôtes premièrement Le client qui désire établir une connexion avec un serveur va envoyer un premier paquet SYN (synchronized) au serveur. Ensuite Le serveur va répondre au client à l'aide d'un paquet SYN-ACK (synchronize, acknowledge). Et pour finir le client va envoyer un paquet ACK au serveur qui va servir d'accusé de réception. Une fois le three-way handshake effectué, une communication full-duplex est maintenant établie entre le client et le serveur.

2.2Structure d'un segment TCP

En bits

Signification des champs :

- Port source : numéro du port source
- Port destination : numéro du port destination
- Numéro de séquence : numéro de séquence du premier octet de ce segment
- Numéro d'acquittement : numéro de séquence du prochain octet attendu
- Taille de l'en-tête : longueur de l'en-tête en mots de 32 bits (les options font partie de l'en-tête)
- Indicateurs ou *Flags* :
- Réservé : réservé pour un usage futur
- ECN/NS : signale la présence de congestion, voir RFC 3168 ; ou Nonce Signaling, voir RFC 3540
- CWR : Congestion Window Reduced : indique qu'un paquet avec ECE a été reçu et que la congestion a été traitée
- ECE : ECN-Echo : si SYN=1 indique la capacité de gestion ECN, si SYN=0 indique une congestion signalé par IP (voir RFC 3168)
- URG : Signale la présence de données **urg**entes
- ACK : signale que le paquet est un accusé de réception (**ack**nowledgement)
- PSH : données à envoyer tout de suite (**p**u**sh**)
- RST : rupture anormale de la connexion (**r**e**s**e**t**)
- SYN : demande de **syn**chronisation ou établissement de connexion
- FIN : demande la **fin** de la connexion
- Fenêtre : taille de fenêtre demandée, c'est-à-dire le nombre d'octets que le récepteur souhaite recevoir sans accusé de réception
- Somme de contrôle : somme de contrôle calculée sur l'ensemble de l'en-tête TCP et des données, mais aussi sur un pseudo en-tête (extrait de l'en-tête IP)
- Pointeur de données urgentes : position relative des dernières données urgentes
- Options : facultatives

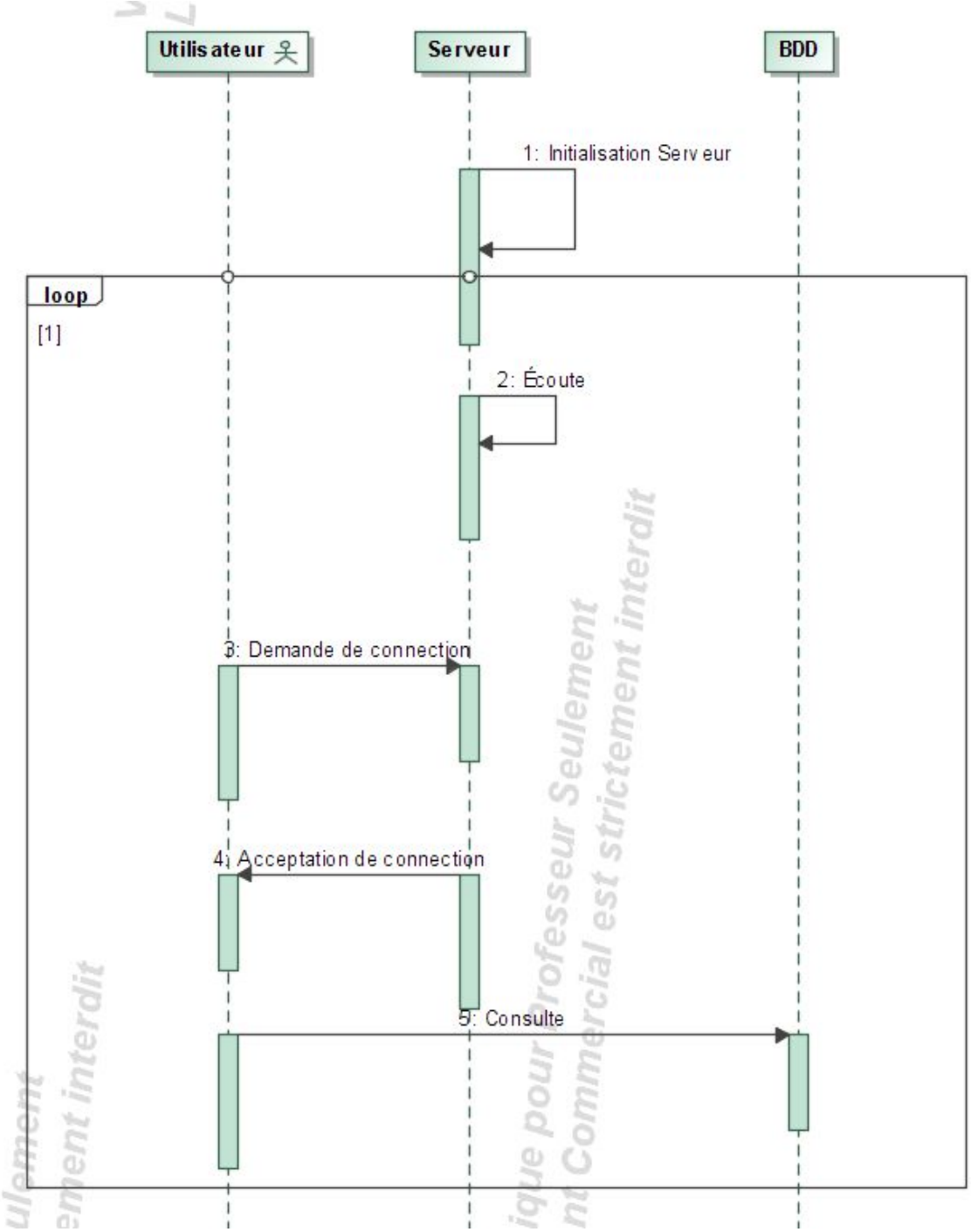
- Remplissage : zéros ajoutés pour aligner les champs suivants du paquet sur 32 bits, si nécessaire
- Données : séquences d'octets transmis par l'application (par exemple : +OK POP3 server ready...)

3. Analyses

Ma tâche consistait à établir la communication entre le PC serveur local et la Raspberry PI par Client/Serveur avec les sockets système, pour cela plusieurs solutions étaient possibles, il aurait pu être possible de mettre en place un serveur Web pour que les utilisateurs puissent consulter ou modifier la base de donnée depuis n'importe quel poste ayant internet, mais les exigences du projet précise que je dois créer un Client lourd, une application donc qui permettra de se connecter a notre serveur et consulter la base de données.

Pour ce qui est du serveur, je trouvais plus pertinent que ce dernier soit sur la Raspeberry, car elle ne sera jamais éteinte, il est possible de la paramétrer avec une adresse IP fixe pour faciliter son accès par les Clients. Il devra être accessible a plusieurs clients en même temps (Si par exemple plusieurs personnes veulent consulter la Base de donnée en même temps) mais il faudra ajouter des rangs aux utilisateurs (Administrateur ou Utilisateur par exemple) pour qu'il n'y ai pas de conflits, que plusieurs utilisateurs modifient en même temps la même ligne de la base de donnée.

3.1 Fonctionnement client serveur TCP



4. Mise en œuvre

4.1 Esquisses

Pour commencer, j'ai développé le client et le serveur en C pour ma familiariser avec le procéder, la marche à suivre etc.

4.1 .1 Serveur TCP en C

Ci-dessous les bibliothèques utiles à la création et la gestion du serveur.

```
#include <stdio.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <strings.h>

#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <unistd.h>
```

Les différentes déclarations de types requis dans les fonctions liés au TCP

```
int sd, sdc; //sd -> socket d'écoute   sdc -> socket du client
socklen_t lg;
struct sockaddr_in client;
struct sockaddr_in mon_adresse;
char buf[5];


bzero(&mon_adresse, sizeof(mon_adresse));//initialisation de l'adresse
mon_adresse.sin_family = AF_INET;
mon_adresse.sin_port = htons(2222);
mon_adresse.sin_addr.s_addr = INADDR_ANY;
```

```
 /*---------------------------------------//Création socket//---------------------------*/

if (sd = ((socket(AF_INET, SOCK_STREAM, 0) == -1))
{
  printf("problème création socket \n");
  exit(0);
}
/*---------------------------------------//Bind Serveur//--------------------------------*/
lg=sizeof(struct sockaddr_in);
if (bind(sd,(struct sockaddr *)&mon_adresse,sizeof(mon_adresse)) == -1)
{
  perror("pb bind");
  printf("erreur bind");
  exit(0);
}
/*---------------------------------------//Ecoute sur la socket//-------------------------*/
if (listen(sd,5) == -1)
{
  printf("erreur listen");
  exit(0);
}
/*-----------------------------------//Accepte la connexion//-------------------------*/

do
{
  lg=sizeof(client);
  sdc=accept(sd,(struct sockaddr*)&client,&lg);
    write(sdc, "COUCOU", 6);
    read(sdc, buf,3);
} while (!strncmp(buf, "FIN", 3));
shutdown(sdc,2);
shutdown(sd,2);
close(sd);
}
```

4.1.2 Client TCP en C

```
#include <stdio.h>
#include <errno.h>
#include <signal.h>

#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define SERVEURNAME "127.0.0.1" //adresse du serveur

int to_server_socket = -1;

void main ( void )
{

char *server_name = SERVEURNAME;
struct sockaddr_in serverSockAddr;
struct hostent *serverHostEnt;
long hostAddr;
long status;
char buffer[512];

bzero(&serverSockAddr,sizeof(serverSockAddr));
hostAddr = inet_addr(SERVEURNAME);
if ( (long)hostAddr != (long)-1)
  bcopy(&hostAddr,&serverSockAddr.sin_addr,sizeof(hostAddr));
else
{
  serverHostEnt = gethostbyname(SERVEURNAME);
  if (serverHostEnt == NULL)
  {
    printf("Erreur gethost\n");
    exit(0);
  }
  bcopy(serverHostEnt->h_addr,&serverSockAddr.sin_addr,serverHostEnt->h_length);
}
serverSockAddr.sin_port = htons(30000);
serverSockAddr.sin_family = AF_INET;

/*-----------------------------// creation de la socket//--------------------------- */
if ( (to_server_socket = socket(AF_INET,SOCK_STREAM,0)) < 0)
{
  printf("Erreur creation socket client\n");
  exit(0);
}
/*-----------------------------// requete de connexion//--------------------------- */
if(connect( to_server_socket,
            (struct sockaddr *)&serverSockAddr,
            sizeof(serverSockAddr)) < 0 )
{
  printf("Erreur demande de connection\n");
  exit(0);
}
/* envoie de donne et reception */
write(to_server_socket,"Toto",4);
read(to_server_socket,buffer,512);
printf(buffer);
/* fermeture de la connection */
shutdown(to_server_socket,2);
close(to_server_socket);
}
```

Ces codes sont fonctionnels ils ne font pas la liaison avec la base de donnée car ce ne sont que des codes que j'ai utilisé pour m'entraîner et avoir une vision de ce à quoi devront ressembler les codes en C++

4.2 Programmation en orienté objet

4.2.1 Client Lourd

Tcpclient.cpp

```
#include "tcpclient.h"
#include <QHostAddress>

TCPClient::TCPClient(QObject *parent) :
    QObject(parent)
{

   connect(&client,SIGNAL(connected()),this,SLOT(startTransfer()));
}

TCPClient::~TCPClient()
{
    client.close();
}

void TCPClient::start()
{
    //QHostAddress addr(Adresse);
    client.connectToHost("192.168.2.173",2222);
    qDebug() << "Demande de connexion.";
}

void TCPClient::startTransfer()
{/*
    QByteArray Trame = "UPDATE METEO SET LUMIERE ='17'";
    client.write(Trame);
    qDebug() << "connecté";*/
}
void TCPClient::sendrequete(QString zone, int nb)
{

}
void TCPClient::requetemode(QByteArray maquette, QByteArray mode)
{
   /* QByteArray requete = "UPDATE "+maquette+"SET AUTO = '"+mode+"'";

    client.write(requete);*/
}
void TCPClient::requetetest()
{
 /*   QByteArray test= "test";
    client.write(test);*/
}
```

Tcpclient.h

```
#ifndef TCPCLIENT_H
#define TCPCLIENT_H

#include <QObject>
#include <QtNetwork>
#include <QString>
#include <QTcpSocket>
#include <QDebug>


class TCPClient : public QObject
{
    Q_OBJECT
public:
    ~TCPClient();
    explicit TCPClient(QObject *parent = 0);
    void start();
    void sendrequete(QString zone, int nb);
    void requetemode(QByteArray maquette, QByteArray mode);
    void requetetest();

public slots:
    void startTransfer();

private:

    QTcpSocket client;

};

#endif // TCPCLIENT_H
```

4.2.2Serveur

Le serveur, lui, n'a pas encore été codé à ce jour, mais devrait être fonctionnel pour le 13 Juin. Les tests ont été faits avec le serveur en C présenté précédemment.

Conclusion

Durant ce projet j'ai eu plusieurs difficultés, une gestion du temps trop approximative, des lacunes en codage qui ont fait que j'ai passé énormément de temps sur un programme de base, ce problème persistant, au bout de quelques semaines j'ai commencé a baisser les bras, de plus notre professeur référent était très peu présent, impossible donc de demander son aide et son point de vue sur les problèmes rencontrés. Quand à l'équipe de projet il y avait une bonne symbiose nous étions tout les 4 sur la même longueur d'onde et coopératifs.

Au début du projet je m'étais lancé sur la création d'un site web utilisant l'HTML, le CSS et le PHP mais sa pertinence est moindre dans le cadre de ce projet car j'étais partis sur l'idée d'un serveur web mais c'était trop compliqué à mettre en œuvre.

Pendant les semaines restantes je me pencherai sur la problematique du serveur en POO et sur la liaison avec la base de données.

