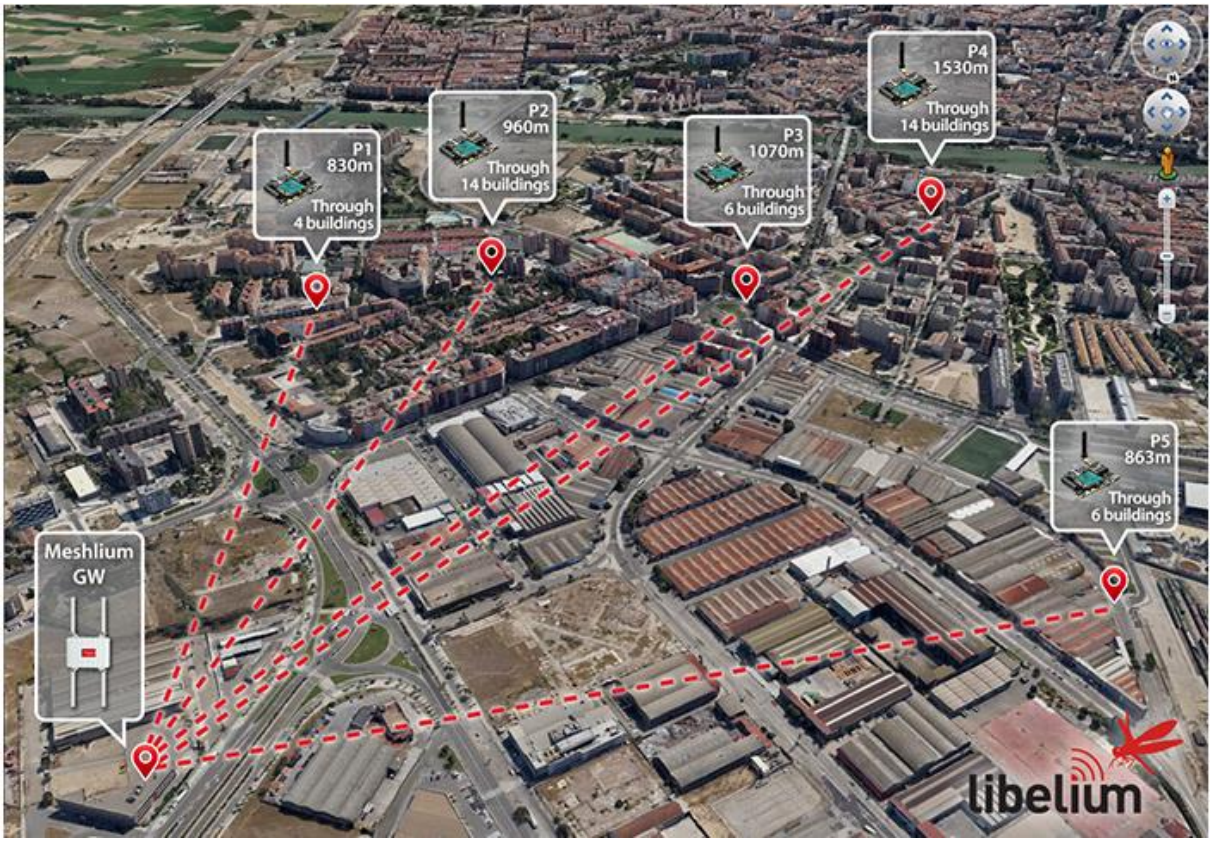
ENSIAME

PROJET 2A-2018



MAISON CONNECTEE PAR RESEAU LORA

ARCHITECTURE AVANCEE DES MICROCONTROLEURS

Réalisé par:

- ACHBAH Fatima Zahra -FOURNET Camille -HERVE Aideen

-KAMDEM Barbara -KANE Soda -OUARGANI Salma -XU pin

Tuteur Enseignant :

DELEBARRE Christophe

Sommaire

Table des figures

Introduction

L'Internet des objets (IoT), également appelé Internet of things, est un nouveau paradigme technologique envisagé comme un réseau mondial de machines et de dispositifs capables d'interagir les uns avec les autres. L'IoT est reconnu comme l'un des domaines les plus importants de la technologie du futur et attire de plus en plus l'attention d'un large éventail d'industries. L'internet se transforme progressivement en un Hyper Réseau qui connecte non seulement des milliers de personnes mais également des objets.

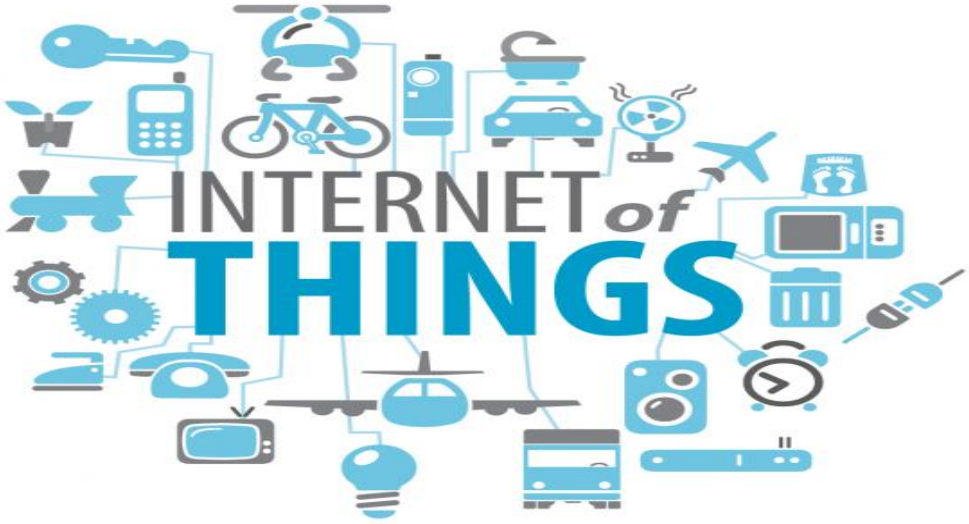
Nous entrons dans une période où il est impossible de considérer l'IoT comme une sorte de projet scientifique semi-intéressant. L'IoT est mis à profit dans les industries et les secteurs verticaux pour améliorer les services et les produits dans le monde entier. Les villes utilisent des capteurs, des compteurs intelligents et des analyses de données massives pour améliorer les services publics, la gestion de l'eau, la gestion des déchets, les services de transport et le stationnement. Les entreprises du secteur de la santé déploient des technologies vestimentaires et des dispositifs intégrés pour recueillir des données en temps réel sur leurs patients et fournir des soins vitaux en temps opportun. Les appareils dans nos maisons sont de plus en plus des appareils IoT qui peuvent ajuster leur fonctionnement en fonction des données recueillies sur nos habitudes et de l'environnement.

Nous proposons d'introduire de l'intelligence au plus près des composants électroniques afin de réaliser une maison intelligente. Cette maison dispose des systèmes automatiques, tel que le contrôle de l'ouverture et la fermeture des différentes issues (portes, fenêtres...), la connaissance de la température en temps réel, l'éclairage…Via une application Android.

Dans ce contexte, notre projet repose sur l'implémentation d'un réseau Lora dédié aux objets connectés. Dans ce rapport, nous introduisons tout d'abord l'importance de "l'IOT" et le contexte du réseau Lora. Nous proposons ensuite une revue détaillée sur les différents objets choisis pour notre réseau. En particulier, nous montrerons les grands pas pour la réalisation d'un tel projet : Choix de composants, routage électrique et la programmation Software.

I) Partie Théorique

1. IOT et Réseau LoRa



L'Internet des objets, ou Ido (en anglais Internet of Things, ou IoT), est l'extension d'Internet à des choses et à des lieux du monde physique.

Alors qu'Internet ne se prolonge habituellement pas au-delà du monde électronique, l'Internet des objets connectés représente les échanges d'informations et de données provenant de dispositifs du monde réel avec le réseau Internet.

Considéré comme la troisième évolution de l'Internet, baptisé Web 3.0 (parfois perçu comme la généralisation du Web des objets mais aussi comme celle du Web sémantique) qui fait suite à l'ère du Web social, l'Internet des objets revêt un caractère universel pour désigner des objets connectés aux usages variés, dans le domaine de la e-santé, de la domotique ou du quantified self.

L'Internet des objets est en partie responsable d'un accroissement exponentiel du volume de données généré sur le réseau, à l'origine du big data (ou mégadonnées en français).

Le réseau LoRa n'est pas comme les autres réseaux mobiles classiques(la 3G ,la 4G ou la 5G).En effet, Il ne peut pas transporter de grandes quantités d'informations, LoRa n'est pas taillé pour satisfaire les besoins d'appareils beaux parleurs, comme les smartphones. Cette technologie de modulation de fréquence ne peut faire circuler que de petits paquets de données, émis par des capteurs de température ou d'humidité par exemple, fixés sur des objets connectés. Elle pourra faire transiter entre 0,3 et 50 kilobits par seconde (le débit du réseau s'adapte à chaque objet pour ne pas grignoter trop de bande passante).Lora WAN est donc un nouveau protocole destiné aux problématiques des objets connectés.

2. Les grand Axe du projet

2.1. La connectivité entre les objets

Le projet consiste à faire une interconnexion entre des objets connectés et un serveur qui serait dans notre cas l'ordinateur de l'utilisateur. La liste des objets utilisés représente les outils qu'on utilise dans le quotidien.

Les objets connectés seront dans notre cas : lampe, ventilateur, capteur luminosité, capteur température, Volet roulant et ouverture/fermeture du portail.

