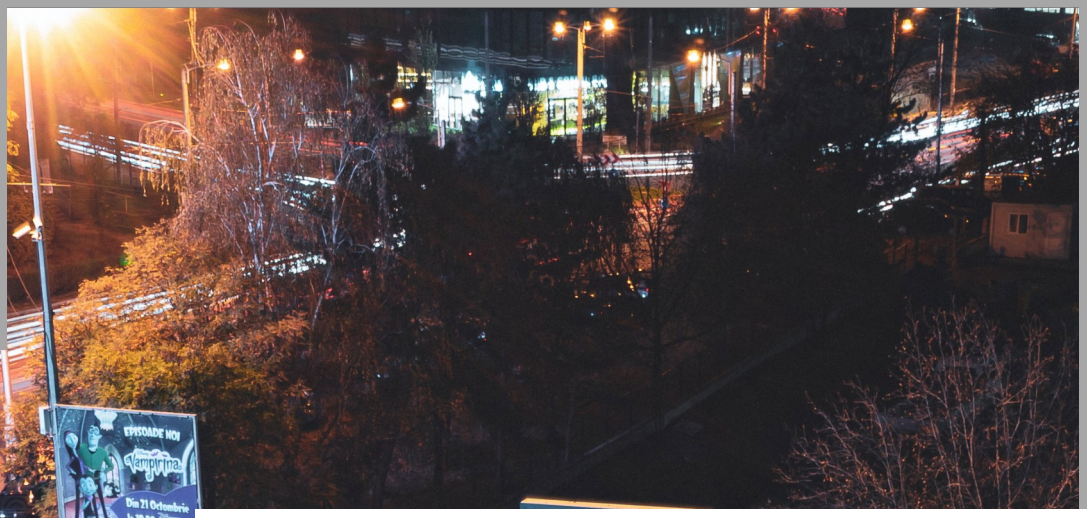
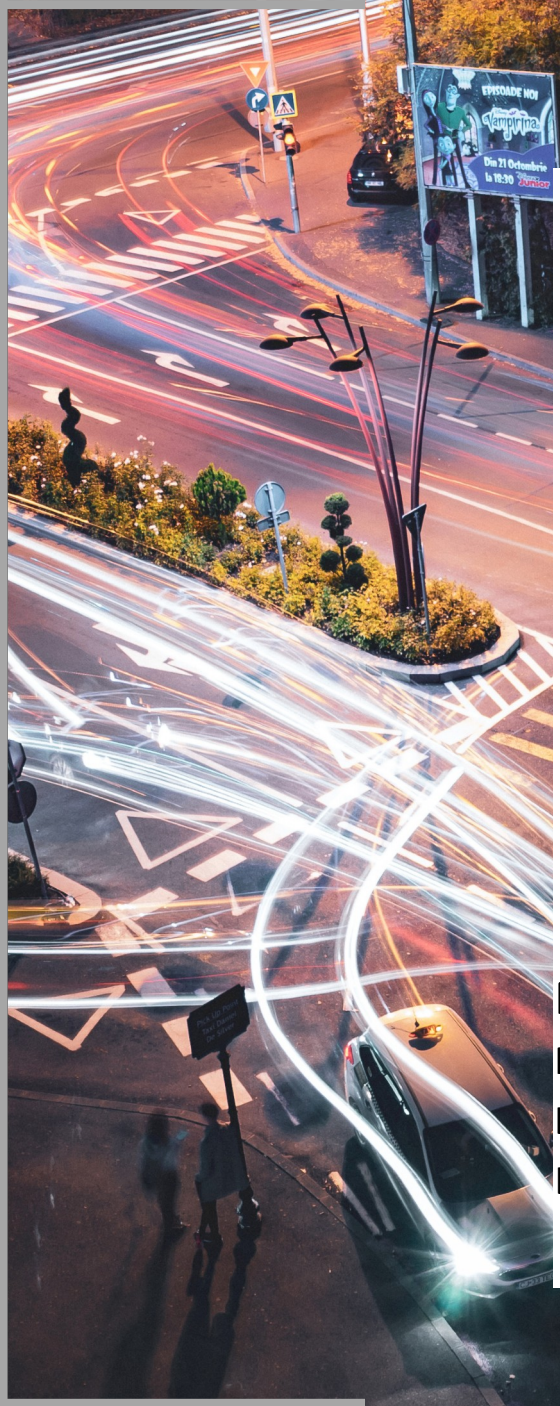
AVIGNON UNIVERSITÉ

CERI

Centre d'Enseignement et de Recherche en Informatique



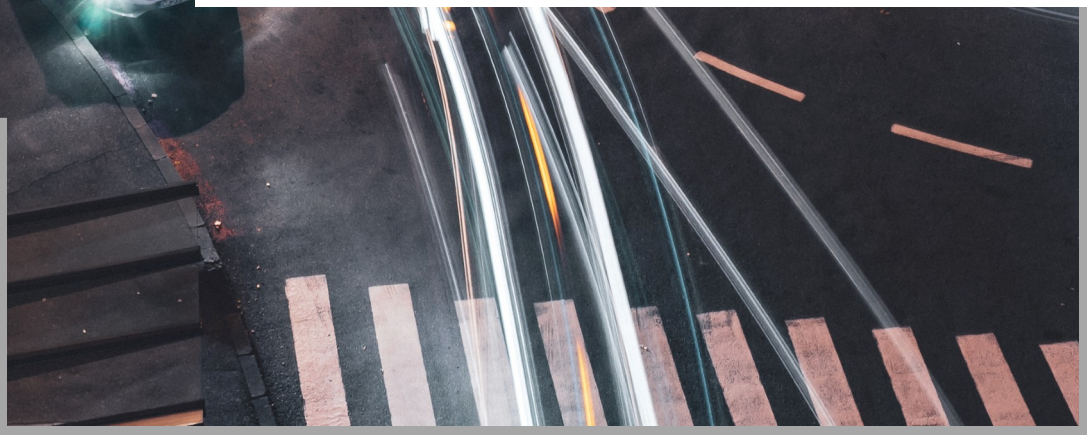


Projet 2021 SMART CITY

BTS SN

Réalisé par

Florian Pichery,Sébastien Ré, Lucas Duboc
Loic Martinelli Mathéo Petit, Joséphine Chevalier
Lohan Dancuo, Nicolas Ninnin, Marine Maronat



smartcity

Académie Aix Marseille
Lycée Alphonse Benoit Isle sur la Sorgue

Table des matières

Partie Commune

Préambule

Le projet SMART CITY est réalisé par deux équipes distinctes respectivement de quatre et cinq personnes.

1. Introduction

1.1 Contexte du projet SMART CITY

Constitution de l'équipe de projet :	Étudiant 1 EC **IR**	Étudiant 2 EC **IR**	Étudiant 3 EC **IR**	Étudiant 4 **EC** IR	Étudiant 5 **EC** IR
Projet développé :		Au lycée ou en centre de formation **Mixte**		En entreprise	
Type de client ou donneur d'ordre (commanditaire) :		Entreprise ou organisme commanditaire : **Oui** Nom : Philippe GOSLAN / Marc SILANUS Adresse : CERI / Université d'Avignon pôle Agroparc Montfavet Contact : Origine du projet : • Idée : Lycée **Entreprise** • Cahier des charges : Lycée **Entreprise** • Suivi du projet : **Lycée** **Entreprise**			
Si le projet est développé en partenariat avec une entreprise :		Nom de l'entreprise : CERI / Adresse de l'entreprise : Université d'Avignon pôle Agroparc Adresse site : Montfavet Tél. : Courriel :			

1.2 Présentation du projet.

Le projet SMART CITY est une demande faite par le CERI (Centre d'Études et de Recherches en Informatique) qui constitue un des départements de l'université d'Avignon.
L'objectif étant de pouvoir piloter une ville dite intelligente sur une maquette, qui contient un parking, un système d'éclairage des voies, une intersection où véhicule et piétons pourraient circuler dans de bonnes conditions.

Cette maquette sera utilisée par les étudiants pour développer des applications de gestion intelligente de la ville. Les logiciels clients créés par les étudiants du CERI utiliseront des technologies avancées telles que les réseaux neuronaux, les systèmes experts, l'intelligence artificielle, etc.

2.Spécifications

2.1. Diagramme UML/SYSML

Voici les différentes parties dont notre système est composé.

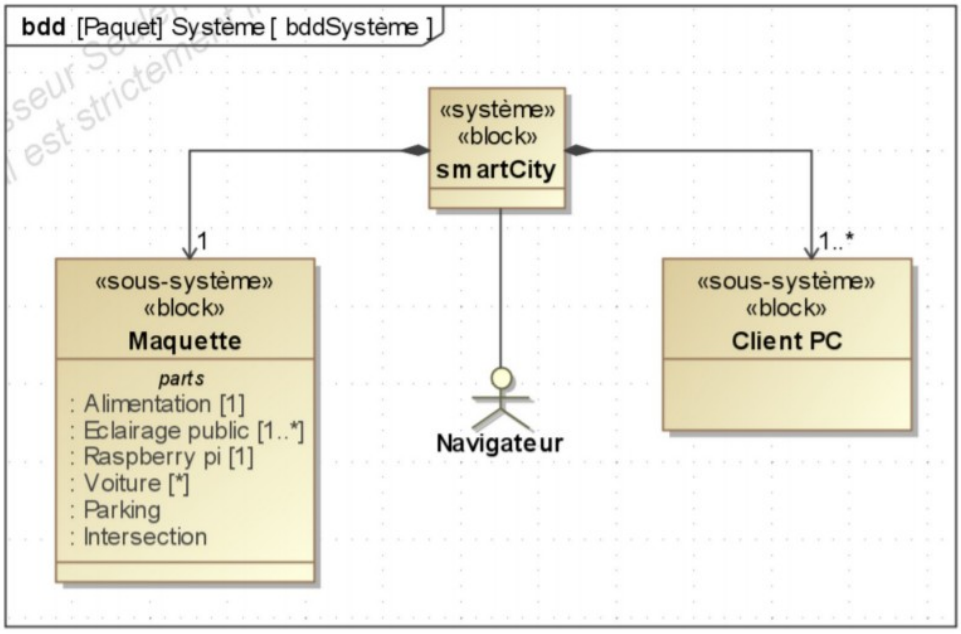


Figure 1 : Diagramme de blocs du système

2.1.1 Diagramme Cas d'Utilisation

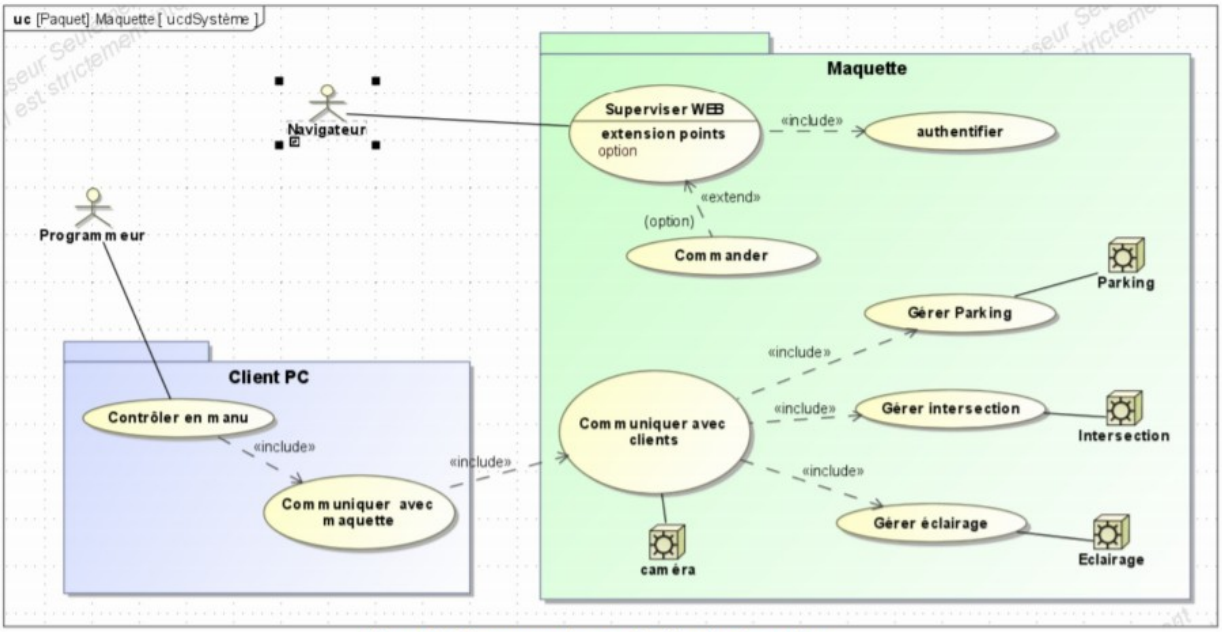


Figure 2 : Diagramme des cas d'utilisation du système

2.1.2 Diagramme de déploiement

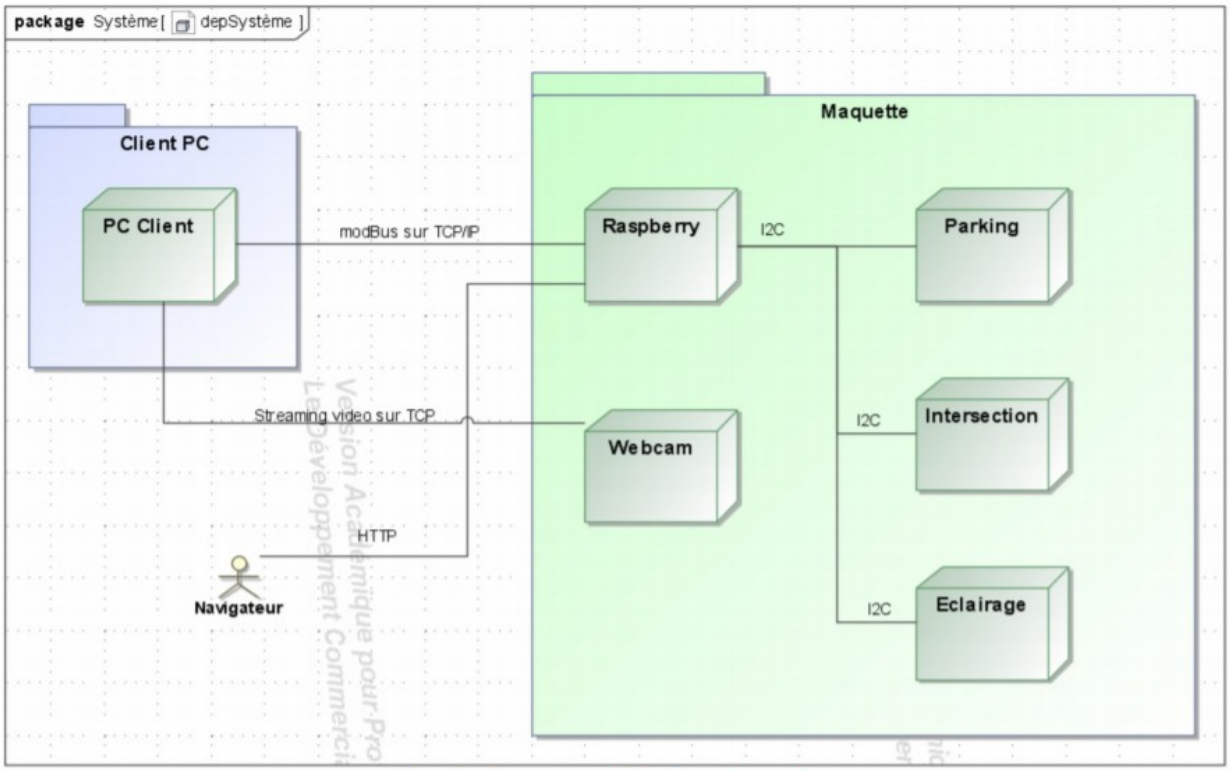


Figure 3 : Diagramme de déploiement du système

2.1.3 Diagramme d'Exigences

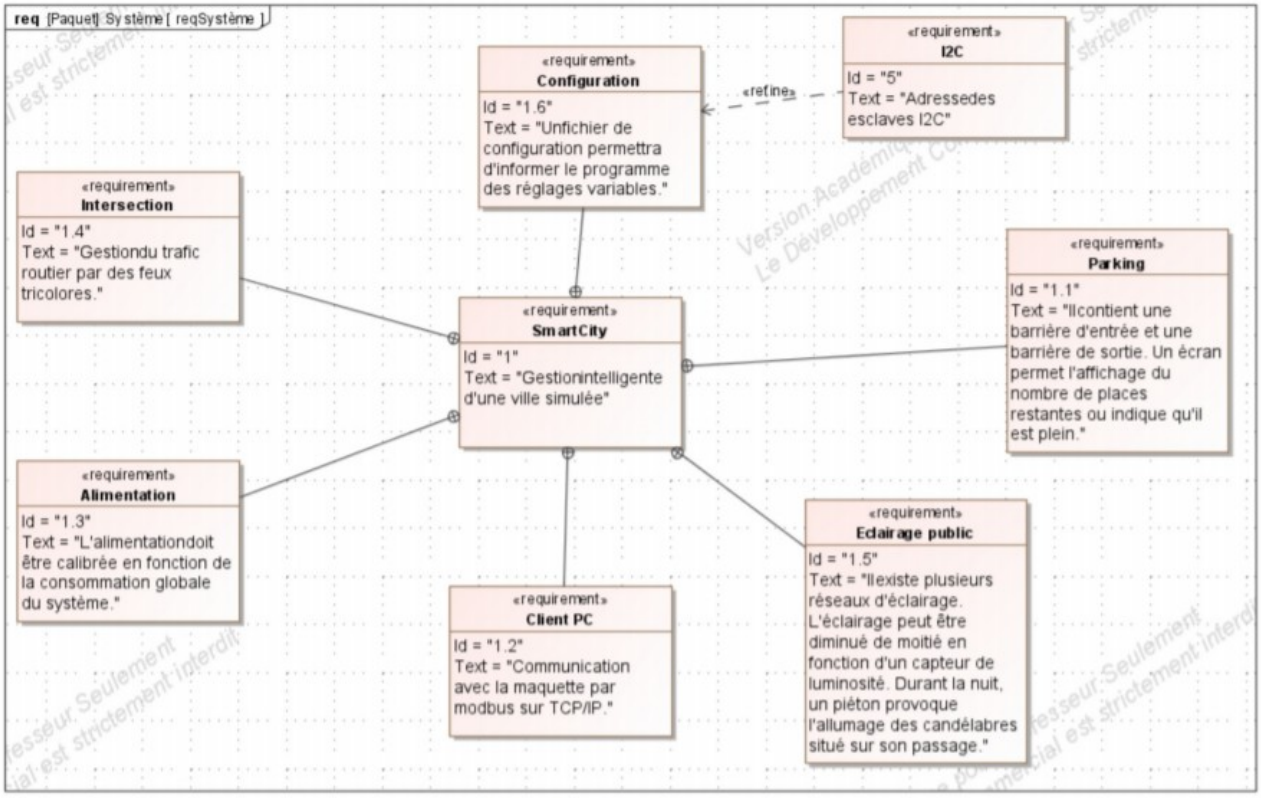
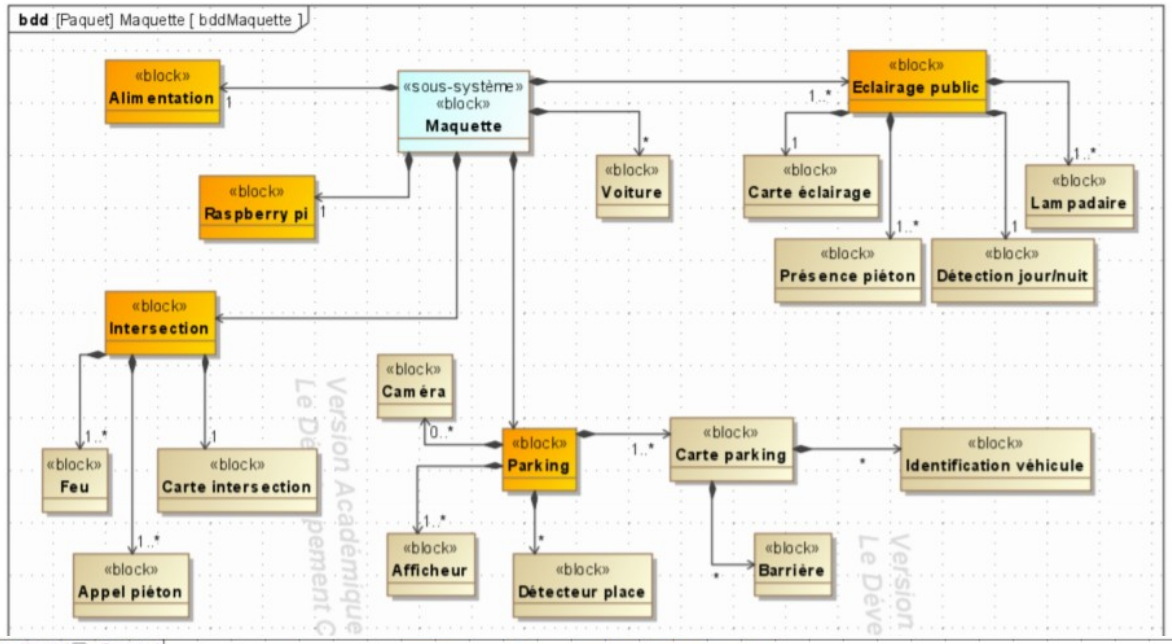
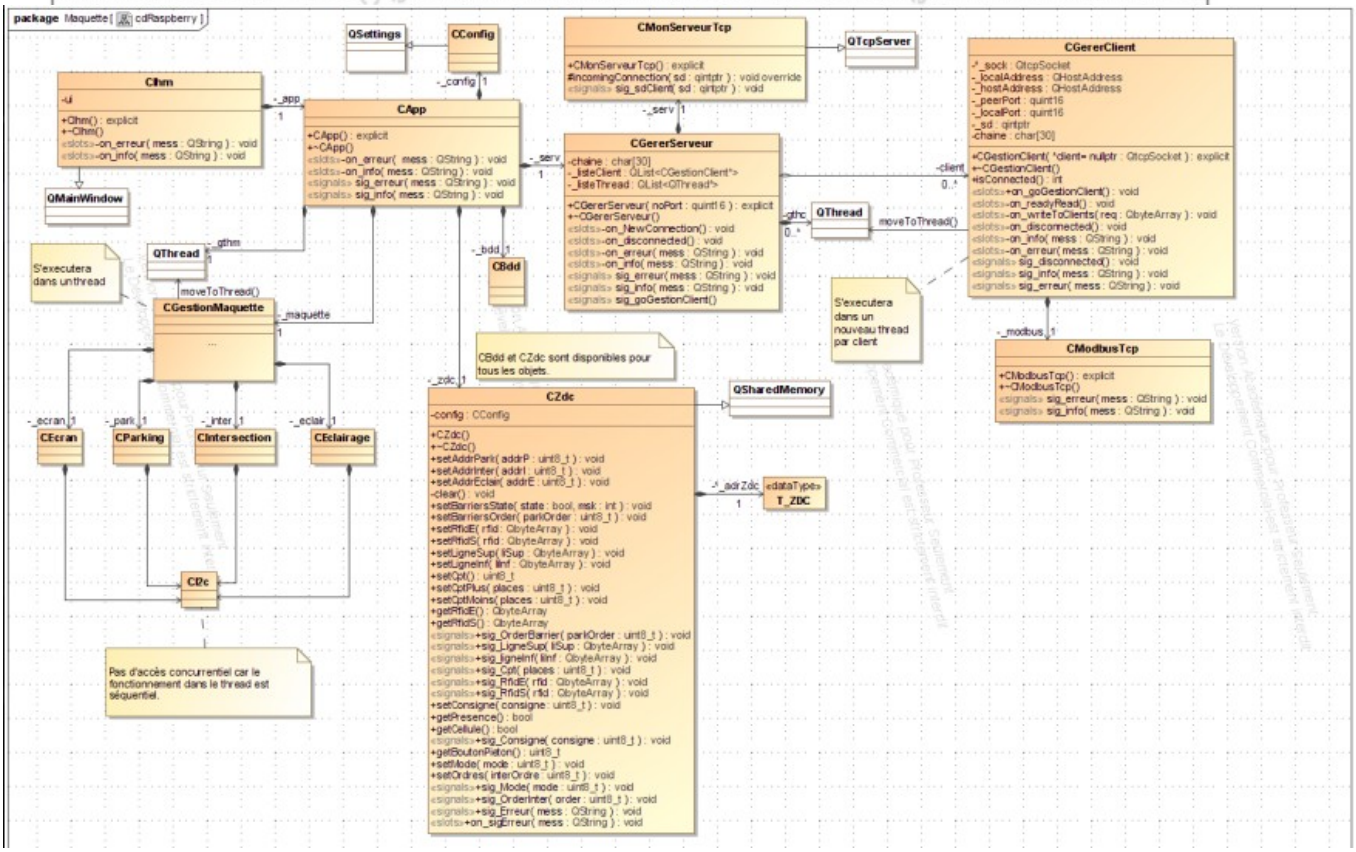


Figure 4 : Diagramme des exigences

2.1.4 Diagramme Architectures Matérielle et Logicielle





2.2 Scénarios des cas d'utilisation

Sous système : Maquette	UC : Superviser WEB
Scénario nominal	
Serveur http/https permettant la supervision distante et graphique de la maquette sous forme de page WEB.	

Sous système : Maquette	UC : Commander
Scénario nominal	
Permet d'émettre des ordres depuis le site WEB vers les éléments de la ville. Ce scénario est optionnel. Il sera implémenté que si le reste fonctionnel.	

Sous système : Maquette	UC : Authentifier
Scénario nominal	
Demande une authentification nom+mot de passe. Un seul compte sera nécessaire.	
Sous système : Maquette	**UC : Communiquer avec clients**
Scénario nominal	
Serveur TCP gérant les clients sous forme de threads. Pour chaque requête d'un client, mettre à jour la zone de données commune et envoyer un signal à l'objet concerné. Répondre au client. Cas particulier des caméras qui seront directement accessibles par leur adresse IP et numéro de port. Le flux vidéo est généré par chaque caméra.	

Sous système : Maquette	UC : Gérer XXX
Scénario nominal	
XXX : Elément de la ville (intersection, parking, éclairage).	

Dès réception d'un signal, lire les données dans la ZDC. Rechercher les modifications à appliquer sur l'élément de la ville. Contrôler la communication avec les éléments de la ville par le bus I2C.

Sous système : Client PC	UC : Contrôler en manu
Scénario nominal	
Permet par un logiciel muni d'une IHM (interface homme machine) de superviser et contrôler les éléments de la ville. Note : Bien que cette partie soit normalement à la charge des étudiant de Master 2, nous développerons une version de test des éléments de la ville.	

Sous système : Client PC	UC : Communiquer avec maquette
Scénario nominal	
Permet la communication avec la maquette suivant les protocoles : • modBus TCP pour superviser et contrôler les éléments de la ville. • http/https/ftp, etc. pour récupérer les flux des caméras et les intégrer dans l'IHM.	

2.3 Contraintes de réalisation

Contraintes financières (budget alloué) :
Budget estimé de 500€ selon avancement du projet à charge partielle du demandeur.
Contraintes de développement (matériel et/ou logiciel imposé / technologies utilisées) :
La spécification, conception et codage seront modélisés.
Contraintes qualité (conformité, délais, …) :
Maintenable, maniable (ergonomie)
Contraintes de fiabilité, sécurité : Les accès logiciels seront sécurisés.

2.4 Ressources mises à disposition des étudiants (logiciels / matériels / documents)

Matériels :

Composants pour la réalisation

Matériel de laboratoire (alimentation, oscilloscope, analyseur logique)

raspberry pi avec rapian

Logiciels :

Système d'exploitation

Logiciel de modélisation SysML/UML : MagicDraw v7.02

Logiciel de conception électronique : KiCAD 5

Logiciel de conception électronique Fritzing uniquement pour illustrer le prototypage rapide

Un logiciel de saisie de schéma et de simulation (Proteus ISIS) pourra éventuellement être utilisé pour illustrer des essais de programmation.

Logiciel Qt C++

Visual Studio code C#

Documents :

Site de la STS SN mettant à disposition les différentes documentations.

3.Fiches de réunions

Compte rendu Réunion		
Projet	***Réalisateur (s)***	***Date***
Smart City	Lohan Dancuo/Marine Maronat	07/01
Sujet		
Réunion de début de projet		
Groupe Présent		***Temps***
IR et EC		1h30

Analyse des ordres : quels sont exactement les ordres qui sont envoyés et les états que nous pouvons recevoir ?
Par exemple pour la barrière :
Ouvre toi : retourne « je suis ouvert » ?
Ferme toi : retourne : « je suis fermé » ?
Questionnement sur les ordres qui transites.

Pour la webcam, elle sera accessible depuis le web (et peut être pour les clients).
L'afficheur du parking sera géré par le groupe IR.

Partie ***ÉCLAIRAGE :***
-plusieurs cartes identiques pour 6 lampadaires contrôlables indépendamment
Allume toi 100 % (2) — **BDD (x?)**
Allume toi 50 % (1) — **BDD (x?)**
Éteint toi (0) — **BDD (x?)**
Déclenchement de test pour savoir si les lampadaires fonctionnent tous
— BDD (une réponse par plaque de 6**) (x?)**
Avoir un retour du courant consommé pour savoir si un lampadaire ne fonctionne pas.
On a un retour d'information en binaire pour la réponse des lampadaires.
Une rue à 1 carte de 6 lampadaires.
Renvoi si il fait jour ou nuit (rajout par rapport à l'ancien Compte Rendu)

-Détecteur de présence (réflectif)
Quelqu'un passe (1) — **BDD (x?)**
Personne n'est là (0) — **BDD (x?)**
2 capteurs par carte de 6 lampadaires.

Partie ***INTERSECTION :***

-Bouton piéton
Quand feu vert des voitures, l'éteint et allume le feu vert des piétons
État de fonctionnement (pour savoir si il s'allume ou non / On ne connaît pas les couleurs)
8 boutons pour l'intersection
-Feu pour les voitures
Feu rouge/orange/vert (cycle automatique) — BDD (x4)
Rouge fix — BDD
Vert fix — BDD
Feu orange clignot. — BDD (x4)
État de fonctionnement (pour savoir si il s'allume ou non / On ne connaît pas les couleurs) — BDD (x4)
Paramétrer la durée du feu rouge et vert.

Partie ***PARKING:***
-Parking
Parking avec obligation d'avoir une puce RFID.
Puce RFID des voitures du parking soit abonné.
Abonnement géré par la base de données.

2 barrières :
Monter (1) **— BDD**
Barrière position haute (1) **— BDD**
Descendre (0) **— BDD**
Barrière position basse (0) **— BDD**
RFID détecter (numéro du badge) **— BDD**

Pour le site web, créer un espace pour ajouter des numéro de badges **— BDD**

-Afficheur
Rouge (plus de places)
Vert
Nombre de place restantes — BDD

-Bouton appel d'urgence
Appelle quelqu'un (notifie les IHM) **— BDD**

Partie Étudiant IR 1 – Équipe 01 : PICHERY Florian

Introduction

1. Cahier des charges initial de l'étudiant IR1.2

Étudiant 2 **IR 12**	*Liste des tâches assurées par l'étudiant* • Développement partiel du logiciel RASPBERRY. Classes à coder : o CGestionMaquette o CParking o CIntersection o CEclairage o CI2c o CZdc	**Installation** : Système d'exploitation **Raspbian**, EDI. **Mise en œuvre :** I2c, Thread, mémoire partagée. Gestion de l'afficheur du parking. **Configuration** : EDI, I2c **Réalisation** : Logiciel Qt C++ avec Qt Creator **Documentation** : Cahier de recettes, défaillances.

2. Objectifs à réaliser.

- Programmation de la Zone de Données Communes (CZdc).
- Programmation de la classe CGestionMaquette.
- Mise en œuvre de la communication I²C entre les éléments de la maquette et la carte Raspberry Pi.
- Emuler les différents éléments pour pouvoir un moyen de tester les ordres ainsi que les réponses attendues.

3. *Planification prévisionnelle*

Projet	296 h	Mar 1/5/21	Mar 6/15/21	
Spécifications du projet	6 h	Mar 1/5/21	Mer 1/6/21	
Étude générale SysML du projet	39 h	Mer 1/6/21	Ven 1/22/21	2
Création de l'IHM debug	3 h	Ven 1/22/21	Ven 1/22/21	3
Conception de CZdc et conception partielle CGestionMaquette	60 h	Mar 1/26/21	Ven 2/19/21	4
Rédaction du dossier	1 h	Ven 2/19/21	Ven 2/19/21	5
Revue 1	30 min	Ven 2/19/21	Ven 2/19/21	
TMO : Communication I2C avec la classe CI2c	39 h	Ven 2/19/21	Mar 3/23/21	7
Conception de la communication avec le parking	23 h	Mar 3/23/21	Jeu 4/1/21	8
Conception de la communication avec l'intersection	23 h	Jeu 4/1/21	Ven 4/9/21	9
Conception de la communication avec l'éclairage	23 h	Mar 4/13/21	Mer 4/21/21	10
Rédaction du dossier	1 h	Mer 4/21/21	Mer 4/21/21	11
Revue 2	30 min	Jeu 4/22/21	Jeu 4/22/21	
Création d'un émulateur pour pouvoir tester les différents composants et voir en réel	16 h	Jeu 4/22/21	Mer 5/12/21	13
Intégration des deux parties logicielles	12 h	Mar 5/18/21	Ven 5/21/21	14
Essais et mise au point avec les deux parties assemblées, et tester avec les clients	10 h	Ven 5/21/21	Mer 5/26/21	15
Rédaction du dossier	1 h	Mer 5/26/21	Mer 5/26/21	16
Dépôt du dossier	1 h	Mer 5/26/21	Mer 5/26/21	
Soutenance finale E62 (Revue 3)	1 h	Mar 6/15/21	Mar 6/15/21	

4. *Planification effective (à la première revue)*

Projet	296 h	Mar 1/5/21	Mar 6/15/21	
Spécifications du projet	6 h	Mar 1/5/21	Mer 1/6/21	
Étude générale SysML du projet	39 h	Mer 1/6/21	Ven 1/22/21	2
Création de l'IHM debug	3 h	Ven 1/22/21	Ven 1/22/21	3
Conception de CZdc et conception partielle CGestionMaquette	36 h	Mar 1/26/21	Mar 2/9/21	4
Rédaction du dossier	12 h	Mer 2/10/21	Ven 2/12/21	5
Revue 1	30 min	Ven 2/19/21	Ven 2/19/21	
TMO : Communication I2C avec la classe CI2c	39 h	Ven 2/19/21	Mar 3/23/21	7
Conception de la communication avec le parking	23 h	Mar 3/23/21	Jeu 4/1/21	8
Conception de la communication avec l'intersection	23 h	Jeu 4/1/21	Ven 4/9/21	9
Conception de la communication avec l'éclairage	23 h	Mar 4/13/21	Mer 4/21/21	10
Rédaction du dossier	1 h	Mer 4/21/21	Mer 4/21/21	11
Revue 2	30 min	Jeu 4/22/21	Jeu 4/22/21	
Création d'un émulateur pour pouvoir tester les différents composants et voir en réel	16 h	Jeu 4/22/21	Mer 5/12/21	13
Intégration des deux parties logicielles	12 h	Mar 5/18/21	Ven 5/21/21	14
Essais et mise au point avec les deux parties assemblées, et tester avec les clients	10 h	Ven 5/21/21	Mer 5/26/21	15
Rédaction du dossier	1 h	Mer 5/26/21	Mer 5/26/21	16
Dépôt du dossier	1 h	Mer 5/26/21	Mer 5/26/21	
Soutenance finale E62 (Revue 3)	1 h	Mar 6/15/21	Mar 6/15/21	

5. *Choix de cette planification*

J'ai choisi cette planification, appelée planification itérative, qui consiste à réaliser les différentes tâches en fonction des différentes itérations, tout en tenant compte de la priorité de chaque tâche. J'ai donc suivi ce schéma-là :

Tâche	Priorité	Itération de réalisation
Mise en place de la Zone de données communes (CZdc)	Haute	1
Programmation de la classe CGestionMaquette (début)	Moyenne	1
Programmation de la classe CGestionMaquette (fin)	Moyenne	2
Mise en œuvre de la communication I²C entre les éléments de la maquette et la carte Raspberry Pi	Haute	2
Emuler les différents éléments pour pouvoir un moyen de tester les ordres ainsi que les réponses attendues	Faible	2/3

1ère itération :

Dans un premier temps, la création de la Z.D.C est essentielle pour gérer les différentes données qui seront transitées entre la maquette et les clients via la Raspberry. L'objectif de cette zone est d'avoir une zone mémoire pour la gestion de la communication I²C pour ce qui est des valeurs et des ordres envoyés par les clients. La classe CGestionMaquette a pour objectif de créer la boucle permettant de demander aux différentes parties leurs états et les mettre à jour dans la Z.D.C.

2ème itération :

Ensuite, je viendrai à bout de la classe CGestionMaquette, tout en commençant la mise en œuvre de la communication par le bus I²C. Cette communication s'effectuera selon un nombre de trames précis vu lors de réunions avec les EC travaillant sur les différents éléments de la maquette.

3ème itération :

Pour finir, cette itération me permettra de pouvoir mettre en commun avec mon co-équipier nos différentes classes tout en faisant attention à la compatibilité des différentes parties afin de ne pas rentrer dans des problèmes. Et j'en profi-

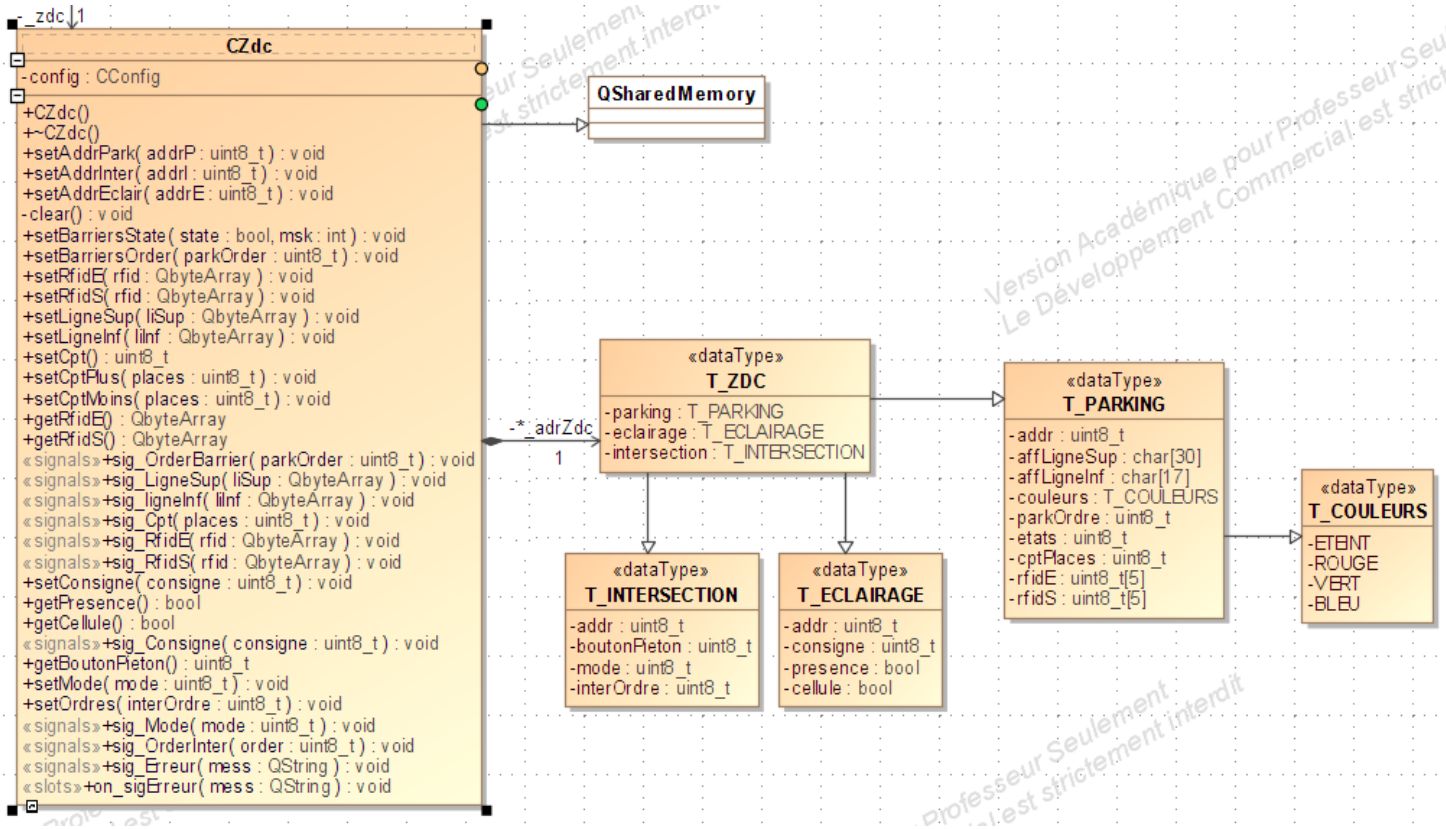
terai pour faire des essais avec un émulateur (qui reste à définir) pour vérifier que mon programme réagit comme il le doit, et qu'il est capable d'agir en fonction.

Ressources logicielles du projet

Désignation	Caractéristiques
OS de la Raspberry Pi 3+	RaspiOS Buster (update du 02/12/2020)
Gestionnaire de versions	GitHub (via Git ou GitHub Desktop)
IDE (environnement de développement)	QtCreator
API GUI	Qt 5.5.1

1ère itération :

Diagramme de la classe CZdc



2nde itération :

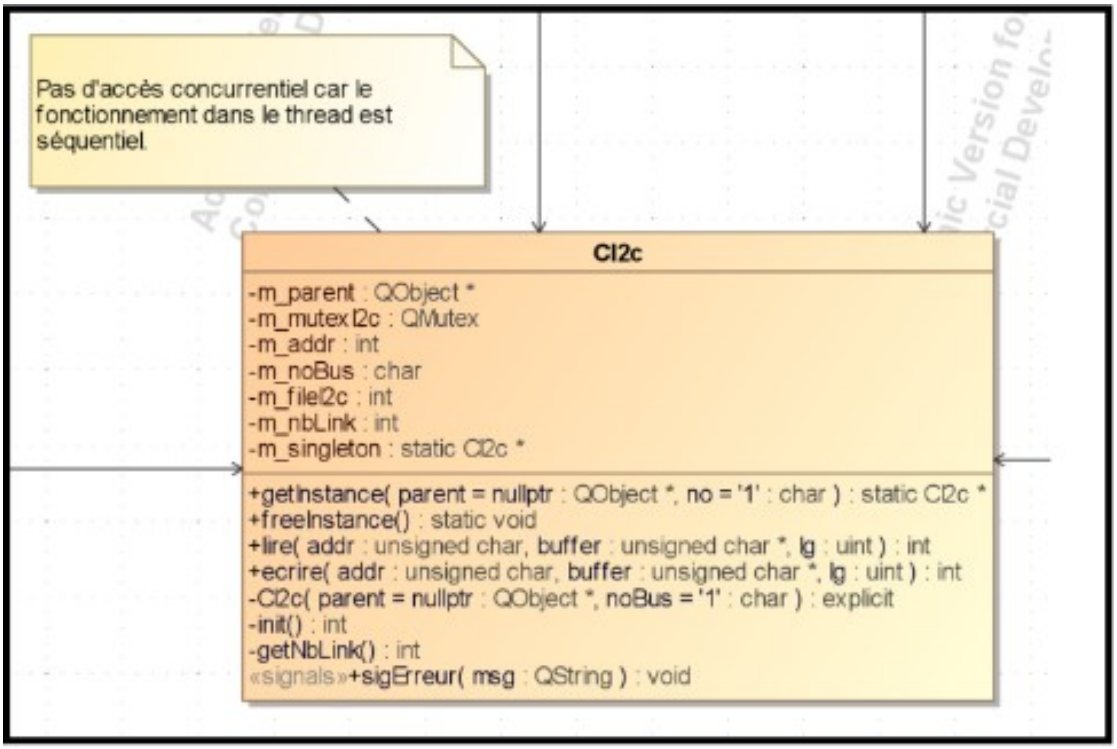
1. Planification effective (à la deuxième revue)

Projet	310 hrs	mar. 05/01/21	mar. 15/06/21	
Spécifications du projet	6 hrs	mar. 05/01/21	mer. 06/01/21	
Étude générale SysML du projet	39 hrs	mer. 06/01/21	ven. 22/01/21	2
Création de l'IHM debug	3 hrs	ven. 22/01/21	ven. 22/01/21	3
Conception de CZdc et conception partielle CGestionMaquette	36 hrs	mar. 26/01/21 2:00 PM	mar. 09/02/21 6:00 PM	4
Rédaction du dossier	1 hr	mer. 10/02/21	mer. 10/02/21	5
Revue 1	30 mins	ven. 19/02/21	ven. 19/02/21	
TMO : Communication I2C avec la classe CI2c	40 hrs	ven. 19/02/21	mar. 23/03/21	7
Conception de la communication avec le parking	10 hrs	mar. 23/03/21	ven. 26/03/21	8
Conception de la communication avec l'intersection	10 hrs	ven. 26/03/21	mer. 31/03/21	9
Conception de la communication avec l'éclairage	10 hrs	mer. 31/03/21	mar. 06/04/21	10
Conception de la communication avec l'écran	10 hrs	mar. 06/04/21	ven. 09/04/21	11
Intégration des deux parties logicielles	12 hrs	mar. 13/04/21	ven. 16/04/21	12
Création d'un émulateur pour pouvoir tester les différents composants et voir en réel	23 hrs	lun. 19/04/21 5:00 PM	ven. 30/04/21 5:00 PM	13
Rédaction du dossier	4 hrs	ven. 16/04/21	ven. 16/04/21	13
Revue 2	0 mins	mar. 04/05/21	mar. 04/05/21	
Essais et mise au point avec les deux parties assemblées, et tester avec les clients	26 hrs	mar. 11/05/21 2:00 PM	mar. 25/05/21 4:00 PM	16
Rédaction du dossier	1 hr	mar. 25/05/21	mar. 25/05/21	17
Dépôt du dossier	1 hr	mer. 26/05/21	mer. 26/05/21	
Soutenance finale E62 (Revue 3)	1 hr	mar. 15/06/21	mar. 15/06/21	

2. Présentation du travail effectué

Durant cette deuxième itération, j'ai pu effectuer la mise en œuvre de la communication en I2C entre la Raspberry Pi et les divers éléments de la maquette, à savoir : un écran Grove LCD sous forme d'une carte fille mise directement sur la Raspberry, une carte permettant de contrôler 6 éclairages USB, une carte permettant de contrôler une barrière de parking avec lecteur RFID et une carte permettant de pouvoir gérer une in-

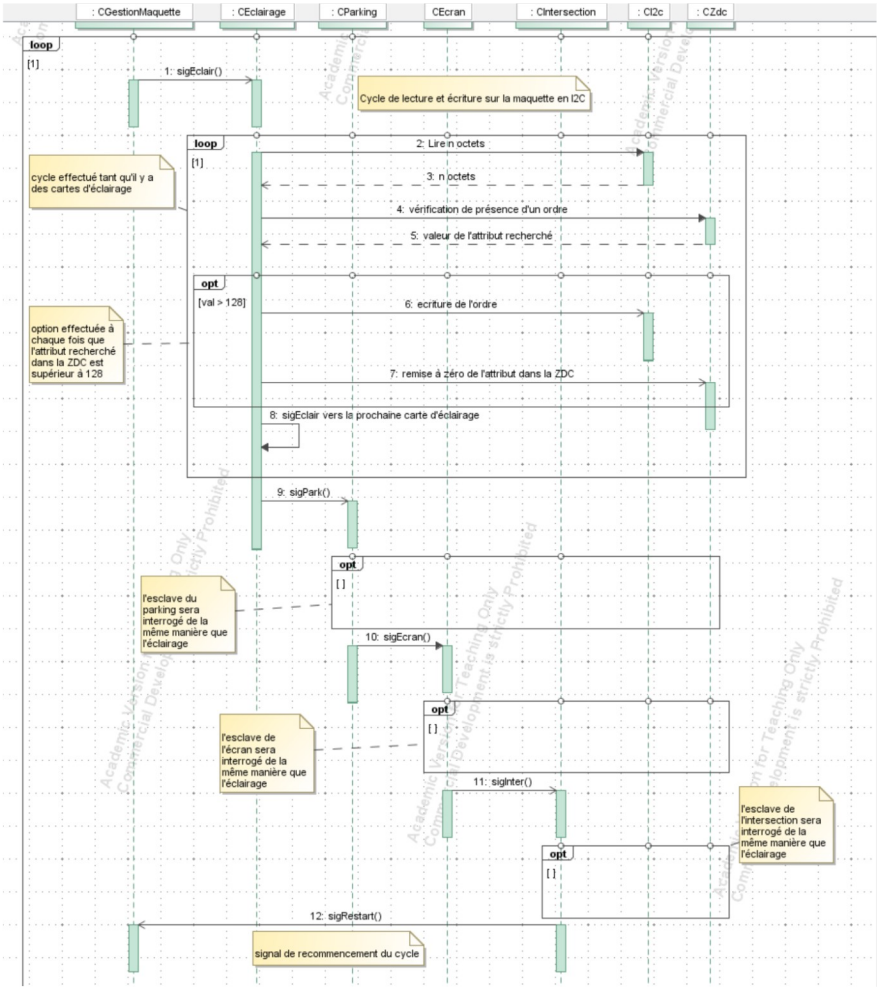
tersection. Voici la classe CI2c dans mon diagramme de classe. Celle-ci m'a servi à communiquer en I2C avec la maquette. Toutes les classes de la maquette ont accès à cette classe. Pour initialiser la communication, chaque classe doit « getInstance() » dans son constructeur et « freeInstance() » dans son destructeur. Et ensuite effectuer des « lire() » et « ecrire() » en mentionnant l'adresse de la partie de maquette, le buffer, où les octets reçus seront stockés, et lg correspondant à la longueur en octets du message attendu.



Partie de cdRaspberry

Pour arriver à ce résultat, j'ai dans un premier temps voulu faire l'éclairage, pour deux raisons. La première raison étant que c'est la carte qui me paraissait, la plus compliquée à gérer. La deuxième raison est que le cycle lecture/écriture peut se faire sur plusieurs cartes, en très grand nombre. En effet, on peut rajouter des cartes d'éclairages en s'alignant sur l'adresse de la carte précédente.
Donc j'ai commencé à voir se dessiner un cycle de lecture, avec en premier lieu, l'éclairage, qui permettra de pouvoir passer à une autre partie après toute(s) la(es) carte(s) vérifiée(s).

Pour expliquer le cycle en général, j'ai fait un diagramme de séquences pour faire transparaître toutes les étapes du cycle de lecture.



sdGestionMaquette

2.1 Partie Éclairage

Au lancement de CApp, cette classe envoie un signal à CGestionMaquette (sig_go) pour lancer le cycle. Donc dans CGestionMaquette, je vais envoyer un signal (sigEclair) à CEclairage qui gère la partie lecture/écriture de l'éclairage de la maquette. En paramètres du signal, je mets l'adresse I2C de la carte d'éclairage, le nombre de cartes d'éclairage de la maquette défini dans un fichier .ini et enfin mettre l'adresse de base, soit l'adresse de la première carte de la maquette, donc l'adresse 20.

```
//Eclairage : lance le cycle de l'éclairage
int addr = _zdc->getAddrEclair();
int nb = _zdc->getNbEclairage();
emit sigEclair(addr, nb, addr);
```

Code dans CGestionMaquette.cpp

Pour lire les données demandées à la carte d'éclairage, j'ai fait une union binaire pour pouvoir récupérer les données précisément sur tel bit et ainsi pouvoir avoir les bonnes informations.

```
U_LAMP octet;
_i2c->lire(static_cast<unsigned char>(addr), &octet.octet, 1); // Lecture

uint8_t indice = static_cast<uint8_t>(addr-addr_base);

_zdc->setCellule(indice, octet.partie.bitJN);
_zdc->setPresence(indice, octet.partie.bitPresence);
_zdc->setLampFonct(indice, octet.partie.bitLamps);
```

Code dans Ceclairage.cpp

Le paramètre « indice » est un paramètre qui permet de choisir la bonne zone mémoire dans la Zone de Données Communes (ZDC). Une fois la lecture faite, je vais vérifier dans la ZDC si un ordre a été reçu, afin d'effectuer l'ordre du client en priorité sur le cycle de lecture.

```
unsigned char ordre = _zdc->getConsigneEclair(indice);

if(ordre > ORDRE_RECU){
    ordre -= ORDRE_RECU;
    _i2c->ecrire(static_cast<unsigned char>(addr), &ordre, 1);
    _zdc->setConsigneEclair(indice, ordre);
}
```

Code dans CEclairage.cpp

Pour passer à la carte suivante, je renvoie un signal de CEclairage à CEclairage en incrémentant la valeur de « addr », d'où l'intérêt d'utiliser le paramètre « addr_base ».

```
addr = addr + 1;

emit sigEclair(addr, nb, addr_base);
```

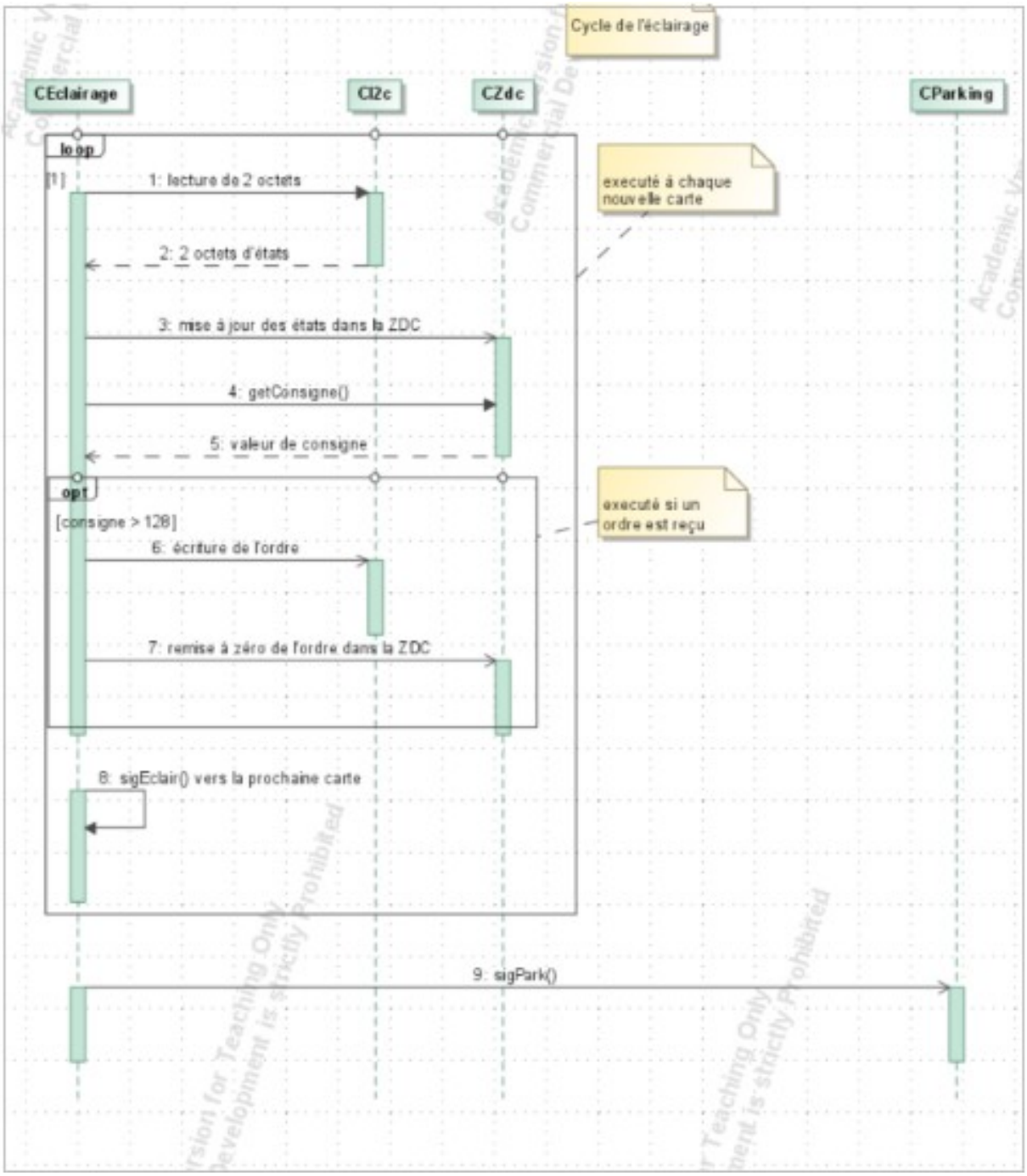
Code dans CEclairage.cpp

Afin de vérifier que plus aucune carte d'éclairage ne soit à traiter, j'ai fait une petite boucle if pour envoyer un signal vers la prochaine partie. La prochaine partie que j'ai choisi de traiter est le parking. La raison est qu'après le parking vient l'écran qui sera géré de manière très rapide.

```
{
    if (addr > (addr_base+nb-1)){
        emit sigParking();
        return;
    }//if addr
```

Code dans Ceclairage.cpp

J'ai donc fait un diagramme de séquences afin de pouvoir visualiser la situation de manière simplifiée et plus courte.



sdEclairage

2.2 Partie Parking

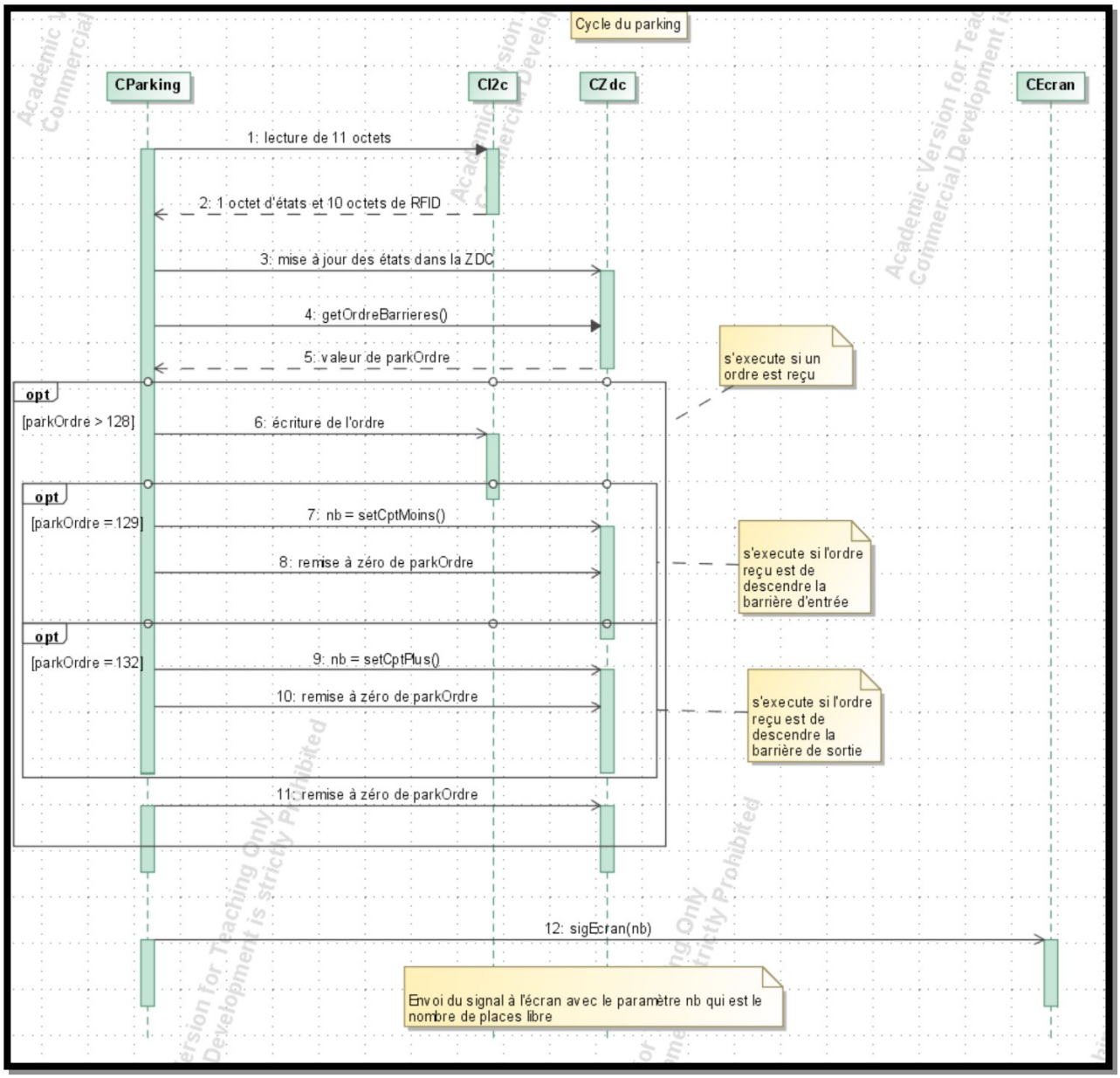
Une fois la partie éclairage effectuée, le reste du travail est très répétitif, ne subissant que de légère modification au niveau du cycle en lecture ou en écriture. La différence entre le parking et l'éclairage au niveau du cycle de lecture est le nombre d'octets qui doivent être lus : passant de 2 octets à 11 octets de lecture. Donc le premier travail est de prendre chaque paquet d'octet et de le traiter au bon endroit. Le premier octet représente le premier paquet. Dans cet octet, les états de la maquette sont lus. Les deux autres paquets de 5 sont donc les paquets RFID, où sont lus les adresses RFID des voitures entrantes et sortantes.

```
unsigned char addr = static_cast<unsigned char>(_zdc->getAddrPark());

unsigned char parking[11];
QByteArray RFIDe;
QByteArray RFIDs;

_i2c->lire(addr, parking, 11); // Lecture
T_PARK_STATE *parkState;
parkState = reinterpret_cast<T_PARK_STATE *>(&parking[0]);
_zdc->setEtatsBarrieres(parkState->bitsStates);

RFIDe = QByteArray(reinterpret_cast<char *>(parking+1), 5); //Conversion de unsigned char * vers QByteArray
_zdc->setRfidE(RFIDe);

RFIDs = QByteArray(reinterpret_cast<char *>(parking+6), 5); //Conversion de unsigned char * vers QByteArray
_zdc->setRfidS(RFIDs);

emit sigEcran(static_cast<QString>(_zdc->getCpt()));
```

Code dans CParking.cpp

Une fois le cycle de lecture terminé, et que chaque paquet et stocké au bon endroit dans la ZDC, et bien, retour à la vérification d'un ordre avant de passer à la partie suivante. Pour passer à la partie suivante, nous envoyons un signal (sigEcran) avec un paramètre cherché directement dans ZDC. Ce paramètre est le nombre de places restantes dans le parking. En effet, selon l'ordre qui est envoyé par la partie client, le nombre de place changera, en s'incrémentant ou en se décrémentant.

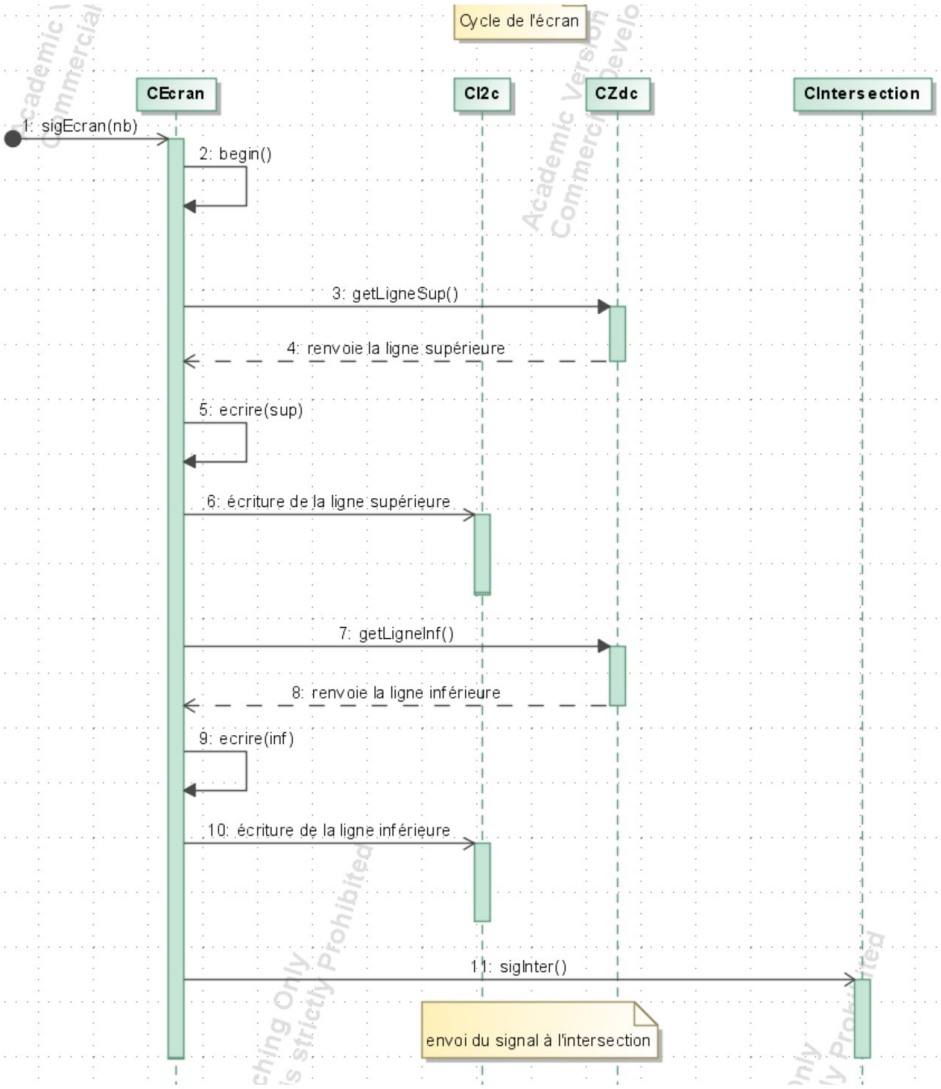
Pour illustrer cela, j'ai donc fait un diagramme de séquences qui a lui aussi pour but de simplifier le processus, sans pour autant négliger la précision.



sdParking

2.3 Partie Écran

Pour l'écran, pas de lecture, ici c'est uniquement de l'écriture. Au début de la création de cette partie du cycle, l'écriture en I2C était comme « privée » dans le sens où les clients n'avaient aucun impact sur l'écran. Mais après une réunion au sein du groupe, nous en sommes venus à changer notre façon de voir le problème et donc de rendre cette partie « publique » au client. L'écran Grove LCD fonctionne sur une bibliothèque qui m'a été donnée par mon professeur. Après avoir vu l'exemple donné et comment cet écran marchait, j'ai pu faire en sorte que cette partie effectue une lecture de ce qui est écrit dans la ZDC, tout en faisant en sorte de l'initialiser à la base, au cas-où rien n'est envoyé par les clients. Pour tout bien expliquer, j'ai un diagramme de séquences qui mentionne les méthodes précises de la bibliothèque de l'écran.



sdEcran

2.4 Partie Intersection

Pour en finir avec le cycle de lecture, et ensuite de pouvoir le recommencer, je vais traiter l'intersection en dernier. Comme le parking, ici rien n'est vraiment différent de l'éclairage en termes de logique, la seule vraie différence est que les données doivent être stockées à deux endroits distincts, étant donné que l'intersection contient 2 voies. Pour les différencier, une voie sera la voie 1 et l'autre voie la voie 2. Ici aussi, comme sur le parking, j'utilise une union binaire afin de cibler les bits et donc de stocker telle valeur à son endroit dans la ZDC.

```
unsigned char addr = static_cast<unsigned char>(_zdc->getAddrInter());

//READ
unsigned char inter[2];
_i2c->lire(addr, inter, 2); // Lecture

U_READ octet;
octet.octet = inter[0];//1er octet
_zdc->setModeVoies(octet.partie.bitMode);
_zdc->setOrdresFeu1(octet.partie.bitCouleurs);
_zdc->setBoutonPietonVoie1(octet.partie.bitBoutons);

octet.octet = inter[1];//2ème octet
_zdc->setOrdresFeu2(octet.partie.bitCouleurs);
_zdc->setBoutonPietonVoie2(octet.partie.bitBoutons);
```

Code dans Cintersection.cpp

Pour l'écriture, je vais regarder dans un premier temps si un ordre de changement de mode est envoyé, et si un ordre est envoyé, trier les ordres qui sont reçu. Je vais mettre d'un côté les ordres de changement de mode vers automatique et orange clignotant et de l'autre, l'ordre de passage en manuel. La raison à cela est qu'en manuel nous pouvons aussi changer les couleurs des voies, quelque chose non géré dans les deux autres cas. Donc la première étape de tri effectuée, je vais ensuite voir si l'ordre de changement de couleur en manuel est effectué sur la voie 1 ou la voie 2, afin de traiter chaque demande comme il faut.

```
    uint8_t mode = _zdc->getModeVoies();

    if(mode > ORDRE_RECU){
        unsigned char ordre;
        T_WRITE *write;
        write = reinterpret_cast<T_WRITE *>(&ordre);
        write->bitMode = _zdc->getModeVoies() - 128;

        _i2c->ecrire(addr, &ordre, 1);
        _zdc->setModeVoies(ordre);
    }//IF ordre reçu
```

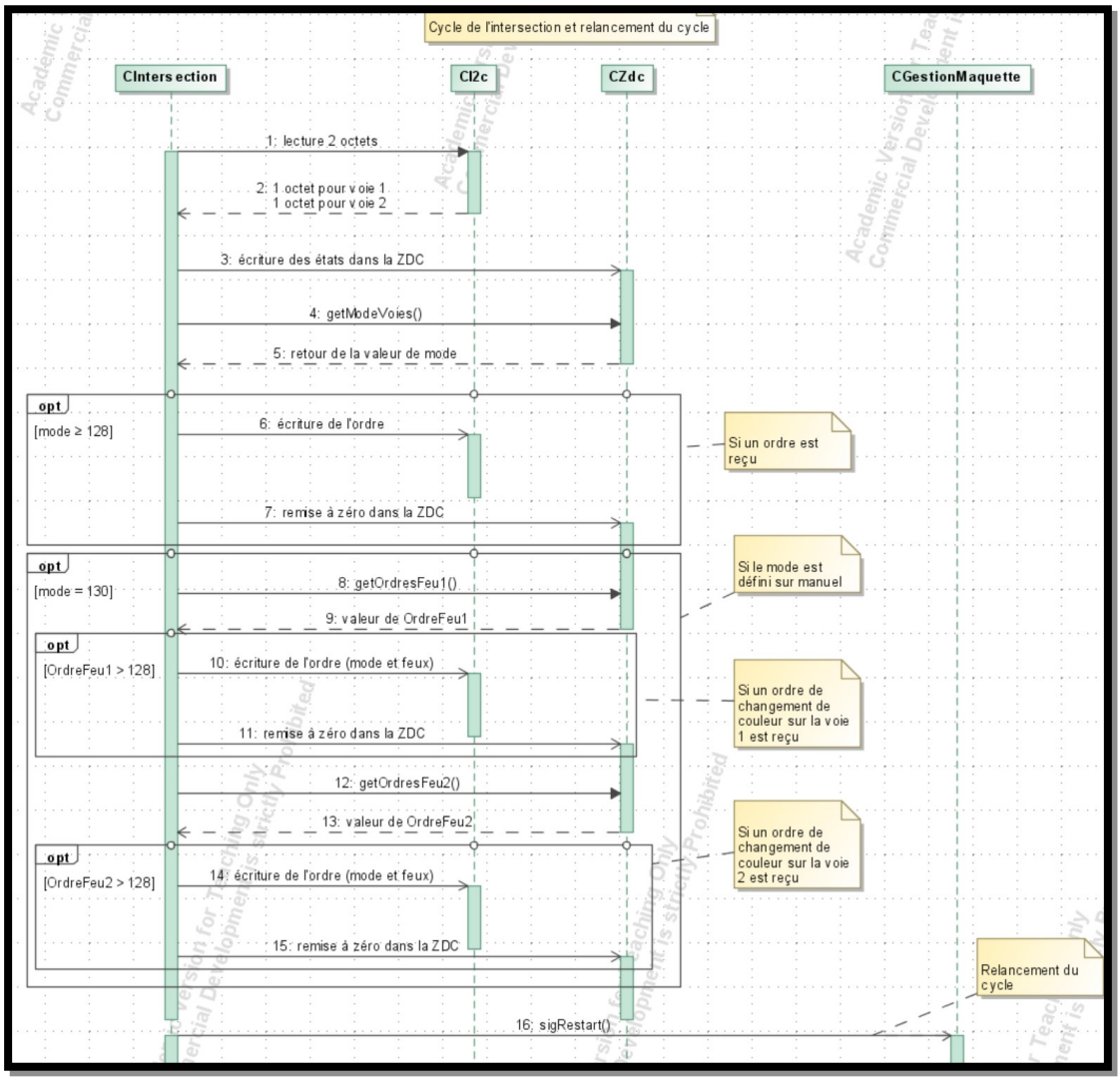
Ordre de changement de voie normal (automatique/orange clignotant) dans CIntersection.cpp

```
if(mode == MODE_MANUEL){
    if(_zdc->getOrdresFeu1() > 128){ //getOrdresFeu1 : pour différencier les ordres pouvant être envoyés
    unsigned char ordre;
        T_WRITE *write;
        write = reinterpret_cast<T_WRITE *>(&ordre);
        write->bitMode = _zdc->getModeVoies() - 128;
        write->bitCouleurs1 = _zdc->getOrdresFeu1() - 128;

        _i2c->ecrire(addr, &ordre, 1);

        _zdc->setModeVoies(ordre - write->bitCouleurs1);
        _zdc->setOrdresFeu1(ordre - write->bitMode);
    }//IF manu voie1
```

Ordre de passage en manuel et changement de couleur sur voie 1 dans CIntersection.cpp

Le cycle se termine par l'émission du signal sigRestart qui a pour but de relancer le cycle de lecture depuis le début en passant par CGestionMaquette. J'ai donc fait un diagramme de séquence pour illustrer le fonctionnement de CIntersection.

sdInter



3.Tests de validation de cette itération

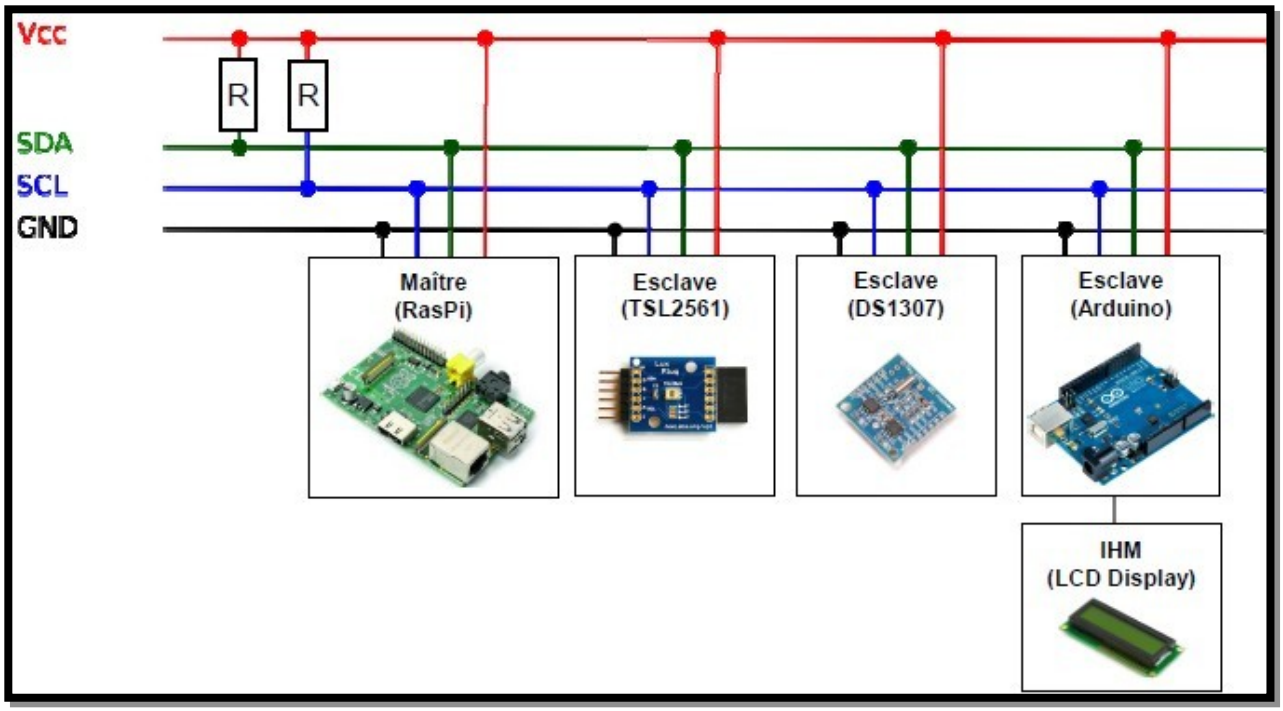
Désignation	Objectif attendu	Résultat obtenu
Avoir un cycle de lecture et écriture fonctionnel.	Possibilité de pouvoir avoir des signaux s'envoyant entre les classes et faire un tour complet.	Les signaux s'envoient et permettent de passer d'une classe à une autre sans souci.
Pouvoir communiquer avec une partie externe à la Raspberry en I2C.	Pouvoir récupérer des données venant d'une partie de la maquette.	Données récupérables par le biais du bus I2C.
Pouvoir stocker les bonnes données au bon endroit dans la ZDC.	Données conforme à la réalité, et stockées dans le bon attribut dans la ZDC.	Les données sont stockées au bon endroit et correspondent à ce qui est sur la maquette.