Partie EC:

Bernard Alex

Communication I2C/HF/HomeEasy

Prototypage

5.1 Présentation partie électronique et communication

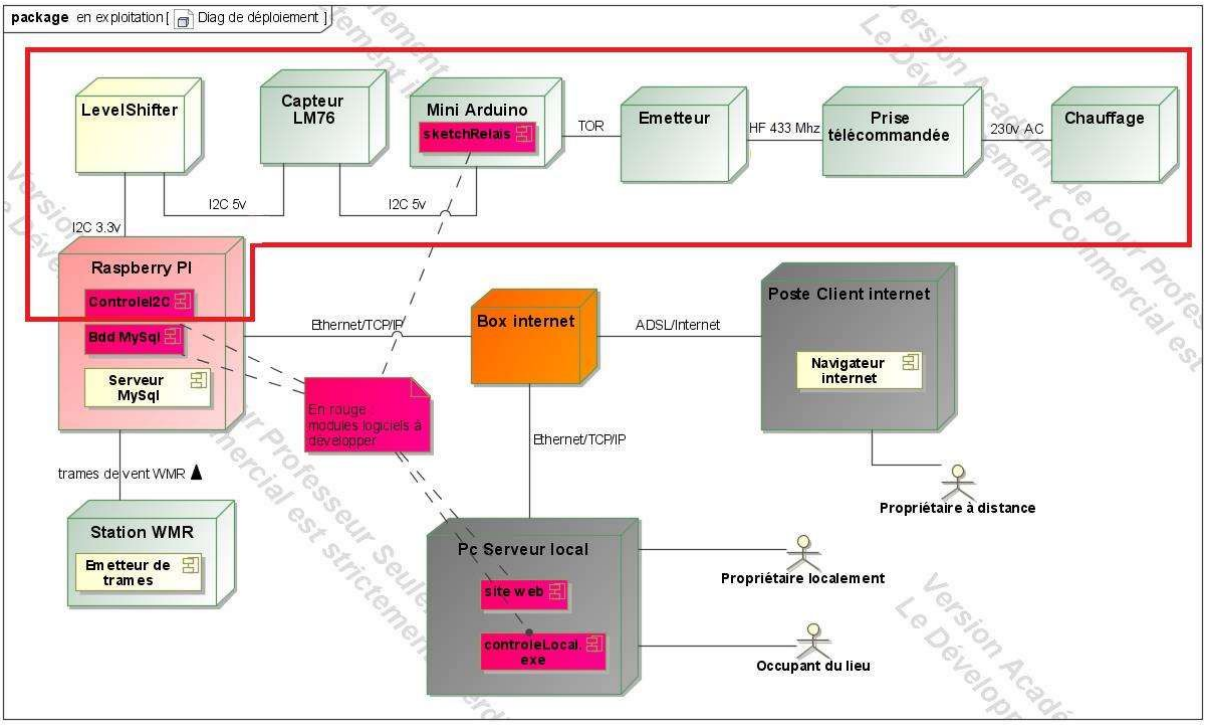
Cette partie du projet consiste à réaliser un prototype et à mettre en place les communications qui permettront, à partir d'une carte Raspberry pi, d'interagir avec des chauffages par une liaison sans fil.

Pour la réalisation du prototype, deux cartes électroniques ont été réalisé. Un level shifter qui est monté directement sur la carte Raspberry , dont le but et de metre à niveau la tension de la ligne de sortie I2c de celle-ci. Et une deuxieme carte, qui comprent une carte programable mini Arduino et un module émeteur Haute Fréquance, qui permet la liaison sans fil avec les chauffages. Le contrôle des chauffages se fait grâce à des prises télécomandée.

Trois principales communication sont utiliser pour ce projet :

-le bus I2c, qui permet la connexion entre la carte arduino et Raspberry
-protocole chacon/home easy ,de la mini arduino à l'émeteur HF.
- haute fréquance 433Mhz qui retransmet le code home easy en onde radio jusqu'aux prise télécomandé.

Dans cette partie, nous aborderons se qui se trouve dans l'encadré rouge si dessous.



5.2 Chacon /émission Radio Fréquence

5.2.1 Prise télécommandé DI-O 54795

J'ai commencé le projet par m'approprier le matériel, en commençant par les prises télécommandées DI-O par Chacon. Ces prises permettent le contrôle à distance des radiateurs par émission radio 433,92Mhz. Ce sont des prises qui se branchent sur le réseau électrique EDF habituel en 230v 50 Hz. La puissance maximal supporté est de 2300w pour 10 A. le principe et d'ouvrir et fermé le circuit à distance comme un simple interrupteur, et donc dans notre cas, d'alimenter ou non le chauffage en électricité.

La configuration des prises est très simple, pour rajouter un contrôleur, il suffit de brancher la prise sur le secteur, et alors qu'elle clignote (pendant 5seconde) envoyer une commande d'allumage, appuyé sur le premier bouton d'allumage de la télécommande par exemple.

Pour supprimer un contrôleur de la sauvegarde interne, il faut débrancher et rebrancher la prise, attendre 5 secondes et cette fois, envoyer une commande d'extinction.

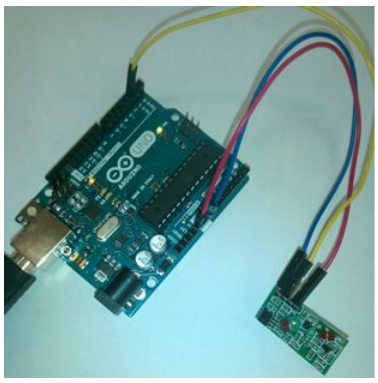
Les prises ont la possibilité d'avoir trois émetteurs maximum.

Le but du projet est de pouvoir contrôler les prises non pas avec la télécommande, mais avec un émetteur HF contrôlé par une carte Arduino qui sera, elle, contrôlé par une carte Raspberry Pi.

5.2.2 Nouvel émetteur

Récupération de l'adresse :

Pour pouvoir utiliser les prises sans la télécommande, il a fallu créer un nouvel émetteur. Pour cela, j'ai d'abords récupérer les informations envoyé par la télécommande grâce à une carte Arduino UNO, un récepteur HF 433Mhz et une librairie mis à disposition sur internet. Le montage réalisé est simple, le capteur HF est relié à la carte Arduino par 3 broche, GND / 5V / DATA (pin 2)



Le programme à exécuter sur la carte Arduino est une librairie trouvé sur le site Bitbucket (https://bitbucket.org/fuzzillogic/433mhzforarduino/wiki/Home), il consiste à récupérer les données envoyé par une télécommande utilisant le protocole HomeEasy. *(voir code annexe 1)*

La librairie s'instale dans le dossier source du logiciel arduino (C:\Program Files (x86)\Arduino\libraries)

Librairie : NewRemoteSwitch exemple: ShowReceivedCode

Un appui sur les touches de la télécommande écrit donc sur le port série :

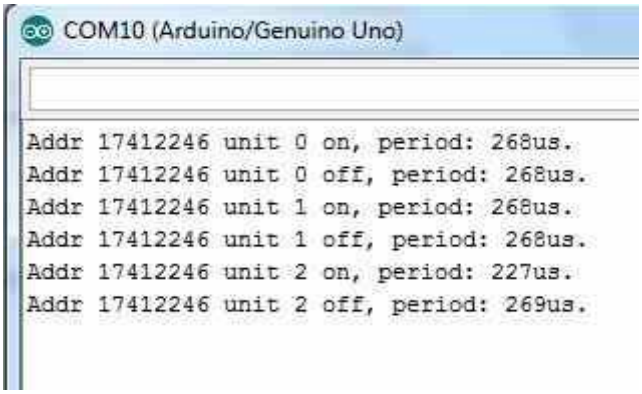
(L'appuient sur les touches est réalisé de gauche à droite et de bas en haut)

Adresse télécommande : 17412246

Unit0 = interrupteur n°1

Unit1= interrupteur n°2

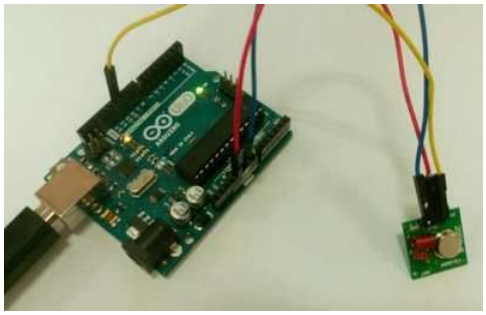
Unit2= interrupteur n°3



Nous récupérons l'adresse de la télécommande pour faciliter la programmation des prises car comme dit plus haut, la programmation se fait directement après les avoir branché, cependant, le projet n'a pas pour but d'être portable et l'utilisateur ne peut donc pas envoyer les commandes en même temps que de manipuler une prise.

Programmation d'un nouvel émetteur :

Le câblage est approximativement le même que le premier, sauf que le broche DATA est relié à la pin 11 de la carte arduino.



Tout comme le premier programme, un exemple est disponible dans la librairie téléchargé précédemment (lightShow) (voir annexe 1), il m'a permis de réaliser un programme qui permet de communiquer avec les prises télécommandées après réception d'un caractère sur le port série.

Ex : si la carte Arduino reçoit un « a » sur le port série, il enverra une trame Chacon à l'émetteur, permettant d'allumer le premier interrupteur, un « b » éteint celui-ci, un « c » allume le deuxième, et ainsi de suite. (Les lettres peuvent être envoyées directement par le moniteur série avec le clavier)

L'inconvénient de l'utilisation de ces prises se fait remarqué lors des tests, ils n'y a pas de retour d'états possible. Pour les tests, j'ai donc branché des PARs a LEDS afin d'avoir un retour visuel de l'état des interrupteurs.

L'adresse de la télécommande récupérée au préalable est utilisée dans le programme.

Le code est envoyer 8 fois afin d'être sûr de la bonne réception à cause des parasites possible.

Le code n'a pas été fonctionnel du 1er coup, les tests ne fonctionner pas, mauvais interrupteur contrôlé, interrupteur ne resté allumé que quelque seconde ou s'allumais et s'éteignait en alternance

```
#include <NewRemoteTransmitter.h>

// 17412246 = adresse de transmission, adresse de la télécommande
//11 = pin de connexion du module hf
//269= durée de la période de transmition en ms
//3= 2^3=8 envoie du code
NewRemoteTransmitter transmitter(17412246, 11, 269, 3);


int var ; //variable corespondent a la commande reçu par port série


void setup() {
  Serial.begin(9600);  // ouverture du port série

}


void loop() {


if (Serial.available() > 0) {
var = Serial.read() ;    //var = comande port série


//totalité des commandes de contrôle des prises
//(var == 'a')   commande exécuté lors de réception d'un 'a' sur port série
// transmitter.sendUnit = commande permettant l'envoie d'une trame à une prise
//transmitter.sendGroup = commande permettant l'envoie d'une trame à l'ensemble des prises
//(x,true) = prise n°x allumée   0= prise n°1 /  1= n°2 / 2= n°3
//(x,false)= prise n°x éteinte
//delay (1000) = pause d'une seconde
```

```
          if(var == 'a'){
                  transmitter.sendUnit(0, true);
                       delay(1000); }

         else if(var == 'b'){
                   transmitter.sendUnit(0, false);
                       delay(1000);}

        else  if(var == 'c'){
                  transmitter.sendUnit(1, true);
                       delay(1000); }

        else  if(var == 'd'){
                     transmitter.sendUnit(1, false);
                       delay(1000); }

         else  if(var == 'e'){
                     transmitter.sendUnit(2, true);
                       delay(1000);}

         else if(var == 'f'){
                  transmitter.sendUnit(2, false);
                       delay(1000); }

// sendgroup = toute les prises

       else if(var == 'g'){
                     transmitter.sendGroup(true);
                       delay(1000);}



       else if(var == 'h'){
                transmitter.sendGroup(false);
                     delay(1000); }
    }
    void loop() {
    }
```

5.2.3Le protocole home Easy:

La communication de la carte Arduino jusqu'aux prises DI-O est réalisé grâce au protocole de communication Chacon : Home easy. Ce protocole est très répandu dans le milieu de la domotique sans fil 433Mhz. Lors du transfert de la carte Arduino aux prises télécommandé, le signal prend deux états. De la carte Arduino jusqu'à l'émetteur HF, le signal passe par la carte imprimé ou un fil en TOUT ou RIEN, 3,3V pour un 1 logique et 0v pour un 0 logique. Et de l'émetteur aux prises, c'est un signal radio qui est envoyé, des ondes haute fréquence à 433,92Mhz, ou un 1 logique est codé avec une émission d'une onde a 433,92Mhz, et un 0 logique, avec aucune émission.

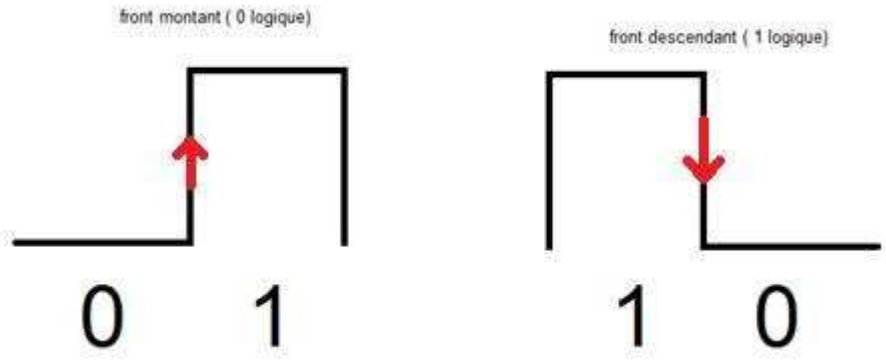
Pour que les prises fonctionnent, elles doivent recevoir une trame complète bien définie, composé de 32 bits.

Les 26 premiers bits correspondent à l'identifiant de la télécommande
Le 27ème bit correspond au numéro de groupe (information non utilisé donc mit à 0)
Le 28ème bit correspond à l'état (ON ou OFF) envoyé, 0=off, 1=on
Les bits 29 à 32 correspondent au numéro de l'interrupteur commandé.

Exemple : commande pour allumer le premier interrupteur

Adresse de la télécommande : 01000010011011000010010110 correspond à 17412246 en décimal
Groupe : 0 (n'est pas utilisé)
Etat : 1 (Active l'interrupteurs)
N° Interrupteurs : 0000 (correspond au 1er interrupteurs)
Soit la trame 01000010011011000010010110010000

Cette trame est transformée en utilisant un codage Manchester, ce qui va transformer le 0 en « 0 1 » et les 1 en « 1 0 » le signal est donc doublé et est maintenant formé de 64 bits.
Le principe est de récupérer l'information au changement d'état, donc un 0 qui est codé en 0 1 est en réalité un front montant (un niveau bas suivis d'un niveau haut)
(Le code Manchester n'a pas de norme, et peut être réalisé de plusieurs façons, les valeurs peuvent être intervertie 0 = 10 et 1 = 01)

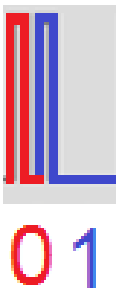


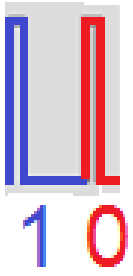
La trame devient maintenant :
0110010101011001011010011010010101011001011001101001 01 10 01010101

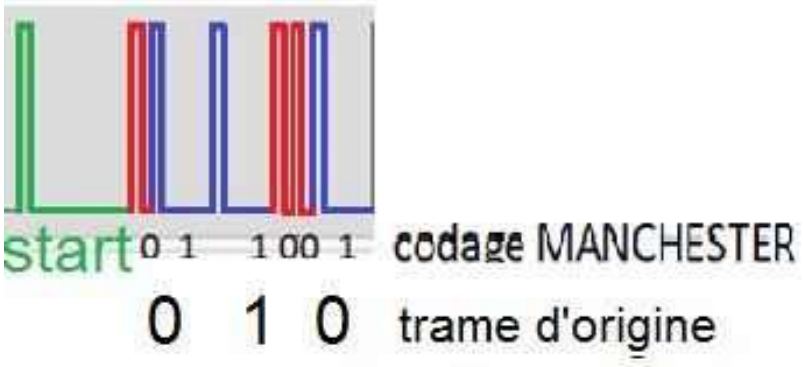
Une deuxième transformation est réalisée, une transformation en “onde”
Nous savons qu’un 0 est traduit par 01 et un 1 est traduit par 10, mais les délais ne sont pas les même, en réalité un 0 est encodé par un niveau haut de 275us puis un niveau bas de 240us, et un 1 est encodé par un niveau haut de 275us puis un niveau bas de 1300us.