La connectivité se fera de la manière suivante :

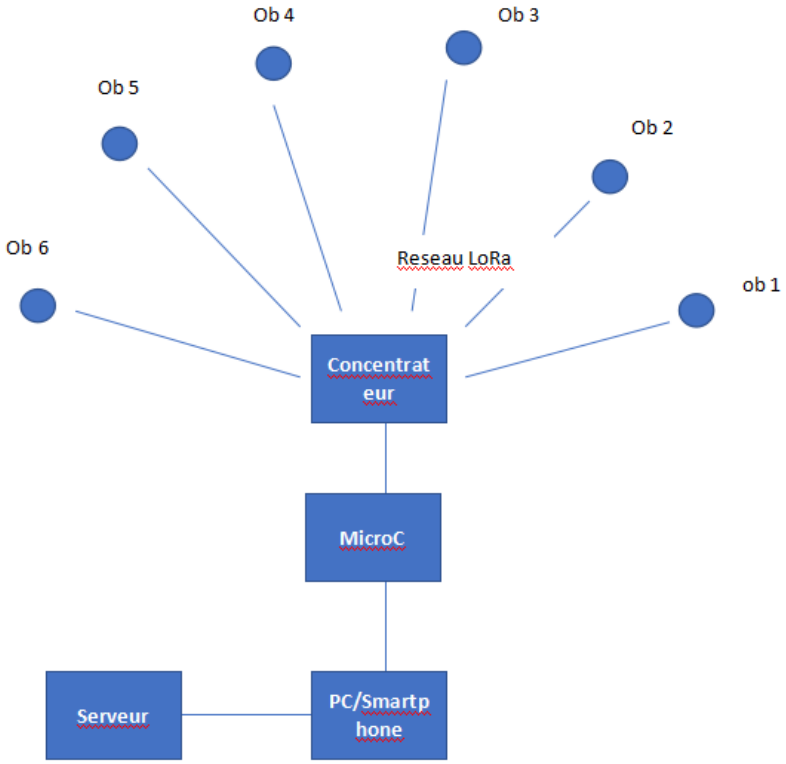


Figure 1: Schéma explicatif

Pour réaliser le réseau LoRa, il nous faut :

- Un serveur
- Un PC/Smartphone
- MicroC
- Concentrateur
- Les Objets connectés

Le serveur permet de traiter des données et connecter plusieurs ordinateurs, mais cela ne sera pas l'objet de l'étude.

Un PC/Smartphone, dans notre cas, nous avons développé une application Android, qui permet à l'utilisateur de contrôler et surveiller les objets connectés à distance.

Un Microcontrôleur centrale associé avec un module LoRa qui permettra de recevoir et transmettre les ordres aux objets connectés. Il sera lié aux objets connectés par le réseau LoRa et au smartphone par la liaison **Bluetooth**.

Un concentrateur, qui fonctionnement comme un multiplexeur, mais dans notre projet, cette partie peut se faire de manière hardware par l'intermédiaire d'un Gateway ou software en travaillant sur plusieurs fréquences différentes.

Les Objets connectés, relié à un microcontrôleur et un module LoRa pour recevoir les ordres ou transmettre les données par le microcontrôleur.

Ainsi chaque **objet + le microcontrôleur central** pourrait transmettre des infos ou en recevoir selon le fonctionnement souhaité. À l'image du protocole de bus I2C, il y a un seul réseau LoRa ou plusieurs objets peuvent prendre 'parole', il est donc important de définir un protocole de communication connue de tous les objets afin d'éviter les interférences.

2.2. Protocole de communication

Pour définir le protocole de communication, au vu de la grande ressemblance avec le bus I2C, nous allons nous inspirer des différentes méthodes pour réaliser un protocole de la manière suivante.

Étant un récepteur ou Emetteur, leur comportement est différent. L'analogie qu'on pourrait faire avec le bus **I2C** est **Emetteur -> Master, Récepteur -> Slave.**

Le schéma de communication se présente de la manière suivante.

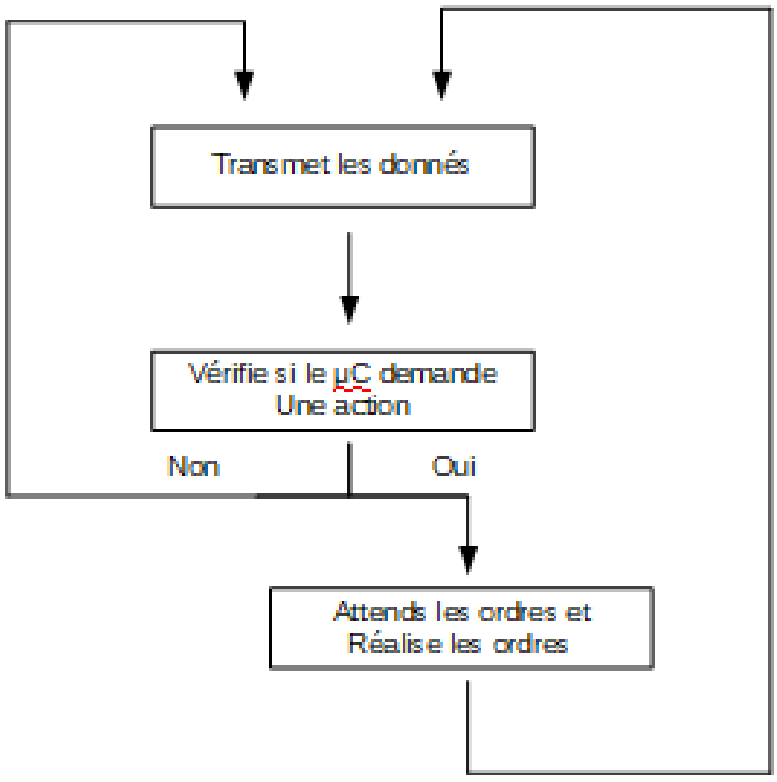


Figure 2: Schéma de fonctionnement de l'objet connecté

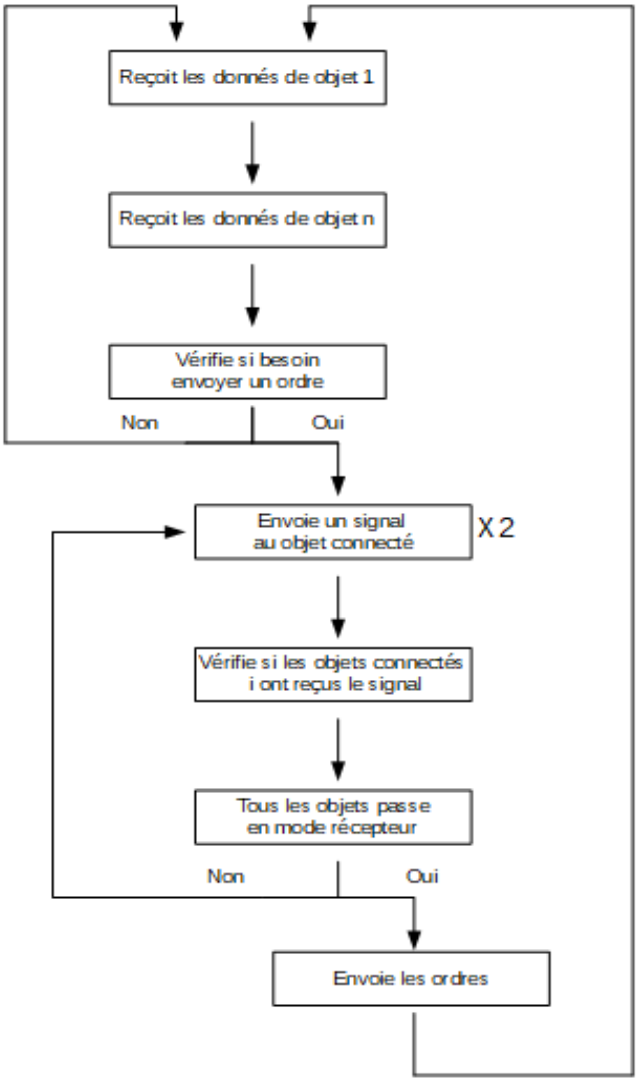


Figure 3:Schémas de fonctionnement microcontrôleur centrale

II) Réalisation Concrète

1. Choix des objets connectés

La maison connectée est une maison qui dispose de systèmes automatiques, tel que le contrôle de l'ouverture et la fermeture des fenêtres, la connaissance de la température, l'éclairage et la communication. Dans de cadre on avait choisi plusieurs capteurs pour concevoir des objets connectées afin d'avoir une maison intelligentes. Nous avons choisi des interrupteurs pour : Allumer/éteindre les lampes. Allumer/éteindre le Ventilo. Utiliser un capteur de température. Un capteur d'humidité et un capteur de luminosité.

i. Lampe intelligente :

On réalise une lampe intelligente en utilisant un interrupteur commandé par le microprocesseur. L'utilisateur envoie l'information par son application Android. La carte de commande centrale reçoit l'information et l'envoie par le réseau LoRa à la petite carte au niveau du l'interrupteur.

Pour se faire, on aura besoin d'une carte Arduino ou MicroC ATMEGA avec des modules qu'on doit souder. On aura besoin aussi d'un relais qui sert à adapter la tension de la maison et qui alimente la lampe (220V) à notre carte Arduino.