4. Partie Physique

Présentation

La norme I2C (Inter-Integrated Circuit) a été créée pour fournir un moyen simple de transférer des informations numériques entre des capteurs et des microcontrôleurs. I2C à l'avantage de n'avoir besoin que de deux connexions de signalisation (l'emploi de plusieurs périphériques sur les deux connexions est assez simple et on reçoit une confirmation que les signaux ont été correctement reçus). Les deux connexions du bus I2C se nomment SCL (Serial Clock Line) et SDA (Serial Data line). Un périphérique sur le bus I2C est considéré comme le périphérique maître. Son travail consiste à coordonner le transfert des informations entre les autres périphériques (esclaves) qui sont connectés. Il ne doit y avoir

qu'un seul maître qui contrôle les autres composants auxquels il est connecté.
La figure ci-dessous montre un maître I2C avec plusieurs esclaves I2C.



Les périphériques I2C ont besoin d'une masse commune pour communiquer.
Les périphériques esclaves sont identifiés par leur numéro d'adresse. Chaque esclave doit avoir une adresse unique. Certains appareils I2C ont une adresse fixe alors que d'autres permettent que l'on configure leur adresse en définissant les broches à HIGH ou LOW ou en envoyant des commandes d'initialisation.

Le bus I2C permet d'échanger des informations sous forme série avec un débit pouvant atteindre 100 kilobits/s ou 400 kilobits/s pour les versions les plus récentes.
Ses points forts sont les suivants :
C'est un bus série bifilaire utilisant une ligne de donnée SDA et une ligne d'horloge SCL ;
Le bus est multi maîtres ;
Chaque abonné dispose d'une adresse codée sur 7 bits, on peut donc connecter simultanément 128 abonnés d'adresses différentes sur le même bus (sous réserve de ne pas le surcharger électriquement = la charge) ;
Un acquittement est généré pour chaque octet de donnée transféré ;
Un procédé permet de ralentir l'équipement le plus rapide pour s'adapter à la vitesse de l'équipement le plus lent lors d'un transfert ;

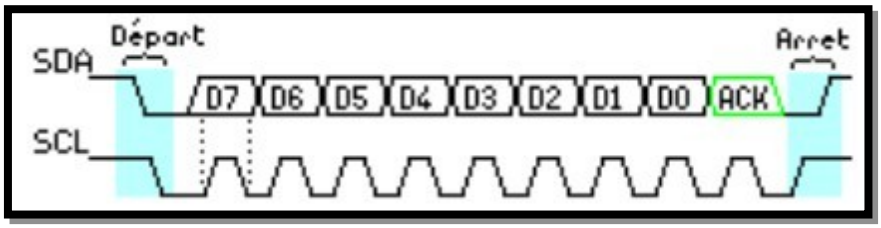
Le nombre maximal d'abonné n'est limité que par la charge capacitive maximale du bus qui peut être de 400 pF ;
Les niveaux électriques permettent l'utilisation des circuits en technologies CMOS, NMOS ou TTL.

Protocole

Le protocole I2C définit la succession des états logiques possibles sur SDA et SCL, et la façon dont doivent réagir les circuits en cas de conflits.

La prise contrôle du bus

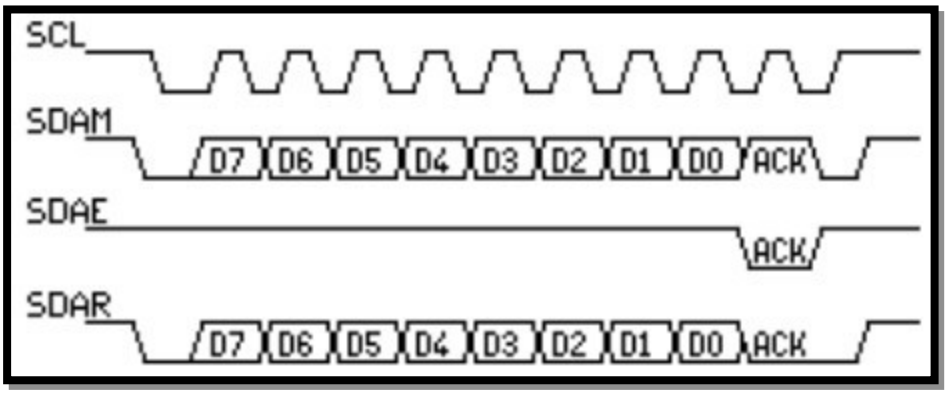
Pour prendre le contrôle du bus, il faut que celui-ci soit au repos (SDA et SCL à '1').
Pour transmettre des données sur le bus, il faut donc surveiller deux conditions particulières :
La condition de départ. (SDA passe à '0' alors que SCL reste à '1')
La condition d'arrêt. (SDA passe à '1' alors que SCL reste à '1')
Lorsqu'un circuit, après avoir vérifié que le bus est libre, prend le contrôle de celui-ci, il en devient le maître. C'est lui qui génère le signal d'horloge.



Exemple de condition de départ et d'arrêt.

La transmission d'un octet

Après avoir imposé la condition de départ, le maître applique sur SDA le bit de poids fort D7. Il valide ensuite la donnée en appliquant pendant un instant un niveau '1' sur la ligne SCL. Lorsque SCL revient à '0', il recommence l'opération jusqu'à ce que l'octet complet soit transmis. Il envoie alors un bit ACK à '1' tout en scrutant l'état réel de SDA. L'esclave doit alors imposer un niveau '0' pour signaler au maître que la transmission s'est effectuée correctement. Les sorties de chacun étant à collecteurs ouverts, le maître voie le '0' et peut alors passer à la suite.

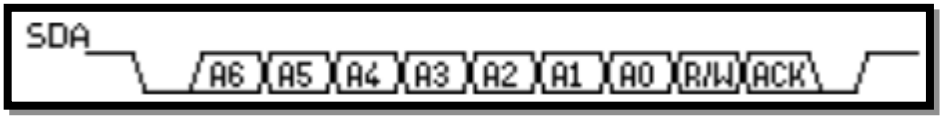


Exemple de transmission réussie.

Dans cet exemple :
SCL : Horloge imposée par le maître.
SDAM : Niveaux de SDA imposés par le maître.
SDAE : Niveaux de SDA imposés par l'esclave.
SDAR : Niveaux de SDA réels résultants.

La transmission d'une adresse

Le nombre de composants qu'il est possible de connecter sur un bus I2C étant largement supérieur à deux, il est nécessaire de définir pour chacun une adresse unique. L'adresse d'un circuit, codée sur sept bits, est défini d'une part par son type et d'autre part par l'état appliqué à un certain nombre de ces broches. Cette adresse est transmise sous la forme d'un octet au format particulier.



Exemple d'octet d'adresse.

On remarque ici que les bits D7 à D1 représentent les adresse A6 à A0, et que le bit D0 et remplacé par le bit de R/W qui permet au maître de signaler s'il veut lire ou écrire une donnée. Le bit d'acquittement ACK fonctionne comme pour une donnée, ceci permet au maître de vérifier si l'esclave est disponible.

Dernière itération :

1 Présentation du travail effectué

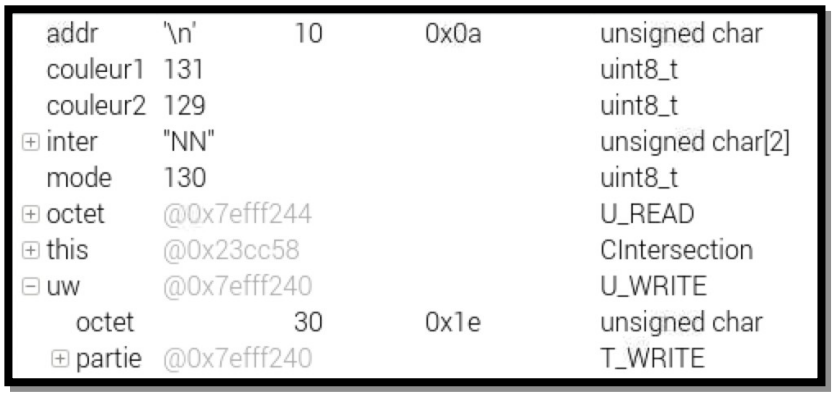
1.1 Partie Raspberry

Durant cette troisième et dernière itération, j'ai pu effectuer des tests avec une Arduino Uno reliée à un level shifter connectée en hat sur la Raspberry. De plus, avec mon coéquipier nous avons pu mettre en place l'intégration de nos deux parties. Cette intégration a permis de faire en sorte que son serveur puisse avoir accès à toutes les méthodes disponibles dans CZdc lui permettant de pouvoir écrire des ordres à effectuer sur les différentes parties. Nous avons travaillé sur la compatibilité de nos deux parties et voir à effectuer les corrections qui étaient nécessaires.

Concernant mes tests, mon travail a été de tout simplement effectuer des demandes I2C à l'Arduino tant en écriture qu'en lecture. Pour effectuer des tests j'ai utilisé les 3 classes qui peuvent recevoir des ordres (CEcran a bénéficié d'un traitement particulier, mentionné plus bas), en modifiant les valeurs des ordres afin de correspondre à ce que la classe en question pourrait attendre et voir comment la classe traite la valeur et comment elle envoie la trame d'écriture de l'ordre en fonction.

```
37      //WRITE
38      uint8_t mode = _zdc->getModeVoies();
39      uint8_t couleur1 = _zdc->getOrdresFeu1();
40      uint8_t couleur2 = _zdc->getOrdresFeu2();
41      mode |= ACK; // Version de test
42      couleur1 |= ACK; //Test
43      couleur2 |= ACK; //Test
44      U_WRITE uw;
45      uw.octet = 0;
46
47      if(mode != MODE_MANUEL){
48          uw.partie.bitMode = mode - ACK;
49          _i2c->ecrire(addr, &uw.octet, 1);
50          _zdc->setModeVoies(uw.octet);
51      }//IF mode =/ manuel
52
53      else{
54          uw.partie.bitMode = mode - ACK;
55          if(couleur1 >= ACK){
56              uw.partie.bitCouleurs1 = couleur1 - ACK;
57          }//IF changement couleur voie1
58          if(couleur2 > ACK){
59              uw.partie.bitCouleurs2 = couleur2 - ACK;
60          }//IF changement couleur voie2
61          _i2c->ecrire(addr, &uw.octet, 1);
62          _zdc->setModeVoies(uw.octet - uw.partie.bitCouleurs2);
63          _zdc->setOrdresFeu1(uw.octet - (uw.partie.bitMode + uw.partie.bitCouleurs2));
64          _zdc->setOrdresFeu2(uw.octet - (uw.partie.bitMode + uw.partie.bitCouleurs1));
65      }//IF Mode manuel
```

Portion de code présente dans CIntersection.cpp



addr	'\n'	10	0x0a	unsigned char
couleur1	131			uint8_t
couleur2	129			uint8_t
⊞ inter	"NN"			unsigned char[2]
mode	130			uint8_t
⊞ octet	@0x7efff244			U_READ
⊞ this	@0x23cc58			CIntersection
⊟ uw	@0x7efff240			U_WRITE
octet		30	0x1e	unsigned char
⊞ partie	@0x7efff240			T_WRITE

Aperçu des valeurs stockées localement dans la méthode onInter() dans CIntersection.cpp

La valeur de l'attribut « couleur1 », « couleur2 » et « mode » est supérieure à 128, ce qui est l'indicateur à la classe qu'un ordre a été envoyé. En l'occurrence ici, il y a 3 ordres qui ont été envoyés : le premier qui sera pris en compte sera l'ordre du changement de mode, étant donné qu'il définit si oui ou non les valeurs de « couleur1 » et « couleur2 » seront prises en compte par ma classe. Si « mode » est égal à 130, alors ma classe considère que le mode est défini par un ordre client comme étant le mode manuel. Une fois cette vérification effectuée, la classe s'intéresse aux valeurs de « couleur1 » et « couleur2 » à qui elle va effectuer le même traitement, étant d'enlever 128, correspondant à l'ACK de l'ordre. Chaque partie de la trame d'écriture est alors implémentée après avoir retiré l'ACK de l'ordre. Étant donné que le mode est en manuel, alors la partie mode de la trame (uw.partie.bitMode) est définie sur 130 – 128 = 2. Pour ce qui est des couleurs des voies, nous avons uw.partie.bitCouleurs1 défini sur 131 – 128 = 3 (la couleur est définie sur rouge sur la voie 1) ; et uw.partie.bitCouleurs2 défini sur 129 – 128 = 1 (la couleur est définie sur vert sur la voie 2).

Dans l'octet de la trame d'écriture, le bit 0 et 1 sont les bits contenant l'ordre du mode qui donne en binaire « 10 » dans notre exemple. Le bit 2 et 3 sont les bits contenant l'ordre de la couleur sur la voie 1 qui donne en binaire « 11 ». Le bit 4 et 5 sont les bits contenant l'ordre de la couleur sur la voie 2 qui donne en binaire « 01 ». L'octet de la trame est donc « 0001 1110 » ce qui donne 30 ou 0x1E. Ce qui correspond donc à ce qui est stocké dans « octet ».

1.1.1 Cas particulier : CEcran.

Pour le cas de CEcran, les tests ont été rapides et ont permis à mon coéquipier de pouvoir modifier les données qui seraient écrites sur l'écran en définissant le texte qui serait contenu dans les variables de l'écran dans CZdc.

1.2 Partie Arduino

La partie Arduino a simplement consisté à faire un programme permettant de lire et écrire sur le bus I2C en tant qu'esclave. L'Arduino était reliée sur les broches GND, A4 (SDA) et A5 (SCL) afin de permettre la communication avec la Raspberry. La Raspberry et l'Arduino avait un level shifter entre les deux afin de pouvoir transformer le signal envoyé par la Raspberry en 3,3V en 5V pour l'Arduino.
Concernant le programme, j'ai pensé à faire un programme simple, consistant à stocker des valeurs prédéfinies à envoyer à la Raspberry quand elle demandera des informations et de recevoir les valeurs de la Raspberry afin de traiter la valeur (ici le traitement se limite à un simple « Serial.print » permettant d'afficher sur le moniteur série des données.

Partie de SLAVE_I2C_RW_INTER.ino

```
#include <Wire.h>

byte i2c_rcv;                // data received from I2C bus
byte i2c_snd1;               // data sent on I2C bus (voie 1)
byte i2c_snd2;               // data sent on I2C bus (voie 2)
unsigned long time_start;    // start time in mSec
int stat_LED;                // status of LED: 1 = ON, 0 = OFF

void setup(){
  Wire.begin(0x0A);           // join I2C bus as Slave (addr = 10)

  // event handler initializations
  Wire.onReceive(dataRcv);    // register an event handler for received data
  Wire.onRequest(dataRqst);   // register an event handler for data requests
  Serial.begin(9600);
  // initialize global variables
  i2c_snd1 = 78;
  i2c_snd2 = 78;
  time_start = millis();
  stat_LED = 0;

  pinMode(13, OUTPUT);        // set pin 13 mode to output
}

void loop(){
  // blink logic code
        if((millis() - time_start) > (1000 * (float)(i2c_rcv/255))) {
    stat_LED = !stat_LED;
    time_start = millis();
  }
  digitalWrite(13, stat_LED);
}

//received data handler function
void dataRcv(int numBytes){
  while(Wire.available()) { // read all bytes received
    i2c_rcv = Wire.read();
    Serial.println(i2c_rcv);
    switch(i2c_rcv){
      case 0 : Serial.println("Orange clignotant"); break;
      case 1 : Serial.println("Mode auto"); break;
      case 2 : Serial.println("Manuel avec v1 sur E et v2 sur E"); break;
      case 6 : Serial.println("Manuel avec v1 sur V et v2 sur E"); break;
      case 10 : Serial.println("Manuel avec v1 sur O et v2 sur E"); break;
      case 14 : Serial.println("Manuel avec v1 sur R et v2 sur E"); break;
      case 18 : Serial.println("Manuel avec v1 sur E et v2 sur V"); break;
      case 22 : Serial.println("Manuel avec v1 sur V et v2 sur V"); break;
      case 26 : Serial.println("Manuel avec v1 sur O et v2 sur V"); break;
      case 30 : Serial.println("Manuel avec v1 sur R et v2 sur V"); break;
      case 34 : Serial.println("Manuel avec v1 sur E et v2 sur O"); break;
      case 38 : Serial.println("Manuel avec v1 sur V et v2 sur O"); break;
      case 42 : Serial.println("Manuel avec v1 sur O et v2 sur O"); break;
      case 46 : Serial.println("Manuel avec v1 sur R et v2 sur O"); break;
      case 50 : Serial.println("Manuel avec v1 sur E et v2 sur R"); break;
```

La communication s'effectue grâce à la librairie « Wire.h ».

Dans un premier temps, il est important de définir l'adresse avec « Wire.begin(0X0A) » soit à l'adresse 10, adresse I2C de la partie Intersection. La méthode « Wire.onReceive(dataRcv) » permet de traiter dans « dataRcv » les données reçues sur le bus I2C. La méthode « Wire.onRequest(dataRqst) » permet à « dataRqst » de gérer les requêtes de données sur le bus I2C.

Dans « dataRcv », l'objectif est de stocker dans une variable appelée « i2c_rcv » la trame reçue tant que des octets sont disponibles et de la traiter au travers d'un simple « switch ».

Dans notre exemple, le trame a une valeur de 30. Donc « i2c_rcv » = 30 et l'Arduino écrira la valeur de « i2c_rcv » ainsi qu'un petit texte donnant le cas provoqué par notre ordre.

Moniteur série

1.3 Partie physique

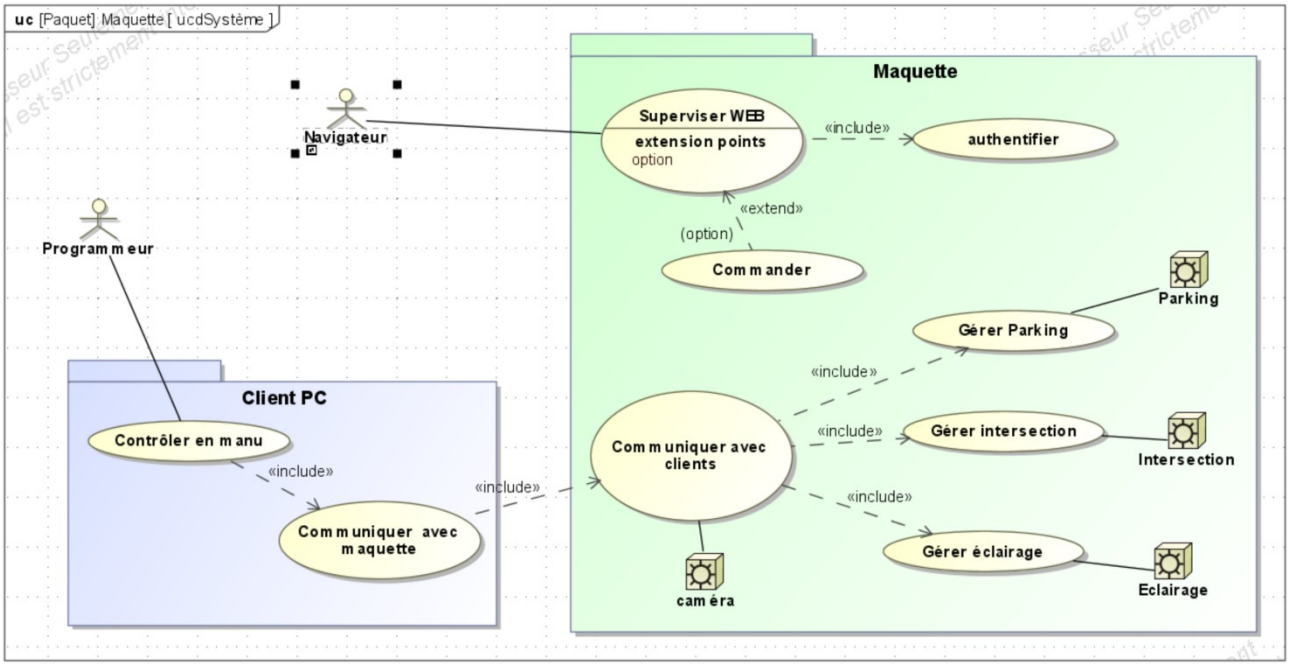
Afin d'avoir une dernière preuve que mon programme était bien fonctionnel, j'ai décidé de tester avec un analyseur de trame (Analog Discovery 2), si la trame était bien envoyée comme il faut. Sans surprise, la trame envoyée est bien correspondante à la trame reçue sur l'Arduino.

Partie Étudiant IR2 – Équipe 01 : RE Sébastien

Objectifs

Étudiant 1 **IR 21**	*Liste des tâches assurées par l'étudiant* 1. Développement partiel du logiciel RASPBERRY. Classes à coder : 1.1. CServeurTcp 1.2. CGestionClient 1.3. CModbusTcp 1.4. CIhm	**Installation** : Système d'exploitation **Raspbian**, EDI. **Mise en œuvre :** ModBus Thread, TCP, communication réseau serveur. **Configuration** : EDI, Réseau **Réalisation** : Logiciel Qt C++ avec Qt Creator **Documentation** : Manuel d'installation et mise en œuvre.

Pour résumer :



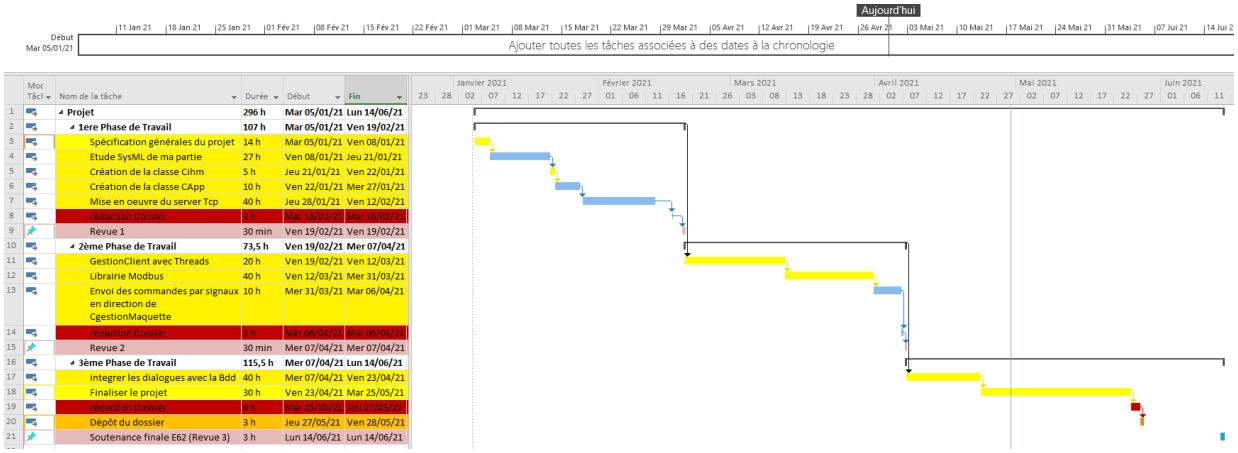
- Création d'une interface homme machine afin de pouvoir consulter la situation.
- Permettre la connexion entre le serveur et le client par le réseau.
- Gestion des clients de manière indépendante.
- Création d'un protocole de communication.
- Transférer les ordres à la maquette.

2. Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation représente le point de vu du client et les différentes actions qu'il peut effectuer. En effet l'étudiant du CERI pourra contrôler en manuel la ville comme il le souhaite, plus précisément le parking, l'intersection et l'éclairage, il pourra également consulter les statuts de chaqu'un d'eux, ces donnés seront également stockés dans une de base donnée à intervalle de temps régulier. Pour chaque requête client, mettre à jour la Zdc, répondre au client.

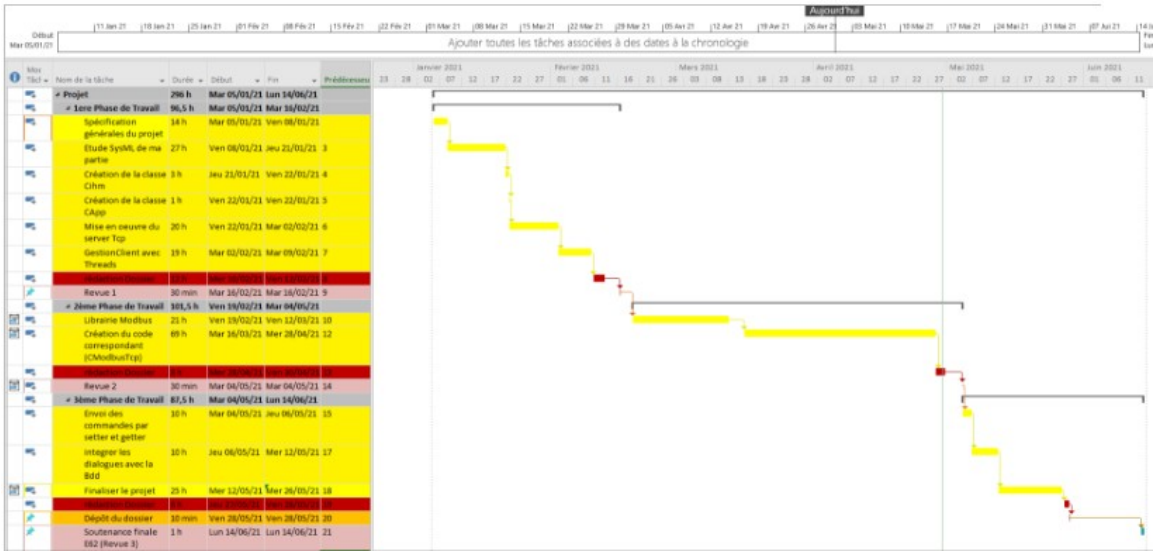
3. Diagrammes de Gantt

Prévisionnel :



	Nom de la tâche	Durée	Début	Fin
1	Projet	296 h	Mar 05/01/21	Lun 14/06/21
2	1ere Phase de Travail	107 h	Mar 05/01/21	Ven 19/02/21
3	Spécification générales du projet	14 h	Mar 05/01/21	Ven 08/01/21
4	Etude SysML de ma partie	27 h	Ven 08/01/21	Jeu 21/01/21
5	Création de la classe Cihm	5 h	Jeu 21/01/21	Ven 22/01/21
6	Création de la classe CApp	10 h	Ven 22/01/21	Mer 27/01/21
7	Mise en oeuvre du server Tcp	40 h	Jeu 28/01/21	Ven 12/02/21
8	rédaction Dossier	3 h	Mar 16/02/21	Mar 16/02/21
9	Revue 1	30 min	Ven 19/02/21	Ven 19/02/21
10	2ème Phase de Travail	73,5 h	Ven 19/02/21	Mer 07/04/21
11	GestionClient avec Threads	20 h	Ven 19/02/21	Ven 12/03/21
12	Librairie Modbus	40 h	Ven 12/03/21	Mer 31/03/21
13	Envoi des commandes par signaux en direction de CgestionMaquette	10 h	Mer 31/03/21	Mar 06/04/21
14	rédaction Dossier	3 h	Mar 06/04/21	Mar 06/04/21
15	Revue 2	30 min	Mer 07/04/21	Mer 07/04/21
16	3ème Phase de Travail	115,5 h	Mer 07/04/21	Lun 14/06/21
17	integrer les dialogues avec la Bdd	40 h	Mer 07/04/21	Ven 23/04/21
18	Finaliser le projet	30 h	Ven 23/04/21	Mar 25/05/21
19	rédaction Dossier	6 h	Mar 25/05/21	Jeu 27/05/21
20	Dépôt du dossier	3 h	Jeu 27/05/21	Ven 28/05/21
21	Soutenance finale E62 (Revue 3)	3 h	Lun 14/06/21	Lun 14/06/21

Réel :

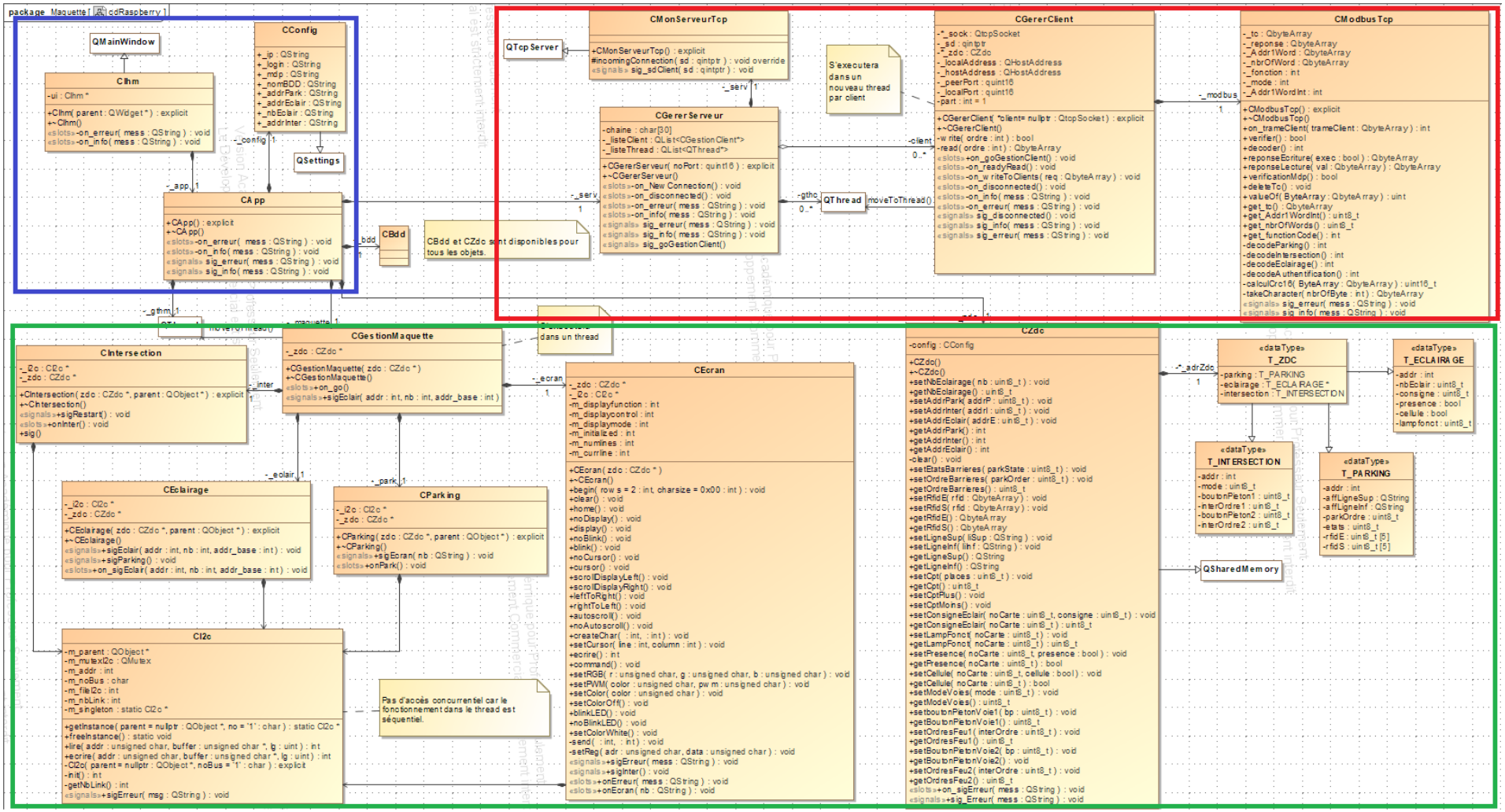


Grâce à ces diagrammes de Gantt, je peux voir là où j'ai mal estimé mon temps de travail pour chacune des taches.

Pour la phase 1, en réalité la mise en œuvre du serveur TCP a pris peu de temps, j'ai donc intégré la partie Gestion Client avec Thread, à partir de là j'ai pris beaucoup de temps à faire fonctionner les deux de la bonne manière.
Dans la 2nd phase de travail, tout s'est décalé, j'ai pris plus de temps que ce que j'aurais pensé pour faire la librairie ainsi que son code.

Dans la 3ème phase de travail, plusieurs tâches se sont ajoutées comme l'explication du protocole aux personnes chargés de faire les clients TCP. cette tache m'a également permis de corriger mes erreurs car il y en avais quelques unes. L'envoie des commandes par accesseur et mutateur correspond à la connexion entre la zone de donnée commune et mon travail. Par la suite, j'ai pu effectuer quelques tests afin de corriger quelques erreurs liées à cette intégration. L'intégration des dialogues avec la Bdd (base de données) est en attente car j'attends l'IR chargé de ce travail.
Pour la finalisation du projet, il restera à faire des tests accompagnés de la maquette. Cette phase permettra de corriger très certainement quelques erreurs.
Au départ, je n'avais aucune idée du temps que chaque tâche allait prendre donc j'avais évalué approximativement. Le confinement a chamboulé le final notamment avec le changement des vacances d'avril.

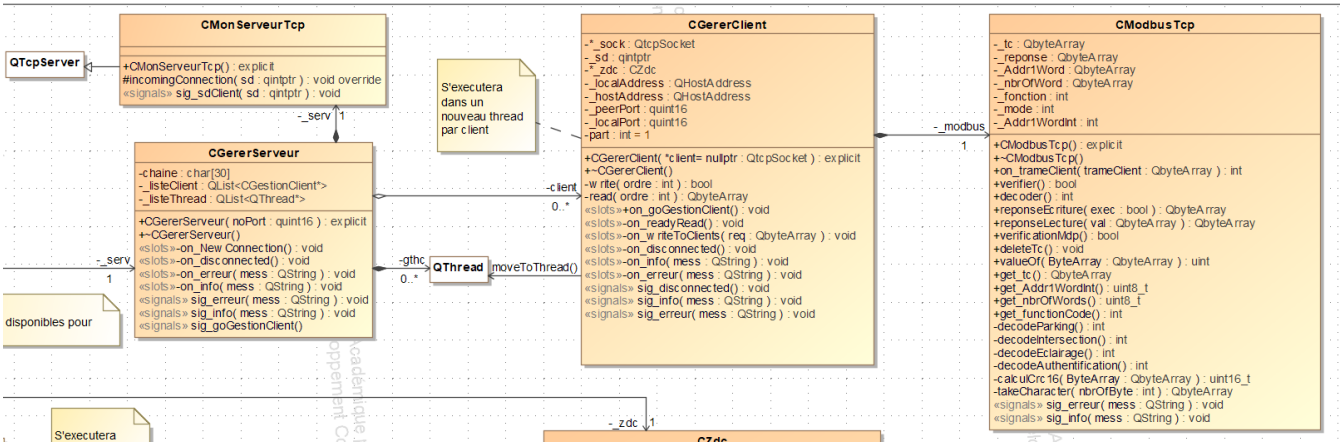
4. Diagramme de classe



Partie encadrée bleu, La classe CIhm contient l'Interface Homme Machine et toutes les interactions liées à l'affichage.
La classe CIhm contient l'Interface Homme Machine et toutes les interactions liées à l'affichage.

La classe Capp créé l'Object CGestionMaquette, CZdc, CBdd et le CGererServeur
La classe CConfig contient des commandes permettant de lire la configuration initiale du système présent sur un fichier .ini

Voici un zoom sur ma partie encadrée en rouge :

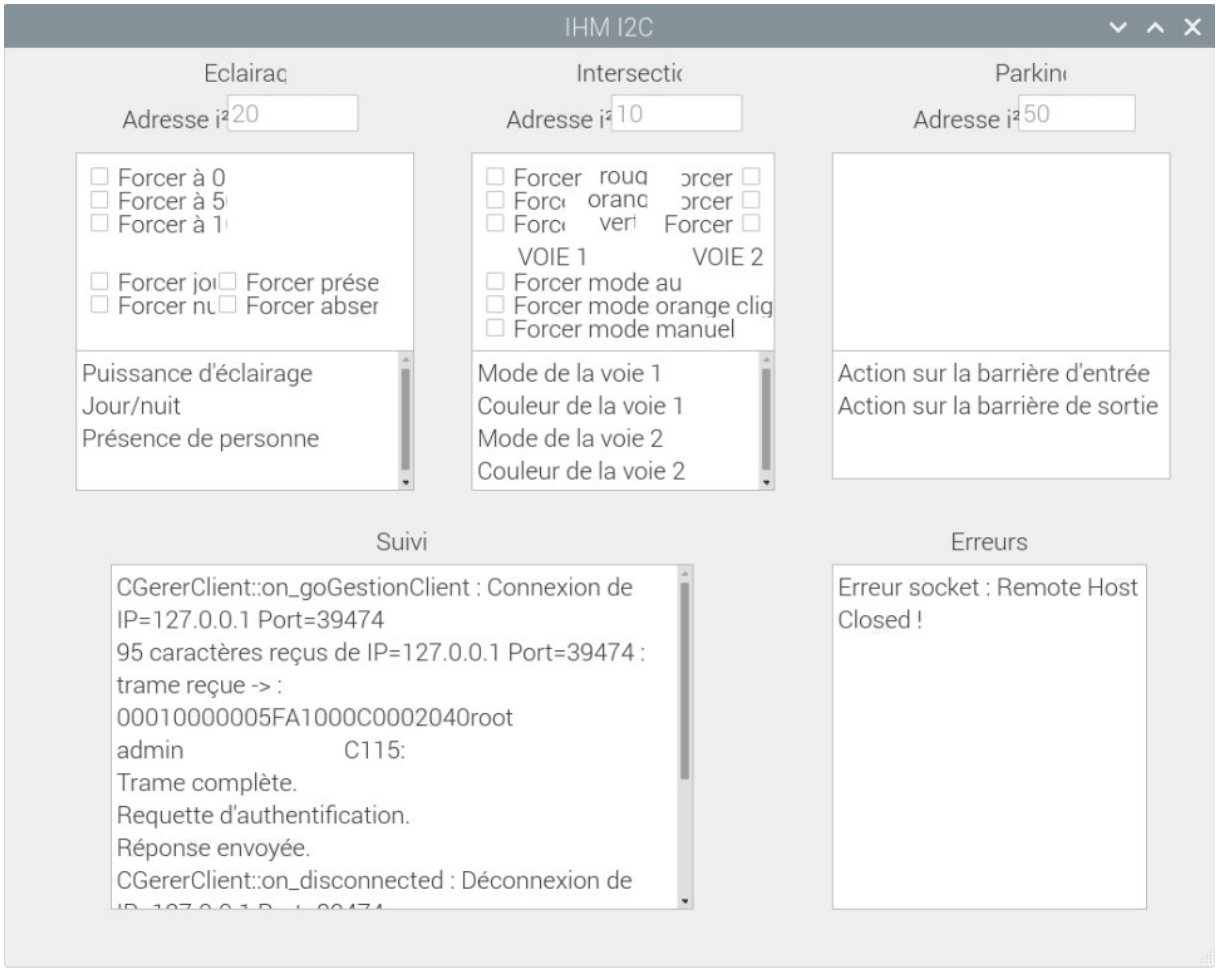


La classe CGererServer gère les connections et déconnections des clients ainsi que leurs instanciations
La Classe CMonServeurTcp est une classe qui hérite de QTcpServer, c'est-à-dire qu'il hérite de ses méthodes. Dans cette classe on override c'est à dire on réécrit la méthode incommingConnection () déjà faite dans QTcpServer. Celle-ci se déclenche à la connexion d'un nouveau client et envoie le descripteur du socket. Celui-ci permettra à l'objet CGererClient d'obtenir les adresses IP et les Ports que l'on pourra afficher dans l'Interface Homme Machine.
La classe CGererClient permet de gérer les échanges fais avec les clients. C'est une classe permettant la communication entre les clients et la Zone de donnée commune, ce qui nous permet d'envoyer des ordres et de recevoir des informations

La classe CModbusTcp permettra de décoder les trames ModBus envoyées par le client ainsi que d'en créer pour lui répondre.

5. Interface homme machine :

Dans notre cas elle permettra de visualiser les différentes actions comme la connexion/déconnection d'un/plusieurs clients, leurs informations, les caractères reçus, envoyés et les erreurs.



C'est une interface simple dite de débogage. Cette interface a changé notamment avec l'intégration des programmes pour n'en faire qu'un. Et donc c'est pour cela que l'on voit également la partie de l'IR1 Ici on peut visualiser une connexion d'un client qui souhaite s'authentifier.

6. Scénarios des cas d'utilisation

Sous système : Maquette	**UC : Authentifier**
Scénario nominal	
Demande une authentification nom+mot de passe. Un seul compte sera nécessaire.	
Sous système : Maquette	**UC : Communiquer avec clients**
Scénario nominal	
Serveur TCP gérant les clients sous forme de threads. Pour chaque requête d'un client, mettre à jour la zone de données commune et envoyer un signal à l'objet concerné. Répondre au client. Cas particulier des caméras qui seront directement accessibles par leur adresse IP et numéro de port. Le flux vidéo est généré par chaque caméra.	

7. Serveur de Communication Serveur-Client

7.1 Le Serveur Tcp

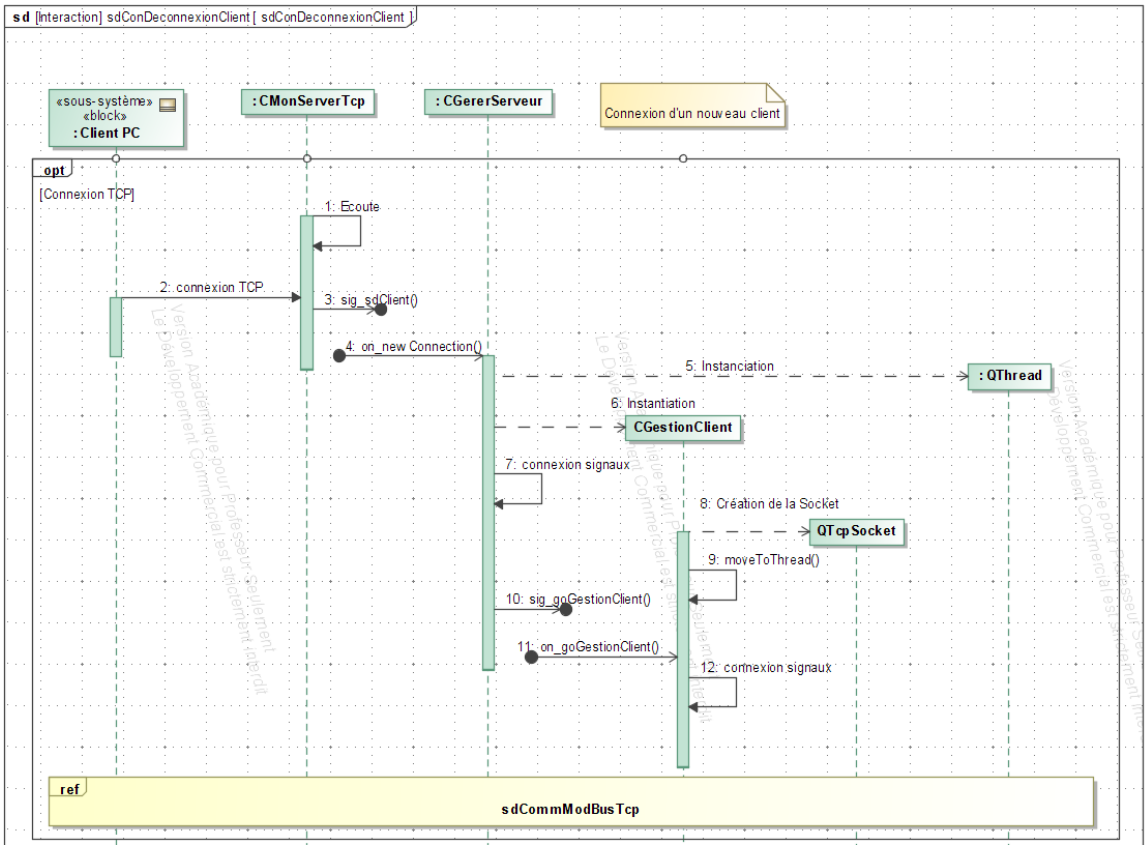
7.1.1 Création du serveur

On a pus voir que Capp créais le serveur, à sa instanciation, on atterrit dans le constructeur c'est dire la méthode qui possède le même nom que la classe, la voici :

```
CGererServeur::CGererServeur(quint16 noPort, QObject *parent) : QObject(parent)
{
    m_noPort = noPort;
    _serv = new CMonServeurTcp();
    connect(_serv, &CMonServeurTcp::sig_sdClient, this, &CGererServeur::on_newConnection);
    _serv->listen(QHostAddress::AnyIPv4, m_noPort);
}
```

On récupère ce qui a été donné en argument (noPort) qui correspond au futur d'écoute de notre serveur TCP, ensuite on instancie la classe CMonServeurTcp puis on connecte son signal à notre slot on_newConnection (pour le détail sur ce que fais CMonServeurTcp se référer au diagramme de classe). Puis on lance la méthode listen (), celle-ci lance l'écoute sur le port que l'on donne dans la variable m_noPort.

7.1.2 Diagramme de séquence connexion client



Ce diagramme de séquence décrit la réaction de l'application à la connexion d'un client ainsi que son instanciation dans la classe CGererClient qui gérera le Client indépendamment dans un Thread. Pour chaque nouveau client cette opération se reproduira.

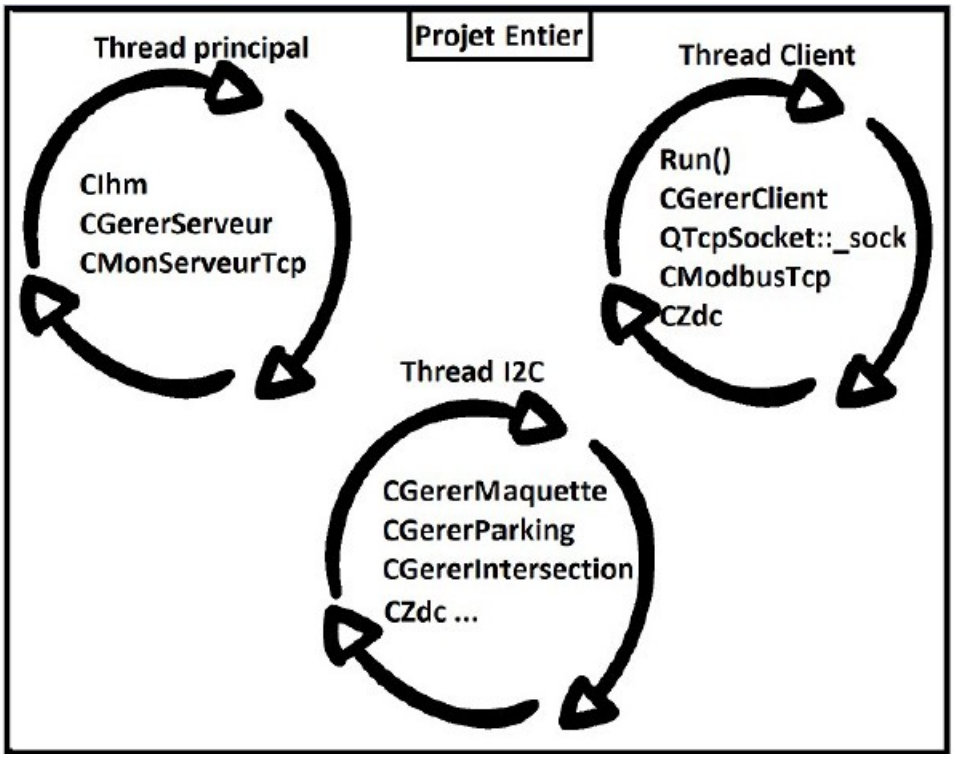
7.1.3 Code de on_newConnection ()

Voici le code de la méthode qui reproduit ce principe :

```
void CGererServeur::on_newConnection(qintptr sd)
{
    // Nouvelle connexion d'un client
    // La méthode héritée QTcpServer::incomingConnection() a été redéfinie.

    QThread *gthc = new QThread();  // création du thread
    gthc->setObjectName("servGcl_"+QString::number(sd));    //changer son nom pour le reconnaitre dans pstree
    CGererClient *client = new CGererClient(sd, nullptr);  // le client créera la socket de comm grace au descripteur (sd)

    client->moveToThread(gthc);// déplacement vers le thread

    //init signaux affichages
    connect(client, &CGererClient::sig_info, this, &CGererServeur::on_info);
    connect(client, &CGererClient::sig_erreur, this, &CGererServeur::on_erreur);

    connect(client, &CGererClient::sig_disconnected, this, &CGererServeur::on_disconnected); // provoque la destruction du client et du thre
    connect(this, &CGererServeur::sig_goGestionClient, client, &CGererClient::on_goGestionClient); // provoque création socket client
    connect(gthc, &QThread::finished, client, &QObject::deleteLater); // L'objet qui gere le client se fais détruire si son thread se fini

    gthc->start(); // départ boucle des événements du thread.
    emit sig_goGestionClient(); // Départ de la gestion du client

    // Mémorisation des objets créés dans des listes
    _listeClient.append(client);
    _listeThread.append(gthc);
}
```

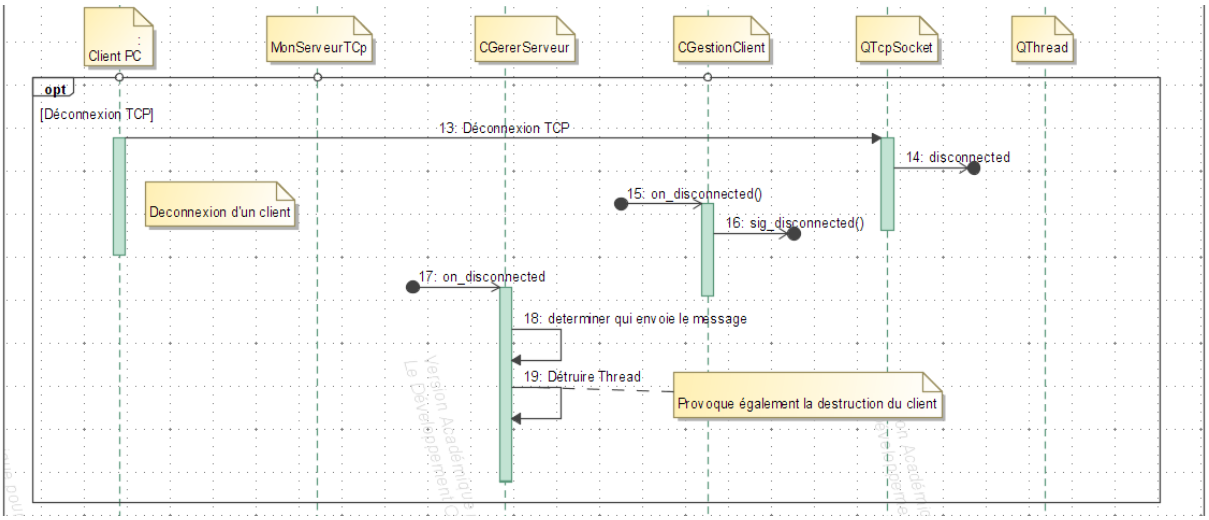
7.1.4 Threads

Un Threads s'agit d'un bloc de code au sein d'une application dont l'exécution est parallèle au thread principal que représente le plus souvent l'IHM de l'application.
Il est ainsi possible d'augmenter la vitesse d'exécution d'un programme en parallélisant les calculs.
Mais il faut faire attention car les threads ne doivent pas entrer en collision, c'est pour cela qu'il faut utiliser des mutex qui ont pour rôle de verrouiller des parties, c'est ce qui est utilisé dans la zone de donnée commune.
Ce qui est bien pratique, c'est qu'il ne nécessite pas l'allocation de segment mémoire comme pour la création d'un processus classique. Sa création est donc plus rapide, mais il est dépendant du processus principal (thread principal). Dans un schéma mêlé à notre projet, cela donnerai ceci :
Dans un schéma et mêlé à notre projet cela donnerai ceci :



Ainsi il y a autant de Thread que de clients. Et pour ce qui est du Thread I2C, il est gérer par l'IR1

7.1.5 Diagramme de séquence déconnexion client



A la déconnection d'un client, QTcpSocket envoie signal disconnected (), CGestionClient le capte et envoie, lui aussi un signal à CGererServeur, cette opération est nécessaire puisque le socket est crée par CGestionClient. Ainsi CGererClient se charge de savoir quel est le client qui viens de se déconnecter pour pouvoir détruire son thread ce qui entrainera la destruction de l'Object correspondant.

7.1.6 Code de on_disconnected ()

Voici le code de la méthode qui reproduit le principe :

```
void CGererServeur::on_disconnected()
{
    CGererClient *client = static_cast<CGererClient *>(sender());
    // le signal disconnected est parfois envoyé 2 fois !!! la 2ème fois, client = 0 !
    // Alors on filtre.
    if (client != nullptr) {
        QThread *gthc = client->thread(); // on récupère le thread qui a envoyé le signal
        int pos = _listeThread.indexOf(gthc); // on cherche la position
        if (pos > -1) {  // s'il existe
            _listeThread.at(pos)->quit();  // demande au thread de se terminer
            _listeThread.at(pos)->wait();  // attends la fin de la boucle événements
            // effacement de la liste des objets dépendant du client
            // client s'effacera automatiquement avec le thread.
            _listeThread.removeAt(pos);
            _listeClient.removeAt(pos);//on enleve l'addresse des objects des listes qui les stockait
        } // if pos
        emit sig_info("CGererServeur::on_disconnected : Nb client restant : "+QString::number(_listeClient.size()));
    } // if 0
}
```

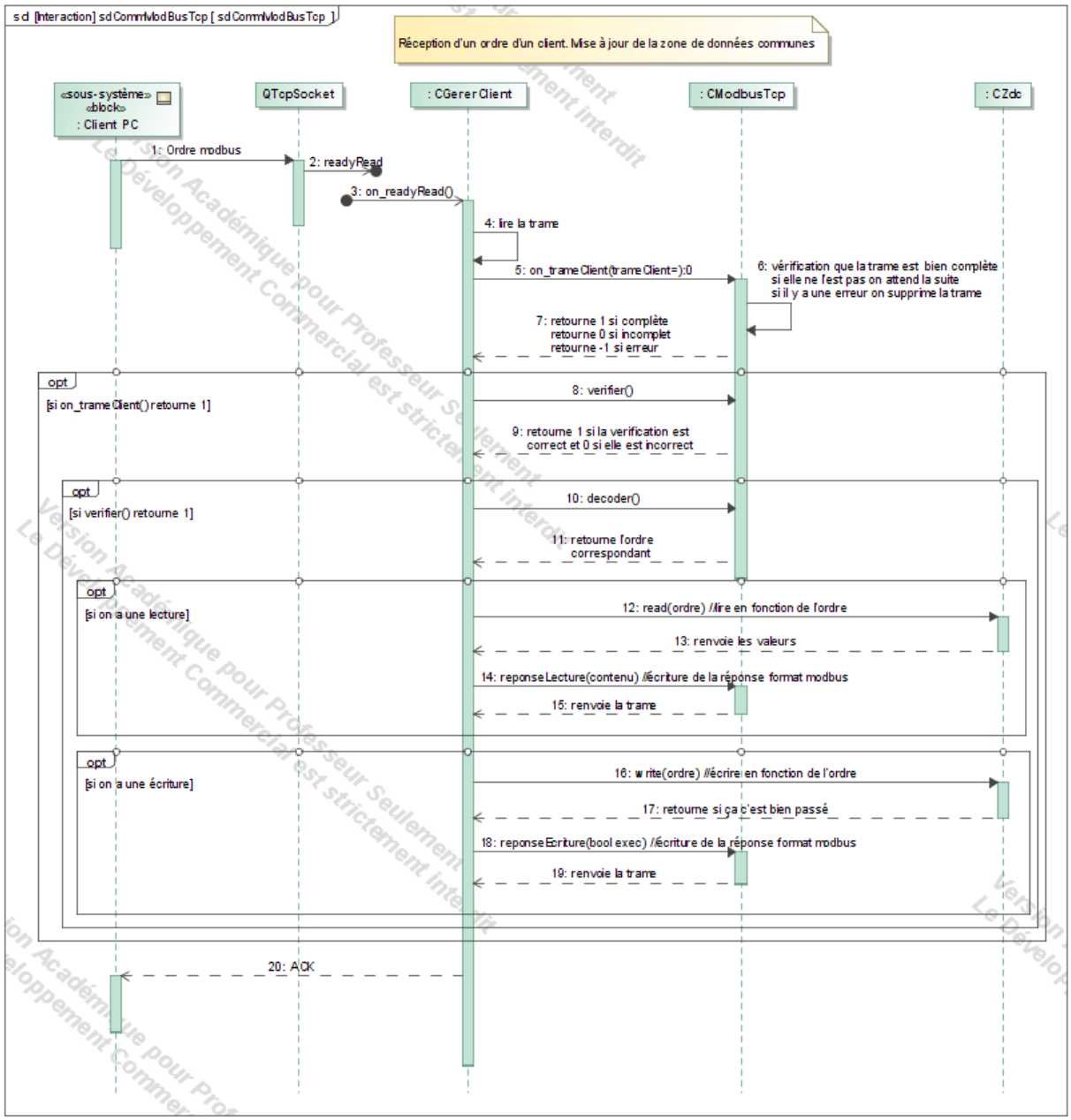
7.2 La gestion des clients

7.2.1 Initialisation de la situation

Nous avons vu dans le serveur Tcp qu'à la connexion d'un client on instancie un Object CGererClient, le socket sera crée par la suite à l'appel du signal sig_goGestionClient permettant ensuite au programme d'initialiser les connections grâce au descripteur obtenu par le constructeur de la même manière que pour noPort dans CGererServeur.

```
void CGererClient::on_goGestionClient()
{
        _sock = new QTcpSocket();  // la socket est gérée par le thread

        // init des params du client et stockage dans variable commune
        if (_sock->setSocketDescriptor(_sd)) {
            _hostAddress = _sock->peerAddress();//addr IP du serveur
            _localAddress = _sock->localAddress();//addr IP du client
            _peerPort = _sock->peerPort();//port du client
            _localPort = _sock->localPort();//port du serveur
            emit sig_info("CGererClient::on_goGestionClient : Connexion de IP="+_hostAddress.toString()
                          +" Port="+QString::number(_peerPort)); //affichage dans l'Ihm

            // signaux de fonctionnement de la socket
            connect(_sock, &QTcpSocket::readyRead, this, &CGererClient::on_readyRead);
            connect(_sock, &QTcpSocket::disconnected, this, &CGererClient::on_disconnected);

            connect(_sock, static_cast<void(QAbstractSocket::*)(QAbstractSocket::SocketError)>(&QAbstractSocket::error),
                  [=](QAbstractSocket::SocketError socketError) {
                    switch(socketError) {
                    case QAbstractSocket::RemoteHostClosedError:
                        emit sig_erreur("Erreur socket : Remote Host Closed !");
                        break;
                    default:
                        emit sig_erreur("Erreur socket : Erreur non précisée : "+QString::number(socketError));
                    } // sw
                  });// cette longue commande gère les erreurs de manière intelligente et
                     // l'affiche ainsi que son code d'erreur sauf si c'est une déconnection

        } // if setsocket....
}
```

7.2.2 Diagramme de séquence réception d'un message client



Explication :

Ce diagramme de séquence décrit l'action de l'application à la réception d'un message, avec plusieurs étapes, tout d'abord le client envoie une trame, on la lis et on vérifie sa forme, si elle est mauvaise on la détruit, si elle est incomplète, on attend la suite. Et si elle est complète, on peut passer à l'étape suivante.

Ensuite on vérifie son contenue en regardant et comparant chacune des parties importantes et on vérifie également le CRC16 qui est une partie de la trame qui permet grâce à un calcul fais sur l'ensemble de la trame avant et après réception de voir si il y a eu des erreurs et défauts lié à la transmission des donnés. Si la trame est incorrecte, on la supprime, si elle est correcte on passe à l'étape suivante.

Ensuite on décode la trame pour en extraire le sens, en bref on veut savoir ce qui nous est demandé.

- Si on est en lecture, on lit les valeurs sur la zone de donnés commune et on en fait une trame de réponse.

- Si on est en écriture, on écrit les valeurs sur la zone de donnés commune et on regarde si ça à bien été appliqué ce qui nous permet de faire une trame de réponse.

Enfin on envoie la trame au client.

7.2.3Code de on_readyRead ()

Voici le code correspondant à la réception d'un message

```
void CGererClient::on_readyRead()
{
    QTcpSocket *client = static_cast<QTcpSocket *>(sender()); //on détermine qui envoie le message
    qint64 nb = client->bytesAvailable();// aquisition du nombre d'octets reçus
    QByteArray trameClient=client->readAll();// lecture de la trame qui nous est retourné en ASCII
    //on_info("IP Local="+_localAddress.toString()+" Port="+QString::number(_localPort));//affichage le l'ip et le port du serveur
    emit sig_info(QString::number(nb)+" caractères reçus de IP="+_hostAddress.toString()+" Port="+QString::number(_peerPort)+" :"
    //affichage du nombre de d'octets reçus, de l'IP, du port de la source, et le contenu du message
    emit sig_info("trame reçue -> " + trameClient);
    //on_writeToClients("Bien reçu !\n"); // ACK (acquittement : informer à l'émmeteur de la bonne reception du message)
    int commande = _modbus->on_trameClient(trameClient);//retourne -1 si mal passé, 0 si trame non complète, 1 si c'est bon

    switch(commande){

    case -1:    //si mal passé
        emit sig_erreur("CGererClient::on_readyRead : Erreur dans le format de la trame, requette supprimée");
        //on_writeToClients("format de requette incorrecte, requete supprimée");
        _modbus->deleteTc();
        break;

    case 0:     //si trame non complète
        part += 1;
        emit sig_erreur("Trame incomplète, attente de la "+QByteArray::number(part)+"ème partie de la requette.");
        //on_writeToClients("attente de la "+QByteArray::number(part)+"ème partie de la requette");
        break;

    case 1:     //trame complète
        part = 1;
        emit sig_info("Trame complète.");
        bool verifier =_modbus->verifier();
        if (!verifier){
            emit sig_erreur("CGererClient::on_readyRead : erreur verification");
            //on_writeToClients("Vérification de la trame échouée");
            _modbus->deleteTc();


            break;
        }//if(!verifier)
        //si le MBAP header + CRC sont bon
        int ordre = _modbus->decoder();
        QByteArray reponse;
        QByteArray valeursMots;
        switch (_modbus->get_functionCode()) {
        case 1://Lecture
            valeursMots = read(ordre);//retourne un QbyteArray de la partie data d'un réponse à une lecture
            reponse = _modbus->reponseLecture(valeursMots);//retourne la trame à envoyer au client
            break;
        case 2://Ecriture
            bool exec = write(ordre);//retourne si ça s'est bien executé
            reponse = _modbus->reponseEcriture(exec);//retourne la trame à envoyer au client
            break;
        }
        on_writeToClients(reponse);
        _modbus->deleteTc();
        emit sig_info("Réponse envoyée.");
        break;

    }//sw
}
```

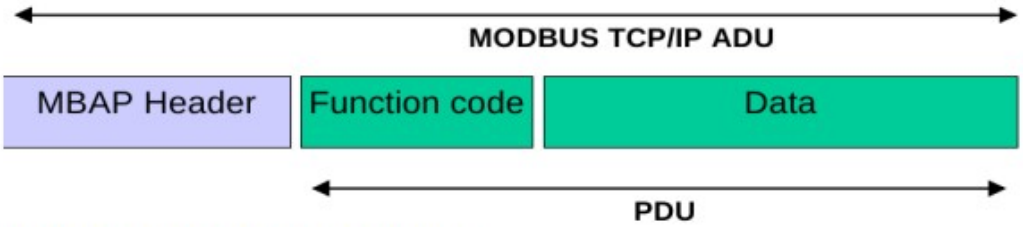
7.3 ModbusTcp/IP Protocol

7.3.1 Introduction

Le protocole ModBus a pour principe fondamental que le maitre initie toujours la communication et l'esclave a toujours le devoir de répondre simplement, l'esclave ne prend jamais d'initiative, il ne fait que répondre.

7.3.2 Forme des trames

La forme d'une trame ModBus est la suivante :



Le contenu du *MBAP Header* est le suivant :

Fields	Length	Description -	Client	Server
Transaction Identifier	2 Bytes	Identification of a MODBUS Request / Response transaction.	Initialized by the client	Recopied by the server from the received request
Protocol Identifier	2 Bytes	0 = MODBUS protocol	Initialized by the client	Recopied by the server from the received request
Length	2 Bytes	Number of following bytes	Initialized by the client (request)	Initialized by the server (Response)
Unit Identifier	1 Byte	Identification of a remote slave connected on a serial line or on other buses.	Initialized by the client	Recopied by the server from the received request

7.3.2.1 TRANSACTION IDENTIFER = 2octets

C'est ce qui correspond à un nom/identifiant, pour nous ce sera toujours 0x0001

7.3.2.2 PROTOCOL IDENTIFIER = 2octets

On est en ModBus donc on met toujours 0x0000

7.3.2.3 LENGTH = 2octets

Taille totale de la trame (ADU+2 car il y a les ':' avant et après la trame) y compris length
exemple : une trame de taille 16octets, on transforme ça en hexa = 0x10,
Et sur 2octets donc Length = 0x0010

7.3.2.4 UNIT IDENTIFIER = 1octet

P = parking
I = intersection
E = Eclairage
A = authentification

7.3.2.5 FUNCTION CODE = 1octet

Le contenu du code fonction dépend de la nature de la demande.
Dans notre système, nous aurons besoin du code fonction 03 ou 04 en ce qu'il concerne la lecture. Et du code fonction 16 en ce qu'il concerne l'écriture. J'ai pu connaître ces valeurs en allant dans la documentation officielle du protocole ModBus.
16 pour écriture n octets = 0x10
03 pour lecture 1 octet = 0x03

7.3.2.6 DATA = X octets

La partie Data pour les demandes seront composées des adresses de premiers mots, celles si correspondent pour chacune d'entre elles à des numéros correspondant à des demandes. Le listage de ces adresses est donné plus tard. Elle est également composée des nombres de mots demandés. Pour l'écriture, en plus de cela on ajoute le nombre d'octets à forcer et de la valeur de ces mots. Pour la réponse, le serveur renvoi pour la demande de lecture, le nombre d'octet lu et les valeurs de ces octets. Pour la réponse à la demande d'écriture, le serveur renvoi l'adresse du 1er mot forcé et le nombre de mots forcés
La partie data se fini par l'ajout du CRC16 qui correspond a un calcul de l'ADU (Application Data Unit)

7.3.3 Construction des trames et exemples

Tout d'abord je spécifie que avant et après chaque trame il y a des ':' car c'est ce qu'il me permet de savoir si j'ai une trame complète ou pas, ceci ne fait pas partie de l'ADU, on peut dire qu'il «englobe l'ADU». Comme ceci :

Ensuite les trames sont envoyées avec des caractères ascii. Ce qui fait que '0x00', qui est 1octet hexadécimal, Donne sur le réseau 2 caractères '00'. Puisque 1 caractère ascii = 1octet la taille de cette trame sur le réseau = 2octets.

7.3.3.1 Fonction 3 (LECTURE)

Demande (du client):

	Transaction identifier	Protocol identifier	length	Unit identifier	Function code	Adresse 1er mot	Nombre de mots = X	CRC16
	2bytes	2bytes	2bytes	1bytes	1bytes	2bytes	2bytes	2bytes
Exemple : lecture parking	0x0001	0x0000	0x001D	Lettre type ='P'	0x03	0x00A1	0x0001 = 1_{10}	0xAAE6
Taille sur le réseau	4octets	4octets	4octets	1octets	2octets	4octets	4octets	4octets

Réponse (du serveur) :

	Transaction identifier	Protocol identifier	length	Unit identifier	Function code	Nombre octets lus	Valeur 1er mot	Valeurs X mot	Valeur dernier mot	CRC16
	2bytes	2bytes	2bytes	1bytes	1bytes	1bytes	2bytes	2bytes CHACUN	2bytes	2bytes
Exemple : Réponse lecture parking	0x 0001	0x 0000	0x0017	P	0x03	0x02	0x0000	-	-	0xEAC7
Taille sur le réseau	4octets	4octets	4octets	1octet	2octets	2octets	4octets	4 octets	4octets	4octets

Pour l'exemple cela donne : (depuis un programme simulant un client TCP)

envoie du message = :00010000001DP0300A10001AAE6:
nouveau message = :000100000017P03020000EAC7:

la valeur 1er mot = 0x0000, cela veut dire que les deux barrières ne sont ni en haut ni en bas, et pas de bouton d'urgence appuyé (détail de cette conclusion plus tard), la raison est que puisque le reste du système n'est pas encore opérationnel, on me renvoie que des valeurs nulles

7.3.3.2 Fonction 16 (ECRITURE)

Demande (du client) :

	Transaction identifier	Protocol identifier	length	Unit identifier	Function code	Adresse du 1er mot à forcer	Nombre de mots à forcer	Nombre d'octets à forcer (2 fois plus qu'avant)	Valeur des mots à forcer	CRC16
	2bytes	2bytes	2bytes	1bytes	1bytes	2bytes	2bytes	1bytes	Nbytes	2bytes
Exemple : Ecriture Intersection	0x0001	0x0000	0x0023	I	0x10	0x00AA	0x0001(16) = 1(10)	0x02 = 2_{10} = N	00B2 (voie 2 uniquement, rouge, manuel)	0xC23E
Taille sur le réseau	4octets	4octets	4octets	1octets	2octets	4octets	4octets	2octets	4 octets	4octets

Réponse (du serveur) :

	Transaction identifier	Protocol identifier	length	Unit identifier	Function code	Adresse du 1er mot forcé	Nombre de mots forcés	CRC16
	2bytes	2bytes	2bytes	1bytes	1bytes	2bytes	2bytes	2bytes
Exemple : Réponse écriture Intersection	0001	0000	001D	I	10	00AA	0001(16) = 1(10)	FF12
Taille sur le réseau	4octets	4octets	4octets	1octet	2octets	4octets	4octets	4octets

Pour l'exemple ça donne : (depuis un programme simulant un client TCP)

envoie du message = :00010000023I1000AA00010200B2C23E:
nouveau message = :00010000001DI1000AA0001FF12:

si on interprète la réponse du serveur on a 1mot qui a bien été forcé, donc notre demande a bien été prise en compte !

7.3.4 Tableau Adressage

7.3.4.1 Général

Le protocole n'utilisera que les codes fonction lecture 1 octet et écriture n octets. Une adresse est codée sur 16 bits et correspond à un octet.

7.3.4.2 Eclairage

Chaque groupe de 6 lampadaires est commandé par le micro contrôleur en PWM.
Chaque groupe de lampadaire peut être commandé indépendamment par exemple pour la carte d'éclairage n°11, l'adresse est 0x0010 et nombre de mot 1, grâce à cela on peut commander les cartes d'éclairages que l'on souhaite sans interférer sur les autres cartes. On peut même augmenter le nombre de mots pour commander un groupe de cartes d'éclairage ex : de 17 à 27, adresse 1er mot = 0x0010 nombres de mots = 11
Les demandes de lecture se font à l'adresse 0x0020 pour le premier mot avec 32 mots.

Adresse modBus (hexa)	Bit	Si 1	Si 0
0x0000 (Écriture seule)	0	Ordre allumer secteur	Ordre éteindre secteur
carte éclairage n°1	1	100% (éclairage maxi)	50% (éclairage diminué)
...			
0x001F (écriture seule)			
carte éclairage n°32			
0x0020 (Lecture seule)	0	Présence piéton	Absence piéton
carte éclairage n°1	1	Détection jour	Détection nuit
...	2	Lampadaire 1 défectueux	RAS
0x003F (Lecture seule)	3	Lampadaire 2 défectueux	RAS
carte éclairage n°32	4	Lampadaire 3 défectueux	RAS
	5	Lampadaire 4 défectueux	RAS
	6	Lampadaire 5 défectueux	RAS
	7	Lampadaire 6 défectueux	RAS

Note : Ce plan mémoire est valable pour chaque secteur d'éclairage. 64 adresses sont prévues, donc on a une capacité de 32 groupes de 6 lampadaires. (On arrive jusqu'a l'adresse 0x003F)

7.3.4.3 Parking

A l'entrée du parking se trouve une barrière, un écran indique avec couleur s'il reste des places. Il y a deux boutons poussoir pour appeler de l'aide à l'entrée et à la sortie et deux lecteurs RFID a l'entrée et à la sortie.
Chaque véhicule doit posséder une carte RFID pour entrer et sortir du parking, Le véhicule est détecté par la présence de la carte RFID et le client décide en fonction de la carte d'ouvrir aux voitures.
Pour l'affichage, les lignes supérieures et inferieures sont éditables et seront fournies par le client en une seule trame à partir de l'adresse 0x0080 (taille 16mots).