Un bit d’origine dure donc 275*2 +240+1300 = 2090 µs ce qui donne une fréquence de 478.47 Hz, un débit de 478.47 Baud/s et 956.94 Bit/s (deux bits pour un symbole).

Avant les données, il y a un niveau haut de 275us puis un niveau bas de 2675us. (bits de Start)

0 logique = 

1logique = 

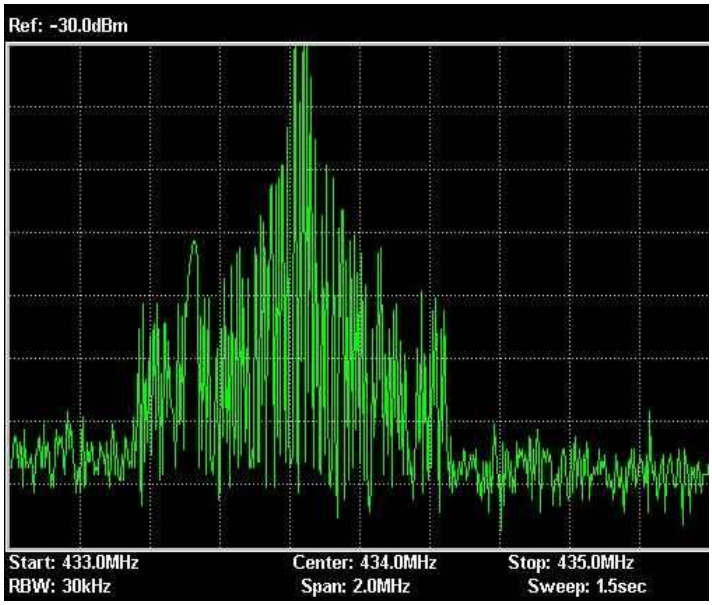


La trame donnée en exemple devient donc :
(Trame récupéré avec un analyseur logique Salae)

En réalité, pour avoir le code d’origine, j’ai tout d’abord récupéré une trame avec l’analyseur logique, pour ensuite traduire manuellement en code Manchester, et enfin récupéré le code d’origine.

Codage Manchester :
0110010101011001011010011010010101011001011001101001 01 10 01010101
Information :
01000010011011000010010110 0 1 0000

Une fois ces transformation réalisé, la trame est envoyé par onde radio a la prise, l'onde est envoyé de façon à ce qu'un 1 soit une émission de 433.92Mhz et un 0 une absence d'émission.
Grace à un analyseur de spectre, j'ai pu récupérer une trace visuelle de l'émission radio.



De plus, pour être sûr que les prises reçoivent correctement les informations malgré les interférences et parasites, la trame est envoyée 8 fois.

5.2.4 Antenne/ émetteur

L'émetteur émet à une fréquence de 433.92 MHZ.

La longueur d'onde est donc :

$$\lambda = \frac{C}{F} = \frac{300\,000\,000\ m/s}{433.92\ Mhz} = 69,14\ cm$$

C= célérité de l'onde
F= fréquence

Pour une meilleure réception, nous pouvons rajouter une antenne quart d'onde, pour savoir la longueur de l'antenne, un calcul simple est réalisé.
Antenne quart d'onde : λ /4 = 17,285 cm

Lors des tests, l'utilisation de l'antenne n'a pas montré son utilité, la distance maximale d'écart que j'ai pu utiliser entre l'émetteur et les prises resté dans le périmètre de portée de l'émetteur seul. Cependant, la portée était déjà de quelque dizaine de mètre avec 3 murs de séparation. La portée théorique de l'émetteur en champs libre est de 50m.

5.3 Raspberry / I2C

5.3.1 La carte Raspberry Pi

La carte Raspberry utilisé, est une Raspberry PI 2 model B.

-Processeur ARM Cortex-A7 de 900 MHz
-Mémoire RAM intégrée de 1 Go
-Contrôleur graphique Broadcom VideoCore IV
-Ports disponibles : 4 x USB, 1 x HDMI, 1 x RJ45
-Lecteur de cartes mémoire micro SD
-Sortie audio Jack (3.5 mm)
-Puissance : 350 mA

La Raspberry Pi est un nano-ordinateur monocarte à processeur ARM.
Le port HDMI permet de connecter un écran, et les ports USB, une souris et un clavier.

Pour notre utilisation, nous utilisons l'OS Raspian, distribué librement par Linux. L'OS est installé sur une carte micro SD à partir d'un ordinateur Windows.

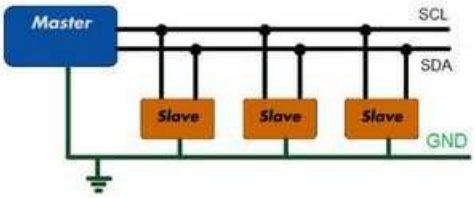
La carte Raspberry sert de base du projet, c'est elle qui contient la plupart des programmes et logicielles, mais aussi le serveur MySql, et c'est à partir de cette carte que se fait la communication I2c.
Quelques programmes qui m'ont étaient utiles on dut être installé, comme Qtcréator pour la réalisation d'une interface graphique. J'ai également initialisé le bus GPIO, UART et I2c qui sont utilisés pour communiquer avec la carte Arduino.

5.3.2 Communication I2C

Le Bus I2C est un bus série synchrone bidirectionnel half-duplex, ce qui signifie qu'il peut y avoir des communications dans les deux sens, mais pas en même temps, de plus, le bus est synchronisé avec une horloge. La communication I2c est une communication Maître/ Esclaves.

Le bus peut contenir plusieurs maitres et plusieurs esclaves, cependant, les communications ne peuvent être seulement de maitre à un ou plusieurs esclaves, et non maitre-maitre, ou esclave-esclave, et toujours à l'initiative du maître .Cependant, rien n'empêche un composant de passer du statut de maître à esclave et réciproquement.

Les connexions sont réalisées en parallèle à l'aide de 3fils :



SDA : donné
SCL : horloge
GND : Masse commune

Chaque composant à une adresse qui lui permet de communiquer, elle est codée sur 7 bits, le 8ème servant à déterminer si c'est une écriture ou lecture. Cette adresse sera envoyée à chaque fois qu'une demande ou information sera envoyé.

Quelque chose à savoir, lié au projet :
Plusieurs vitesses de transmission sont possibles, mais nous utilisons le mode standard à 100kbit/s

-Le codage utilisé est un codage NRZ (non-retour à zéro) ce qui signifie que la tension sur les liaisons n'est jamais à 0v pour ne pas confondre avec une absence de tension du a un disfonctionnement.

-La lecture de chaque bit s'effectue sur SDA après chaque front montant de SCL.

-Au repos, les lignes SDA et SCL sont au niveau haut.

- Start: c'est une transgression de la règle de codage des bits qui est utilisé pour signifier le début d'une trame. Cette condition est caractérisée par le passage de SDA du niveau haut au niveau bas pendant que la ligne SCL est maintenue au niveau haut.

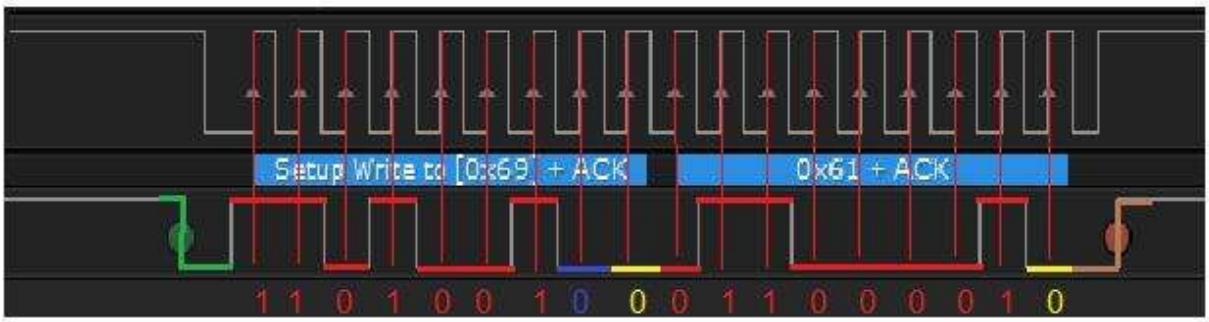
-ACK: est un accusé de réception, mise à 0 si bien reçu, et mise à 1 si la réception a échoué.

-Stop : c'est une transgression de la règle de codage des bits qui est utilisé pour signifier la fin d'une trame. Cette condition est caractérisée par le passage de SDA du niveau bas au niveau haut pendant que la ligne SCL est maintenue au niveau haut.

-R/W : écriture impose un bit à 0 et la lecture un bit à 1 (dans notre cas, seul l'écriture sera réalisée)

Notre communication I2Cest formé de cette façon :

Start – 7bits d'adresse (adresse de l'esclave) –bit écriture(0)-ACK- 8 bits de donné-ACK- stop



Cette trame envoi à l'adresse hexa 69 (écriture) le caractère ascii « a » (tableau Ascii annexe 2)

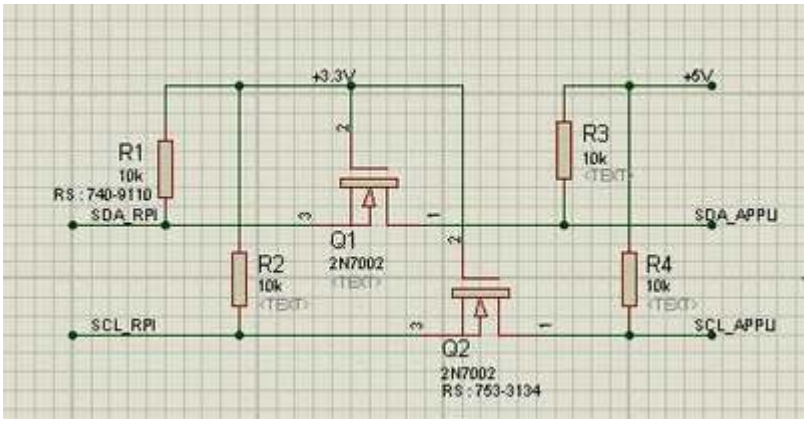
5.3.3 LevelsShifter

La communication I2C sort de la carte Raspberry avec une tension de 3,3V. Hors, pour communiquer avec la carte Arduino et le capteur de température LM76, une tension de 5v est nécessaire.
Pour régler ce problème, j'ai réalisé un LevelsShifter qui a pour but de monter la ligne SCL et SDA de la Raspberry à 5v sans perturber le signal envoyé.

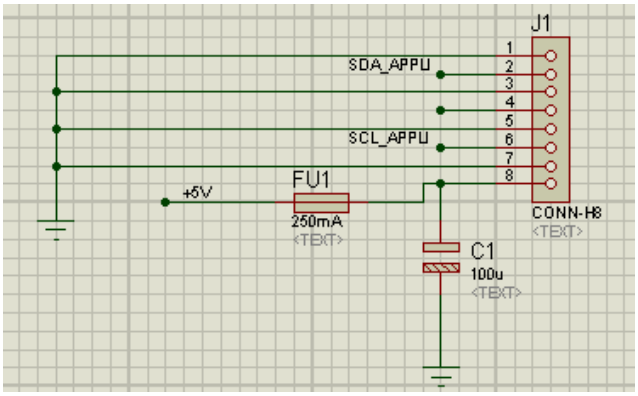
Une carte électronique intégrant un LevelsShifter a été créé, elle est composée d'un LevelsShifter, d'un connecteur Rj45 pour la communication I2c et d'un connecteur 20pin pour la connexion en tant que Schild sur la carte Raspberry. La carte comprend également l'alimentation 5v du capteur de température, qui est active sur l'un des fils du câble Ethernet utilisé pour la liaison
Les schémas électroniques on était réalisé avec le logiciel Proteus ISIS.

Le LevelsShifter utilisé est bidirectionnel, il permet l'adaptation de tension des deux côtés, le passage de 3,3v a 5v pour les donné sortant de la carte Raspberry, et de 5v a 3,3 v pour les donné rentrante. Cela afin de permettre le dialogue entre maître et esclave.

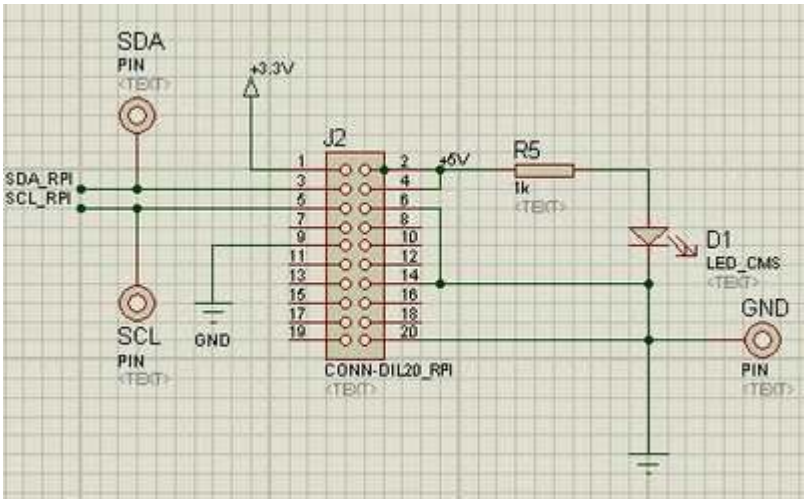
La partie LevelsShifter est composée de 5 résistances de 10kohm et deux transistors Mos. Le schéma original utilisé a été trouvé sur internet, un lien fourni par le lycée, mais reproduit sur le logiciel Proteus. https://www.youtube.com/watch?v=t-yuYasIKtY



La partie connexion I2c est réalisée avec un connecteur RJ45, les connexions seront faites avec un câble Ethernet droit, mais il ne faut pas brancher de connexion Ethernet sur ce périphérique, cela pourrait l'endommager. Cette partie de la carte comprend également un fusible 250ma pour éviter les surtensions du a l'alimentation du capteur lm76, les 5v sont récupéré directement sur le bus GPIO de la carte Raspberry. La liaison I2C passe par les fils 2 et 6, et les fils 1, 3,5 et 7 sont reliés à la masse.