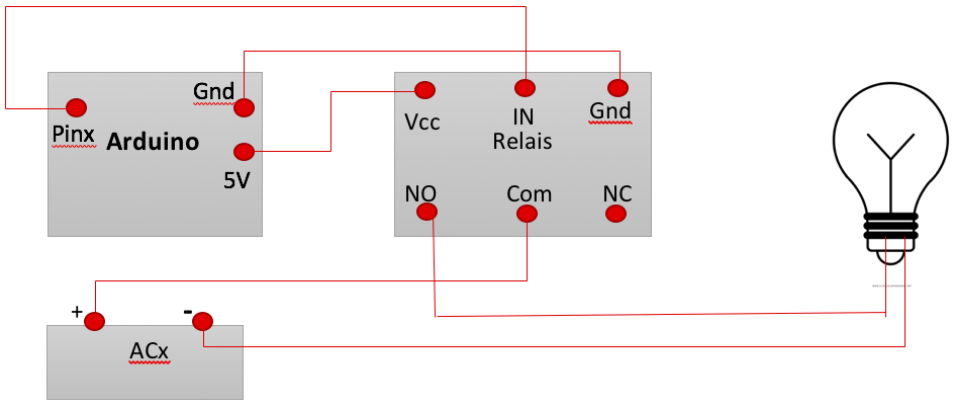


Figure 4: Schéma de lampe intelligente

ii. Radiateur intelligent :

De même pour un radiateur on utilise un interrupteur commandé par le microprocesseur. L'utilisateur envoie l'information par son application Android. La carte de commande centrale reçoit l'information et l'envoie par le réseau LoRa à la petite carte au niveau du l'interrupteur.

Pour se faire, on aura besoin d'une carte Arduino ou MicroC ATMEGA avec des modules qu'on doit souder. On aura besoin aussi d'un relais qui sert à adapter la tension de la maison et qui alimente la lampe (220V) à notre carte Arduino.

iii. Capteur de température et d'humidité (Applications) :

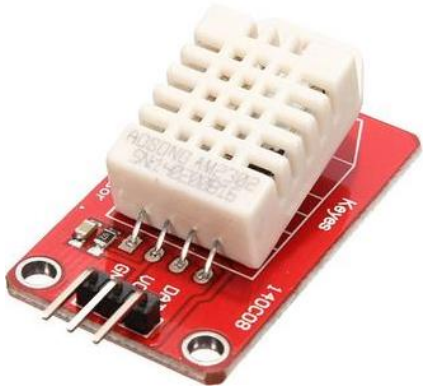


Figure 5: Capteur DHT22 d'humidité et de température

Le capteur de température et d'humidité DHT22 (ou RHT03) communique avec un microcontrôleur via un port série. Le capteur est calibré et ne nécessite pas de composants supplémentaires pour pouvoir être utilisé. On utilisera ce capteur pour communiquer la température et l'humidité de la salle avec l'utilisateur. Les données de température et d'humidité seront traitées numériquement et envoyer par le module LoRa qui serait branché avec notre microprocesseur.

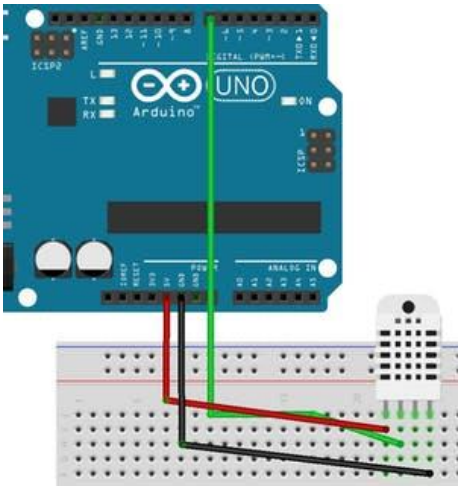


Figure 6: Liaison capteur DHT22 Arduino

Alimentation: 3,3 à 6 Vcc
Consommation maxi: 1,5 mA
Consommation au repos: 50 µA
Plage de mesure:
- température: -40 à +80 °C
- humidité: 0 à 100 % RH
Précision:
- température: ± 0,5 °C
- humidité: ± 2 % RH

iv. Capteur Luminosité

Capteur de luminosité TSL2561 permettant de mesurer une luminosité de 0,1 à 40000 Lux. Il communique avec un microcontrôleur type Arduino via le bus I2C.
Son capteur précis couvre la lumière visible et les infrarouges grâce à plusieurs diodes. Cela signifie que vous pouvez mesurer séparément la lumière infrarouge, la lumière visible ou le spectre complet.
Il est possible d'utiliser ce circuit sur des plaques de connexions sans soudure en soudant le connecteur inclus.

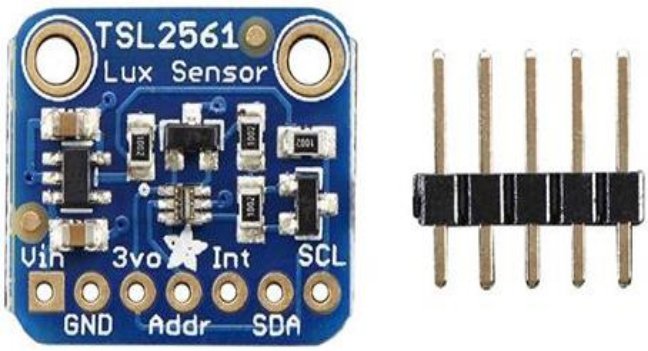


Figure 7: Capteur de luminosité TSL2561

Alimentation: 2,7 à 3,6 Vcc
Interface I2C
Plage de mesure: 0,1 à 40000 Lux
T° de service: -30 à +80 °C

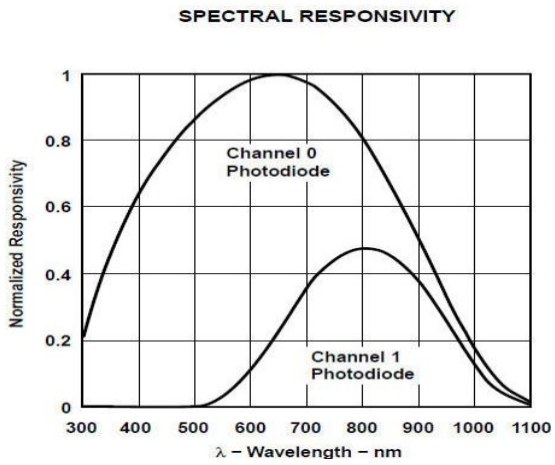


Figure 8: Réponse capteur de luminosité TSL2561

On aura besoin d'un microcontrôleur afin de traiter les données captées par le capteur de luminosité et d'un module LoRa afin de communiquer ces informations traitées avec le microcontrôleur centrale.

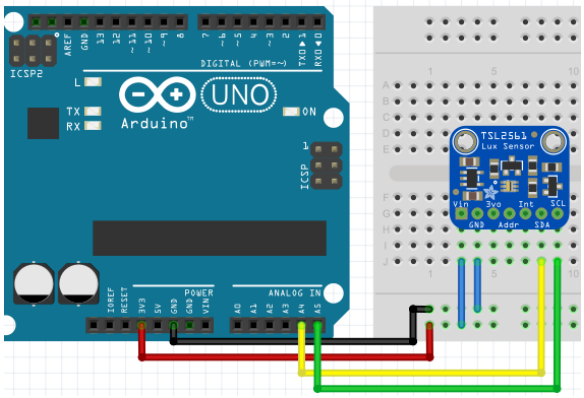


Figure 9: Liaison Capteur Arduino

v. Portail intelligent

Portail. On utilise un détecteur d'ouverture et fermeture du portail. Pour ça on utilise un capteur qui permettent de détecter la pression physique, la torsion et le poids. Ils sont simple à utiliser et financièrement abordables.

Figure 10: Capteur de pression flex FSR

Ce capteur nous indique si le portail est ouvert ou fermé. Il y aura une communication entre ce capteur et l'Arduino après on communique l'information en utilisant le module LoRa.

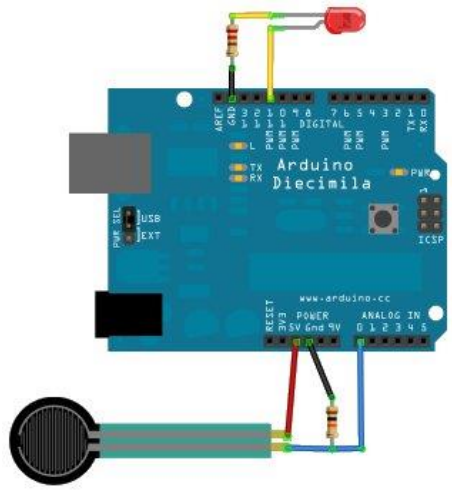


Figure 11: Communication Arduino capteur de pression

2. Le module LoRa

Nous allons dans cette partie présenté le module. Comme expliqué plus haut, chaque objet connecté est associé à un microcontrôleur qui sert de bouche et d'oreille à l'objet connecté.

À proprement parler le module LoRa utilisé est une puce électronique intégré, fabriqué par la société Semtech portant la référence SX1272/73. Le marché des modules LoRa est quasi entièrement dominé par cette société, il existe pratiquement peu autre entreprise fabriquant la puce

Cependant il n'est pas conseillé utilise le module Lora directement, c'est pourquoi nous avons décidé utilise de prendre un package de composant préétablit qui intègre tous les composant électronique pour un fonctionnement optimal de la puce

Il s'agit d'un composant Transceiver qui est la contraction de Transmitter + Receiver. Elle permet la transmission et la réception d'information à travers le module Lora.