Adresse ModBus (hexa)	Désignation/Bit	1	0
0x0080 - 0x0087 (Écriture seule)	Texte affichage ligne supérieure (16 caractères)		
0x0088 - 0x008F (Écriture seule)	Texte affichage ligne inférieure (16 caractères)		
0x00A0 (Écriture seule)	bit 0	Ordre monter barrière entrée	-
	bit 1	Ordre descendre barrière entrée	-
	bit 2	Ordre monter barrière sortie	-
	bit 3	Ordre descendre barrière sortie	-
0x00A1 (lecture seule)	bit 0	Barrière montée entrée	-
	bit 1	Barrière descendue entrée	-
	bit 2	Barrière montée sortie	-
	bit 3	Barrière descendue sortie	-
	bit 4	Bouton d'appel de l'entrée appuyer	-
	Bit 5	Bouton d'appel sortie appuyer	
0x00A2 - 0x00A5 (lecture seule)	RFID Entrée du parking (RFID = 5octets => taille réelle = 3 mots de 2octets = 6octets)		
0x00A6 - 0x00A9 (Lecture seule)	RFID Sortie du parking (RFID = 5octets => taille réelle = 3 mots de 2octets = 6octets)		

7.3.4.4 Intersection

A l'intersection, deux voies se croisent, il y a possibilité pour le client de lire la quantité de boutons appuyés par voie ainsi que de modifier l'état des feux avec 3 modes : orange clignotant, auto, manuel avec possibilité d'allumer le feu vert, rouge ou orange, et d'éteindre les feux. On peut également décider de ne pas intervenir sur le contrôle d'une autre voie pour le mode manuel.

Adresse modBus (hexa)	Bit	Si 1	Si 0
0x00AA (Ecriture seule)	0	voir tableau mode	
	1	voir tableau mode	
	2	voie 1 voir tableau feux	
	3	voie 1 voir tableau feux	
	4	voie 2 voir tableau feux	
	5	voie 2 voir tableau feux	
	6	manuel voie 1 activé	Off
	7	manuel voie 2 activé	Off
0x00AB (Lecture seule)	0	1 Bouton appuyé voie 1	Off
	1	2 Boutons appuyés voie 1	Off
	2	3 Boutons appuyés voie 1	Off
	3	4 Boutons appuyés voie 1	Off
	4	1 Bouton appuyé voie 2	Off
	5	2 Boutons appuyés voie 2	Off
	6	3 Boutons appuyés voie 2	Off
	7	4 Boutons appuyés voie 2	Off

tableau mode		
bit 1	bit 0	mode
0	0	orange clignotant
0	1	auto
1	0	manuel
1	1	usage futur

tableau feux		
bit 1	bit 0	mode
0	0	éteint
0	1	vert
1	0	orange
1	1	rouge

7.3.5 Authentification

32 caractères max pour le mot de passe et l'identifiant, 64 caractères au final donc 32 mots au final

Adresse modBus (hexa)	Désignation
0x00C0 - 0x00CF (Ecriture seule)	Identifiant
0x00D0 - 0x00DF (Ecriture seule)	mot de passe

Ces informations seront fournies par le client en une seule trame à partir de l'adresse 0x00C0. Le caractère " " sera inutilisable car ils servent au programme qui décode le mot de passe et l'identifiant. L'authentification est actuellement facultative car elle n'est pas nécessaire réellement pour accéder aux commandes sur la maquette. Ceci reste tout de même modifiable et donc peut devenir obligatoire pour chaque client désirant accéder au système. Cette décision a été prise par mon professeur ayant pour objectif de rendre l'accès plus simple pour les étudiants du CERI.

7.3.5 Exemple utilisation du protocole

Pour illustrer la manière dont je décode les trames client et envoie une réponse en fonction de celuici nous allons reprendre l'exemple de l'écriture intersection. Se référer au 7.3.3.2 :

envoie du message = :000100000023I1000AA00010200B2C23E:
nouveau message = :00010000001DI1000AA0001FF12:

Nous n'allons pas expliquer ici toute la partie appelée MBAP header et fonction code car ces parties sont vérifiés dans la méthode verifier() qui s'occupe de cela. Nous allons donc nous pencher sur la partie data de cette trame soit (00AA 0001 02 00B2 C23E)

```
int CModbusTcp::decoder()
{
    //décodage de "_tc" pour traduire un ordre
    //cet odre sera affilié a un nombre et retourné par la fonction, dans on ready read

    //return 1 //Ecriture ecran
    //return 2 //Ecriture parking
    //return 3 //Lecture parking
    //return 4 //Lecture RFID
    //return 5;//Ecriture éclairage
    //return 6;//Lecture éclairage
    //return 7 //Ecriture inter
    //return 8 //Lecture inter
    //return 9;//Ecriture authentification
    switch (_mode)
    {
    case 1 ://si parking
        return decodeParking();

    case 2 ://si Eclairage
        return decodeEclairage();

    case 3 ://si intersection
        return decodeIntersection();

    case 4 ://si Authentification
        return decodeAuthentification();

    }//switch
    return 0;
}
```

Donc après avoir vérifié, on décode. Ici on a à faire à l'intersection donc grâce au caractère 'I' de la trame envoyée par le client, on le sait. Ensuite en fonction du mode, la méthode fait appel à d'autres méthodes qui vont cette fois réellement décoder.

```
int CModbusTcp::decodeIntersection()
{
    _Addr1Word = takeCharacter(4);//recupere l'adresse du premier mot
    _nbrOfWord = takeCharacter(4);//recupere le nombres de mots
    QByteArray nbrOfBytes;

    switch (_fonction) {
    case 16:

        if (_Addr1Word != "00AA"){
            qDebug() << "erreur 1er mot";
            return false;
        }//if
        if (_nbrOfWord != "0001"){
            qDebug() << "erreur nombre de mots";
            return false;
        }//if
        nbrOfBytes = takeCharacter(2);//recupere le nombre d'octets
        if (nbrOfBytes != "02"){
            qDebug() << "erreur nombre d'octets";
            return false;
        }//if
        return 7;//ecriture inter
```

Ici on décode avec les parties adresses premier mot, nombres de mots et nombres d'octets. Ceci pour en décoder une demande, et si il n'y a pas de problème, on en déduit que c'est l'écriture d'une intersection.

La méthode takeCharacter(valeur) permet de récupérer sur la trame envoyée par le client le nombre de caractères demandé. Elle retourne en type QByteArray, le soustrait à la trame pour qu'il ne reste plus que (00B2) donc la valeur de l'écriture. (Le CRC16 qui est de 'C23E' avait été enlevé au préalable afin de le vérifier la trame donc il ne figure pas dans la variable commune qui possède la trame du client '_tc')

```
QByteArray CModbusTcp::takeCharacter(int nbOfBytes)
{
    QByteArray byteArrayTaked = _tc.left(nbOfBytes); //on récupère les nbrOfBytes premiers caractère de la variable commune _tc
    _tc = _tc.right(_tc.size()-nbOfBytes); //on enleve ces caractères a la variable commune
    return byteArrayTaked;//retourne les nbOfBytes premiers caractères de _tc
}
```

Donc on revient à on_readyRead, ordre = 7, ensuite on regarde si on a une écriture ou une lecture. Ici on entre dans la méthode write() dans laquelle on donne en argument l'ordre.

```
int ordre = _modbus->decoder();
QByteArray reponse;
QByteArray valeursMots;
switch (_modbus->get_functionCode()) {
case 1://Lecture
    valeursMots = read(ordre);//retourne un QbyteArray de la partie data d'un réponse à une lecture
    reponse = _modbus->reponseLecture(valeursMots);//retourne la trame à envoyer au client
    break;
case 2://Ecriture
    bool exec = write(ordre);//retourne si ça s'est bien executé
    reponse = _modbus->reponseEcriture(exec);//retourne la trame à envoyer au client
    break;
}
on_writeToClients(reponse);
_modbus->deleteTc();
emit sig_info("Réponse envoyée.");
break;
```

Au début de la méthode write() on prépare de décodage de '00B2' :

```
bool CGererClient::write(int ordre)
{
    bool REturn = true;
    uint8_t AddrlwordInt;
    QByteArray tc = _modbus->get_tc();
    QByteArray caracterTaked;
    QString ligne;
    uint8_t nbrEclair;
    uint value[8];
    bool bit[8];

    if (ordre == 2 || ordre == 7){
        QByteArray valueWordToForce = tc;
        QString stringByteArray = QString::fromStdString(valueWordToForce.data());
        value[0] = stringByteArray.toUInt(nullptr,16);
        //permet de décomposer la partie valeur donc dans l'exemple = "00B2" en valeur entière = 178(10)
        bit[0] = value[0]%2;
        for (int i=0; i!=7; i++) {
            value[i+1] = (value[i]-bit[i])/2;
            bit[i+1] = value[i+1]%2;
        }// permet de transformer cette valeur en bits grâce a des divisions euclidiènnes
    }

    switch(ordre){

    case 7://Intersection
        if (bit[0] == 0 && bit[1] == 0) _zdc->setModeVoies(128);//orange clignotant
        if (bit[0] == 1 && bit[1] == 0) _zdc->setModeVoies(128+1);//auto
        if (bit[0] == 0 && bit[1] == 1) {//manuel
            _zdc->setModeVoies(128+2);
            if (bit[6] == 1){
                //voie 1
                if (bit[2] == 0 && bit[3] == 0) _zdc->setOrdresFeu1(128);//eteint
                if (bit[2] == 1 && bit[3] == 0) _zdc->setOrdresFeu1(128+1);//vert
                if (bit[2] == 0 && bit[3] == 1) _zdc->setOrdresFeu1(128+2);//orange
                if (bit[2] == 1 && bit[3] == 1) _zdc->setOrdresFeu1(128+3);//rouge
            }
            if (bit[7] == 1){
                //voie 2
                if (bit[4] == 0 && bit[5] == 0) _zdc->setOrdresFeu2(128);//eteint
                if (bit[4] == 1 && bit[5] == 0) _zdc->setOrdresFeu2(128+1);//vert
                if (bit[4] == 0 && bit[5] == 1) _zdc->setOrdresFeu2(128+2);//orange
                if (bit[4] == 1 && bit[5] == 1) _zdc->setOrdresFeu2(128+3);//rouge
            }

        }
        break;
```

Ici on suit le protocole pour en traduire les ordres précisément et ordonner les mutateurs des valeurs qui correspondent a ces ordres. Le 128+x mis en argument est le bit d'ACK, il permet à la ZDC de savoir si les clients ont modifiés les valeurs. Et si tout s'est bien passé write() renvoie 1.

Ensuite on demande à l'objet _modbus de créer une réponse pour le client grâce à la méthode reponseEcriture(). Celle-ci renvoie la trame de réponse qui sera par la suite envoyé au client.

```
QByteArray CModbusTcp::reponseEcriture(bool exec)
{
    _reponse ="";
    _reponse += ":";
        //MBAP header
    QByteArray data = "00010000001D";
    switch (_mode) {//Unit identifier
    case 1://si parking
        data += "P";
        break;
    case 2://si eclairage
        data += "E";
        break;
    case 3://si intersection
        data += "I";
        break;
    case 4://si authentification
        data += "A";
        break;
    }
        //fonction code
    data += "10";
        //data
    data += _Addr1Word;//adresse du premier mot forcé
    if (exec) data += _nbrOfWord;//nombre de mots forcés si tout s'est bien passé
    else data += "0000";//sinon rien
    uint16_t crc16Calc = calculCrc16(data);//calcul du CRC16
    QString crcCalcString = QString::number( crc16Calc, 16 ).toUpper();//conversion hexadécimale
    QByteArray crcCalcArray = crcCalcString.toLatin1();//convertion QByteArray

    data += crcCalcArray;//ajout du CRC16 dans la trame

    _reponse += data;
    _reponse += ":";//encapsulation dans les '::'
    return _reponse;
}
```

Evidement tout cela n'est qu'un exemple, j'ai du faire cette démarche pour chacun des éléments du protocole en le suivant à la lettre, c'était plutôt long mais satisfaisant car c'est aussi là que les premiers tests ont pu être donnés.

8. Intégration

8.1 Introduction :

Tout d'abord l'intégration est une étape très importante car elle met en communs deux travails. Dans notre projet l'intégration devais se faire avec deux parties :
- Ma partie contenant les classes permettant la mise en place du serveur TCP, la gestion des clients et le protocole ModBus.
- La partie de Florian Pichery (IR1 contenant les classes permettant la gestion en I2C de la ville, La Zone de donnée commune, et la classe visualisant le fichier de configuration)

8.2 Ma démarche :

J'ai tout d'abord regardé quelles étais les parties communes et les liens entre les classes.
Cihm et Capp sont les parties communes du projet, Capp est le centre du projet, il instancie la classe zone de donnée commune, la classe gestion maquette, la classe qui visualise le fichier de configuration et le serveur TCP. Les données contenues dans la zone de donnée commune doivent pouvoir êtres modifiés et lus par la classe de gestion des clients ce qui fais un lien en plus entre nos parties.
Donc j'ai tout d'abord ajouté les classes du serveur TCP au projet Qt de Florian car il avait plus remplis de choses que moi, après ça j'ai mis dans le constructeur du serveur TCP l'object correspondant à la classe CZdc afin de pouvoir l'utiliser dans CGererClient afin de lire et écrire des valeurs grâce aux setters et getters de la Zdc.
ensuite j'ai ajouter les signaux/slots correspondant à l'affichage des informations concernant le serveur TCP, j'ai tester et après avoir régler quelques petits détails tout fonctionnais comme avant et ce, pour les deux parties, à présent nous utilisons GitHub afin de gérer les versions de notre logiciels les parties communes étant tout particulièrement à surveiller dans la prévention des problèmes.
Enfin pour l'Ihm j'ai ajouter mon interface à celle de Florian pour n'en faire qu'une.

9 . Fiches recettes

9. 1 Lancement du programme

Nom du système :	smartcity	Code de la campagne de test :	GR1 – IR1 - 01
Technicien 1 :	Ré Sébastien	Date :	23 / 05 / 2021

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Lancement du programme.
Objectif du scénario :	Avoir un programme lancé et prêt a l'emploi.

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Avoir une raspberry Pi, avoir le programme compilé et exécutable sur la machine.

EXECUTION DU TEST

Description courte :	Brancher la raspberry, lancer l'exécutable du programme nommé Smartcity2021.
Résultats attendus :	Affichage de l'interface homme machine composée de l'interface I2C et l'affichage des donnés serveur.

BILAN

Affichage de l'interface homme machine, prête à l'emploi.

REMARQUES

RAS

CONCLUSION

VALIDE	X	NON VALIDE	

9.2 Gestion des connexions/déconnexions de plusieurs clients

Nom du système :	smartcity	Code de la campagne de test :	GR1 – IR1 - 02
Technicien 1 :	Ré Sébastien	Date :	23 / 05 / 2021

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Gestion des connexions/déconnexions de plusieurs clients
Objectif du scénario :	Ce scénario va permettre de démontrer que le programme est capable de gérer plusieurs clients à la fois avec un exemple de deux clients connectés simultanément.

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Avoir lancé au préalable le programme SmartCity2021, préparer la connexion des clients notamment grâce à Telnet ou http.
Lancer un navigateur internet et un invite de commande comme le LXTerminal de Raspbian et avoir télécharger le paquet permettant l'envoie de requêtes Telnet.

EXECUTION DU TEST

Description courte :	Ecrire dans la barre d'URL du navigateur l'adresse IP de la machine ' :' le port, ce qui donne : «http://127.0.0.1:2222». Pour le terminal LXTerminal de Raspbian, taper «telnet 127.0.0.1 2222». fermer le navigateur, fermer le terminal
Résultats attendus :	constater la connexion sur l'interface homme machine après l'opération sur le navigateur. constater la connexion sur l'interface homme machine après l'opération avec la commande Telnet. constater leurs déconnexions sur l'interface homme machine après les fermetures.

BILAN

Tout fonctionne.

REMARQUES

La connexion http se suis par la requête http du protocole http donc on peut en déduire quel est quel client facilement.

CONCLUSION

VALIDE	X	NON VALIDE	

9.3 Communication utilisant le protocole ModBus lié au projet

Nom du système :	smartcity	Code de la campagne de test :	GR1 – IR1 - 03
Technicien 1 :	Ré Sébastien	Date :	23 / 05 / 2021

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Communication utilisant le protocole ModBus lié au projet
Objectif du scénario :	Montrer que le système utilise le protocole ModBus de manière opérationnelle (effective)

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

Avoir lancé au préalable le programme SmartCity2021, Créer une trame à partir du protocole, n'importe laquelle fera l'affaire, exemple : « :00010000003FP100080001020SmartCity2021 CERI, MasterCMI 3CCB: » cette trame correspond à la demande d'écriture sur l'écran du parking.

EXECUTION DU TEST

Description courte :	Envoie de la trame « :00010000003FP100080001020SmartCity2021 CERI, MasterCMI 3CCB: » au serveur SmartCity.
Résultats attendus :	Affichage sur la première ligne de : «SmartCity2021 » et sur la seconde «CERI, MasterCMI »

BILAN

Affichage effectué.

REMARQUES

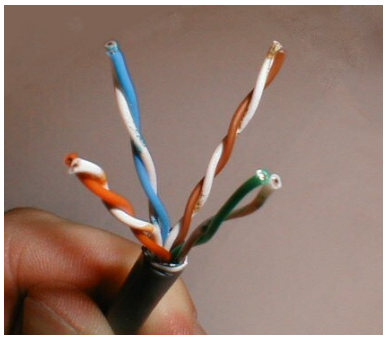
Pour le moment l'affichage sur l'écran est la seule chose qui fonctionne du coté de l'I2C, de mon côté toutes les commandes sont fonctionnelles, il me faut pour tester les autres en situation réelle que mes camarades ai fini de faire fonctionner les éléments de la maquette.

CONCLUSION

VALIDE	X	NON VALIDE	

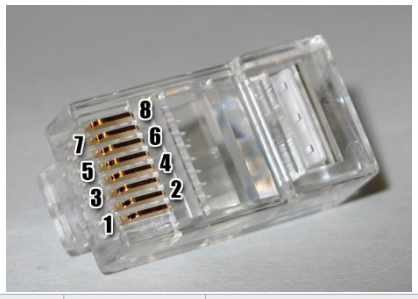
10. Partie physique

N'ayant quasiment aucun sujet qui coïncide avec de la physique j'ai décidé de vous présenter les caractéristiques d'un câble RJ45. Souvent appelé câble Ethernet mais c'est parce que il utilise le protocole de communication Ethernet, il est utilisé dans les réseaux privé, branchés à une box d'un operateur internet en raison de sa faible portée (égal ou moins de 100mètres) ce qui ne lui permet pas de parcourir des continents, pour cela on utilise la fibre optique qui elle, possède une plus grande portée.
Déjà, il y a plusieurs type de câbles Ethernet les premiers sortait du 100Mbits/secondes avec une bande passante de 100MHz pour 100mètres et les derniers arrives actuellement à 10'000Mbits/secondes avec une bande passante de 1'000MHz pour toujours 100mètres et peuvent même aller plus vite en sacrifiant une partie de leurs mètres. La différence s'explique par le blindage tout d'abord les paires sont torsadées comme ceci :



Ensuite le blindage est différent à chaque génération afin de limiter les perturbations extérieurs on rajoute du blindage en feuille d'aluminium généralement soit au tour de chaque paire torsadée soit comme sur l'image autour du tout … en bref plus le blindage est efficace, plus on peut augmenter la valeur de la bande passante ce qui permet d'augmenter le débit du câble.

Le câble RJ45 possède 8fils de cuivre qui sont arrangés différemment sur les prises RJ45 situé à chaque bout du câble, l'association du brochage de chaque prise donne le type du câbles, et on a deux type de câbles, les câbles droit et les câbles croisés



Cable droit il possède le même brochage des deux cotés, c'est celui qui est le plus utilisé car son utilité c'est que l'on peut brancher un switch, commutateur, routeur à un ordinateur:

Prise RJ45 câblée en T568A		Câble			Prise RJ45 câblée en T568A	
broche	couleur	paire	—	paire	couleur	broche
1	blanc-vert	3	—	3	blanc-vert	1
2	vert				vert	2
3	blanc-orange	2	—	2	blanc-orange	3
4	bleu	1	—	1	bleu	4
5	blanc-bleu				blanc-bleu	5
6	orange	2	—	2	orange	6
7	blanc-marron	4	—	4	blanc-marron	7
8	Marron				Marron	8

Câble croisé il inverse les brochages des paires vertes et orange ce qui permet de brancher deux ordinateur ensemble par un câble RJ45 (ce qui me fais penser au RX/TX sur une arduino que l'on doit parfois inverser pour que le rx soit avec le tx et que le tx soit avec le rx de l'autre) :

Prise RJ45 câblée en T568A		Câble			Prise RJ45 câblée en T568B	
broche	couleur	paire	—	paire	couleur	broche
1	blanc-vert	3	\ /	2	blanc-orange	1
2	vert				orange	2
3	blanc-orange	2	/ \	3	blanc-vert	3
4	bleu	1	—	1	bleu	4
5	blanc-bleu				blanc-bleu	5
6	orange	2	/ \	3	vert	6
7	blanc-marron	4	—	4	blanc-marron	7
8	marron				marron	8

11. Objectifs pour la suite du projet

A présent le Serveur TCP accompagné de sa gestion des clients avec Threads est en marche.
Le protocole est rédigé et appliqué dans le code. L'utilisation des accesseurs et mutateurs en direction de la zone de donnés commune est faite.
Il ne reste plus qu'a tester le tout pour savoir si cela fonctionne. Quand la personne chargée de la base de données aura fini son travail, il restera à intégrer sa classe au programme.

Partie Étudiant EC 1 – Équipe 01 : DUBOC Lucas

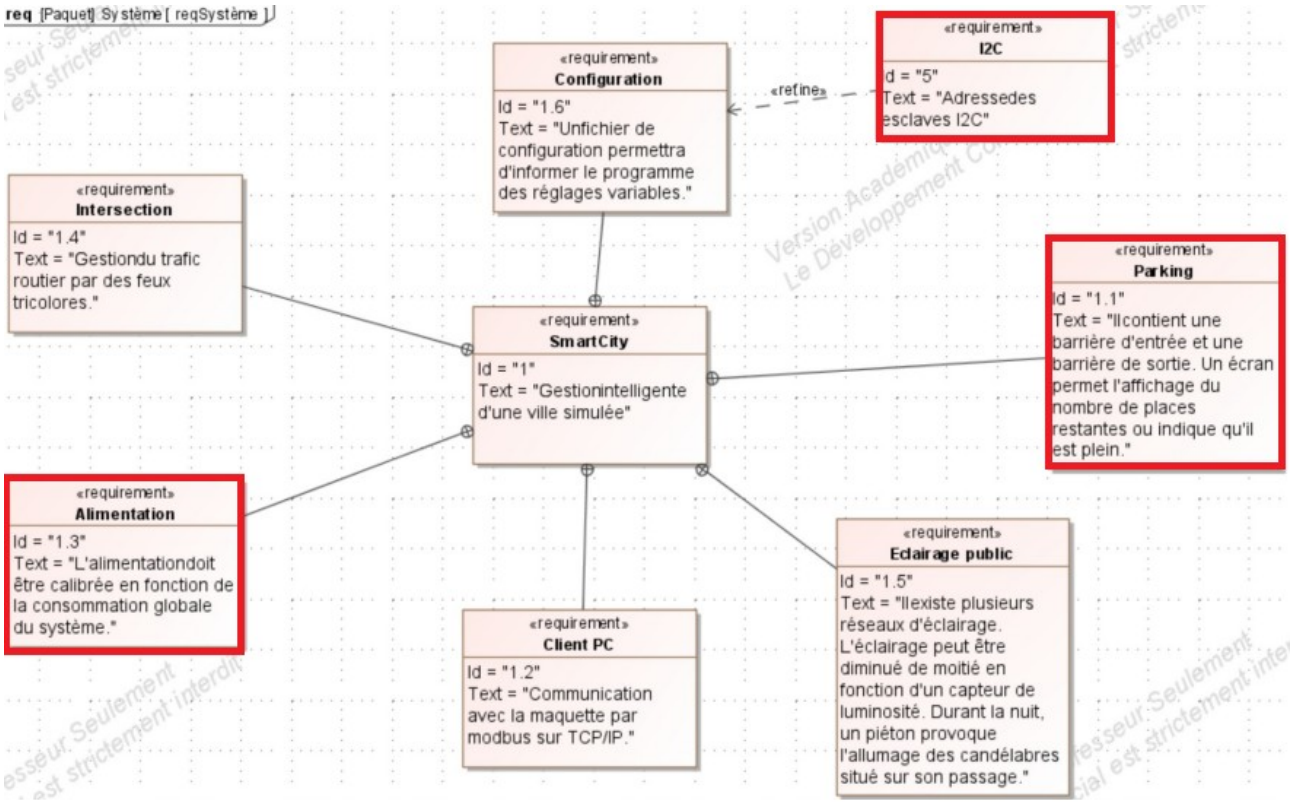
1.Présentation

1.1 Diagramme des exigences EC11

Étudiant 3 **EC 11**	*Liste des tâches assurées par l'étudiant* ***Accès parking version Arduino Nano*** • Concevoir une carte de gestion du parking *(identification des usagers, commande des barrières, gestion des appels d'urgence)* gérée par une carte Arduino Nano, à partir d'un schéma proposé. • Travail en collaboration avec l'étudiant EC12 dont l'objectif est identique mais avec un microcontrôleur différent. • Effectuer tous les tests nécessaires pour valider la structure, en la modifiant si nécessaire. • Participer à l'élaboration d'un protocole de communication sur le bus I2C avec l'étudiant IR concerné. • Participer à la conception des parties mécaniques du parking. • Effectuer un travail de documentation afin de déterminer une structure permettant de piloter une vraie barrière, et si possible l'intégrer sur la carte. • Effectuer la saisie du schéma et le routage de la solution retenue. Produire les fichiers Gerber afin que la fabrication du PCB soit sous-traitée. • Câbler la carte et effectuer les essais. • Documenter la mise en service de la carte finalisée.	**Installation :** IDE Arduino. **Mise en œuvre :** Tester/valider/modifier une structure utilisant une carte Arduino nano pour gérer le parking afin de piloter les servo-moteurs des barrières, et gérer les capteurs de badges RFID pour identifier les usagers. La carte fonctionnera en tant que circuit esclave sur le bus I2C. L'analyse devra être menée conjointement avec l'étudiant EC12 dont la carte est gérée par un autre circuit. Prévoir la disposition de l'afficheur LCD avec lui également. Suite aux essais, modifier si nécessaire le schéma structurel proposé. Participer à une réflexion commune avec tous les étudiants EC du projet SmartCity pour le câblage du faisceau distribuant l'alimentation et le bus I2C sur tout le démonstrateur. **Réalisation :** Après validation de la solution, concevoir un circuit imprimé devant être fabriqué industriellement. Mettre en œuvre le faisceau de communication/alimentation. **Documentation :** • Avantages / inconvénients de cette solution par rapport à celle d'une carte avec microcontrôleur Attiny3217. • Schéma de câblage rapide (Fritzing) pour documenter la phase d'essais. • Documents de fabrication de la carte (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. • Schéma structurel avec contours IBD. • Liste complète des composants avec leurs sources d'approvisionnement et leurs prix. • Programme de gestion de la carte, accompagné des commentaires et diagrammes nécessaires à sa compréhension. • Fiche de mise en service. • Fiche de dépannage.

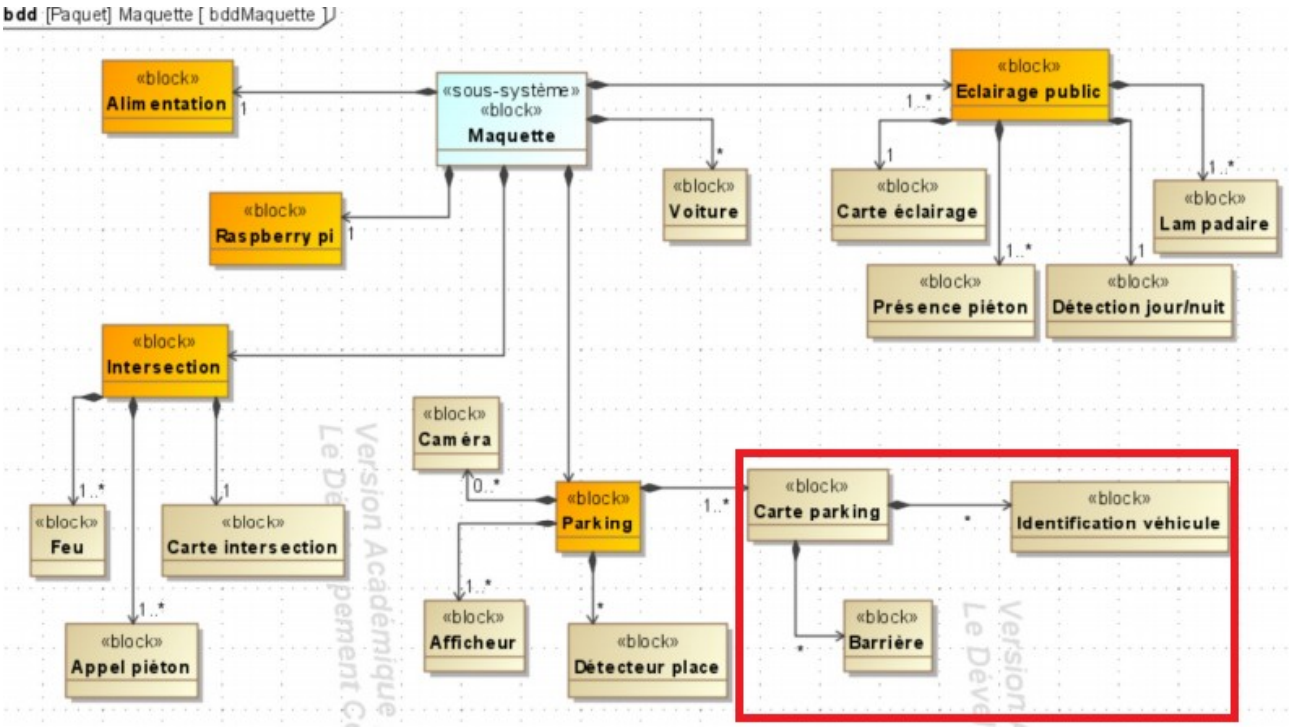
2 Diagrammes

2.1 Diagramme des exigences EC11



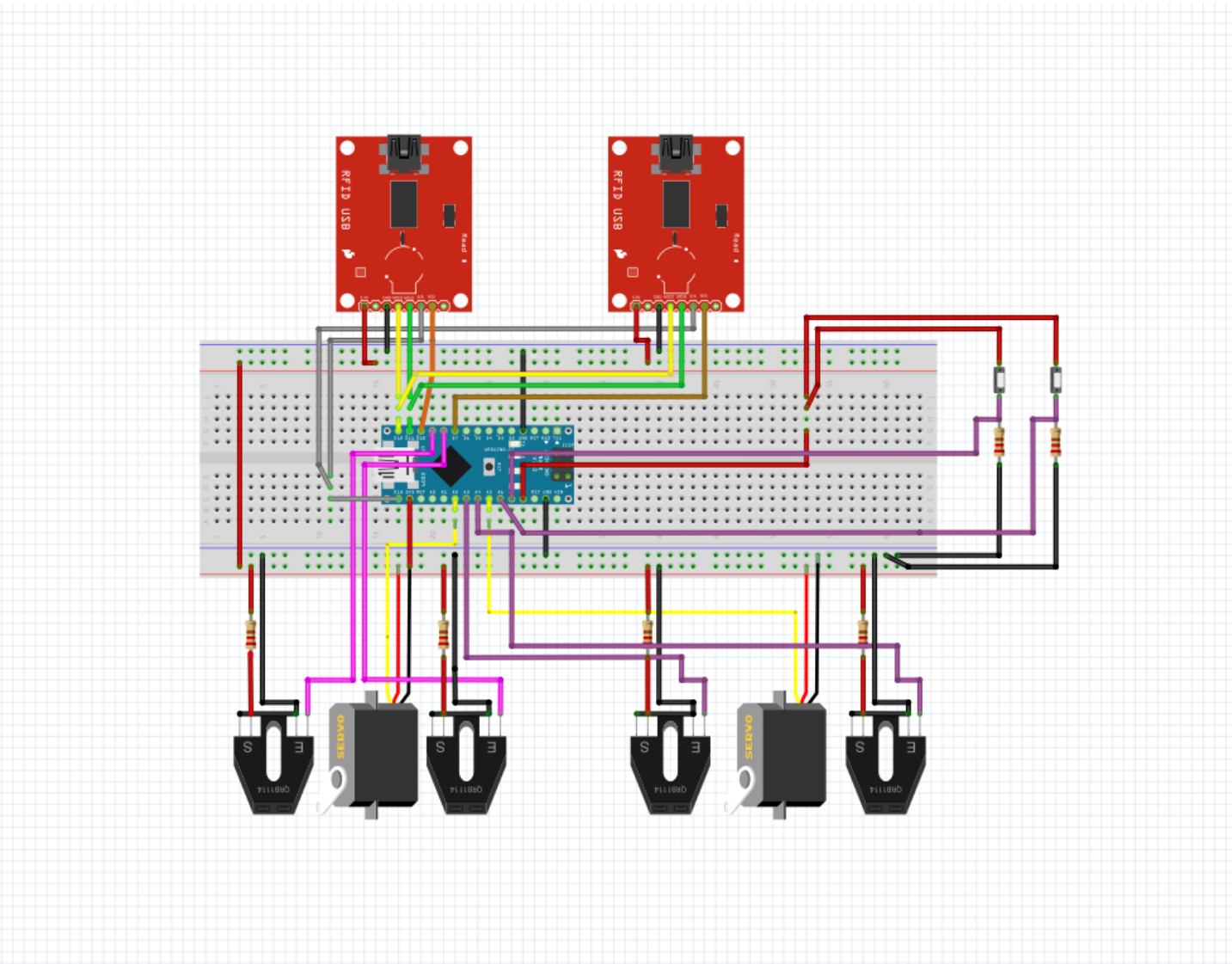
Le projet consiste en une ville intelligente, plus précisément je travail sur un parking intelligent.
En ce qui me concerne, je dois réaliser un système de barrière fonctionnant sous la technologie RFID, sous support Arduino NANO en utilisant des cartes d'identification, permettant donc de savoir qui rentre dans le parking .
Je dois également pouvoir communiquer avec une base de donnée en utilisant le protocole I2C, qui a l'avantage d'être très fiable et simple d'utilisation .

2.2 Diagrammes de bloc



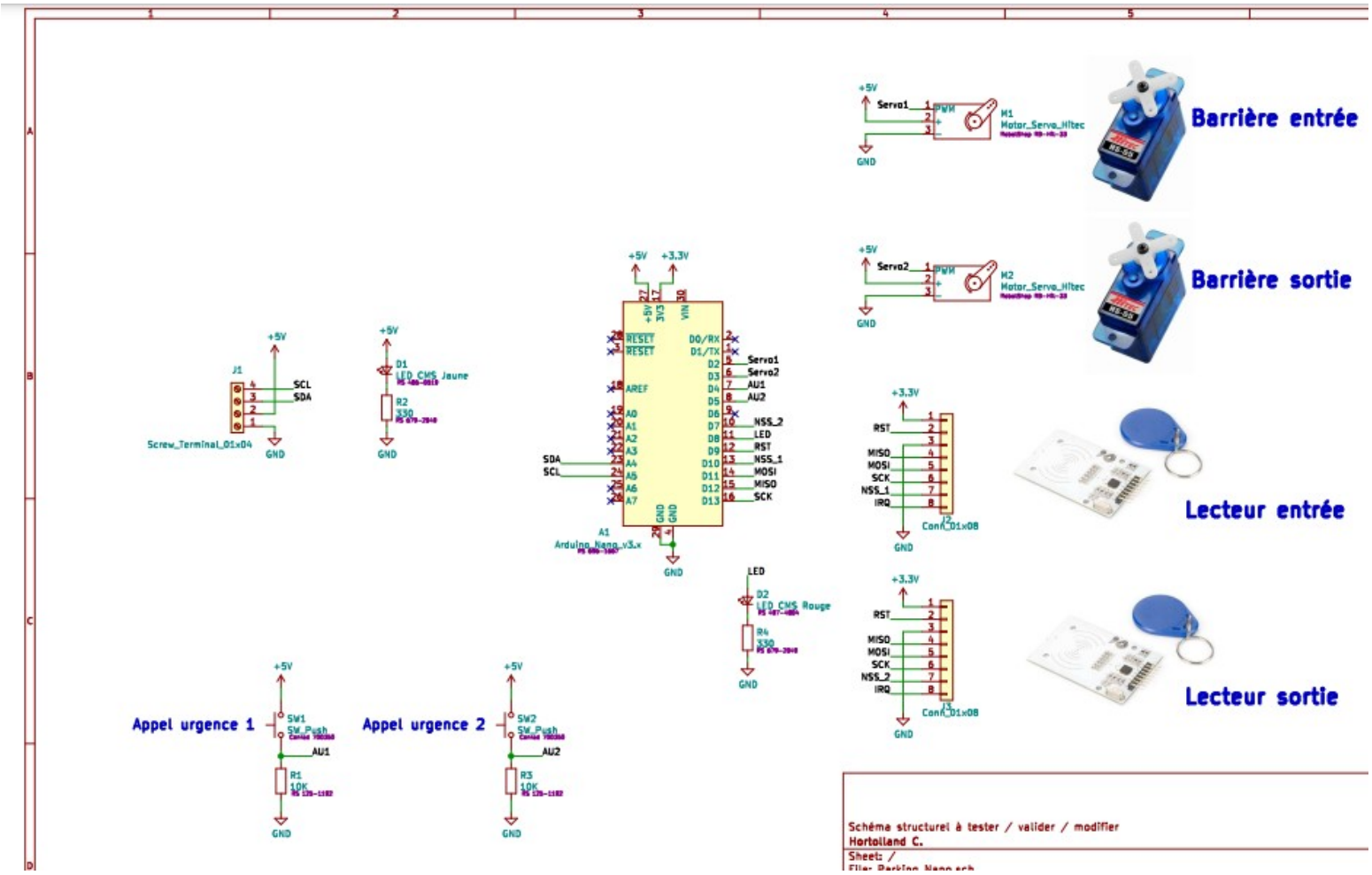
On peut voir ici que mon projet est directement lié au parking, en outre on peut voir que les barrières sont gérées par le système d'identification RFID .

3. Fritzing



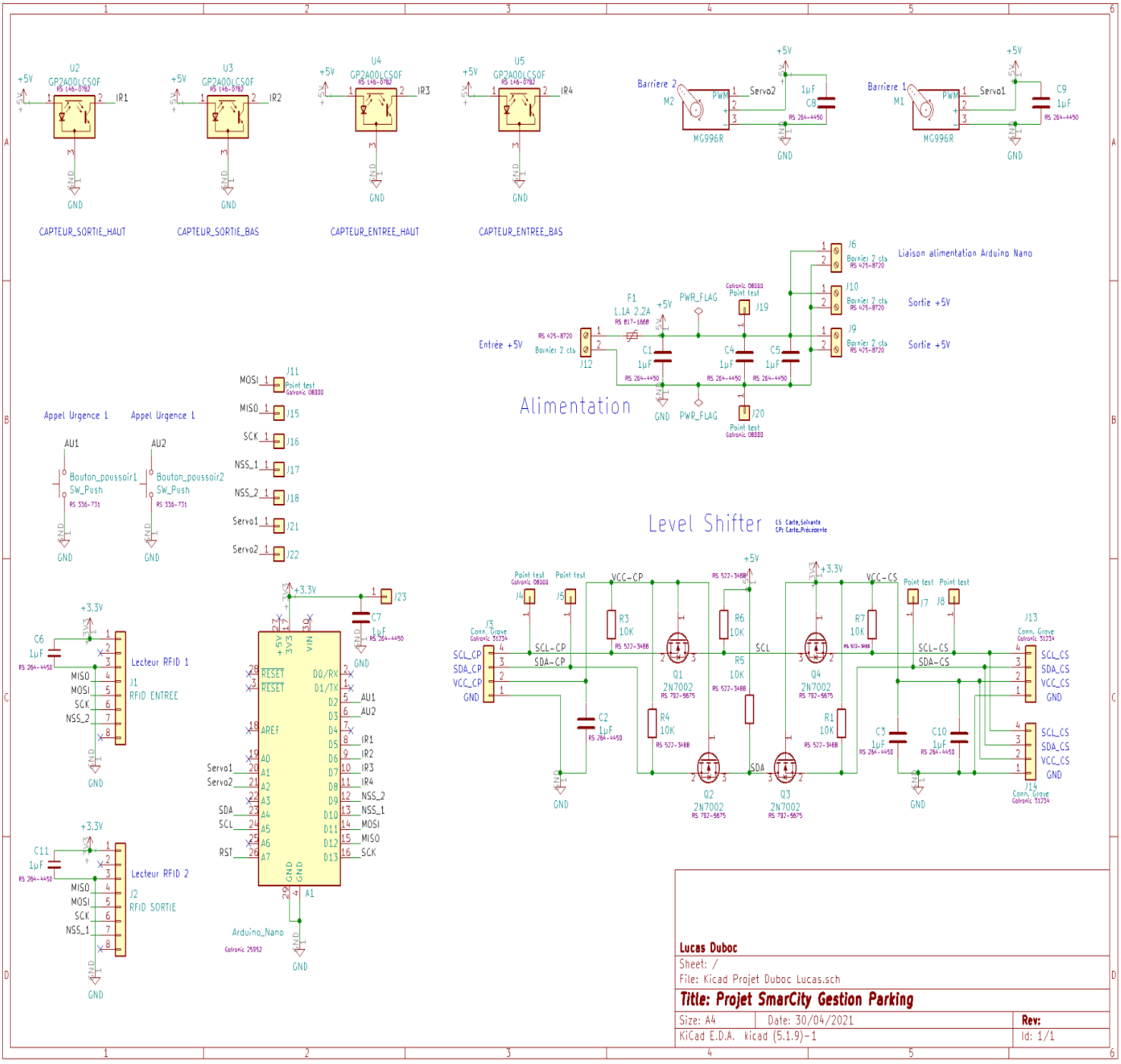
Ci dessus un fritzing représentant les 2 barrières avec leurs capteurs de position, ainsi que les lecteurs RFID .

4. Schéma structurel

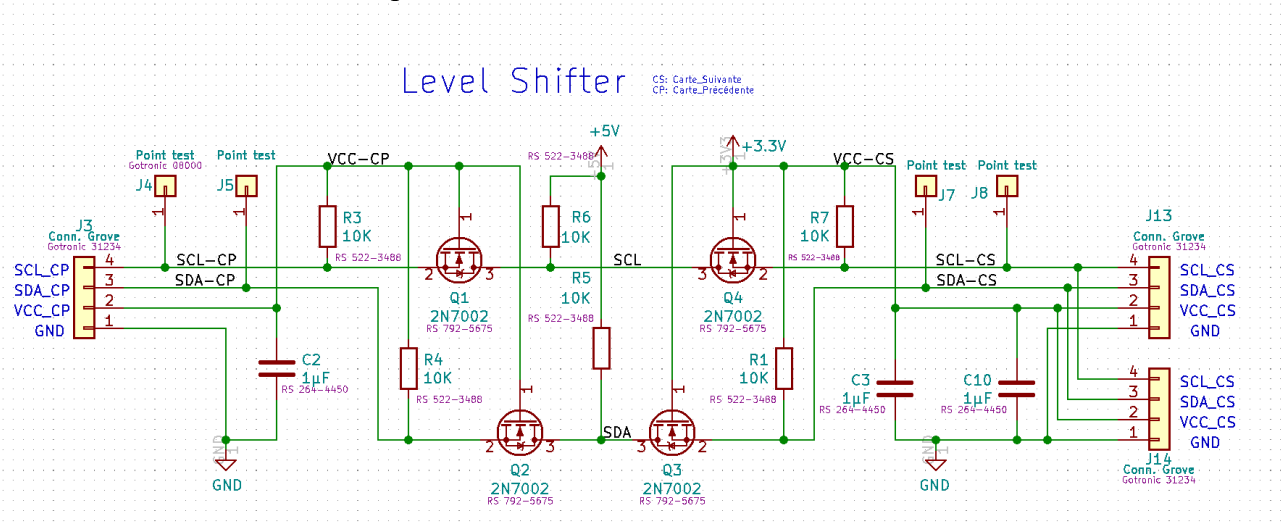


Ce schéma m'a été d'une grande aide pour comprendre mes composants ainsi que comment les mettre en œuvres et les utiliser, notamment pour le câblages et pour définir les pins une fois dans le code .

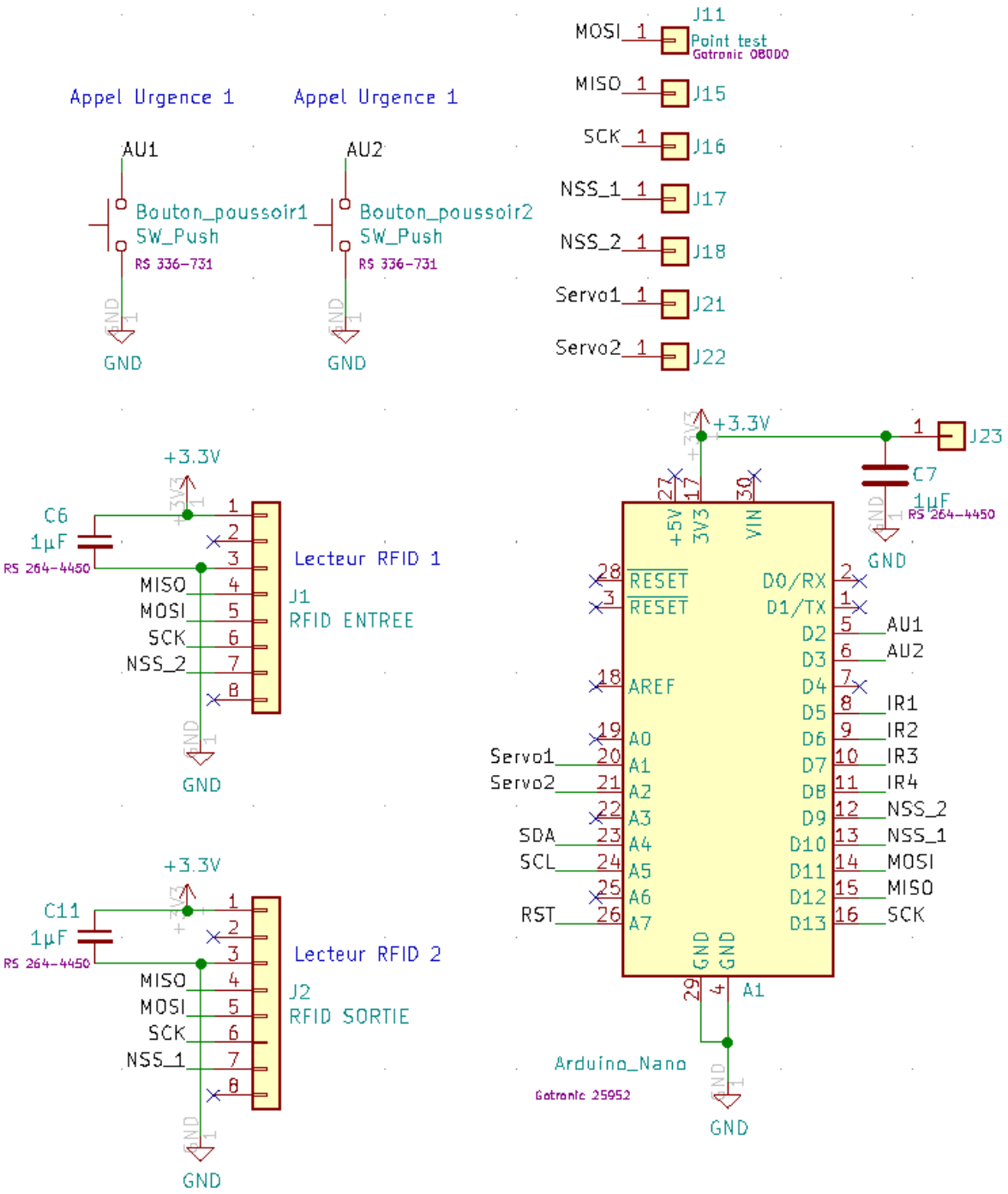
En utilisant Kicad, un software de saisie et de routage gratuit j'ai pu réaliser mon propre schéma structurel. En utilisant le schéma se trouvant ci-dessus


CAPTEUR_SORTIE_HAUT
CAPTEUR_SORTIE_BAS
CAPTEUR_ENTREE_HAUT
CAPTEUR_ENTREE_BAS
Barriere 2
Barriere 1
MG996R
Liaison alimentation Arduino Nano
Sortie +5V
Entrée +5V
Alimentation
Appel Urgence 1
Appel Urgence 1
Level Shifter
Lecteur RFID 1
RFID ENTREE
Lecteur RFID 2
RFID SORTIE
Arduino_Nano
Lucas Duboc
Sheet: /
File: Kicad Projet Duboc Lucas.sch
Title: Projet SmarCity Gestion Parking
Size: A4
Date: 30/04/2021
Rev:
KiCad E.D.A. kicad (5.1.9)-1
Id: 1/1

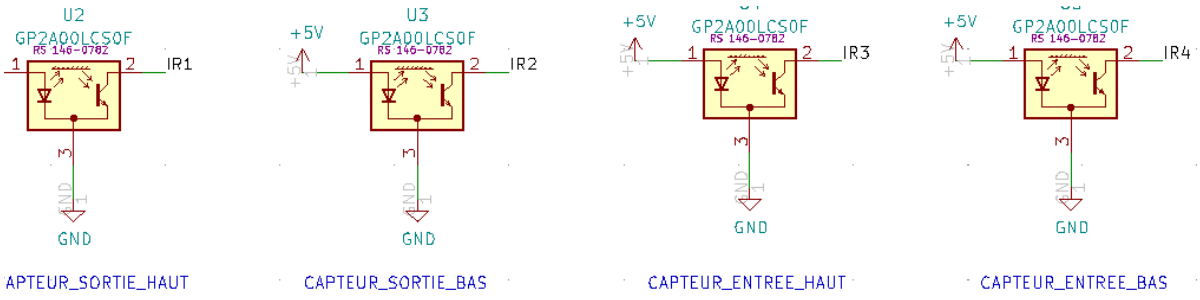
Ci dessous les différentes composantes du schéma structurel :

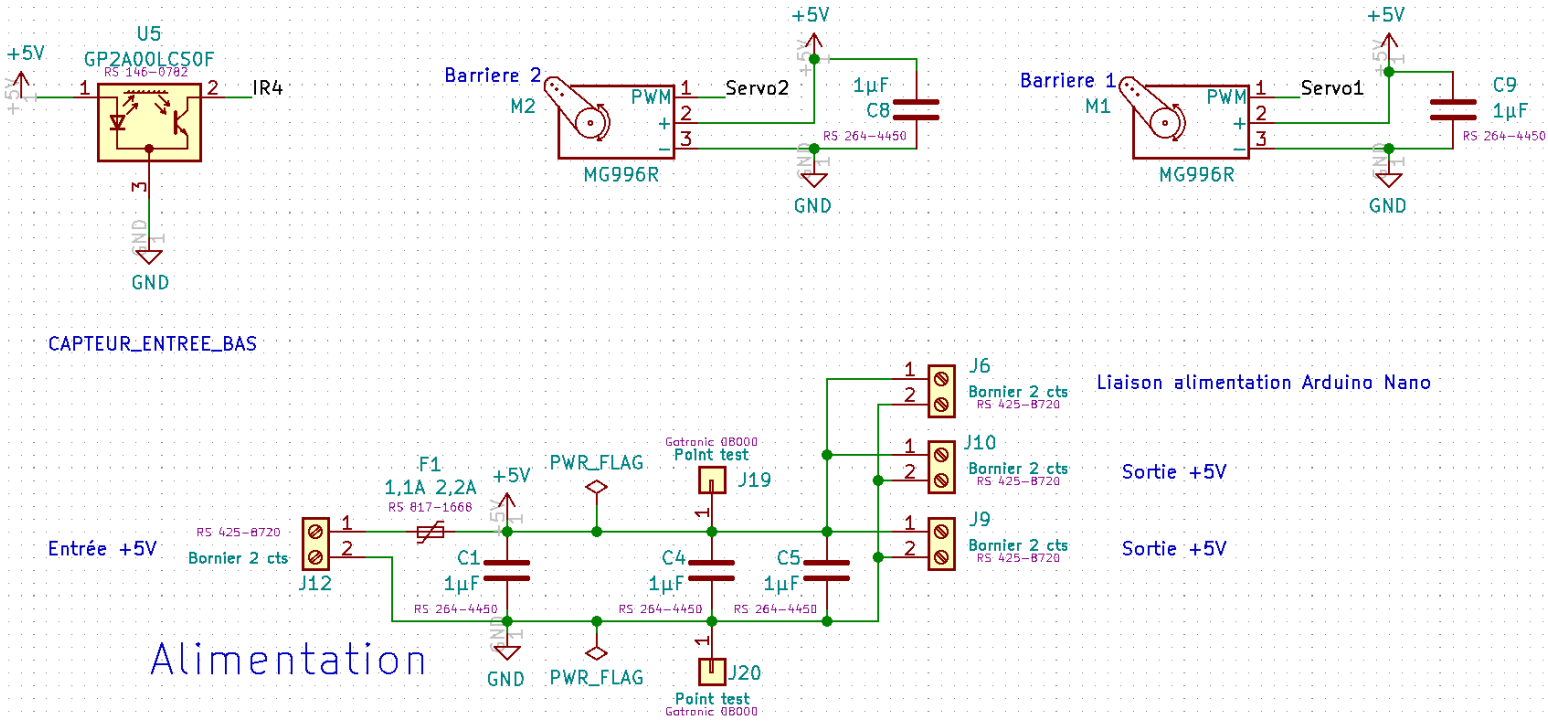


Ici on peut voir le « level shifter » qui permet d'adapter la tension entre les différentes cartes présentes dans le projet. La particularité de ce « level shitfter » est qu'il a la capacité à transformer une tension 5v en 3v et vice versa.



Ici on peut apercevoir la carte arduino, les lecteurs RFID ainsi que les boutons d'appel d'urgence et les multiples points tests





On aperçoit ici les capteurs de position (U1,U2,U3, et U4) ainsi que les servomoteurs et

l'alimentation de la carte. Il est important de noter la présence sur l'alimentation, d'une diode « Schottky D1 » permettant de forcer le sens du courant, son avantage par rapport aux autres diodes est sa tension minimale de crête bien inférieure aux autres diodes, limitant ainsi sa consommation (si on venait par exemple à inverser la polarité à l'entrée, la carte ne serait pas endommagée pour autant grâce à la propriété de la diode qui force le sens du courant).

Notons aussi la présence d'un Polyfuse ou Polyswitch F1, un fusible réarmable CTP (CTP = coefficient en température positif) bloquant le courant si la température est trop élevée (empêchant alors des dégâts sur la carte), l'un de ses avantages par rapport à un fusil traditionnel, c'est qu'il retrouve son état original après un chute de température, et qu'il n'y a pas de nécessité à le remplacer.

Sur l’ensemble du schéma structurel on constate la présence de condensateurs (C1, C2, C3…) chacun possède une valeur de 1µf. Il s’agit de condensateur de découplage ayant pour mission de stabiliser le signal, ils se trouvent à proximité de l’Arduino, des moteurs ou des RFID, il est important de les placer au plus près des composants afin qu’ils remplissent au mieux leur rôle.

5 Diagramme de Gantt

N°	Mode Tâche	Nom de la tâche	Début
1		**Evaluations du projet**	**Mer 10/02/21**
2		Revue 1	Mer 10/02/21
3		Revue 2	Mer 14/04/21
4		Dépôt du dossier	Ven 28/05/21
5		Soutenance finale E62 (Revue 3)	Lun 14/06/21
6		**Epreuve E4**	Jeu 13/05/21
7		**Epreuve E1**	Mar 11/05/21
8		Vacances d'hiver	Lun 22/02/21
9		Vacances printemps	Lun 26/04/21
10		**Projet**	**Mer 06/01/21**
11		**Specifications générales**	**Mer 06/01/21**
12		Prise de connaissance des documents	Mer 06/01/21
13		Création d'un rapport journalier	Mer 06/01/21
14		Réunion Projet	Jeu 07/01/21
15		Conception du gantt	Jeu 07/01/21
16		**Essais et validation des structures (prototypage rapide)**	**Jeu 14/01/21**
17		Analyse du shcéma structurel	Mer 13/01/21
18		Test sur Proteus	Mer 20/01/21
19		**Prototypage rapide et routage**	**Mer 03/02/21**
20		**Conception sur plaque à essai**	**Mer 03/02/21**
21		Circuit Servo-Moteur	Mer 03/02/21
22		Circuit RFID	Mer 10/02/21
23		Circuit I2C	Jeu 18/02/21
24		Fritzing	Mer 03/03/21
25		Mise en commun des plaques à essai	Mer 10/03/21
26		Réfléchir en équipe sur la connectique I2C	Mer 17/03/21
27		Création du schéma structurelle sur Kicad	Ven 19/03/21
28		Liste de matériel	Mer 31/03/21
29		Routage	Ven 02/04/21
30		Réalisation Gerber + Commande du circuit PCB	Mer 14/04/21
31		**Fabrication et essais**	**Mer 12/05/21**
32		Test de la carte	Mer 12/05/21
33		Brasage du PCB	Mer 19/05/21
34		Intégration	Mer 26/05/21
35		**Rédaction du dossier**	**Mer 02/06/21**
36		Se mettre d'accord pour le dossier	Mer 02/06/21
37		Faire les schémas de blocs	Ven 04/06/21
38		Création et rédaction du dossier	Jeu 23/09/21
39		**Epreuves BTS**	**Mar 16/02/21**
40		1er BTS Blanc	Mar 16/02/21
41		2nd BTS Blanc	Mar 20/04/21
42		**Soutenance**	**Mer 23/06/21**
43		Soutenance E6.2	Mer 23/06/21

6. Matériel

6.1 Matériel utilisé et à disposition

Une carte Arduino Nano
- Deux moteurs MG996R
- Deux lecteurs RFID Velleman VMA 405
- Deux badges et une carte RFID
- Deux Boutons poussoirs

6.2 Changement de matériel

J'ai dû remplacer les moteurs "HS-55 Feather" utilisés précédemment par des moteurs Réf : "MG996R". La première référence utilisée à présenter des problèmes de rotation rédhibitoires ne permettant pas de les maintenir en lieu et place.
Il est important d'orienter son choix vers un modèle dont la rotation est limitée à 180°, car sur les modèles à 360° la rotation ne peut pas être arrêtée .

6.3 Liste de matériel

Reference(s)	Value
A1	Arduino_Nano
Bouton_poussoir1, Bouton_poussoir2	SW_Push
C1, C2, C3, C4, C5, C7, C8, C9, C10	1µF
C6, C11	1µF
D1	PMEG3010ER
F1	1,1A 2,2A
J1	RFID ENTREE
J2	RFID SORTIE
J3, J13, J14	Conn. Grove
J4, J5, J7, J8, J11, J15, J16, J17, J18, J19, J20, J21, J22, J23	Point test
J6, J9, J10, J12	Bornier 2 cts
M1, M2	MG996R
Q1, Q2, Q3, Q4	2N7002
R1, R3, R4, R5, R6, R7	10K
U2, U3, U4, U5	GP2A00LCS0F

7 . Travail réalisé

Qty	Reference(s)	Value	Code commande	Code commande	Code commande 2	Code commande 3	Code commande 4
1	A1	Arduino_Nano	Gotronic 25952				
2	Bouton_poussoir1, Bouton_poussoir2	SW_Push	RS 336-731		RS 220-4260		
2	C1, C2	1¬BF	RS 264-4450				
9	C3, C4, C5, C6, C7, C8, C9, C10, C11	1¬BF	RS 264-4450				
1	D1	PMEG3010ER	RS 816-6855				
1	F1	1,1A 2,2A	RS 817-1668				
1	J1	RFID ENTREE					
1	J2	RFID SORTIE					
16	J3, J4, J5, J6, J7, J8, J9, J10, J13, J14, J15, J16, J20, J21, J24, J25	Point test	Gotronic 08000	Gotronic 08000			
3	J11, J22, J23	Conn. Grove	Gotronic 31234				
4	J12, J17, J18, J19	Bornier 2 cts	RS 425-8720				
2	M1, M2	MG996R					
4	Q1, Q2, Q3, Q4	2N7002	RS 792-5675				
6	R1, R2, R3, R4, R5, R6	10K	RS 125-1192				
4	U1, U2, U3, U4	GP2A00LCS0F	RS 146-0782		RS 532-096	RS 680-1395	RS 680-5094

7.1 Test du matériel

Avant de mettre en oeuvre le projet, j'ai d'abord essayé mon matériel, j'ai donc utilisé le programme test d'Arduino pour faire fonctionner les servomoteurs.

```
#include <Servo.h>

Servo myservo;
/

int pos = 0;    // Position de départ à 0 degres

void setup() {
  myservo.attach(6);  // Moteur relié à la pin 6
   Serial.begin(9600);   //on regle l'horloge à 9600 bauds

}


void loop() {



  for (pos = 0; pos <= 180; pos += 1) { // aller de  0 degres à 180 degres
    // in steps of 1 degree
    myservo.write(pos);              // indique au moteur d'aller au coordonnées de la variable "pos"
    delayl(15);                       // le moteur a 15ms pour atteindre la position

   Serial.println(pos); // on lit le caractère

  }

  for (pos = 180; pos >= 0; pos -= 1) { // aller de 180 degres à 0 degres
    myservo.write(pos);              //  indique au moteur d'aller au coordonnées de la variable "pos"
    delay(15);                       // le moteur a 15ms pour atteindre la position

   Serial.println(pos); // on écris la position du moteur sur la console

  }
delay(1000); // on inclut un délai  de 1 seconde

```

En modifiant les positions de départ et de fin de course, j'ai constaté qu'un des moteur forçait, il généré énormément de bruit.
J'ai dans un premier temps pensé à un problème de moteur, en fait j'ai réalisé que j'avais tout simplement trop vissé la pale du seromoteur, occasionnant par conséquent un blocage.

7.2 Test des cartes et des lecteurs RFID

J'ai ensuite essayé de comprendre le fonctionnement des lecteurs RFID en me documentant sur le site fabricant, j'ai également utilisé leur programme pour tester chaque carte et badges, permettant ainsi de connaître leurs identifiants.

```
if (rfid.readCardSerial()) {
  Serial.print(rfid.serNum[0]);
  Serial.print(" ");
  Serial.print(rfid.serNum[1]);
  Serial.print(" ");
  Serial.print(rfid.serNum[2]);
  Serial.print(" ");
  Serial.print(rfid.serNum[3]);
  Serial.print(" ");
  Serial.print(rfid.serNum[4]);
  Serial.println("");

  rfidRecu[0]= (unsigned char) rfid.serNum[0];
  rfidRecu[1]= (unsigned char) rfid.serNum[1];
  rfidRecu[2]= (unsigned char) rfid.serNum[2];
  rfidRecu[3]= (unsigned char) rfid.serNum[3];
  rfidRecu[4]= (unsigned char) rfid.serNum[4];

  for (int x = 0; x < sizeof(cards); x++) {

    for (int i = 0; i < sizeof(rfid.serNum); i++ ) {
      if (rfid.serNum[i] != cards[x][i]) {

        access = false;
        break;
      }
      else {

        access = true;
      }

    }


if (access) {

  Serial.println("Carte reconnue");
```

```
101 2 25 211 173
Carte reconnue
```

Défilement automatique Afficher l'horodatage

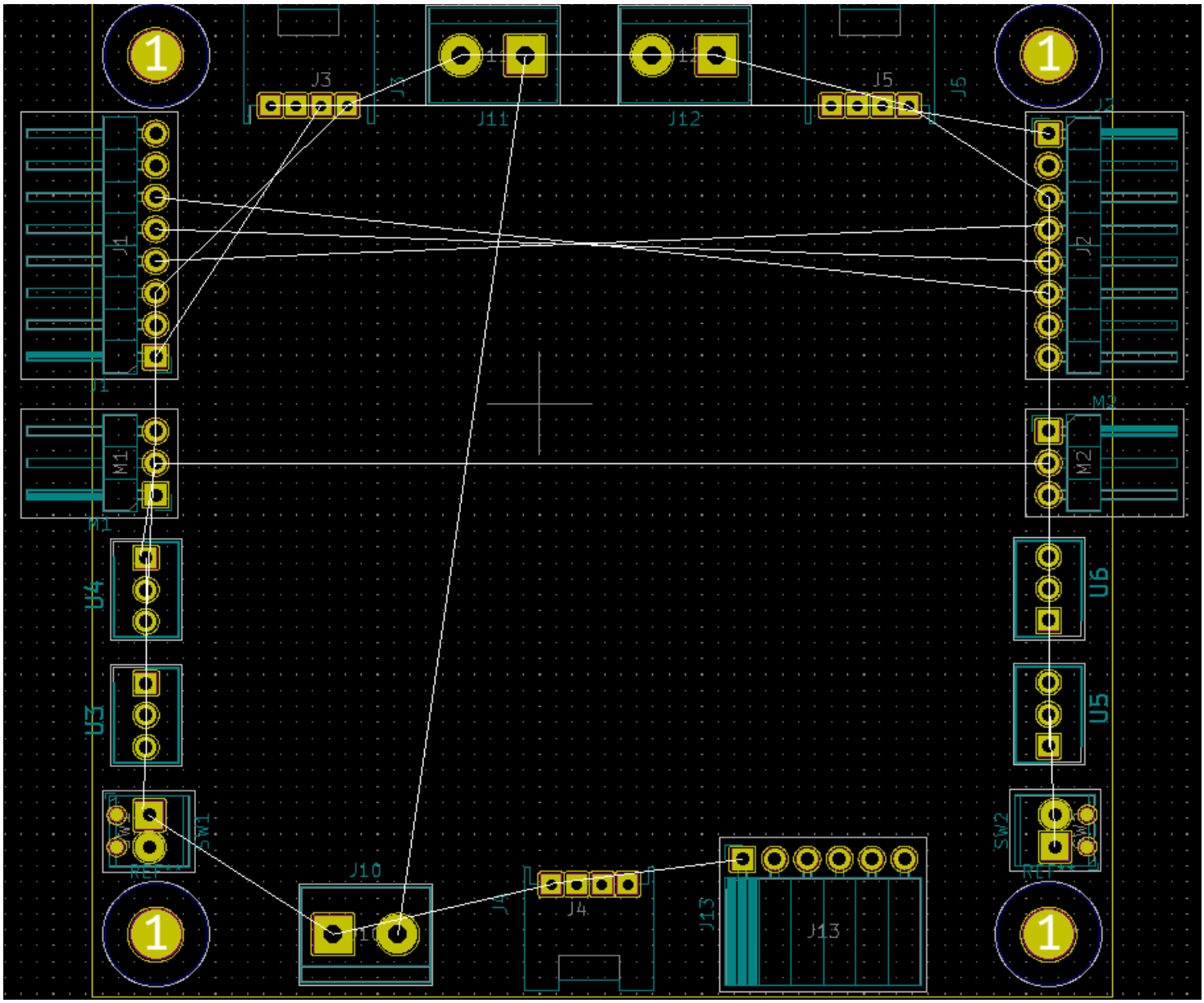
Ici à titre d'exemple la carte est reconnue avec l'ID 101 225 211 17

L'utilisateur ayant le numéro cité ci-dessus a été reconnu par le RFID, par conséquent il est autorisé à entrer, la barrière s'ouvrira.

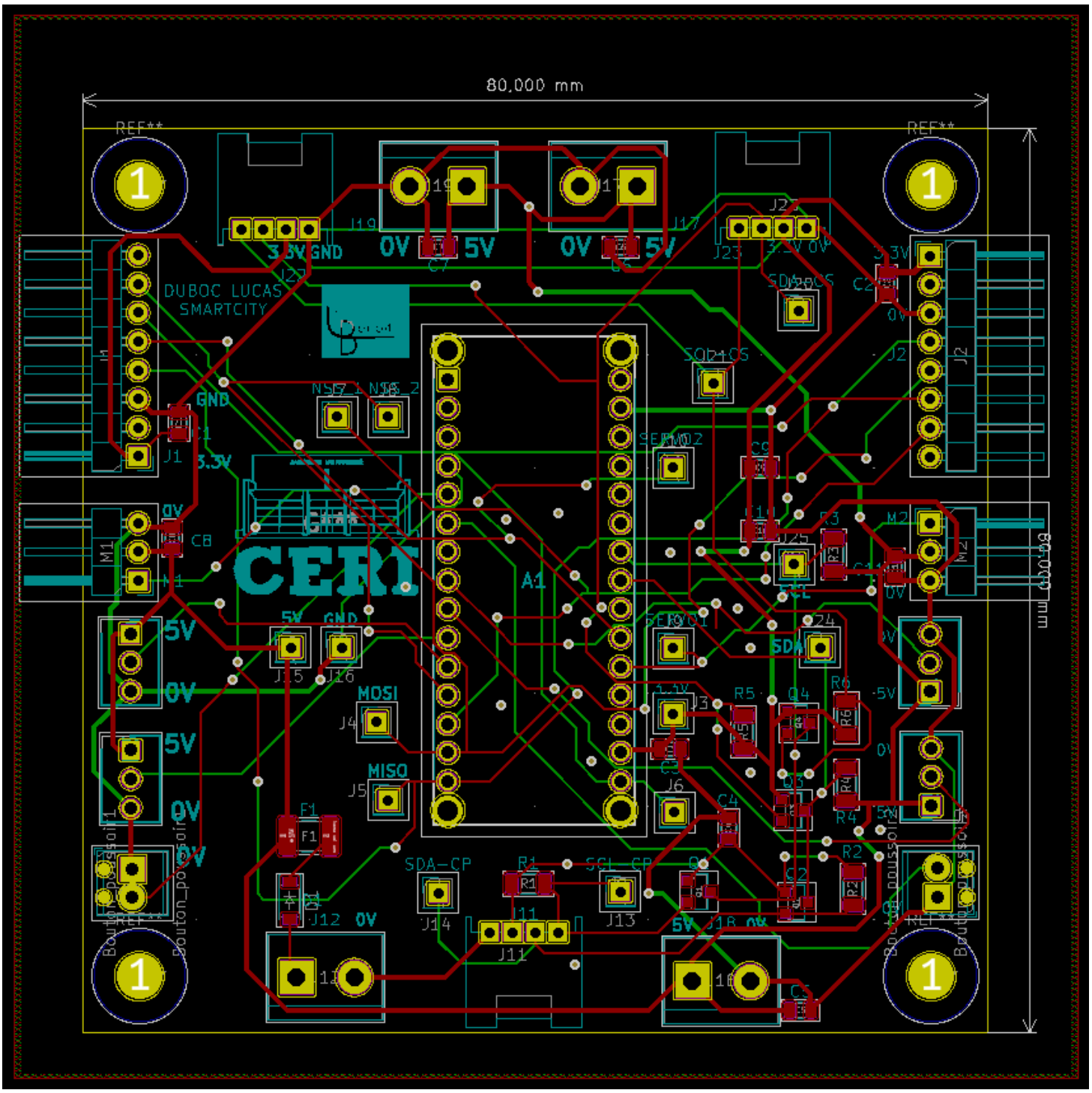
7.2 Schéma structurel et routage

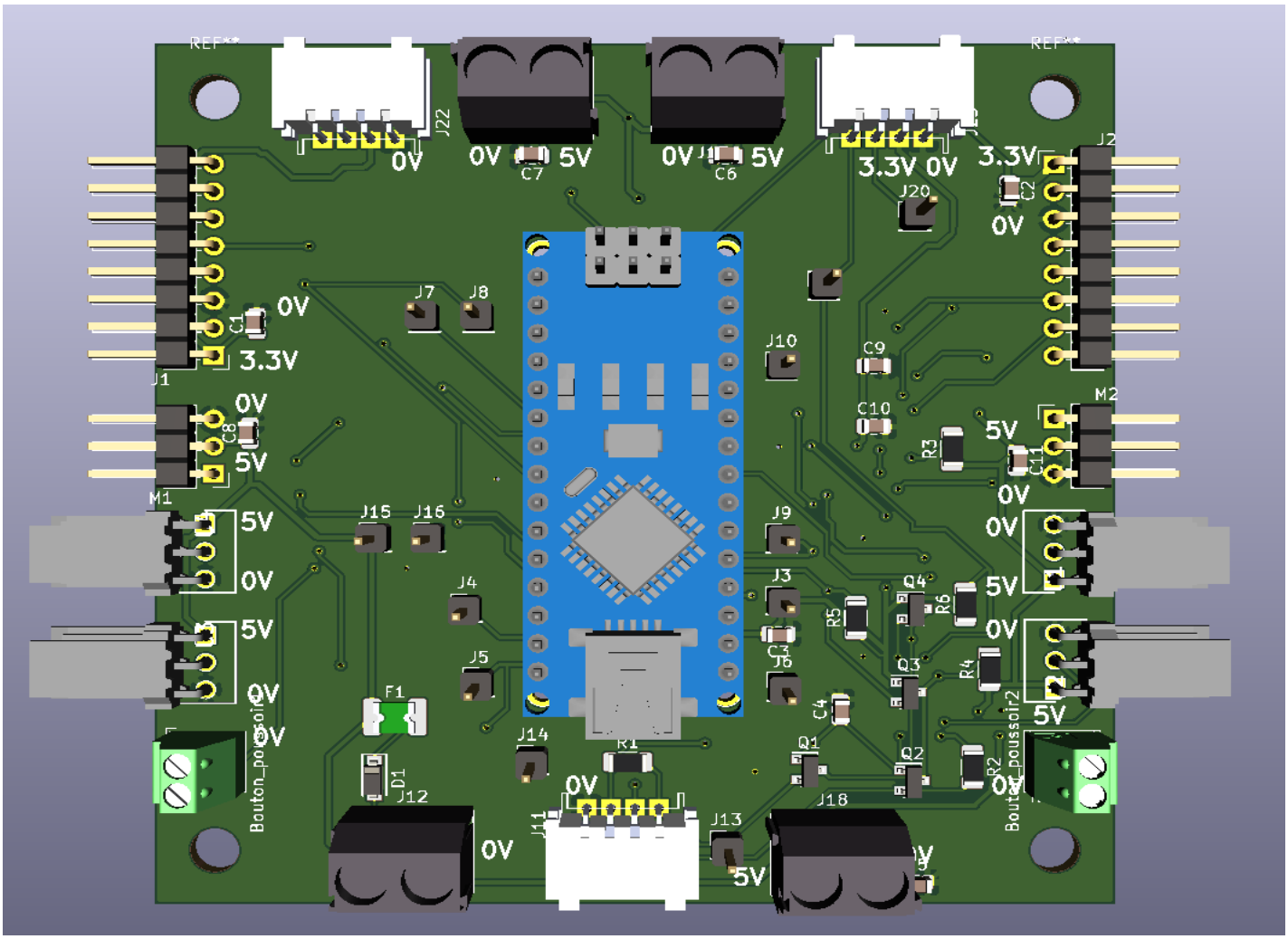
J'ai passé une grande partie du projet à réaliser mon schéma structurel, qui devait répondre aux besoins de ma carte mais également du projet en général, avec notamment le "level shifter" qui permet de communiquer dans les 2 sens . Il y a de nombreux ajouts, comme les fameux condensateurs de 1µf, les différents points test et des retraits, comme des résistances de "pull down" qui devaient être présentes au niveau des capteurs, cependant l'arduino remplissant déjà cette fonction cela permis de supprimer les résistances.
En ce qui concerne le routage, ce dernier devra être fait avec les mêmes contraintes que Loic du fait que nos cartes sont similaires.