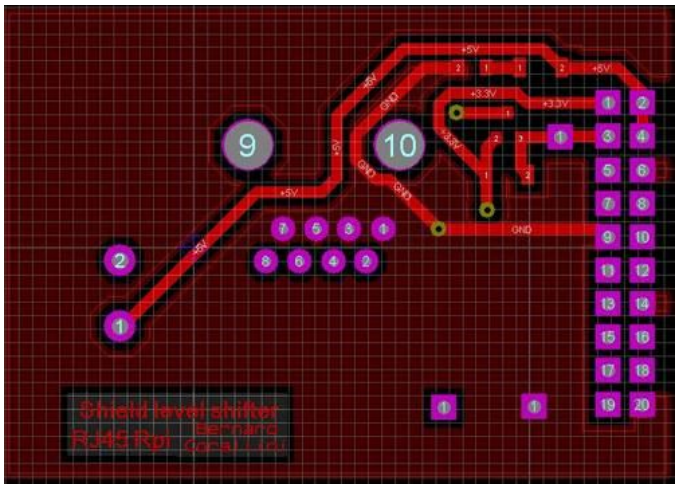


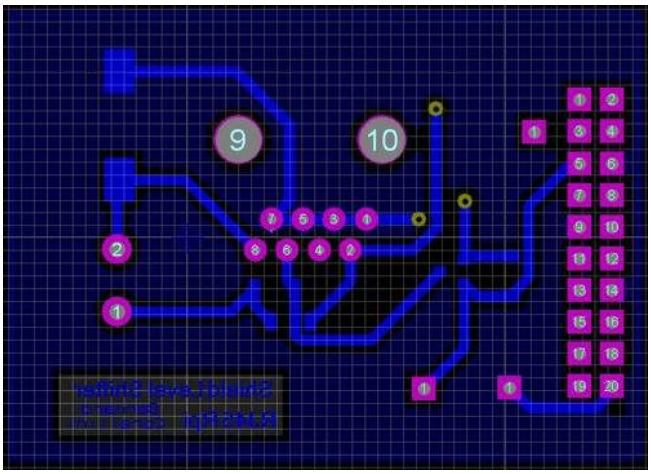
La troisième partie de la carte est le connecteur qui sera branché sur la carte Raspberry. Elle comprend un connecteur 20 broche, une LED qui a pour but de montrer visuellement une présence de tension sur la ligne 5v, une résistance lier à celle si, et 3 point de test permettant une prise de mesure facile. Les broches 3 et 5 sont respectivement SDA et SCL de la liaison I2c.



Le routage a était réalisé sur le logiciel Proteus ARES. Ils permettent la future fabrication de la carte électronique par l'emplacement des composants, leur taille, mais aussi la taille des pistes et le plan de masse. Tous les composants de la partie LevelsShifter sont des composants CMS (Composant monté en surface), ce sont des composants plus petit qui ne nécessite pas de traverser la carte pour être soudé.

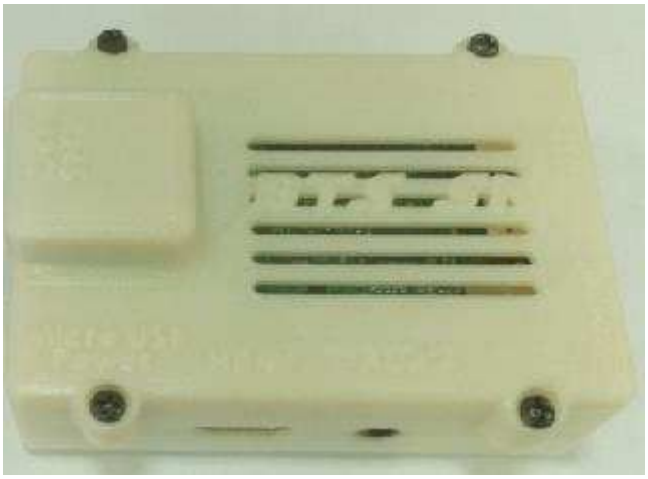
Face avant et arrière de la carte (voir annexe2 pour placement des composants) :

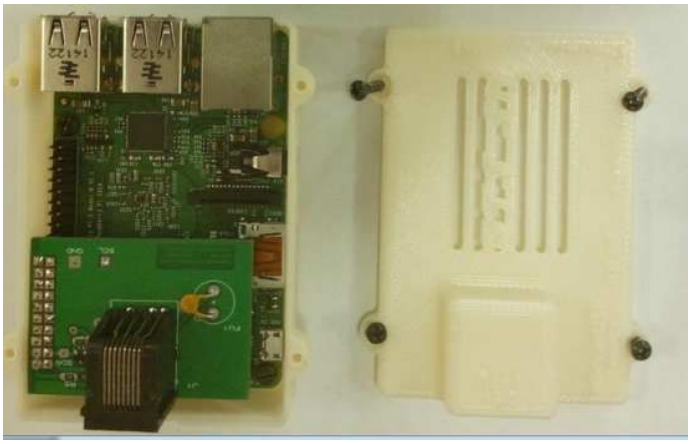




La fabrication de la carte a était sous-traiter par une entreprise après leur avoir envoyé les fichiers Gerber, mais la mise en place des composant a étais réalisé au lycée. Après réception de la carte, j'ai réalisé la soudure des composants, en commençant par les CMS (résistance, LED et transistor) puis par taille croissante (les connecteurs, condensateur, point de test, connecteur rj45).

Le lycée a réalisé un boitier sur mesure pour intégrer le LevelsShifter à la carte Raspberry, ce qui lui donne un côté esthétique et surtout une meilleure protection.





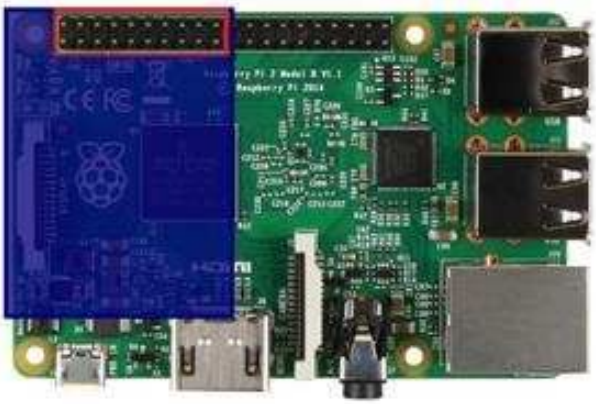
Pour permettre une utilisation facilitée aux autres membres du groupe utilisant le LevelsShifter, j'ai réalisé une fiche de lecture croisée expliquant sa mise en place et sa mise en service.

Fiche de lecture croisée

Level Shifter

Attention, le level shifter doit être installé hors tension.

Il doit être installé sur les premières pins, le corps de la carte doit être du coté intérieur de la Raspberry. (Voir schéma)



Vous pouvez maintenant mettre sous tension la Raspberry, qui devrait s'allumer.

Le bus i2c doit être en marche sur la Raspberry, les lignes de commande suivantes indiquent l'état du bus i2c :

dmesg | grep i2c sudo ls /dev/i2c*

Si le bus n'est pas accessible :

- Ouvrir le fichier : /etc/modules
- ajouter la ligne : i2c-bcm2708
- ajouter la ligne : i2c-dev
- Puis lancer l'interface de configuration : sudo raspi-config
- Advanced Options ... valider tout ce qui concerne l'i2c
- accepter : reboot

Le Level Shifter est équipé d'un port rj45, il est utilisé pour le relier à un équipement i2c comportant également un port rj45. Le câble utilisé doit être un **câble Ethernet droit**.

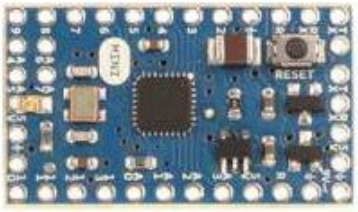
Pour réaliser un test de connexion, effectuer la commande « i2cdetect –y 1 »
Un tableau doit apparaitre, si un chiffre se trouve dans le tableau, c'est que l'équipement est bien détecté (adresse en hexadécimal), et par conséquent, l'alimentation et l'i2c du Level Shifter fonctionne.

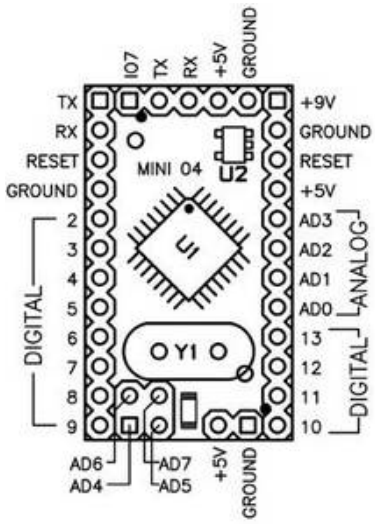
Bernard Alex
BTS SN – EC 14/03/2016

5.4 Mini Arduino/ carte module HF

5.4.1 Mini Arduino

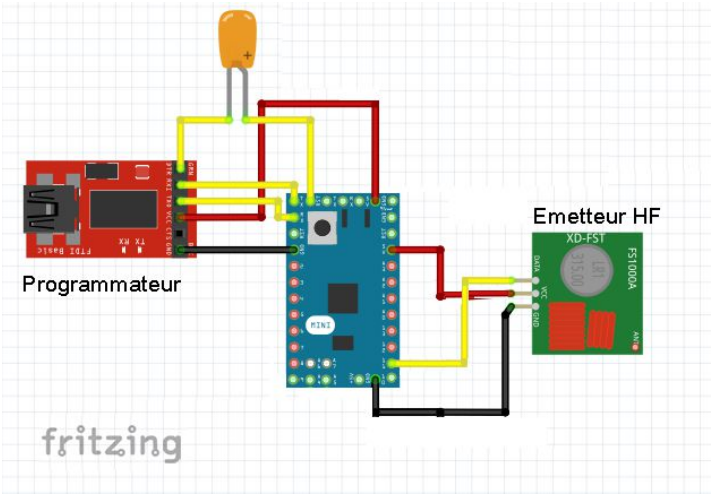
La carte Mini Arduino utilise le microcontrôleur ATmega328, le choix de cette carte est sa facilité de programmation et également le fait qu'elle se comporte comme une carte Arduino UNO avec laquelle nous travaillons régulièrement, tout en étant plus concentrée. L'inconvénient de cette carte est qu'elle est très fragile et une connexion incorrecte peut facilement détruire la carte.





Microcontrôleur ATmega328
Tension de fonctionnement 5V
Tension d'entrée 7-9 V
E / S numériques Pins 14
Entrée analogique Pins 8
Intensité par E/S: 40 mA
Mémoire Flash 32 Ko (dont 2 Ko utilisé par boot loader)
SRAM 2 Ko
EEPROM 1 Ko
Vitesse d'horloge de 16 MHz

Pour tester la carte, j'ai utilisé le même programme qu'avec la carte Arduino Uno. Le programmateur une fois relier au pc permet de programmer la carte avec le logiciel Arduino habituel.
Cependant, l'ajout d'un condensateur sur la broche reset a été nécessaire pour une compilation correcte du programme.
La connexion de programmateur et de l'émetteur sur la carte est montée comme sur ce schéma réalisé sur le logiciel Fritzing. L'alimentation des composants est fournie par la connexion du programmateur au pc.



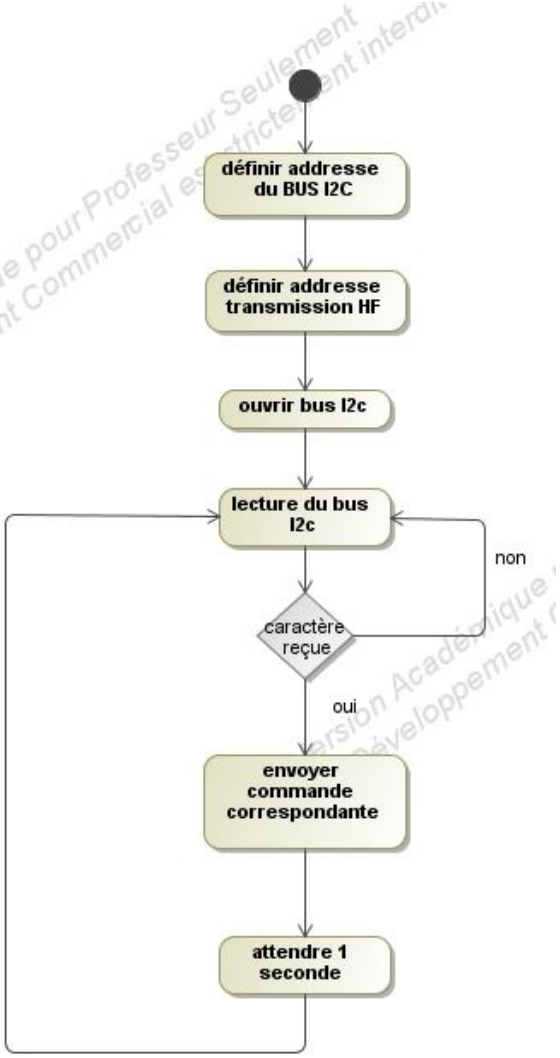
5.4.2 Modification du code série/ i2c

Le code précédemment utilisé réagis avec les caractères reçu sur le port série, cependant, la carte Raspberry enverra les caractères via le bus I2c, une modification du programme est donc nécessaire. L'ajout de la librairie wire.h est utilisé pour la communication i2c.

```
#include <Wire.h>    // librairie pour comunication I2c
#include <NewRemoteTransmitter.h>   // librairie communication home easy

#define SLAVE_ADDRESS 0x69       // adresse I2c de la carte arduino
int dataReceived = 0;       //initialisation de la variable dataReceived a 0


NewRemoteTransmitter transmitter(17412246, 11, 269, 3);
// 17412246 = adresse de transmission, adresse de la télécommande
//11 = pin de connexion du module hf
//269= durée de la période de transmition en ms
//3= 2^3=8 envoie du code



int var ; //variable correspondant a la commande reçu par communication I2c



void setup() {
  Serial.begin(9600);  // ouverture du port série
  Wire.begin(SLAVE_ADDRESS);  // initialise communication Arduino "esclave"
    Wire.onReceive(receiveData); //définit la fonction à appeler sur réception
                                 //de données en provenance du maître
}

void receiveData(int byteCount){

  var = Wire.read();  // variable var= données en provenance maître



//totalité des comande de controle des prises
//(var == 'a')   comande exécuté lors de réception d'un a par I2c
// transmitter.sendUnit = commande permettant l'envoie d'une trame à une prise
//transmitter.sendGroup = commande permettant l'envoie d'une trame à l'ensemble des prises
//(x,true) = prise n°x allumée   0= prise n°1 /  1= n°2 / 2= n°3
//(x,false)= prise n°x éteinte
//delay (1000) = pause d'une seconde
```