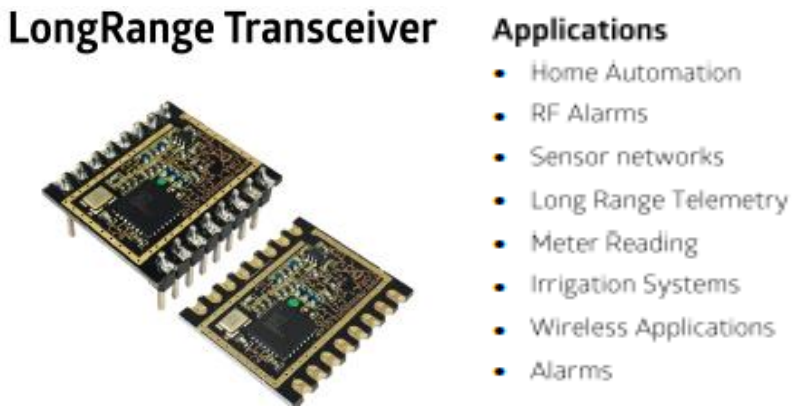


Figure 12: Module LoRa

3. Arduino Vs Atmega

Afin de développer des applications avec le module LoRa, donc le LoRa transceiver, il nous faut un environnement de développement, un IDE.

Avant toute chose, il faut déjà se décider sur un langage de programmation. Les langage phare de la programmation sont le C/C++ et Assembleur, dont nous allons présenter les avantages et inconvénients

	C/C++	**Assembleur**
Avantage	* Langage haut niveau, plus proche d'utilisateur * Beaucoup de ressource en Ligne disponible sur internet	* Très proche du langage machine * Inclus la gestion de mémoire permet la détection des erreurs de manière très précise jusqu'au niveau physique du composant
Inconvénient	* Est assez loin du langage machine * Non optimisé pour un microcontrôleur	* Difficile d'accès pour programmeur non confirmé

Figure 13: Tableau de comparaison entre C/C++ et Assembleur

D'après ce tableau (figure 13) il en ressort que dans notre cas, le développement d'une application relativement complexe, le langage C/C++ serait un meilleur choix stratégique.

Apres avoir choisi le langage de programmation, qui donc le C/C++ , il faut choisir un IDE qui permet la conversion du code écrit en C/C++ en langage machine. 2 IDE, Arduino, et Atmel Studio, ressortent du lot principalement. Nous allons en faire une comparaison aussi.

	Arduino	**Atmel Studio**
Avantage	* Beaucoup de bibliothèque préétablie * Facile de prise en main	* Permet d'accéder la mémoire du microcontrôleur * Une interface de l'IDE de programmation proche de l'assembleur * compatible avec les codes sur le logicielle Arduino
Inconvénient	* N'a pas accès en mémoire interne du microcontrôleur.	* Moins évidant à prendre en main. * Beaucoup d'outil de programmation payante.

Figure 14: Tableau de comparaison entre Arduino et Atmel Studio

Il devient évidant que programmer avec Atmega est la meilleure solution. Cependant plusieurs points sont à faire attention.

***Comme nous n'avons pas acheté outils de programmation de Atmel Studio, il faudra configurer nous même les paramètres du microcontrôleur. Nous avons besoin d'un programmeur qui permet de faire la liaison entre le microcontrôleur et le PC. Plusieurs solution existe, la solution adoptée est la technologie USBaps en utilisant les ports de la méthode ICSP.

Nous nous sommes arrêtées sur cette solution car nous ne savons pas comment exploiter correctement, le port universelle USB présent sur les plaques Arduino. De plus sur les cartes de prototypage final, les ports de ICSP seraient un bien meilleur gain de place.

Figure 15: Un programmeur USBaps

4. Application Android

i. Principe

L'application permet d'allumer ou d'éteindre le radiateur et la lampe et d'afficher leur état (allumé ou éteint), de l'état de la porte (ouverte ou fermée), ainsi que les valeurs de la luminosité (en Lux), de la température (en °C) et de l'humidité (en %RH) grâce à une connexion Bluetooth.

Afin de réaliser cette application Android, nous avons également écrit un code Arduino qui envoie les informations à l'application grâce à un module Bluetooth. La communication de l'Arduino se faisant par l'envoi et la réception d'un caractère, nous avons adapté la valeur des variables reçues et envoyées par l'application.

Chaque objet connecté est distingué grâce à une adresse : lampe = 1, radiateur=2, …

Afin de garantir la réception de l'adresse et des données de l'Arduino et l'application Android nous avons introduit des variables de contrôle représentant un caractère, ayant pour valeurs décimales 36, 37,38 et 39 et ne correspondent pas à des valeurs numériques (ne pouvant donc pas être transmises par l'Arduino).

Au premier démarrage de l'application, l'interface visible par l'utilisateur est la suivante :

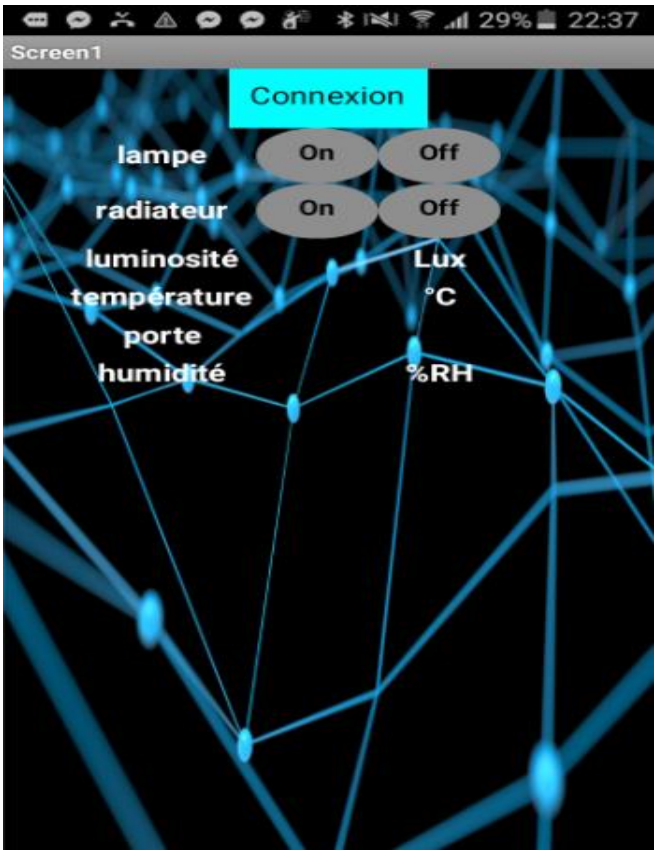
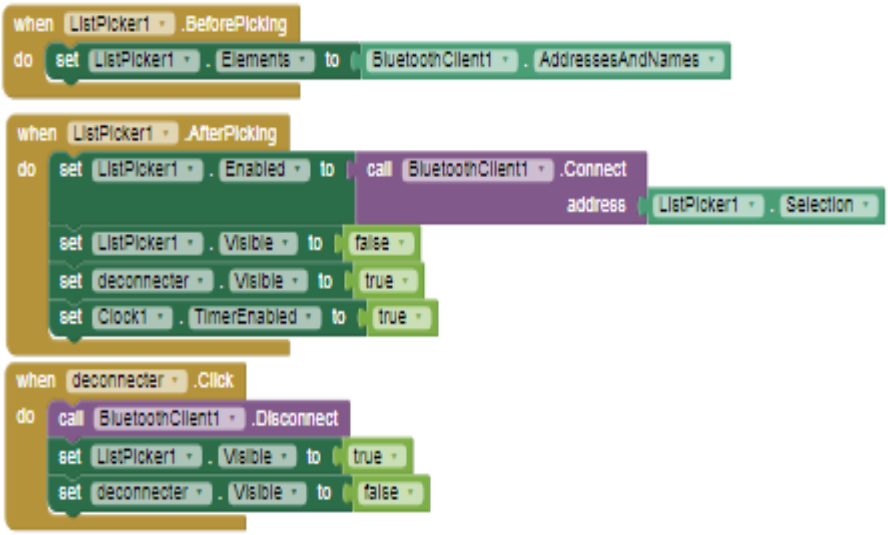
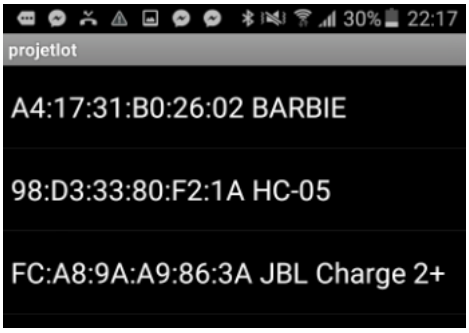


Figure 16: Interface application Android

ii. L'application Android





Avant toute connexion et action (BeforePicking), l'ensemble des appareils Bluetooth pouvant être connectés est rassemblé dans une liste (ListPicker1) et le bouton connexion est visible. L'utilisateur peut choisir un de ces appareils :

Une fois sélectionné (AfterPicking), le smartphone est connecté au Bluetooth sélectionné via l'application et le timer est mis en route.