Je dû par conséquent me baser sur l'implantation de sa carte, et plus particulièrement sur la position des trous de fixation 4 mm, ainsi que la disposition des connecteurs.
L'orientation des références du PCB à due se faire de manière uniforme pour assurer une bonne lecture.
Ci-dessous, on peut apercevoir le routage de Loic, avec seulement ses connecteurs (sur les bords droit et gauche) car il s'agit de la partie commune sur laquelle je devais me



Ci-dessous le rendu de ma carte :




REF**
REF**
J22
0V
0V
C7
5V
0V
C6
5V
3.3V
0V
3.3V
J2
J20
C2
0V
J7
J8
C1
0V
3.3V
J1
J10
C9
C10
M2
0V
C8
5V
R3
5V
C11
0V
M1
J15
J16
J9
0V
5V
J4
J3
Q4
R6
R5
0V
5V
J5
C3
Q3
R4
0V
F1
C4
5V
J14
R1
Q1
Q2
R2
0V
D1
J12
J11
J18
J13
0V
5V
Bouton_poussoir
Bouton_poussoir2

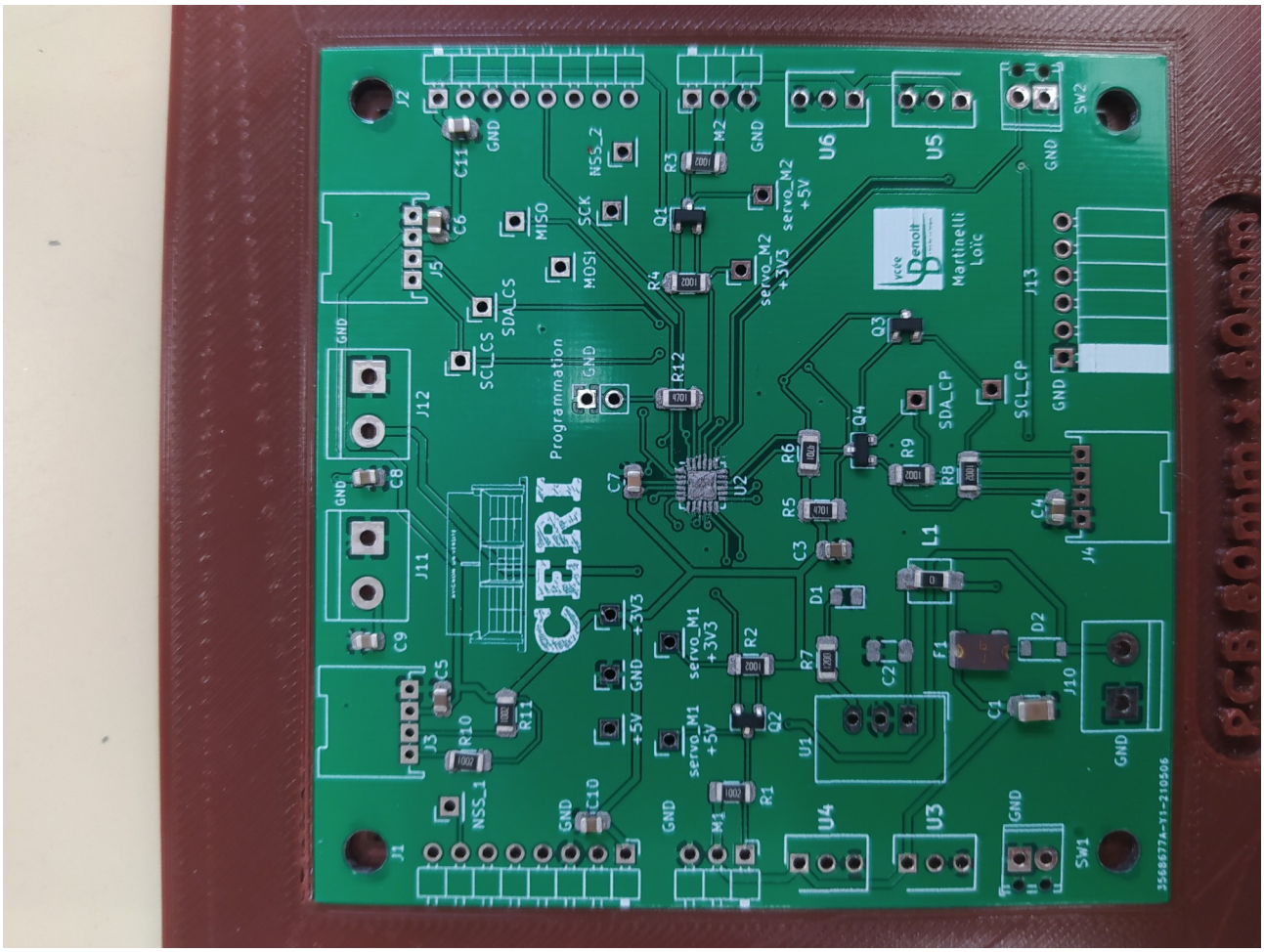
8. Assemblage de ma carte

Une fois le matériel commandé et reçu, il a fallu l'assembler, le souder, ayant des composants CMS et traversant j'ai d'abord soudé les composants CMS car la règle dicte de souder les composants les moins hauts en premiers .
Pour souder en CMS il faut respecter une procédure, il faut évidemment dresser une liste de composants, les récupérer, nettoyer sa carte à l'acide isopropylique, se munir du stencil associer à sa carte .

(le stencil est une fine plaque de métal qui sert de guide pour l'application de la pâte à braser)

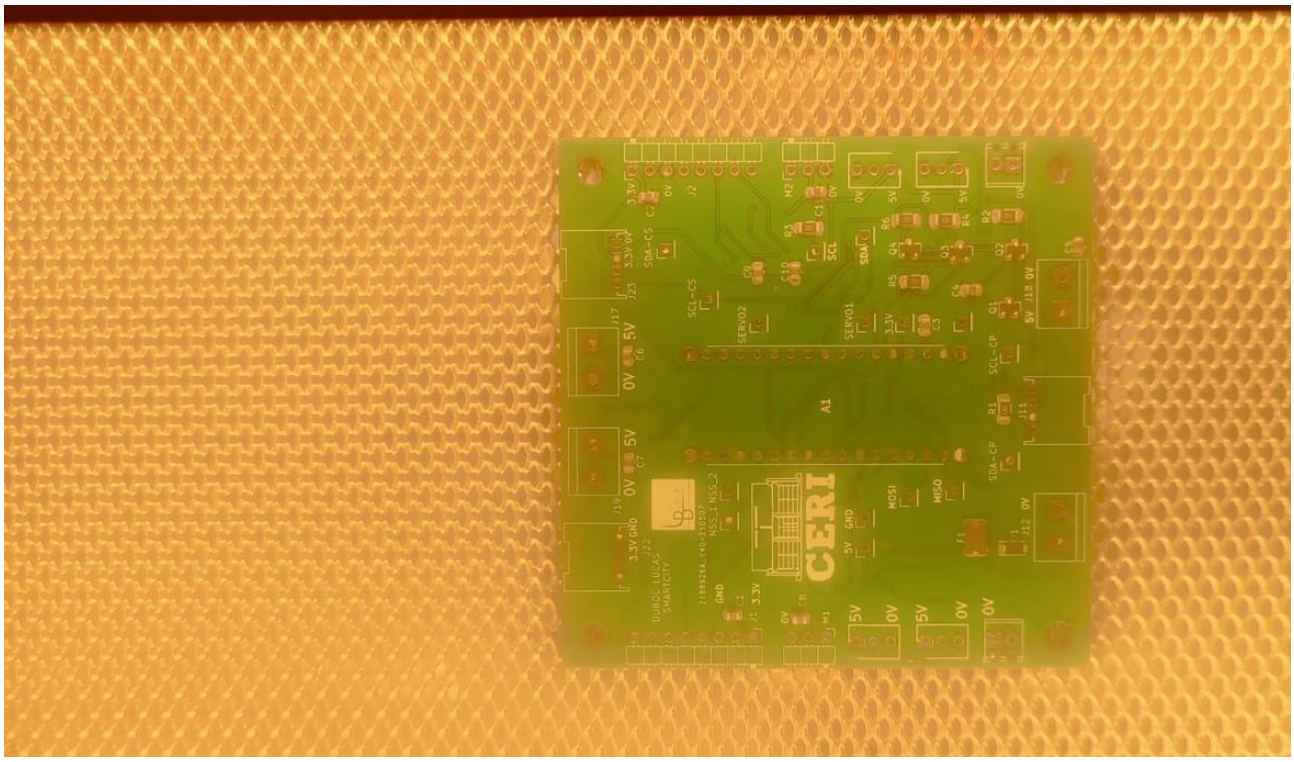
Il faudra ensuite associer; la carte, le stencil ainsi que le support stencil

(Ici on peut apercevoir un support stencil imprimer en 3D, on utilisera des aimants pour immobiliser la structure)

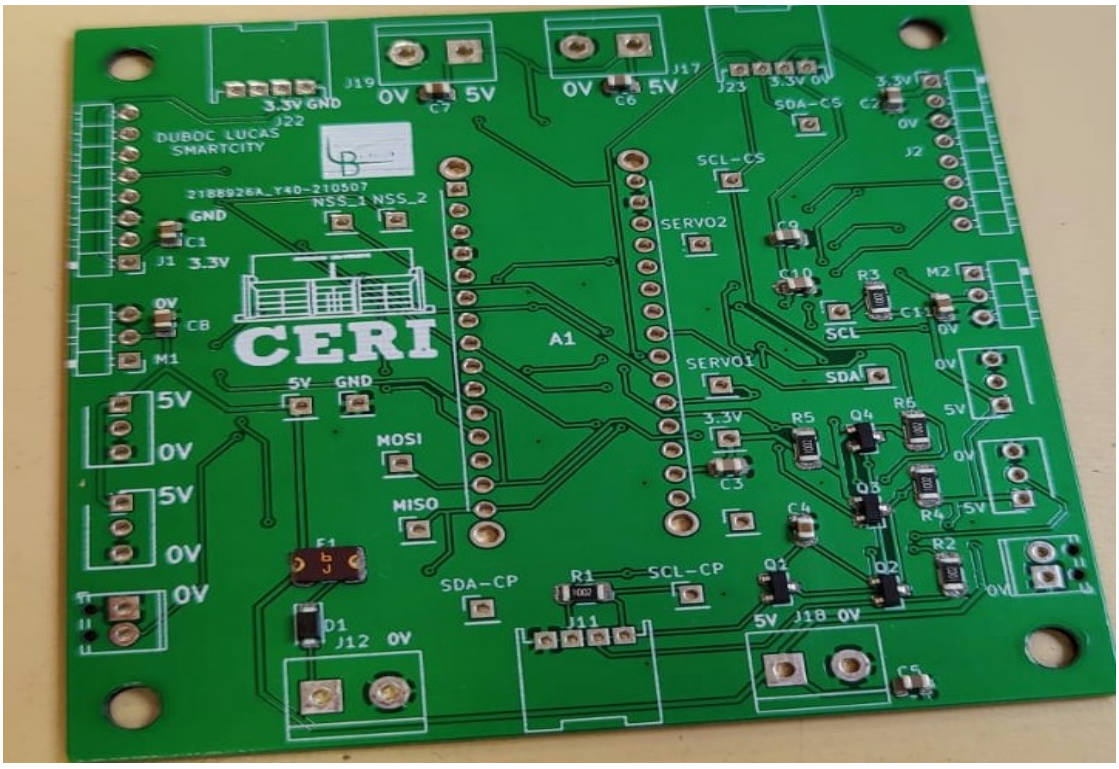


Une fois le tout « assembler » il déposera la pâte à braser qui grâce au stencil se déposera aux emplacements associés, on pourra alors placer les composants CMS à leur emplacements avec précaution et en utilisant un microscope si besoin.

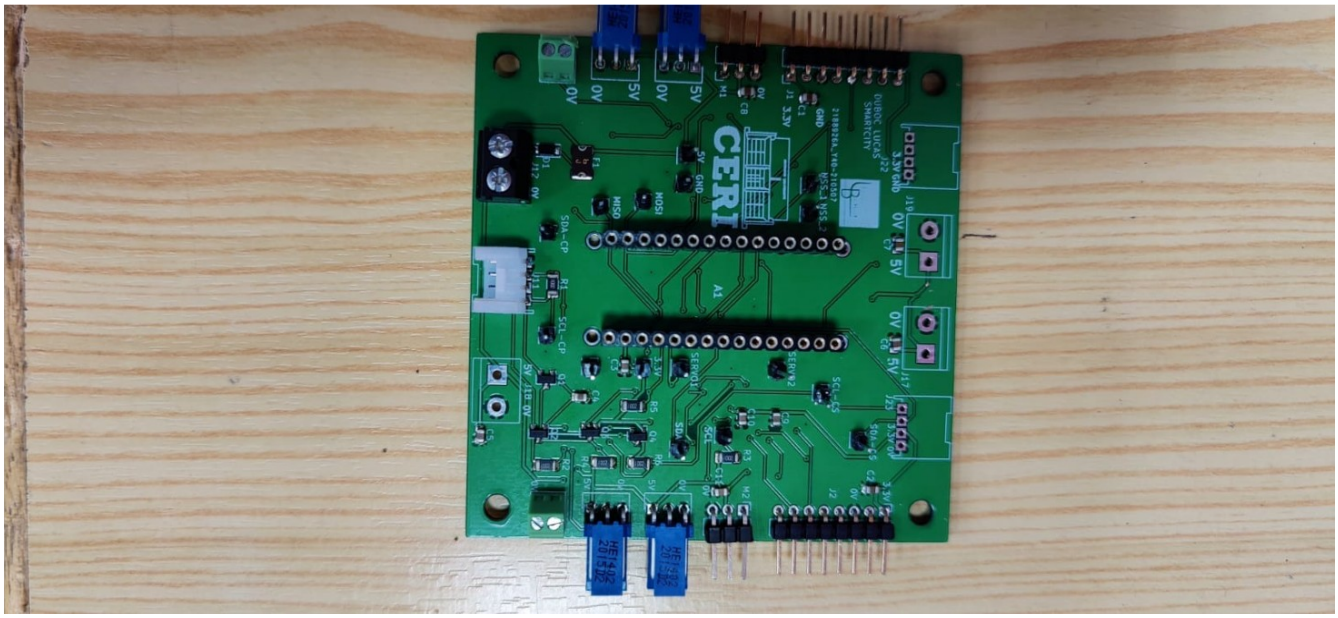
IN FINE, on pourra passer la carte au four pour durcir la pâte à braser et fixer les CMS.



Un PCB sortant du four aura alors le résultat ci-dessous :

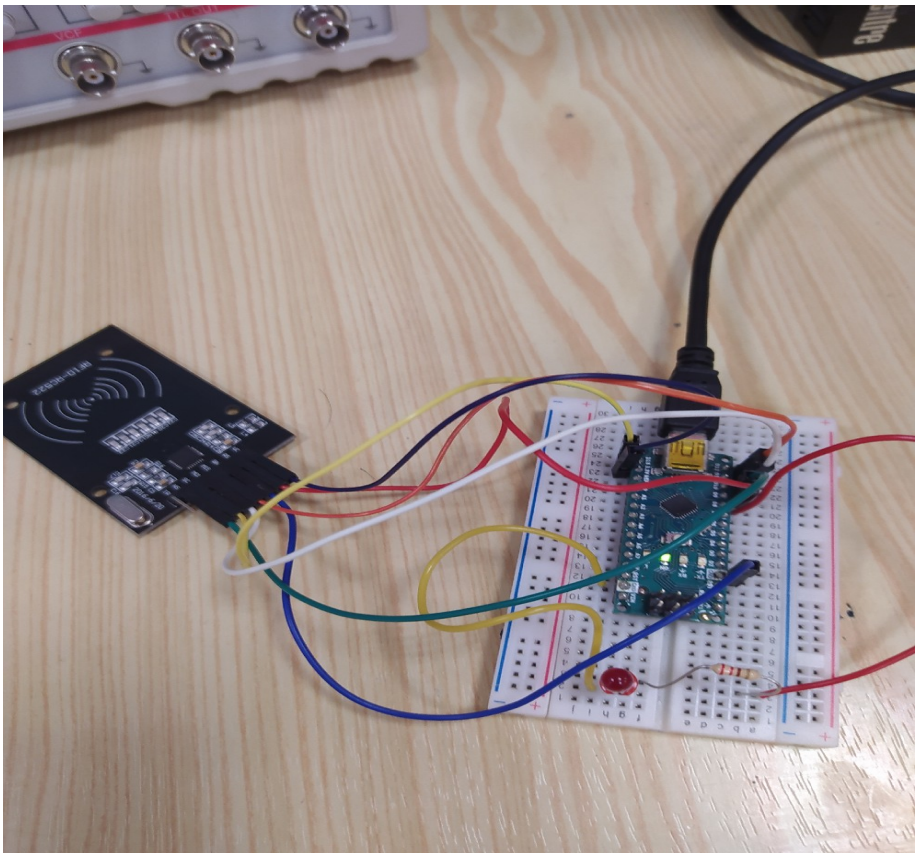


Il faudra cependant vérifier à la loupe, ou microscope si d'éventuels problèmes sont présents.
Une fois la recherche de défaut terminée, on pourra alors souder les composants dit, traversants qui eux se soudent de manière traditionnelle :



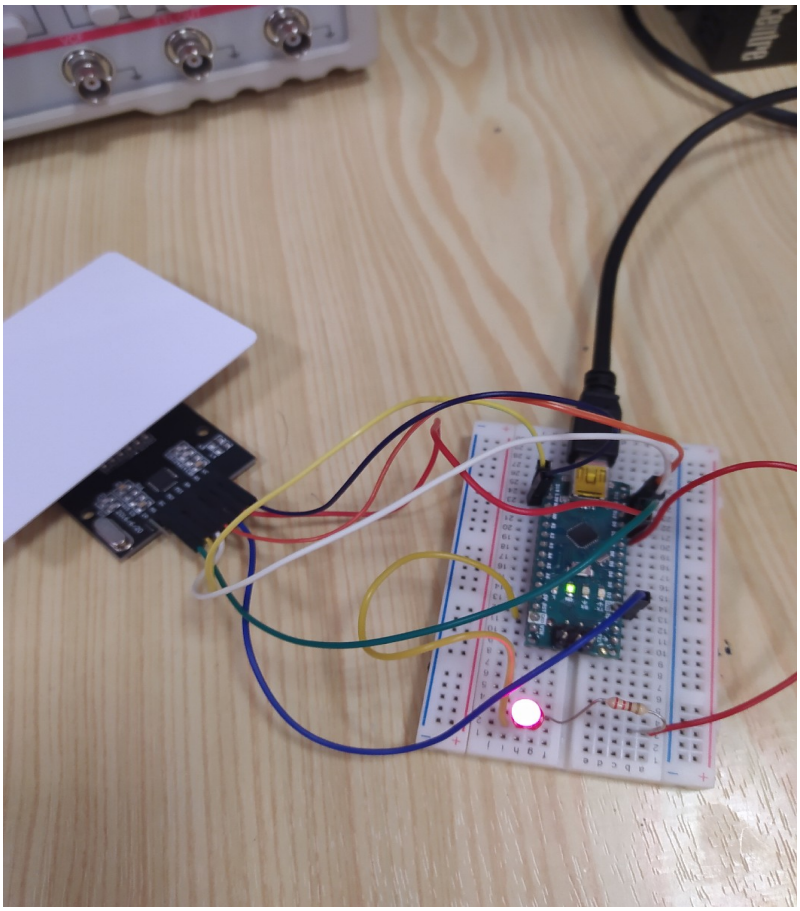
9. Ouverture des barrières avec RFID

9.1 Câblages



Ci-contre la carte Arduino avec un seul lecteur.

Disposition de la carte RFID par rapport au lecteur



9.2Programme

```
#include <SPI.h>
#include <RFID.h>
#include <Servo.h>
#include <Wire.h>


Servo myservo;  // create servo object to control a servo


Servo myservo2;  // create servo object to control a servo

int pos = 90;    // variable to store the servo

#define SS_PIN2 7
#define SS_PIN 10
#define RST_PIN 9

RFID rfid(SS_PIN, RST_PIN);
RFID rfid2(SS_PIN2, RST_PIN);

int power = 7;
int led = 8;
int serNum[5];

int cards[][15] = {{234, 54, 157, 46, 111}, {101, 2, 25, 211, 173} , {135, 246, 76, 1, 60}};

bool access = false;

unsigned char rfidRecu[5];
unsigned char rfidRecu2[5];

```

```
#include <SPI.h>
#include <RFID.h>
#include <Servo.h>
#include <Wire.h>


Servo myservo;  // create servo object to control a servo


Servo myservo2;  // create servo object to control a servo

int pos = 90;    // variable to store the servo

#define SS_PIN2 7
#define SS_PIN 10
#define RST_PIN 9

RFID rfid(SS_PIN, RST_PIN);
RFID rfid2(SS_PIN2, RST_PIN);

int power = 7;
int led = 8;
int serNum[5];

int cards[][15] = {{234, 54, 157, 46, 111}, {101, 2, 25, 211, 173} , {135, 246, 76, 1, 60}};

bool access = false;

unsigned char rfidRecu[5];
unsigned char rfidRecu2[5];


if (rfid.isCard()) {


  if (rfid.readCardSerial()) {
    Serial.print(rfid.serNum[0]);
    Serial.print(" ");
    Serial.print(rfid.serNum[1]);
    Serial.print(" ");
    Serial.print(rfid.serNum[2]);
    Serial.print(" ");
    Serial.print(rfid.serNum[3]);
    Serial.print(" ");
    Serial.print(rfid.serNum[4]);
    Serial.println("");
```

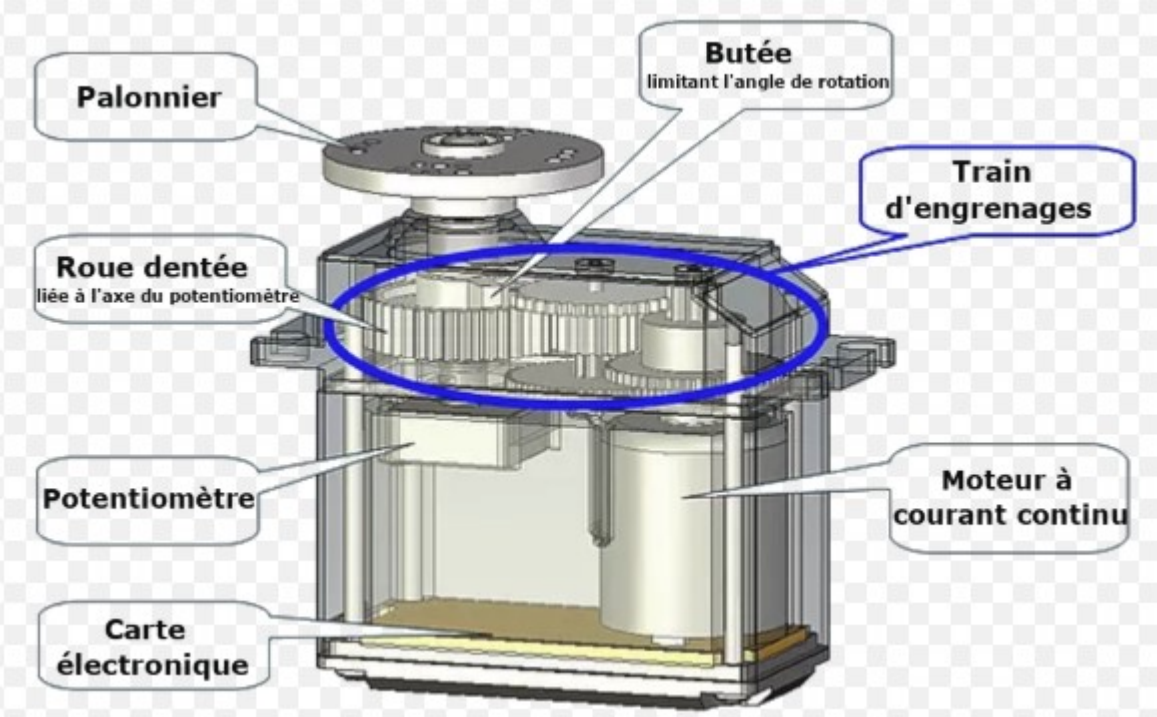
Les données reçues par les lecteurs sont inscrites dans un tableau

Ci-dessus, on distingue l'ouverture d'une barrière avec une temporisation de 2 secondes où la barrière reste en position ouverture (verticale) avant de redescendre, permettant donc, dans un cas concret à un véhicule de pouvoir passer .

10. Partie physique

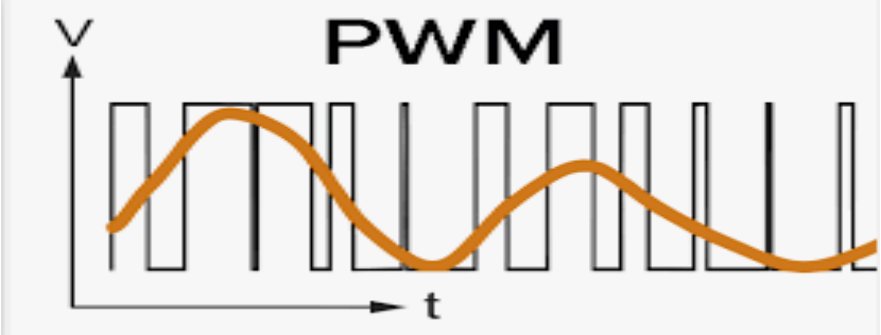
10.1 Servomoteur

Le servomoteur est composé; d'un moteur DC, d'un potentiomètre servant à déterminer son angle de rotation ainsi que de son système d'engrenage, et surtout d'une carte électronique intégrée.

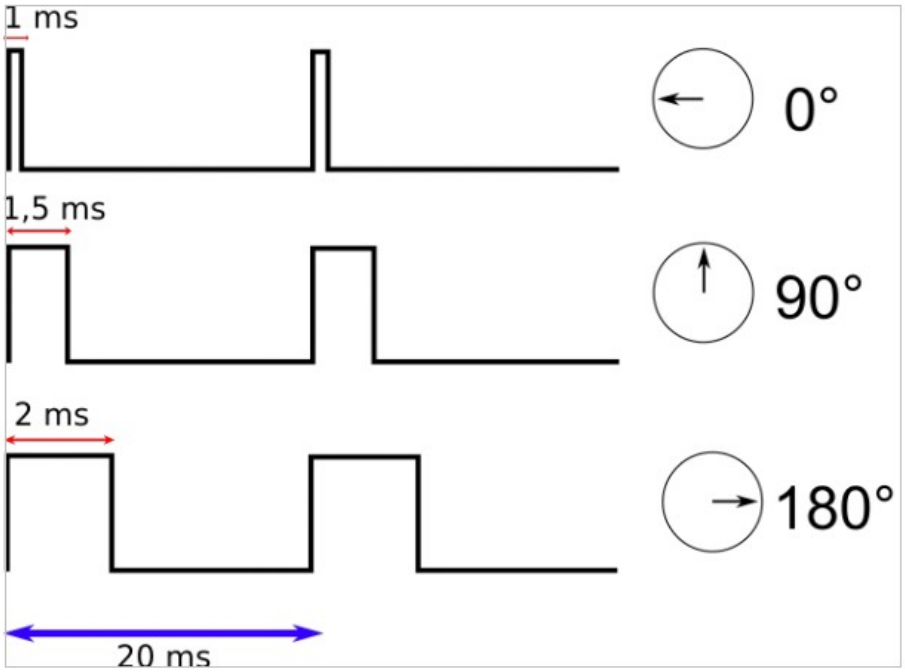


Lorsque l'on parle de servomoteur il faut également parler du PWM :
L'une des particularités du moteur à courant continu est son fonctionnement en PWM (**Pulse Width Modulation**).

Dans le cas de l'Arduino,il s'agit de succession de tension de 0/5V, qui permet, en modifiant la durée de chacun de ces états, de simuler une source de tension continue.
Si l'on fait une PWM avec 20% d'état haut, on aura l'équivalent en valeur moyenne de 20% de 5V : on aura donc 1V.



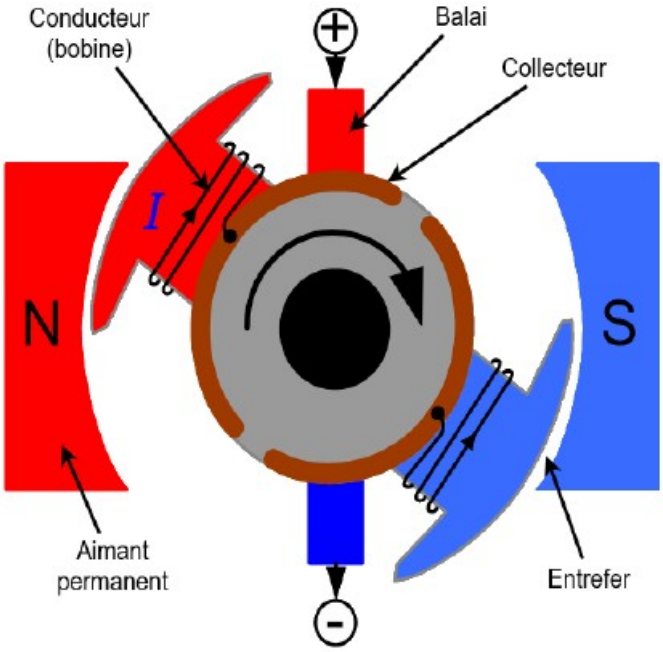
Ci-dessous, un exemple de l'angle pris par le servomoteur en fonction de la durée du signal.
Ici le cycle du signal étant de 20ms



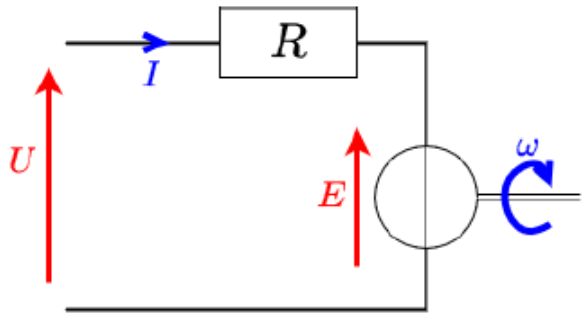
Ci-dessous un exemple de l'asservissement du moteur par le programme Arduino

```
for (pos = 106; pos >= 6; pos -= 1) { // goes from 106 degrees to 6 degrees
  // in steps of 1 degree
  myservo.write(pos);              // tell servo to go to position in variable 'pos'
  delay(20);                       // waits 20ms for the servo to reach the position
```

Cependant il ne faut pas oublier le moteur à courant continu:
Le principe de fonctionnement d'un moteur à courant continu repose sur la création d'un couple électromagnétique sur un rotor porteur de conducteurs parcourus par un courant continu, et qui se déplacent dans un champ magnétique. Un inducteur (stator) crée un champ magnétique de direction fixe.



Un moteur à courant continu peut être modélisé par le circuit équivalent suivant :



10.2 RFID

Il existe 2 types de RFID :

Le RFID passif : Le tag passif est un tag sans batterie, placé dans le champ électromagnétique du lecteur, la puce du tag est alimentée par couplage électromagnétique, puis des données peuvent être échangées entre puce et lecteur. Un simple numéro peut être lu, mais on peut également procéder à des échanges sécurisés de données comme c'est le cas pour les badges de contrôle d'accès en entreprise.

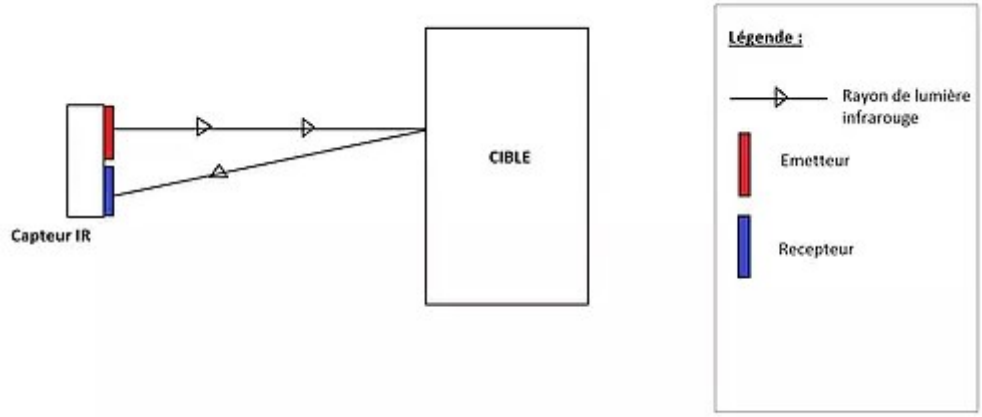
Le RFID actif : En ajoutant une batterie à un tag, il peut devenir une balise autonome et émettre à intervalle de temps régulier son identifiant et par exemple sa température. Grâce à sa batterie, la communication radio est de grande qualité et la distance au récepteur bien plus importante qu'un tag passif (dont le signal peut facilement être entravé par un élément métallique au sein d'un hangar).
Le gros avantage du tag actif est la grande distance de lecture observable.

En effet, grâce à sa batterie, les échanges d'informations peuvent se faire jusqu'à 100 m de distance.
On peut ainsi imaginer un hangar muni de récepteur en réseau sur un maillage de 50 mètres de côté et un tag actif
posé sur des biens de consommation comme des machines à laver ou des réfrigérateurs.

Voici la représentation d'un TAG RFID

En ce qui nous concerne, notre TAG RFID sera alors le badge, qui sera évidemment un RFID passif

10.3 capteurs infrarouge



Les capteurs infrarouges que nous allons implanter dans notre système, sont capables de détecter la distance entre deux objets. Ils sont constitués d'un récepteur qui détecte l'intensité lumineuse dans la gamme des lumières infrarouge et d'un émetteur de lumière infrarouge. S'il n'y a aucun obstacle devant le capteur, alors la valeur qu'il lira (tension) restera constante. Ainsi, si un obstacle est détecté, la valeur augmentera.

11. Conclusion

Le projet qui m'a été confié en est au stade final, c'est-à-dire qu'il ne me reste qu'à tester ma carte et finaliser mon code pour dire d'avoir terminé, c'est-à-dire avoir répondu au cahier des charges initial.

Ci-après les points clefs composants mon projet :

- l'assimilation du besoin de départ

- l'analyse complète et méthodique d'un cahier des charges

- la détermination et l'utilisation des ressources matériel et logiciel, mes choix techniques

- se situer dans le temps, pour évaluer l'avancement de son projet

- le travail en équipe, introduire dans ses réflexions les contraintes de son équipe

- les différentes procédures de test

Partie Étudiant EC 2 – Équipe 01 : MARTINELLI Loic

1-Introduction

1.1-Cahier des charges

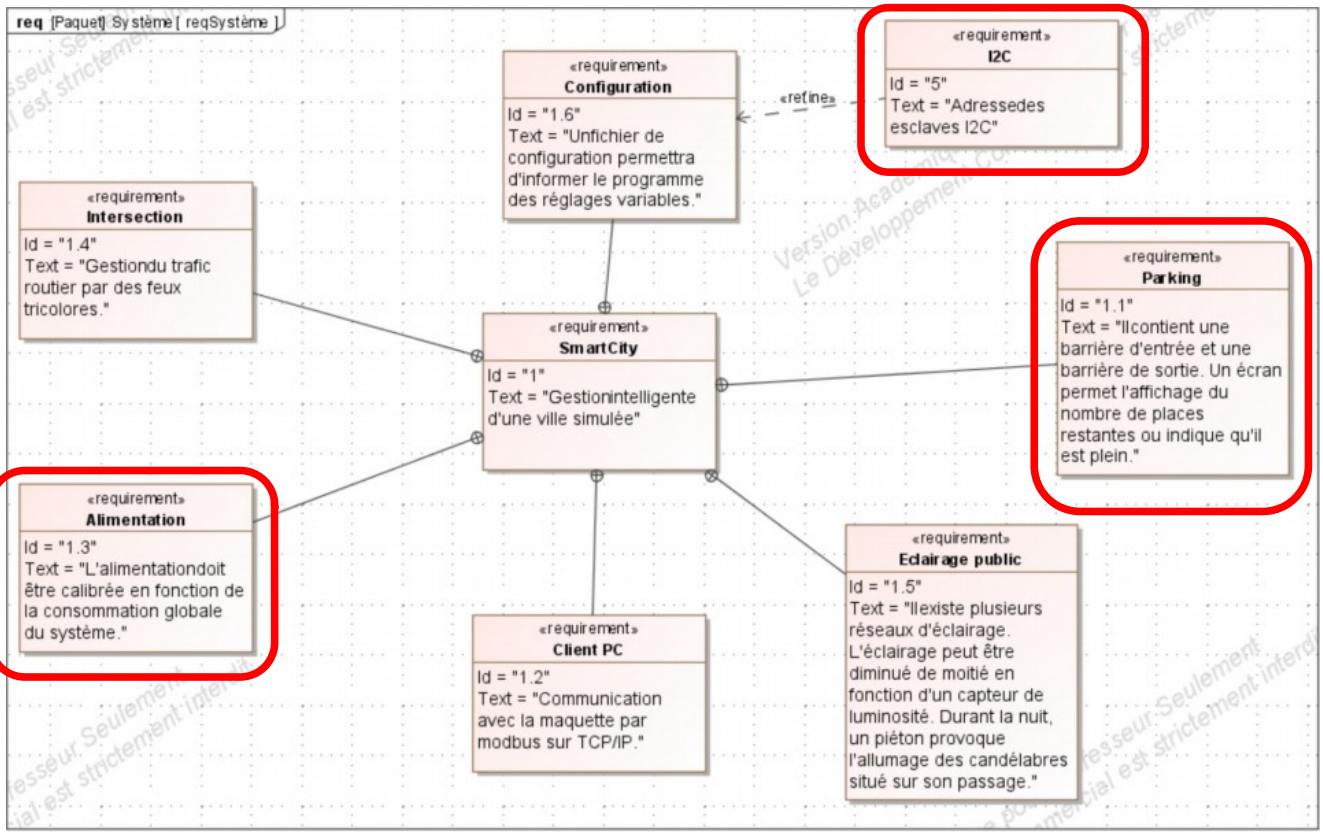
Étudiant 4 **EC 12**	*Liste des tâches assurées par l'étudiant* **Accès parking version ATtiny3217** • Concevoir une carte de gestion du parking *(identification des usagers, commande des barrières, gestion des appels d'urgence)* gérée par un microcontrôleur ATtiny3217 de nouvelle génération *(mega Core)*, à partir d'un schéma proposé. • Travail en collaboration avec l'étudiant EC11 dont l'objectif est identique, mais avec un microcontrôleur différent. • Effectuer la mise au point en comparant les IDE MPLAB X et Ardino, ainsi que les cartes de développement correspondantes. • Effectuer tous les tests nécessaires pour valider la structure, en la modifiant si nécessaire. • Participer à l'élaboration d'un protocole de communication sur le bus I2C avec l'étudiant IR concerné. • Participer à la conception des parties mécaniques du parking. • Effectuer un travail de documentation afin de déterminer une structure permettant de piloter une vraie barrière, et si possible l'intégrer sur la carte. • Effectuer la saisie du schéma et le routage de la solution retenue. Produire les fichiers Gerber afin que la fabrication du PCB soit sous-traitée. • Câbler la carte et effectuer les essais. • Documenter la mise en service de la carte finalisée.	**Installation :** IDE Microchip le mieux adapté au circuit, sous Windows associé à une carte de développement pour microcontrôleur ATtiny de type Curiosity Nano. IDE Arduino avec librairies megaTinyCore. **Mise en œuvre :** Tester/valider/modifier une structure utilisant un microcontrôleur ATtiny3217 pour gérer le parking afin de piloter les servo-moteurs des barrières, et gérer les capteurs de badges RFID pour identifier les usagers. Le microcontrôleur fonctionnera en tant que circuit esclave sur le bus I2C. L'analyse devra être menée conjointement avec l'étudiant EC11 dont la carte est gérée par un autre circuit. Prévoir la disposition de l'afficheur LCD avec lui également. Suite aux essais, modifier si nécessaire le schéma structurel proposé. Participer à une réflexion commune avec tous les étudiants EC du projet SmartCity pour le câblage du faisceau distribuant l'alimentation et le bus I2C sur tout le démonstrateur. **Réalisation :** Après validation de la solution, concevoir un circuit imprimé devant être fabriqué industriellement. Mettre en œuvre le faisceau de communication/alimentation. **Documentation :** • Avantages / inconvénients de cette solution par rapport à celle d'une carte avec Arduino Nano. • Avantages / inconvénients des environnements de travail et des cartes de développement pour ATtiny3217. • Schéma de câblage rapide (Fritzing) pour documenter la phase d'essais. • Documents de fabrication de la carte (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. • Schéma structurel avec contours IBD. • Liste complète des composants avec leurs sources d'approvisionnement et leur prix. • Programme de gestion de la carte, accompagné des commentaires et diagrammes nécessaires à sa compréhension. • Fiche de mise en service. • Fiche de dépannage.

1.2-Présentation

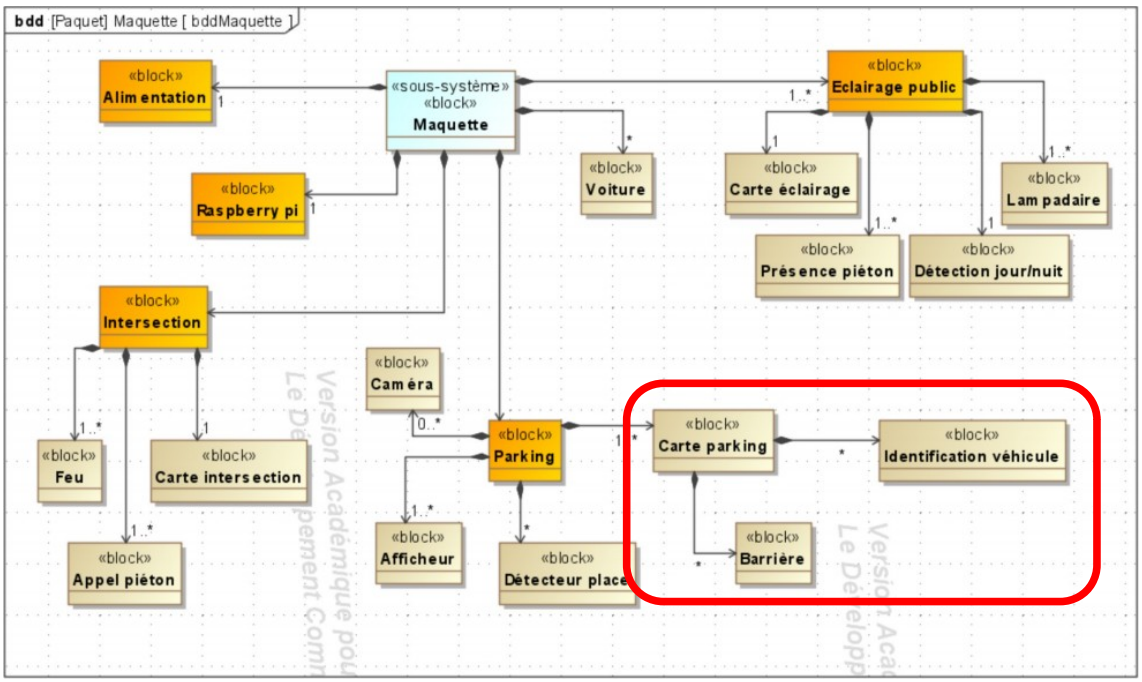
Dans le projet Smart City je m'occupe de l'entrée et la sortie du parking. C'est à dire le contrôle de barrières grâce à des cartes RFID et l'envoi de donné par liaison I2C à une carte Raspberry pi. A la différence de l'EC 11 la gestion des barrières ce fera avec un microcontrôleur ATtiny3217.

2-Diagrammes

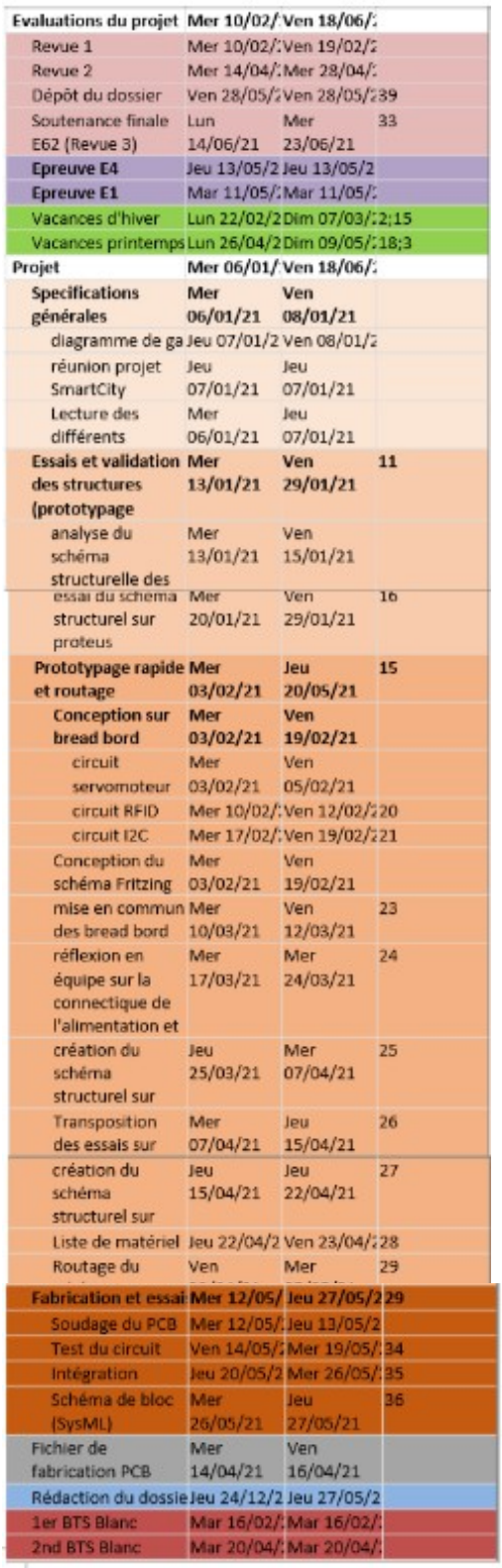
2.1-Diagramme des exigences



2.2-Diagramme de bloc



2.3-Diagramme de Gantt

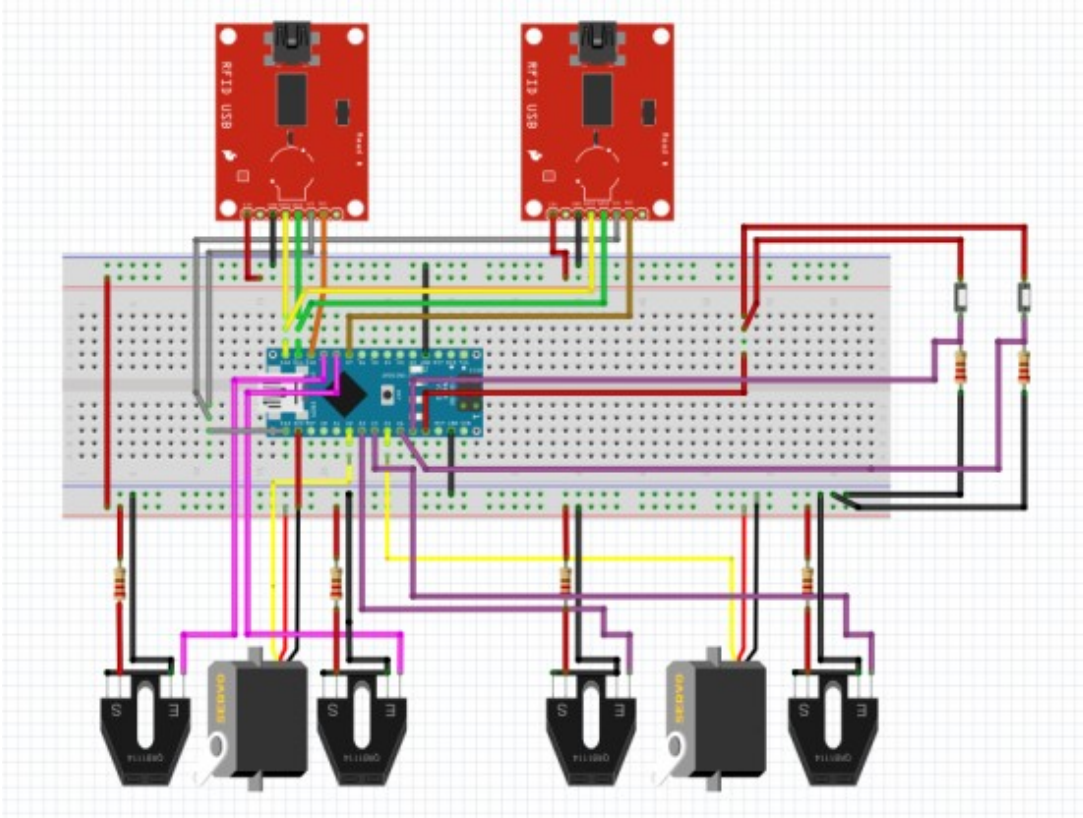


Evaluations du projet	**Mer 10/02/**	**Ven 18/06/**	
Revue 1	Mer 10/02/	Ven 19/02/	
Revue 2	Mer 14/04/	Mer 28/04/	
Dépôt du dossier	Ven 28/05/	Ven 28/05/	39
Soutenance finale E62 (Revue 3)	Lun 14/06/21	Mer 23/06/21	33
Epreuve E4	Jeu 13/05/2	Jeu 13/05/2	
Epreuve E1	Mar 11/05/	Mar 11/05/	
Vacances d'hiver	Lun 22/02/2	Dim 07/03/	2;15
Vacances printemps	Lun 26/04/2	Dim 09/05/	18;3
Projet	**Mer 06/01/**	**Ven 18/06/**	
Specifications générales	**Mer 06/01/21**	**Ven 08/01/21**	
diagramme de ga	Jeu 07/01/2	Ven 08/01/	
réunion projet SmartCity	Jeu 07/01/21	Jeu 07/01/21	
Lecture des différents	Mer 06/01/21	Jeu 07/01/21	
Essais et validation des structures (prototypage	**Mer 13/01/21**	**Ven 29/01/21**	**11**
analyse du schéma structurelle des	Mer 13/01/21	Ven 15/01/21	
essai du schema structurel sur proteus	Mer 20/01/21	Ven 29/01/21	16
Prototypage rapide et routage	**Mer 03/02/21**	**Jeu 20/05/21**	**15**
Conception sur bread bord	**Mer 03/02/21**	**Ven 19/02/21**	
circuit servomoteur	Mer 03/02/21	Ven 05/02/21	
circuit RFID	Mer 10/02/	Ven 12/02/	20
circuit I2C	Mer 17/02/	Ven 19/02/	21
Conception du schéma Fritzing	Mer 03/02/21	Ven 19/02/21	
mise en commun des bread bord	Mer 10/03/21	Ven 12/03/21	23
réflexion en équipe sur la connectique de l'alimentation et	Mer 17/03/21	Mer 24/03/21	24
création du schéma structurel sur	Jeu 25/03/21	Mer 07/04/21	25
Transposition des essais sur	Mer 07/04/21	Jeu 15/04/21	26
création du schéma structurel sur	Jeu 15/04/21	Jeu 22/04/21	27
Liste de matériel	Jeu 22/04/2	Ven 23/04/	28
Routage du	Ven	Mer	29
Fabrication et essai	**Mer 12/05/**	**Jeu 27/05/2**	**29**
Soudage du PCB	Mer 12/05/	Jeu 13/05/2	
Test du circuit	Ven 14/05/	Mer 19/05/	34
Intégration	Jeu 20/05/2	Mer 26/05/	35
Schéma de bloc (SysML)	Mer 26/05/21	Jeu 27/05/21	36
Fichier de fabrication PCB	Mer 14/04/21	Ven 16/04/21	
Rédaction du dossie	Jeu 24/12/2	Jeu 27/05/2	
1er BTS Blanc	Mar 16/02/	Mar 16/02/	
2nd BTS Blanc	Mar 20/04/	Mar 20/04/	

3-Réalisation du montage

3.1-Test du montage arduino

Pour débuter le projet j'ai aidé Duboc Lucas à câbler le montage du parking avec une carte Arduino Nano pour pouvoir le transposer plus tard avec un microcontrôleur ATtiny3217. Nous avons donc fait des montage en câblage rapide pour tester les composants comme les servomoteurs ou les lecteurs RFID d'abord séparément puis ensemble. A la fin de nos test nous avons abouti à un montage fonctionnel.



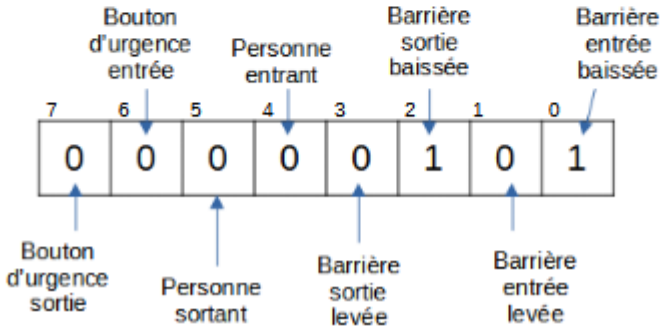
3.2-Protocole I2C parking

Une fois que le montage fonctionné correctement je me suis détaché de Duboc Lucas pour m'occuper de la partie liaison I2C. Je suis donc aller parlé avec Pichery Florian pour établir un protocole.

Protocole I2C Parking

-Le maître envoi l'adressage (50) en mode lecture.
-L'esclave envoi 11 octets.

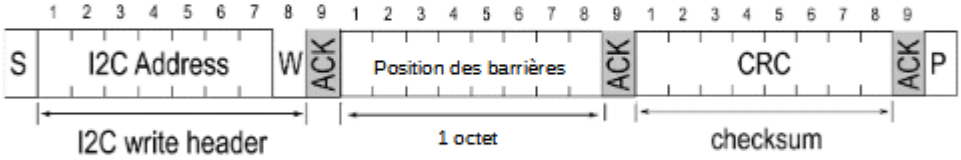
États barrières :

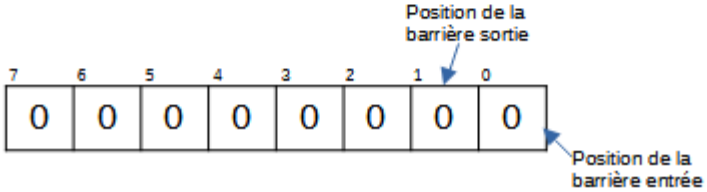


Cet octet correspond à l'état initiale des barrières. C'est à dire que les barrières de sortie et d'entrée sont baissées.

RFID entrée et sortie :
Ils commencent par l'octet de poids fort et finissent par l'octet de poids faible.

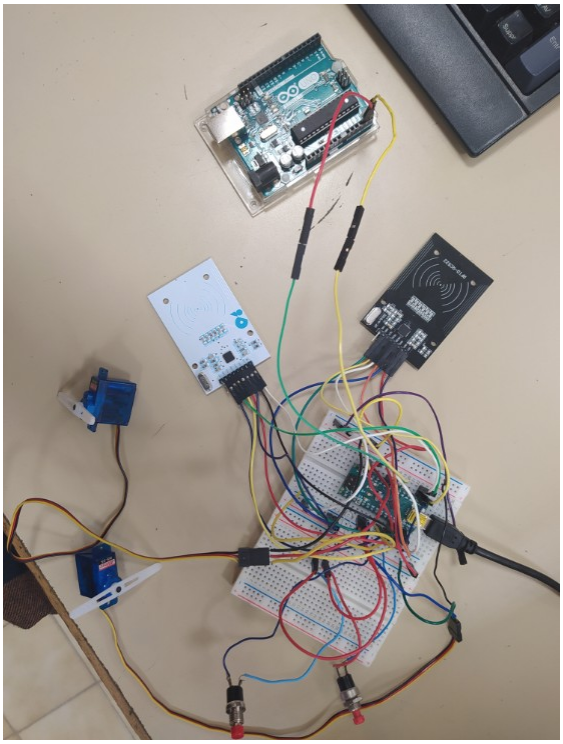
-Le maître envoi l'adressage (50) en mode écriture et définit la position des barrières.





3.3-I2C Arduino à Arduino

Pour apprendre à communiquer en I2C j'ai commencé par câbler le montage avec une arduino UNO. Après avoir trouvé les codes maître et esclave pour communiquer en I2C entre arduino je l'ai modifié petit à petit pour correspondre à la trame que j'ai créée dans le protocole I2C du parking.



3.3.1-code maître

```
#include <Wire.h>

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

void loop() {
  Wire.requestFrom(50, 11);
  while (Wire.available()) {
    int  c = Wire.read();
    Serial.println(c, DEC);
    delay(400);
  }
  delay(1000);
}
```

Ceci est le programme du maître.
Il permet d'envoyer une lecture à l'adresse 0x50 pour recevoir les 11 octets.
Ils sont affiché en décimal sur le moniteur série du maître.

3.3.2-code esclave

```
#include <SPI.h>
#include <RFID.h>
#include <Servo.h>
#include <Wire.h>

int oct1 = 0x0A;
unsigned char rfidRecu[5];
unsigned char rfidRecu2[5];

Servo myservo;
Servo myservo2;
void setup() {
  pinMode(buttonPin, INPUT);
  pinMode(buttonPin2, INPUT);

  Wire.begin(50);
  Wire.onRequest(requestEvent);

  Serial.begin(9600);
```

oct1 est le premier octet que j'envoie avec l'esclave. Sa valeur correspond à l'état de base des barrières.

Les tableaux rfidRecu et rfidRecu2 permettent de contenir les adresses des carte RFID présenté au capteur RFID

Wire.begin permet de démarrer la communication à l'adresse 0x50.

Wire.onRequest enregistre la fonction requestEvent à appeler quand le maître demande des données à l'esclave

```
void loop() {
  if (rfid.isCard()) {
    if (rfid.readCardSerial()) {
      Serial.print(rfid.serNum[0]);
      Serial.print(" ");
      Serial.print(rfid.serNum[1]);
      Serial.print(" ");
      Serial.print(rfid.serNum[2]);
      Serial.print(" ");
      Serial.print(rfid.serNum[3]);
      Serial.print(" ");
      Serial.print(rfid.serNum[4]);
      Serial.println("");
      rfidRecu[0]= (unsigned char) rfid.serNum[0];
      rfidRecu[1]= (unsigned char) rfid.serNum[1];
      rfidRecu[2]= (unsigned char) rfid.serNum[2];
      rfidRecu[3]= (unsigned char) rfid.serNum[3];
      rfidRecu[4]= (unsigned char) rfid.serNum[4];
```

J'assigne chaque octet de l'adresse RFID à différente partie du tableau rfidRecu pour les envoyer au maître. J'ai fait la même manipulation pour le deuxième RFID.

```
  if (access) {
    Serial.println("Yes");
    for (pos = 90; pos <= 180; pos += 1) {
      // in steps of 1 degree
      myservo.write(pos);
      delay(5);

      octl &= 0xfd;
      octl |= 0x1;
    }
    delay(2000);
    for (pos = 180; pos >= 90; pos -= 1) {
      myservo.write(pos);
      delay(5);

      octl &= 0xfe;
      octl |= 0x2;
    }
    octl |= 0x10;
  }
```

Les fonctions & (et) et | (ou) me permettent de positionner chaque bits de l'octet oct1 indépendamment

```
void requestEvent() {
buttonState = digitalRead(buttonPin);
buttonState2 = digitalRead(buttonPin2);
  if (buttonState == HIGH) {
    octl &= 0xbf;
    octl |= 0x40;
  }
  if (buttonState2 == LOW) {
    octl &= 0x7f;
    octl |= 0x80;
  }
  Wire.write(octl);
      Wire.write( rfidRecu[0] );
       Wire.write( rfidRecu[1] );
        Wire.write( rfidRecu[2] );
         Wire.write( rfidRecu[3] );
          Wire.write( rfidRecu[4] );

 Wire.write( rfidRecu[0] );
       Wire.write( rfidRecu2[1] );
        Wire.write( rfidRecu2[2] );
         Wire.write( rfidRecu2[3] );
          Wire.write( rfidRecu2[4] );
```

Quand le maître demande à l'esclave un envoi de donnée la fonction requestEvent démarre. Elle regarde l'état des boutons d'urgence pour les mettre ou non dans oct1.

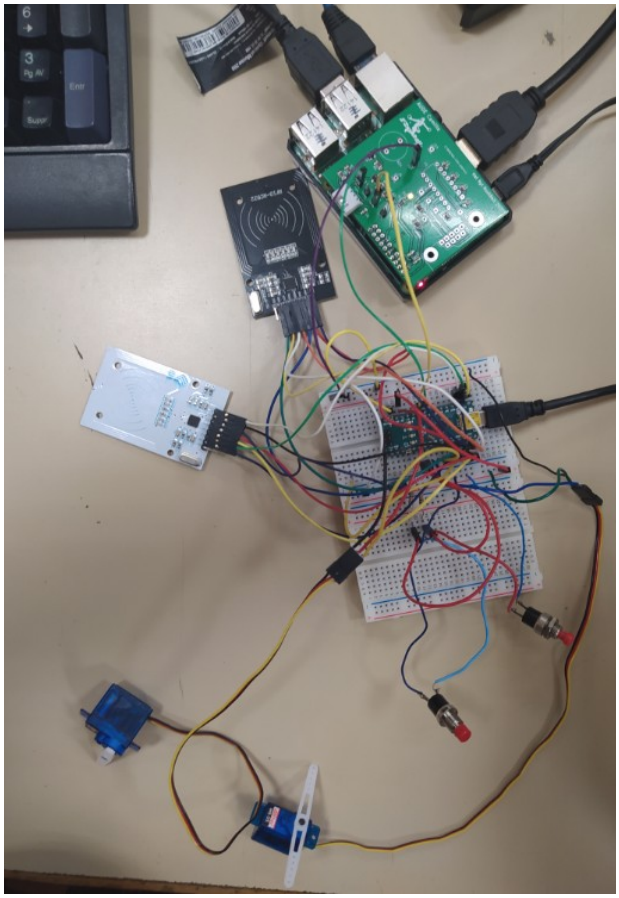
Elle envoie ensuite tous les octets au maître.

```
  rfidRecu[0] =0 ;
  rfidRecu[1] =0 ;
  rfidRecu[2] =0 ;
  rfidRecu[3] =0 ;
  rfidRecu[4] =0 ;

  rfidRecu2[0] =0 ;
  rfidRecu2[1] =0 ;
  rfidRecu2[2] =0 ;
  rfidRecu2[3] =0 ;
  rfidRecu2[4] =0 ;
   oct1 &= 0x0f;
}
```

Pour ne pas que les adresses des cartes RFID présentées au capteur restent on les remet à 0 de même que l'état des boutons d'urgence dans oct1.

3.4-I2C Arduino à Raspberry pi

Après avoir fait les test entre cartes arduino j'ai tester la communication I2C entre l'arduino Nano et une Raspberry pi. Pour permettre cette communication j'ai du mettre un level shifter sur la Raspberry car la carte arduino délivre du 5V et que la Raspberry ne supporte que du 3,3V.

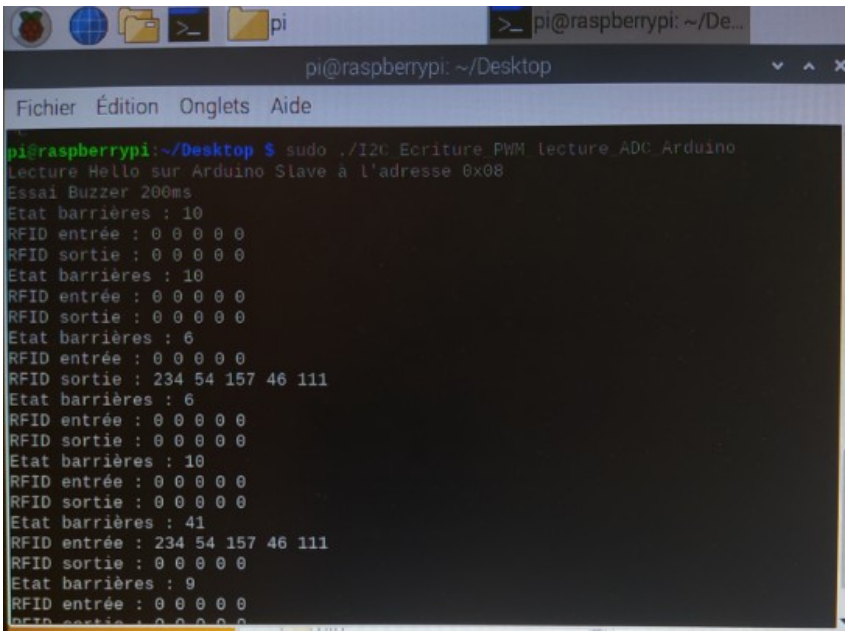


3.4.1-Programme

```
int main(void)
{

    unsigned int slave_address=0x50; // Adresse Panneau
    char i2cOut[6];      // Tableau des données I2C à sortir
    char i2cIn[6];       // Tableau des données I2C entrées
    double barrieres;
    double RFIDE1;
    double RFIDE2;
    double RFIDE3;
    double RFIDE4;
    double RFIDE5;
    double RFIDS1;
    double RFIDS2;
    double RFIDS3;
    double RFIDS4;
    double RFIDS5;

    init_I2c_bcm2835();
    bcm2835_i2c_setSlaveAddress(slave_address); // Affectation de l'adresse de l'esclave
    bcm2835_gpio_fsel(BUZZER, BCM2835_GPIO_FSEL_OUTP); // Broche BUZZER (broche 13 = GPIO27) en sortie
        bcm2835_gpio_write(BUZZER, LOW); //Arrêt Buzzer

    cout<<"Lecture Hello sur Arduino Slave à l'adresse 0x50"<<endl;

    delay(1000);                    //
    cout<<"Essai Buzzer 200ms"<<endl;
        bcm2835_gpio_write(BUZZER, HIGH); //BUZZER = 1
    delay(200);
        bcm2835_gpio_write(BUZZER, LOW); //BUZZER = 1
while(1)
    {

//Lecture

bcm2835_i2c_read(i2cIn,11);
barrieres = (double)i2cIn[0];

RFIDE1 = (double)i2cIn[1];
RFIDE2 = (double)i2cIn[2];
RFIDE3 = (double)i2cIn[3];
RFIDE4 = (double)i2cIn[4];
RFIDE5 = (double)i2cIn[5];

RFIDS1 = (double)i2cIn[6];
RFIDS2 = (double)i2cIn[7];
RFIDS3 = (double)i2cIn[8];
RFIDS4 = (double)i2cIn[9];
RFIDS5 = (double)i2cIn[10];


cout<<"Etat barrières : "<<barriere <<endl;
cout<<"RFID entrée : "<<RFIDE1 <<" "<<RFIDE2 <<" "<<RFIDE3 <<" "<<RFIDE4 <<" "<<RFIDE5 <<endl;
cout<<"RFID sortie : "<<RFIDS1 <<" "<<RFIDS2 <<" "<<RFIDS3 <<" "<<RFIDS4 <<" "<<RFIDS5 <<endl;



    delay(1000);
    }

bcm2835_close();
return 0;
```

L'esclave étant toujours une carte arduino je n'ai pas eu besoin de changer son code. En revanche pour le maître j'ai pris un code déjà fait et je l'ai adapté à ma situation.
Sur Raspberry pi le maître peut prendre chaque octet indépendamment grâce à un tableau j'ai donc initialisé l'octet d'état des barrières et chaque octet des adresses RFID pour pouvoir afficher toute l'adresse d'une carte sur une ligne.

3.4.2-visualisation



Quand je lance mon programme le terminal donne l'adresse de l'esclave et donne successivement l'état des barrières, l'adresse RFID reçu en entrée et celle en sortie.

4-Matériels

4.1-Servomoteurs

Pour la barrière du parking nous avons utilisé un servomoteur qui nous permet de pouvoir contrôlé le moteur et de le laisser à une position précise grâce à sont électronique intégré. Nous nous sommes servis du Servomoteur HS-55 **qui est souvent utilisé dans des montages arduino mais un problème au moment où on le faisait s'arrêter pouvait le faire forcer c'est pour cela que nous avons changé pour un** Servomoteur MG 996R 180°. Ce dernier type de servomoteur peut ce trouver en 180° ou en 360° mais le 360° ne permet pas des s'arrêter à un angle précis comme l'on voudrait.

Servomoteur HS-55

Servomoteur MG 996R 180°

4.2-Lecteurs RFID

Pour contrôler la lever de la barrière à l'arrivée des voitures nous avons utilisé des lecteurs RFID Velleman VMA405. Le RFID utilisé est passif car les tags ne contiennent pas de source d'alimentation embarquée. Le lecteur va donc envoyer une onde électromagnétique en direction du tag qui va la capté et lui renvoyé les données stocké dans sa puce.
Chaque tags RFID ayant une adresse différente nous pouvons contrôler quel carte peut passer.

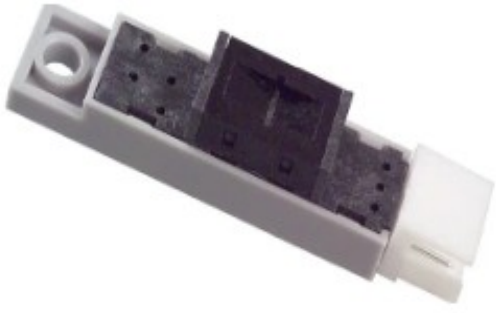
Lecteur RFID

Tag RFID

4.3-Capteurs infrarouge

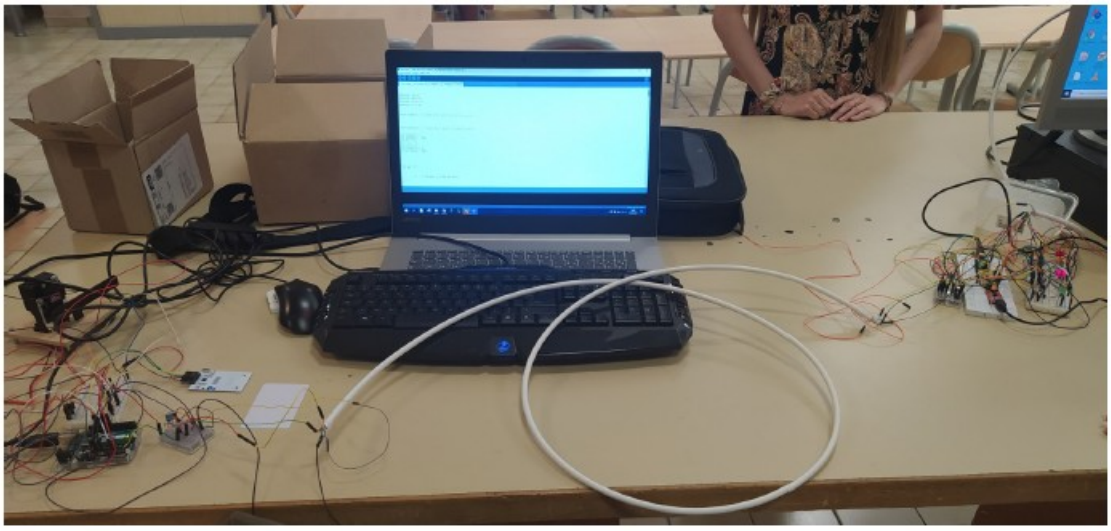
Au début pour connaître la position des barrières on utilisait le code mais cette façon de faire ne correspondait pas à la réalité. Nous avons donc décidé de rajouté deux capteur réfléchissant Sharp GP2A200LCS0F pour avoir l'information véritable de la position en bas ou en haut des barrières.
En comparant les capteur et le code on peut voir des anomalie comme une barrière cassée, un objet obstruant la vision du capteur ou encore un problème de positionnement des barrières.

Sharp GP2A200LCS0F



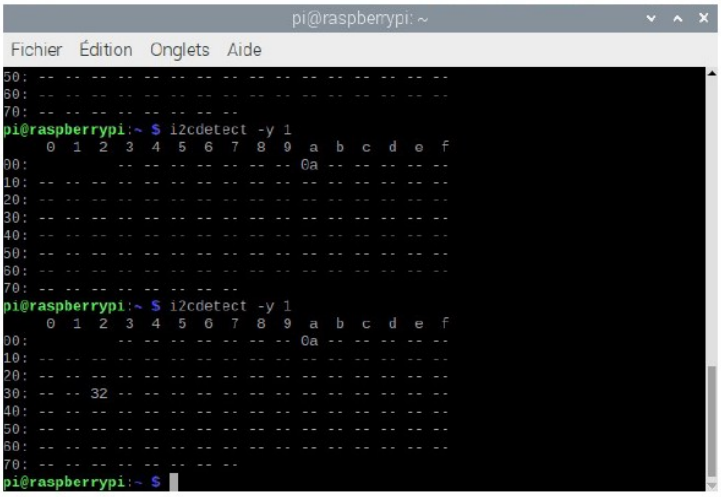
5-Liaison I2C entre plusieurs cartes

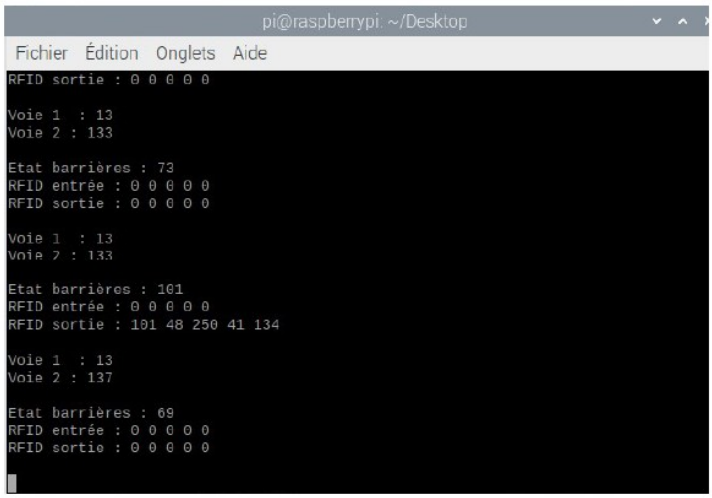
Étant donné que dans notre projet chaque partie doit communiquer en I2C en passant d'une à une autre avec une grande distance entre elles, nous avons décidé de mettre un level shifter sur chaque carte pour régénérer et stabilisé le signal transmis. Pour tester notre communication nous avons donc connecté deux montages, ici celui de Duboc Lucas (barrière parking) et Maronat Marine (feux d'intersection) en mettant entre les deux une grande longueur de câble pour finir sur une carte Raspberry pi ayant un programme similaire avec celui utilisé précédemment.



Nous pouvons voir que sur la console deux adresses I2C sont reconnu 0a et 32. Une fois le

programme lancé on peut voir les lignes voie 1 et voie 2 qui correspondent au montage de Marine et les trois dernières lignes qui correspondent au montage de Lucas. Les valeurs sont correctes ce qui montre la possibilité de connecté plusieurs cartes à la suite sur de grandes distances sans perturber les données envoyées.

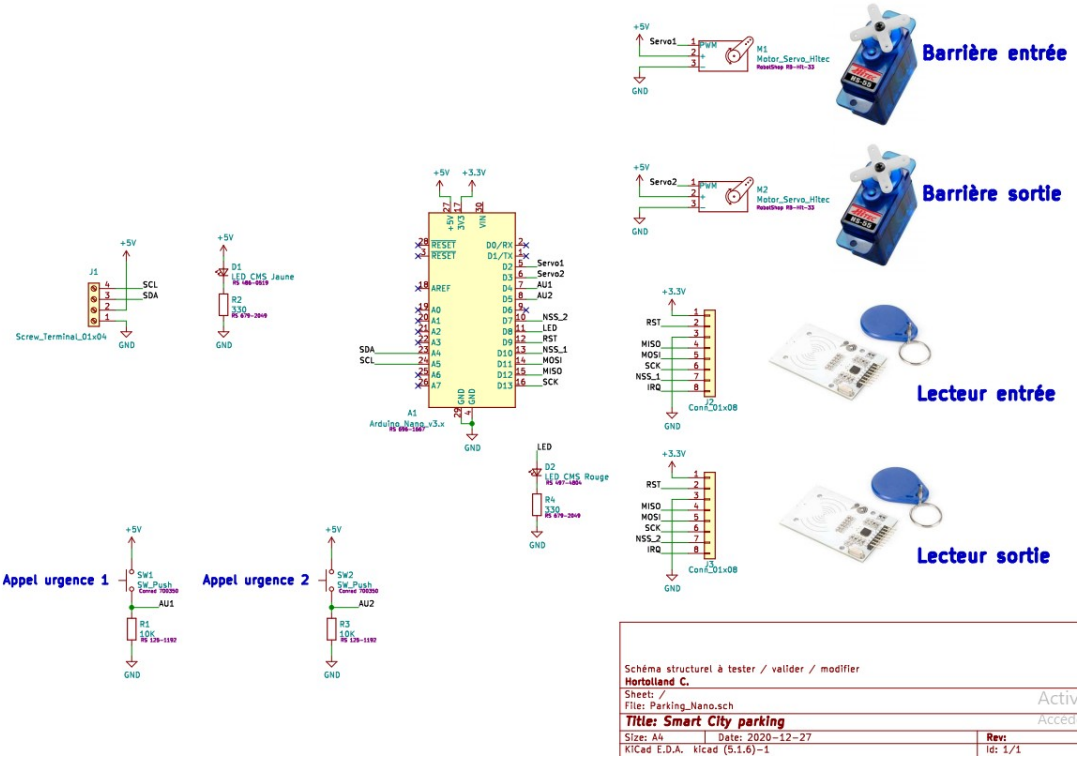




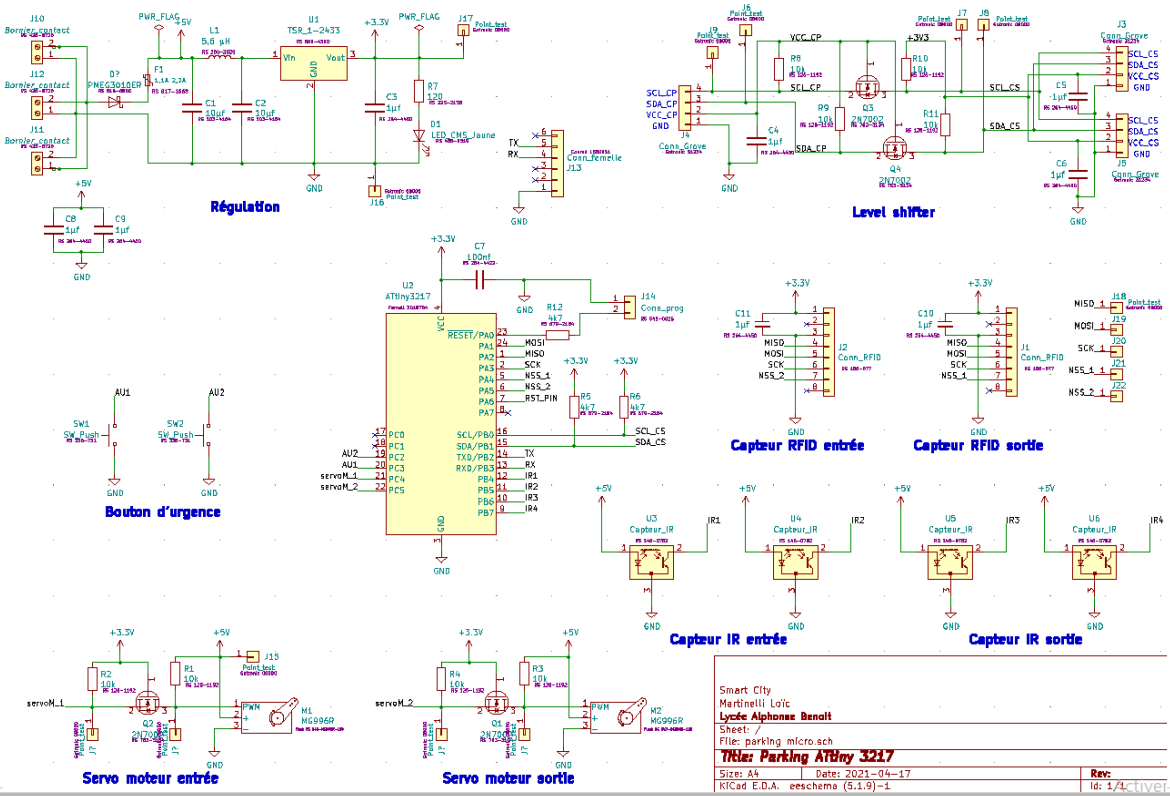
6-Réalisation du pcb

6.1-Schéma structurel

Pour commencer Lucas Duboc et moi avons pris exemple sur le schéma structurel fournit par le prof pour le parking gérer par une carte arduino Nano.

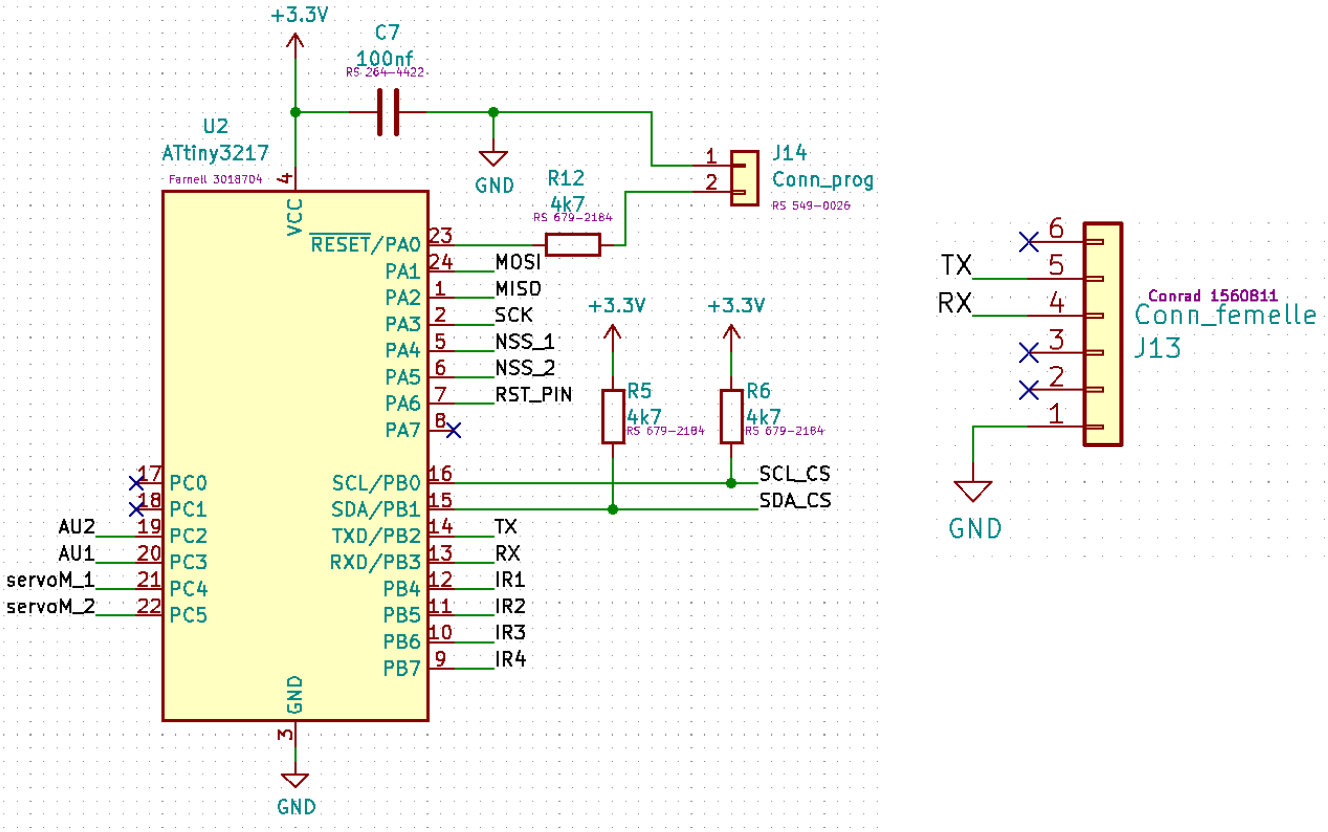


Ce schéma ne correspondant pas au circuit que je devais effectué je l'ai modifié presque entièrement



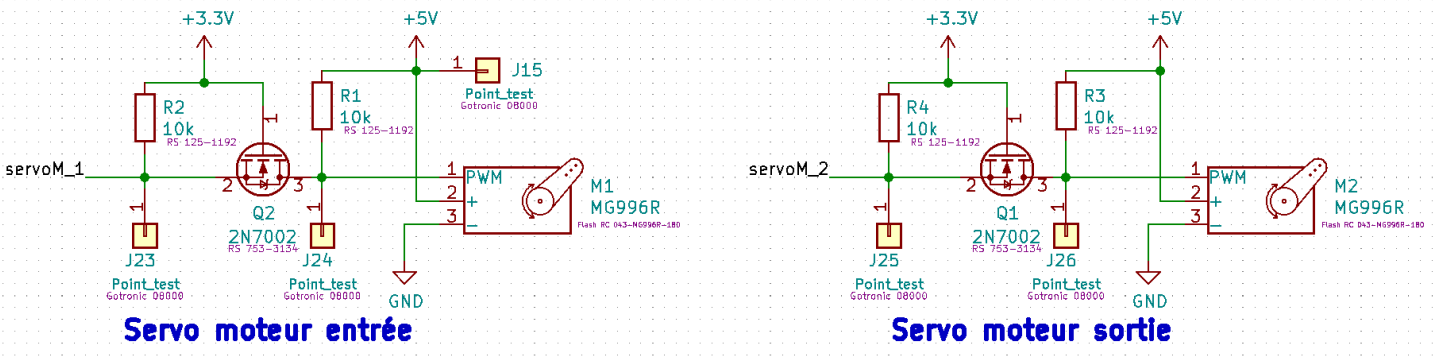
6.1.1-ATtiny 3217

Pour mon microcontrôleur j'ai mis un condensateur de découplage pour limiter les perturbation, deux broche pour pouvoir le programmer,un connecteur femelle six broches pour déboguer et rajouter des résistances de pull-up pour le SCL et SDA. L'assignation des broches n'a pas était compliqué car seule quelques broches comme MISO, MOSI, SCL, SDA, TX et RX avait des broches spécifique.



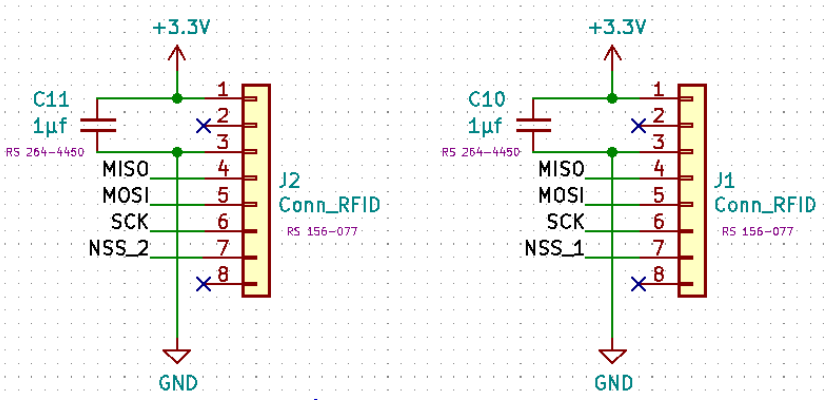
6.1.2-Servomoteur

Pour les servomoteurs étant donné qu'ils sont alimenté en 5v et que mon microcontrôleur est alimenté en 3,3v j'ai du mettre un abaisseur de tension.



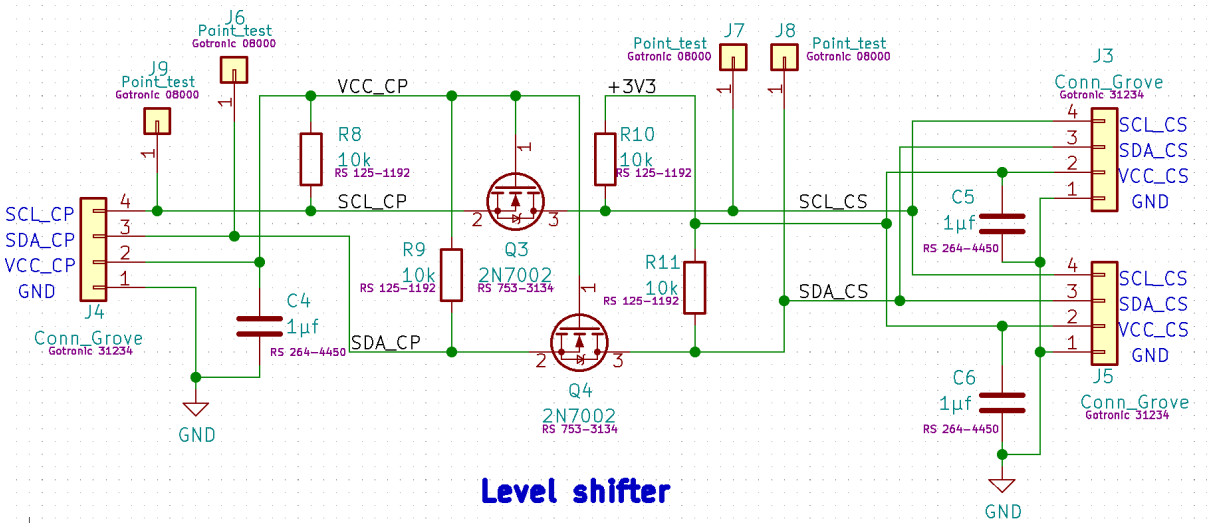
6.1.3-Capteur RFID

Le seul changement pour les capteur RFID est le rajout de condensateur de découplage.



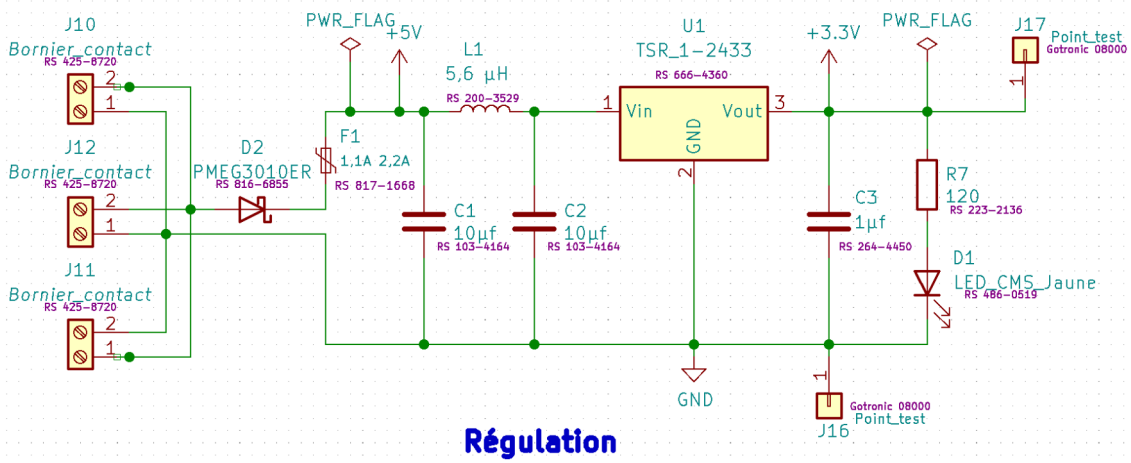
6.1.4-Level shifter

Pour la liaison I2C l'ajout d'un level shifter et de condensateur de découplage a été obligatoire pour stabilisé le signal.



6.1.5- Régulateur

L'alimentation passant par plusieurs carte l'ajout d'un régulateur de tension été indispensable pour éviter les fluctuation de tension.



6.2-Liste de matériel

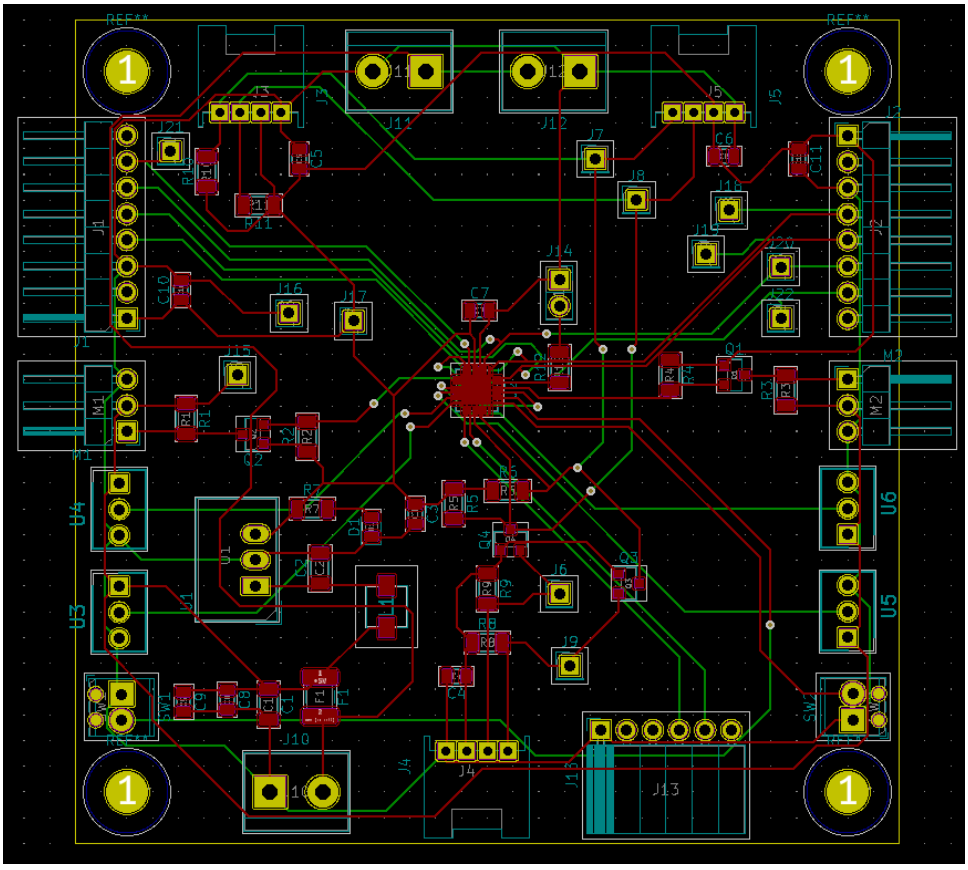
Qty	Reference(s)	Value	Footprint	Code Commande
2	C1, C2	10µf	Capacitor_SMD:C_1206_3216Metric	RS 103-4164
6	C3, C4, C5, C6, C8, C9	1µf	Capacitor_SMD:C_0805_2012Metric	RS 264-4450
1	C7	100nf	Capacitor_SMD:C_0805_2012Metric	RS 264-4422
1	D1	LED_CMS_Jaune	LED_SMD:LED_0805_2012Metric	RS 486-0519
1	F1	Polyfuse	0_ModulesProjet:Polyswitch_MICROSMD110F-2	RS 517-7105
2	J1, J2	Conn_RFID	Connector_PinHeader_2.54mm:PinHeader_1x08_P2.54mm_Horizonta	RS 156-077
3	J3, J4, J5	Conn_Grove	0_ModulesProjet:Embase_Grove_Horizontale	Gotronic 31234
12	J6, J7, J8, J9, J15, J16, J17, J18, J19, J20, J21, J22	Point_test	Connector_PinHeader_2.54mm:PinHeader_1x01_P2.54mm_Vertical	Gotronic 08000
3	J10, J11, J12	Bornier_contact	0_ModulesProjet:Bornier_5-08_2cts_Weidmuller_Noir	RS 425-8720
1	J13	Conn_femelle	Connector_PinSocket_2.54mm:PinSocket_1x06_P2.54mm_Horizonta	Conrad 1560811
1	J14	Conn_prog	Connector_PinSocket_2.54mm:PinSocket_1x02_P2.54mm_Vertical	RS 549-0026
1	L1	5,6 µH	0_ModulesProjet:TCK141	RS 200-3529
2	M1, M2	MG996R	Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Horizonta	Flash RC 043-MG996R-180
4	Q1, Q2, Q3, Q4	2N7002	Package_TO_SOT_SMD:SOT-23	RS 753-3134
8	R1, R2, R3, R4, R8, R9, R10, R11	10k	Resistor_SMD:R_1206_3216Metric	RS 125-1192
3	R5, R6, R12	4k7	Resistor_SMD:R_1206_3216Metric	RS 679-2184
1	R7	120	Resistor_SMD:R_1206_3216Metric	RS 223-2136
2	SW1, SW2	SW_Push	0_ModulesProjet:Bornier_Phoenix_Contact_x2cts_1725656	RS 336-731
1	U1	TSR_1-2433	Converter_DCDC:Converter_DCDC_TRACO_TSR-1_THT	RS 666-4360
1	U2	ATtiny3217	Package_DFN_QFN:QFN-24-1EP_4x4mm_P0.5mm_EP2.6x2.6mm	Farnell 3018704
4	U3, U4, U5, U6	Capteur_IR	0_ModulesProjet:Embase_AMPMODU_HE14_Angle_Droit_3	RS 146-0782

6.3-Routage

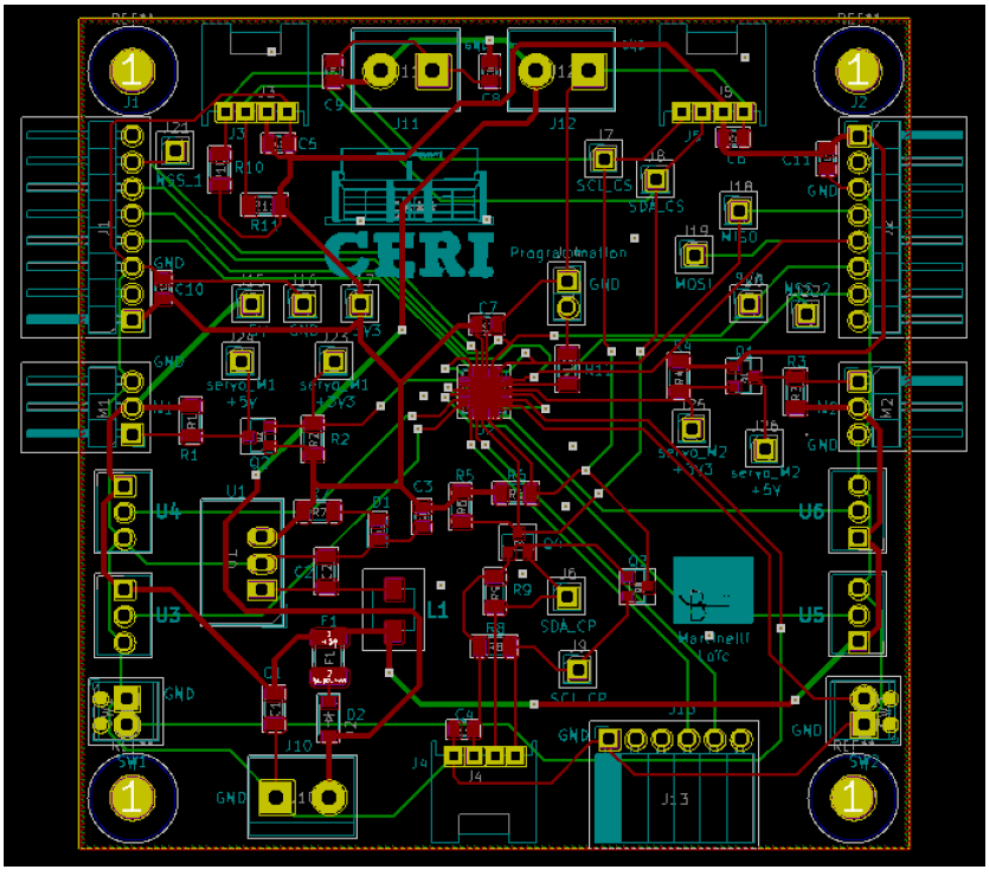
Après avoir fini le schéma j'ai commencé le routage de la carte en suivant certaine contrainte :

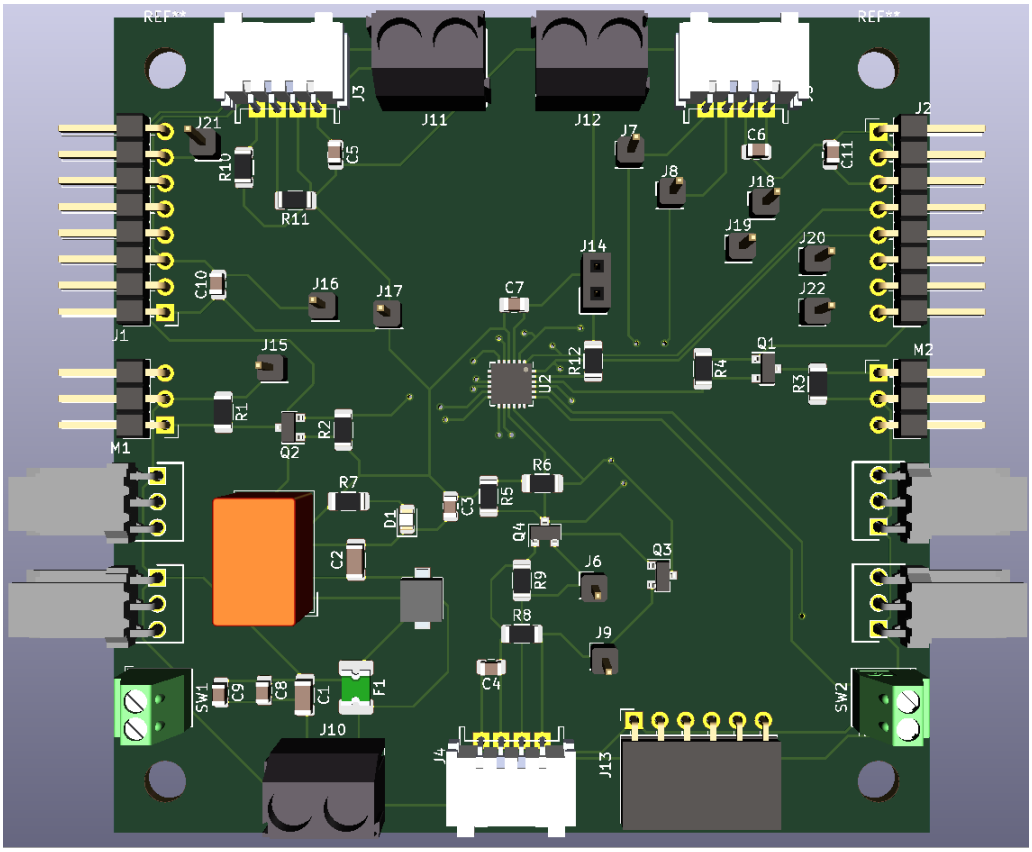
-le PCB ne doit pas dépasser 10x10cm pour ne pas augmenter les prix de la carte
- tout les CMS doivent ce trouver sur la même face pour avoir moins de problème lors de la brasure
- trous de fixation de 4cm dans chaque coins
-disposition judicieuse des borniers et connecteur pour réduire la longueur des fils

Le premier routage que j'ai effectué n'était pas bon car la contrainte des connecteur n'était pas respecté. Le dernier routage respecte les contrainte mais certains éléments comme le plan de masse, les références, l'ajout du nom du projet et des logos manque encore.



Le routage final est l'évolution du précédent avec les corrections dites juste avant et le placement plus judicieux de certains composants comme les condensateurs de découplages.

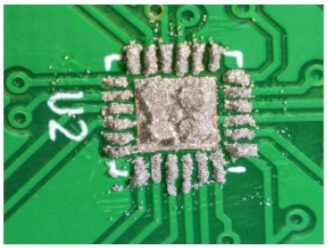


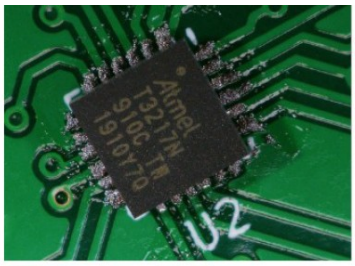


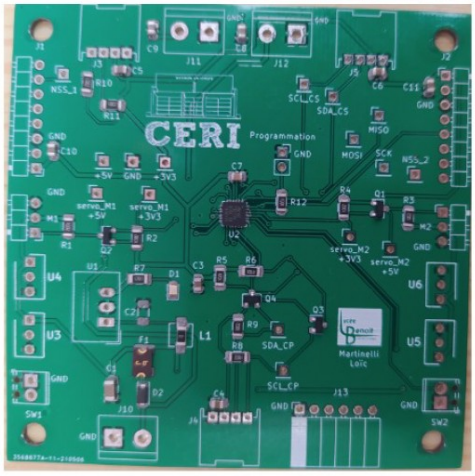
6.4-sondage du circuit

Une fois que la carte commandée est reçue j'ai imprimé ma liste de matériel pour savoir exactement les composants qu'il me fallait et scotché les CMS qui étaient les premiers composants que j'allais placer.

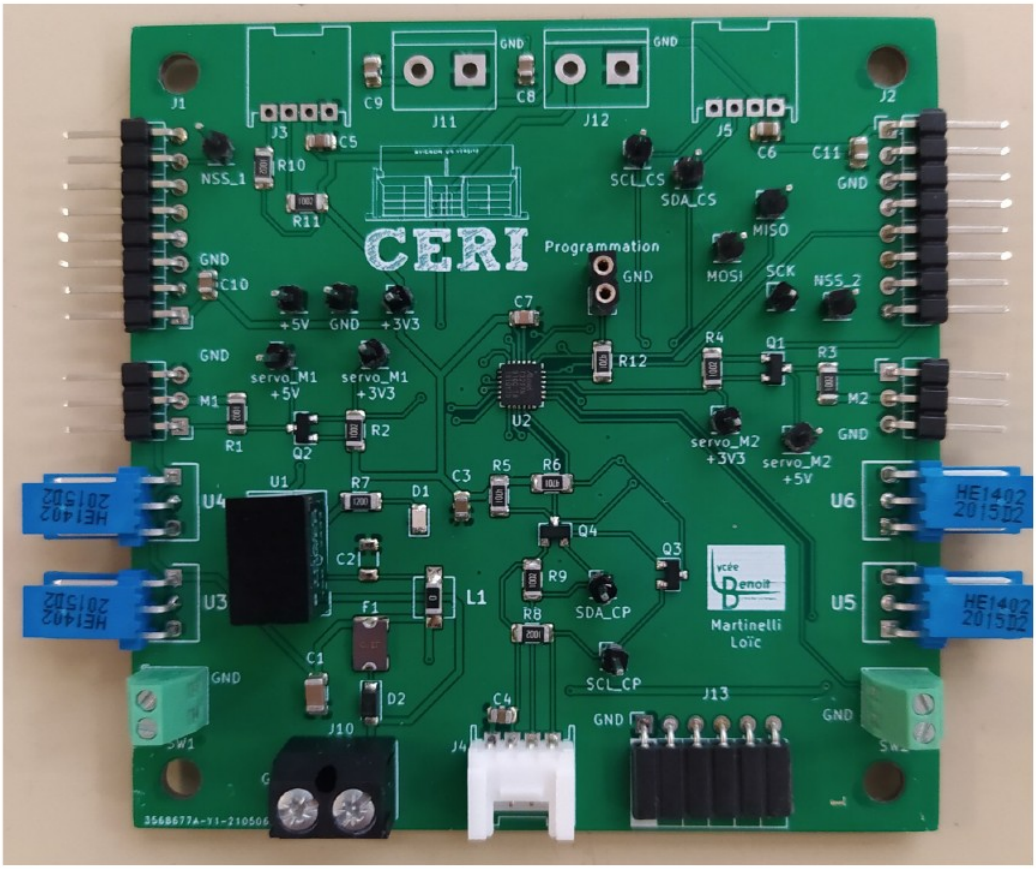
Pour ce faire j'ai placé un stencil qui est un pochoir avec les emplacements de CMS, j'ai appliqué de la pâte à braser, j'ai ensuite retiré le stencil et j'ai placé les CMS pour mettre la carte dans un four à refusion ce qui est plus facile que de soudé à la main.







Pour finir j'ai soudé les composant traversant à la main commençant du plus petit au plus grand en hauteur pour facilité la soudure.



7-Conclusion

Les tests avec le microcontrôleur ATtiny 3217 fonctionnent et la conception de la carte est finie.
Il reste encore à tester la carte et faire fonctionner tout le projet.

Partie Étudiant IR 1 – Équipe 02 : PETIT Mathéo

1. Présentation générale de mes tâches

1.1 Objectifs

Développé sur PC Windows une application cliente en C# pour communiquer avec la maquette.

1.2 Planification

	Mod Tâch	Nom de la tâche	Début	Fin	Durée	Prédécessi	Noms ressources
1		**Evaluations du projet**	**Mer 10/02/21**	**Ven 18/06/21**	**241 h?**		
2		Revue 1	Mer 10/02/21	Jeu 18/02/21	4,14 jrs		
3		Revue 2	Mer 14/04/21	Mar 11/05/21	5,29 jrs		
4		Dépôt du dossier	Ven 28/05/21	Mar 01/06/21	1 jr	15	
5		Soutenance finale E62 (Revue 3)	Lun 14/06/21	Jeu 17/06/21	2,86 jrs		
6		**Projet**	**Mar 05/01/21**	**Mar 06/04/21**	**176 h?**		**IR21**
7		spécifications générale	Mar 05/01/21	Mer 06/01/21	7 h		
8		Etude SysML	Mer 06/01/21	Ven 08/01/21	9 h	7	
9		**Contrôler en manu**	**Mar 12/01/21**	**Mer 10/02/21**	**67 h**	**8**	**IR21**
10		IHM camera	Mar 12/01/21	Mar 26/01/21	4,86 jrs	8	IR21
11		IHM reseau	Mar 26/01/21	Mer 10/02/21	4,71 jrs	10	IR21
12		**Communiquer avec la maquette**	**Mer 10/02/21**	**Mar 06/04/21**	**93 h**	**9**	**IR21**
13		modBus TCP: superviser et contrôler les	Mer 10/02/21	Mer 17/03/21	6,71 jrs	11	IR21
14		récupérer les flux des caméras + les intégrer dans l'IHM	Mer 17/03/21	Mar 06/04/21	6,57 jrs	13	IR21
15		Rédaction du dossier	Mar 05/01/21	Ven 02/04/21	24,57 jrs?		

2. Présentation de mon avancement dans le projet

J'ai commencé par développer une maquette l'application Windows Forms en C# sur Visual Studio. Visual Studio est un ensemble complet d'outils de développement permettant de générer des applications web ASP.NET, des services web XML, des applications bureautiques et mobiles. Parmi ces outils, il y a Windows Forms qui est une application avec une interface utilisateur pouvant être utilisé par un système d'exploitation Windows.

Pour la réalisation du projet, j'ai utiliser deux modules :

MySQL for Visual Studio : Permet de se connecter à une base de donnée et la modifier

Notification-Popup-WindowNotification : Permet de créer une fenêtre de notification qui apparaît dans la partie inférieure droite de l'écran pour .NET

2.1 Diagramme de classe

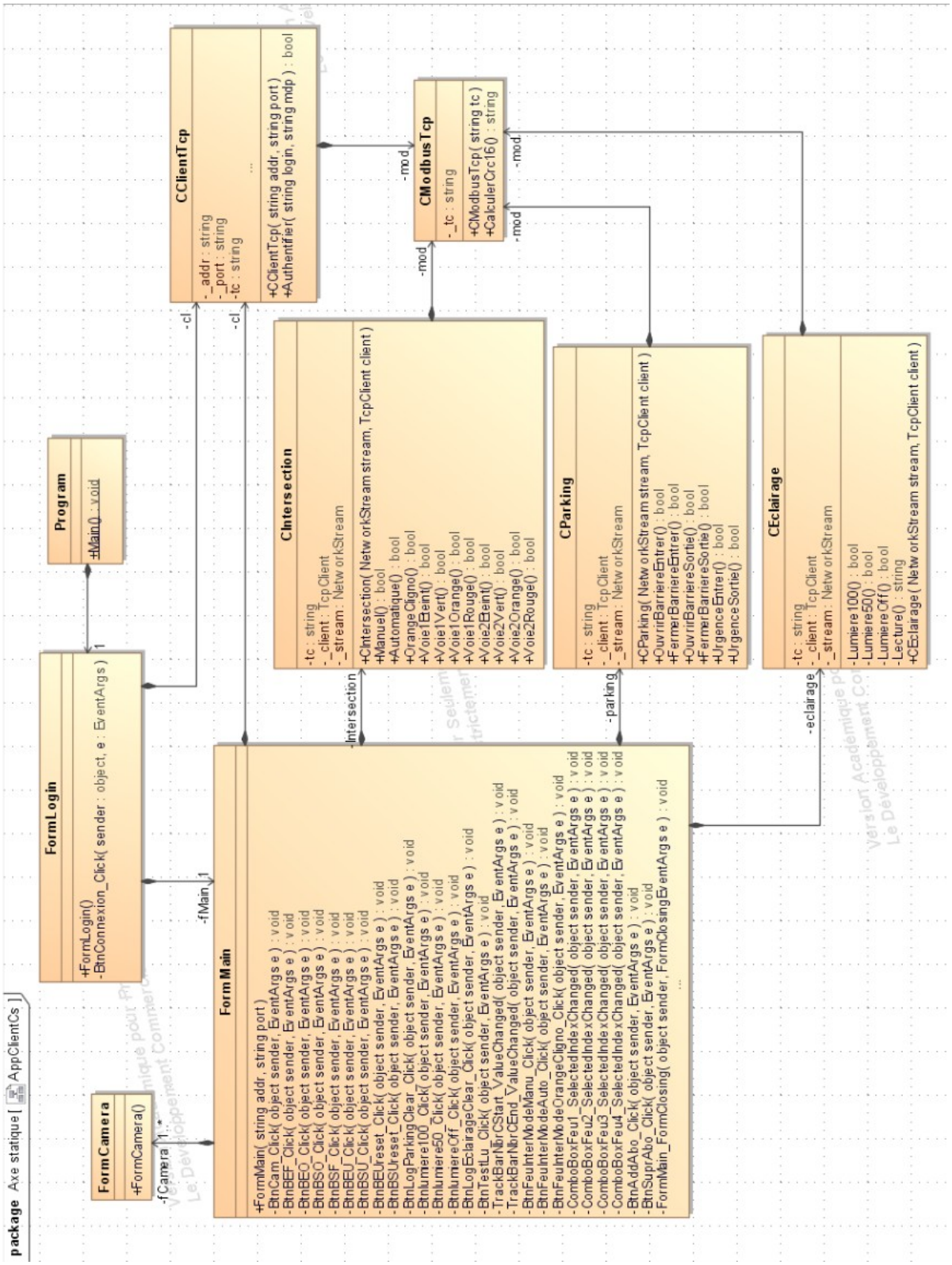


Diagramme de classe de application cliente en C#

La classe **Program** contient la méthode qui fait d'office de point d'entrée principal de l'application.

Les classes **FormLogin**, **FormMain** et **FormCamera** contiennent toutes les trois l'Ihm (Interface Homme Machine) et toutes les interactions liées à l'affichage.

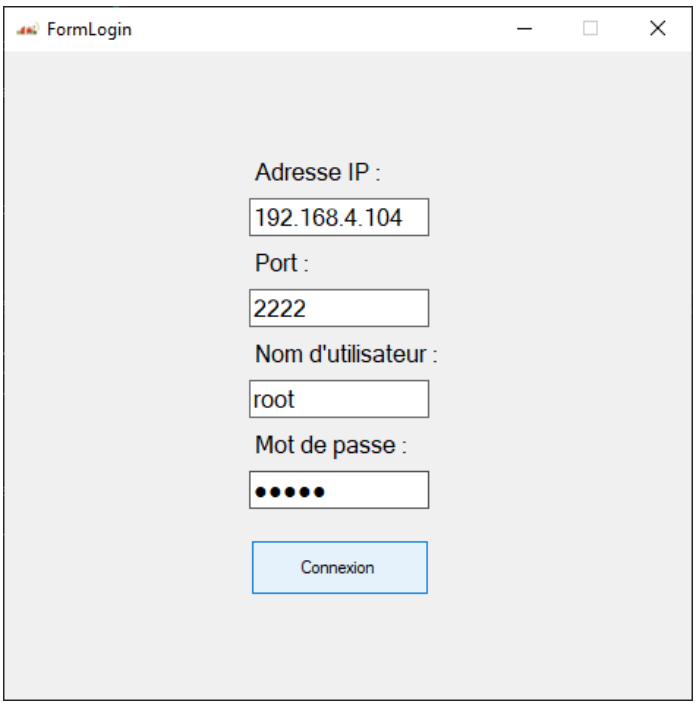
La classe **CClientTcp** va gérer la partie authentification.

Les classes **CIntersection**, **CParking** et **CEclairage** contiennent les méthodes qui vont permettre la communication avec la maquette. Chaque classe est liée à son domaine.

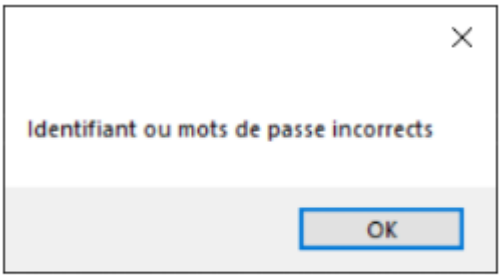
La classe **CModbusTcp** contient la méthode permettant de calculer le CRC16 pour chaque trame envoyée.

2.2 Page Authentification

Au lancement de l'application, la page d'authentification apparaît à l'écran. Pour avoir accès à la page principale, il faut mettre l'adresse IP du serveur, donner le port de communication, le nom d'utilisateur et le mot de passe. Sinon un message d'erreur apparaît signalant que l'identifiant ou le mot de passe est incorrecte.



Maquette de la page d'authentification



Message d'erreur quand on entre le mauvais identifiant et/ou du mot de passe

```
namespace AppClientCs
{
    public partial class FormLogin : Form
    {
        public FormLogin()
        {
            InitializeComponent();
        }

        private void BtnConnexion_Click(object sender, EventArgs e)
        {
            CClientTcp cl = new CClientTcp(textBoxAddr.Text, textBoxPort.Text);

            bool rep = cl.Authentifier(textBoxUtil.Text, textBoxMdp.Text);

            if (rep == true) //ouvre FormMain et cache FormConnect
            {
                this.Hide();
                FormMain fMain = new FormMain(textBoxAddr.Text, textBoxPort.Text);
                fMain.ShowDialog();
            }
            else
            {
                MessageBox.Show("Identifiant ou mots de passe incorrecte");

                return;
            }

        }
    }//class
```

Code de la page d'authentification

2.3 CClientTcp

La classe CClientTcp permet la communication avec le serveur Tcp pour envoyer et recevoir des informations et ainsi interagir avec la maquette.

Il récupère les valeurs qui ont été données par l'utilisateur dans la page d'identification pour la communiquer avec le serveur et il lui envoie un string qui contient un message plus.
Pour chaque interaction avec la maquette, une méthode sera créée. Pour le moment, il n'y a que pour s'identifier.
Pour l'identification, il envoie un string qui contient un message + l'identifiant et le mot de passe ainsi que crc16 qui sera calculer dans CModbusTcp. Si le message est envoyé et reçus avec succès, la variable res est égal à true.

```
8   class CClientTcp
9   {
10      private string _addr; // Adresse IP du serveur
11      private string _port; // Numéro de port du serveur
12      private string tc; // trame client
13
14      public CClientTcp(string addr, string port)
15      {
16          _addr = addr;
17          _port = port;
18      }
19
20      public bool Authentifier(string login, string mdp)
21      {
22          bool res = false;
23
24          try
25          {
26              Int32 port = Int32.Parse(_port);
27              TcpClient client = new TcpClient(_addr, port);
28              string message = ":00010000005FA1000C0002040";
29              // login
30              message += login;
31              int nbe = 32 - login.Length;
32              message = message.PadRight(message.Length + nbe, ' ');
33              // mdp
34              message += mdp;
35              nbe = 32 - mdp.Length;
36              message = message.PadRight(message.Length + nbe, ' ');
37              tc = message.Remove(0, 1); // Enleve le caractère ":" au débuts du message
38              CModbusTcp mod = new CModbusTcp(tc);
39              string calCrc16 = mod.CalculerCrc16();
40              message += calCrc16 + ":"; // ajoute "C115" + ":"
41
42              NetworkStream stream = client.GetStream();
43              stream.ReadTimeout = 5000; // attends la réponse jusquà 5s.
44              byte[] MessageE = Encoding.ASCII.GetBytes(message); // MessageE = Message Envoyé
45              stream.Write(MessageE, 0, message.Length);
46              Int32 nbLu = 0; // nombre de bits lu
47              byte[] MessageR = new byte[30]; // MessageR = Message Reçus
48              MessageR[0] = 0;
```

```
49
50                  if (stream.CanRead)
51                  {
52                      while (!stream.DataAvailable) ;
53                      nbLu = stream.Read(MessageR, 0, MessageR.Length); // message lu
54                      stream.Close();
55                      client.Close();
56                      MessageR[nbLu] = 0;
57                      string reponse = Encoding.ASCII.GetString(MessageR, 0, 29);
58                      Console.WriteLine("Server sent message: {0}", reponse);
59                      if (nbLu == 29)
60                      {
61                          if (reponse == ":00010000001DA1000C00020C996:") // si bons paramètres
62                              res = true;
63                      }   // if nbLu
64                  } // if canRead
65              } // try
66              catch (ArgumentNullException ane)
67              {
68                  Console.WriteLine("ArgumentNullException : {0}", ane.ToString());
69              }
70              catch (SocketException se)
71              {
72                  Console.WriteLine("SocketException : {0}", se.ToString());
73              }
74              catch (Exception e)
75              {
76                  Console.WriteLine("Unexpected exception : {0}", e.ToString());
77              }
78              return res;
79          }
```

2ème partie du code de la méthode Authentifier de la classe CClientTcp

2.4 Page Principale

2.4.1 Caméra

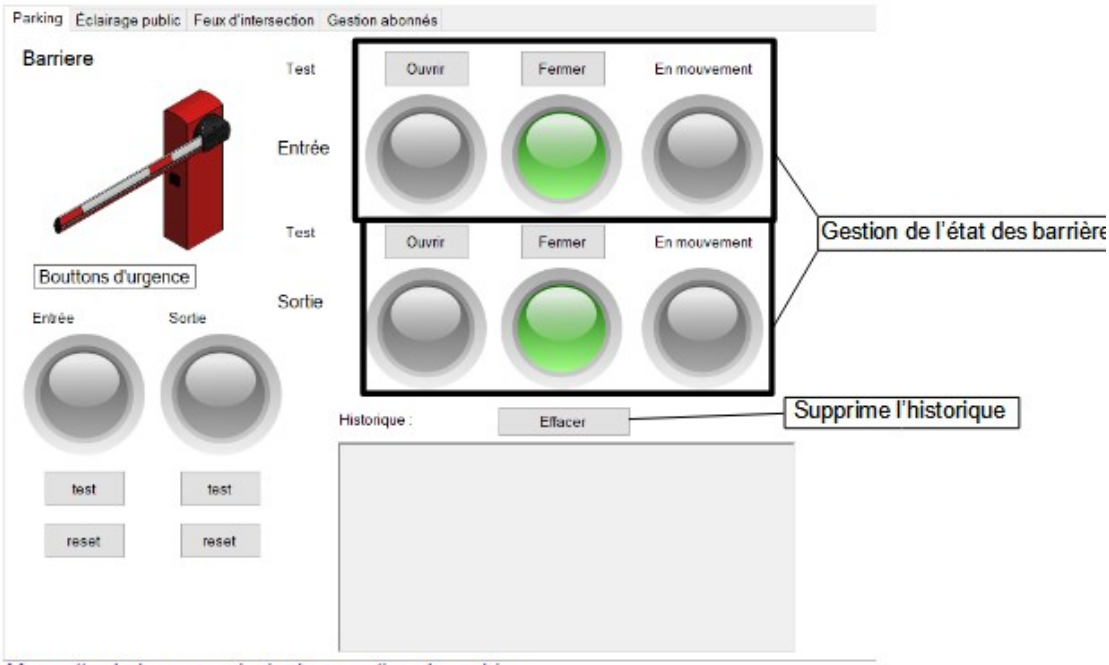
Pour la caméra, il a était envisager que le client devait avoir la possibilité d'accéder à la camera à n'importe quel moment. J'ai donc fait en sorte que dans la page principale de l'application, il y a un accès direct à la caméra qui est pour l'instant représenté par une image. Juste au-dessus de l'image, il y a un bouton appelé « Zoom » qui permet la caméra en fenêtré.

Maquette de la page principale – gestion du parking

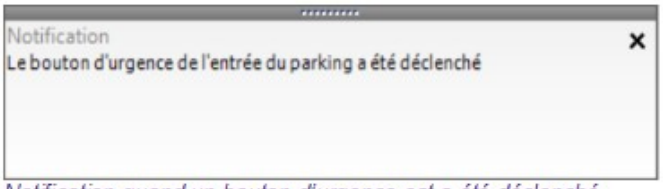
Maquette de la page caméra

2.4.2 Parking

Dans la gestion du parking, il y a trois butons pour chaque barrière, une pour ouvrir, une autre pour fermer et la dernière sert juste à allumer le voyant qui annonce quand la barrière est en mouvement. Grace au voyant, il permet de connaître l'état des barrières en temps réels. Le parking possède deux boutons d'urgence une à l'entrer et une autre en sortie. Quand celle-ci est appuyée, un pop-up apparaît dans le coin de l'écran. Il y a aussi un historique de la communication avec la maquette.



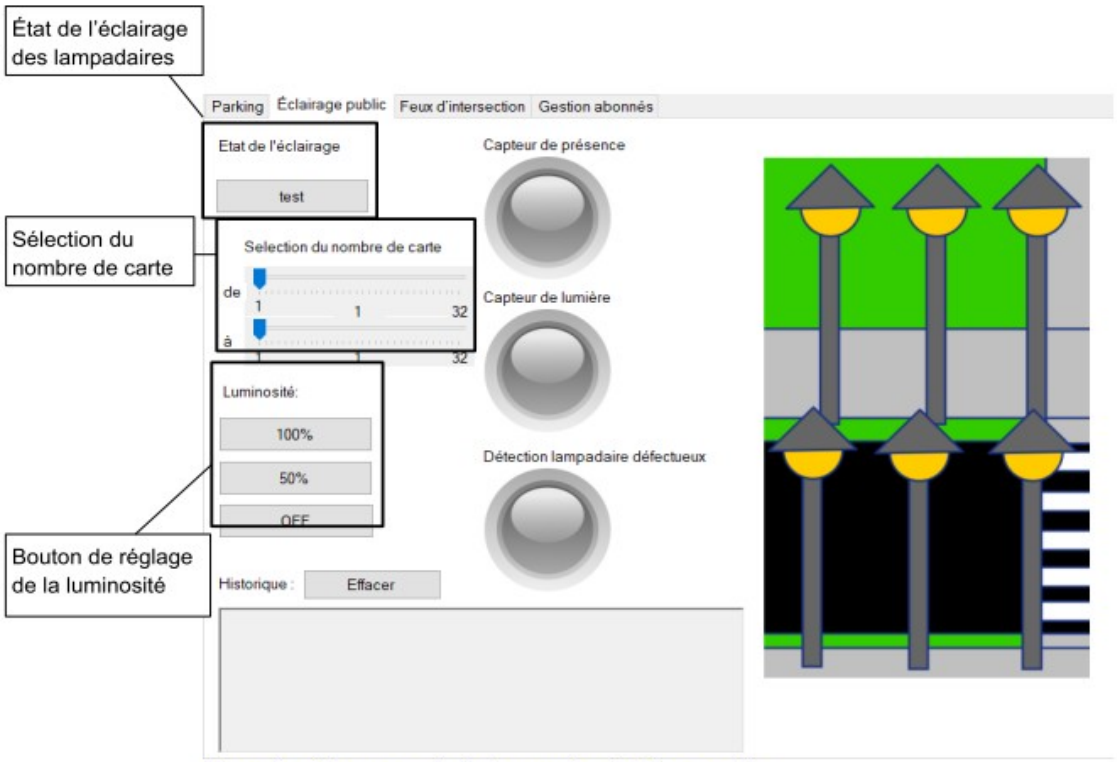
Maquette de la page principale – gestion du parking



Notification quand un bouton d'urgence est a été déclenché

2.3.3 Éclairage public

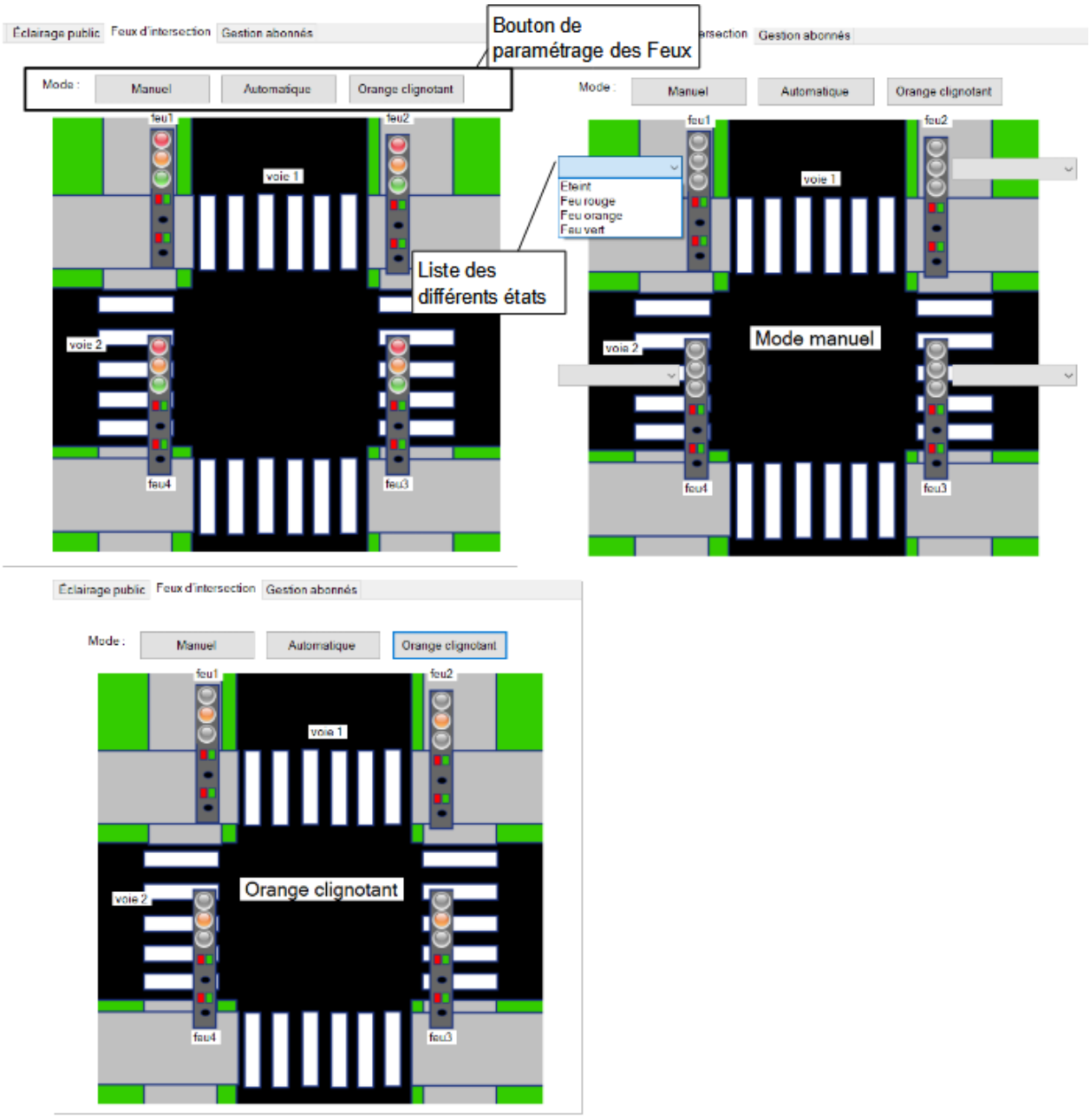
La gestion de l'éclairage public possède plusieurs boutons, un bouton pour simuler la présence d'un piéton, un bouton pour connaître l'état de l'éclairage et enfin trois autres boutons qui envoient l'ordre d'allumer l'éclairage avec de différents niveaux de luminosité. Il y a aussi un historique de la communication avec la maquette.



Maquette de la page principale – gestion éclairage public

2.3.4 Feux d'intersection

La gestion des feux d'intersection deux modes de configurations : « Manuel » qui éteint les feux et permet au client de choisir l'état des feux et « Automatique » qui active le cycle automatique des feux.



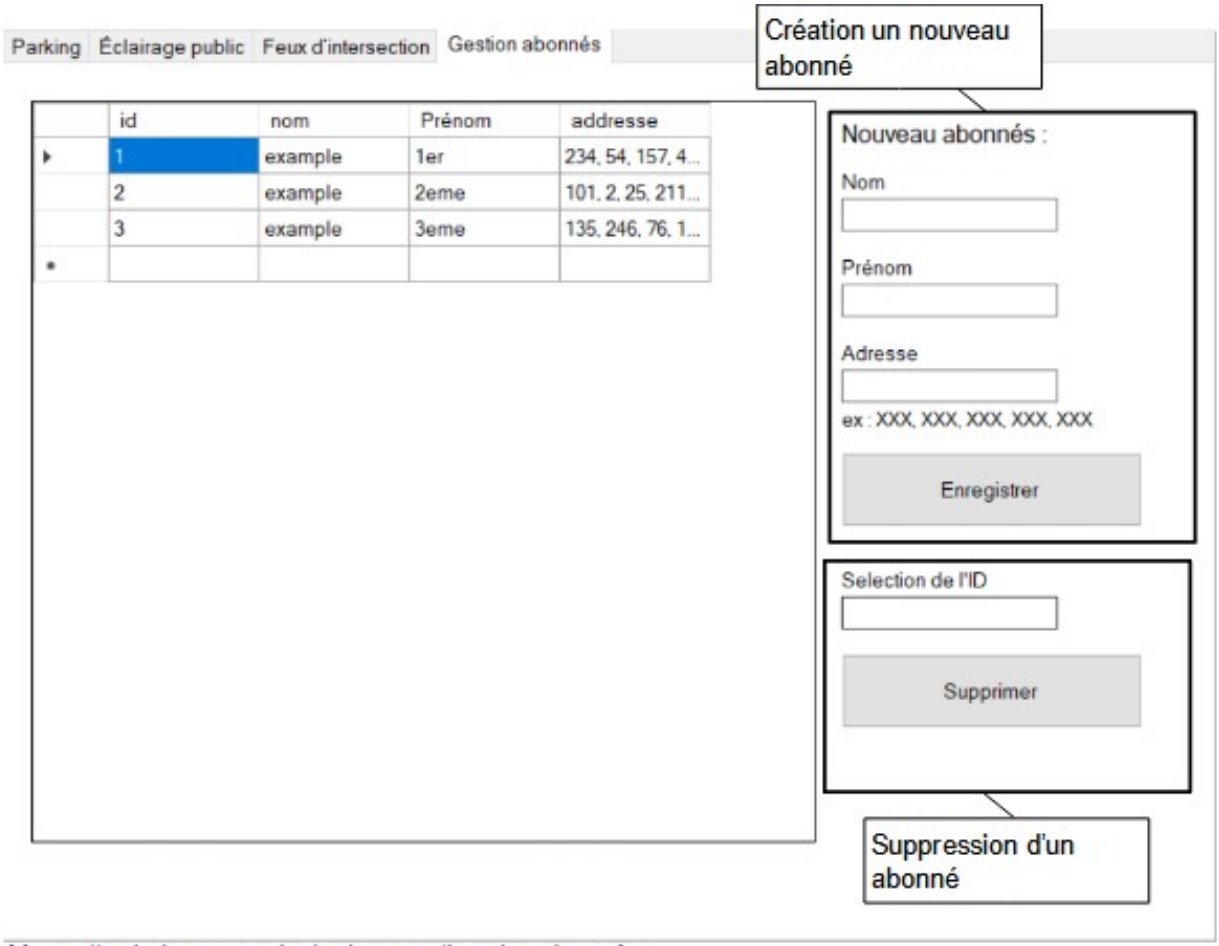
Maquette de la page principale – gestion feux intersection en mode « Automatique », « Manuel » et « Orange clignotant »

2.4.5 Gestion abonnés

Dans la gestion des abonnés, il existe une liste d'abonnés qui est une base de données SQLite située sur l'ordinateur où vous pouvez ajouter de nouveaux abonnés et également les supprimer.

Le choix d'avoir utilisé une base de données SQLite est qu'il s'agit d'un système de gestion de base de données embarqué. Autrement dit, il est stocké dans un fichier.

Lors de la création ou de la suppression d'un abonné, une connexion est établie à la base de données pour l'envoyer sous forme de commande en SQL pour la modifier et la mettre à jour.



Maquette de la page principale – gestion des abonnés

```
CREATE TABLE abonnement (
    id       INTEGER PRIMARY KEY AUTOINCREMENT,
    nom      TEXT,
    Prénom   TEXT,
    addresse TEXT
);
```

Création de la table « abonnement »

2.5 Diagramme des cas d'utilisation

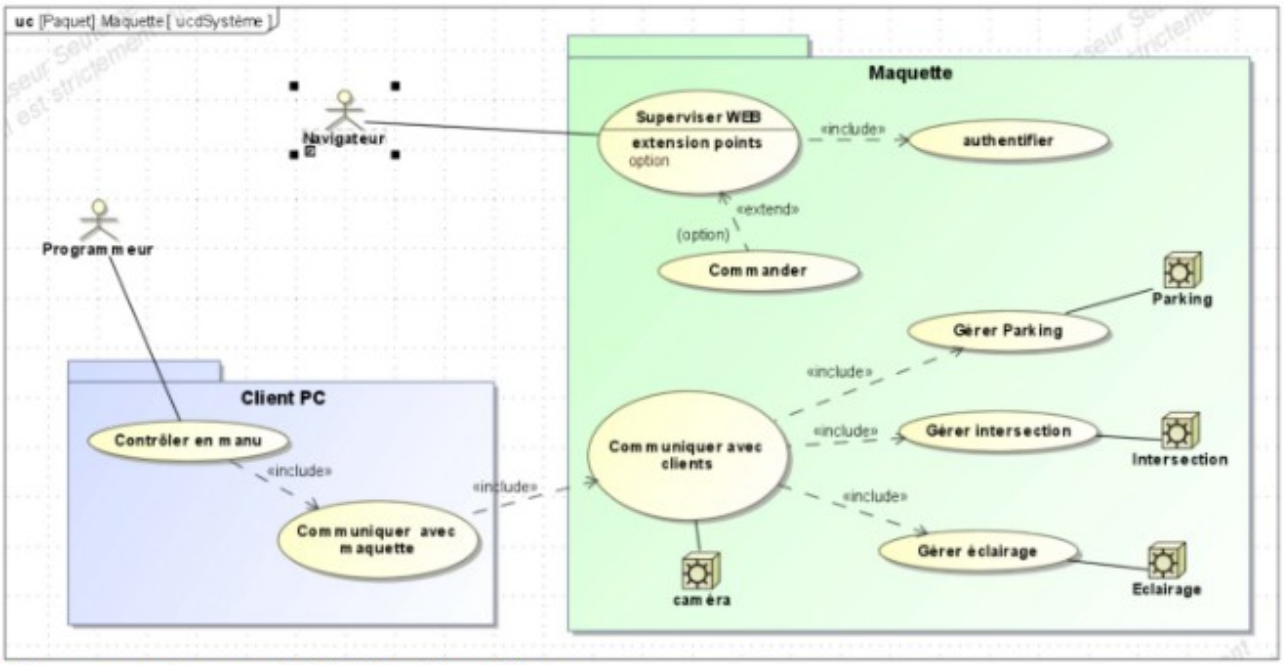


Diagramme des cas d'utilisation du système

2.6 Diagramme de séquence

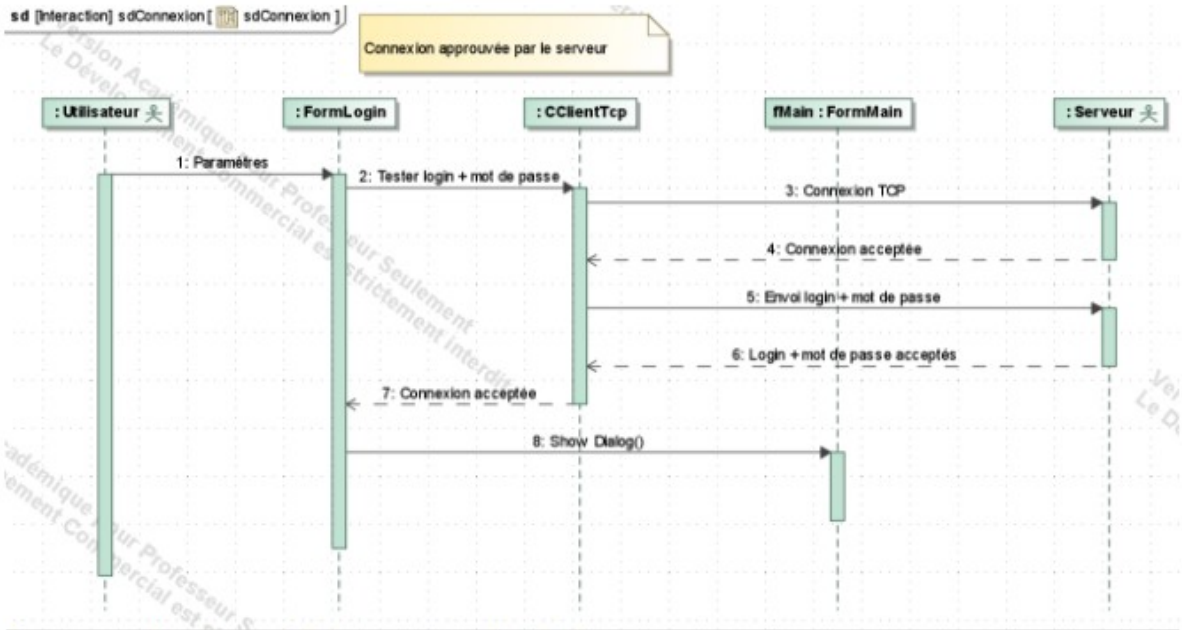
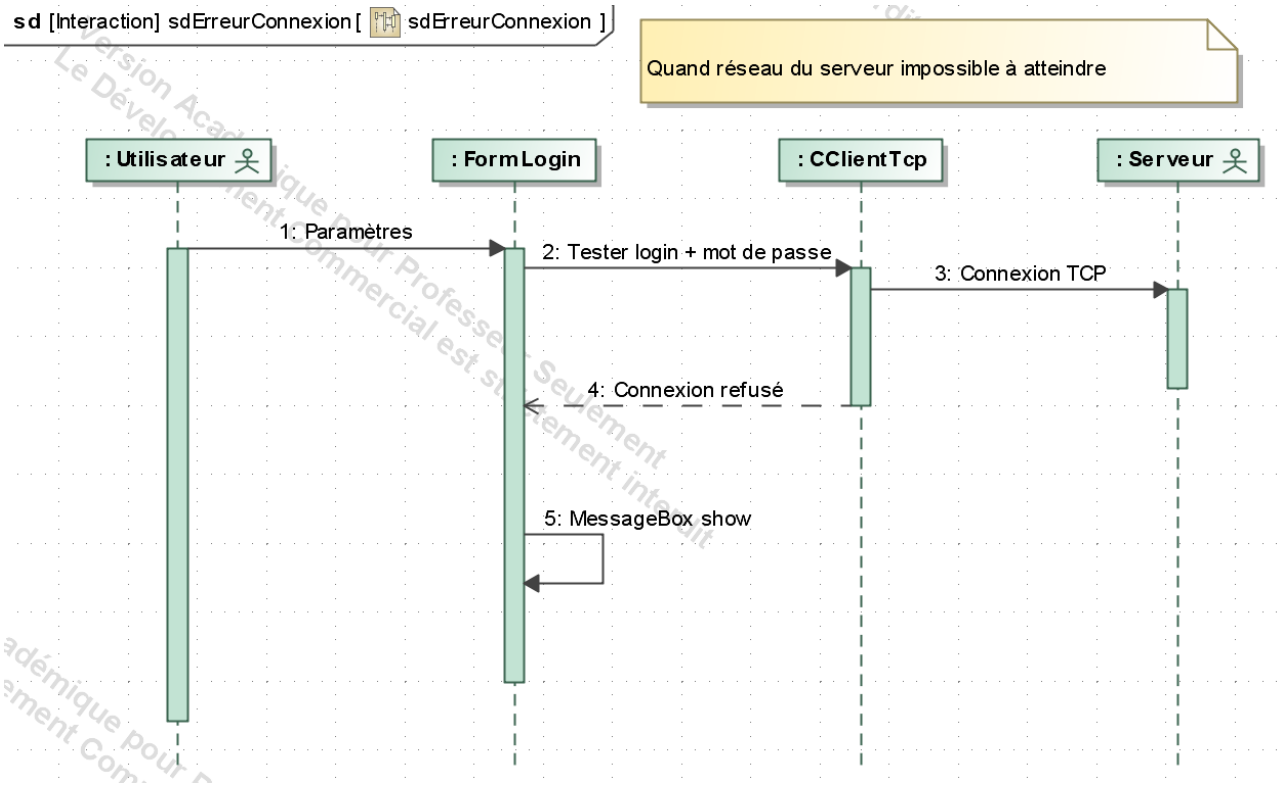


Diagramme de séquence de la connexion du client


sd [Interaction] sdErreurConnexion [sdErreurConnexion]
Quand réseau du serveur impossible à atteindre
: Utilisateur
: FormLogin
: CClientTcp
: Serveur
1: Paramètres
2: Tester login + mot de passe
3: Connexion TCP
4: Connexion refusé
5: MessageBox show

2.7 Fiches recettes

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :		Code de la campagne de test :	GR2 - IR1 - 01
Technicien 1 :	PETIT Mathéo	Date :	25/05/2021

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Authentification
Objectif du scénario :	Réussir à se connecter au serveur et à s'authentifier pour interagir avec la maquette.

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

- On doit être connecter dans le même réseau que le serveur et connaître son adresse IP et son numéro de port.
- Avoir mis le bon identifiant « root » et mot de passe « admin ».

EXECUTION DU TEST

Description courte :	Ajout de l'identifiant root et le mot de passe admin et appuis
Résultats attendus :	Cache FormLogin et ouvre FormMain

BILAN

Cache FormLogin et ouvre FormMain

REMARQUES

CONCLUSION

VALIDE	X	NON VALIDE	

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTEME

Nom du système :	smartcity	Code de la campagne de test :	GR2 - IR1 - 02
Technicien 1 :	PETIT Mathéo	Date :	25/05/2021

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Ajout d'un nouveaux client
Objectif du scénario :	Ajouter un nouveaux client

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

- Réussir à s'authentifier sur FormLogin pour être sur FormMain.

EXECUTION DU TEST

Description courte :	Création d'un nouveau client Nom : JEAN Prénom : Pascal Adresse : 145, 80, 137, 212, 12
Résultats attendus :	Voir sur l'Ihm l'ajout du nouveaux client dans la base de données.

BILAN

La base de données a été mise à jour avec l'ajout d'un nouveau client avec les informations correctes et dans la colonne id, une valeur a été ajoutée.

REMARQUES

CONCLUSION

VALIDE	X	NON VALIDE	

(Mettre une croix dans la case correspondante)

3. Partie Physique

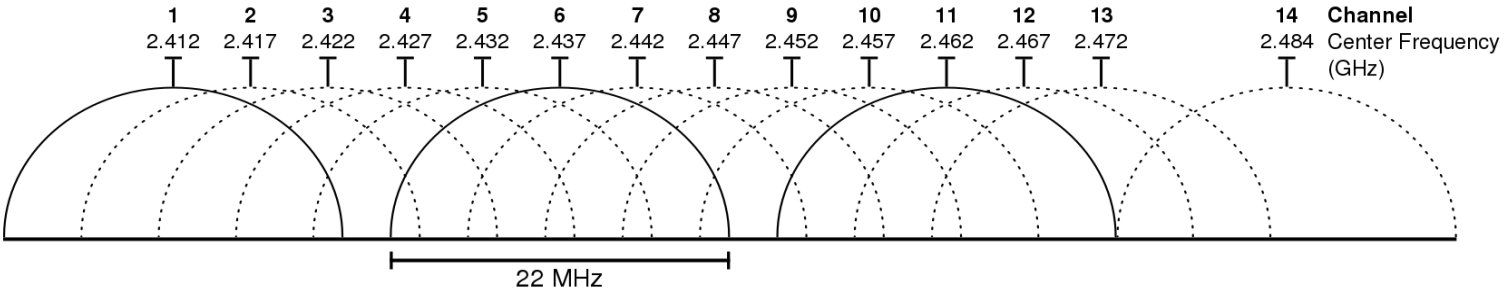
Dans le projet, il y a un Raspberry PI 3B+ qui possède une carte réseau compatible avec le protocole **WiFi** (Wireless Fidelity). Je vais expliquer comment cela fonctionne.

Le WiFi est un réseau informatique numérique qui connecte différents postes ou systèmes entre eux. L'information est transmise par signal électrique qui est ensuite convertie en onde radio(grâce à une antenne) qui va voyager dans les airs et sera ensuite récupérée par un récepteur qui grâce à une antenne de réception la reconvertie en signal électrique. Il existe deux bandes de fréquence dédiées au WiFi (2,4GHz et 5GHz). Le protocole WiFi est régie par **IEEE 802.11** qui est un ensemble de norme.

Standard	Frequency	Maximum Speed	Backwards compatibility
802.11	2.4 GHz	2 Mbps	-
802.11a	5 GHz	54 Mbps	-
802.11b	2.4 GHz	11 Mbps	-
802.11g	2.4 GHz	54 Mbps	802.11b
802.11n	2.4 and 5 GHz	600 Mbps	802.11a/b/g
802.11ac	5 GHz	1300 Mbps	802.11a/n
802.11ad	2.4 GHz, 5 GHz and 60 GHz	7 Gbps	802.11a/b/g/n/ac

Amendements principaux à la norme Wi-Fi IEEE 802.11

Dans la bande 2,4GHz le spectre radio attribué au WiFi débute à 2400 MHz, il se termine à 2483,5 MHz. Dans le spectre il y a 13 canaux (le 14ème n'est utilisé qu'au Japon) dont chacun à une plage de fréquence de +/-11 MHz autour de sa fréquence centrale. Les canaux sont utilisés pour que deux appareils arrivent à communiquer entre eux. Il est recommandé d'utiliser des canaux libres et laisser au moins 3 canaux vides entre les canaux voisins et celui que l'on veut utiliser pour éviter de causer et de subir des interférences radio.



En France, la **PIRE** (puissance isotrope rayonnée équivalente) est de 100 mW dans la bande 2,4GHz.

4. Objectifs à réaliser pour la suite du projet

- Récupérer les flux des caméras
- Finir les méthodes CEclairage

Partie Étudiant IR 2 – Équipe 02 : CHEVALIER Joséphine

1 – Présentation générale de mes tâches

1.1 – Objectifs

- Contrôler, superviser la maquette d'une ville à partir d'un logiciel muni d'une IHM.
- Communiquer avec la maquette en utilisant le protocole modBus TCP et http/https/ftp

1.2 – Planification des tâches

	Mode	Nom de la tâche	Durée	Début	Fin	Prédécesseurs
1		**<Création Client PC>**	**288 h**	**Mer 06/01/2**	**Ven 18/06/2**	
2		Spécifications Générales	7 h	Mer 06/01/2	Jeu 07/01/21	
3		Étude SysML	9 h	Ven 08/01/2	Mer 13/01/2	2
4		**<Contrôler en manuel>**	**71 h**	**Mar 19/01/2**	**Ven 19/02/2**	**3**
5		Création du Prototype	30 h	Mar 19/01/2	Ven 29/01/2	
6		Codage de l'IHM	30 h	Mar 02/02/2	Ven 12/02/2	5
7		Revue 1	4 h	Ven 19/02/2	Ven 19/02/2	6
8		**<Communication avec Maquette>**	**104 h**	**Mar 09/03/2**	**Mar 11/05/2**	**4**
9		ModBus TCP	30 h	Mar 09/03/2	Ven 19/03/2	
10		Requêtes Récupération Flux	40 h	Mar 23/03/2	Jeu 08/04/21	9
11		Intégration dans IHM (flux)	30 h	Jeu 08/04/21	Ven 23/04/2	10
12		Revue 2	4 h	Ven 23/04/2	Mar 11/05/2	11
13		Réunion des Projets	20 h	Mar 11/05/21	Mer 19/05/21	8
14		Tests Projet Finaux	20 h	Jeu 20/05/21	Ven 28/05/21	13
15		Revue Finale - E62	5 jrs	Jeu 03/06/21	Ven 18/06/2	14

2 – Présentation de l'avancement du projet

2.1 – Maquette de l'interface graphique

2.1.1 – La page d'authentification

Maquette de la page d'authentification

La page d'authentification est faite à partir d'un Qdialog. Quand on lance l'application, seul la page permettant la connexion apparaît à l'écran.

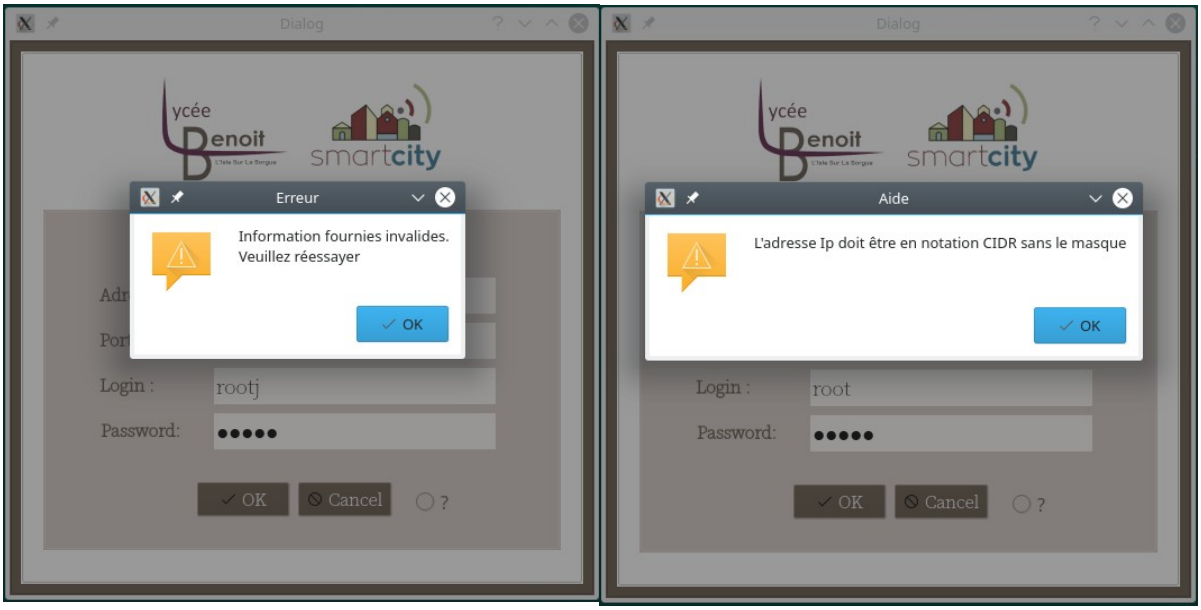
Si on appuie sur le bouton Cancel , l'application se ferme et on n'a pas accès à la page d'accueil.

Si les informations saisies sont correctes et que l'on appuie sur Ok alors la page se ferme et laisse place à la page principale.

Code montrant la connexion entre les boutons et les signaux :

```
5   CIhmLogin::CIhmLogin(QWidget *parent) :
6       QDialog(parent),
7       ui(new Ui::CIhmLogin)
8   {
9       ui->setupUi(this);
10      connect(ui->buttonBox, &QDialogButtonBox::accepted, this, &CIhmLogin::on_ok);
11      connect(ui->buttonBox, &QDialogButtonBox::rejected, this, &CIhmLogin::reject);
12  }
```

Sinon, il y a l'apparition d'un pop-up notifiant que les informations fournies sont invalides.
De même si on clique sur ? une aide s'affiche.



Pour réaliser cela, j'ai utilisé des QMessageBox. Voici le code utilisé pour l'image de gauche.

```
49  QMessageBox::warning(this, tr("Erreur"), tr("Information fournies invalides.\nVeuillez réessayer"));
```

Finalement, j'ai rencontré un problème : quand on cliquait sur la croix, la page d'authentification se fermait mais la page d'accueil s'affichait.
J'ai réussi à faire en sorte que l'application se ferme grâce au code suivant :

```
6  int main(int argc, char *argv[])
7  {
8      QApplication a(argc, argv);
9      CIhmLogin l;
10     CIhm w;
11     int res = l.exec(); // exec() renvoie un résultat DialogCode
12     if (res == 0)
13         return -1;
14     w.show();
15     return a.exec();
16 }
```

La fonction exec() affiche le Qdialog en modal et renvoie un résultat DialogCode.

Si exec() renvoie 0 alors une erreur est retournée.

Code slot **on_ok()** :

```
27  void CIhmLogin::on_ok()
28  {
29
30      // Numéro de Port
31      _port = ui->lePort->text();
32
33      // Adresse IP
34      _ip = ui->leIp->text();
35
36      // Mot de passe
37      _pswd = ui->lePassword->text();
38
39      // Indentifiant
40      _login = ui->leLogin->text();
41
42      connect(&_sock, SIGNAL(readyRead()), this, SLOT(onReadyRead()));
43      _sock.connectToHost(_ip, _port.toInt());                    ⚠ implicit conve
44      if (!_sock.isOpen()) {
45          qDebug() << "Connexion au serveur HS !";
46          QMessageBox::warning(this, tr("Erreur"), tr("Information fournies invalides.\
47      } // if erreur
48
49      QByteArray trame =  contructTrame();
50      int nb = _sock.write(trame.toStdString().c_str());          ⚠ implicit
51      if (nb == -1)
52          qDebug() << "CClientTcp::emettre Erreur écriture.";
53
54  }
```

On récupère les informations saisies sur l'IHM.
On connecte le **readyRead()** (signal envoyé par la sock quand elle reçoit un message) avec le slot **onReadyRead()** (récupère le message et le traite).

On place la sockect (**_sock.connectToHost()**) qui permet de se connecter, d'envoyer et recevoir des message.

Le **if** sert lorsque la socket n'est pas ouverte d'envoyer un message d'erreur.

QbyteArray trame = constructTrame() sert à construire la trame.
_sock.write() permet au message de s'envoyer.

2.1.2 – La page principale

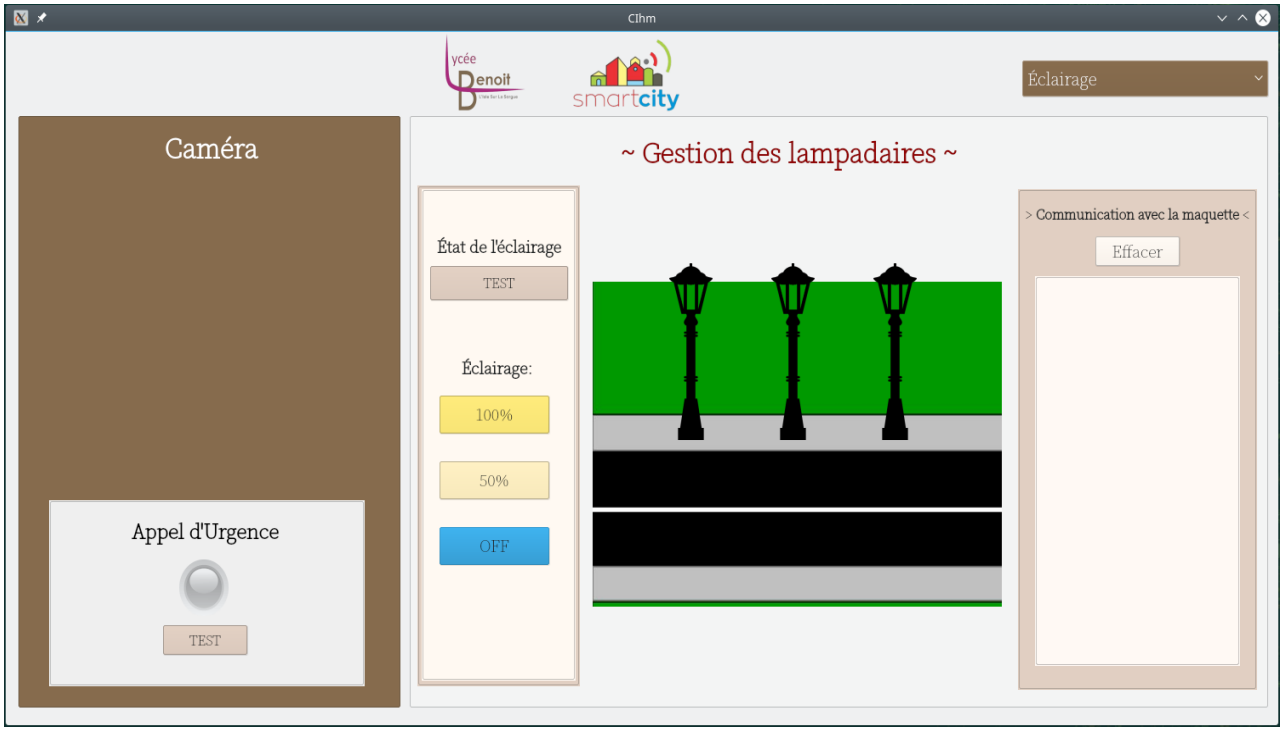
Voici si-dessous la page principale. Sur la partie de gauche, nous avons un service d'appel d'urgence. Quand sur le parking, on appuie sur le bouton d'appel, le voyant ici gris va se mettre à clignoter en rouge jusqu'à ce que l'utilisateur du client réponde. Nous pouvons voir aussi qu'il y a un emplacement qui servira à afficher les images envoyées par la caméra.

La partie de droite est composée de 4 onglets. En haut à droite, nous avons à disposition d'un menu déroulant qui va nous permettre de naviguer entre les différents onglets.

Ce systèmes divisé en deux partie permet que lorsqu'on change d'onglet dans la partie de droite, on puisse voir la caméra et l'appel d'urgence.

Dans les trois premiers onglets, nous allons retrouver une zone « Communication avec la maquette ». Celle-ci sert à afficher tous les ordres envoyés et mais aussi les informations reçu par le client.

L'onglet sélectionné par défaut est celui de la gestion de l'éclairage.



Maquette de la page principale – Onglet gestion de l'éclairage

Boutons :

- **TEST** : demande l'état des lampadaires et l'affiche dans le client
- **100 %** : envoie l'ordre de mettre en marche l'éclairage à 100 % de sa luminosité
- **50 %**: envoie l'ordre de mettre en marche l'éclairage à 50 % de sa luminosité
- **Off** : envoie l'ordre d'éteindre
- **Effacer** : supprime l'historique de la communication

Il y a deux jauges qui permettent de sélectionner le nombre de cartes de lampadaires.
La première pour le numéro de la première à sélectionner et l'autre pour la dernière.
Il y a au maximum 32 cartes de 6 lampadaires.

Dans l'onglet gestion du parking, il y a deux boutons pour ouvrir ou fermer la barrière entrée et de même pour la sortie, un affichage des places libres, des voyant pour savoir si les barrières sont ouvertes ou fermée et pour finir un bouton qui permet d'afficher l'état des barrières.

Maquette de la page principale - Onglet gestion du parking

Boutons :

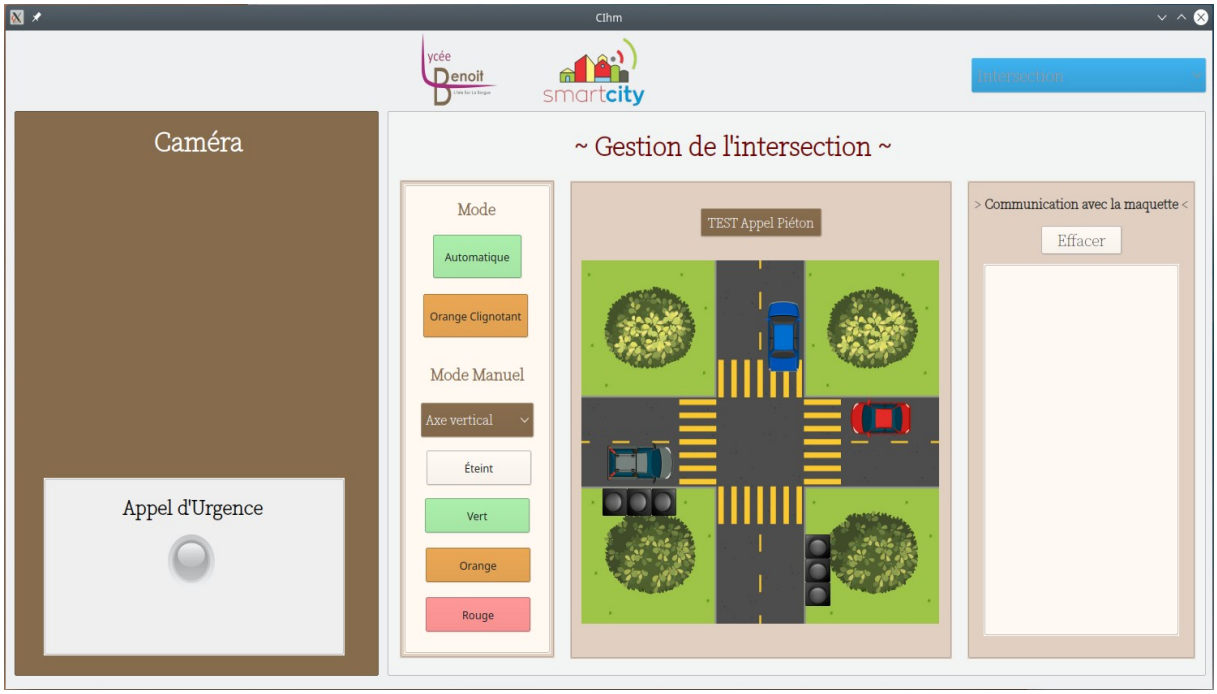
- **TEST** : demande l'état des barrières et l'affiche dans le client
- **TEST RFID** : récupérer le numéro RFID
- **Effacer** : supprime l'historique de la communication

Pour l'entrée et la sortie :

- **Ouvrir** : envoie l'ordre d'ouvrir la barrière
- **Fermer**: envoie l'ordre de fermer la barrière

Il y a deux entrées de texte qui permettront d'afficher un message sur le panneau d'affichage. Un entrée pour la première ligne du tableau d'affichage et l'autre pour la deuxième.

L'onglet de gestion de l'intersection présente deux modes : automatique et manuel.



Maquette de la page principale – Onglet gestion de l'intersection

Boutons :

- **Test** : demande le nombre de boutons piétons appuyés par axe
- **Effacer** : supprime l'historique de la communication

~Mode automatique

~Mode Orange clignotant

~Mode éteint

~Mode manuel : les boutons servent à afficher sur le feux la couleur voulue

- **Rouge**
- **Orange**
- **Vert**
- Au dessus de l'image on retrouve un bouton qui permet de changer l'axe de l'intersection. Il est sur « axe vertical » par défaut

Les voies fonctionnent par deux c'est pour cela que j'ai appelé cela des axes.

Maquette de la page principale – Onglet gestion des abonnés

Boutons :

- **Ajouter** : ajoute une ligne pour un abonné
- **Supprimer**: supprime un abonné sélectionné

2.2 Diagramme de cas d'utilisation

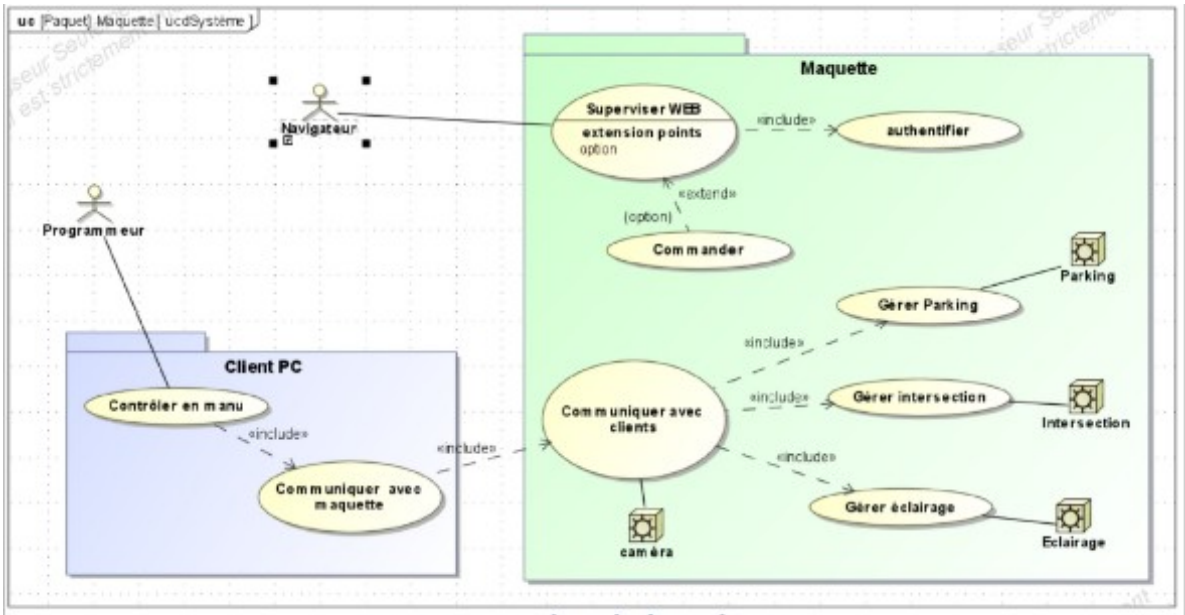


Figure 2 : Diagramme des cas d'utilisation du système

Le diagramme des cas d'utilisation permet d'avoir une vue d'ensemble des fonctionnalités du point de vue de l'utilisateur autrement dit le programmeur.
L'utilisateur doit pouvoir contrôler, superviser les éléments de la maquette, mais aussi avoir une vision globale l'état de chaque système. Les éléments contrôlables via le client sont les deux barrières du parking, la série des six lampadaires et les feux de l'intersection. L'utilisateur peut aussi ajouter un ou plusieurs abonnements (Numéros RFID).

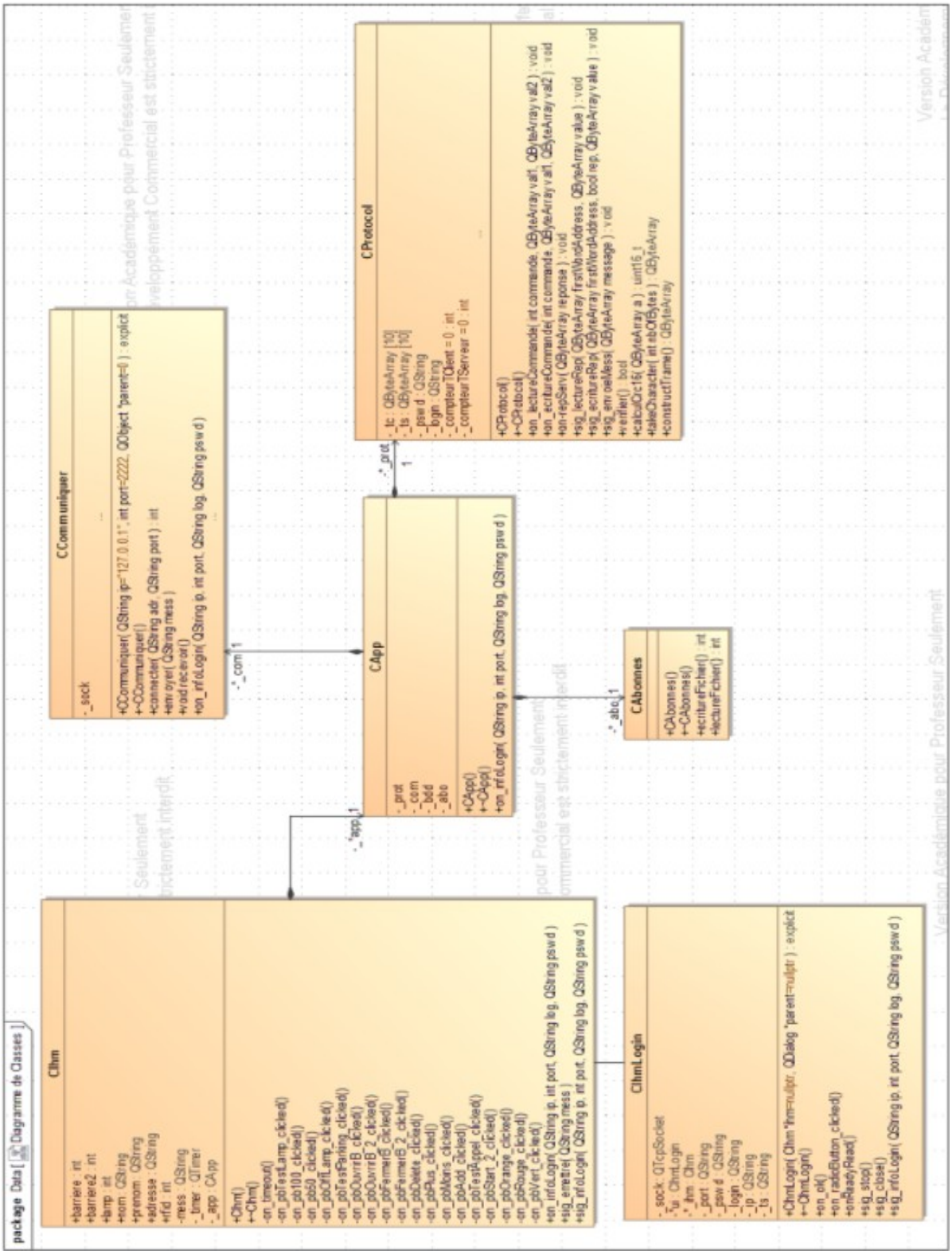
2.3 Les classes

2.3.1 Présentation des classes

Les classes commençant par le mot-clé CIhm correspondent aux différentes fenêtres de l'application :
- CIhmLogin correspond à la page d'authentification
- Cihm correspond à la page principale

La classe Capp est la classe qui lie les autres classes à l'Ihm, c'est à dire aux classes CihmLogin et Cihm.
Capp hérite de plusieurs classes :
- Cabonnes permet la gestion des abonnés (recherche, ajout, et suppression)
- CCommuniquer est la classe qui va permettre la communication avec la maquette
- Cprotocol va gérer les trames qui vont être envoyé à la maquette en modBus TCP
- Cprotocol va gérer les trames qui vont être envoyé à la maquette en modBus TCP

2.3.2 Diagramme de classes



2.3 – Les trames d'écriture

J'ai actuellement préparé toutes les trames d'écriture en suivant le formulaire créée par l'IR 01 du groupe 1 (Ré Sébastien), qui permettront d'envoyer les ordres à la maquette.
Chaque trame commence et finie par « : ». Elle se divise en 10 parties.
« **Transaction identifier** » correspond à un nom/identifiant, pour nous ce sera toujours 0x0001. (2 octets)
« **Protocol identifier** » est toujours de 0x0000 car on est en modBus. (2 octets)
« **Length** » est la taille totale de la trame. (2 octets)

« **Unit identifier** » correspond à : P = parking, I = intersection, E = Eclairage, A = authentification. (1 octet)
« **Function code** » dépend de la nature de la demande. Dans notre système, nous aurons besoin du code fonction 03 ou 04 en ce qu'il concerne la lecture. Et du code fonction 16 en ce qu'il concerne l'écriture.
« **Data** » est une partie qui comprends l'adresse du premier mot à forcer, le nombre de mots à forcer, le nombre d'octet forcer et la valeur des mots à forcer. (x octets)
« **CRC16** » correspond a un calcul de l'ADU (Application Data Unit). Ce CRC calculé sur 16 bits est partie intégrante du message et il est vérifié par le destinataire. Il est calculé sur tous les octets de la trame à part lui-même bien-entendu.

2.3.1 - Le parking

	Transaction identifier	Protocol identifier	Length	Unit identifier	Function code	Adresse du 1er mot à forcer	Nombre de mots à forcer	Nombre d'octets à forcer	Valeur des mots à forcer	CRC16
Ouverture barrière entrée	0001	0000	0021	P	10	00A0	0001	02	0001	
Fermeture barrière entrée	0001	0000	0021	P	10	00A0	0001	02	0002	
Ouverture barrière sortie	0001	0000	0021	P	10	00A0	0001	02	0004	
Fermeture barrière sortie	0001	0000	0021	P	10	00A0	0001	02	0008	

2.3.2 – Eclairage

	Transaction identifier	Protocol identifier	Length	Unit identifier	Function code	Adresse du 1er mot à forcer	Nombre de mots à forcer	Nombre d'octets à forcer	Valeur des mots à forcer	CRC16
100 %	0001	0000	005F	E	10	0000	0020	40	0003 (x32)*	
50 %	0001	0000	005F	E	10	0000	0020	40	0001 (x32)*	
OFF	0001	0000	005F	E	10	0000	0020	40	0000 (x32)*	

* (x32) car il y a 32 cartes ce qui donnerait comme valeur de mot à forcer :
ex : éclairage 100 % →
0003000300030003000300030003000300030003000300030003000300030003000300
0300030003000300030003000300030003000300030003000300030003

2.3.3 – Intersection

- Mode Automatique :

Transaction identifier	Protocol identifier	Length	Unit identifier	Function code	Adresse du 1er mot à forcer	Nombre de mots à forcer	Nombre d'octets à forcer	Valeur des mots à forcer	CRC16
0001	0000	0021	I	10	00AA	0001	02	0001	

- Mode Manuel :

Voie 1

	Transaction identifier	Protocol identifier	Length	Unit identifier	Function code	Adresse du 1er mot à forcer	Nombre de mots à forcer	Nombre d'octets à forcer	Valeur des mots à forcer	CRC16
Éteint	0001	0000	0021	P	10	00AA	0001	02	0042	
Vert	0001	0000	0021	P	10	00AA	0001	02	0046	
Orange	0001	0000	0021	P	10	00AA	0001	02	004A	
Rouge	0001	0000	0021	P	10	00AA	0001	02	004E	

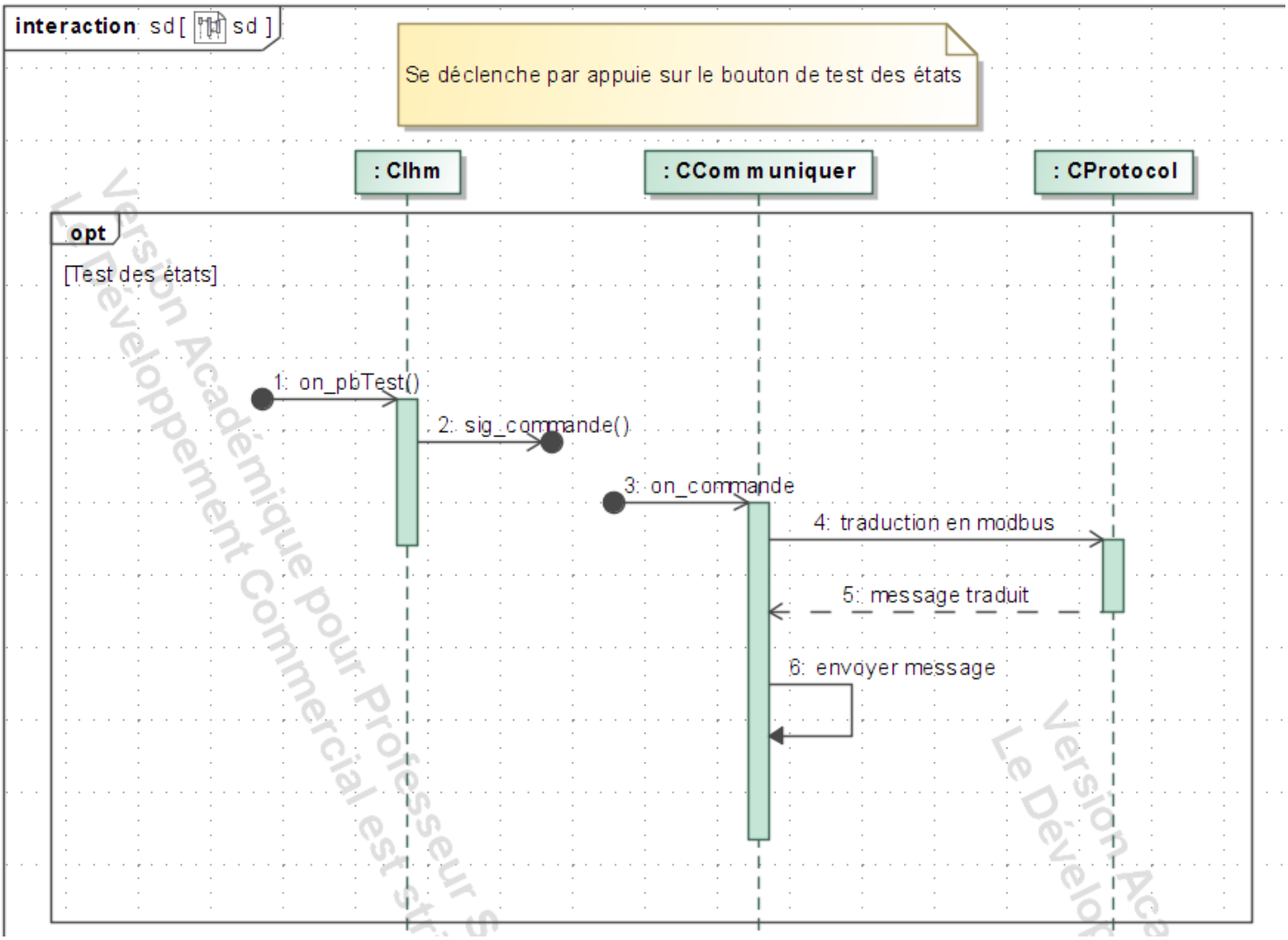
Voie 2

	Transaction identifier	Protocol identifier	Length	Unit identifier	Function code	Adresse du 1er mot à forcer	Nombre de mots à forcer	Nombre d'octets à forcer	Valeur des mots à forcer	CRC16
Éteint	0001	0000	0021	P	10	00AA	0001	02	0082	
Vert	0001	0000	0021	P	10	00AA	0001	02	0092	
Orange	0001	0000	0021	P	10	00AA	0001	02	00A2	
Rouge	0001	0000	0021	P	10	00AA	0001	02	00B2	

Le CRC16 est calculé grâce à cette portion de code:

```
103 uint16_t CIhmLogin::calculCrc16(QByteArray a)
104 {
105     uint8_t nbDec;
106     uint8_t yaun;
107     uint8_t indice;
108     uint16_t crc;
109
110     crc = 0xFFFF;
111     indice = 0;
112
113     do {
114         crc ^= a[indice];
115         nbDec = 0;
116
117         do {
118             if ( (crc&0x0001) == 1)
119                 yaun = 1;
120             else
121                 yaun = 0;
122             crc >>= 1;
123             if (yaun == 1)
124                 crc ^= 0xA001;
125             nbDec++;
126         } while (nbDec < 8);
127         indice++;
128     } while (indice < (a.size()-3));
129
130     return crc;
131 }
```

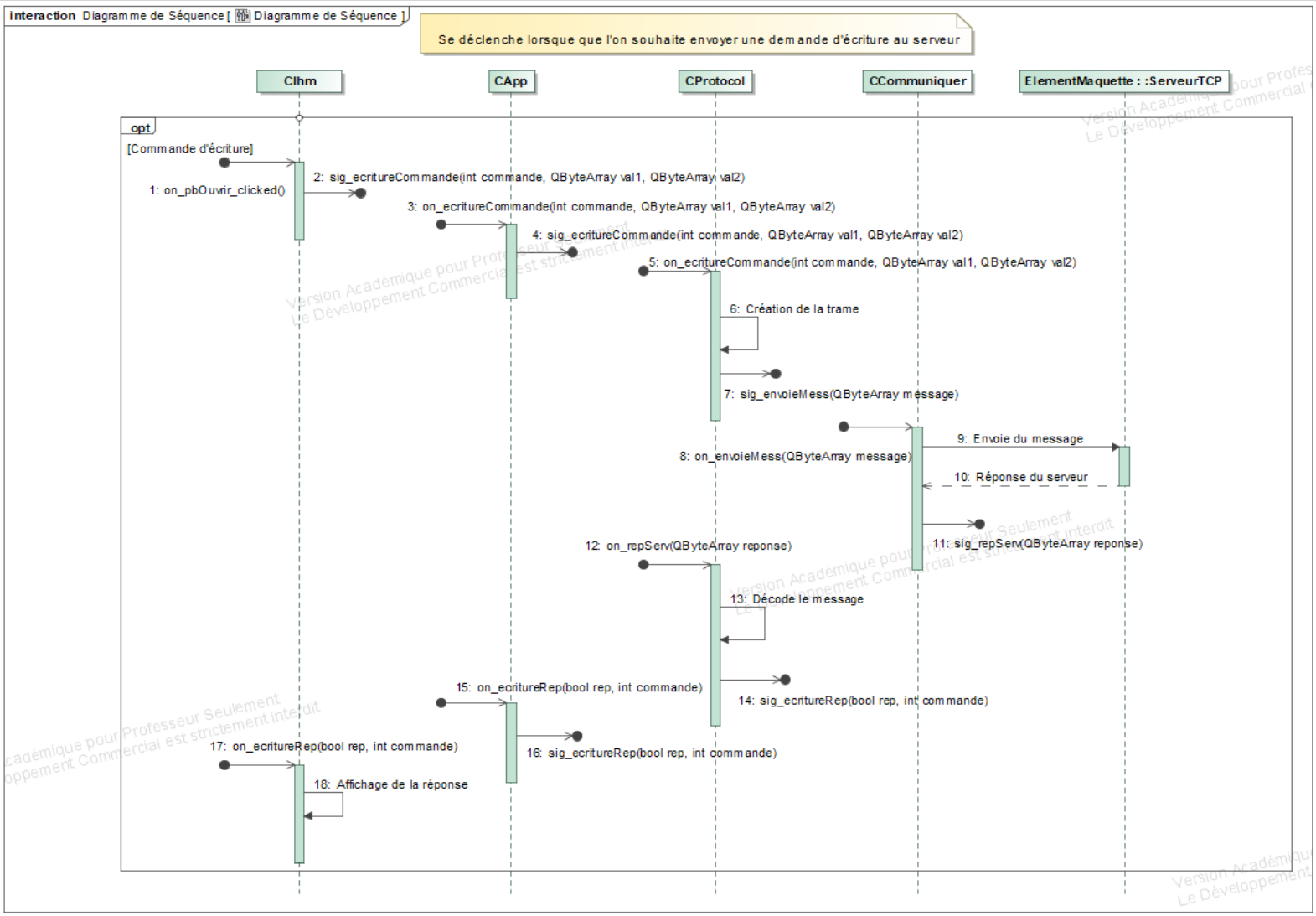
2.4 Diagramme de séquence



Voici un diagramme de séquence qui image ce qui se passe lorsque l'on appuie sur un bouton de test des états.
Le slot on_pTest() est appelé à l'appui du bouton, le signal est déclenché et est capté dans CCommuniquer par le slot on_commande. On demande à Cprotocol de créer une trame. CCommuniquer envoie la trame au serveur.

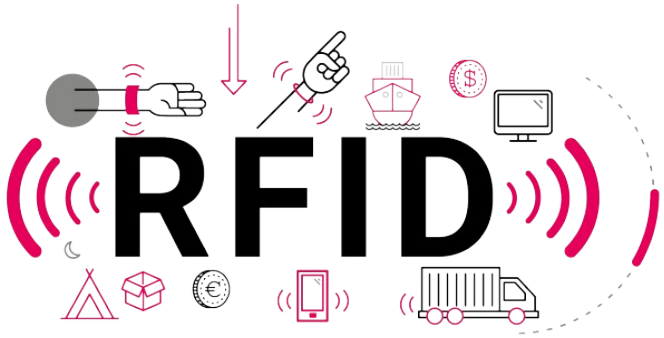
Dans Qt creator :
- Un slot est une fonction appelée en réponse à un signal particulier.
- Un signal est émis lorsqu'un événement particulier se produit.

Lorsqu'un signal est émis, les slots qui lui sont connectés sont généralement exécutés immédiatement, tout comme un appel de fonction normal.



Ce diagramme de séquence décrit ce qu'il se passe lorsque l'on fait une demande d'écriture.
Tout d'abord le slot on_pbOuvrir_clicked() (exemple de l'appuie bouton ouverture barrière) est appelé à l'appui du bouton, le signal sig_ecritureCommande() est déclenché et va voyager jusqu'à la classe CProtocol.
Cprotocol va créer la trame puis déclencher le signal sig_envoieMess() qui est capté par le slot on__envoieMess() dans CCommuniquer qui va envoyer la trame au serveur.
CCommuniquer après avoir reçu la réponse de serveur, va déclencher le signal sig_repServ() qui va à son tour voyager jusqu'à la classe Cprotocol, où le message sera décodé. Le signal sig_ecritureRep() est envoyé après le décodage et va voyager jusqu'au slot on_ecritureRep() dans Cihm. La réponse est ensuite affichée sur le client.

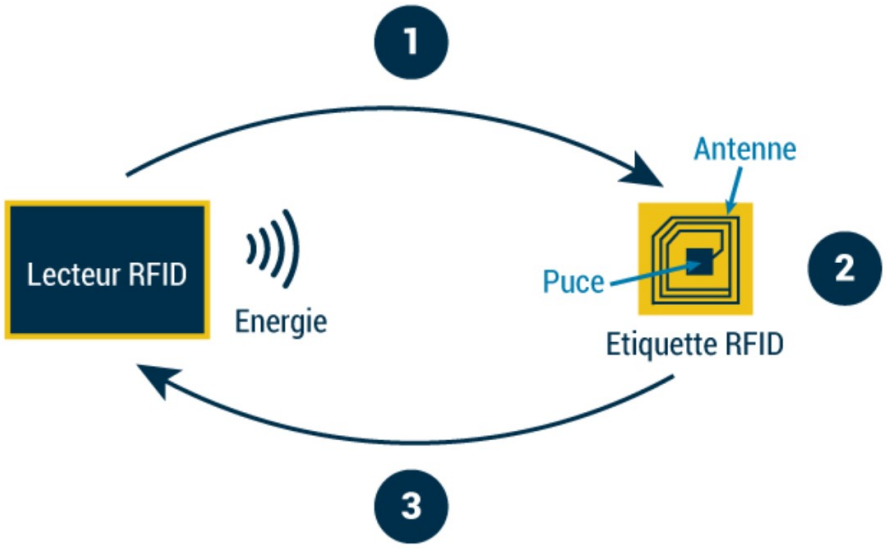
2.6 RFID



Qu'est-ce que la technologie RFID ?

La RFID, pour "Radio Frequency Idendification", est une technologie permettant de mémoriser, stocker, enregistrer des données sur un support et de les récupérer à distance. Elle existe depuis les années 1940 et servait, à l'époque, à identifier les avions de guerre entrant dans l'espace aérien du Royaume-Uni afin de les distinguer. D'abord utilisée par l'armée, la RFID s'est répandue dans différents secteurs industriels à partir des années 1980, de l'agroalimentaire à la santé, en passant par les transports.

Comment fonctionne la RFID ?



Les étiquettes RFID, qui peuvent aussi prendre la forme de balises ou de tags, sont composées d'une puce RFID et d'une antenne et sont collées sur un produit. Elles enregistrent les données et un lecteur électromagnétique lit ensuite les ondes radio présentes sur la puce RFID grâce à l'antenne. Le laboratoire de Conception et

d'Intégration des Systèmes à Grenoble INP travaille sur de nouvelles solutions RFID pour accroître les possibilités.

Quelles puces utiliser pour la RFID ?

Les puces se différencient en grande partie par la fréquence utilisée et la distance de lecture. Plus la fréquence est élevée, plus la distance de lecture peut être élargie. Il existe par ailleurs trois types de puces RFID :

- Les puces actives qui émettent elles-mêmes un signal sur une portée de 100 mètres et ont leur propre source d'alimentation
- Les puces semi-actives qui émettent un signal uniquement après en avoir reçu un
- Les puces passives qui ne communiquent leurs informations en lecture seule que lorsque l'action est demandée dans un rayon de 25 mètres seulement par le lecteur, dans lequel elles puisent leur énergie.

La puissance d'une étiquette RFID est en partie définie par les fréquences reçues et émises par l'antenne. Plus les fréquences sont basses, moins la puce est puissante et inversement. Ce facteur influence donc fortement la distance de lecture et peut être classé ainsi :

- Basse fréquence (système LF) : 125kHz

- Haute fréquence (système HF) : 13,56MHz

- Ultra haute fréquence (système UHF) : 433 et 860-960MHz

- Super haute fréquence (système SHF) : 2,45GHz

3 Fiches recettes

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :		Code de la campagne de test :	GR2 - IR2 - 01
Technicien 1 :	Chevalier Joséphine	Date :	15/04/2021

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Authentification au serveur
Objectif du scénario :	Envoyer une trame de connexion au serveur et recevoir sa réponse.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

- Serveur fonctionnel
- Syntaxe des trames correcte
- Code permettant l'authentification dans le client fonctionnel

EXÉCUTION DU TEST

Description courte :	Depuis le client Qt/C++, remplir les informations demandées valides : adresse Ip, port, identifiant et mot de passe, puis appuyer sur OK. Une trame est envoyé au serveur.
Résultats attendus :	Le serveur renvoie une trame disant si les informations sont correctes. Si c'est le cas la page d'authentification sur le client se ferme et nous avons accès à la partie contrôle de la maquette.

BILAN

L'authentification au serveur est fonctionnelle.

REMARQUES

RAS

CONCLUSION

VALIDE	X	NON VALIDE	

(Mettre une croix dans la case correspondante)

FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	smartcity	Code de la campagne de test :	GR2 - IR2 - 02
Technicien 1 :	Chevalier Joséphine	Date :	15/04/2021

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Système de sécurité
Objectif du scénario :	Recevoir un message sous forme de pop-up lorsqu'il y a un problème de connexion au serveur ou lorsque les informations fournies par l'utilisateur sont invalides. Impossible d'avoir accès à l'application.

CONDITIONS INITIALES NÉCESSAIRES POUR EFFECTUER LA RECETTE

- Serveur fonctionnel
- Syntaxe des trames correcte
- Code permettant l'authentification dans le client fonctionnel

EXÉCUTION DU TEST

Description courte :	Depuis le client Qt/C++, il faut envoyer des informations invalides ou fermer le serveur puis appuyer sur OK. Une trame est envoyé au serveur.
Résultats attendus :	Si le serveur est ouvert il envoie une trame disant si les informations sont invalides. Sinon un message disant que la connexion au serveur est impossible.

BILAN

Le système de sécurité fonctionne.

REMARQUES

RAS

CONCLUSION

VALIDE	X	NON VALIDE	

(Mettre une croix dans la case correspondante)

4 - Objectifs pour la suite du projet

A ce jour le Client Qt/C++ est presque terminé il faut juste intégrer les trames d'écriture et de lecture dans le code.
Je vais utiliser le temps qu'il me reste jusqu'à la fin du projet pour le faire.
Il ne restera plus qu'à tester le client après avoir rassemblé toutes les parties du projet pour savoir si cela fonctionne.
Je pense que tout sera opérationnel dans peu de temps.

Partie Étudiant IR 3 – Équipe 02 : DANCUO Lohan

1. Matériel :

-1 Raspberry Pi 2
-1 Câble Ethernet
-1 Câble HMDI
-1 Micro Carte SD 8Go

2. Objectifs :

-Mettre en place un site internet affichant les états de la maquette Smart City.

-Installer un serveur Apache2 et PHP sur la Raspberry Pi 2
-Créer une base de donnée sur le serveur PHP My Admin de la Raspberry
-Créer un programme en C++ permettant la réception des données entre la maquette et les clients (Web, C++, C#)
-Créer le site internet

3. Planification :

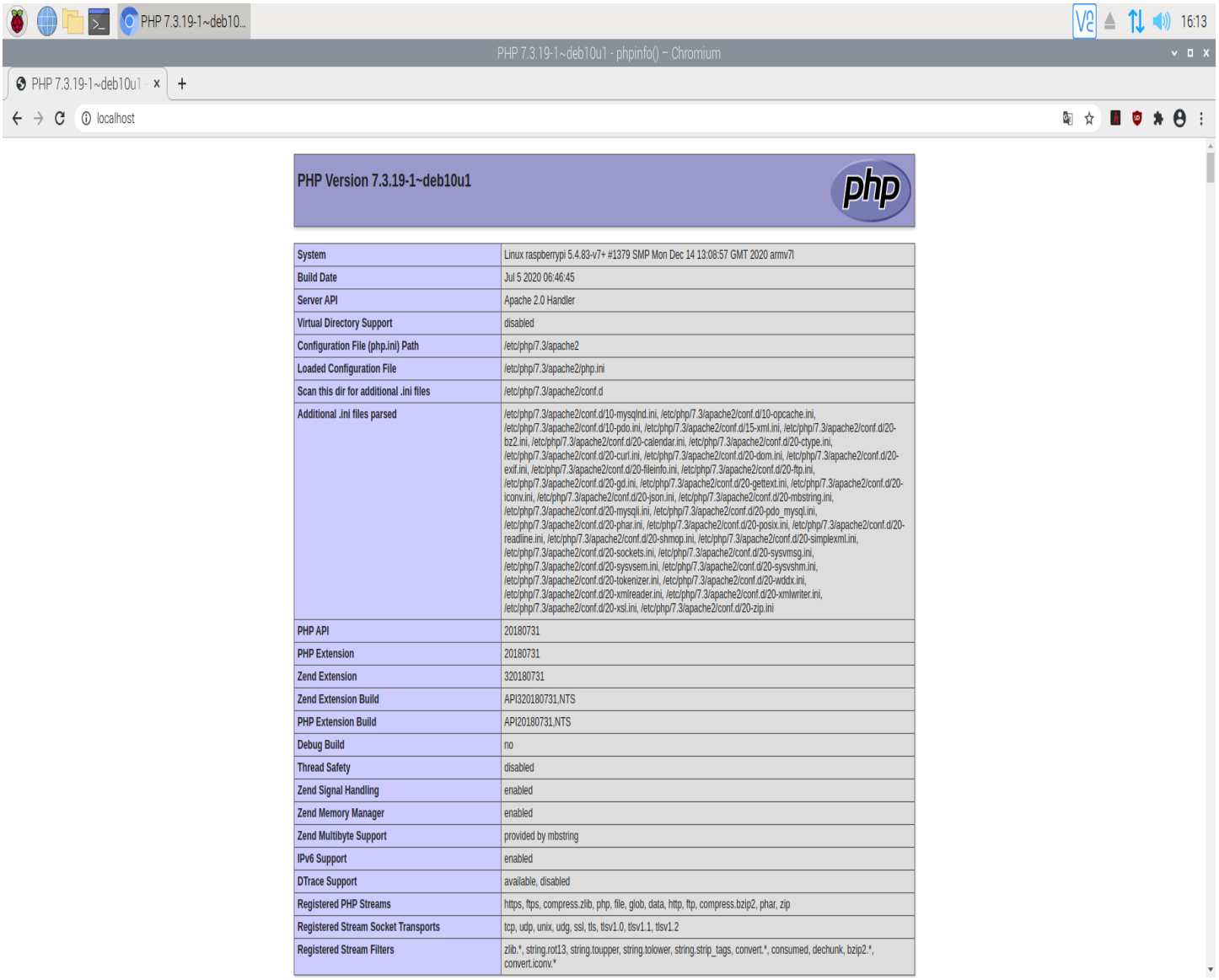
Spécification générale	Mar 05/01/21	Mer 06/01/21	5 h
Analyse et conception SYSML	Mer 06/01/21	Mar 12/01/21	13 h
Installation LAMP	Mar 12/01/21	Jeu 14/01/21	7 h
Conception de la BDD	Jeu 14/01/21	Mar 26/01/21	26 h
Mise en service de la BDD	Mar 26/01/21	Jeu 28/01/21	7 h
Conception de la classe CBdd	Jeu 28/01/21	Mar 09/02/21	26 h
Programme de mise en oeuvre de la classe CBdd	Mer 10/02/21	Ven 12/02/21	12 h
Préparation dossier revue N°1	Mar 16/02/21	Ven 19/02/21	12 h
Revue 1	Ven 19/02/21	Ven 19/02/21	1 h

Revue 1	Ven 19/02/21	Ven 19/02/21	1 h	9
Création d'un modèle pour l'IHM	Ven 19/02/21	Mer 10/03/21	8 h	10
Création de l'IHM (HTML+CSS)	Mer 10/03/21	Jeu 18/03/21	22 h	11
Mise en lien de la caméra sur le site web	Ven 19/03/21	Ven 26/03/21	20 h	12
Mise en lien du site avec la BDD	Ven 26/03/21	Jeu 08/04/21	26 h	13
Test de la BDD depuis le site	Jeu 08/04/21	Mer 14/04/21	14 h	14
Test avec les autres parties du groupe	Mer 14/04/21	Mar 04/05/21	27 h	15
Revue 2	Mar 04/05/21	Mar 04/05/21	1 h	16

Revue 2	Mar 04/05/21	Mar 04/05/21	1 h	16
Résolution d'éventuels bugs	Mar 11/05/21	Mar 11/05/21	4 h	17
Intégration avec toutes les autres parties du groupe	Mer 12/05/21	Ven 21/05/21	24 h	18
Test et résolutions d'éventuels bugs	Ven 21/05/21	Mar 01/06/21	24 h	19
Peaufinage	Mer 02/06/21	Jeu 17/06/21	35 h	20
Revue 3	Jeu 17/06/21	Jeu 17/06/21	1 h	17

4/ Installation du serveur Apache2 et PHP :

Au tout début du projet, j'ai installé un serveur Apache2 et PHP sur la Raspberry pour pouvoir ensuite créer la base de donnée. L'installation est assez simple, et une fois terminée nous avons cette page qui s'affiche qui nous informe que les deux serveurs se sont bien installé. J'ai ensuite installé le serveur PHP My Admin pour pouvoir mettre la base de donnée dessus.

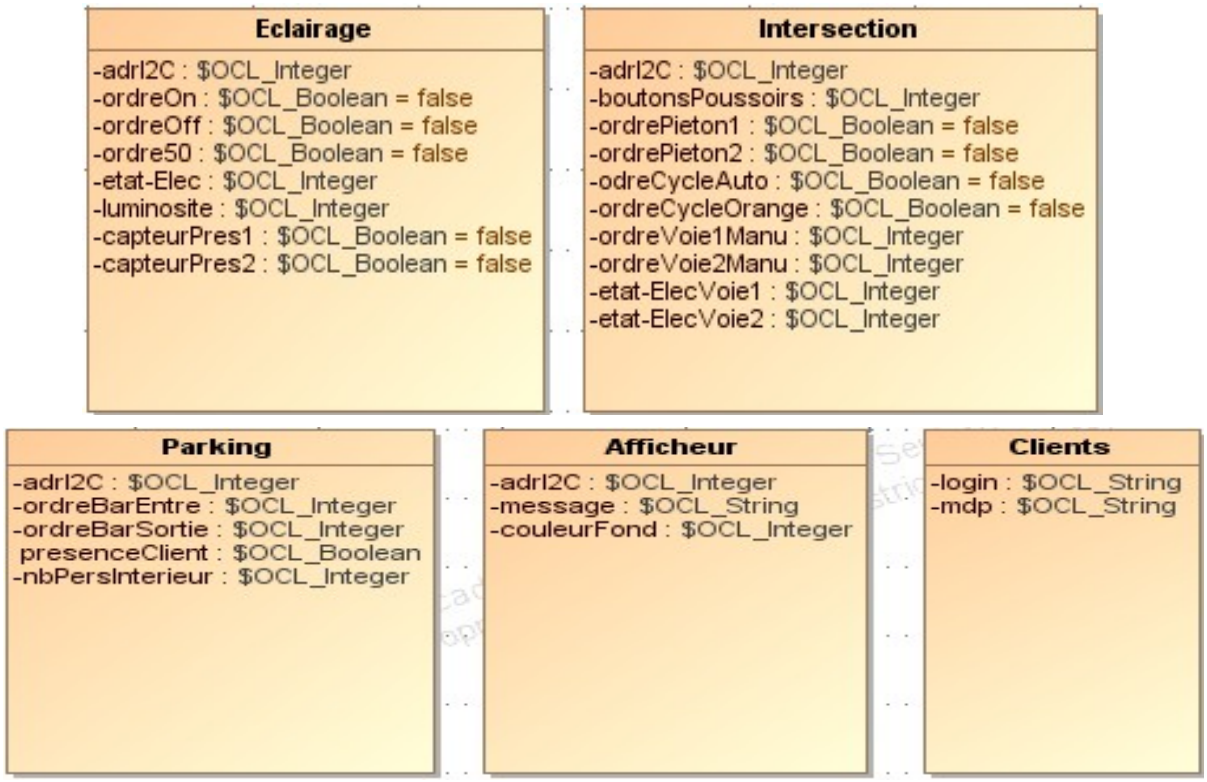


System	Linux raspberrypi 5.4.83-v7+ #1379 SMP Mon Dec 14 13:08:57 GMT 2020 armv7l
Build Date	Jul 5 2020 06:46:45
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.3/apache2
Loaded Configuration File	/etc/php/7.3/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.3/apache2/conf.d
Additional .ini files parsed	/etc/php/7.3/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.3/apache2/conf.d/10-opcache.ini, /etc/php/7.3/apache2/conf.d/10-pdo.ini, /etc/php/7.3/apache2/conf.d/15-xml.ini, /etc/php/7.3/apache2/conf.d/20-bz2.ini, /etc/php/7.3/apache2/conf.d/20-calendar.ini, /etc/php/7.3/apache2/conf.d/20-ctype.ini, /etc/php/7.3/apache2/conf.d/20-curl.ini, /etc/php/7.3/apache2/conf.d/20-dom.ini, /etc/php/7.3/apache2/conf.d/20-exif.ini, /etc/php/7.3/apache2/conf.d/20-fileinfo.ini, /etc/php/7.3/apache2/conf.d/20-ftp.ini, /etc/php/7.3/apache2/conf.d/20-gd.ini, /etc/php/7.3/apache2/conf.d/20-gettext.ini, /etc/php/7.3/apache2/conf.d/20-iconv.ini, /etc/php/7.3/apache2/conf.d/20-json.ini, /etc/php/7.3/apache2/conf.d/20-mbstring.ini, /etc/php/7.3/apache2/conf.d/20-mysqli.ini, /etc/php/7.3/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.3/apache2/conf.d/20-phar.ini, /etc/php/7.3/apache2/conf.d/20-posix.ini, /etc/php/7.3/apache2/conf.d/20-readline.ini, /etc/php/7.3/apache2/conf.d/20-shmop.ini, /etc/php/7.3/apache2/conf.d/20-simplexml.ini, /etc/php/7.3/apache2/conf.d/20-sockets.ini, /etc/php/7.3/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.3/apache2/conf.d/20-sysvsem.ini, /etc/php/7.3/apache2/conf.d/20-sysvshm.ini, /etc/php/7.3/apache2/conf.d/20-tokenizer.ini, /etc/php/7.3/apache2/conf.d/20-wddx.ini, /etc/php/7.3/apache2/conf.d/20-xmlreader.ini, /etc/php/7.3/apache2/conf.d/20-xmlwriter.ini, /etc/php/7.3/apache2/conf.d/20-xsl.ini, /etc/php/7.3/apache2/conf.d/20-zip.ini
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	API320180731,NTS
PHP Extension Build	API20180731,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, compress.bzip2, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, bzip2.*, convert.iconv.*

Sur cette page nous avons la version du serveur PHP installé, et tout les suppléments de PHP ou de Apache téléchargé. Comme par exemple l'extension MySQL qui nous servira à la création de la base de donnée.

5/ Création de la base de donnée :

Diagramme de classe

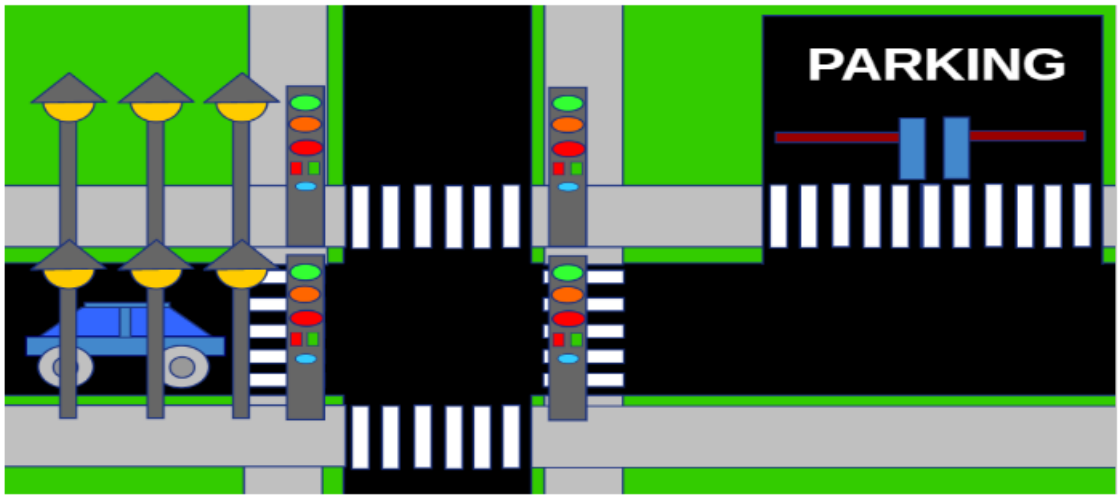


Chaque classe aura comme clé unique l'adresse I2C des micro contrôleurs qui seront en rapport avec leurs fonctions.
Par exemple, les adresses I2C de la fonction Éclairage iront de 20 à 50.
Ce diagramme montre juste les données stockés dans chaque classe

6/ Création de maquettes :

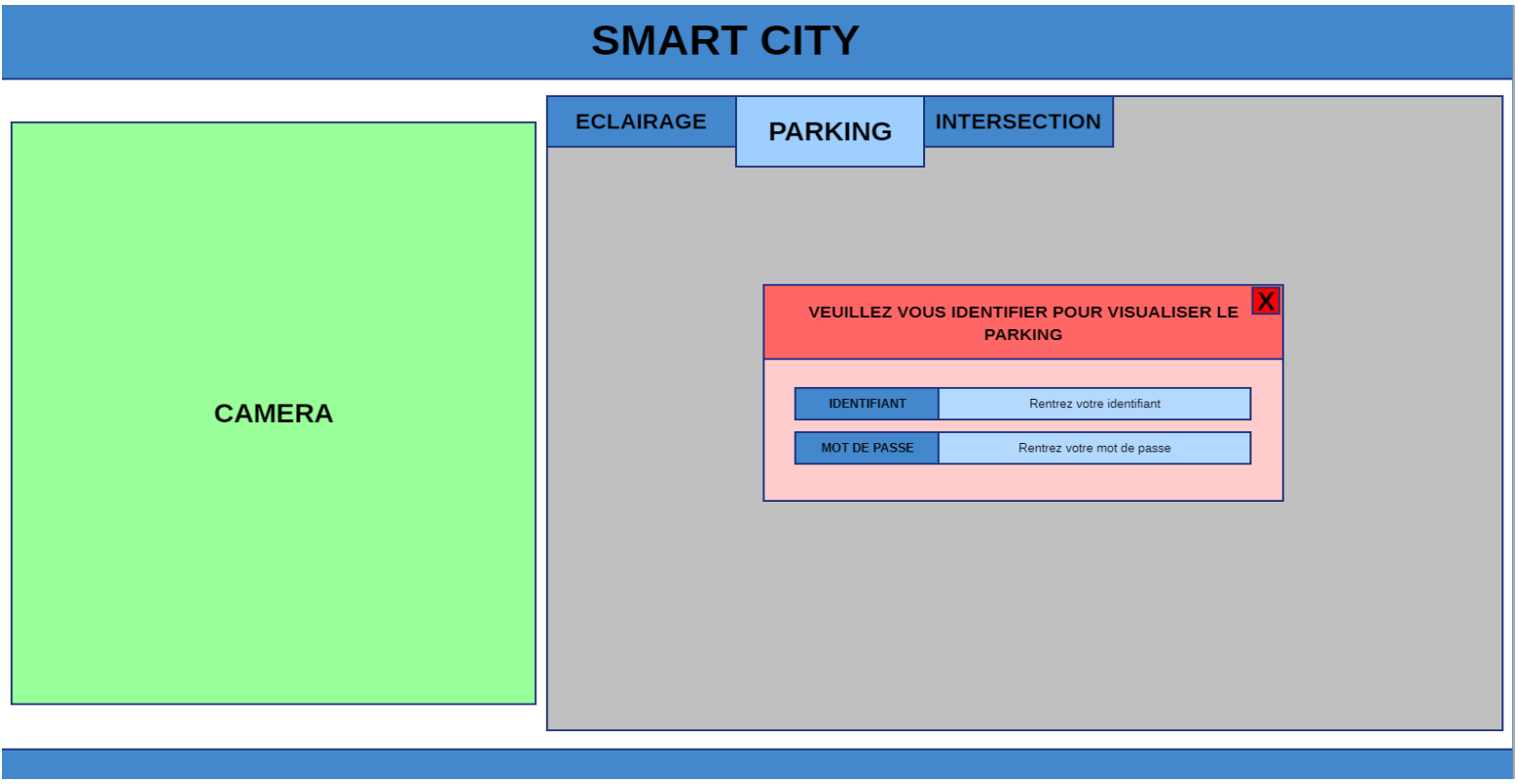
Maquette factice

J'ai créer une maquette factice pour que notre groupe puisse se baser dessus, car nous n'avions aucunes photos ou exemple de la maquette réelle.



Maquette du site internet

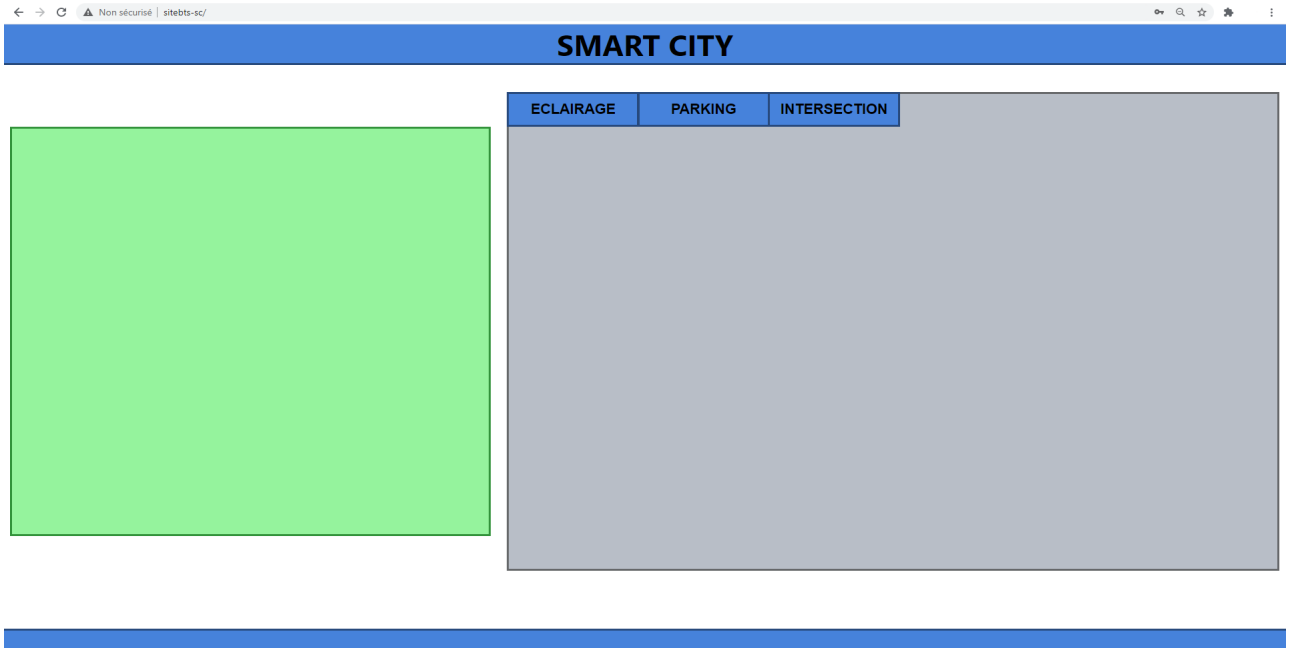
Pour le site internet j'ai voulu créer sa maquette pour être fixé pendant sa réalisation. Je m'impose donc de respecter ma maquette dans la réalisation du site internet. Je l'ai réalisé grâce au logiciel Pencil.



Nous devrons nous identifier pour avoir accès à la caméra et à la visualisation de la maquette, que ce soit du parking, de l'éclairage ou de l'intersection.

7/ Conception du site web :

Grâce à la maquette j'ai pu aisément créer le site web du projet

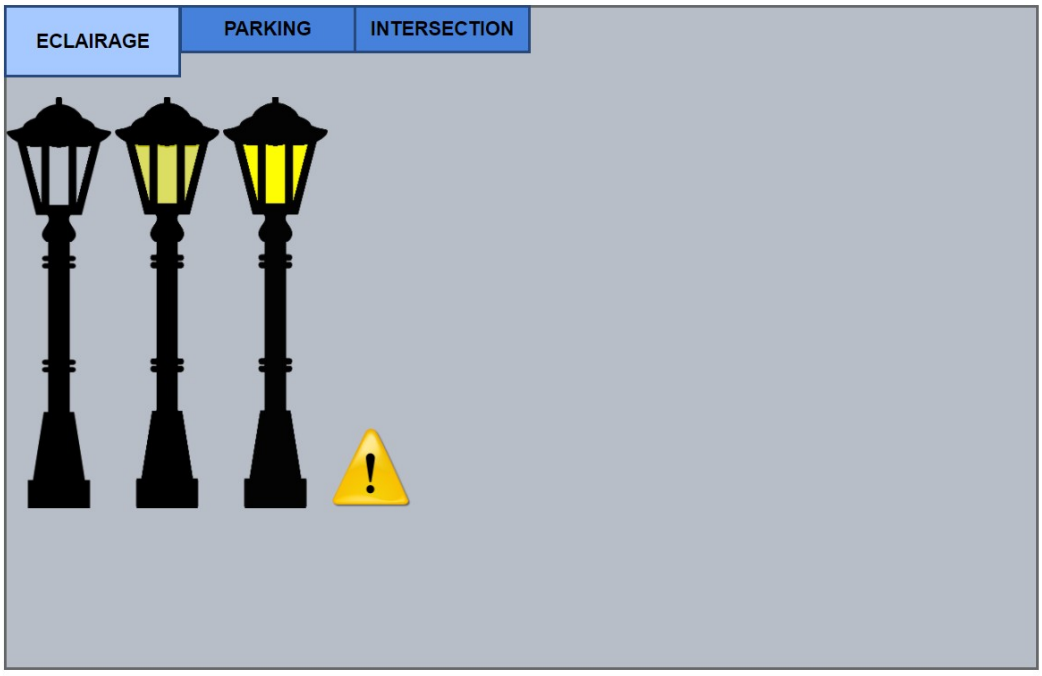


Il nous faut ensuite ajouter du javascript pour pouvoir choisir/changer d'onglet, c'est un peu plus complexe que de rediriger vers une autre page web mais au moins, on n'aura pas besoin de recharger le flux vidéo de la caméra à chaque fois.
Tout d'abord je vais modifier mon fichier CSS pour modifier les bouton « ECLAIRAGE », « PARKING » et « INTERSECTION », je veux que quand la sourie passe dessus, qu'ils changent de couleur et deviennent un peu plus grand.

Il faut ensuite créer un fichier javascript qui permet de faire apparaître ou non le contenu des différentes sections. Pour la section éclairage ça donne ceci :

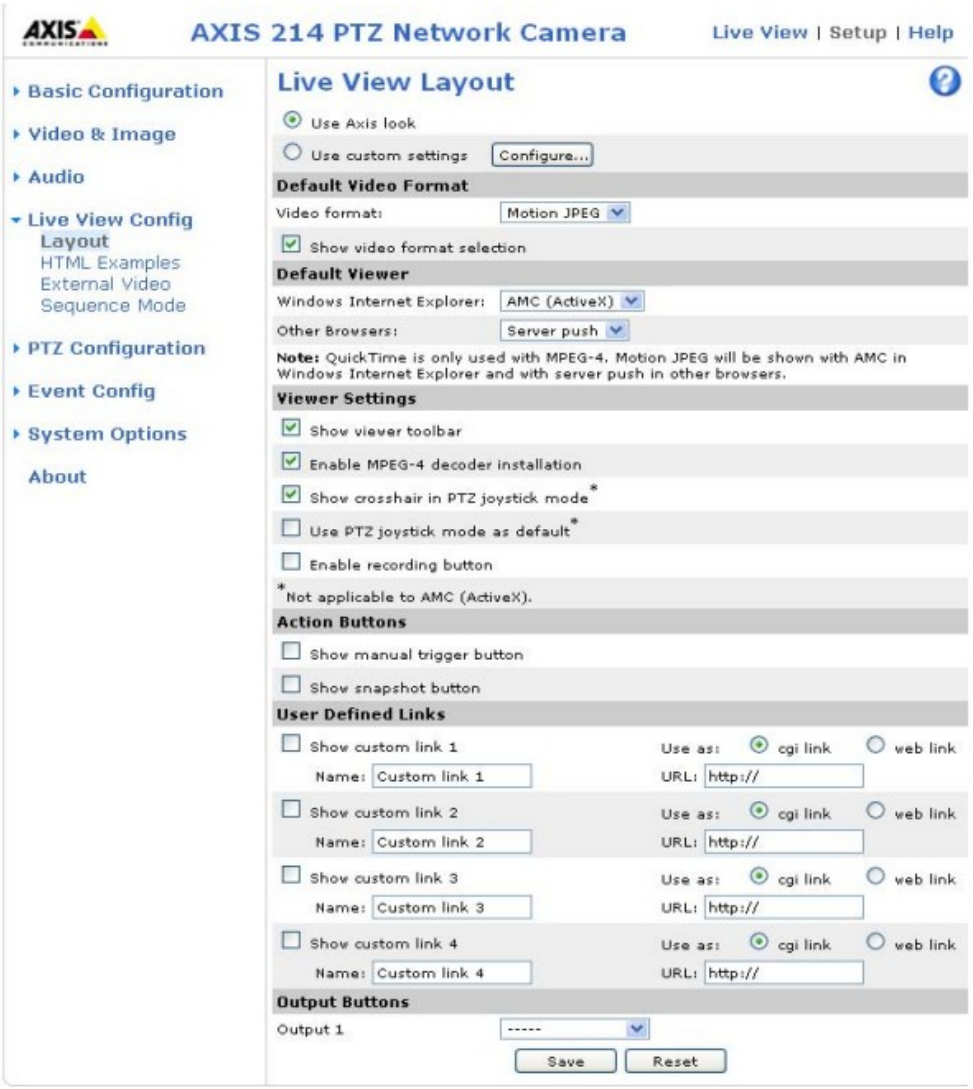
```
$("#btEclairage").click(function () {
    document.getElementById("vide").hidden = true;
    document.getElementById("eclairage").hidden = false;
    document.getElementById("btEclairage").classList.add("eclairageOn");
    document.getElementById("parking").hidden = true;
    document.getElementById("btParking").classList.add("parking");
    document.getElementById("btParking").classList.remove("parkingOn");
    document.getElementById("intersection").hidden = true;
    document.getElementById("btIntersection").classList.add("intersection");
    document.getElementById("btIntersection").classList.remove("intersectionOn");
```

Pour résumer, quand on clique sur le bouton « ECLAIRAGE » on laisse le bouton en grand (comme si on avait laissé la souris dessus), ensuite on fait disparaître visuellement les autres sections si elles ont déjà étaient chargées. Pour ensuite chargé la section éclairage.



8/ Mise en relation du site web et de la caméra :

J'ai à ma disposition une caméra IP Axis PTZ214. Nous pouvons nous connecter à son site web via son adresse ip :



Sur le site de la caméra, tout comme sur son mode d'emploi, on remarque que nous pouvons voir le flux vidéo de la caméra via un code HTML :

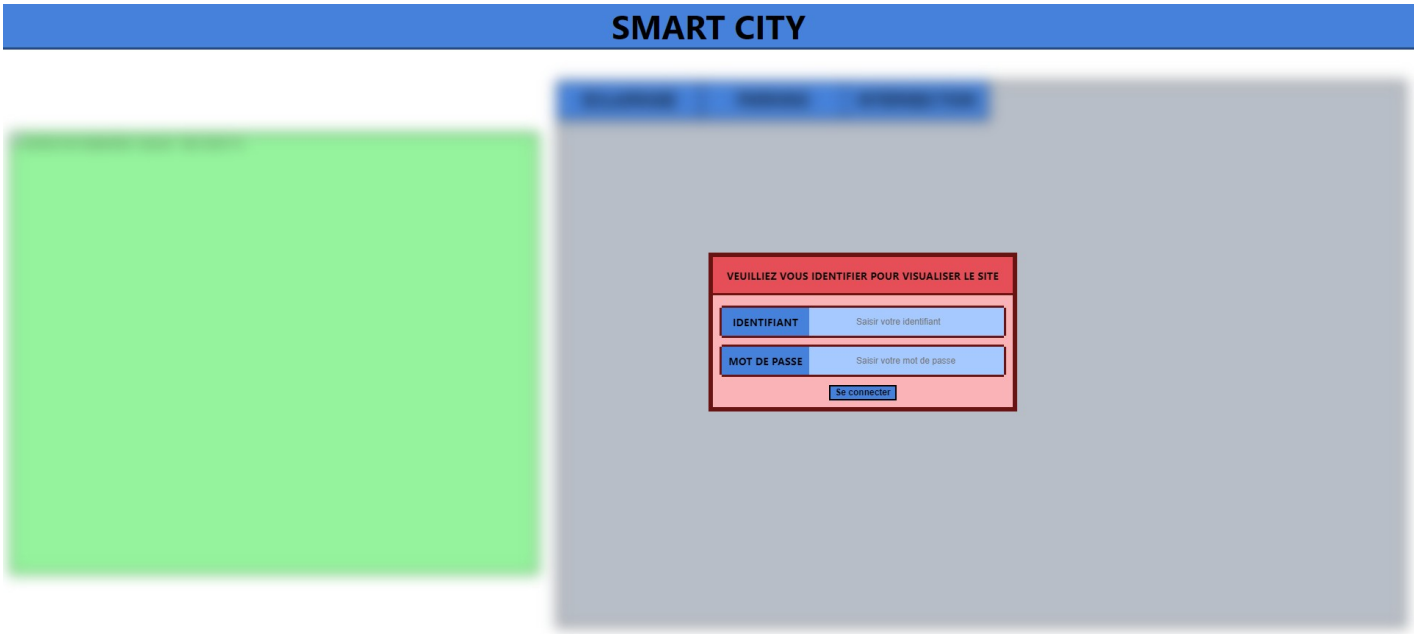
```
<img draggable="false"
src="http://192.168.4.116/axis-cgi/mjpg/video.cgi?resolution=4CIF&amp;clock=1&amp;date=1&text=1&textstring=CAMERA%20PARKING%20:%20"
alt="Camera non disponible / source : 192.168.4.116" class="shrinkToFit" width="100%" min-height="100%">
```

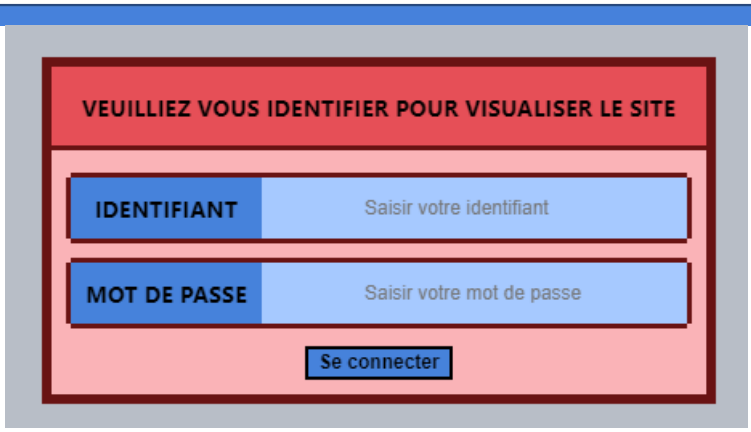
On instancie donc le flux vidéo de la caméra comme étant une image sur le site web. J'ai rajouté un message d'erreur si le flux vidéo n'apparaît pas.

Camera non disponible / source : 192.168.4.116

9/ Ajout d'un pop-up de connexion obligatoire :

Il faut ensuite pouvoir se connecter pour accéder au site. Pour ne pas que quiconque puisse avoir accès à la supervision de la maquette. Pour se faire je vais créer en premier plan un pop-up qui rendra flou l'arrière plan du site.





Actuellement pour tester si l'utilisateur saisi est le bon, je vérifie si la chaîne de caractère rentrée correspond à une autre chaîne de caractère. Je ne l'ai pas encore relié à la base de données.

```
$(document).keyup(function(event) {
    if (($("#identifiant").is(":focus") || $("#motdepasse").is(":focus")) && event.key == "Enter") {
        id = document.getElementById("identifiant").value;
        mdp = document.getElementById("motdepasse").value;
        if (id == 'root' && mdp == 'admin'){
            document.getElementById("blur").classList.remove("html-blur");
            document.getElementById("msgConnexion").hidden = true;
        }else {
            window.alert("Identifiant ou mot de passe incorrecte !");
        }
    }
});
```

Sur le code ci-dessus, on test juste si l'identifiant rentré correspond à « root » et le mot de passe rentré à « admin ». Si les deux chaînes de caractères rentrées correspondes à ce qui est attendu. Le pop-up de connexion disparaît et enlève le flou du site web.

10 Capteur RFID

Dans le projet et plus particulièrement pour le parking, des capteurs RFID sont utilisés pour vérifier si un abonné est proche ou non des barrières du parking. La radio-identification est utilisé dans différents domaines, par exemple pour le paiement sans contact, l'identification et la traçabilité de produits, l'identification de personnes via des badges etc… Il y a différents type de capteur RFID, ceux passifs et ceux actifs. Les passifs attendent un signal pour renvoyé leurs contenu, ils ne possèdent pas de source d'alimentation. Les actifs peuvent envoyer un signal de façon autonome car ils possèdent une source d'alimentation.
Plus un capteur RFID possède un haute fréquence, plus la distance de leurs signal émis est grande.

	Technologie	Fréquence
Badge d'accès	Passive	125 kHz
Identification d'animaux	Passive	134,2 kHz
Colis postaux	Passive	13,56 MHz
Passeport biométrique	Passive	13,56 MHz
Identification textiles blanchisserie	Passive	13,56 MHz
Titre de transport	Passive	13,56 MHz
Paiement sans contact	Passive	13,56 MHz
Bagagerie aéroport	Passive	UHF
Identification wagons	Passive	UHF
Logistique chaîne d'approvisionnement	Passive	UHF
Péage autoroutier	Active	2,45 GHz

Par exemple pour un badge d'accès avec une fréquence de 125 kHz, la distance du signal émis sera d'environ une dizaine de centimètres. Tandis que pour un péage routier qui à une fréquence de 2,45 GHz peut avoir une distance du signal émis de plusieurs dizaines de mètres

11/ Objectifs pour la prochaine revue :

Pour être en corrélation avec mon cahier des charges, je dois finir l'IHM du site internet et améliorer la réactivité entre la base de données et sa visualisation sur le site web. Il me reste aussi à effectuer les derniers tests d'intégration.

12 - FICHE RECETTE, QUALIFICATION DU SYSTÈME

Nom du système :	smartcity	Code de la campagne de test :	GR2 - IR1 - 01
Technicien 1 :	DANCUO Lohan	Date :	26/05/2021

IDENTIFICATION DU SCENARIO

Identification du scénario de recette	
Titre :	Lecture de données sur la base de données depuis le serveur web en local.
Objectif du scénario :	Lire les données de la BDD et les visualiser sur le site internet.

CONDITIONS INITIALES NECESSAIRES POUR EFFECTUER LA RECETTE

-Serveur web fonctionnel

-Site web pouvant se connecter à la base de données

-Base de données établie

Description courte :	Depuis le site internet, pouvoir visualiser les changements de données de la base de données et les afficher en direct.
Résultats attendus :	Le site n'a pas besoin d'être actualiser pour afficher les changements de données de la base de données.

BILAN

Les données modifiées de la BDD sont bien visualisées par le site internet. Les changements brutaux de données sont aussi visualiser sur le site internet.

REMARQUES

R.A.S.

CONCLUSION

VALIDE	X	NON VALIDE	

(Mettre une croix dans la case correspondante)

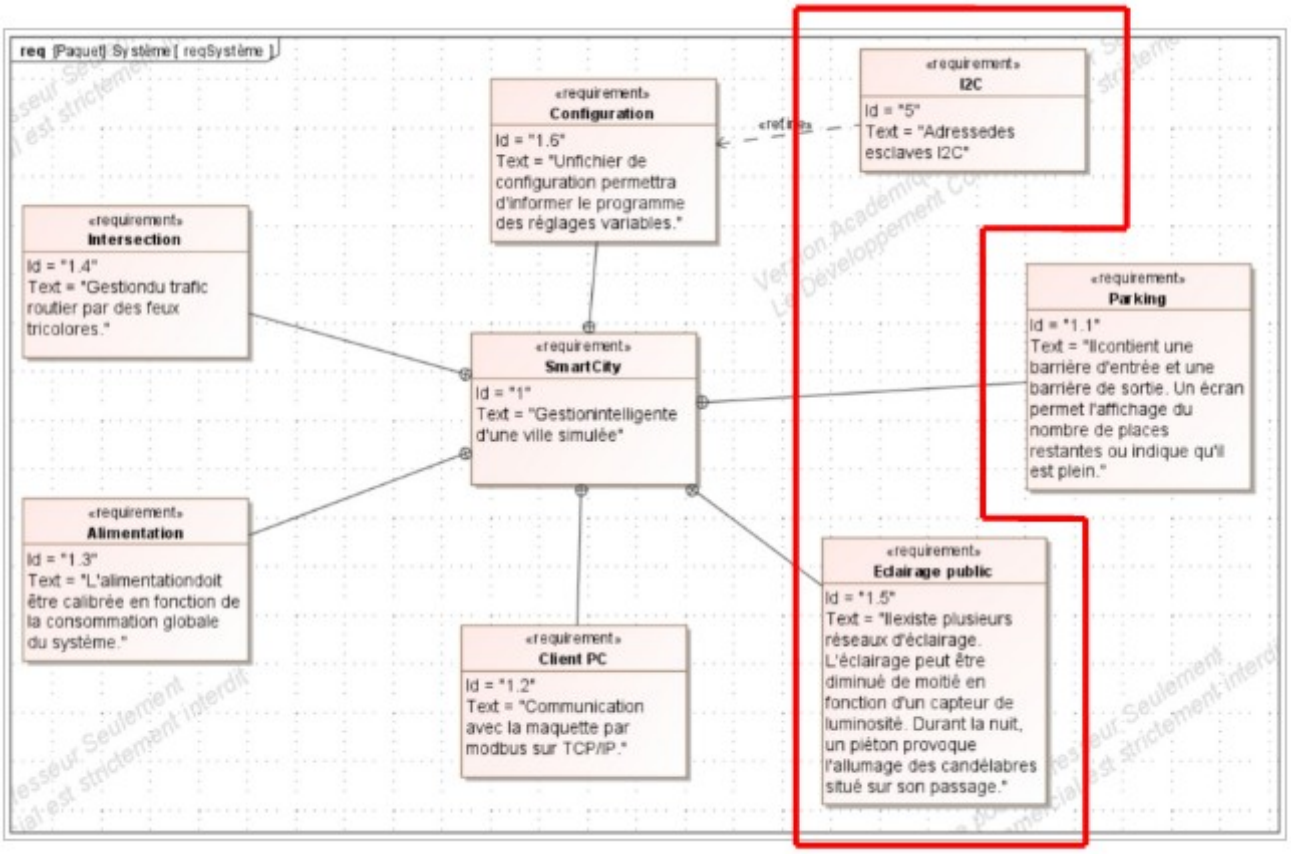
Partie Étudiant EC 1 – Équipe 02 : NINNIN Nicolas

1. Introduction

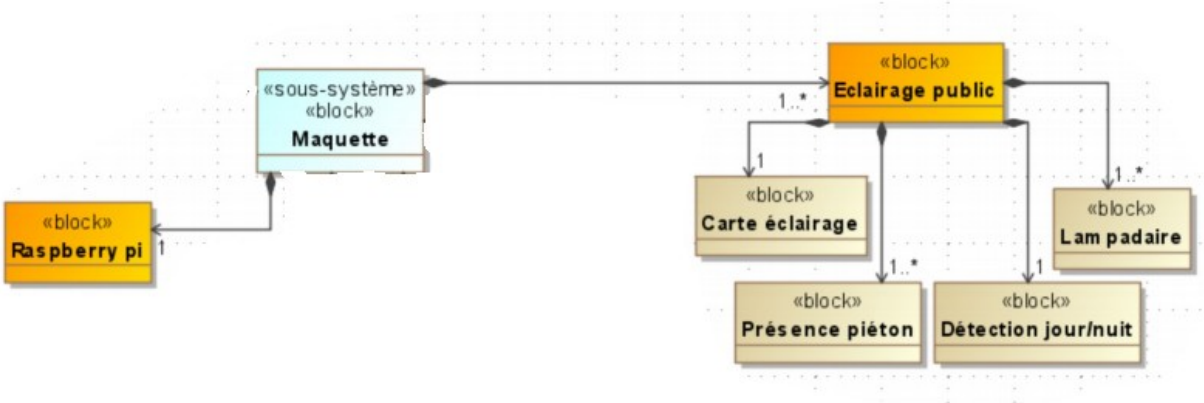
1.1 Cahier des charges de l'étudiant EC21

Étudiant 4 **EC 21**	*Liste des tâches assurées par l'étudiant* **Éclairage public** • Concevoir une carte d'éclairage public permettant de gérer un tronçon de 6 lampadaires, avec détection de présence à chaque extrémité du tronçon et mesure de l'intensité lumineuse pour adapter l'éclairement. • Un schéma structurel est proposé, effectuer tous les tests nécessaires pour valider les structures, et les modifier si nécessaire. • Participer à l'élaboration d'un protocole de communication sur le bus I2C avec l'étudiant IR concerné. • Participer à la conception des parties mécaniques du parking. • Effectuer la saisie du schéma et le routage de la solution retenue. Produire les fichiers Gerber afin que la fabrication du PCB soit sous-traitée. • Câbler la carte et effectuer les essais. • Documenter la mise en service de la carte finalisée.	**Installation :** IDE Arduino. **Mise en œuvre :** Tester/valider/modifier une structure utilisant un microcontrôleur ATtiny84 pour piloter une carte permettant de gérer 6 lampadaires, 2 détecteurs de présence et un capteur de luminosité. La carte fonctionnera en tant que circuit esclave sur le bus I2C. Si, suite aux essais, le microcontrôleur ne convient pas, un autre devra être proposé et testé . Participer à une réflexion commune avec tous les étudiants EC du projet SmartCity pour le câblage du faisceau distribuant l'alimentation et le bus I2C sur tout le démonstrateur. **Réalisation :** Après validation de la solution, concevoir un circuit imprimé devant être fabriqué industriellement. Participer à la partie réalisation du démonstrateur. Mettre en œuvre le faisceau de communication/alimentation. **Documentation :** • Schéma de câblage rapide (Fritzing) pour documenter la phase d'essais. • Documents de fabrication de la carte (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. • Schéma structurel avec contours IBD. • Liste complète des composants avec leurs sources d'approvisionnement et leurs prix. • Programme de gestion de la carte, accompagné des commentaires et diagrammes nécessaires à sa compréhension. • Fiche de mise en service. • Fiche de dépannage.

1.2 Diagramme des exigences EC 21

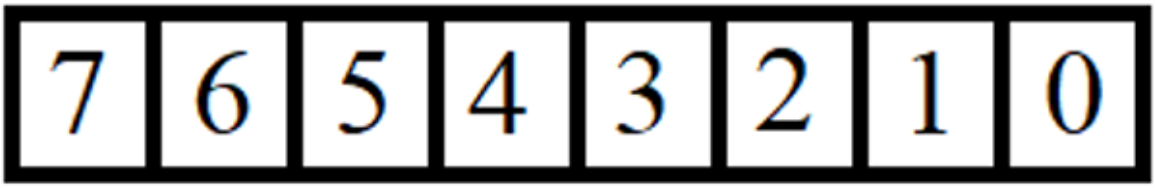


1.3 Diagramme de bloc



Après l'étude des différents diagrammes et du cahier des charge. Je me rends compte que ma partie du projet consiste à mettre au point un réseau de lampadaire avec 3 modes d'éclairage (éteint, allumé de moitié et allumé). Elle consistera aussi à mettre en place un capteur de luminosité pour détecter s'il fait jour où nuit. Mais aussi d'un capteur de présence pour détecter les piétons. Toutes ses informations remonteront à un Raspberry qui commandera aussi la carte.

1.4 Protocole I2C



En lecture :	En écriture :
Bit7 : État du 6ème lampadaire	Bits 2 : Ordre d'éclairage à 100%
Bit6 : État du 5ème lampadaire	Bits 1 : Ordre d'éclairage à 50%
Bit5 : État du 4ème lampadaire	Bits 0 : Ordre d'éclairage à 0%
Bit4 : État du 3ème lampadaire	
Bit3 : État du 2ème lampadaire	
Bit2 : État du 1er lampadaire	
Bit1 : Jour/Nuit	
Bit0 : Présence/Absence	

Avec l'élève d'IR qui gère la communication en I2C, on a établi un protocole de communication entre ma carte et la Raspberry Pi. La communication se fera donc sur 1 octet en lecture et en écriture.

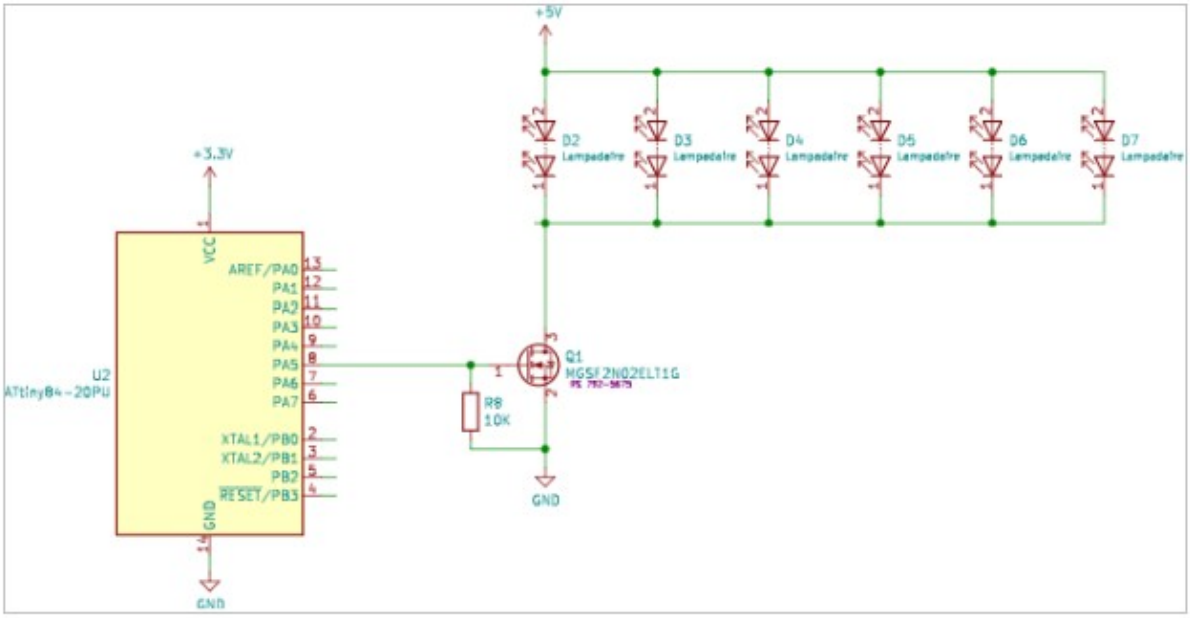
1.6 Matériel à ma disposition

- ATTiny 84
- Photorésistance : NSL-19M51
- Capteur réfléchissant : Sharp GP2A200LCS0F
- Adaptateur double USB femelle / connecteur 10 broches
- Lampe LED USB Flexible Xiaomi Mi – Blanc
- Transistor : MGSF2N02ELT1G

2. Travail réalisé pour la revue

2.1 Validation des lampadaires

2.1.1 Schéma structurel

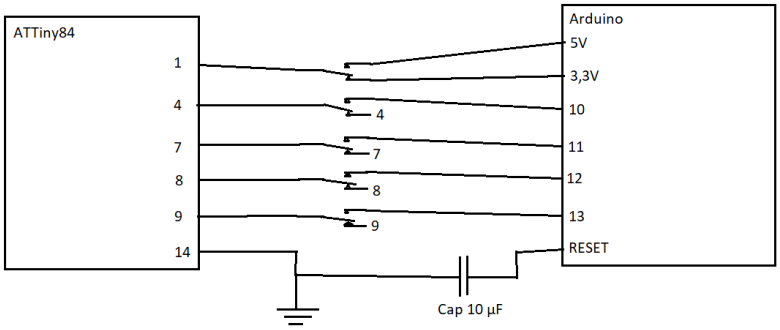


J'ai d'abord essayé le schéma proposé pour l'éclairage de mes LED commandé par une ATTiny84 (alimenté en 3.3V) mais séparé par un transistor pour alimenter les LED en 5V.

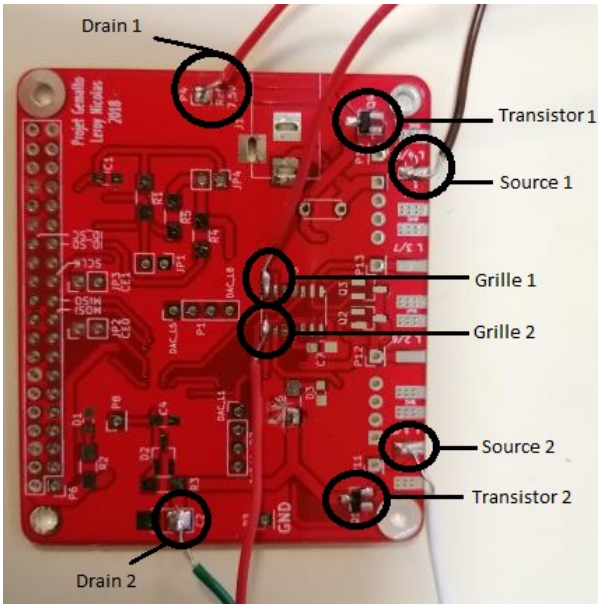
2.1.2 Problèmes et Solutions

1er problème : Programmation de l'ATTiny84

L'ATTiny84 est programmable via un Arduino. Cependant certaines broches sont utilisées pour la programmation mais aussi en sortie. Donc pour ne pas avoir à recâbler à chaque nouvelle programmation j'ai mis en place des commutateurs pour alterner entre le mode programmation et fonctionnement normal.



2eme problème : Les transistors en cms



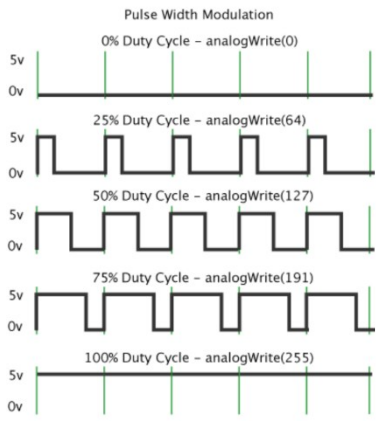
Le transistor étant en cms je l'ai soudé à une carte pour pouvoir récupérer les broches facilement. J'ai donc soudé les deux transistors dont j'aurais besoin.

3eme Problème : Fonctionnement câble USB

Mes lampadaires fonctionnant en USB j'ai dû m'informer sur le fonctionnement de celui-ci. Et j'ai trouvé ce schéma facile de compréhension :

L'USB est principalement constitué de 4 fils, le rouge pour l'alimentation, le noir pour la masse et le blanc et vert comme bus de données en différentiels. Cependant je n'aurais pas besoin des fils de données mais seulement du Vcc (en 5V) et de la masse.

4emeproblème : Programmation LED en PWM



Etant donné que les LED doivent être réglable, je choisis de commander le transistor par PWM (Pulse Width Modulation).
Cette méthode permet grâce à une variation du rapport cyclique d'obtenir des résultats similaires a une sortie analogique avec une sortie numérique. Donc dans mon cas on pourra faire varier l'intensité des LED entre 100% (5V), 50% (2,5V) et 0% (0V).

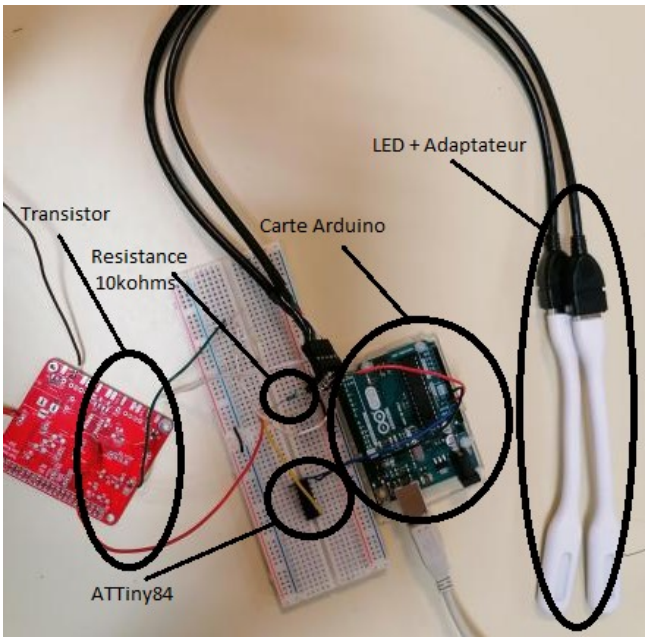
2.1.3 Le programme

```
#define LED 5

void setup()
{
  pinMode(LED,OUTPUT);
}

void loop()
{
  analogWrite(LED, 255);
  delay(1000);
  analogWrite(LED, 127);
  delay(1000);
  analogWrite(LED, 0);
  delay(1000);
}
```

D'abord j'ai attribué le « nom » LED à la broche qui commande les lampadaire (la broche 5 de l'ATTiny84) et je l'attribue ensuite en sortie.

Ensuite je fais clignoter les LED a 100%, 50% et 0% d'intensité avec 1 seconde de pause.

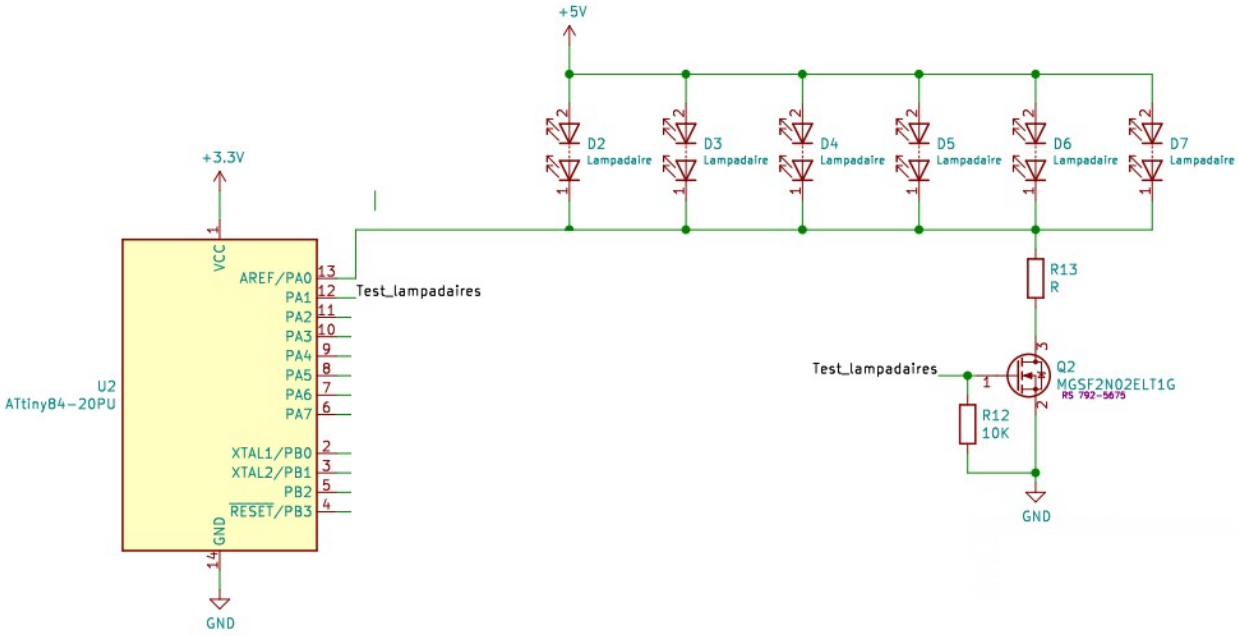
2.1.4 Montage réel



À la suite de ces études j'ai finalisé mon montage et pu le tester grâce au programme réalisé. Cependant en réalité l'ATTiny84 ne commande pas automatiquement les LED mais elle reçoit des ordres d'une Raspberry PI.

2.2 Validation du testeur des lampadaires

2.2.1 Schéma structurel



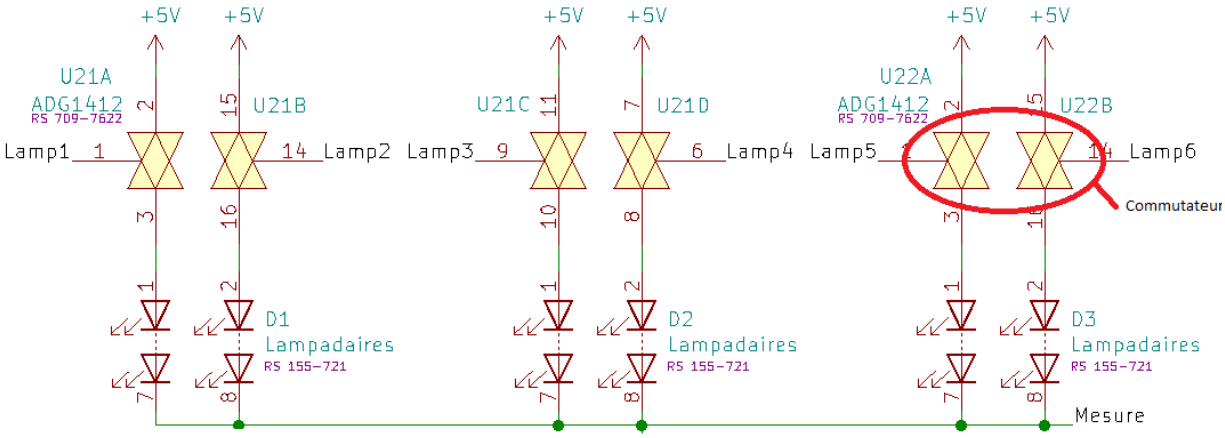
L'objectif est de pouvoir faire état d'un lampadaire qui ne fonctionnerait plus. Pour ceci les lampadaires sont toujours pilotés par un transistor mais a 100% pour que le microcontrôleur puisse faire l'acquisition de la tension aux bornes des lampadaires et selon cette variation de tension il devra prévenir d'une panne.

2.2.2 Problèmes et Solutions

1er Problème : Visualisation du résultat
À la suite de quelques essaies, je m'aperçois qu'il est compliqué de visualiser si ça fonctionne ou d'où vient le problème. Je suis donc passé sur une carte Arduino. Ce qui me permettra de visualiser les trames reçues directement grâce à l'IDE mais aussi de ne plus perdre de temps en programmant l'ATTiny84 par l'Arduino.

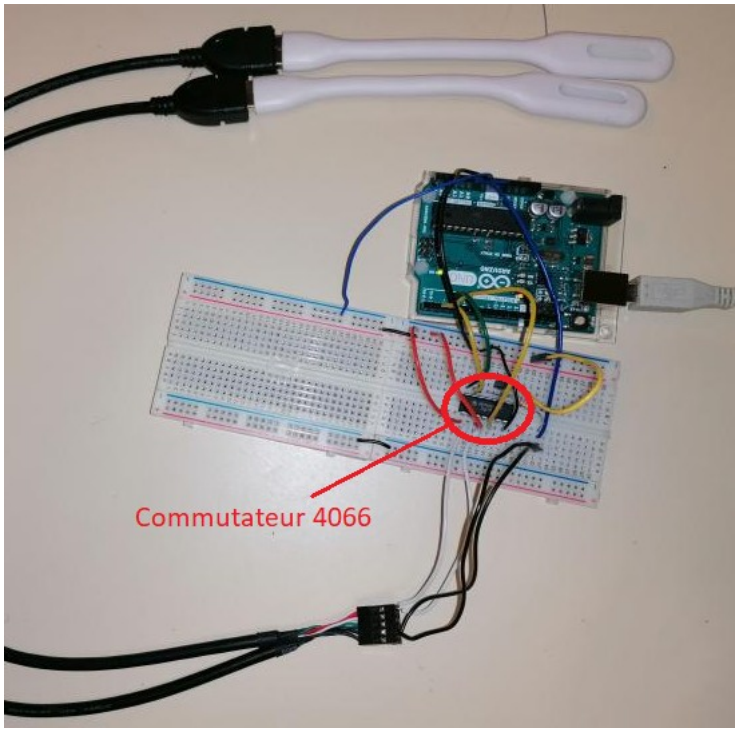
2eme Problème : Résultat trop approximatif
Après être passé sur l'Arduino, je me rends compte que ma carte reçoit des données peu stables et une variation moindre lors de la simulation d'un lampadaire « cassé ». À la suite d'une longue réflexion avec mon professeur on a décidé de tester chaque lampadaire l'un après l'autre grâce à des commutateurs analogique. Cela permettra de bien identifier un lampadaire cassé car le niveau sera de 0V.

2.2.3 Schéma corrigé est résultat du programme



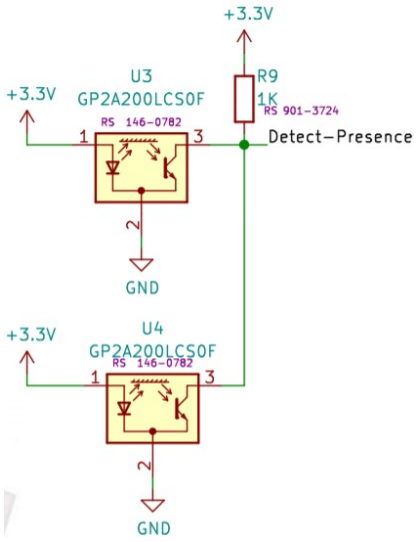

```
16:40:05.233 -> Lampadaire 1 : Fonctionne
16:40:05.737 -> Lampadaire 2 : HS
16:40:06.252 -> Lampadaire 1 : Fonctionne
16:40:06.765 -> Lampadaire 2 : Fonctionne
```

2.2.4 Montage réel



2.3 Validation capteur de présence

2.3.1 Schéma structurel

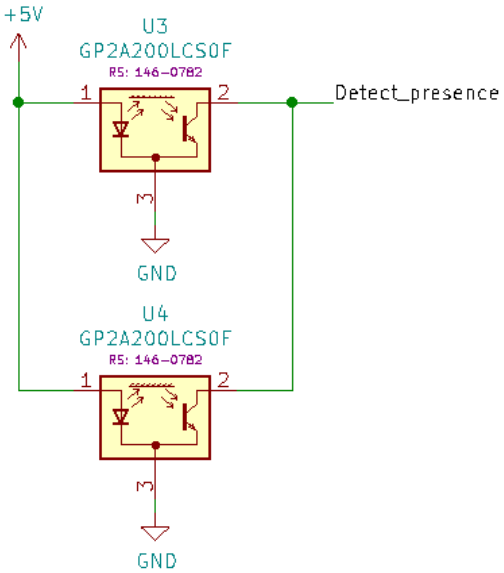


L'objectif est de détecter le passage d'un individu et de le remonter à la Raspberry.

2.3.2 Problème et Solution

1er Problème : Gérer l'information
Il fallait déterminer l'information à envoyer à la Raspberry car quand un individu passe devant le détecteur on ne peut pas savoir le sens de passage. Donc après réflexion on a décidé de renvoyer l'information de chaque passage à la Raspberry qui la remontera au client qui aura la décision d'allumer ou non les lampadaires.

2.3.3 Schéma final

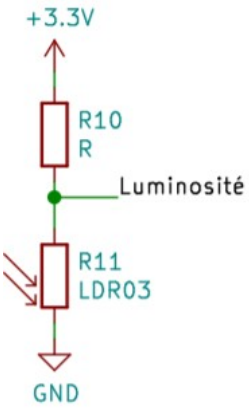


La résistance de pull-up n'est plus nécessaire car elle est disponible dans le microcontrôleur, activable avec la commande suivante :

```
pinMode(PRE, INPUT_PULLUP);
```

2.4 Validation capteur Jour/Nuit

2.4.1 Schéma structurel



L'objectif est de renvoyer l'information du moment de la journée grâce à une photorésistance. Qui fonctionne comme une résistance variable selon l'éclairement.

2.4.2 Problèmes et Solutions

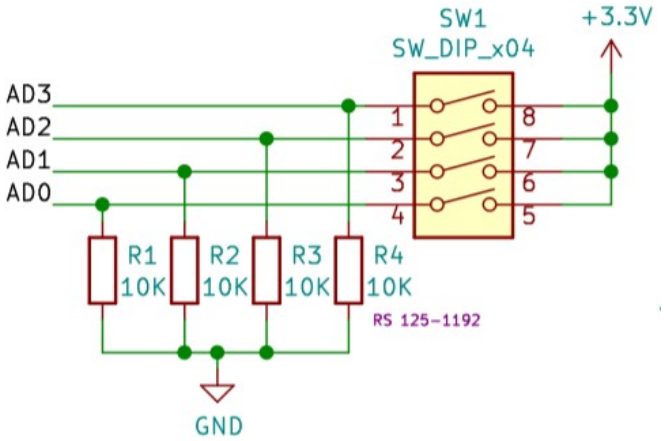
1er Problème : Déterminer la résistance

Pour récupérer l'information on utilise un pont diviseur de tension qui nous sortira une tension variable selon l'éclairement. Pour déterminer la résistance il faut sortir de la documentation la résistance de la photorésistance selon l'éclairement.

Light Resistance	10 lux., 2854°K[3]	20	-	100	KΩ
	100 lux., 2854°K[3]	-	5	-	

Suite à ça on peut réaliser le calcul. Après quelque test la résistance de 47 kΩ fonctionnais bien.

2.5.1 Schéma structurel



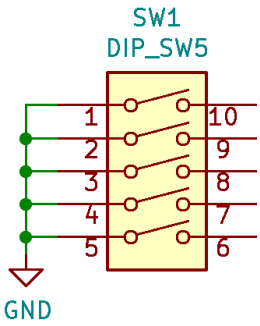
L'objectif est d'affecter une adresse à chaque carte manuellement pour pouvoir les reconnaitre dans les transactions d'information.

2.5.2 Problèmes et Solutions

1er Problème : Le nombre d'adresse disponible
Dans le cahier des charges il est dit qu'il peut y avoir jusqu'à 20 cartes sur la maquette. On a donc dû revoir la taille du switch de 4 à 5 car avec seulement 4 switch il est possible d'attribuer que 16 adresses qu'avec 5 switches il est possible d'adresser 32 cartes.

2.5.3 Schéma et programme

Les résistances ne sont plus nécessaires car remplacé par les pull-up interne.

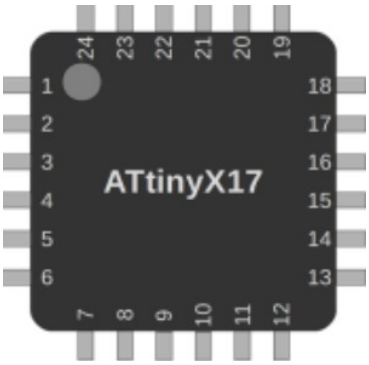



```
pinMode(SW0, INPUT_PULLUP);
pinMode(SW1, INPUT_PULLUP);
pinMode(SW2, INPUT_PULLUP);
pinMode(SW3, INPUT_PULLUP);
pinMode(SW4, INPUT_PULLUP);
//atribution adresse
ADDR = (!digitalRead(SW0) + 2*!digitalRead(SW1) + 4*!digitalRead(SW2) + 8*!digitalRead(SW3) + 16*!digitalRead(SW0)) + 20;
```

La dernière ligne est la ligne de calcul de l'adresse selon les broches connectées.

3. Passage à l'ATTiny1617

3.1 Raison du changement

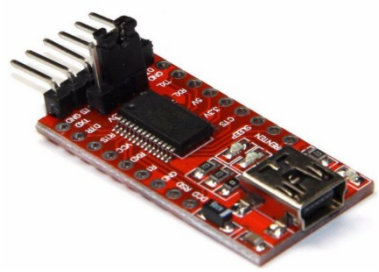


J'ai commencé mon projet avec l'optique d'utiliser un microcontrôleur de type ATtiny84 (à gauche) qui possède 14 broches. Cependant à la suite de l'ajout des commutateurs ainsi que l'agrandissement du switch, ce microcontrôleur est devenu trop petit. Donc je suis passé sur un ATtiny1617 (à droite) qui lui possède 24 broches.

3.2 Accessoires de programmation

L'ATtiny1617 étant en CMS, il est nécessaire de disposer d'une interface pour la programmer via l'IDE Arduino (à l'aide d'un Arduino nano) et effectuer les tests. Il est aussi pratique de l'associer à un convertisseur USB série pour sortir les signaux Rx et Tx pour les visualiser, car l'interface ne le fait pas.

Interface programmation et test



Convertisseur USB

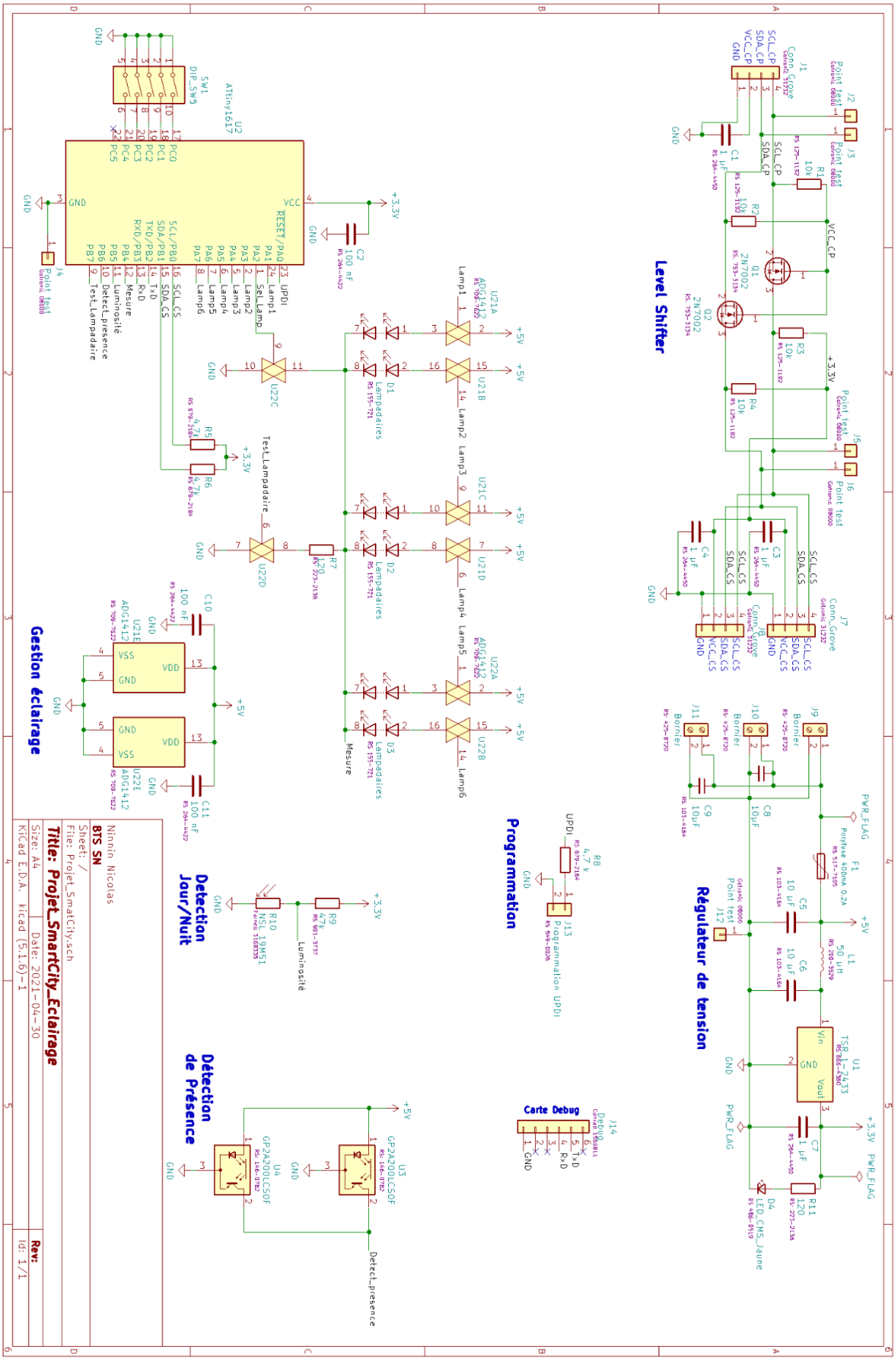
3.3 Attributions des entrées/sorties

~(A)0 (PA4)	: Lampadaire 3
~(A)1 (PA5)	: Lampadaire 4
(A)2 (PA6)	: Lampadaire 5
(A)3 (PA7)	: Lampadaire 6
4 (PB7)	: Test lampadaire
5 (PB6)	: Capteur présence
(A)6 (PB5)	: LDR
(A)7 (PB4)	: Mesure
8 (PB3)	: Rxd
~ 9 (PB2)	: TxD
~(A)10 (PB1)	: SDA
~(A)11 (PB0)	: SCL
~ 12 (PC0)	: ADDR 0
~ 13 (PC1)	: ADDR 1
14 (PC2)	: ADDR 2
15 (PC3)	: ADDR 3
16 (PC4)	: ADDR 4
17 (PC5)	:
(A)18 (PA0)	: Programmation
(A)19 (PA1)	: Lampadaire 1
~(A)20 (PA2)	: Commande lampadaire
(A)21 (PA3)	: Lampadaire 2

~ : Broche à PWM disponible
(A) : Broche analogique

J'ai attribué les broches de l'ATtiny en prenant en compte les fonctionnalités de chaque broche.

4 Schéma Final



Partie Étudiant EC 2 – Équipe 02 : MARONAT Marine

1. Introduction

1.1 Exigences

Étudiant 5 **EC 22**	*Liste des tâches assurées par l'étudiant* **Feux d'intersection et signalisation des places libres/ occupées du parking** • Concevoir une carte de gestion de feux d'intersection. Cette carte devra pouvoir fonctionner selon plusieurs modes prédéterminés qui correspondent à la norme en vigueur, la sélection du mode se fera par réception de consignes véhiculées par le bus I2C, sur lequel la carte fonctionnera en esclave. • Concevoir une carte de signalisation de l'état, libre ou occupé, de chacune des places du parking. • Un schéma structurel sera proposé pour chacune des cartes. • Effectuer tous les tests nécessaires pour valider les structures, et les modifier si nécessaire. • Participer à l'élaboration d'un protocole de communication sur le bus I2C avec l'étudiant IR concerné. • Participer à la conception des parties mécaniques du feu d'intersection et du parking. • Effectuer un travail de documentation afin de proposer une structure permettant de piloter de vrais feux et voyants, et si possible les intégrer sur chacune des cartes. • Effectuer la saisie du schéma et le routage des solutions retenue. Produire les fichiers Gerber afin que la fabrication du PCB soit sous-traitée. • Câbler les cartes et effectuer les essais.	**Installation :** IDE Arduino. **Mise en œuvre :** Tester/valider/modifier une structure utilisant un microcontrôleur Attiny pour gérer des feux tricolores. La carte fonctionnera en tant que circuit esclave sur le bus I2C. Tester/valider/modifier une carte permettant de signaler l'état de chacune des places du parking par une LED verte ou rouge à l'emplacement du véhicule. En principe cette carte ne nécessitera pas de circuit programmable. Participer à une réflexion commune avec tous les étudiants EC du projet SmartCity pour le câblage du faisceau distribuant l'alimentation et le bus I2C sur tout le démonstrateur. Mettre en œuvre ce faisceau. **Réalisation :** Après validation des solutions, concevoir un circuit imprimé pour chacune d'elles. Ces cartes devant être fabriquées industriellement. Mettre en œuvre le faisceau de communication/alimentation. **Documentation :** • Schémas de câblage rapide (Fritzing) pour documenter les phases d'essais. • Documents de fabrication des cartes (KiCAD). Ces documents devront avoir un niveau de qualité permettant une fabrication industrielle du circuit imprimé. • Schéma structurel avec contours IBD. • Liste complète des composants avec leurs sources d'approvisionnement et leurs prix. • Programme de gestion de la carte, accompagné des commentaires et diagrammes nécessaires à sa compréhension. • Fiche de mise en service. • Fiche de dépannage.

1.2 Présentation de ma partie

Pour cette partie deux cartes seront nécessaires afin de réaliser les différentes exigences que vous avez pu rencontrer au-dessus. Une carte qui concernera la partie gestion des places de parking qui ne nécessitera pas de microcontrôleur et une seconde qui elle s'occupera de la signalisation de notre intersection qui contient des feux et des appels piétons, cette carte nécessitera quant à elle la présence d'un microcontrôleur.

2. Diagramme

2.1 Diagramme ULM/SYSML

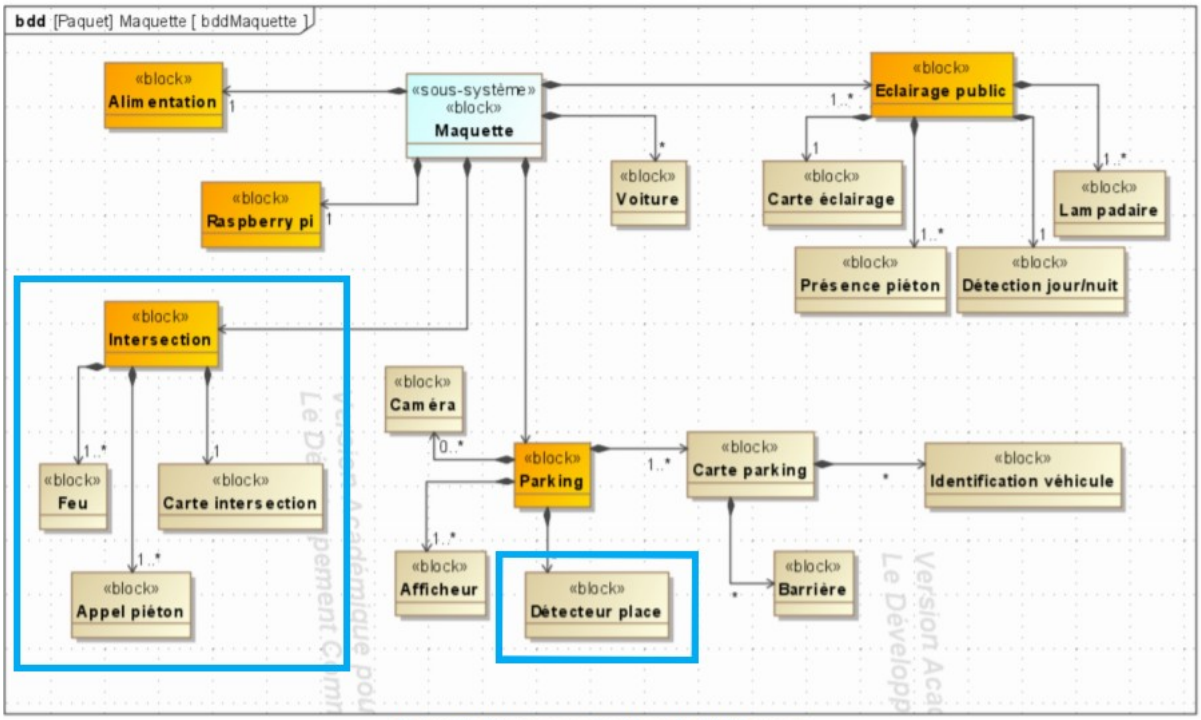


Figure 5 : Diagramme des blocs du SS Maquette

2.1.1 Diagramme d'Architecture et Logiciel

Voici les deux parties dans le diagramme de Bloc qui me concerne.

2.2 Diagramme de Gantt

2.2.1 Prévisionnel

Projet	139 heures	Mer 06/01/2	Jeu 15/04/21	
Specifications générales	**6 heures**	**Mer 06/01/21**	**Jeu 07/01/21**	
Essais et validation des structures (prototypage rapide)	**48 heures**	**Jeu 07/01/21**	**Ven 05/02/21**	**6**
Analyse des schéma structurels	3 heures	Jeu 07/01/21	Ven 08/01/21	6
Test cablage et plaque	20 heures	Ven 08/01/21	Jeu 21/01/21	8
Programation arduino	15 heures	Ven 22/01/21	Ven 29/01/21	9
Réunion Protocole	3 heures	Mer 03/02/21	Mer 03/02/21	10
Ecriture Dossier Revue 1	2 heures	Jeu 04/02/21	Jeu 04/02/21	11
Schéma fritzing	5 heures	Jeu 04/02/21	Ven 05/02/21	12
Prototypage rapide et routage	**62 heures**	**Ven 05/02/21**	**Jeu 01/04/21**	**13**
Fabrication et essais	**23 heures**	**Jeu 01/04/21**	**Jeu 15/04/21**	**20**

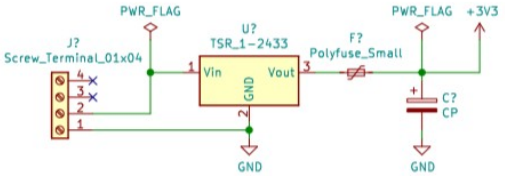
2.2.2 Réel

Projet	138 heures	Mer 06/01/2	Jeu 15/04/21
Specifications générales	6 heures	Mer 06/01/2	Jeu 07/01/21
Essais et validation des structures (prototypage rapide)	51 heures	Jeu 07/01/21	Mer 10/02/21
Analyse des schéma structurels	3 heures	Ven 08/01/2	Ven 08/01/21
Test cablage et plaque	24 heures	Jeu 07/01/21	Jeu 21/01/21
Programation arduino	20 heures	Mer 20/01/2	Ven 29/01/21
Réunion Protocole	3 heures	Mer 03/02/2	Mer 03/02/21
Protocole I2c	4 heures	Jeu 04/02/21	Jeu 04/02/21
Ecriture Dosser 1	5 heures	Ven 05/02/2	Mer 10/02/21
Schéma Structurel	5 heures	Jeu 04/02/21	Ven 05/02/21
Prototypage rapide et routage	62 heures	Mer 10/02/2	Jeu 01/04/21
Fabrication et essais	23 heures	Jeu 01/04/21	Jeu 15/04/21

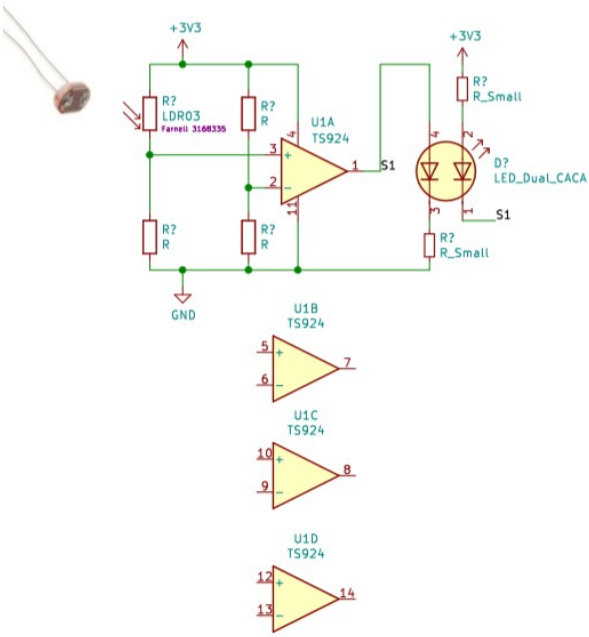
3. Le parking

Pour cette partie l'objectif est de montrer si la place de parking est libre ou occupé.

3.1 Schéma Structurel proposé

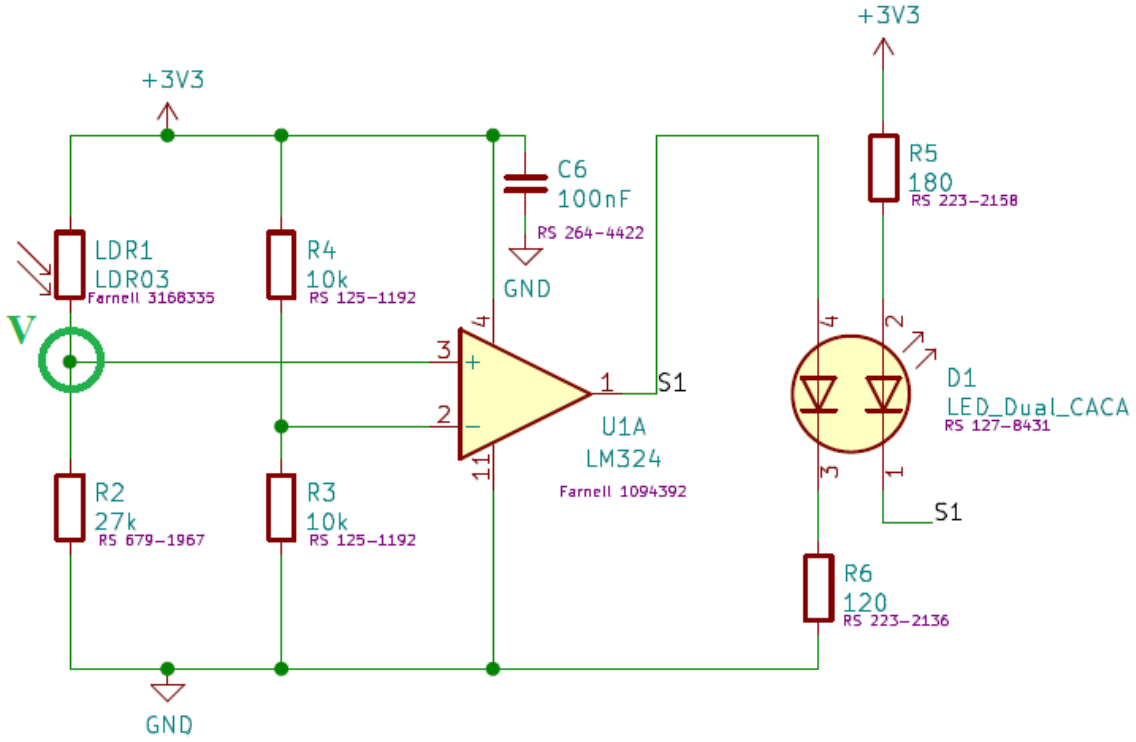


Voici le schéma structurel qui m'a été proposé pour la gestion des places de parking.
Il est composé d'un régulateur, d'un ampli, de 6 Résistances et une DualLed(Rouge et Verte)



3.1.1 Comment fonctionne la LDR ?

Une LDR **Light dependent resistor** ou plus communément appelé « photorésistance » est un composant de type résistif il est donc passif. Sa résistance va varier en fonction de la luminosité qu'elle capte, plus elle sera éclairée plus sa résistance elle diminuera. Voici dans notre cas le fonctionnement de la LDR avec l'ampli.



Si la LDR est dans l'ombre sa résistance sera élevée donc la tension au point V sera faible.Alors l'ampli lui va fournir un niveau bas et va allumer la LED Rouge pour signaler que la place est occupée.
Si la LDR est éclairé alors sa résistance sera faible donc la tension V sera elle élevée.Alors l'ampli lui va fournir un niveau haut et allumer la LED Verte pour signaler que la place est libre.

Pour que cela fonctionne les résistances choisies pour l'entrée inverseuse de l'ampli seront de 10k et celle en série de la LDR de 27k (cette valeur n'est d'ailleurs pas fixe, plusieurs résistances auraient pu être choisies dans notre cas).

L'objectif étant que celle-ci nous permet de basculer en dessous ou dessus de Vdd/2 de la tension d'alimentation soit ici 1,65 V.
Si V < 1,65 V alors nous aurons un niveau bas.Si V>1 ,65 V alors nous aurons un niveau haut.

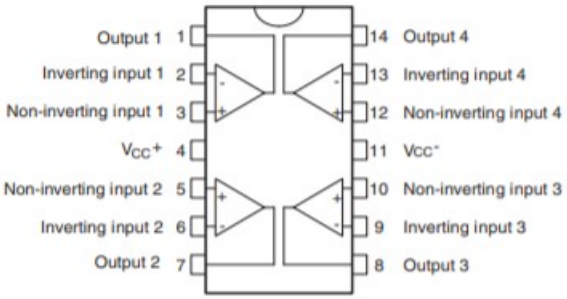
3.1.2 Quel amplificateur opérationnel avons-nous choisis et pourquoi ?

Nous utilisons l'ampli suivant :
- Le TS924

Ce choix est dû à divers avantages qu'il présente.
Il s'agit d'un ampli dit « rail to rail », ce qui est élément majeur pour notre choix. Cela signifie que la tension d'alimentation de l'ampli sera similairement la même en sortie (Soit Vsat +, soit Vsat-)

V_{OH}	High level output voltage R_L= 10 kΩ $T_{min} \le T_{amb} \le T_{max}$ R_L = 600 Ω $T_{min} \le T_{amb} \le T_{max}$ R_L = 32 Ω	2.90 2.87	 2.63		V

Il fonctionne également sous mono-tension et il est en réalité composé de 4 amplis, ce qui présente un réel avantage, car il permettra donc de gérer 4 places de parking en même temps. Dans notre projet, on souhaite gérer 8 places de parking, il nous faudra donc uniquement deux amplis.



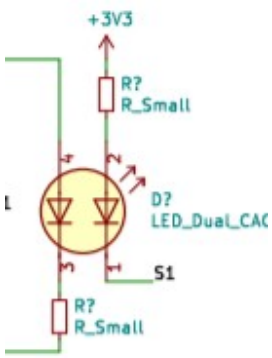
<u>L'ampli permet-il de fournir assez de courant pour 4 sorties dans notre cas ?</u>
Afin de déterminer si oui ou non, notre ampli permet de délivrer un courant suffisant pour éclairer nos LED, j'ai tous simplement branché toutes mes LED et j'ai effectué une mesure à l'aide de l'ampèremètre, en série entre l'alimentation en 3,3 de l'Arduino et le moins de l'alimentation sur le breadboard.
J'ai pu relever environ 34 mA pour 4 LED en fonctionnement.

Or l'amplificateur, d'après sa documentation peut fournir un courant allant jusqu'à 80 mA

I_o	Output short-circuit current	50	80		mA
I_{CC}	Supply current /operator - no load, $V_{out} = V_{CC+}/2$ $T_{min} \le T_{amb} \le T_{max}$		1	1.5 1.6	

Il est donc possible de commander les quatre sorties.

3.1.3 Choisir les résistances de protections pour nos LED



Voici les deux calculs qui m'ont permis de trouver les résistances nécessaires.
On considère pour nos essais que la LED verte fonctionnera sous une tension de 2V et la rouge sous 1,6V.

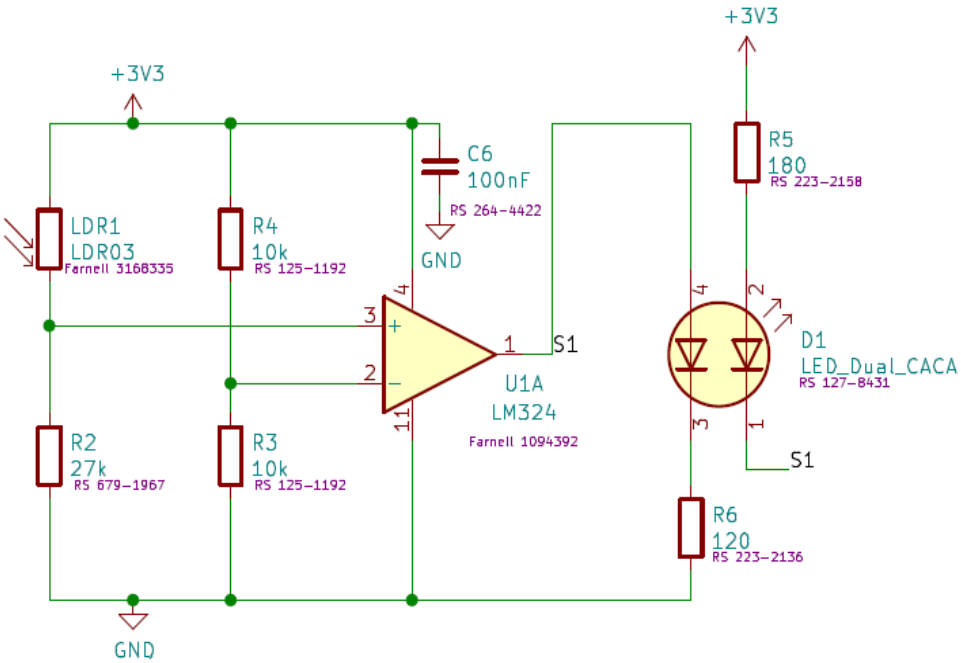
$$Rred = (Ualim - Uled) / Iled = (3{,}3 - 1{,}6) / 0{,}010 = 170\,\Omega$$

Je vais donc prendre 180 Ω

$$Rgreen = (Ualim - Uled) / Iled = (3{,}3 - 2) / 0{,}010 = 130\,\Omega$$

Je vais donc prendre 120 Ω

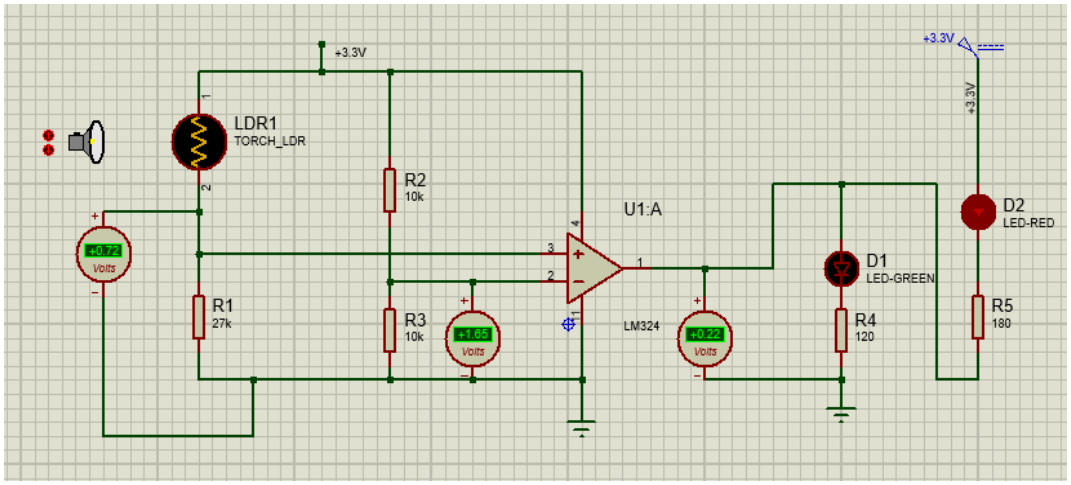
Voici la structure que l'on obtient mis bout à bout.

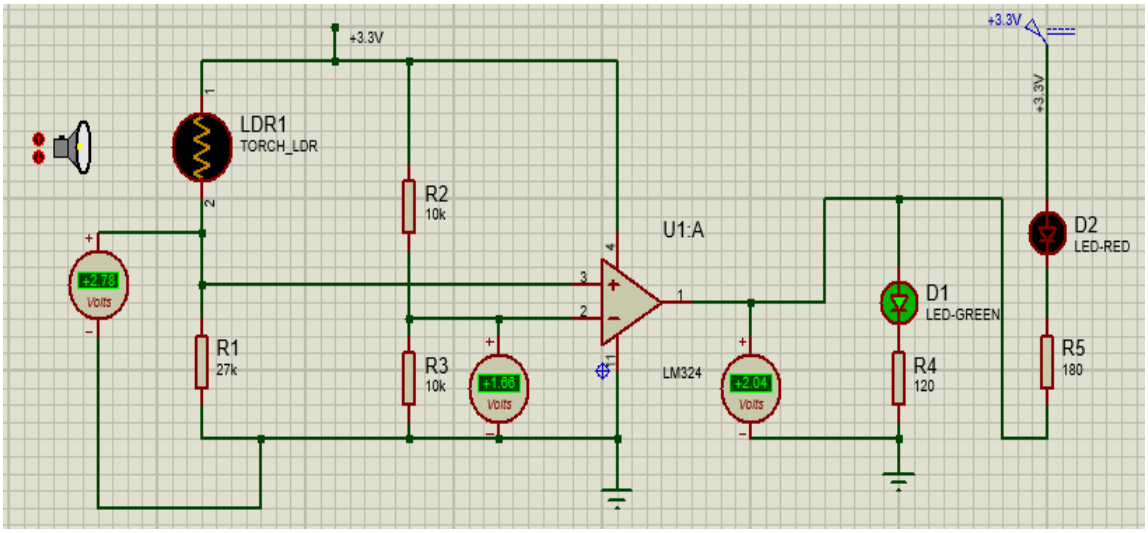


3.2 Schéma et essais

3.2.1 Simulation

À l'aide de Proteus, j'ai effectué une simulation de notre structure afin de déterminer si mes calculs et mes conclusions semblent fonctionné.
Voici un aperçu lorsqu'un véhicule se trouve sur la place.





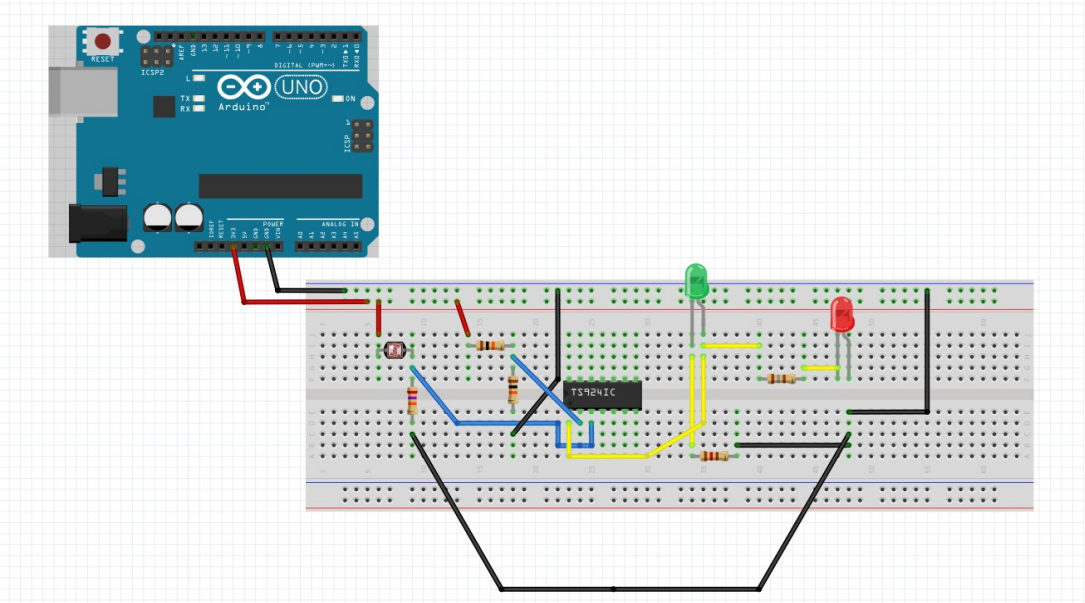
Dans le cas suivant, la place est libre.
Pour paramétrer la LDR je me suis appuyé sur la documentation

Light Resistance	10 lux., 2854°K[3]	20	-	100	KΩ
	100 lux., 2854°K[3]	-	5	-	
Dark Resistance	10 sec after removal of test light.	20	-	-	MΩ

Lorsqu'un véhicule est présent sur la place de parking, j'ai configuré la LDR avec une résistance de 20 MΩ, lorsqu'elle est libre 5kΩ.
On remarque que la résistance choisie de 27kΩ permet bien comme prévue de faire varier le seuil au-dessus de Vdd/2 ou alors en dessous de Vdd/2.
À savoir qu'ici nous ne sommes pas dans la meilleure des dispositions puisque je n'ai pas utilisé l'amplificateur à ma disposition n'étant pas sur Protéus, ici on remarque que la tension en sortie de l'ampli perd plus d'1,3V pour un étant Haut, ce qui ne sera pas le cas de notre ampli puisqu'il s'agit d'un « rail to rail ».

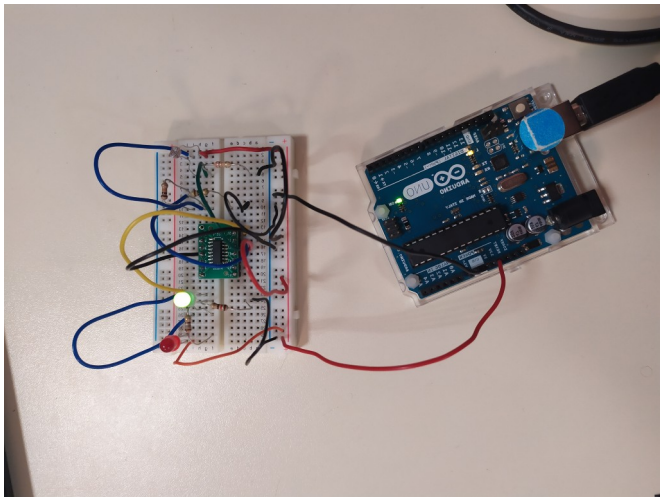
3.2.2 Schéma Fritzing

Après la phase de prototypage, voici le câblage rapide que j'ai effectué.

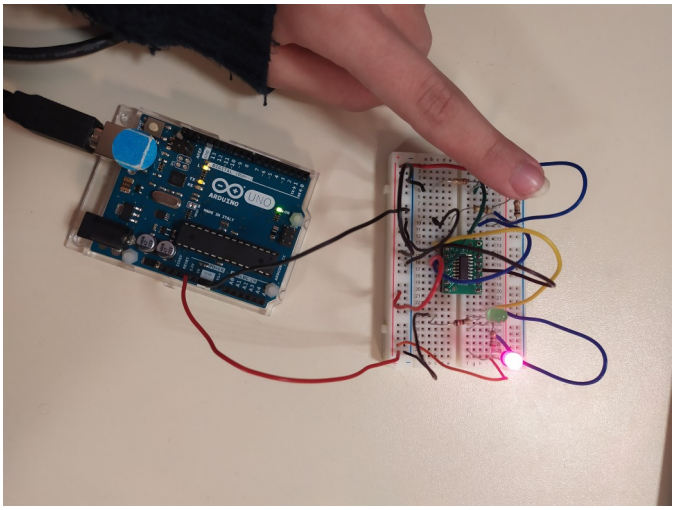


3.2.3 Essais sur breadboard

Sans obstruction de la lumière :

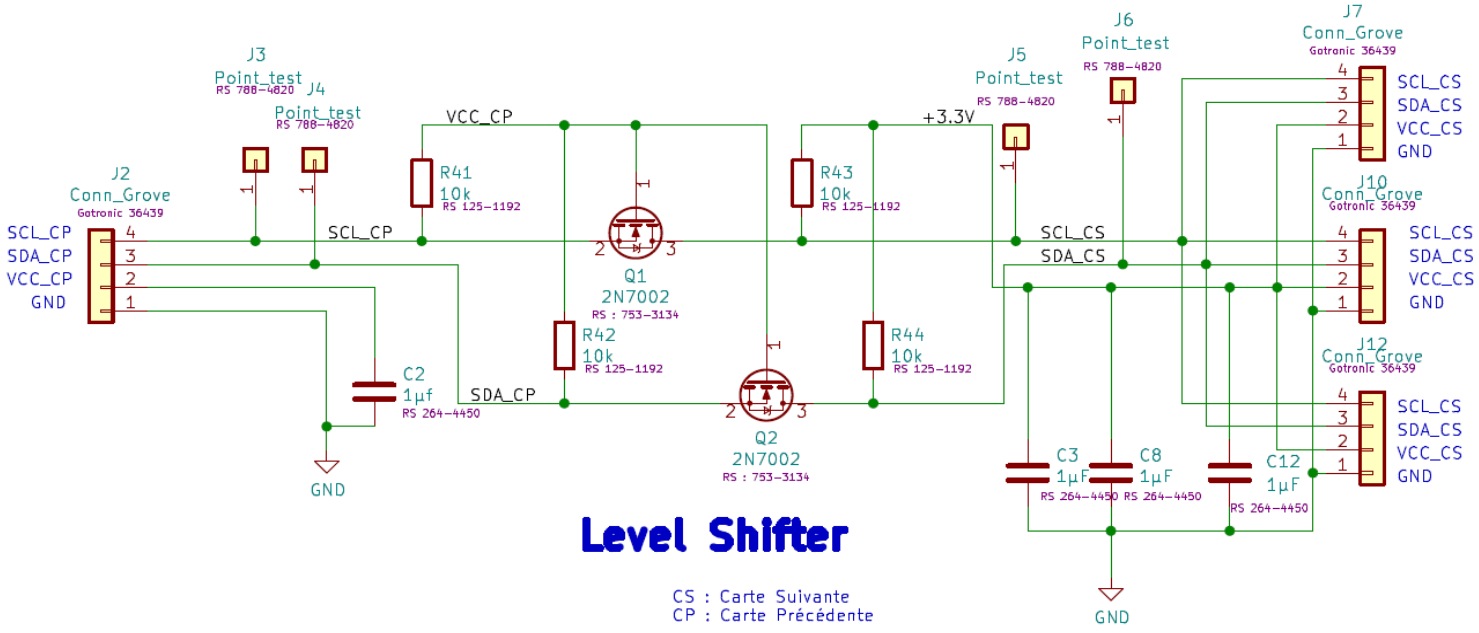


Avec obstruction sur la LDR

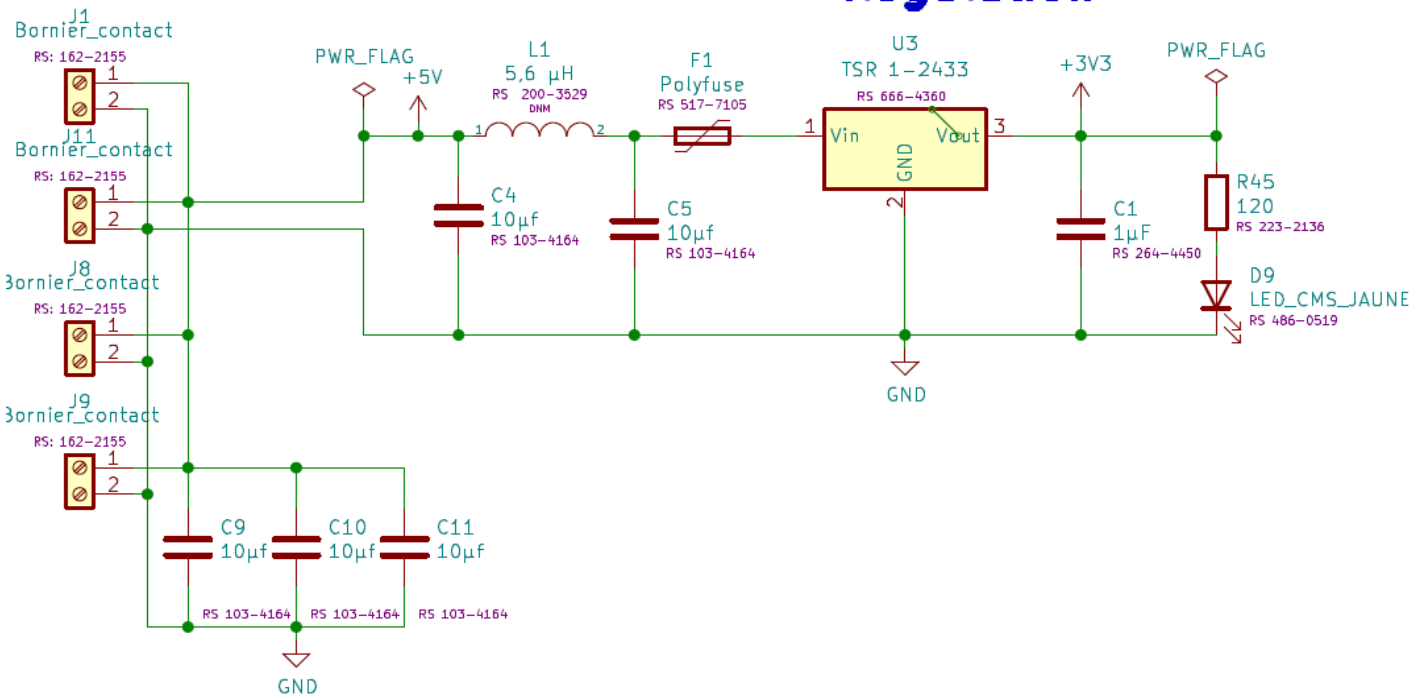


3.2.4 Intégration du level shifter et du régulateur.

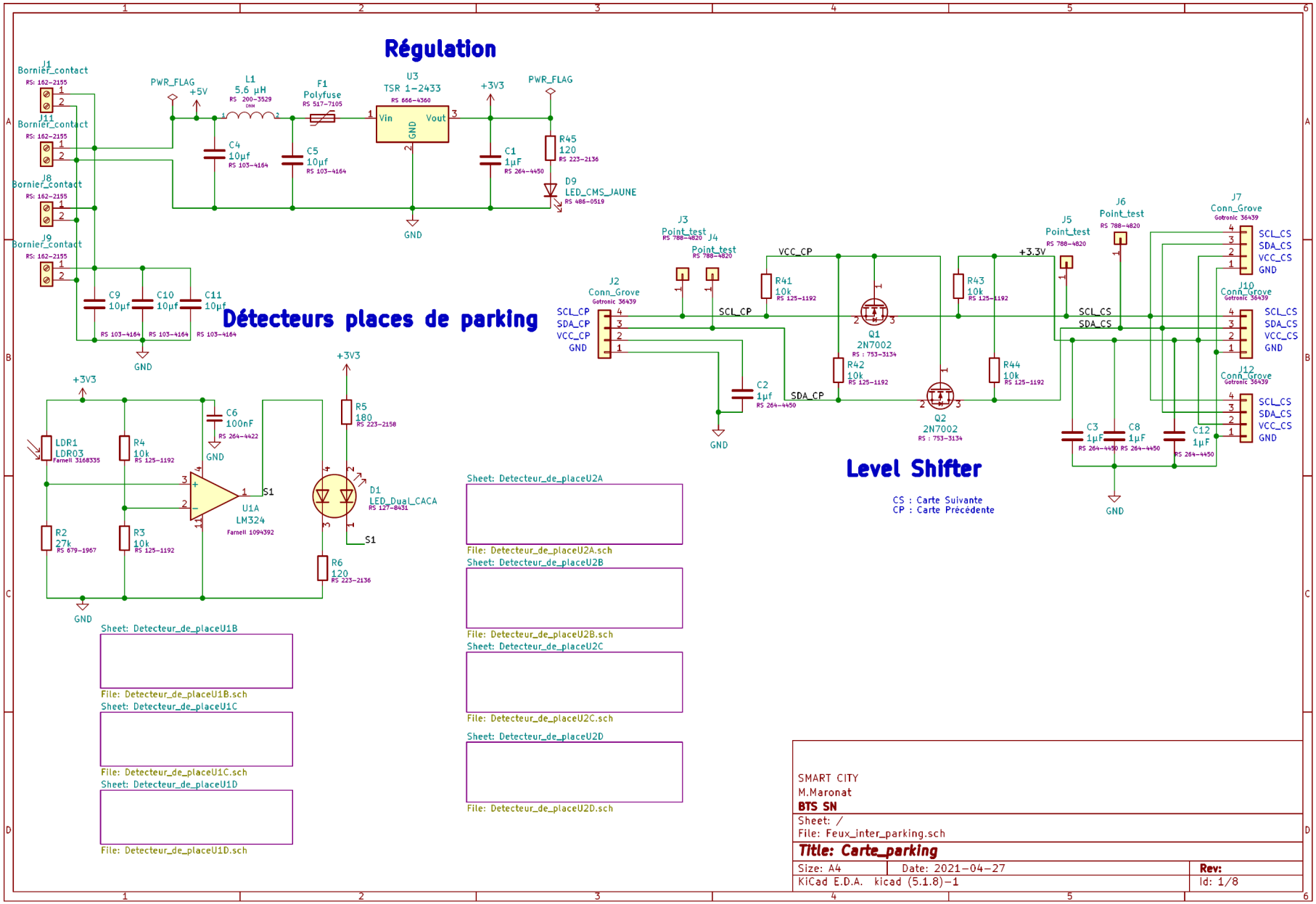
Afin de pouvoir relier toutes les cartes du projet entre elles, nous avons également mis un level shifter sur chacune de nos cartes. Bien que la principale fonctionnalité soit l'échange des données en I2C, pour la carte du parking, la communication elle n'est pas nécessaire puisque l'objectif de cette carte est de détecter si la place et libre ou non et en fonction allumer la LED correspondante. Il permettra tout de même de régénérer le signal.Comme ma carte est grande j'ai multiplié le nombre de connecteurs groove et de borniers pour qu'il puisse y en avoir de chaque cotés. Le régulateur lui permettra de basculer la tension de 5V à 3,3V, il est indispensable, car l'une des cartes du projet sera justement alimentée en 5V.



Régulation

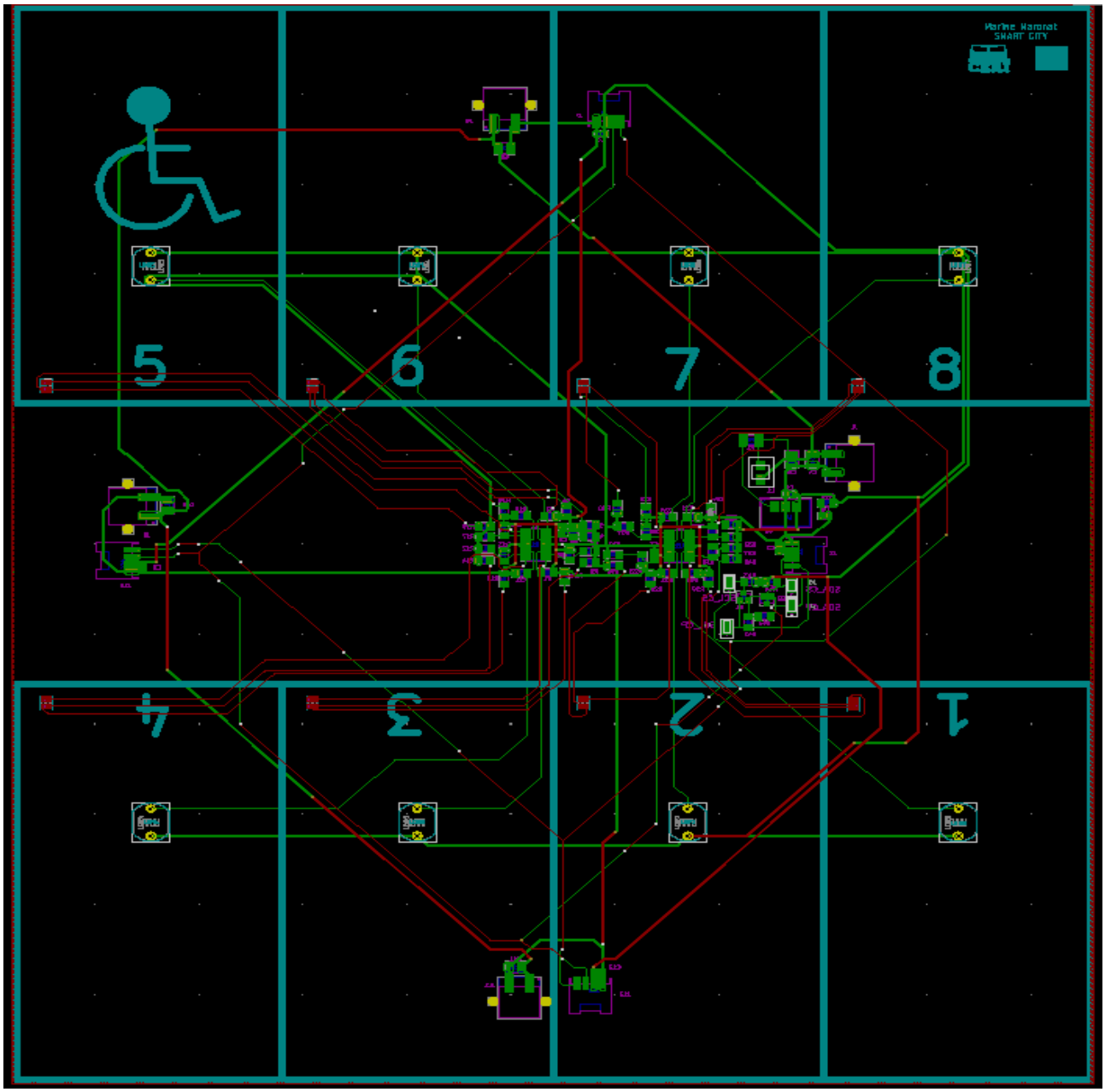

J1
Bornier_contact
RS: 162-2155
J11
Bornier_contact
RS: 162-2155
J8
Bornier_contact
RS: 162-2155
J9
Bornier_contact
RS: 162-2155
PWR_FLAG
+5V
L1
5,6 µH
RS 200-3529
DNM
F1
Polyfuse
RS 517-7105
U3
TSR 1-2433
RS 666-4360
Vin
GND
Vout
+3V3
PWR_FLAG
C4
10µf
RS 103-4164
C5
10µf
RS 103-4164
C1
1µF
RS 264-4450
R45
120
RS 223-2136
D9
LED_CMS_JAUNE
RS 486-0519
GND
C9
10µf
C10
10µf
C11
10µf
RS 103-4164
RS 103-4164
RS 103-4164
GND

Voici le schéma structurel finale de la carte parking.
Pour faire ce schéma, j'ai également appris à faire des schémas dît hiérarchiques (les rectangles que l'on peut voir), dans chacun de ses sous-schémas se trouve mes différentes structures Ampli, LDR et LED. Ils rendent le schéma plus lisible et moins encombré.



3.3 Routage de la carte parking

Maintenant voici le routage effectué pour cette carte



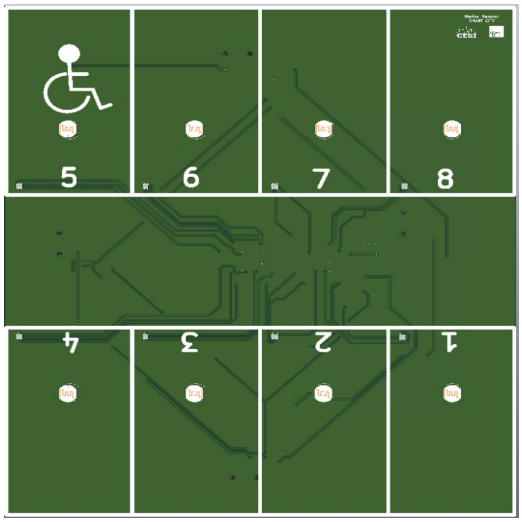
La carte de l'intersection mesure 300x260mm.

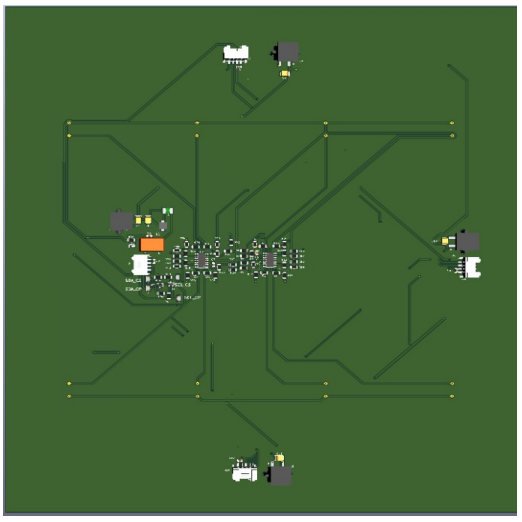
Il faut savoir que pour mes deux cartes aussi bien celle-ci ou celle dont je vais vous parler par la suite, chacune doit être à taille réelle, du parking et de l'intersection, c'est pour cela qu'elles sont toutes deux assez grande. En ce qui concerne la disposition, comme expliqué plus tôt, de chaque côté de la carte, on retrouve un bornier et un connecteur grove pour pouvoir se brancher à l'alimentation, mais aussi au level shif-

ter de sorte à se raccorder dans la direction que l'on souhaite, en prévision des autres cartes du projet.

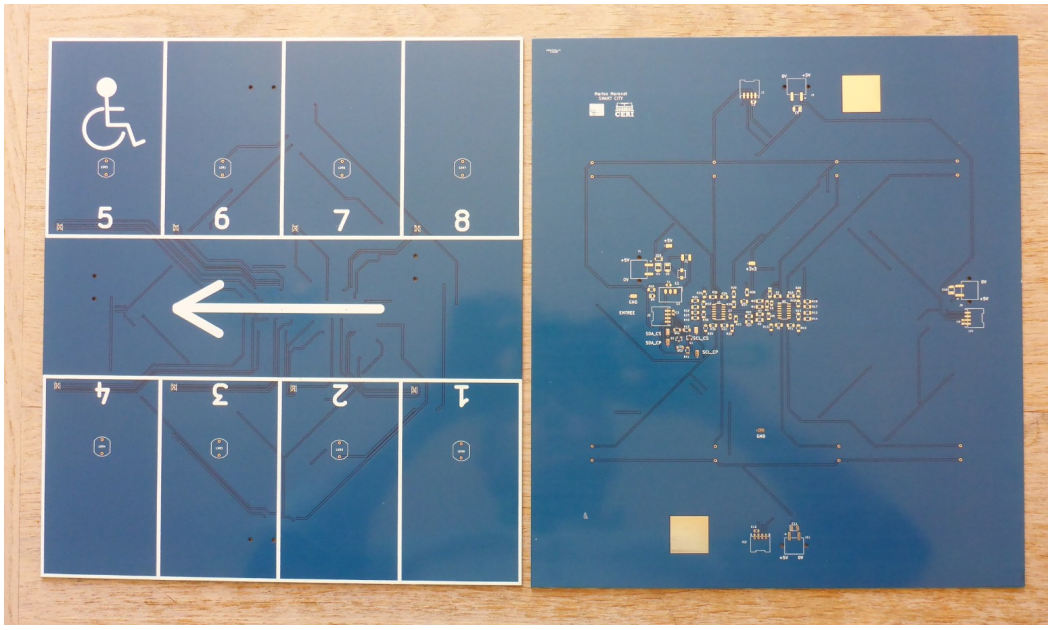
Du côté face, j'ai uniquement mis les éléments qui devront être apparent sur l'intersection : c'est-à-dire uniquement les LDR qui détectent la luminosité ainsi que les LED qui vont indiquer aux usagers si la place est libre ou non. Ainsi, de l'autre côté, se trouvent tous les autres composants.
J'ai tout d'abord commencé à router au centre les composants tels que les amplis et résistances le plus proche possible entre eux, pour pouvoir faire fabriquer un stencil de petite taille, qui reviendrai moins chère. Stencil qui me permettra de souder plus facilement par la suite. J'ai ensuite rapproché la structure d'entrée d'alimentation avec le connecteur grove afin de les inclure à ma structure centrale toujours dans les mêmes conditions, le plus proche possible.
Une fois, mes composants bien disposés, j'ai pu commencer à router toujours en commençant par la structure centrale.
Pour passer de dessus a dessous, j'ai utilisé des vias qui permettent une liaison entre les couches, j'ai donc pu raccorder comme je lui souhaitais mes différents chevelus.

Voici un aperçut 3D de ma carte.



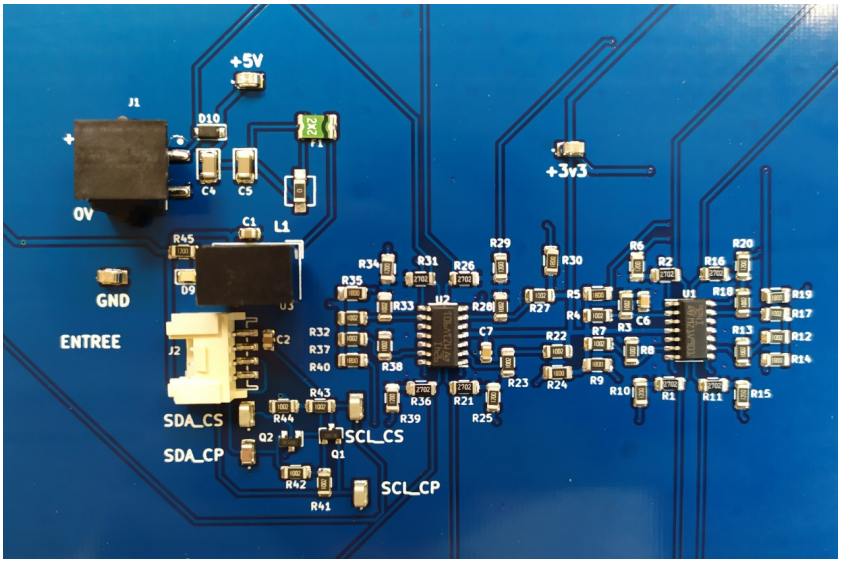


Voici un aperçu de la carte une fois reçu :



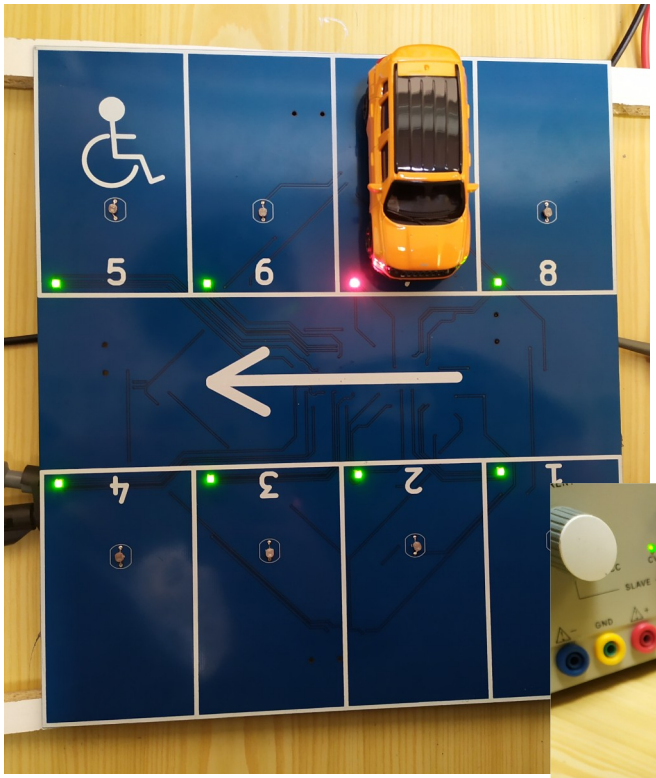
Procédé de routage :

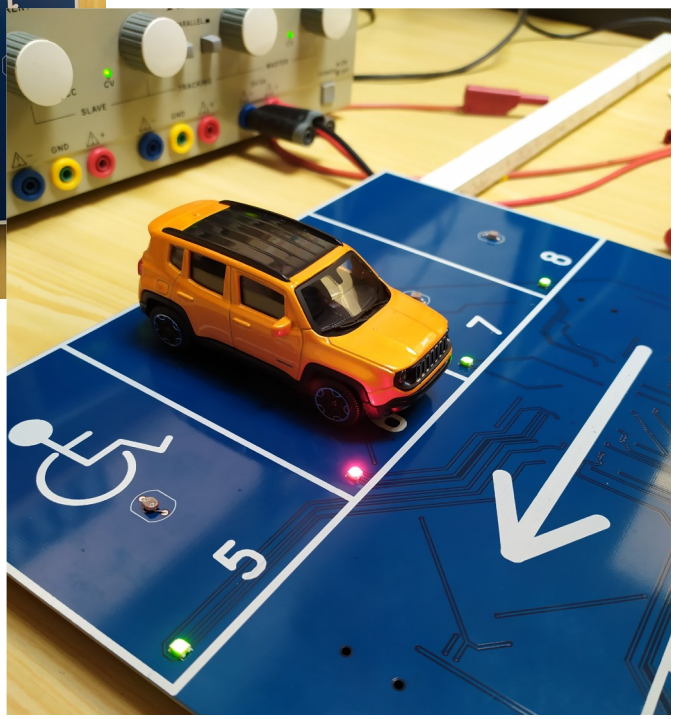
Comme expliqué précédemment ma carte est très grande ce qui implique qu'elle ne rentre pas dans le four. J'ai donc dû procéder différemment.
Pour commencer, j'ai nettoyé ma carte et son stencil avec de l'acide isopropylique.
Ensuite, j'ai associé ma carte à son stencil correspondant.
J'ai positionné la pâte à braser sur le PCB.
Afin de chauffer la pâte à braser et ainsi souder mes composants.
J'ai utilisé un pistolet à air chaud, c'était la toute première fois que
j'utilisai ce type d'appareil.
Voici un aperçu après l'utilisation du pistolet :



A savoir que j'ai dû chauffer à des températures différentes puisque j'ai utilisé deux pâtes différente. La première m'a servi pour tous les composants présents avec le stencil. En revanche, j'ai utilisé une autre pâte plus « liquide » pour les composants extérieur au stencil. Pour les composants de la première pâte, une température de 350 degrés était nécessaire. Pour la seconde plutôt 250.
Il ne me restera maintenant plus qu'à souder les élements côtés top que sont les duals Leds et les LDR.

J'ai pu une fois tous les composants soudés, tester ma carte.
Il m'a simplement suffis de l'alimenter sur les points tests de 5V et de GND avec une alimentation externe.
Aucun problème n'est à signaler voici le rendu du parking avec et sans véhicule.





Liste de matériels :

	Liste de Matériels		Lycée Alphonse Benoit	
Projet SMART CITY Maronat Marine	Carte parking		Isle sur la Sorgue	
Références des composants	Valeur	Code Commande	Nombre dans le paquet	Prix TTC
C1, C2, C3, C8, C12	1 µF	RS 264-4450	25	1,05
C4, C5, C9, C10, C11	10µf	RS 103-4164	20	4,12
C6, C7	100nF	RS 264-4422	25	3,825
D1, D2, D3, D4, D5, D6, D7, D8	LED_Dual_CACA	RS 127-8431	20	9,95
D9	LED_CMS_JAUNE	RS 486-0519	25	6,225
F1	Polyfuse	RS 517-7105	10	3,45
J1, J8, J9, J11	Bornier_contact	RS: 162-2155	vendu à l'unité (2,49)	9,96
J2, J7, J10, J12	Conn_Grove	Gotronic 36439	vendu à l'unité (0,20)	0,8
J3, J4, J5, J6	Point_test	RS 788-4820	10	1,84
L1	5,6 Ï¼H	RS 200-3529	1	6,81
LDR1, LDR2, LDR3, LDR4, LDR5, LDR6, LDR7, LDR8	LDR03	Farnell 3168335	vendu à l'unité (1,07)	8,56
Q1, Q2	2N7002	RS : 753-3134	100	5,4
R1, R2, R11, R16, R21, R26, R31, R36	27k	RS 679-1967	50	1,1
R3, R4, R7, R8, R12, R13, R17, R18, R22, R23, R27, R28, R32, R33, R37, R38, R41, R42, R43, R44	10k	RS 125-1192	100	2,4
R5, R9, R14, R19, R24, R30, R35, R40	180	RS 223-2158	50	0,65
R6, R10, R15, R20, R25, R29, R34, R39, R45	120	RS 223-2136	50	0,65
U1, U2	LM324	Farnell 1094392	vendu à l'unité (2,19)	4,38
U3	TSR 1-2433	RS 666-4360	vendu à l'unité (5,6)	5,6
			Total Fournisseur RS	63,03
			Total Gotronic	0,8
			Total Farnell	12,94
			Coût total	76,77

Afin de créer cette liste de matériel, je suis passé par l'outil Bill of materials de Kicad, qui va me générer automatiquement une liste.En revanche, j'ai tout de même fait pas mal de modification puisque de nombreux éléments n'étaient pas utiles. J'ai ensuite ajouté les prix des composants en allant à l'aide des codes commandes sur les sites fabricants.

4. Intersection

L'objectif de cette partie est de gérer les quatre feux tricolores qui gère la signalisation de l'intersection ainsi que les appels piétons qui permettent l'interruption sur la voie où il souhaite traverser.

4.1 Approche de l'intersection.

4.1.1 Respect d'une Norme pour la durée des feux.

http://www.equipementsdelaroute.equipement.gouv.fr/spip.php?page=sommaire

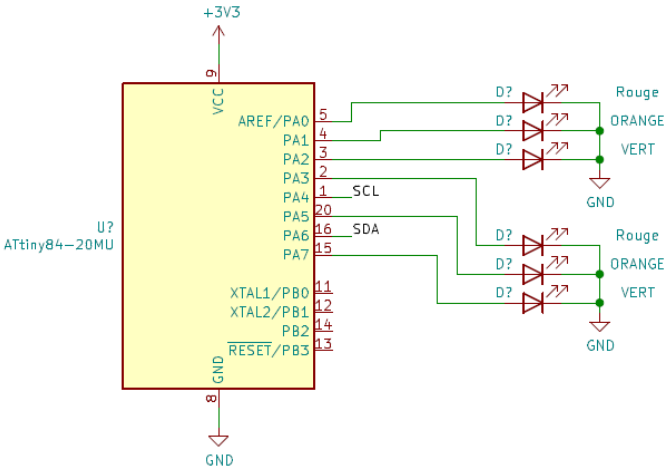
Signaux tricolores (R11, R13 et R14)

- le déroulement des couleurs est le suivant : vert - jaune fixe - rouge - vert ; exceptionnellement pour les signaux R11j, R13cj et R13bj il peut être : jaune clignotant sur le feu du bas - jaune fixe - rouge - jaune clignotant sur le feu du bas ;
- la durée minimale du vert (ou du jaune clignotant) est de six secondes ;
- la période jaune fixe dure obligatoirement soit trois secondes soit cinq secondes ;
- la durée de trois secondes est la règle générale en agglomération ;
- la durée de cinq secondes est obligatoire aux intersections hors agglomération, ainsi que pour tout signal tricolore fonctionnant au jaune clignotant sur le feu du bas (R11j, R13cj et R13bj).

4.1.2 Structure et les différents modes possibles.

Nous avons décidé d'inclure 4 feux tricolores qui s'occuperont de la signalisation de la voie, ainsi que 8 boutons piétons qui pourront à tout moment intervenir sur le mode classique des feux tricolores.
L'intersection va pouvoir fonctionner de différentes manières. Elle possédera un mode classique, celui que nous avons l'habitude de voir, c'est-à-dire vert-orange-rouge. Mais également un mode Orange Clignotant qui laisse donc place à une priorité à droite et un mode manuel, où le client par l'intermédiaire du raspberry pourra demander d'intervenir sur la voie de son choix afin d'y mettre la couleur souhaitée.
Nous parlerons plus en détail de cet échange avec le client dans une prochaine partie.

4.2 Schéma Structurel



4.2.1 Prise en main du schéma structurel.

Après la première mise en main de ce schéma, j'ai pu relever certaines choses à modifier.
Nous cherchons à piloter une intersection, qui est constituée de deux voies. Chaque voie possède deux feux tricolores qui fonctionnent en même temps et sont synchro l'un par rapport à l'autre. Il faut donc deux LEDS pour chaque broche.
Il faudra également changer de microcontrôleur, car sur celui-ci, le nombre de broches est assez faible, sachant que l'on compte inclure plusieurs appels piétons qui demanderont des boutons-poussoirs, il nous faudra plus de broches. C'est pour cela que l'on va prendre le microcontrôleur ATTINY3217 qui possède jusqu'à 24 broches
Il se trouve d'ailleurs qu'il est presque tout aussi performant que celui présent sur la carte Arduino. Il prend peu de place et il est peu cher.

4.2.2 Le microcontrôleur permet t'il de fournir le courant nécessaire ?

I_{total}	Maximum combined I/O sink current per pin group(1)		-	-	100	mA
	Maximum combined I/O source current per pin group(1)		-	-	100	

Ici, notre microcontrôleur peut supporter au maximum 100 mA
Nous établissons entre 10 et 15 mA le courant pour une LED, sachant qu'il en faudra 12, piloter deux par deux , cela nous donne au minimum 120 mA, ce qui dépasse ce que le microcontrôleur peut nous fournir, on risquerait alors de le détériorer.
Pour palier à ce problème, nous allons installé sur chaque broche où se trouveront les LED, un transistor MOS, qui lui permettra également de ne pas tirer de tension et de ne pas chauffer. Ils sont d'ailleurs peu cher et suffisent donc en terme d'intensité.
Le **2N7002**

4.2.3 Choix des boutons poussoirs.

Voici les boutons poussoirs que nous avons choisi pour les boutons piétons de l'intersection.
Lors des premiers essais nous avions choisis ces boutons-ci :

Mais au final, la proportion vis-à-vis de la taille de l'intersection et des feux n'était pas idéal, c'est pour cela que nous avons finis par choisir ceux là.

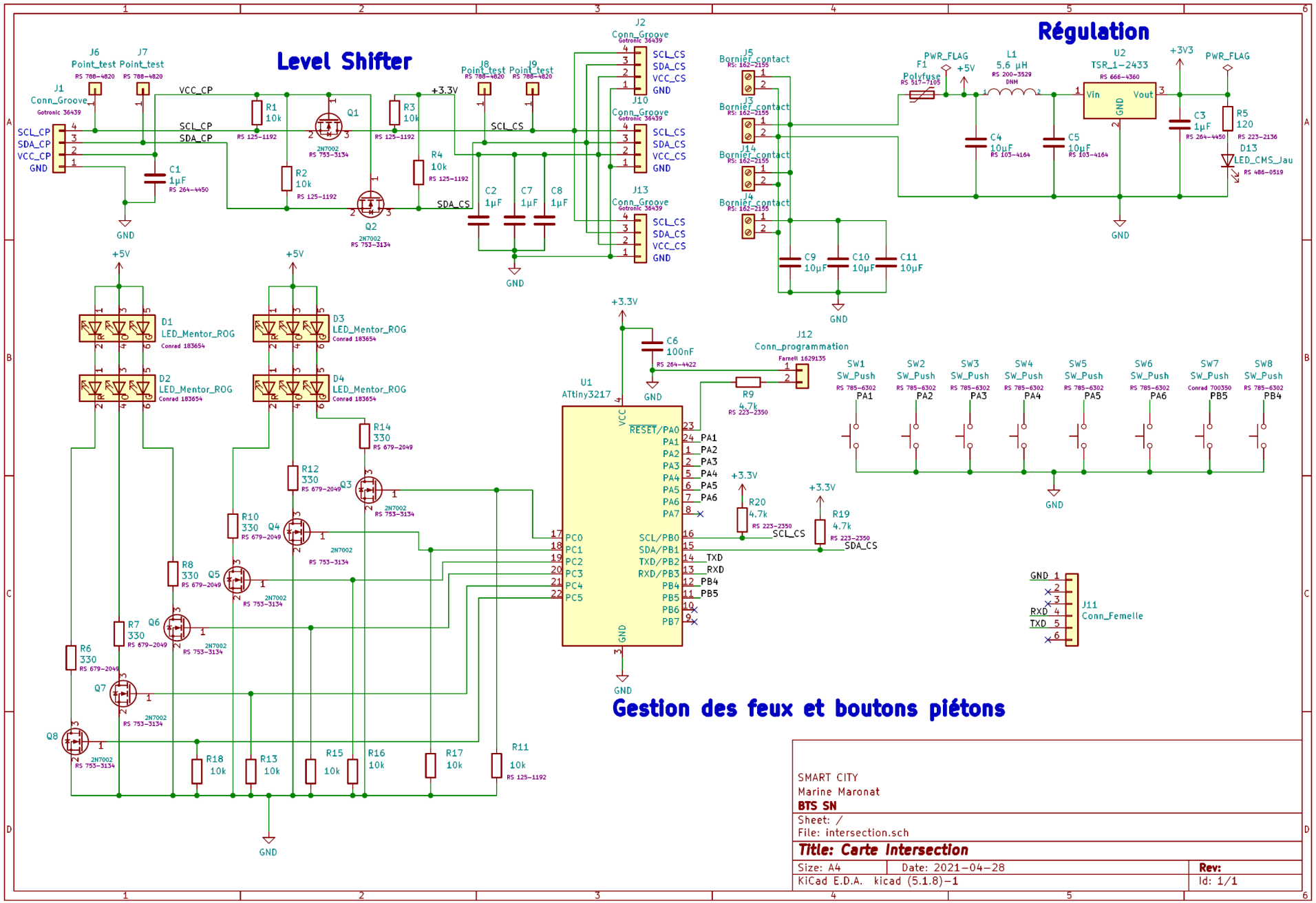
Il y a au total 8 boutons de part et d'autre de l'intersection. Je les ai associés au pin de l'attiny 3217. En sachant que plusieurs broches ne sont pas utilisables comme les broches SDA, SCL, TX, RX etc. Ces broches sont donc inutilisables pour nos boutons.

4.2.4 Les Feux tricolores

Malgré que dans nos essais j'utilise des LEDS distinctes nous avons pu trouver un modèle pour notre intersection, qui est plus pratique et surtout plus esthétique car il ressemble de près à un feu tricolore.

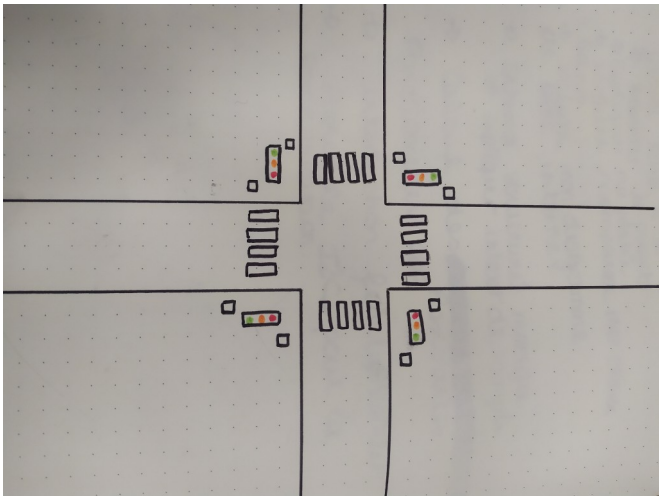
4.2.5 Schéma structurel après modification.

Voici le schéma structurel final de la carte intersection



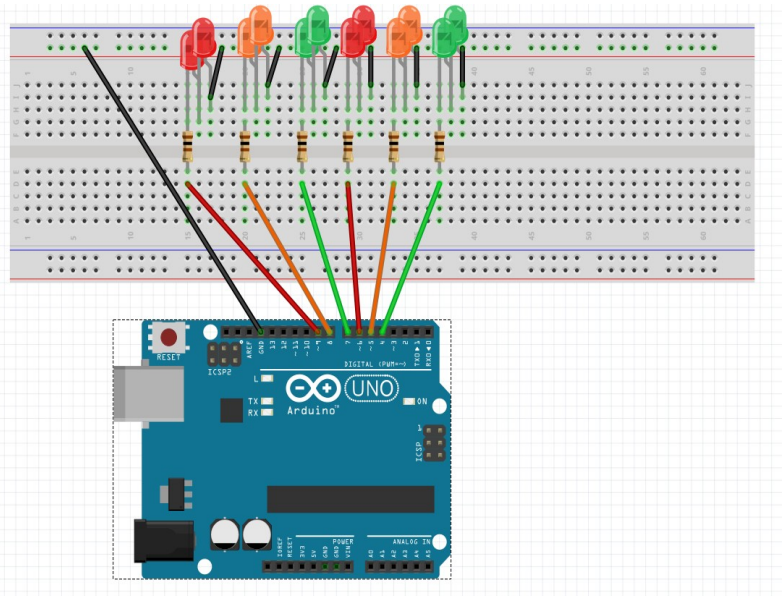
4.3 Schémas et essais

4.3.1 Schématisation rapide de l'intersection.

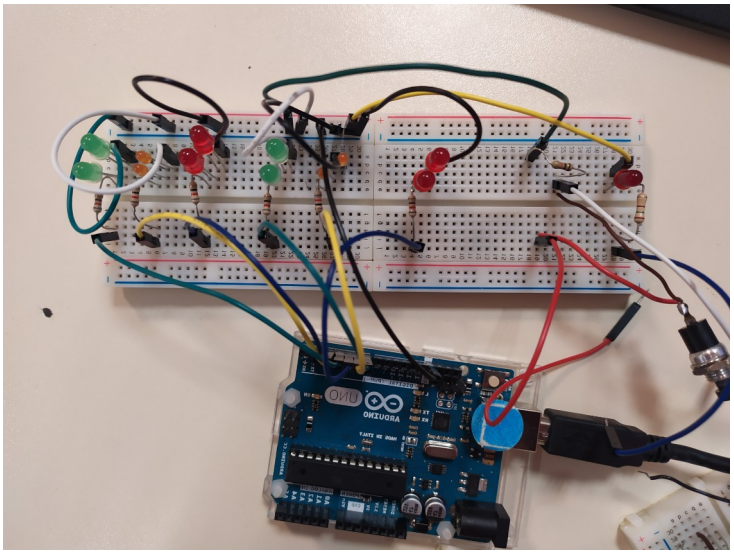


4.3.2 Schéma Fritzing

Voici le schéma Fritzing, il correspond au tout premier essai réalisé pour simplement apprendre à piloter les LED.



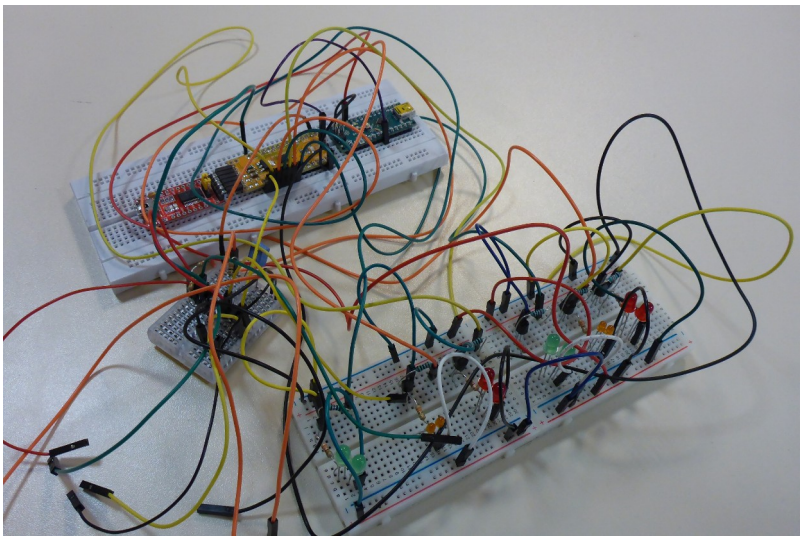
4.3.3 Essais sur breaboard avec Arduino



Programme pour le feu automatique
Dans ce cas-là, j'ai défini des durées, la norme n'est pas encore appliqué

```
void feu_auto() {
  digitalWrite(feu_vert_1, HIGH);
  digitalWrite(feu_rouge_2, HIGH);
  delay(3000);
  digitalWrite(feu_vert_1, LOW);
  digitalWrite(feu_orange_1, HIGH);
  delay(1000);
  digitalWrite(feu_orange_1, LOW);
  digitalWrite(feu_rouge_1, HIGH);
  delay(1000);
  digitalWrite(feu_rouge_2, LOW);
  digitalWrite(feu_vert_2, HIGH);
  delay(3000);
  digitalWrite(feu_vert_2, LOW);
  digitalWrite(feu_orange_2, HIGH);
  delay(1000);
  digitalWrite(feu_orange_2, LOW);
  digitalWrite(feu_rouge_1, LOW);

}
```

4.3.3 Essais sur breaboard avec le microcontrôleur Attiny 3217



Voici donc à quoi ressemble ma structure de l'intersection.
Comme on peut le constater, la programmation ne se fait plus par Arduino uno, mais nano.
On retrouve à côté l'attiny 3217 ainsi que la carte debug.
Pour réaliser ses essais j'ai dû créer un nouveau compilateur de sorte à ce que le programme ce renvoi bien dans le microcontrôleur et y reste. Ce nouveau compilateur a été créé via Arduino.
Nous pouvons aussi observer le level shifter entre les deux grandes breadboard, qui permet de faire la liaison en I2C, avec les fils SDA,SCL,GND et 5V.
Puis pour finir sur la dernière breadboard on retrouve nos feux ainsi que les résistances et les transistors de la structure que j'ai expliqué précédemment.

4.3.4 Programme Anti-rebond

Souvent lorsque l'on appuis sur un bouton-poussoir, il y a un effet rebond. C'est comme si nous avions appuyé plusieurs fois sur le bouton alors que ce n'est pas le cas. Bien évidement, pour les boutons piétons, il ne faut pas que ce cas de figure intervienne. Il est possible de créer une structure avec des composants ou alors un programme pour supprimer ses rebonds. Dans notre cas nous avons choisi d'utiliser un programme.

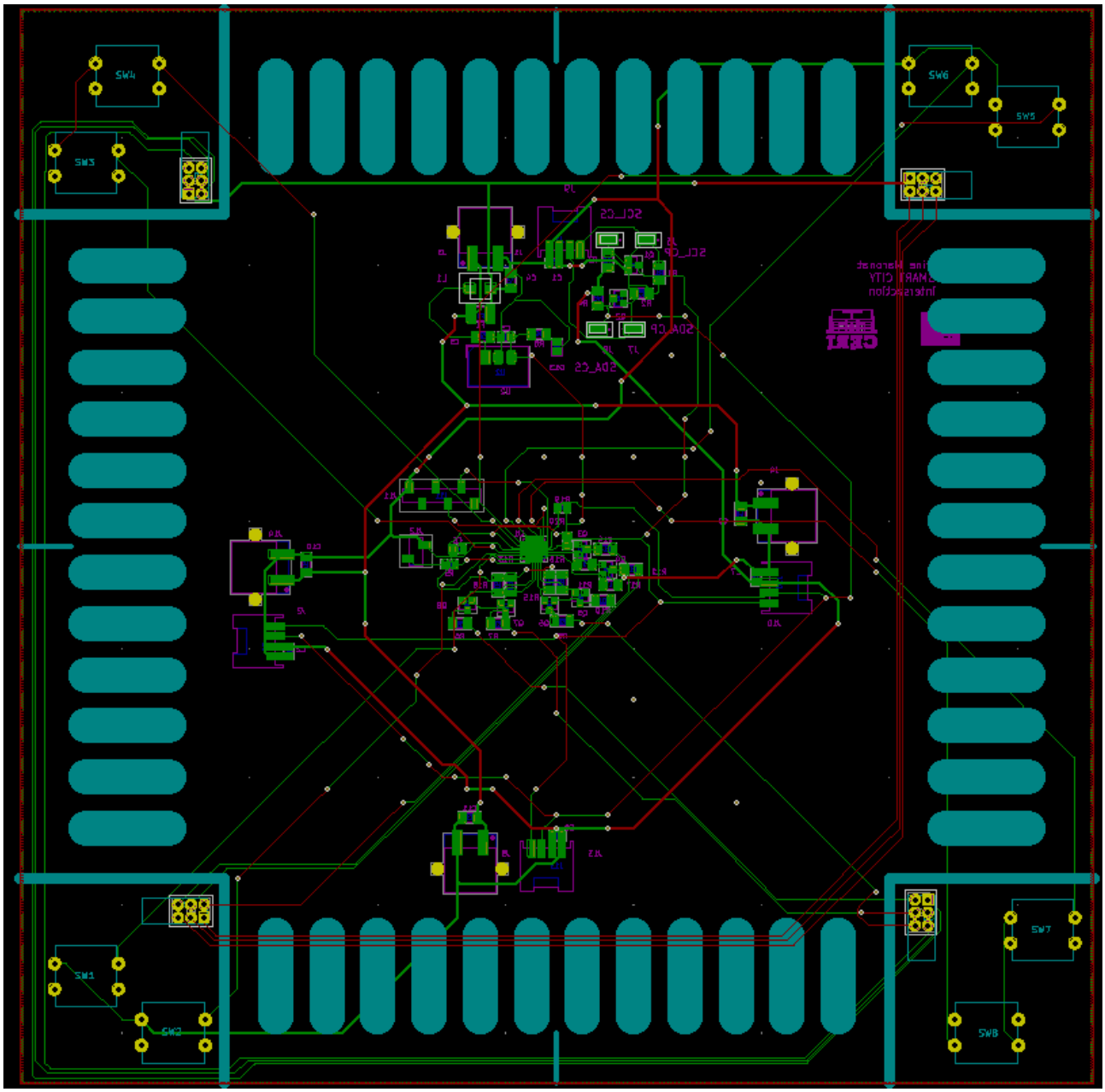
```
void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= 1000) {
    previousMillis = currentMillis;
   // digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN));
  }

}

void debounceInterrupt() {
 static unsigned long lastInterrupt = 0;
 unsigned long interruptTime = millis();
 if (interruptTime - lastInterrupt > 200)
 {
 digitalWrite(LED2, !digitalRead(LED2));
 Serial.print("Comptage : ");
 Serial.println(++val);
 }
 lastInterrupt = interruptTime;
}
 }

 void loop() {
   unsigned long currentMillis = millis();
   if (currentMillis - previousMillis >= 1000) {
```

Afin de voir si ce programme est bel et bien effectif, je vais visualiser la console avec le port com qui correspond à la carte débug.

J'utilise pour cela la console Tera Term.

4.4 Routage de la carte intersection

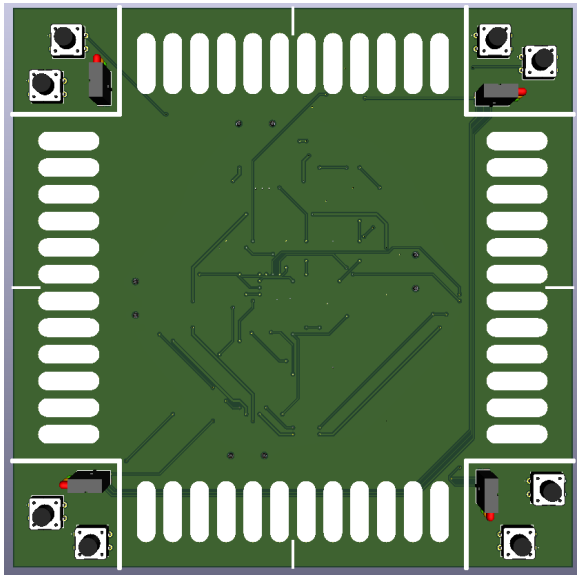
Maintenant voici le routage effectué pour cette carte

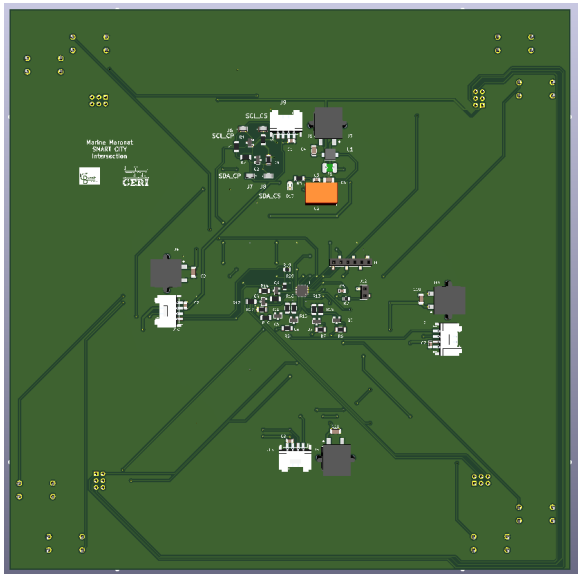


La carte de l'intersection mesure 21x21cm.

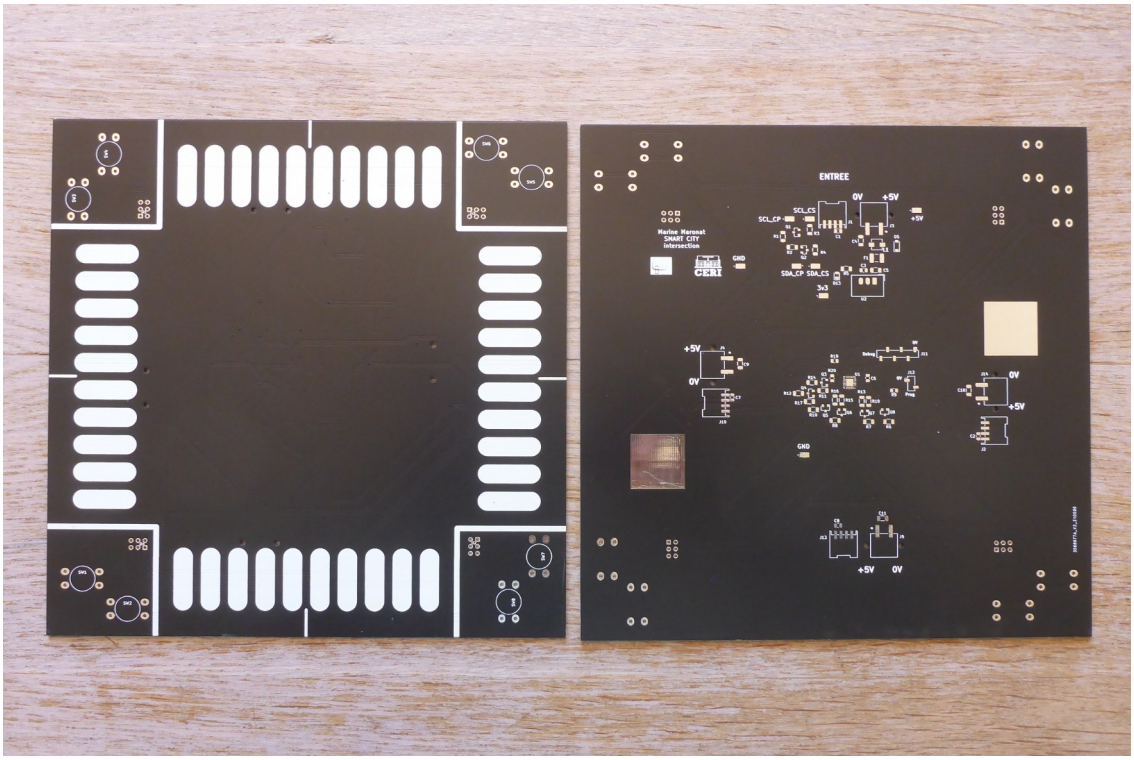
Elle est aux dimensions réelle de l'intersection. En ce qui concerne la disposition, comme expliqué plus tôt, de chaque côté de la carte, comme pour la carte parking, on retrouve les borniers et connecteurs groves de chaque côté. Pour passer de dessus à dessous, j'ai utilisé des vias qui permettent une liaison entre les couches, j'ai donc pu raccorder comme je le souhaitais mes différents chevelus.

Du côté face, j'ai uniquement mis les éléments qui devront être apparent sur l'intersection : les boutons piétons et les feux. Ainsi de l'autre côté se trouve tout les autres composants.
Afin de supporter l'alimentation, j'ai également dû venir élargir les pistes de 3.3V en sortie des connecteurs groves ainsi que des borniers. Nos chevelus avaient de base une épaisseur de 0,25mm. Pour ces chevelus-ci, nous sommes passés à 0,6mm.
Voici un aperçut 3D de ma carte.





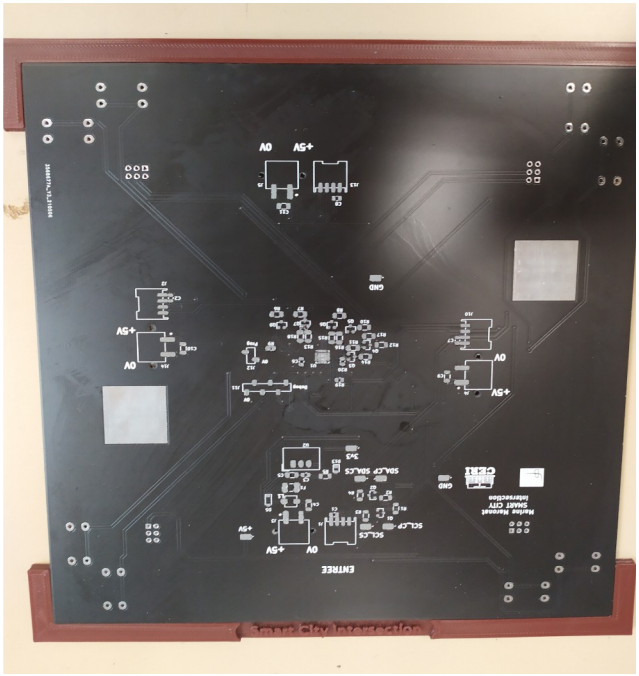
Voici à quoi ressemble cette carte une fois reçu et fabriqué.



Procédé du routage :

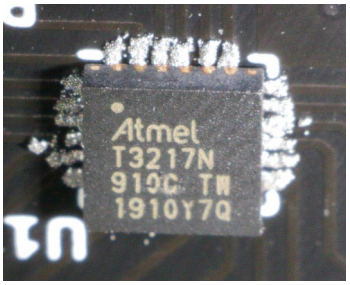
Je vais maintenant décrire les différentes étapes qui m'ont permis de souder ma carte.
Pour commencer, j'ai nettoyé ma carte et son stencil avec de l'acide isopropylique.

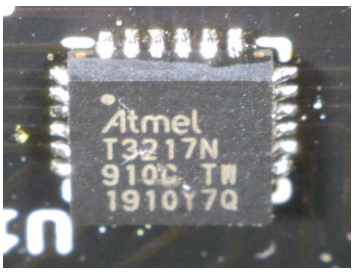
Ensuite, j'ai associé ma carte à son stencil et à son support correspondant.
J'ai positionné la pâte à braser sur le PCB.
Et j'ai une fois retiré le stencil, positionné tous les éléments, en dehors des feux, boutons et connecteurs.



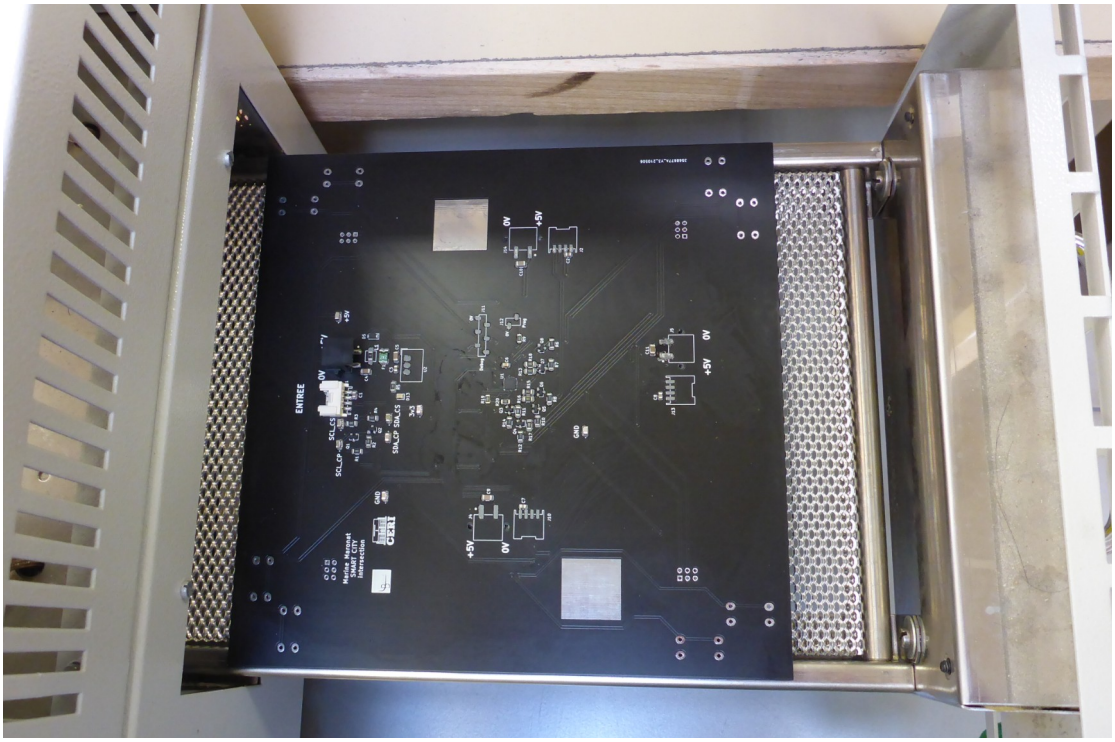
J'ai dû veiller à faire attention au sens des Leds/diodes, que j'ai pu vérifier à l'aide d'un ohmmètre.
Ensuite j'ai regardé à la loupe si tous les éléments étaient bien soudés, en particulier le microcontrôleur et j'ai placé ma carte dans le four pour un programme de 2 minutes 50 avec une température de 253°C.

Voici un avant après de mon microcontrôleur et la carte une fois soudée.

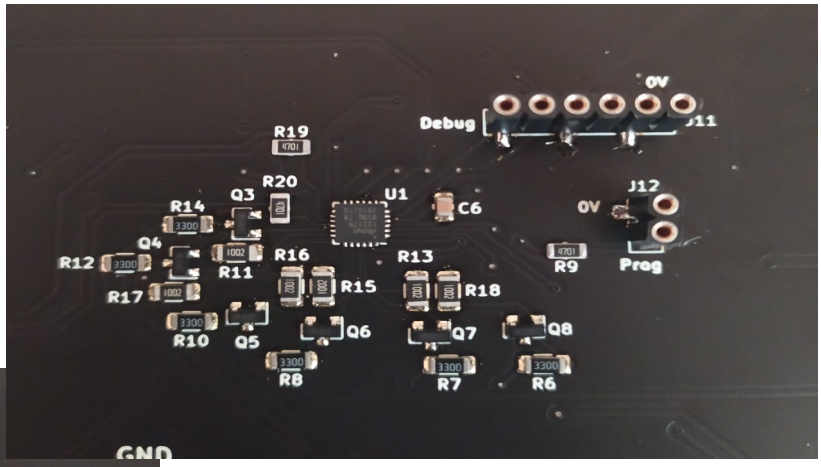


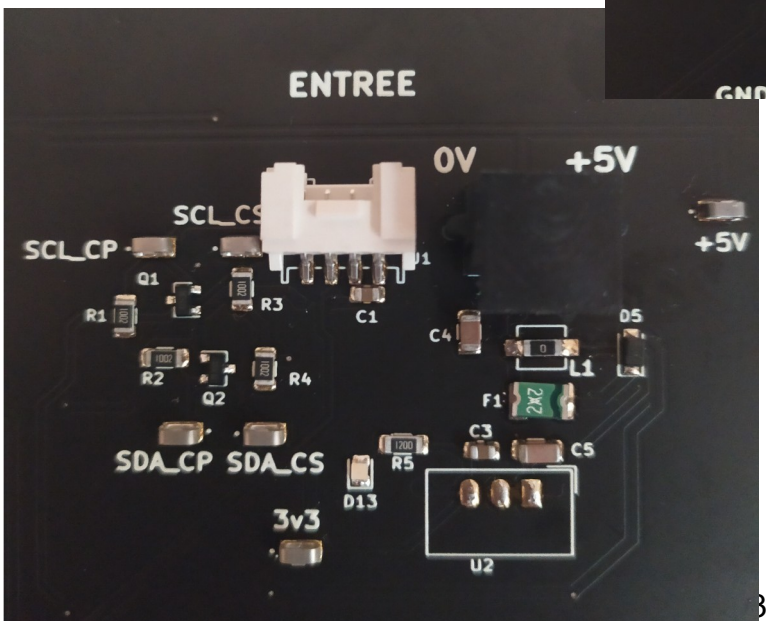


Ma carte en sortant du four :



Éléments de la carte vus de près.





Liste de matériels :

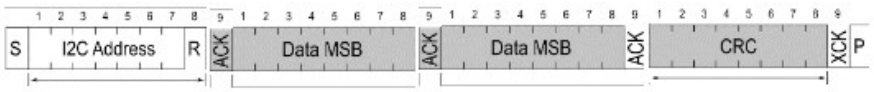
Afin de créer cette liste de matériel, je suis passé par l'outil Bill of materials de Ki-cad, qui va me générer automatiquement une liste.
En revanche, j'ai tout de même fait pas mal de modification puisque de nombreux éléments n'étaient pas utiles. J'ai ensuite ajouté les prix des composants en allant à l'aide des codes commandes sur les sites fabricants.

	Liste de Matériels		Lycée Alphonse Benoit	
Projet SMART CITY Maronat Marine	Carte Intersection		Isle sur la Sorgue	
Références des composants	Valeur	Code Commande	Nombre dans le paquet	Prix TTC
C1, C2, C3, C7, C8	1 µF	RS 264-4450	25	1,05
C4, C5, C9, C10, C11	10 µF	RS 103-4164	20	4,12
C6	100nF	RS 264-4422	25	3,825
D1, D2, D3, D4	LED_Mentor_ROG	Conrad 183654		
D13	LED_CMS_Jau	RS 486-0519	25	6,225
F1	Polyfuse	RS 517-7105	10	3,45
J1, J2, J10, J13	Conn_Groove	Gotronic 36439	vendu à l'unité (0,20)	0,8
J3, J4, J5, J14	Bornier_contact	RS: 162-2155	vendu à l'unité (2,49)	9,96
J6, J7, J8, J9	Point_test	RS 788-4820	10	1,84
J11	Conn_Femelle	Farnell 1629135	vendu à l'unité (5,89)	5,89
J12	Conn_programmation	Farnell 1629135	vendu à l'unité (5,89)	5,89
L1	5,6 Î¼H	RS 200-3529	1	6,81
Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8	2N7002	RS 753-3134	100	5,4
R1, R2, R3, R4, R11, R13, R15, R16, R17, R18	10k	RS 125-1192	100	2,4
R5	120	RS 223-2136	50	0,65
R6, R7, R8, R10, R12, R14	330	RS 679-2049	50	4,25
R9, R19, R20	4.7k	RS 679-2184	50	1,1
SW1, SW2, SW3, SW4, SW5, SW6, SW7, SW8	SW_Push	RS 785-6302	vendu à l'unité (1,41)	11,92
U1	ATtiny3217			
U2	TSR_1-2433	RS 666-4360	vendu à l'unité (5,6)	5,6
			Total Fournisseur RS	68,6
			Total Gotronic	0,8
			Total Farnell	11,78
			Coût total	81,18

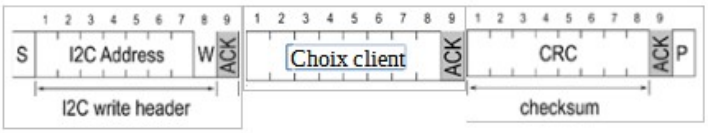
4.5 Protocole I2C

Nous devons renvoyer :
- Le mode
- L'état des feux
- L'état des boutons
- La voie de la trame en question

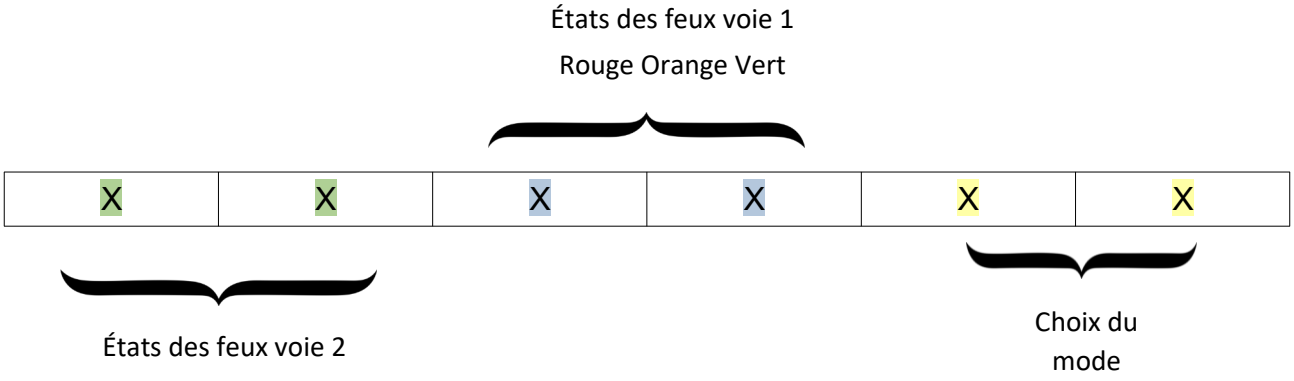
Trame de Lecture :



Trame écriture :



Décodage de la trame d'écriture du maître :



États des feux de la voie 1 :

Code	États des feux
01	Vert
10	Orange
11	Rouge
00	Éteint

Choix du mode

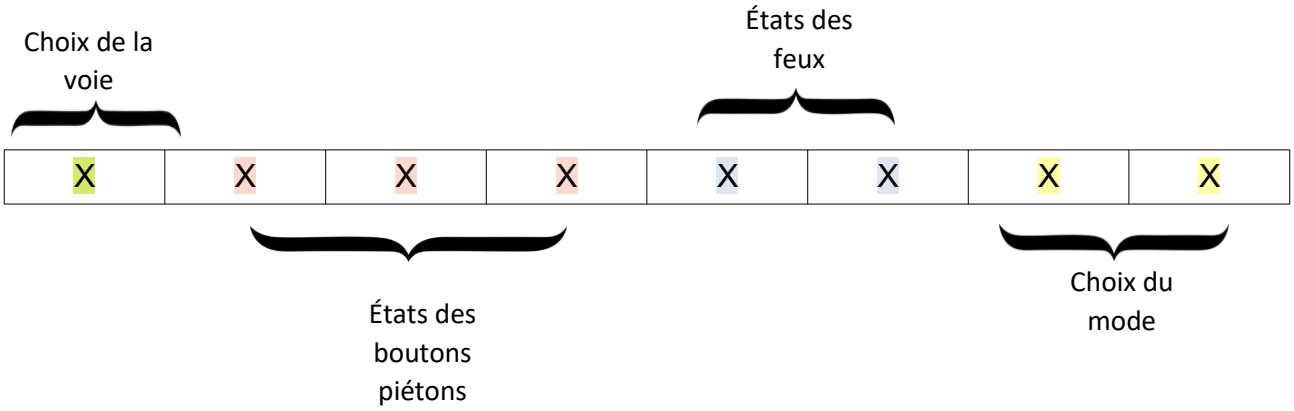
Code	Mode
00	Orange clignotant
01	Auto
10	Manuel

États des feux de la voie 2 :

Code	États des feux
01	Vert
10	Orange
11	Rouge
00	Éteint

Décodage de la trame de lecture (renvoi des données) :

2 Octets sont envoyés au client (un pour chaque voies).



Choix du mode

Code	Mode
00	Orange clignotant
01	Auto
10	Manuel

Identification de la voie de la trame envoyé

Code	Voie
0	Voie 1
1	Voie 2

État des feux de la voie :

Code	États des feux
01	Vert
10	Orange
11	Rouge
00	Éteint

État des quatre boutons de la voie

Code	État Bouton
000	Aucun n'est enclenché
001	Bouton piéton 1 Enclenché
010	Bouton piéton 2 Enclenché
011	Bouton piéton 3 Enclenché
100	Bouton piéton 4 Enclenché

4.5.1 Programme Esclave

Afin de pouvoir communiquer avec le client par l'intermédiaire de la raspberry, il faut créer un programme où figura l'adresse qui m'a été attribué, dans mon cas nous sommes convenue de l'adresse 10. (Fonction wire.begin)

```
#include <Wire.h>
#define feu_rouge_1 12
#define feu_orange_1 13
#define feu_vert_1 14

#define feu_rouge_2 15
#define feu_orange_2 16
#define feu_vert_2 17

void setup() {
  Wire.begin(10);
  Serial.begin(9600);
  Wire.onRequest(requestEvent); // recevoir ou envoyer des données ( evenement )
  pinMode (feu_rouge_1, OUTPUT);
  pinMode (feu_orange_1, OUTPUT);
  pinMode (feu_vert_1, OUTPUT);
  pinMode (feu_rouge_2, OUTPUT);
  pinMode (feu_orange_2, OUTPUT);
  pinMode (feu_vert_2, OUTPUT);


}
int voie1 = 0x00;  // Etat de la voie au départ
int voie2 = 0x80;  // Etat de la voie au départ

void loop() {
  delay(100);

mode_feu();

}
 void requestEvent() {

    Wire.write(voie1);
    Wire.write(voie2);

 }
 void mode_feu() {
 voie1 &= 0xFC;
 voie1 |= 0x01;
 voie2 &= 0xFC;
 voie2 |= 0x01;
   digitalWrite(feu_orange_2, LOW);
   digitalWrite(feu_rouge_1, LOW);
   digitalWrite(feu_vert_1, HIGH);
 voie1 &= 0xF3;
 voie1 |= 0x04;
```

Voici un extrait du programme pour retranscrire à l'aide porte logique, l'état du feu (couleurs). Pour ce faire je suis partie de porte logique ET pour mettre tous les bits de la voie à zéro et à l'aide du porte logique OU j'ai mis les 1 à l'état correspondant.
Par exemple dans notre cas, on souhaite à ce stade du programme renvoyé que le feu de la voie 1 est vert.
On part d'une voie où tout est à 1.

VOIE 1 ET F3 1111 1111
1111 0011
= 1111 0011

On reprend le résultat avec une porte logique OU 04

1111 0011
0000 0100
= 1111 0111

Dans le protocole établit plus haut, les bits correspondant à la couleur sont ceux surlignés. Nous avons 01 ce qui correspond bien à la couleur vert. L'état des autres bits n'a pas d'importance, puisque dans ce cas présent, on souhaite renvoyer la couleur du feu.
J'ai donc reproduis ce mécanisme pour chaque changement de couleur.

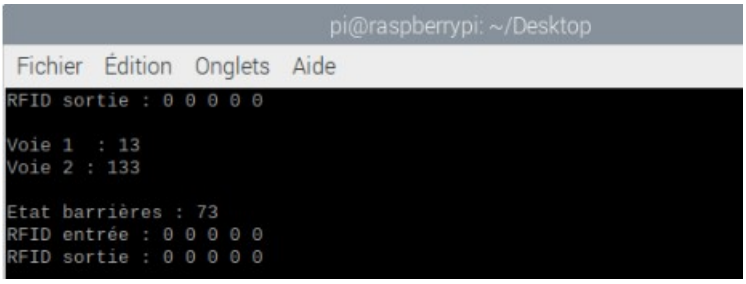
4.5.2 Test de réception des données avec le client

J'ai pu ensuite après avoir finalisé ma partie du programme, tenter de communiquer avec le client et ainsi vérifier si tout se passe comme convenu.
Le client à l'aide de la commande I2cdetect dans l'invite de commande à bien pu lire que ma structure était présente à l'adresse 10.

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- 0a -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

A l'aide de son programme, il a pu également recevoir l'état en permanence de mes feux. Le test est donc concluant, la communication fonctionne.

De notre côtés, nous avions quand même moyen de vérifier sans le client, si la raspberry recevait bien les données.
Voici ce que nous renvoyer le terminal.



Ici donc la voie 1 à 13 ce qui correspond donc à la trame suivante : 0000 1101.
Décodage de la trame :
- Le bit de poids fort est à 0, nous sommes sur la voie 1.
- Les trois bits suivants sont à zéro ce qui indique qu'aucun bouton piéton est enclenchés.
- Le feu est rouge.
- Mode automatique.

Pour la voie 2 on reçoit 133 ce qui correspond à la trame 1000 0101
Décodage de la trame :

- bit de poids fort à 1 pour indiquer que l'on est sur la voie 2
- Les trois bits suivants sont à zéro ce qui indique qu'aucun bouton piéton est enclenchés.
- le feu est vert
- mode auto

De plus, ce test nous a également permis de vérifier si la structure choisis pour le level shifter fonctionnait. La structure choisis est donc idéal pour nos projets.

5 Conclusion

Maintenant, que j'ai pu réaliser mes deux cartes, je vais pouvoir faire différents tests notamment l'I2C. Côté programmation, il me reste encore à terminer le programme qui permet de faire fonctionner mon intersection dans le mode manuel. C'est-à-dire

que c'est le client qui « choisit » l'état de mes deux voies..

Du point de vue personnelle, j'ai beaucoup aimé travailler sur ce projet, nous sommes parties de zéro et de fil en aiguille, nous avons pu créer toute une structure répondant aux attentes de nos contrats. De mon côté, j'ai aimé faire la démarche de recherche et de tests avec le microcontrôleur. J'ai hâte maintenant de voir mes deux cartes en fonctionnement.