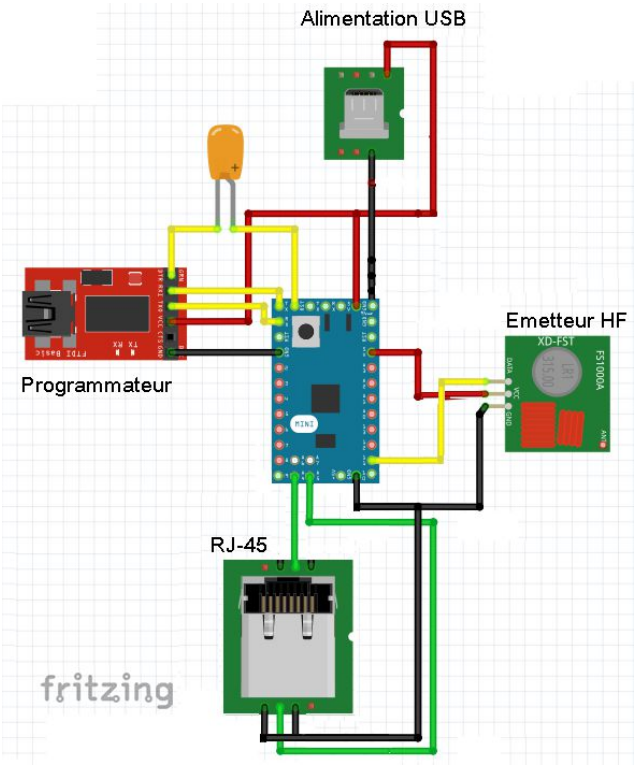
```
    if(var == 'a'){
            transmitter.sendUnit(0, true);
                delay(1000); }


    else if(var == 'b'){
             transmitter.sendUnit(0, false);
                delay(1000);}


  else  if(var == 'c'){
            transmitter.sendUnit(1, true);
                delay(1000); }


  else  if(var == 'd'){
             transmitter.sendUnit(1, false);
              delay(1000); }


    else  if(var == 'e'){
             transmitter.sendUnit(2, true);
                delay(1000);}



      else if(var == 'f'){
               transmitter.sendUnit(2, false);
                  delay(1000); }

// sendgroup = toute les prises

    else if(var == 'g'){
              transmitter.sendGroup(true);
                delay(1000);}


      else if(var == 'h'){
               transmitter.sendGroup(false);
                  delay(1000); }


}

void loop() {

}
```

Le programme est expliqué grossièrement par se diagramme d'activité et ce tableau.



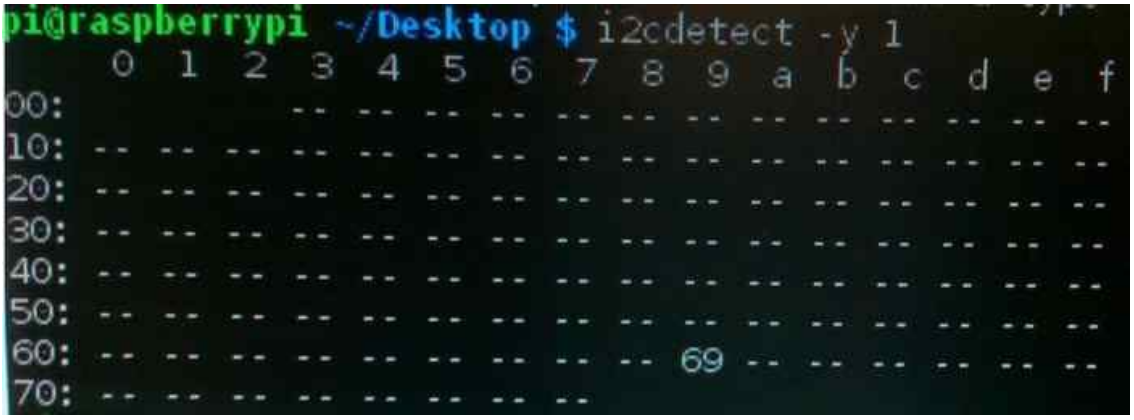
Commande	code hexadécimal	caractère Ascii
allumer intérrupteur n°1	0x61	a
éteindre intérrupteur n°1	0x62	b
allumer intérrupteur n°2	0x63	c
éteindre intérrupteur n°1	0x64	d
allumer intérrupteur n°3	0x65	e
éteindre intérrupteur n°1	0x66	f
allumer tout les intérrupteurs	0x67	g
éteindre tout les interrupteurs	0x68	h

Un nouveau test est réalisé, avec cette fois, l'envoi d'une commande via la Raspberry par i2c en passant par le LevelsShifter et un câble Ethernet droit. Le nouveau câblage de la carte Arduino dispose d'un connecteur Rj45 relié aux broches analogique 4 et 5 utilisé pour la liaison I2C et de sa propre alimentation en micro USB (SDA et SCL).



Pour le test, j'utilise une nouvelle fois les Par à LED branché sur les prise DI-0.

Je réalise tout d'abord un test de connexion I2c avec la commande « i2cdetect –y 1 »



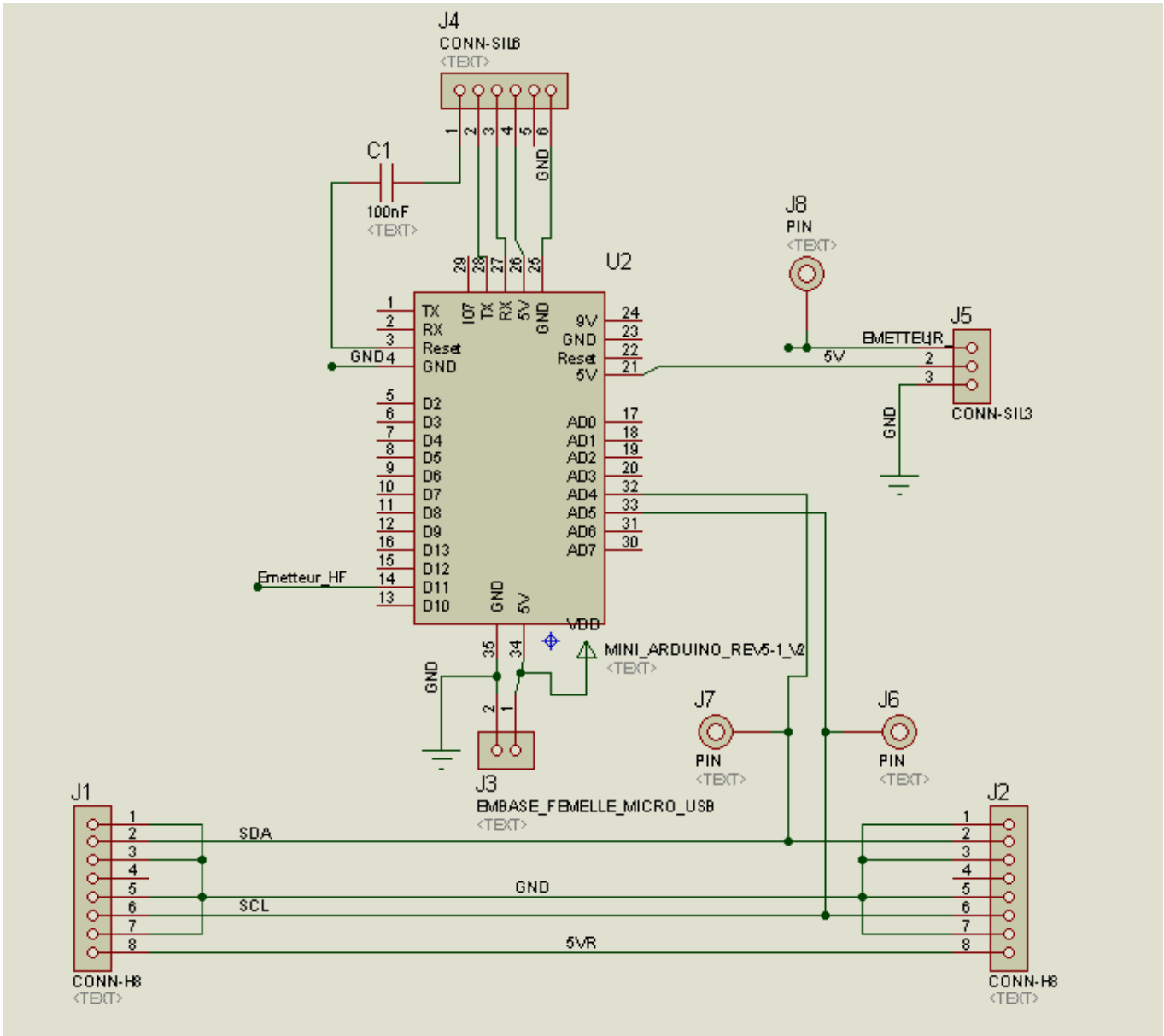
A partir du terminal de la carte Raspberry, j'envoie la commande « i2cset –y 1 0x69 0x61 »
Qui correspond à l'allumage de la prise n°1. (0x69 = adresse de l'esclave 0x61 = caractère « a »)

Les éclairages s'allument correctement, dans l'ordre voulut, le contrôle des prises à partir de la carte Raspberry et le bus I2c est opérationnel.

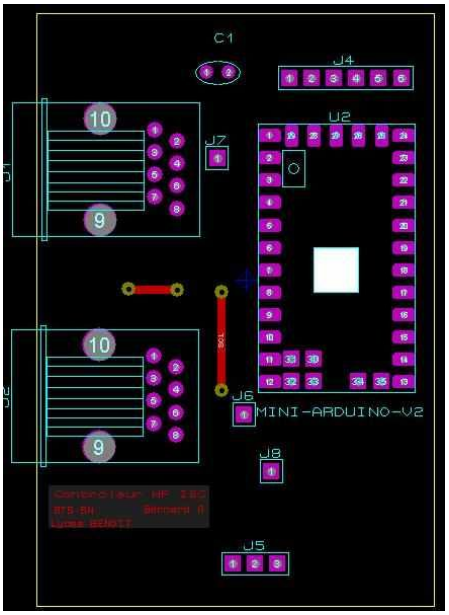
5.4.3 Réalisation de la carte électronique Arduino-module HF

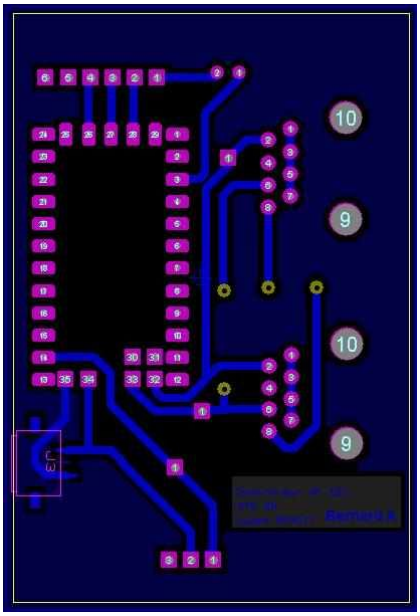
Le code étant opérationnel, il faut maintenant réaliser le montage des composants sur une carte électronique. Cette deuxième carte électronique comprend la carte mini Arduino, le module HF, deux ports Rj45 (l'un pour une connexion à la carte Raspberry, l'autre pour pouvoir connecté la sonde de température), et dispose de sa propre alimentation mini USB. Il y est également inclus 6 broches male afin de connecter le programmateur en cas de besoin, le condensateur est présent permanemment sur la carte. 3 broches de test sont présentes afin de faciliter les mesures, elles sont reliées aux lignes SDA et SCL du bus I2c et à la ligne DATA de l'émetteur Hf. La masse est récupérable sur l'une des 6 pins utilisés pour le programmateur (la broche situé le plus à l'extérieur de la carte). L'alimentation 5V en sortie de la Raspberry est seulement reliée entre les deux ports Rj45 et n'est pas utilisé par celle-ci.

Schéma réalisé avec le logiciel Proteus Isis :



Le routage de la carte est effectué avec le logiciel Proteus Ares. Contrairement à la première carte, celle-ci est une simple face, ce qui signifie que les soudures ne seront effectué que d'un côté du circuit imprimé, en l'occurrence dans notre cas par le dessous (face bleu). Par conséquent deux straps sont utilisé, ce sont des fils qui ont pour seul but de contourner une piste, de « passé par-dessus» (les straps sont les fils rouge présent sur la 1ere image du routage)





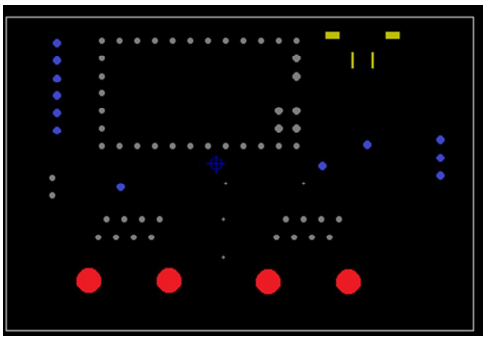
Procédé de fabrication :

La carte est entièrement réalisée dans l'atelier du lycée.