Bouton ON/OFF (exemple de la lampe) :

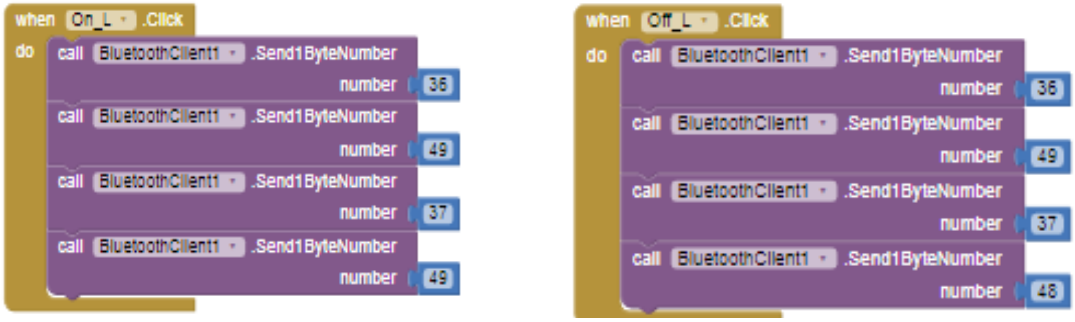


Figure 17: Bouton ON/OFF

L'appui sur chaque bouton (Click) envoi quatre bytes successivement. Un premier byte correspondant au code ASCII du dollar qui servira à annoncer le deuxième byte envoyé à savoir l'adresse et permettra de vérifier que c'est bien l'adresse qui est reçue. Dans la même logique, nous envoyons le code ASCII du point d'interrogation pour assurer que le byte suivant est bien la data.

<u>Apparence bouton ON/OFF et état (exemple de la lampe) :</u>

Si la valeur de la data est de 0, cela signifie que le bouton OFF a été appuyé. Dans ce cas, le bouton OFF est grisé et l'état éteint de l'élément (ici la lampe) est indiqué à l'utilisateur par un texte.

A l'inverse, si nous recevons un 1, cela signifie que la lampe a été allumée et le bouton ON est alors grisé.

Dans le cas où nous ne recevrons rien ou une valeur autre que celles définies précédemment, un texte d'erreur sera affiché afin de signifier à l'utilisateur le problème.

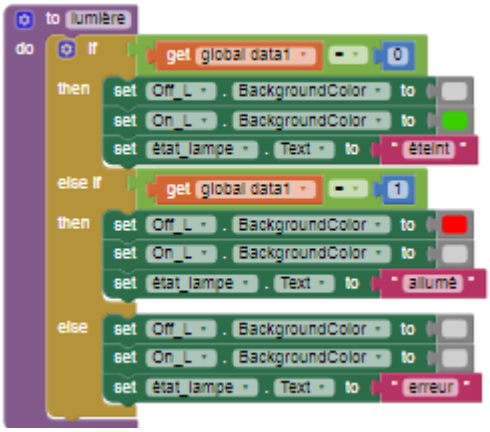


Figure 18: Apparence bouton ON/OFF et état (exemple de la lampe)

Affichage valeur (exemple de la luminosité) :

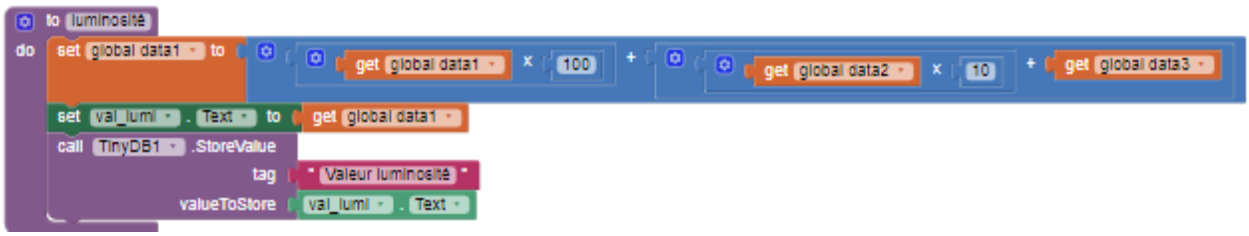


Figure 19: Affichage valeur (Luminosité)

Pour chaque valeur à afficher, trois caractères sont envoyées : le caractère des centaines, dizaines et unités. La valeur est reconstituée dans une variable gobale (« data1 ») puis affichée dans la zone de texte « val_lumi ». Cette valeur est également enregistrée dans une base de données (TinyDB1) sous le nom « Valeur luminosité ».

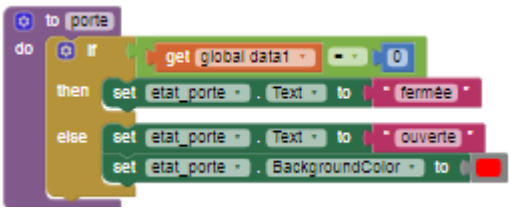


Figure 20: Affichage état de la porte

Affichage de l'état de la porte :

Dans le cas de la porte, si la valeur de la data est de 0, la porte est fermée et cet état est signifié à l'utilisateur par le texte correspondant. Toute autre valeur sera considérée comme une erreur, et la porte sera affichée comme ouverte par souci de sécurité.

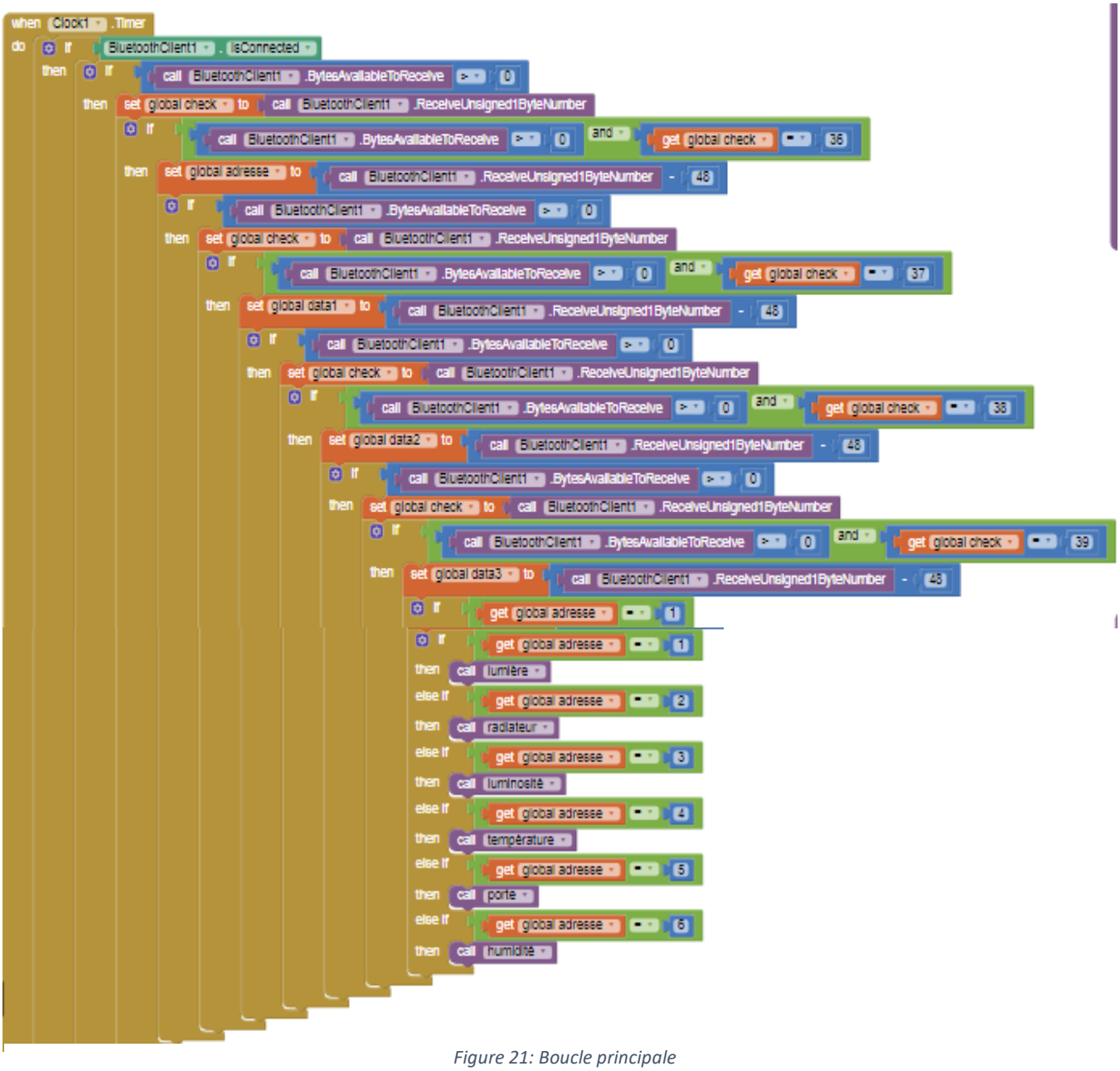


Figure 21: Boucle principale

Dans la boucle principale, on vérifie tout d'abord que le bluetooth est toujours connecté. Puis nous vérifions que l'application peut recevoir des bits avant d'affecter le byte reçu à la variable « check » qui sera comparée à la valeurs de contrôle de l'adresse. Si les valeurs correspondent, la variable « adresse » reçoit la valeur du prochain byte reçu -48. Le même protocole est répété pour les trois caractères correspondants aux données. Le fait de soustraire 48 aux bytes reçus permet d'obtenir le caractère de la valeur correspondant et donc de reconstituer les valeurs à afficher.

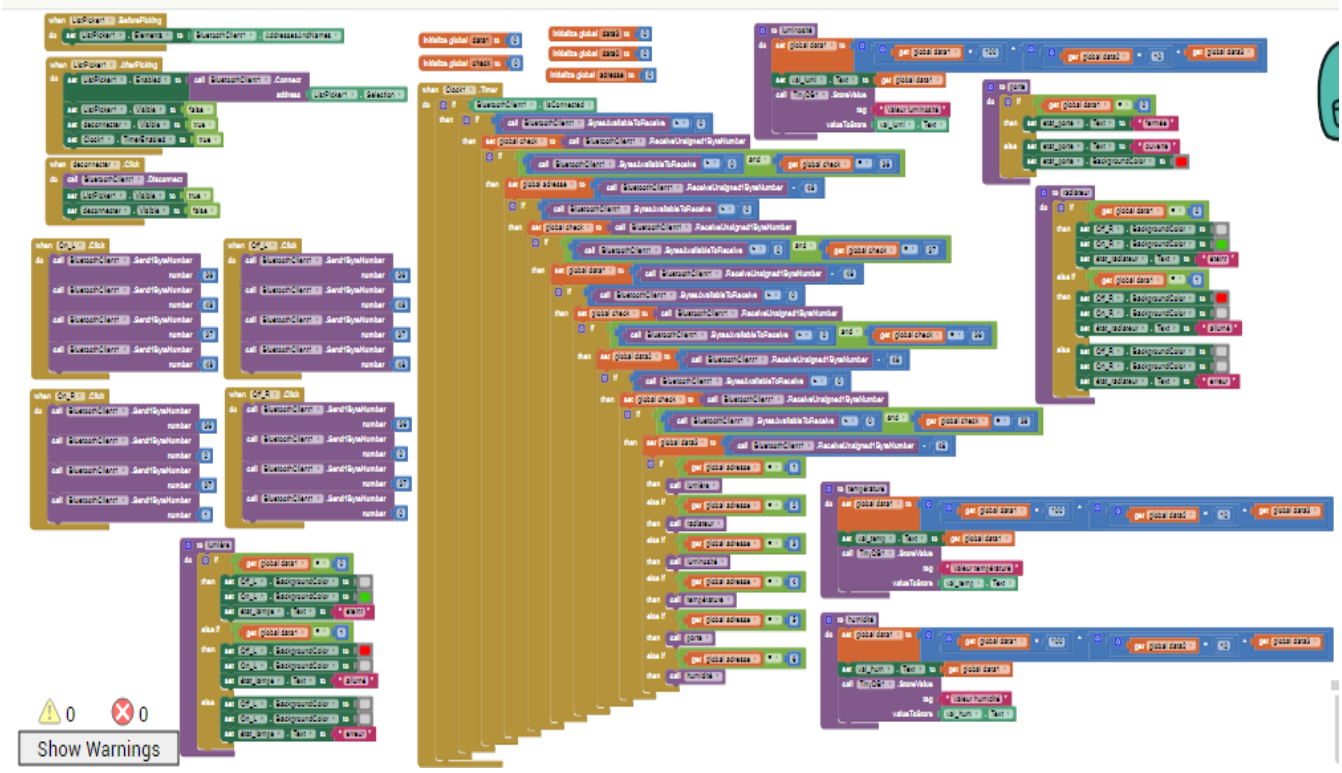


Figure 22: L'application finale

iii. Code Arduino équivalent

```
#include <SoftwareSerial.h>
SoftwareSerial COMserie_4(10,11); // RX, TX
int start_address = 36; //Variable de contrôle de l'adresse: si start_address reçu, le byte suivant correspond à l'adresse de l'objet connecté
int start_data1 = 37; //Variable de contrôle du premier caractère de la data (centaine de la valeur): si start_data reçu, le byte suivant reçu correspond à la valeurs correspondant à cet objet
int start_data2 = 38;//Variable de contrôle du premier caractère de la data (dizaine de la valeur):
int start_data3 = 39;//Variable de contrôle du premier caractère de la data (unité de la valeur):
void setup() {
  Serial.begin(9600);
  while (!Serial) {
    ; // Attente de la connexion USB }
  COMserie_4.begin(9600);//Connexion avec le module bluetooth
```

```
}
void loop() { // run over and over
  if (COMserie_4.available()) {;si la connexion bluetooth est établie
    Serial.write(int(COMserie_4.read()));//Envoi sur le port série des variable reçues depuis l'application android
  }
  if (Serial.available()) {
    envoi(start_address);//Envoi de la variable de contrôle de l'adresse
    delay(100); //délai de 100ms pour assurer la réception et le traitement par l'application android
    envoi(int(Serial.read()));//Envoi de l'adresse de l'objet connecté pour lequel la valeur physique correspondant change
    delay(100);
    envoi (start_data1); // Envoi de la première variable de contrôle des données (pour la centaine)
    delay(100);
    envoi(int(Serial.read()));//envoi du premier caractère reçu depuis le port série (correspondant à la centaine) à l'application );
    delay(100);

    envoi (start_data2);// Envoi de la deuxième variable de contrôle des données (pour l'unité)
    delay(100);//envoi du premier caractère reçu depuis le port série (dizaine) à l'application
    envoi(int(Serial.read());
    delay(100);

    envoi (start_data3);// Envoi de la troisième variable de contrôle des données (pour l'unité)
    delay(100);
    envoi(int(Serial.read());//envoi du premier caractère reçu depuis le port série (unité) à l'application
    delay(1000);//
  }
  Serial.flush();
}
void envoi(int val){
Serial.write(val); //Permet le contrôle des variables envoyées à l'application en les affichant sur le port série
   COMserie_4.write((val)); //Envoie les valeurs vers l'application
```

5. Schéma électrique et routage

On retrouve dans notre schéma électronique différent composant nécessaire au bon fonctionnement du module LORA. Notre schéma électronique est découpé en 5 zones distinctes :

- Partie alimentation
- Partie USB
- Partie Microcontrôleur
- Partie liaison Bluetooth
- Partie Lora

Pour fonctionner la carte a besoin d'alimentation. La carte fonctionnant sous une tension de 5V, il a été convenu qu'elle sera alimentée via un port USB.

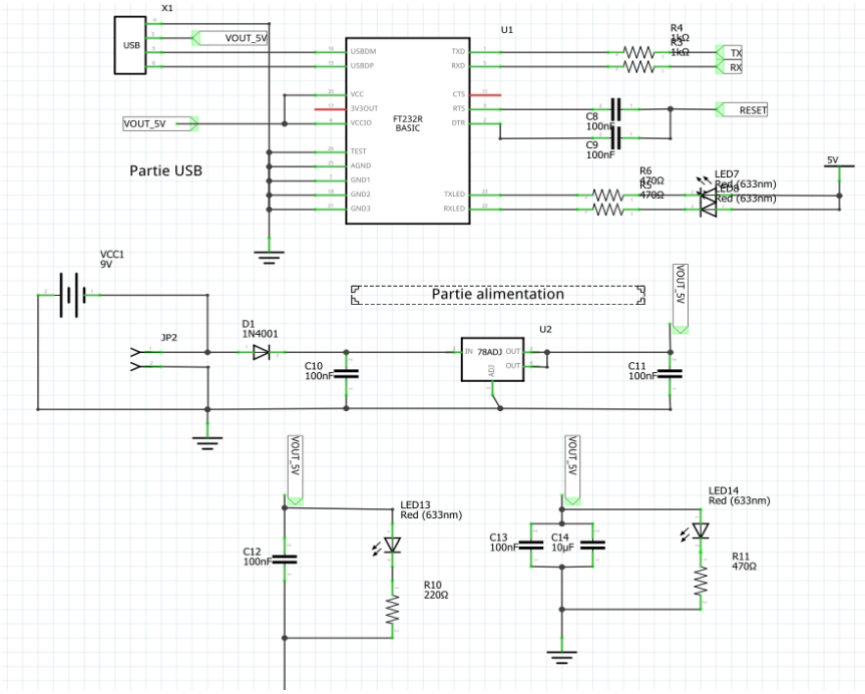


Figure 23: Schéma d'alimentation

Les périphériques reliés au microcontrôleur communiqueront avec ce dernier via la liaison SPI. Le bus SPI est un bus série synchrone constitué de 4 signaux logiques. SCK,MOSI,SS générés par le maître et MISO généré par l'esclave. Le maître s'occupe de la communication. Plusieurs esclaves peuvent communiquer sur le même bus à condition qu'il y ait une ligne de sélection dédiée à chaque esclave.

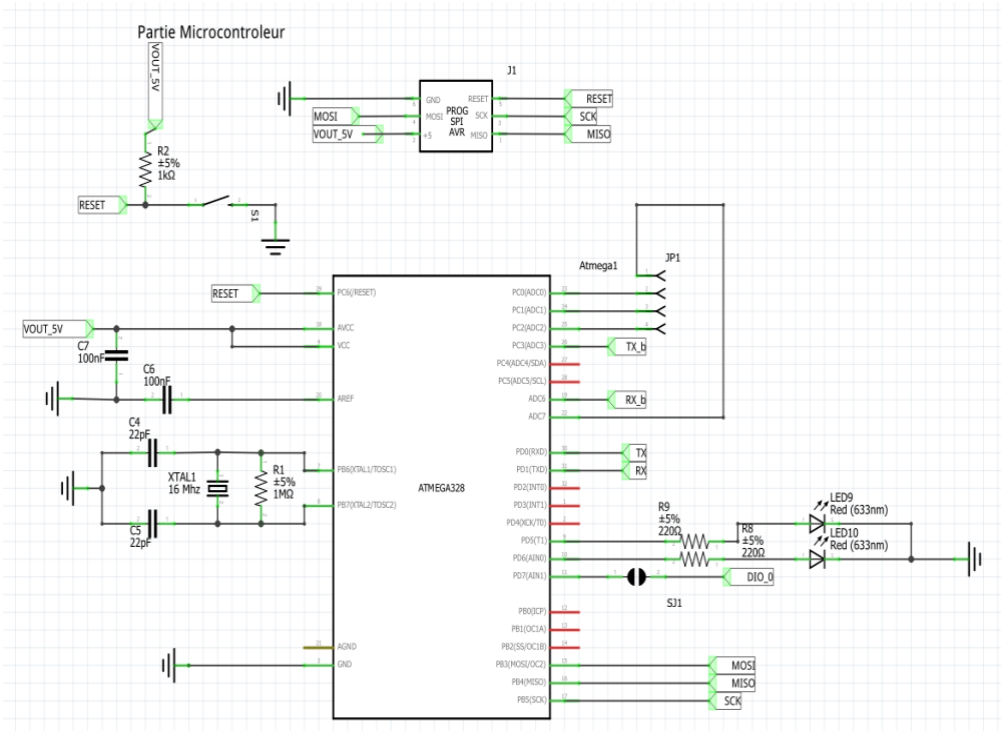


Figure 24: Schéma avec le module LoRa

La partir LoRa est gérée par un émetteur-recepteur SX1276 doté d'un modem LoRa de chez Semtech offrant une communication longue portée. Cette communication repose sur le principe de la modulation à étalement de spectre.