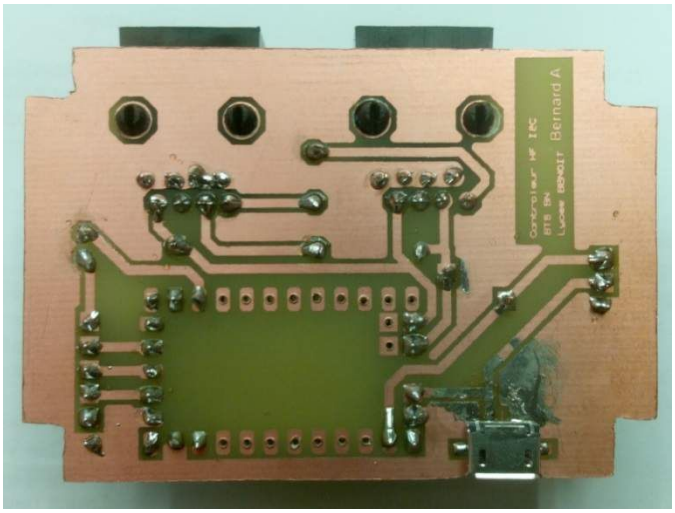
La fabrication d'une carte électronique s'opère sur un circuit imprimé. Il s'agit d'une plaque d'époxy gravée de sillons de cuivre. Pour savoir comment vont s'organiser les éléments, nous imprimons un typon, un plan transparent qui dessine les éléments de la carte électronique. Le typon (voir anexe3) se plaque ensuite sur la plaque époxy et grâce à un procédé chimique se dessine directement sur ce support. Ces sillons tracés à la surface de la plaque sont ensuite soumis à un rayonnement UV, ce procédé s'appelle l'insolation. Il a pour mission de fixer des particules de cuivre afin de créer les lignes du circuit imprimé de la carte. Comme en photographie on utilisera alors un révélateur pour que la plaque soit gravée de manière définitive. Plusieurs procédés existent qui mêlent du chlorure de fer par exemple pour rendre le circuit imprimé définitivement opérationnel. La fabrication d'une carte électronique nécessite ensuite de fixer les composants sur la plaque grâce au perçage opéré par une perceuse à colonne suivi d'une soudure à l'étain.

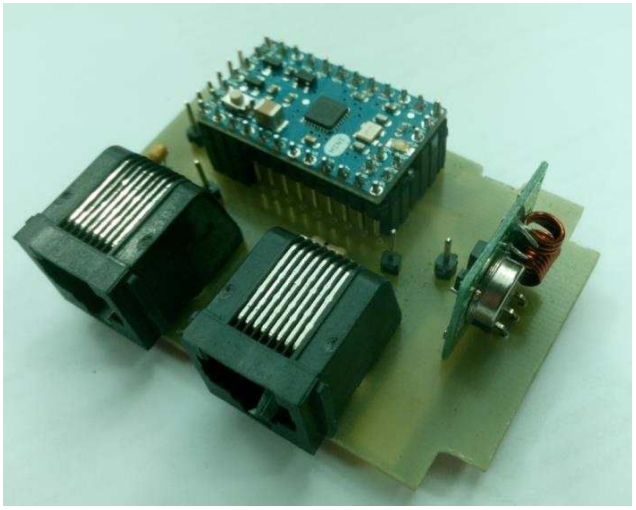
Plan de perçage :

Taille du foret : Rouge : 3mm/bleu : 1mm /gris : 0.8mm /jaune : aucun perçage (composant cms)

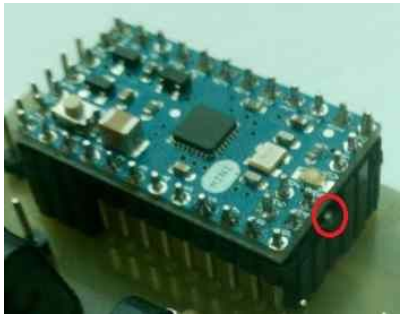


Le premier composant à être soudé est l'alimentation micro USB, c'est un composant CMS dont la soudure est délicate, composant fragile est pate de connexion très fine. Vien ensuite les straps, les connecteurs Header, les point test, le condensateur, l'émetteur HF et enfin les connecteurs rj45.
La carte mini Arduino est installer sur des header, elle n'est pas souder directement pour pouvoir la remplacer en cas de besoin.
Une fois les composant soudé, je test les piste afin de vérifier l'absence de court-circuit. Le test est pratiqué à l'aide d'un multimètre en position testeur de LED.

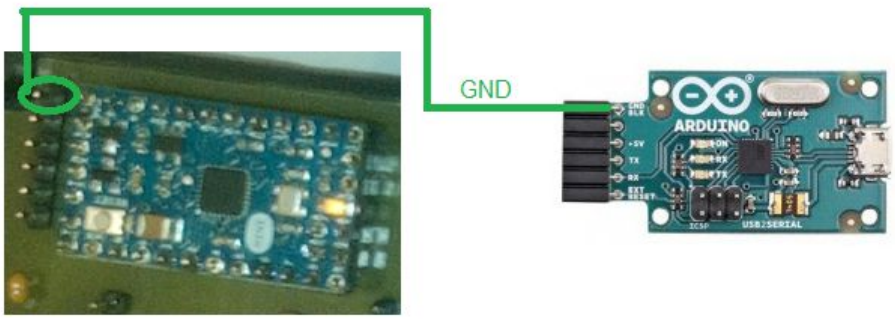




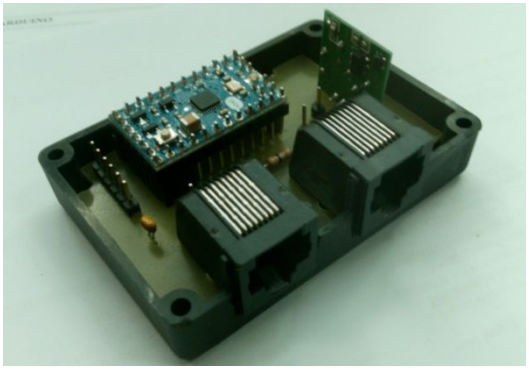
La carte Arduino à un sens, pour éviter de la brancher à l'envers, un détrompeur a été mis en place, ils s'agissent simplement d'un point de soudure sur l'un des connecteurs non utilisé pour boucher celui-ci. Dans le sens normal, aucun connecteur de la carte Arduino ne se trouve à ce niveau, mais si la carte est inversée, la carte ne peut pas être connectée grâce à ce détrompeur.



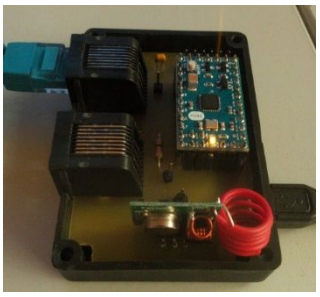
Le même procéder n'a pas pu être réalisé pour le programmateur, cependant son sens est important si l'on ne veut pas dégrader le matériel, ou tout simplement pouvoir programmer la carte Arduino. La broche GND (masse) doit être connectée sur la pin le plus à l'extérieur de la carte.



J'ai également fabriqué un boitier pour contenir la carte électronique afin de la rendre plus esthétique et plus résistante. La connexion I2c étant en parallèle, il n'y a pas de port défini pour la carte Raspberry et le capteur LM76.



Une antenne a été rajouté afin d'optimiser la portée (antenne ¼ d'onde 17.3 cm).



Les test sont a nouveau fait pour s'assuer que la carte et les liaisons fonctione. L'utilisation des PARs LED semble étre le plus utiles donc réutilisé pour ceux-ci.

Les comandes sont envoyés via le moniteur de la carte Raspberry, en testant tout les caractère programé (a,b,c,d,e,f,g,h) affin de vérifier le bon fonctionement du programme et de la carte. La ligne de comande est constituer de la même façon que les test précédent : « i2cset –y 1 0x69 0x61 » Qui correspond à l'allumage de la prise n°1. (0x69 = adresse de l'esclave 0x61 = caractère « a ») aucun problème survenu, toute les commandes fonctionne.

Lors des test, j'ai constaté que la carte n'était pas détecté à tout les coup par la carte Raspberry. Après quelque test, j'ai pu trouver le moyen de remédier a se problème. Pour connecter la carte, il faut d'abort l'alimenter par le connecteur mini USB situé a l'arrière, attandre quelque seconde que la Mini arduino s'initialise corectement, et ensuite connecté la liaison I2C. j'ai également testé les 2connecteur Rj45 qui fonctione correctement.

5.5 Interface graphique

L'interface graphique a été réalisée avec le logiciel QtCreator sur la carte Raspberry pi. Ce programme permet une utilisation plus rapide et pratique à l'aide d'un outil visuel. Une application qui se comporte comme une télécommande, avec des boutons pour allumer et éteindre les interrupteurs, et également un bouton pour tester la connexion avec la carte Arduino. Les librairies utilisées sont wiringPi.h et wiringPiI2c.h (code commenté voir annexe 4). J'ai partagé mon code Qt avec plusieurs membres de la classe dont certain différent de mon groupe de projet.

Un test de connexion est présent sur l'interface graphique, il fonctionne en récupérant grâce aux commandes « fd=wiringPiI2CSetup(0x69); » et « erreur = wiringPiI2CRead(fd); »
Ces commande permette de récupéré une valeur qui est différente si l'esclave est connecté ou non. Si l'esclave n'est pas connecté, la variable « erreur » a pour valeur -1 et s'il est connecté, il obtient une valeur positive (127). A partir de ces résultats, j'ai créé un programme qui permet d'afficher dans un label si l'esclave est connecté.

```
int erreur;
ui->setupUi(this);
wiringPiSetupGpio();
fd=wiringPiI2CSetup(0x69);
erreur = wiringPiI2CRead(fd);

if(erreur < 0)
{

    ui->label_4->setText("erreur connexion");
    ui->label_4->setStyleSheet("color: #FF0000;");

}
else
{
    ui->label_4->setText("connexion reussie");
    ui->label_4->setStyleSheet("color:#096A09;");
}
```

S'il n'y a pas de connexion, un message d'erreur s'affiche en rouge, à l'inverse, si la connexion est réussie, un message le confirmant est affiché en vert. Les couleurs sont en valeur hexadécimal sous forme de 3octet, un pour chaque couleur primaire soit (0= 0% -> 255= 100%):

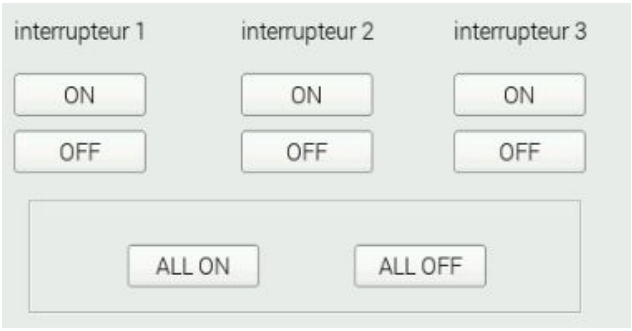
ff0000 = 255 rouge/ 0 vert/ 0bleu

096A09= 9rouge/ 106 vert / 9 bleu

J'ai également ajouté un bouton « Test » afin de réaliser le test de connexion une fois le programme ouvert, afin de ne pas avoir besoin de le relancer.

```
void MainWindow::on_pushButton_9_clicked()
{
    int erreur;

    erreur = wiringPiI2CRead(fd);
    if(erreur < 0)
    {
        ui->label_4->setText("erreur connexion");
        ui->label_4->setStyleSheet("color: #FF0000;");

    }
    else
    {
        ui->label_4->setText("connexion reussie");
        ui->label_4->setStyleSheet("color:#096A09;");
    }
}
```

Pour l'allumage d'un interrupteur, le code est très simple, lors de l'appui d'un bouton,



Commande	code hexadécimal	caractère Ascii
allumer intérrupteur n°1	0x61	a
éteindre intérrupteur n°1	0x62	b
allumer intérrupteur n°2	0x63	c
éteindre intérrupteur n°1	0x64	d
allumer intérrupteur n°3	0x65	e
éteindre intérrupteur n°1	0x66	f
allumer tout les intérrupteurs	0x67	g
éteindre tout les interrupteurs	0x68	h

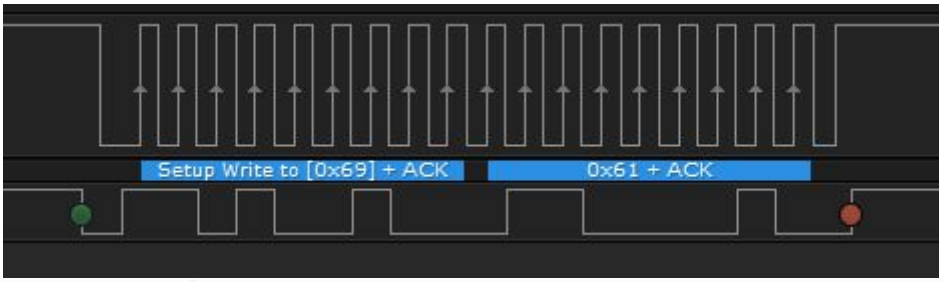
Une commande « wiringPiI2cWrite » est envoyé. Par exemple, pour l'allumage de la première prise, la commande est de forme : « wiringPiI2CWrite(fd, 0x61); » fd étant l'adresse de l'esclave, dans le cas présent, 0x69 et 0x61 étant le caractère « a » utiliser pour allumer la prise n°1.

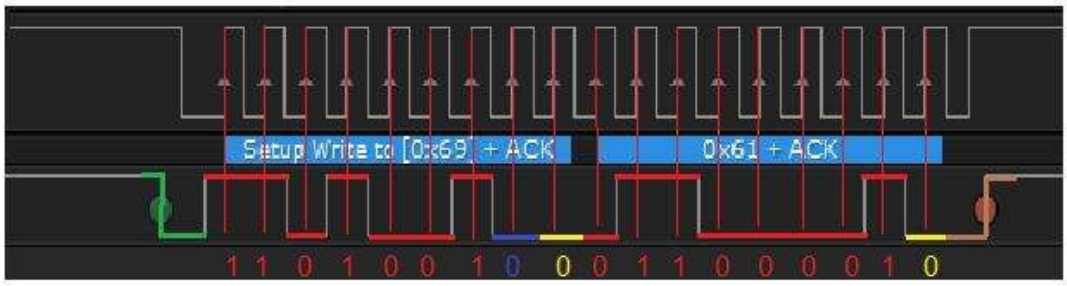
```
//---------------Allumage chauffage-------------------//
void MainWindow::on_pushButton_clicked()
{
    wiringPiI2CWrite(fd, 0x61);
}
```

Grace a un analyseur logique Salae, j'ai pu récupérer la trame envoyé (vue précédement dans la partie I2c) :

SCL=

SDA=





Ox69= 110 1001b

0x61= 0110 0001b

Comme nous pouvont le constater, la trame envoyé est minimaliste.

La 'form' de l'interface graphique est celle-ci :
(insertion du logo de l'entreprise en tant qu'icône)

Elle permet une utilisation plus facile est à la portée de tous. Pour lancé l'application, il faut taper dans le moniteur, dans le répertoire ou se trouve le fichier exécutable, la commande « sudo./Mari-chau ».

Un test est réalisé avec le montage complet des deux cartes électroniques. Le test est effectué comme pour les premiers à l'aide de PARs LED branché sur les prises DI-O dispersé a différent endroit. chaque commande est envoyé à tour de rôle, et pour chaque bouton liée à un ou plusieurs interrupteurs, ceux si réponde correctement par l'allumage des PARs. Bouton interrupteur 1 on = Par 1 allumé / Bouton interrupteur 1 off = par 1 éteint
Bouton interrupteur 2 on = Par 2 allumé… Résultat positif pour tous les boutons.

Pour tester le bouton « test », j'ai débranché le câble Ethernet relier à la carte Arduino et appuyer sur le bouton test, qui change aussi tôt le statut « connexion réussie » en « erreur connexion » avec le changement de couleur prévu. (vert / rouge)

L'intégralité du programme peut se présenté sous cette forme :

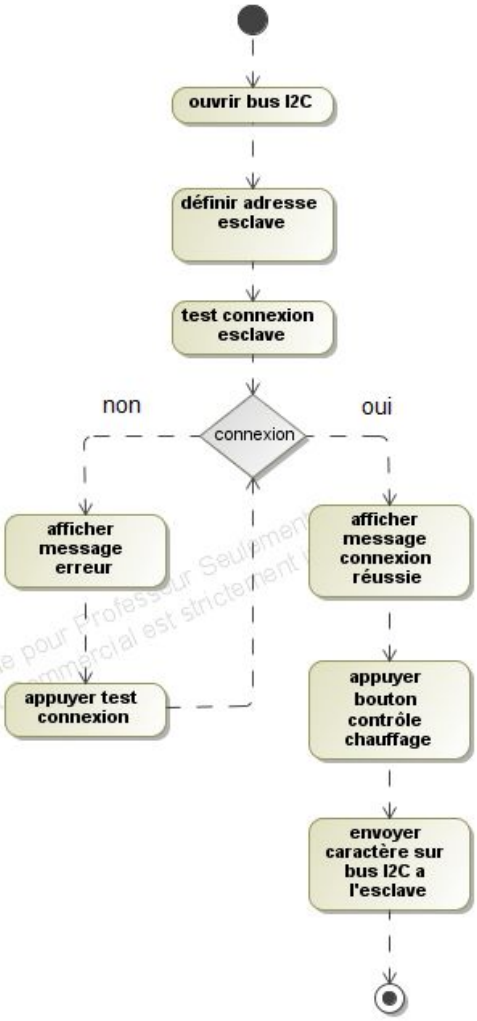


Diagramme de séquence du système complet partie EC en annexe 5

5.6 L'installation :

La mise en place du matériel et très simples :

- La carte Raspberry doit être équipée du levelsShifter avant son alimentation. Pour son installation, voir fiche lecture croisé chapitre 5.3.3 LevelsShifter.

-La carte Raspberry est ensuite relier a un écran par câble HDMI, un clavier et une souris sur les connecteurs USB, et une alimentation 5v micro USB, alimentation secteur équiper d'un transformateur 230/5v

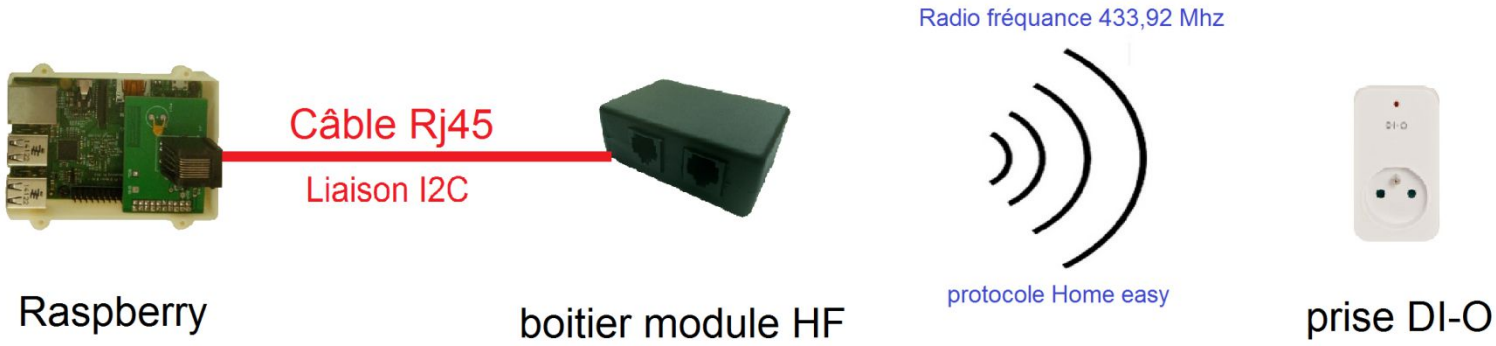
-Le boitier Module HF est alimenté de la même façon que la carte Raspberry par une alimentation 5V Micro USB.

- les deux boitiers sont reliés entre eux par un câble Rj45 droit, il faut relier le connecteur Rj45 du LevelsShifter a l'un des deux connecteurs Rj45 du boitier module HF.

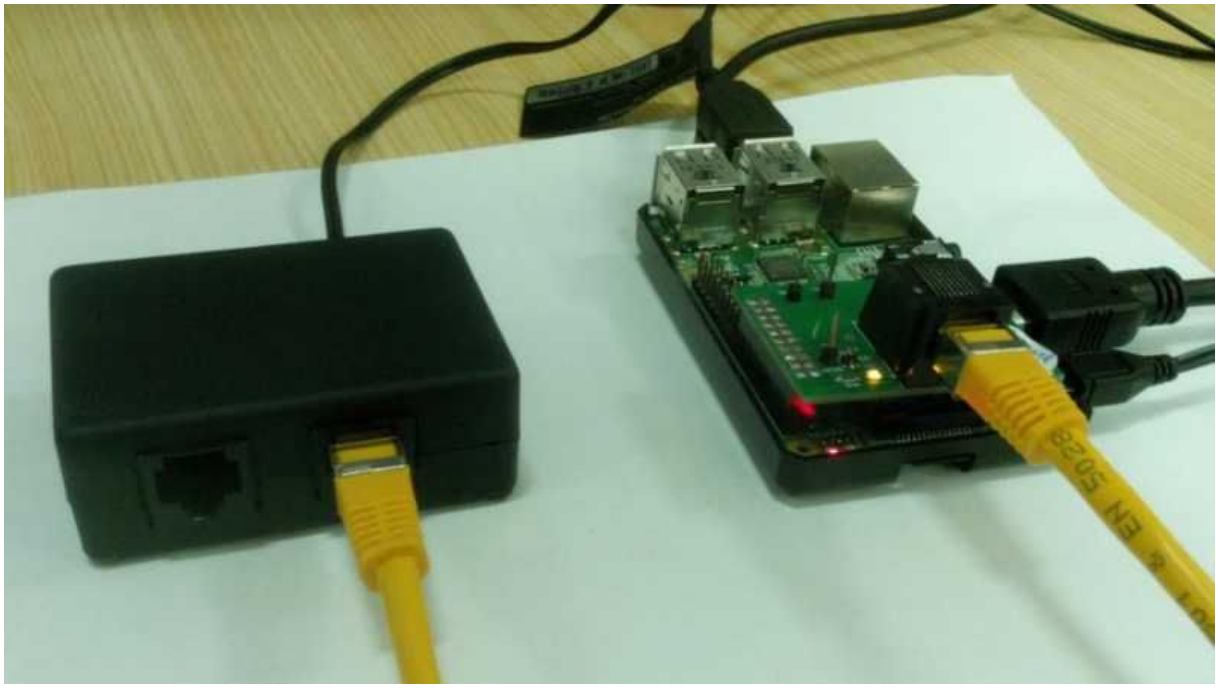
-les prises DI-O sont branchées sur secteur, si les prises ne sont pas encore programmées, elles doivent l'être à ce moment-là. La programmation se fait comme indiqué au chapitre 5.2.1 prise Chacon.
Les prise communique avec la carte module HF par liaison radio fréquence 433Mhz.

-le chauffage électrique ou autre matériel à contrôler doit être branché sur les prises DI-O.

-Utilisation de l'interface graphique pour le contrôle des prises. Pour ouvrir l'interface graphique, il faut taper dans le moniteur, a l'emplacement de l'exécutable, la commande «sudo./Mari-chau ».



5.7 Test supplémentaire.



J'ai réalisé différents test une fois les programmes terminé,

L'utilisation d'un câble Ethernet droit 80m : pour tester le câble, j'ai d'abord vérifié la connexion avec un câble court habituel. Une fois assuré que la connexion est réussie, j'ai interconnecté les deux cartes à l'aide d'un câble de 80 m. après un appui sur le bouton test de l'interface graphiques, le label m'informant de la connexion passe sur « erreur connexion » en rouge (pas de changement tant qu'il n'y a pas d'appuis sur le bouton). Pour m'assurer que sel avien bien de la longueur du câble, j'ai tout d'abord réutilisé le câble court pour vérifier si la connexion pouvais être établi, ce qui fut le cas. J'ai ensuite testé la continuité des fils du câble de 80m à l'aide d'un testeur approprié (vérification de l'état du câble). Le câble en bon état, la longueur est donc la cause du disfonctionnement. Conclusion : Un câble trop long ne peut pas être utilisé.

L'utilisation d'un câble Ethernet droit 15m : tout comme pour le premier câble, j'ai d'abord testé la connexion avec le câble habituel, puis ensuite connecté les deux cartes a l'aide d'un câble droit de 15m. Après l'appui du bouton de test, le label resta sur connexion réussi, j'ai donc appuyé sur l'un des boutons pour commander une des prises, qui s'activa. Conclusion : un câble de 15m peut être utilisé pour relier les deux cartes.

Test de portée : pour le test de portée, j'ai installé les prises DI-O à 3 endroits différents, de plus en plus éloigné de l'émetteur HF. Le premier a une vingtaine de mètres en champs libre, le deuxième une vingtaine de mètres avec 3 murs dans le champ démission, et un a plus de quarante mètres avec 3 murs également dans le champ démission du signal. Après l'envoi de la commande d'activation de tous les interrupteurs, je suis allé voir leur état. Tous ont capté la commande et ce sont allumé. Conclusion : l'émetteur a une portée suffisante pour émettre dans une maison ou des bureaux dans une périphérie de 40m de distance.

5.8 Conclusion

Toute la partie étant à ma charge a été effectuée en respectant les contraintes imposées (programme, matériel et communication). De plus le budget de 300€ a été respecté pour un pris d'environ 118 € (annexe 6).

Il ne reste plus qu'à rassembler toute les parties du projet des différents étudiants afin que le projet soit opérationnel dans sa totalité.

Certaine modification liée a ma partie sont possible :

- Remplacer la carte Mini Arduino par un contrôleur attiny 85. Les 8 broches de ce composant suffiraient pour faire fonctionner la liaison I2C et l'émetteur HF. Ce micro contrôleur ne coute que 1.92€ contrairement a la carte Mini Arduino qui elle coute 19,43 €. Un gain de place et d'argent.

- L'ajout d'un témoin lumineux pour le fonctionnement du boitier Arduino afin de vérifier si la carte est correctement alimentée.

- L'utilisation de la carte Arduino-Module HF est remise en question, la liaison I2c ne pouvant pas être très longue, l'émetteur HF pourrait être installé directement sur la carte Raspberry, se qui engendrerait moins de coût.

- Dans le même cas, l'utilisation du Bus I2C est une mauvaise Idée, mais obligatoire dans les contrainte de réalisation du projet, en effet il pourrait être remplacé par une liaison filaire longue porté comme un protocole Ethernet Rj45, ou utiliser un répéteur.

- L'intégration d'un récepteur Hf dans le boitier Arduino-module Hf afin de pouvoir programmer plus facilement L'adresse utilisé pour l'émission des commandes.

- Les prises Chacons DI-O ne sont approprié au projet, en effet il aurait fallu un équipement pouvant envoyer un retour d'état afin de confirmer la réception de la commande et sa réalisation. De plus le principe de l'appareil est simple a réaliser, et sa fabrication aurait était possible se qui aurait couté bien moins cher. (prix prise DI-O = 31.90€) cependant sont utilisation était mentionner dans les contrainte de réalisation du projet.

- L'interface graphique pourrait intégrer un test de connexion permanant (programme boucle) afin de ne pas avoir à appuyer sur le bouton, et êtres avertie en cas de coupure de connexion entres les deux cartes.

Ce projet ma permis de mètre en pratique se que j'ai appris durant ces deux années de BTS. Il ma permis de me donné une idée sur mon niveau de compétence des les domaines de la communication, l'électronique et la programmation. Aujourd'hui, je suis capable de crée un système électronique répondant a un cahier des charge, et fabriquer un prototype a des fins de démonstration. Pour la suite, je ne conte pas travailler dans le ce domaine, je souhaiterai intégrer les force de l'armée de terre. Mais les compétences acquises durant ces années me seront surement utile dans l'avenir. Je remercie tout mes professeurs qui on usée de leur temps pour nous apprendre leur savoir faire, mais également tout les patrons et techniciens avec qui j'ai travaillé durant mes formations professionnel.

Annexes

Annexes 1

Code NewRemoteSwitch:

```
#include <NewRemoteReceiver.h>

void setup() {
  Serial.begin(9600);

  // initialise le récepteur en interrupteur 0 (= pin 2), appelle la classe "showcode"
  // aprés 2 code identique ( laisser appuyer quelque seconde sur la télécomande)

  NewRemoteReceiver::init(0, 2, showCode);
}

void loop() {
}

// La fonction  est appelée lorsqu'un code valide est reçu ..
void showCode(NewRemoteCode receivedCode) {

 // Note : les interruptions sont désactivées . Vous pouvez les réactiver si nécessaire

  // affiche le code reçu.

  //affiche l'adresse de l'émetteur
  Serial.print("Addr ");
  Serial.print(receivedCode.address);

  //affiche le numéro de groupe ou le numéro de l'interrupteur
  if (receivedCode.groupBit) {
    Serial.print(" group");
  }

  else {
    Serial.print(" unit ");
    Serial.print(receivedCode.unit);
  }
// affiche si la commande active /désactive ou sélectione le variateur
  switch (receivedCode.switchType) {
    case NewRemoteCode::off:
      Serial.print(" off");
      break;
    case NewRemoteCode::on:
      Serial.print(" on");
      break;
    case NewRemoteCode::dim:
      Serial.print(" dim");
      break;
  }
// si variateur, affiche son niveau
  if (receivedCode.dimLevelPresent) {
    Serial.print(", dim level: ");
    Serial.print(receivedCode.dimLevel);
  }
// affiche le temps de la période du signal en µs
  Serial.print(", period: ");
  Serial.print(receivedCode.period);
  Serial.println("us.");
}
```

code lightshow:

```
#include <NewRemoteTransmitter.h>

// Create a transmitter on address 123, using digital pin 11 to transmit,
// with a period duration of 260ms (default), repeating the transmitted
// code 2^3=8 times.
NewRemoteTransmitter transmitter(123, 11, 260, 3);

void setup() {
}

void loop() {
  // éteindre l'interupteur 3
  transmitter.sendUnit(2, false);

 // éteindre tout les interrupteur dans le groupe
  transmitter.sendGroup(false);

  // mettre le niveau du variateur de l'intérupteur 2 à 3 (niveau 3-15)
  transmitter.sendDim(1, 3);

  // attendre 5 secondes
  delay(5000);


  // allumer l'interrupteur3
  transmitter.sendUnit(2, true);

  // allumer tout les interrupteur dans le groupe
  transmitter.sendGroup(true);

 // mettre le niveau du variateur de l'intérupteur 2 à 1 (niveau 3-15)
  transmitter.sendDim(1, 15);

  // Wait 5 seconds
  delay(5000);
}
```

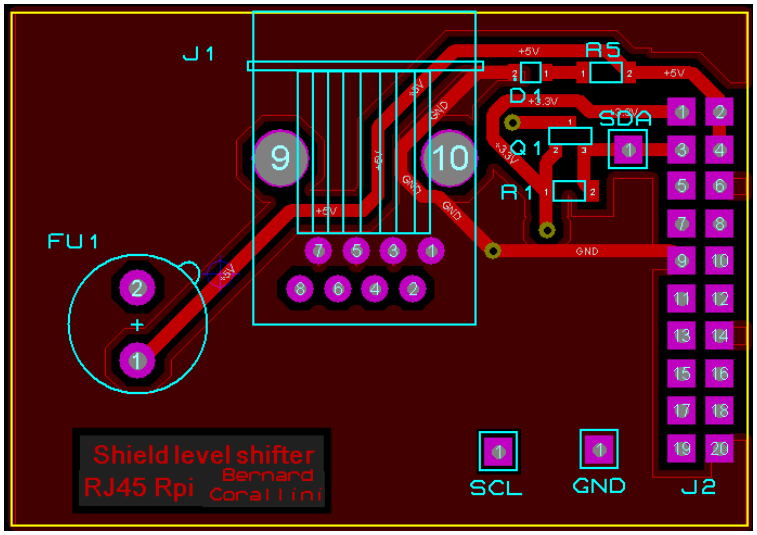
Annexes 2

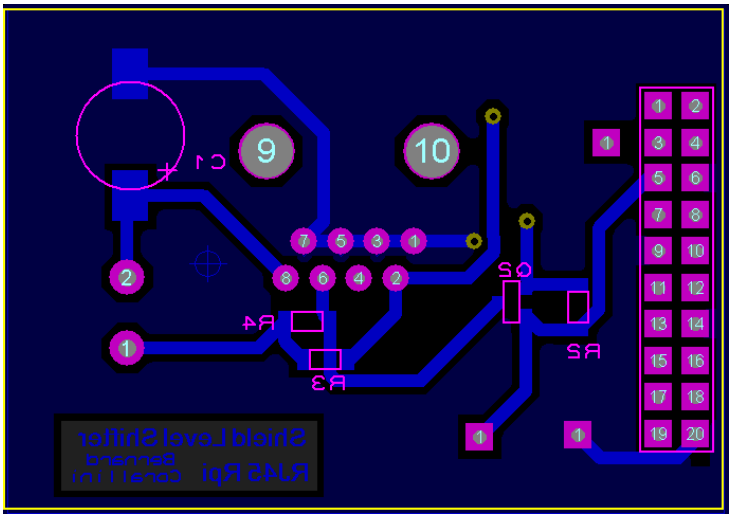
Tableau Ascii :

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

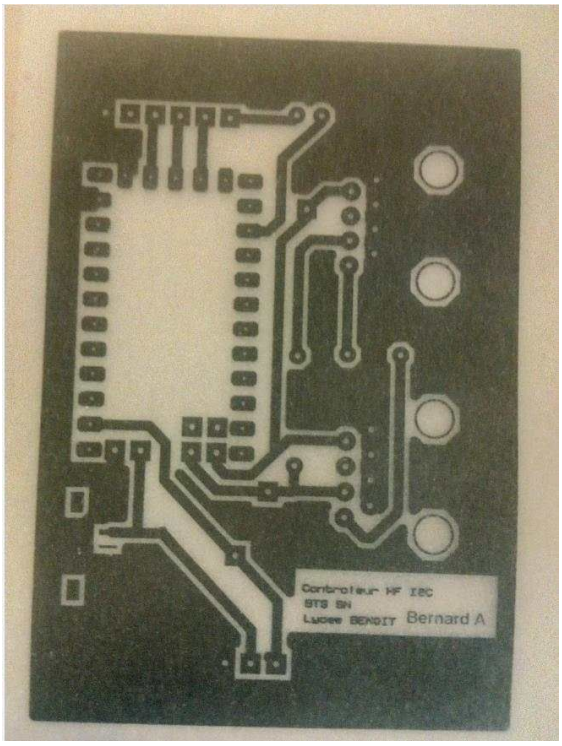
Routage Level shifter avec composant :





Annexe 3 :

Typon :



Annexe 4 :

Code Raspberry commenté :

mainwindow.cpp :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMainWindow>

#include<wiringPi.h>
#include <wiringPiI2C.h>




MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)

{
    int erreur;     // variable pour vérification connexion avec esclave
    ui->setupUi(this);
    wiringPiSetupGpio();
    fd=wiringPiI2CSetup(0x69);  // adresse de l'esclave
    erreur = wiringPiI2CRead(fd);



    // test de connexion avec l'esclave

        //    si variable erreur est inférieur à 0
        //affiche message d'erreur en rouge
   if(erreur < 0)
        {
        ui->label_Connection->setText("erreur connexion");
        ui->label_Connection->setStyleSheet("color: #FF0000;");
        }




        // si variable erreur est supérieur ou égal à 0,
        //affiche un message connexion réussie en vert
   else
        {
        ui->label_Connection->setText("connexion reussie");
        ui->label_Connection->setStyleSheet("color:#096A09;");
        }

}

MainWindow::~MainWindow()
```

```
{
    delete ui;
}

// un apuis sur un des boutons envois un caractère en hexadécimal à l'esclave
// commande servant a allumer ou éteindre un ou plusieur des chauffages



        //Allumage chauffage 1
    void MainWindow::on_pushButton_clicked()
    {
    wiringPiI2CWrite(fd, 0x61);
    }


        //Allumage chauffage 2
    void MainWindow::on_pushButton_3_clicked()
    {
   wiringPiI2CWrite(fd, 0x63);
    }


          //Allumage chauffage 3
    void MainWindow::on_pushButton_5_clicked()
    {
    wiringPiI2CWrite(fd, 0x65);
    }
```

```
        //inxtinction chauffage 1
void MainWindow::on_pushButton_2_clicked()
{
wiringPiI2CWrite(fd, 0x62);
}


        //inxtinction chauffage 2
void MainWindow::on_pushButton_4_clicked()
{
wiringPiI2CWrite(fd, 0x64);
}



        //inxtinction chauffage 3
void MainWindow::on_pushButton_6_clicked()
{
wiringPiI2CWrite(fd, 0x66);
}


        //Allumage chauffage 1,2 et 3

void MainWindow::on_Button_All_On_clicked()
{
wiringPiI2CWrite(fd, 0x67);
}


        //inxtinction chauffage 1,2 et 3
void MainWindow::on_Button_All_Off_clicked()
{
wiringPiI2CWrite(fd, 0x68);
}
```

```
            //inxtinction chauffage 1,2 et 3
    void MainWindow::on_Button_All_Off_clicked()
    {
    wiringPiI2CWrite(fd, 0x68);
    }



// test de connexion aprés lancement, par bouton test

void MainWindow::on_Test_Button_clicked()
{
    int erreur;

    erreur = wiringPiI2CRead(fd);
    if(erreur < 0)
    {
        ui->label_Connection->setText("erreur connexion");
        ui->label_Connection->setStyleSheet("color: #FF0000;");

    }
    else
    {
        ui->label_Connection->setText("connexion reussie");
        ui->label_Connection->setStyleSheet("color:#096A09;");
    }
}
```

Main.cpp :

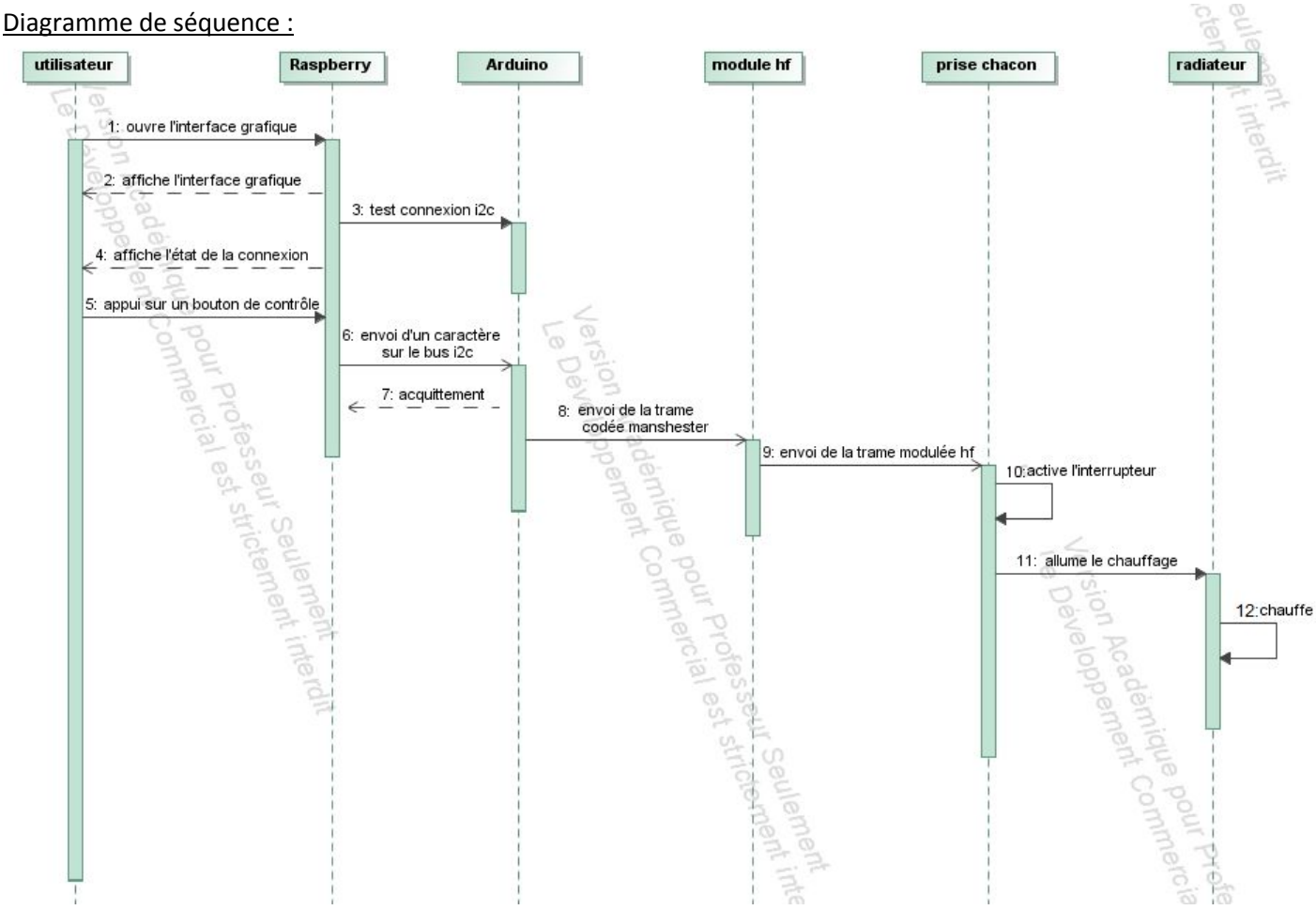
```
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

MainWindows.h :

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

    void on_pushButton_3_clicked();

    void on_pushButton_4_clicked();

    void on_pushButton_5_clicked();

    void on_pushButton_6_clicked();

    void on_Button_All_On_clicked();

    void on_Button_All_Off_clicked();


    void on_Test_Button_clicked();


private:
    Ui::MainWindow *ui;

    int fd;
};

#endif // MAINWINDOW_H
```

Annexe 5 :

Diagramme de séquence :



Annexe 6 :

Tableau prix :

LevelsShifter					
Composant	Fabricant @ Fournisseur	Qté	Conditionnement	Prix U HT	Prix T HT
fabrication carte imprimé	seeedstudio	1	10	0,990	0,990
résistance 10k	rs	4	50	0.021	0,084
résistance 1k	rs	1	50	0.008	0,008
conecteur 20 pin	gotronic	1	1	0,500	0,500
Connecteur Rj45	rs	1	5	1,916	1,916
condensateur	gotronic	1	1	0,250	0,250
Fusible	rs	1	2	0,900	0,900
LED cms	rs	1	50	0,050	0,050
Transistor	rs	2	25	0,130	0,260
					0,000
					0,000
			TOTAL HT GLOBAL		4,958

Page 1

carte module HF

Composant	Fabricant @ Fournisseur	Qté	Conditionnement	Prix U HT	Prix T HT
emetteur recepteur hf	cdiscount	1	1	2,070	2,070
Mini usb	rs	1	5	0,390	0,390
arduino mini	arduino	1	1	16,190	16,190
conecteur 20 pin	gotronic	1	1	0,500	0,500
Connecteur Rj45	rs	2	5	1,916	3,832
condensateur	gotronic	1	1	0,250	0,250
header femelle	gotronic	17	32	0,880	1,500
Fusible	rs	1	2	0,900	0,900
header male	gotronic	6	10	0,025	0,150
boitier plastique		1			4,000
			TOTAL HT GLOBAL		29,782

Page 1

suplément

Composant	Fabricant @ Fournisseur	Qté	Conditionnement	Prix U HT	Prix T HT
Raspberry pi 2	farnell	1	1	38,380	38,380
boitier carspberry	MICROCHIP	1	1	4,710	4,710
allimentation 5v	farnell	2	1	4,110	8,220
prise chacon (3)	getdio	1	1	31,900	31,900
					0,000
			TOTAL HT FEUILLE		83,210

total

Composant	Fabricant @ Fournisseur	Qté	Conditionnement	Prix U HT	Prix T HT
supplément		1			83,210
Levels Shifter		1			4,958
Carte module HF		1			29,782
			TOTAL HT FEUILLE		117,950