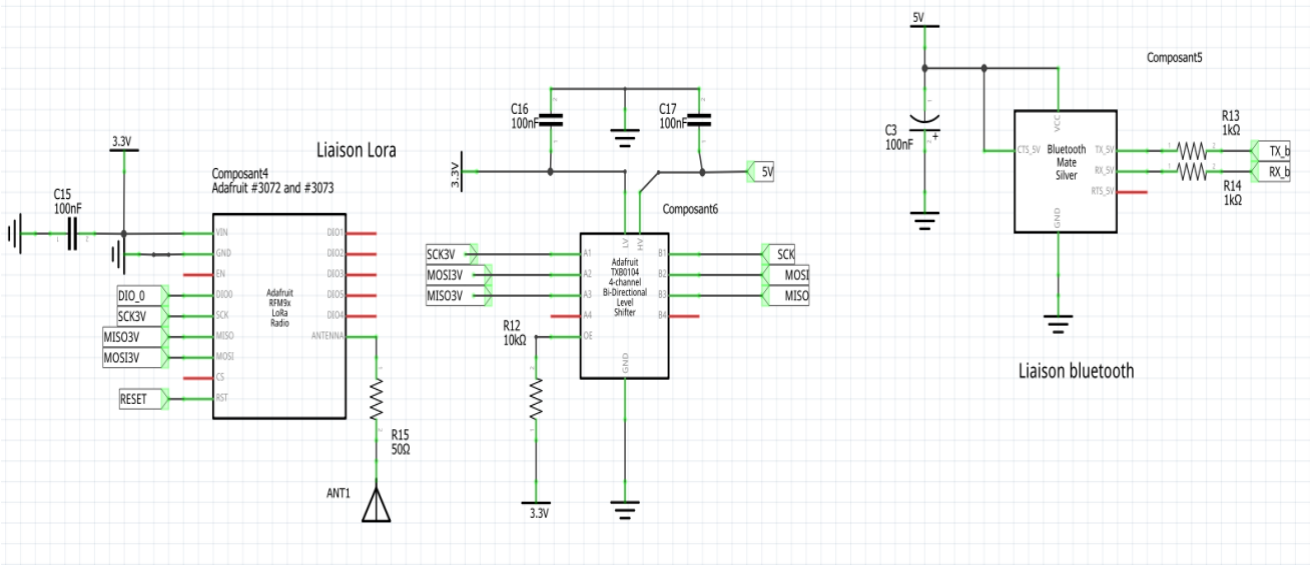


Figure 25: Le schéma électrique de différents modules

Vous trouverez en annexe les datasheets des différents composants utilisés.

Une fois le schéma électronique défini, pour permettre la création de la carte, il faut router les différents composants entre eux.

Le routage consiste à déterminer la route des signaux d'alimentation qui vont relier les composants entre eux et à placer les composants sur la carte. La carte a été conçue en respectant quelques contraintes de conception dimensionnelles :

- L’angle entre les pistes ne doit pas être droit

- La carte est en double face

Pour faciliter l’auto routage, nous avons réduit la taille de quelques connections. Ensuite nous avons procéder à un routage manuellement pour router les câbles manquants.

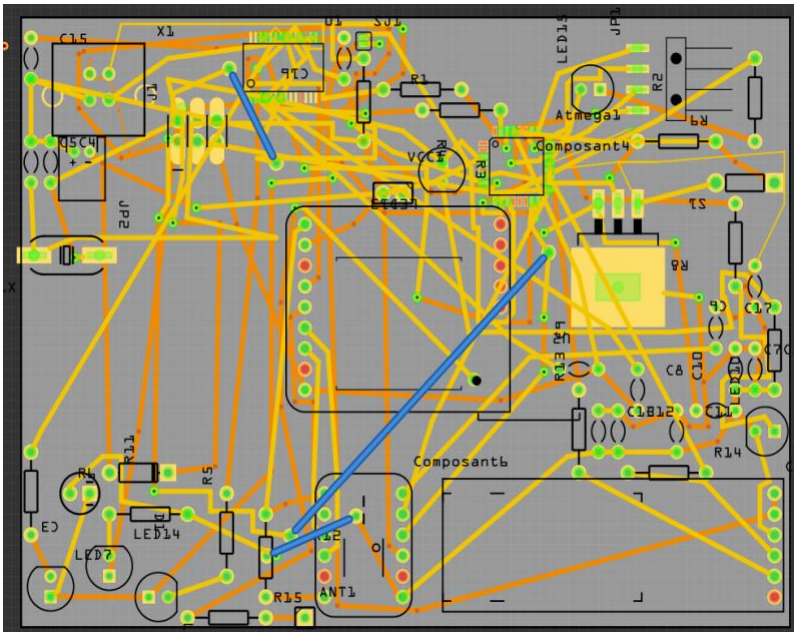


Figure 26: Le schéma électrique PBCA

III) Code Arduino pour la carte central LoRa

1. Code annexe pour le microcontrôleur central

```
#include <SPI.h>
#include <LoRa.h>
const byte interruptPin = 2;  // définition du pin d'interruption
byte activity;
int messageObjet1;
int messageObjet2;
int messageObjet3;
int messageObjet4;
int messageObjet5;
int messageObjet6;
String message1;
String message2;
String message3;
String message4;
String message5;
String message6 ;
void setup() {
```

```
LoRa.setSPIFrequency(9600); // Regle la frequence SPI
LoRa.setPins(3,4,2);       //  Définition de pin de connection du module
LoRa.begin(915E6);         // Initialise le module Lora avec la frequence de 915e6 hzt
pinMode(interruptPin, INPUT_PULLUP);
attachInterrupt(digitalPinToInterrupt(interruptPin), bluetooth, FALLING);
}
void loop() {
   messageObjet1= LoRa.parsePacket(); //Recoit un message surla fréquence donné
   if (messageObjet1) {
// le message a été bien reçus , entier messageObjet1 different de 0
// début de la lecture -> conversion en char
     message1=""; // Vide le message precedent
     while (LoRa.available()) {
       message1= String( message1 + (char)LoRa.read());
     }
   }
  LoRa.setFrequency(910E6);   // mulplixage en fréquence du module Lora.
                  //  Chaque objet est connecté à une frequence particulier

   messageObjet2= LoRa.parsePacket();
     if (messageObjet1) {
     message2="";
     while (LoRa.available()) {
       message2= String( message2 + (char)LoRa.read());
     }
   }
  LoRa.setFrequency(905E6);
   messageObjet3= LoRa.parsePacket();
   if (messageObjet3) {
     message3="";
     while (LoRa.available()) {
       message3= String( message3 + (char)LoRa.read());
     }
   }
  LoRa.setFrequency(900E6);
   messageObjet4= LoRa.parsePacket();
   if (messageObjet4) {
     message4="";
     while (LoRa.available()) {
       message4= String( message4 + (char)LoRa.read());
     }
   }
  LoRa.setFrequency(895E6);
   messageObjet5= LoRa.parsePacket();
```

```
   if (messageObjet5) {
     message5="";
     while (LoRa.available()) {
       message2= String( message5 + (char)LoRa.read());
     }
   }
  LoRa.setFrequency(890E6);
   messageObjet6= LoRa.parsePacket();
        if (messageObjet6) {
     message6="";
     while (LoRa.available()) {
       message6= String( message6 + (char)LoRa.read());
     }
   }
  LoRa.setFrequency(915E6); // Refixe à la frequence de debut
  delay(100);
}
void bluetooth (){
// demande à changer tout les objets en reception
  activity = 192;   // correponds 1100 0000
  while (activity=! 255){
   LoRa.setFrequency(850E6);
   LoRa.beginPacket();// entre en mode emmission
   LoRa.print(255); // Envoie un message signal de changement
   LoRa.endPacket(); // fin de envoie
   delay(100);
   LoRa.print(255);
   LoRa.endPacket();
// Verifie si les object sont en mode recption
     LoRa.setFrequency(915E6);
   messageObjet1= LoRa.parsePacket(); // objet ne envoie rien
   if (messageObjet1) {}
   else{bitSet(activity,1);} // met le bit1 de activity à 1
     LoRa.setFrequency(910E6);
   messageObjet2= LoRa.parsePacket();
   if (messageObjet2) {}
   else{bitSet(activity,2);}
     LoRa.setFrequency(905E6);
   messageObjet3= LoRa.parsePacket();
   if (messageObjet3) {}
   else{bitSet(activity,3);}
     LoRa.setFrequency(900E6);
   messageObjet4= LoRa.parsePacket();
   if (messageObjet4) {}
```

```
  else{bitSet(activity,4);}
    LoRa.setFrequency(895E6);
  messageObjet5= LoRa.parsePacket();
  if (messageObjet5) {}
  else{bitSet(activity,5);}
    LoRa.setFrequency(890E6);
  messageObjet6= LoRa.parsePacket();
  if (messageObjet6){}
  else{bitSet(activity,6);}
}
//.......
// Les actions et les demande du bluetooth
//.......
}
```

2. Code annexe pour objet connecté

```
// la valeur de la frequence est à changer selon les objet
// Dans ce code ici present on prend uniquement pour objet1
// avec une frequence de 915E6
#include <SPI.h>
#include <LoRa.h>
char donne[]="donne à envoyer";
const int frequence = 915E6 ;
int ordre;
void information(){
 // fonction qui récupere les information information
 // par exemple
 donne[1]= (char)digitalRead(4);
}
void action(){
 // Définition de l'ensemble de ordre de possible
 }
void setup() {
LoRa.setSPIFrequency(9600);
LoRa.setPins(3,4,2);
LoRa.begin(frequence);
}
void loop() {
 information();
 LoRa.beginPacket();
 LoRa.print(donne);
 LoRa.endPacket();
 LoRa.setFrequency(850E6);
 ordre=LoRa.parsePacket();
```

```
 if (ordre=255) {
   action();
   }
 LoRa.setFrequency(frequence);
}
```

3. Code capteur de pression:

```
int fsrAnalogPin = 0; // FSR branché sur pin Analog 0
 int LEDpin = 11; // connecter LED rouge sur pin 11 (pin PWM)
  int fsrReading;  // Lecture analogique de la tension du pont
                   //    diviseur FSR + Resistance Pull-Down
  int LEDbrightness;

  void setup(void) {
    Serial.begin(9600); // Envoi de message de déboggage sur connexion série
                        // Visible dans le Moniteur Série d'Arduino IDE
    pinMode(LEDpin, OUTPUT);
  }
  void loop(void) {
    fsrReading = analogRead(fsrAnalogPin);
    Serial.print("Analog reading = ");
    Serial.println(fsrReading);
    // Nous devons convertir la valeur analogique lue (0-1023)
    // en une valeur utilisable par analogWrite (0-255).
    // C'est ce que fait l'instruction map!
    LEDbrightness = map(fsrReading, 0, 1023, 0, 255);
    // LED gets brighter the harder you press
    analogWrite(LEDpin, LEDbrightness);
    delay(100);
  }
```

Conclusion et perspectives

Ce projet se compose de deux parties : Partie théorique et partie pratique. En deuxième année on essaye de comprendre réaliser la partie théorique et commencer à élaborer les codes et réaliser le schéma électrique. En 3eme année on commence à faire la partie pratique de ce projet et à réaliser la carte de commande et tester les codes élaborés en 2eme année. Pour le moment on a réussi à faire le schéma électrique et à faire l'application Android qui serait communiqué par Bluetooth avec la carte de commande centrale. Ainsi nous avons réussi à faire les codes nécessaires pour faire fonctionner le module LoRa qui serait utilisée comme moyen de communication entre les objets intelligent et la carte de commande central commandé par l'utilisateur. en termes de perspective, la partie manquant de ce projet et de réaliser un serveur.

Annexe

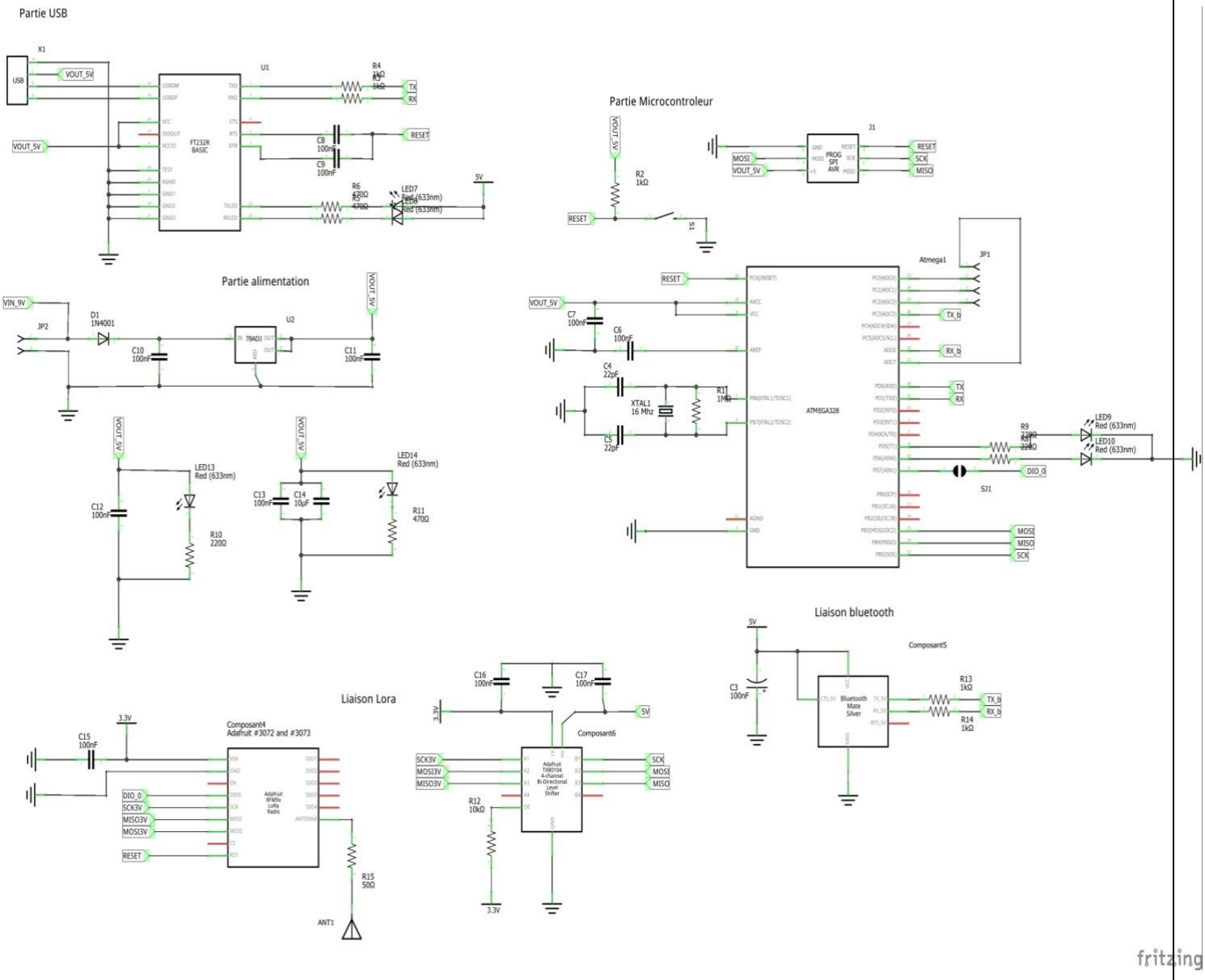
ATMEGA328 : http://www.farnell.com/datasheets/2047852.pdf?_ga=2.130063183.197929673.1528032827-448699733.1526477981&_gac=1.229430766.1526477981.Cj0KCQjwre_XBRDVARIsAPf7zZiqwToQu7H51SPr-dZRjli0GuwIbigLd3HXnNAkPL-g5t6vkL5zlcgaAoTEEALw_wcB

FT232R : http://www.farnell.com/datasheets/1803231.pdf?_ga=2.130063183.197929673.1528032827-448699733.1526477981&_gac=1.229430766.1526477981.Cj0KCQjwre_XBRDVARIsAPf7zZiqwToQu7H51SPr-dZRjli0GuwIbigLd3HXnNAkPL-g5t6vkL5zlcgaAoTEEALw_wcB

Module Lora : https://docs-emea.rs-online.com/webdocs/1536/0900766b815369be.pdf

TXB0104 : http://www.ti.com/lit/ds/symlink/txb0104.pdf

Module Bluetooth : https://www.silabs.com/documents/login/data-sheets/BLE112-DataSheet.pdf



Schema électronique de notre circuit

Les Codes annexes:

```
#include <SPI.h>
int e;
char message1 [60];
int ldr;
int ntc;
void setup()
{
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  // Power ON the module
  sx1272.ON();
  // Set transmission mode and print the result
  e = sx1272.setMode(4);
  Serial.println(F("Setting Mode: state "));
  Serial.println(e, DEC);
  // Select frequency channel
  e = sx1272.setChannel(CH_12_868);
  Serial.println(F("Setting Channel: state "));
  Serial.println(e, DEC);
  // Select output power (Max, High or Low)
  e = sx1272.setPower('H');
  Serial.println(F("Setting Power: state "));
  Serial.println(e, DEC);
  // Set the node address and print the result
  e = sx1272.setNodeAddress(2);
  Serial.println(F("Setting node address: state "));
  Serial.println(e, DEC);
  // Print a success message
  Serial.println(F("SX1272 successfully configured"));
}
void loop(void)
{
  ldr = analogRead(1);
  get_temp();
  sprintf(message1,"LDR value: %i \r\nNTC value: %i\r\n",ldr,ntc);
  e = sx1272.sendPacketTimeout(3, message1);
  Serial.println(message1);
  Serial.print(F("Packet sent, state "));
  Serial.println(e, DEC);
  delay(5000);
}
void get_temp() {
  ntc = analogRead(0) * 5 / 1024.0;
  ntc = ntc - 0.5;
  ntc = ntc / 0.01;
}
```

Capteur TSL2561: https://cdn-learn.adafruit.com/downloads/pdf/tsl2561.pdf

Capteur DHT22: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf