



BTS SN SESSION 2017 E6_2

LYCÉE BRANLY

PROJET ROBOT BLOCS

QUIVET CYPRIEN
ASTIER MATHIEU
MARGUIN GAEL
GIANINO YOAN
ANDREUX ALEXANDRE
POSSICH SAMEL
DIAZ VINCENT
PERONNET VINCENT

SOMMAIRE:

0 Partie commune	4
0.1 - Entreprise	4
0.2 - Situation du projet dans son contexte	4
0.3-Spécifications	5
0.3.1 - Diagramme SYML.....	5
0.3.2 Contrainte de réalisation.....	7
0.4- Répartition des fonctions ou cas d'utilisation par étudiant	8
0.5 - Planification	12
0.6 - Diagramme de Gantt.....	13
I) Etude et conception	16
1) Cahier des charges.....	16
1) Solutions retenues	17
2) Diagramme de bloc interne	18
3) Communication USB.....	19
4) Communication Wifi	20
5) Planification	21
6) Conception des composants assistée par ordinateur.....	22
II) L'interface homme machine	24
1) Présentation	24
2) Objectifs.....	25
3) Cahier des charges.....	26
4) Aperçu de l'IHM achevée	26
5) Présentation de l'environnement de développement.....	27
6) Explications et validations	29
III) Configuration Wifi	39
1) Configuration du module ESP8266	39
2) Configuration du serveur.....	40
3) Programmation de la communication Arduino/ESP8266	41
4) Validations	42
Annexes étudiant 1.....	1
Annexe n°1 : Extrait de la documentation FT232RL.....	1
Annexe n°2 : Extrait de la documentation ESP8266	2
Annexe n°3 : Commandes AT	3
1ère Partie : Objectifs.....	5
Mes attributions :	5
Cahier des charges.....	6
IBD détaillé (Diagramme de bloc interne).....	7

Alimentation	8
2ème Partie : Choix et présentation des composants	10
Carte Arduino mini Pro.....	12
Le module Xbee Pro	13
Ecran Oled 128 x 64	14
Le chargeur de Batterie MCP73831	15
Le régulateur de tension LM317T	16
3ème Partie : Programmation	17
4ème Partie : Schéma structurel et Typon.....	19
Schéma structurel.....	19
Typon	20
5ème Partie : Mesures et validation	21
Réalisation finale	27
ANNEXES	28
Étudiant 3 : Lecture du programme construit et communication robot.....	33
Présentation.....	33
Le protocole 802.15.4	36
Le protocole ZigBee.....	36
Caractéristiques des modules XBee série 1	37
Données techniques	38
Types d'antennes	38
4. PARTIE INDIVIDUELLE YOAN GIANINO	48
4.1 Présentation générale du système supportant le projet :	48
4.2 Étudiant 4 blocs fonction	49
4.3 Cahier des charges.....	50
4.4 Planning.....	51
4.5 Choix des composants.....	52
4.5 Principe et branchement	55
4.6 Relever des tests	59
4.7. Document de fabrication	65
4.8. Coût.....	66
Conclusion	69
Lexique :	70
Programme	70
Partie Personnelle Alexandre ANDREUX.....	83

0 Partie commune

0.1 - Entreprise

Le projet est réalisé en partenariat avec l'association d'éducation populaire "les petits débrouillards", qui à été fondée en 1986. Cette association regroupe de nombreuses structures dont le but commun est de faire partager la curiosité scientifique au plus grand nombre.

0.2 - Situation du projet dans son contexte

L'objectif du projet est de concevoir un système permettant à des enfants âgées de 6 ans ou plus, sachant lire ou non, pouvant avoir des difficultés motrices, d'apprendre les bases de la programmation et de l'algorithmique.

Il s'agit de créer un système similaire à la programmation sous Scratch, avec des blocs universels pouvant être configurés pour effectuer différentes instructions, par exemples des actions ou des boucles. Les étiquettes de Scratch sont remplacées par des afficheurs LED. Le système doit pouvoir piloter des parties opératives diverses par l'intermédiaire d'un PC ou d'un Smartphone.

L'enfant assemble les blocs de manière à constituer son programme.

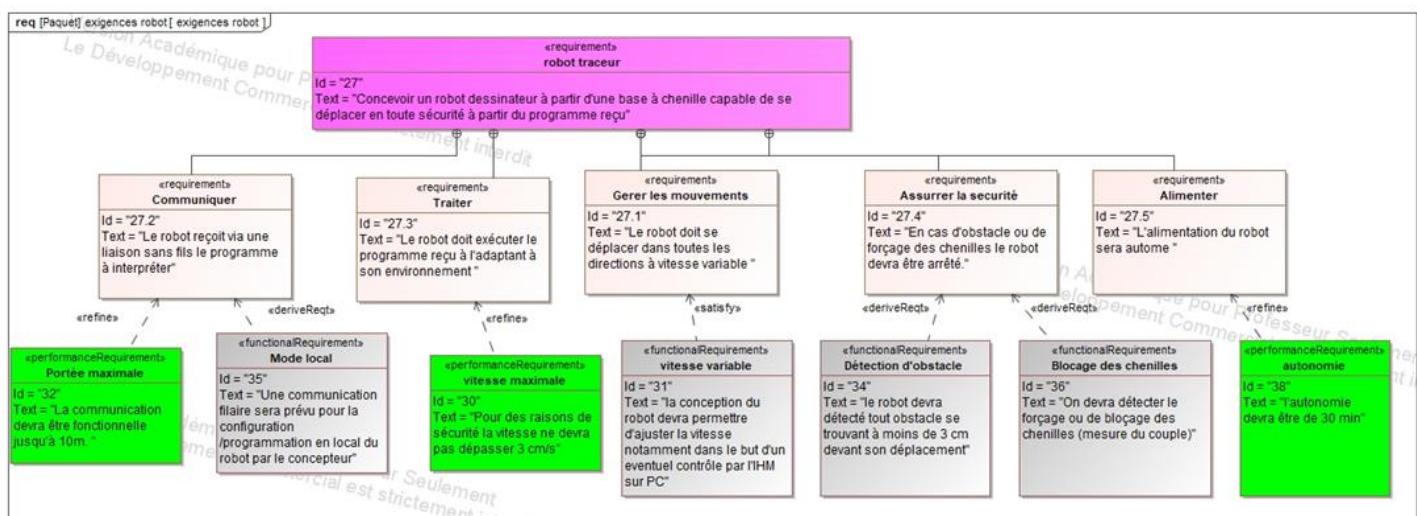
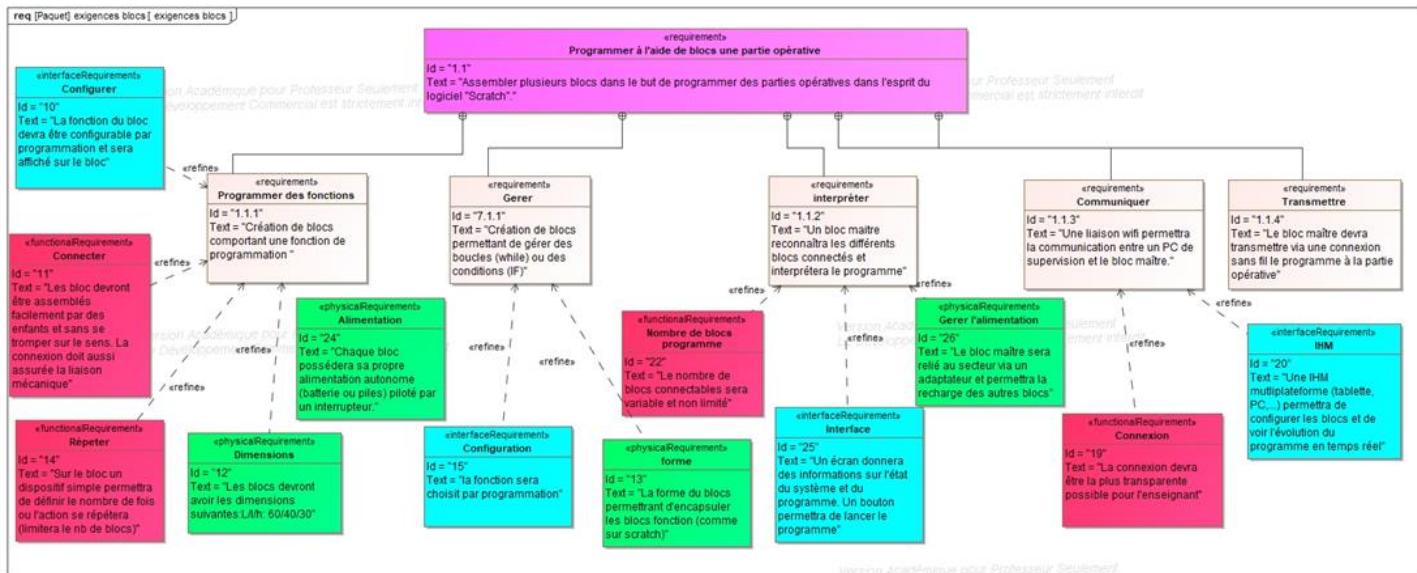
Toutes les 5 secondes, le blocs maître demande l'instruction au bloc suivant, qui transmet à son tour la demande au suivant jusqu'au dernier. Ensuite les informations remontent jusqu'au bloc maître.

L'appui sur le bouton "Départ" du bloc maître permet de transmettre le programme vers la partie opérative, c'est-à-dire le robot.

0.3-Spécifications

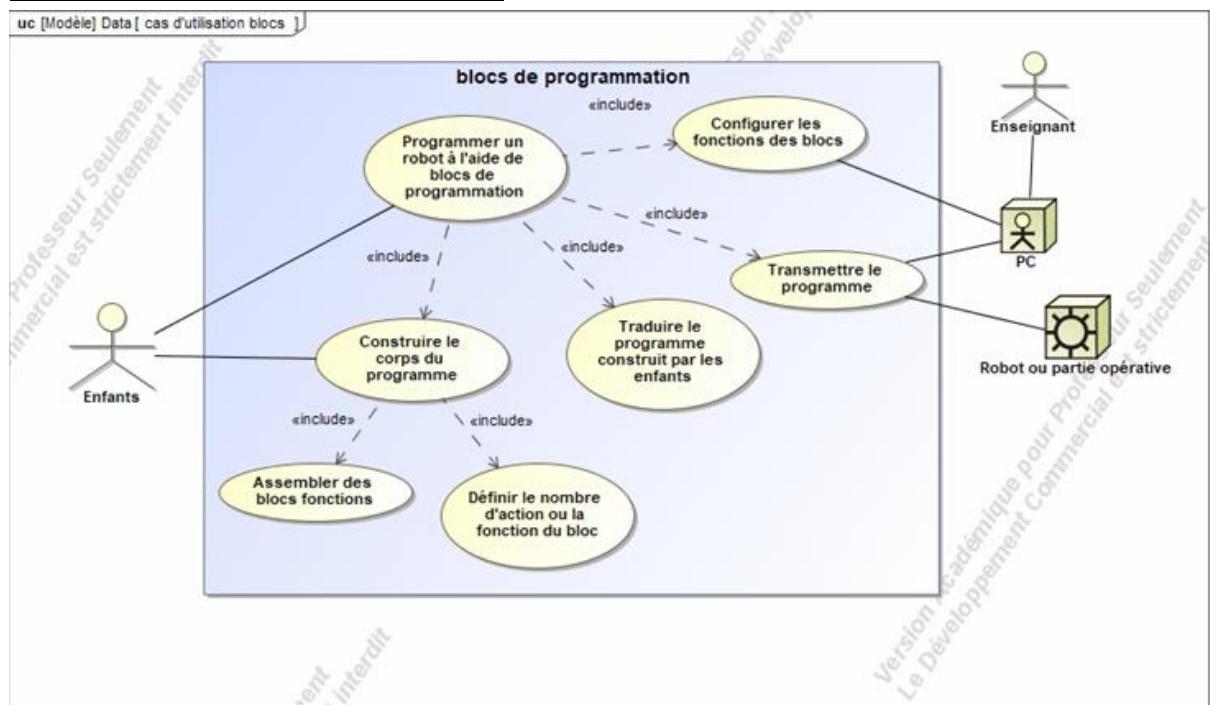
0.3.1 - Diagramme SYSML

0.3.1.1. Diagramme d'exigences

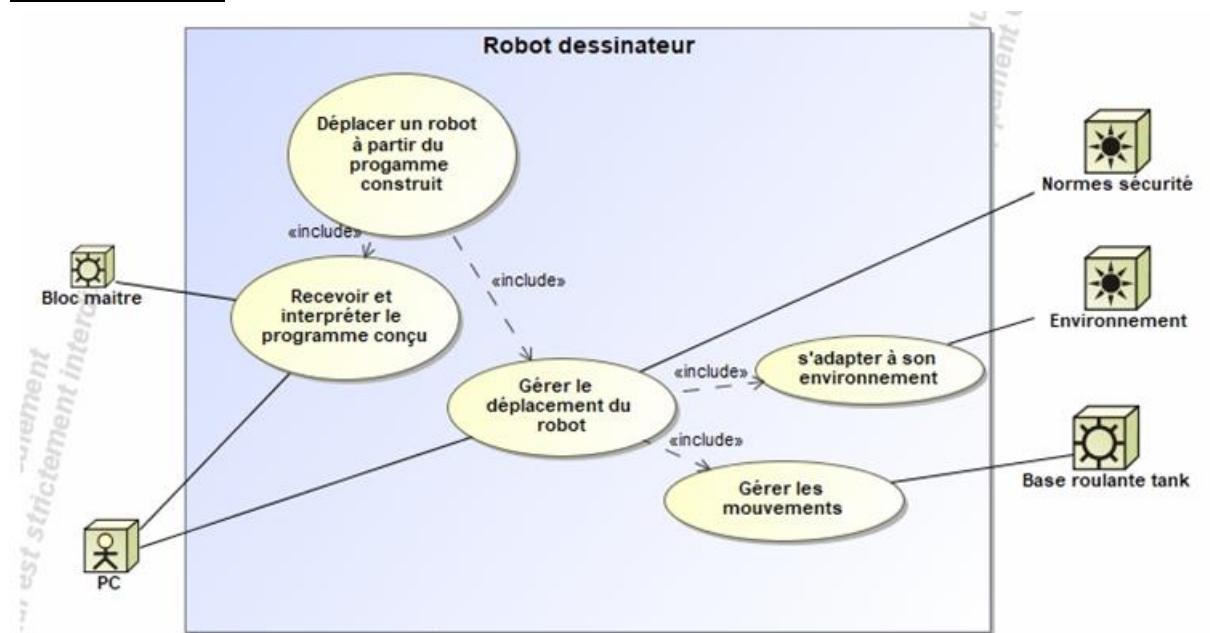


Cas d'utilisation :

Pour les blocs de programmation :



Pour le Robot :



0.3.2 Contrainte de réalisation

Contrainte matérielle et logiciel :

- Les moyens matériels et logiciels utilisés sont ceux disponibles au lycée.
- Utilisation de micro-contrôleur ESP32.
- Les connecteurs entre les blocs sont de types USB 2.

Outil développement :

- On utilise [l'IDE](#) Arduino ou Visual Studio pour le développement des cartes.
- L'Interface Homme Machine IHM est réalisée à l'aide de Windows Forum dans un environnement C++.
- Xctu : Configuration du Xbee

Exigences de développement :

- Rédaction du dossier de projet conforme aux exigences de l'épreuve E6.2, l'ensemble des documents est fourni en version pdf.

Programmes :

- Fournir les fichiers sources des programmes, commentés et documentés.

Tests :

- Création d'un environnement de test pour chacune des tâches indépendantes.

Schémas :

- Réalisation des schémas à l'aide du logiciel EAGLE.

Essais :

- Fournir des relevés des grandeurs internes au système en vue de sa maintenance et du transfert de technologie vers l'entreprise donneur d'ordre.

Routage :

- Afin de minimiser les coûts, les blocs doivent être identiques.

0.4- Répartition des fonctions ou cas d'utilisation par étudiant

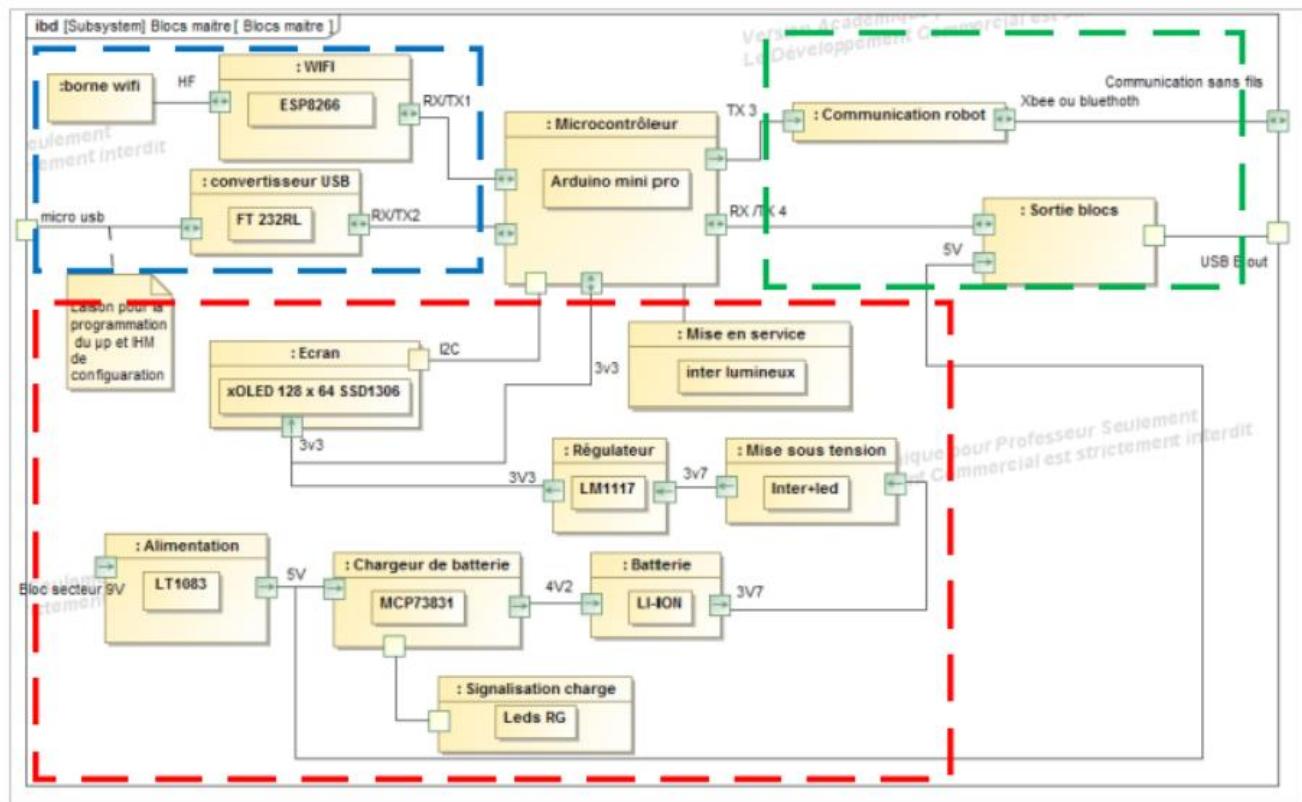
N° étudiant du projet	NOM prénom
Etudiant 1	QUIVET Cyprien
Etudiant 2	ASTIER Mathieu
Etudiant 3	MARGUIN Gaël
Etudiant 4	GIANINO Yoan
Etudiant 6	ANDREUX Alexandre
Etudiant 7	POSSICH Samel
Étudiant 8	DIAZ Vincent
Étudiant 9	PERONNET Vincent

0.4.2.1. Etudiant 1, 2 et 3

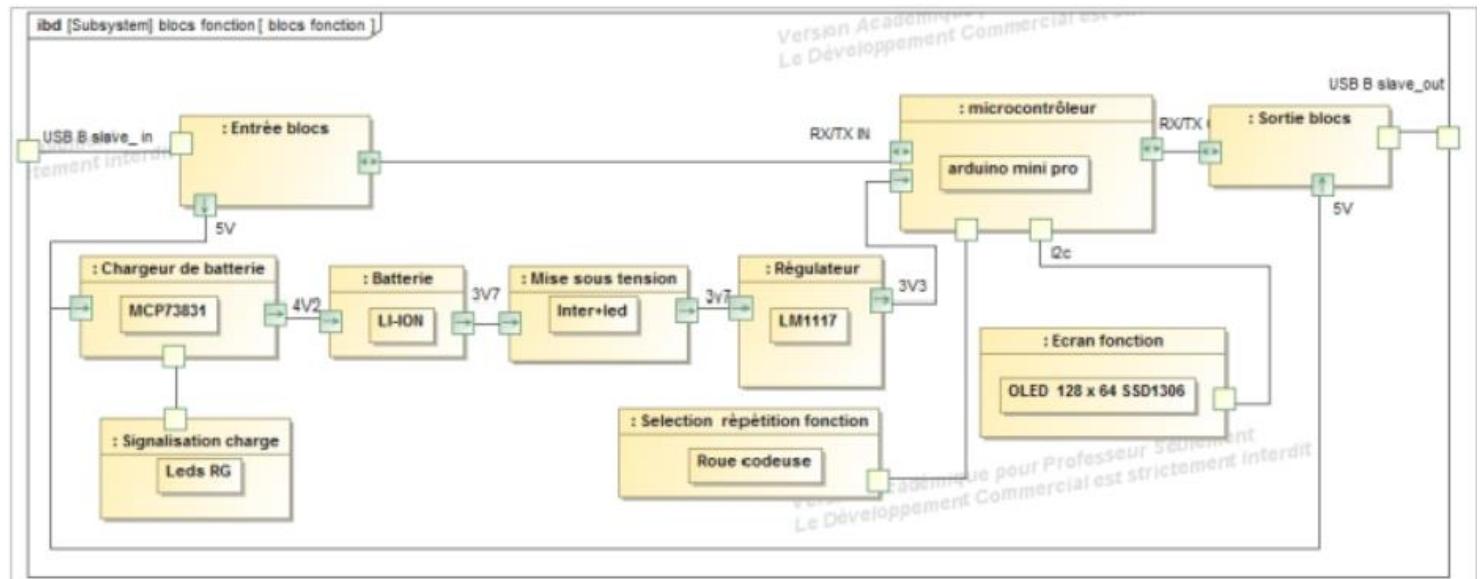
Etudiant1: IHM PC et communication (USB et WIFI)

Etudiant2: Gestion alimentation et écran carte

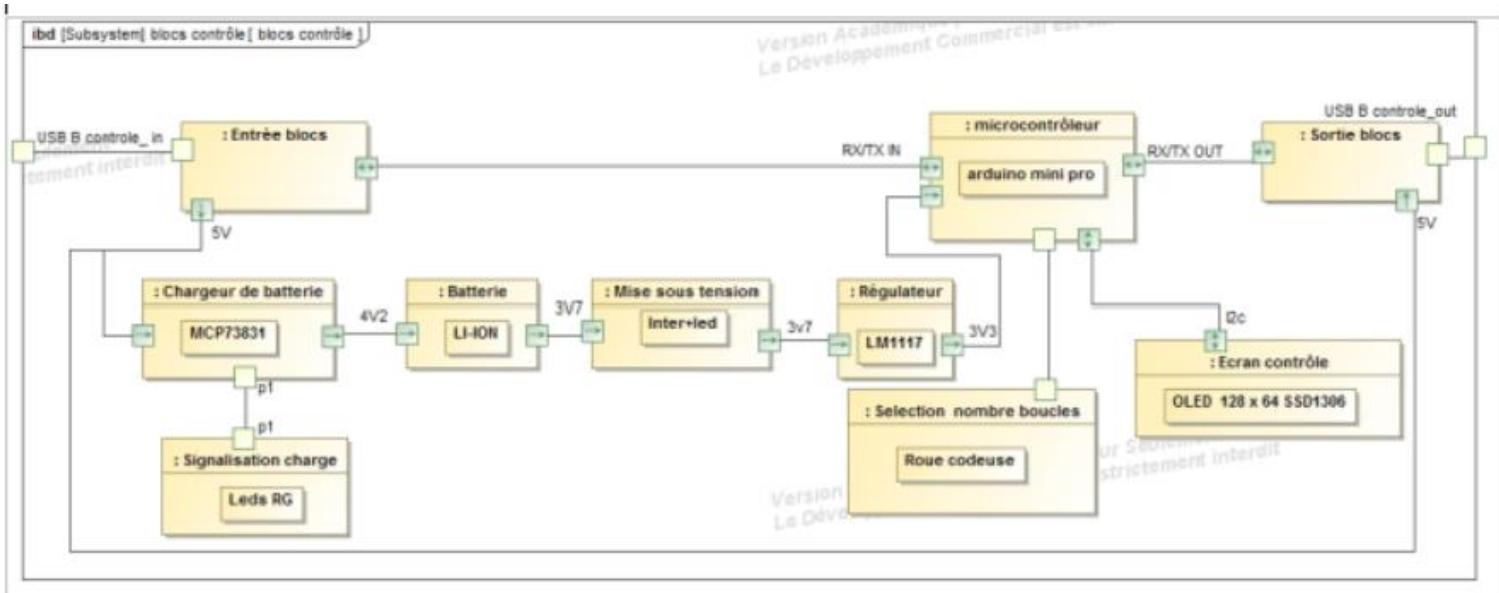
Etudiant3: Lecture du programme construit et communication robot



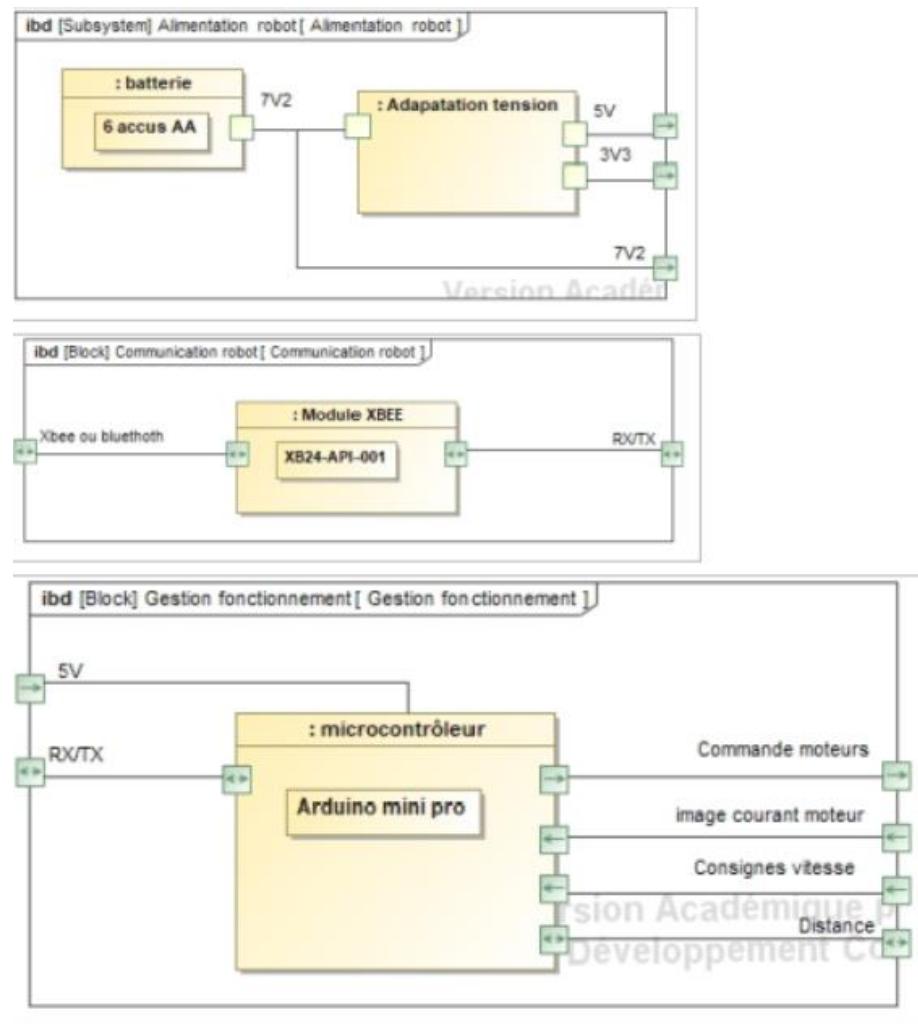
0.4.2.2 étudiant 4 : Blocs fonction avec Arduino Mini Pro



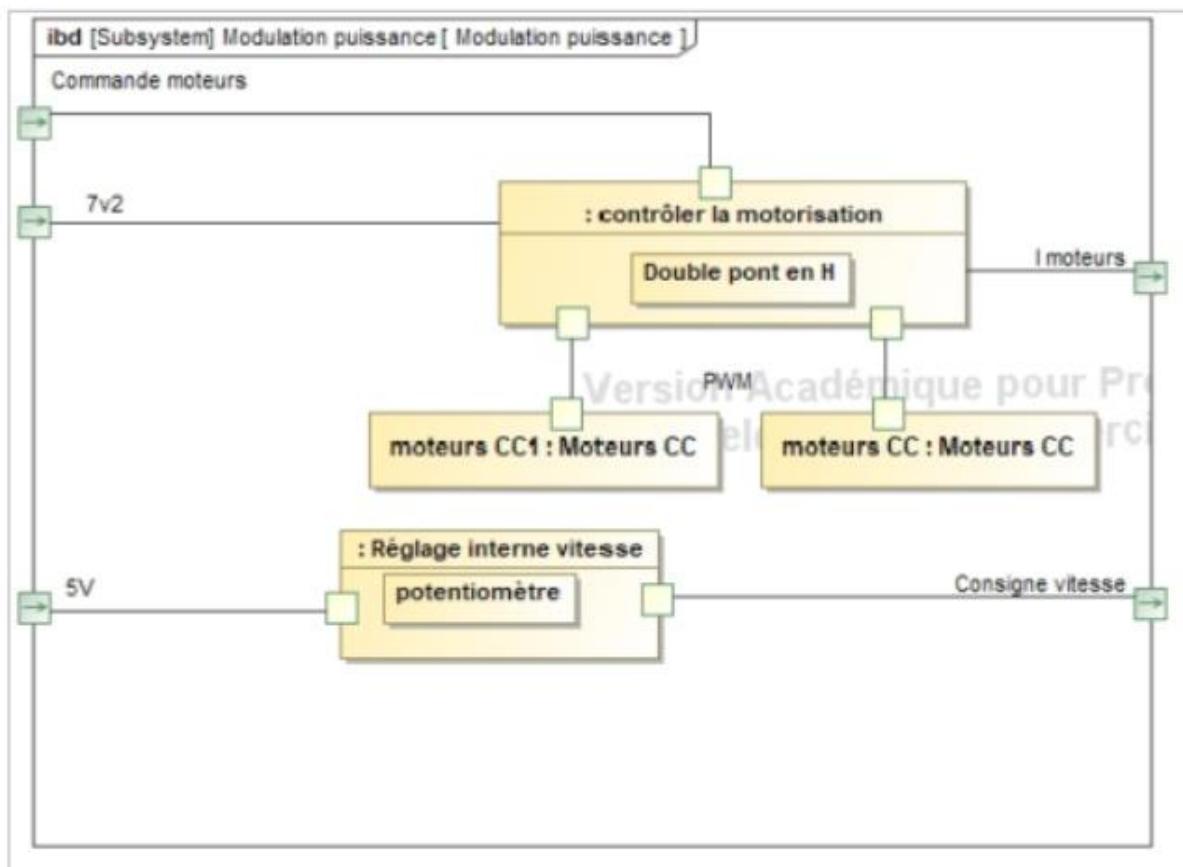
0.4.2.3 étudiant 6 : bloc contrôle



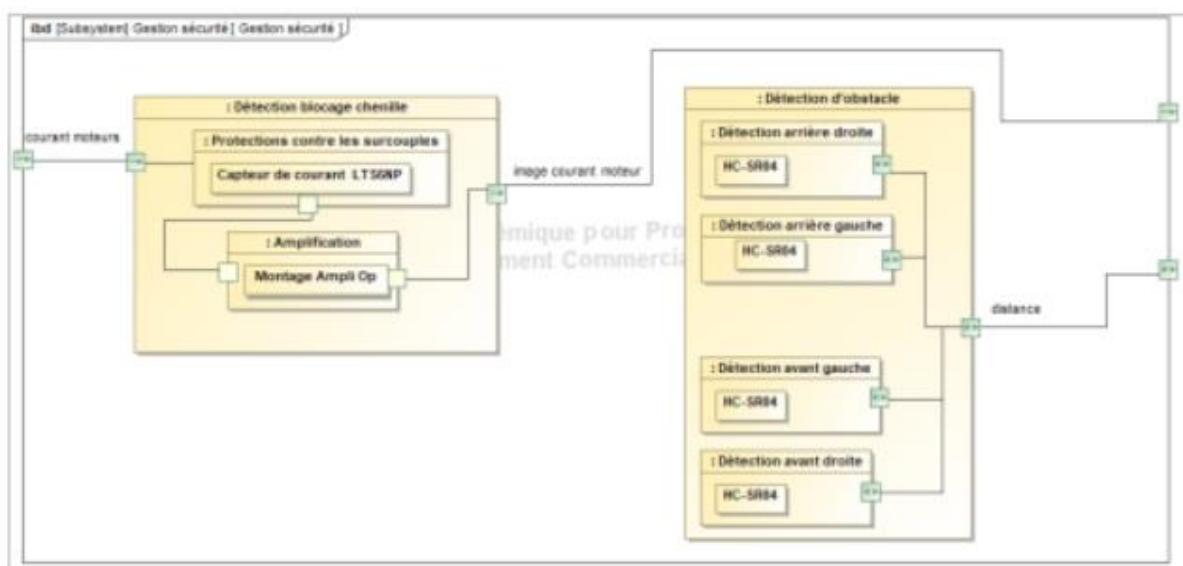
0.4.2.4 étudiant 7 : gestion communication et fonctionnement du robot



0.4.2.5 étudiant 8 : motorisation robot



0.4.2.6 étudiant 9 : sécurité robot



0.5 - Planification

Début du projet	Janvier 2018
Revue 1	01/02/2018
Revue 2	13/03/2018
Revue 3	03/05/2018
Remise du rapport	07/06/2018
Soutenance finale	19/06/2018 au 20/06/2018

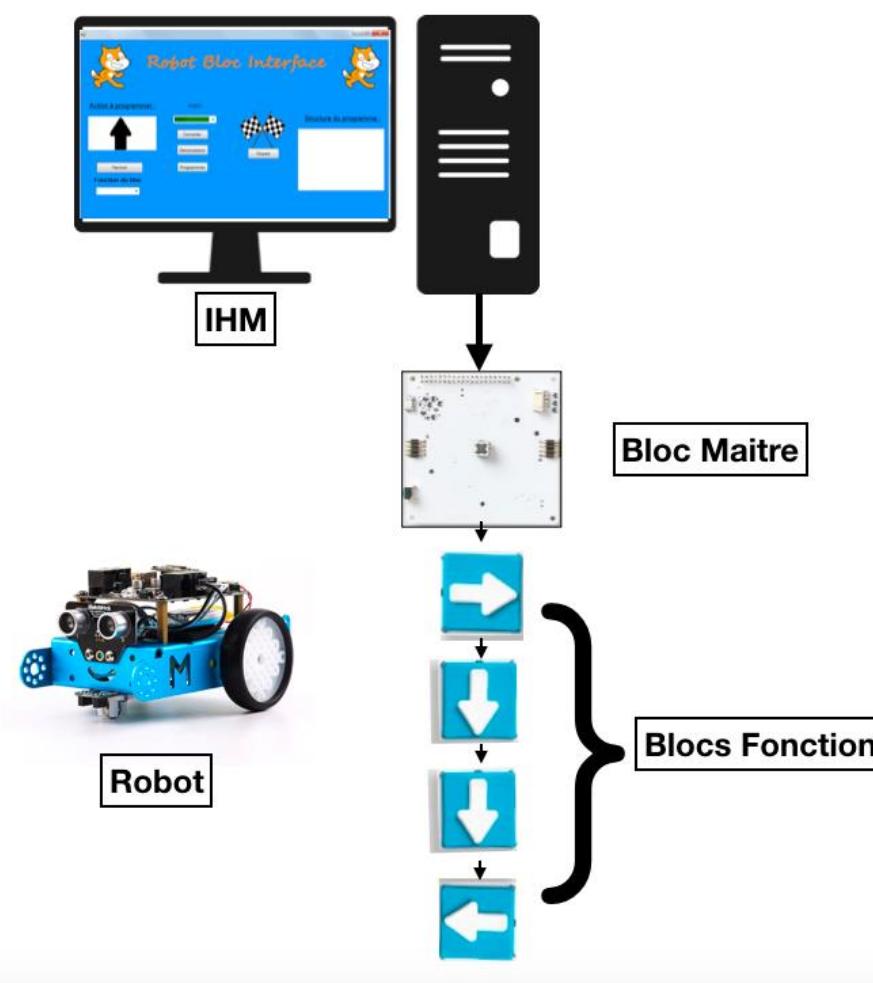
0.6 - Diagramme de Gantt

	Plan...	Nom de tâche	Durée	Début	Fin	Prédécesseurs	Progression
1	⚠️ ➔	Projet BTS SN 2017	53 jours?	08/01/2018	28/05/2018		0%
2	➡️	▣ Appropriation du Cahier des Charges	5,5 jours	09/01/2018	19/01/2018		0%
3	📅 ➔	Répartition du travail entre les étudiants	4 h	09/01/2018	09/01/2018		0%
4	📅 ➔	Description des contraintes spécifiques pour cha...	14 h	12/01/2018	19/01/2018	3	0%
5	➡️	▣ Ajustement du diagramme de blocs	5 jours	19/01/2018	29/01/2018		0%
6	📅 ➔	Réalisation des diagrammes de bloc internes	20 h	19/01/2018	29/01/2018	4	0%
7	📅 ➔	Description des interface (protocoles)	20 h	19/01/2018	29/01/2018	4	0%
8	➡️	▣ Recherche du schéma structurel	11,25 jours	05/02/2018	13/03/2018		0%
9	📅 ➔	Calcul des composants	10,75 jours	05/02/2018	12/03/2018		0%
10	📅 ➔	Validation en simulation	10,75 jours	05/02/2018	12/03/2018		0%
11	📅 ➔	Test sur platine de développement	10,75 jours	05/02/2018	12/03/2018		0%
12	📅 ➔	Dernier jour des commandes de composants	0 jours	13/03/2018	13/03/2018		0%
13	➡️	▣ Réalisation de la maquette	18,75 jours?	12/03/2018	03/05/2018		0%
14	📅 ➔	Routage des circuits	10 jours	12/03/2018	30/03/2018	11	0%
15	📅 ➔	Dernier jour de commande des circuits imprimés	0 jours?	27/04/2018	27/04/2018		0%
16	📅 ➔	Soudure des cartes	8,5 jours	02/04/2018	03/05/2018		0%
17	📅 ➔	Rédaction des procédures de validation du matériel	12,5 jours	13/03/2018	05/04/2018		0%
18	📅 ➔	Validation du matériel	9,25 jours	23/03/2018	24/04/2018		0%
19	➡️	▣ Recherche des solutions logicielles	28 jours	26/01/2018	05/04/2018		0%
20	📅 ➔	Réalisation des diagrammes de séquence	13,25 jours	26/02/2018	22/03/2018		0%
21	📅 ➔	Réalisation des diagrammes d'activité	19,25 jours	26/01/2018	20/03/2018		0%
22	📅 ➔	Réalisation des diagrammes d'état	19,25 jours	26/01/2018	20/03/2018		0%
23	📅 ➔	Rédaction des procédures de validation des logic...	19,25 jours	09/02/2018	03/04/2018		0%
24	📅 ➔	Ecriture des programmes	13,25 jours	12/03/2018	05/04/2018		0%
25	📅 ➔	Validation des logiciels	13,25 jours	12/03/2018	05/04/2018		0%
26	➡️	▣ Assemblage et validation globale	10 jours	23/04/2018	17/05/2018		0%
27	📅 ➔	Validation in situ	5,75 jours	26/04/2018	11/05/2018		0%
28	📅 ➔	Vérification de conformité et mise au point	9,25 jours	24/04/2018	17/05/2018		0%
29	📅 ➔	Tests d'ensemble	10 jours	23/04/2018	17/05/2018		0%
30	➡️	▣ Rédaction du projet	53 jours	08/01/2018	28/05/2018		0%
31	📅 ➔	Rédaction du rapport	37,75 jours	05/02/2018	25/05/2018		0%
32	➡️	▣ Préparation des supports pour les présentatio...	53 jours	08/01/2018	28/05/2018		0%
33	📅 ➔	Préparation du Power point 1	8 jours	15/01/2018	30/01/2018		0%
34	📅 ➔	Revue 1	4 jours	01/02/2018	08/02/2018	33;6	0%
35	📅 ➔	Préparation du Power point 2	6 jours	05/02/2018	01/03/2018		0%
36	📅 ➔	Revue 2	8 jours	13/03/2018	27/03/2018	35;9;10	0%
37	📅 ➔	Préparation du Power Point 3	3,5 jours	08/01/2018	12/01/2018		0%
38	📅 ➔	Revue 3	6 jours	03/05/2018	18/05/2018	37;18;25;16;17	0%
39	✓ ➔	Rendu du rapport	0 jours	28/05/2018	28/05/2018	28;29;27	100%

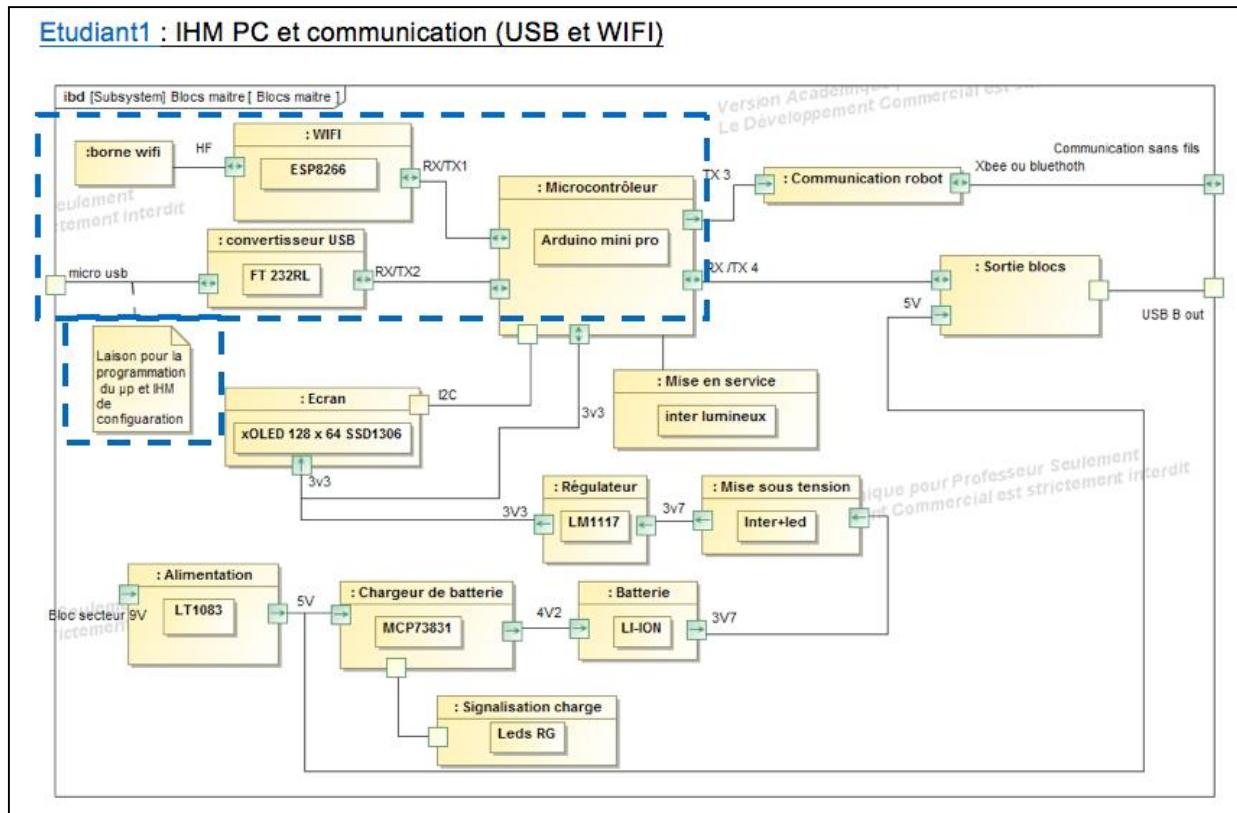
RAPPORT:

Projet Robot Blocs

(partie Bloc Maître)



Présentation de la tâche



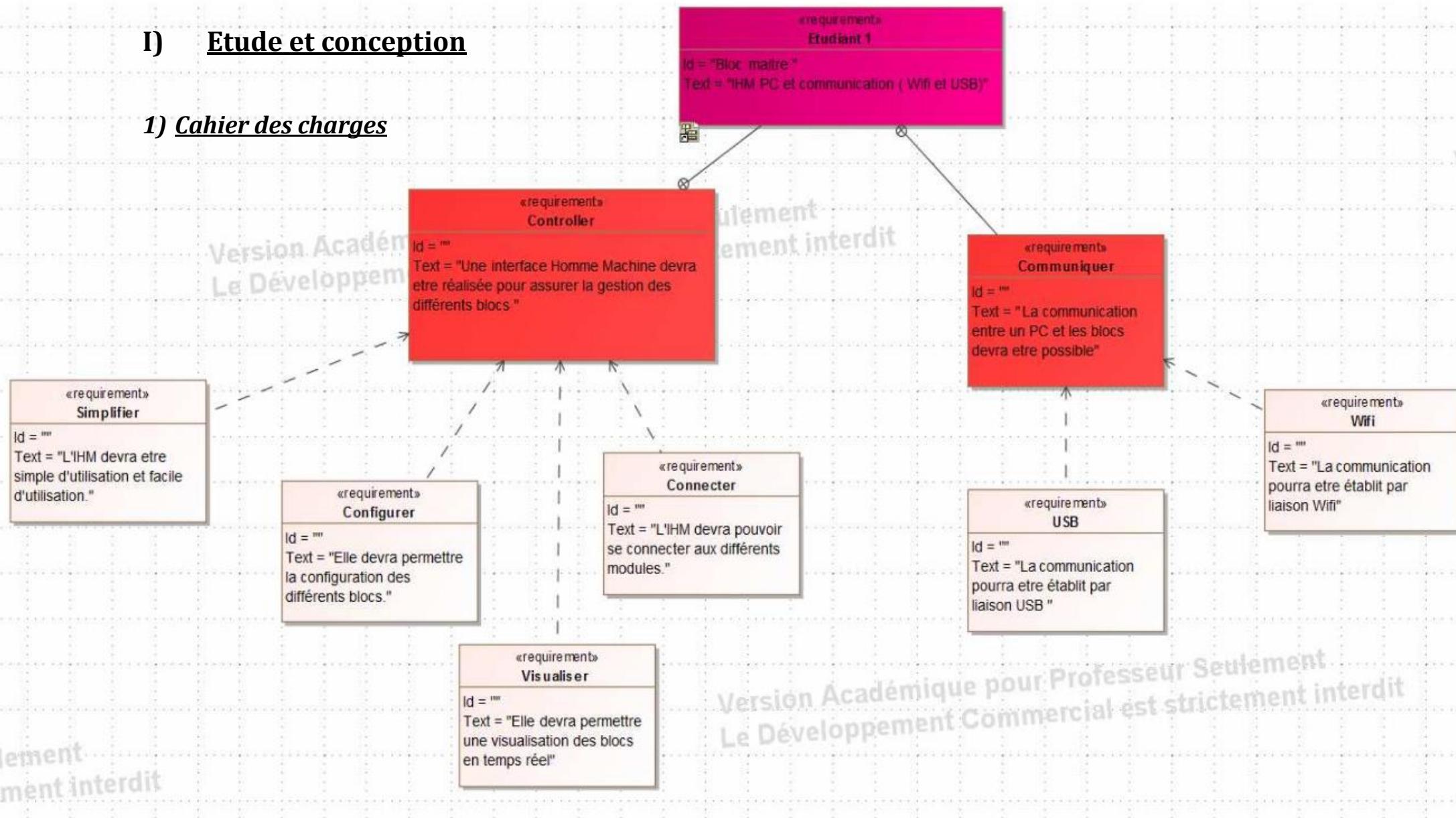
Tout au long de notre projet, ma mission fut divisée en deux parties :

- La première fut la création d'une Interface Homme machine complète qui devait permettre de répondre à diverses contraintes que nous détaillerons par la suite.
- La seconde consista à gérer la communication entre notre bloc maître, les blocs fonctions et l'IHM présente sur un ordinateur. Cette communication devait principalement se faire à l'aide d'une connexion série par le biais d'un câble USB. Dans un second temps l'étude d'une transmission de données par Wifi fût étudiée.

Pour répondre à ces deux missions, j'ai dû m'aider des connaissances acquises durant ces 2 années de BTS ainsi que celles de mes professeurs et de mes camarades de classe. J'ai pu travailler sur différents logiciels de programmations comme Visual Studio, Arduino mais aussi sur un logiciel de CAO (conception assistée par ordinateur) de circuit imprimé, Eagle.

I) Etude et conception

1) Cahier des charges



1) Solutions retenues

Pour répondre correctement au cahier des charges, j'ai divisé mes tâches en deux parties, une pour la communication électronique, une autre pour la conception et la création. Nous pouvons donc distinguer les solutions retenues en deux catégories. D'un côté les solutions matérielles pour réaliser physiquement une communication, et de l'autre côté les solutions logicielles qui permettent de répondre aux critères de conception et de réalisation du cahier des charges :

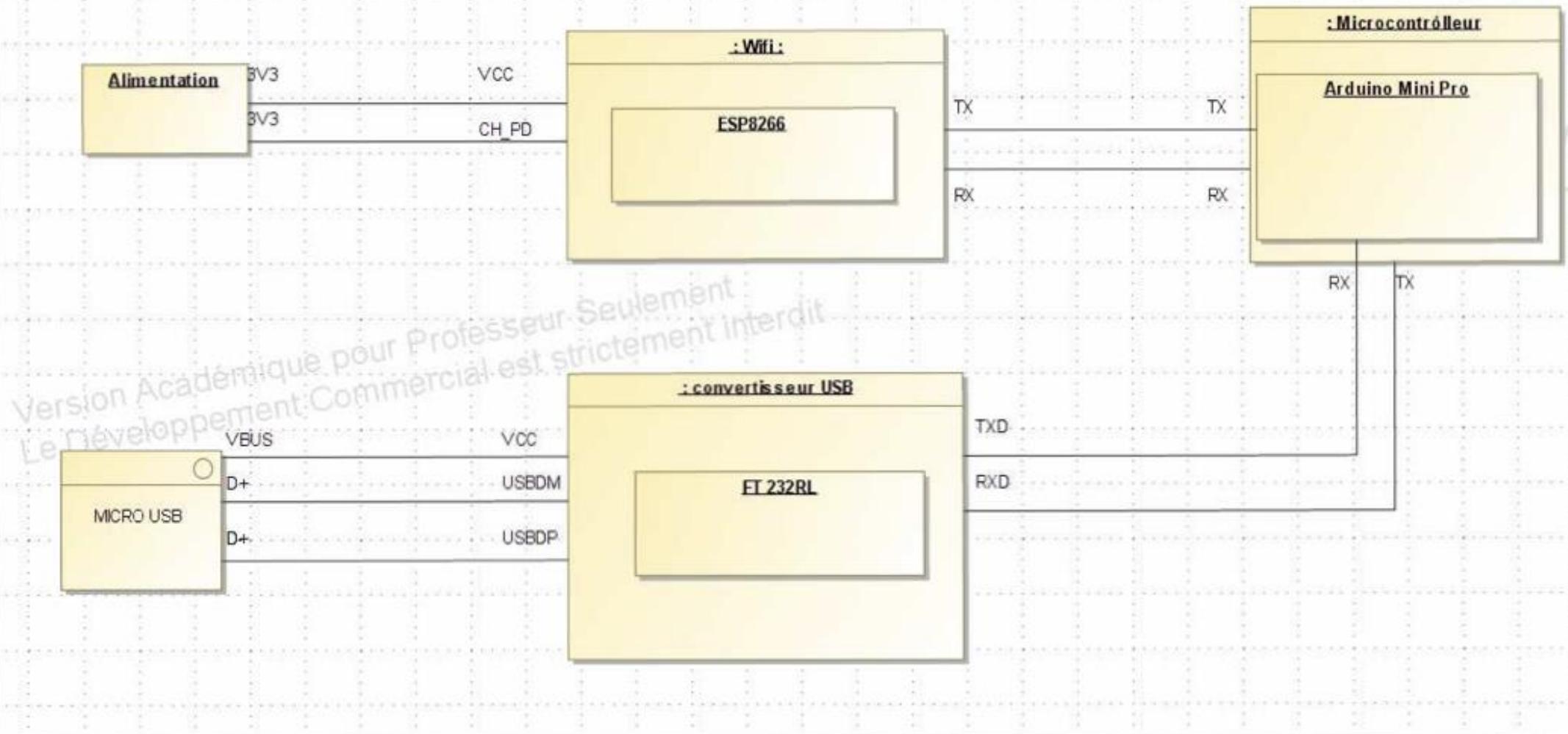
- Pour créer une communication filaire entre le microcontrôleur Arduino Mini Pro et l'IHM présente sur le PC, il a été nécessaire d'utiliser un composant permettant de convertir une interface série USB en une interface série UART. En effet l'Arduino Mini Pro ne gère pas le protocole série USB mais uniquement la transmission asynchrone universelle (UART). Pour réaliser cette conversion, j'ai utilisé le FTRL232 qui par sa facilité d'utilisation correspondait à notre projet.

Pour créer une communication en Wifi entre l'Arduino Mini Pro et un serveur, nous avons retenu le choix du composant ESP8266 qui utilise et convertit une interface UART du microcontrôleur de l'Arduino en signal Wifi.

- La solution logicielle retenue pour créer l'Interface Homme Machine sur le PC a été le choix de Visual Studio. Cette suite logicielle complète permet le développement d'application web, bureautique et mobiles.

J'ai aussi travaillé sur le logiciel Eagle pour répondre au besoin de la conception du circuit électronique imprimé et de réaliser les empreintes de mes composants FTRL232 et ESP8266 ainsi que celle de l'Arduino Mini Pro.

2) Diagramme de bloc interne



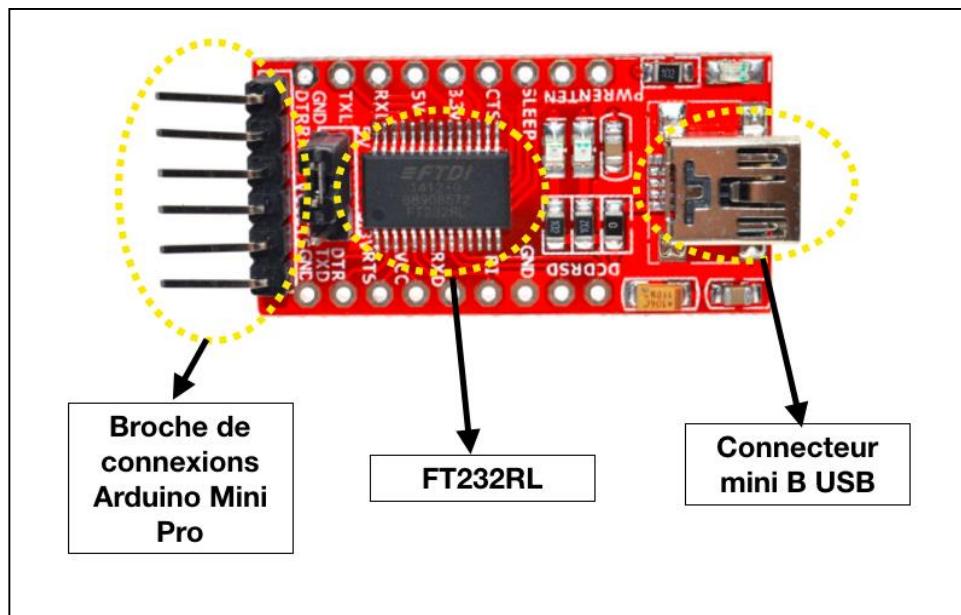
3) Communication USB

Pour que le bloc maître puisse communiquer avec l'IHM par le biais d'un câble USB, il fut nécessaire de convertir le signal en sortie du microcontrôleur du bloc maître. En effet le microcontrôleur du Bloc Maître est un Arduino Pro Mini qui est capable de transmettre et de recevoir des données en liaison série UART Rx/Tx. En revanche l'ordinateur contenant l'IHM transmet et reçoit des données avec le protocole série USB.

Comme les protocoles de communication du PC et du Bloc Maitre sont différents, il nous fallait trouver un moyen de les adapter et de trouver une solution permettant de réaliser une conversion de ces deux protocoles.

Le FT232RL de FTDI répondait à ce besoin et permet donc d'adapter une interface série USB vers UART.

Pour simplifier la conception et la réalisation de la carte électronique du bloc maître, nous avons fait l'acquisition d'un module qui intègre le FTRL232 et qui est prêt à être connecté d'un côté à l'Arduino Mini Pro et de l'autre au connecteur USB :

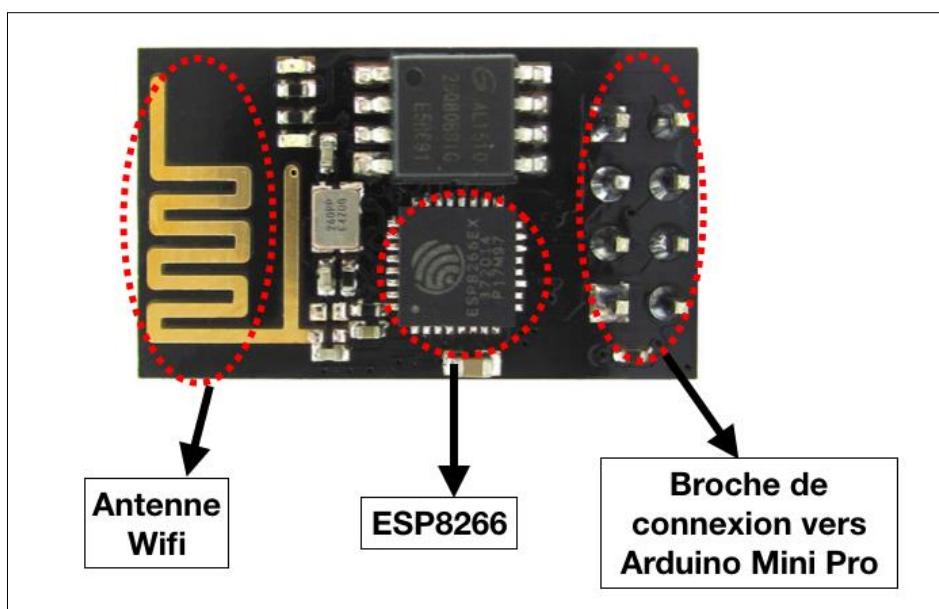


4) Communication Wifi

A la demande de la classe de BTS SN option Informatique et réseaux, le bloc maître devait pouvoir communiquer des informations à un serveur en wifi. Ma tâche personnelle sur cette partie n'était pas de communiquer directement des informations à un serveur mais de configurer, paramétriser et tester cette communication.

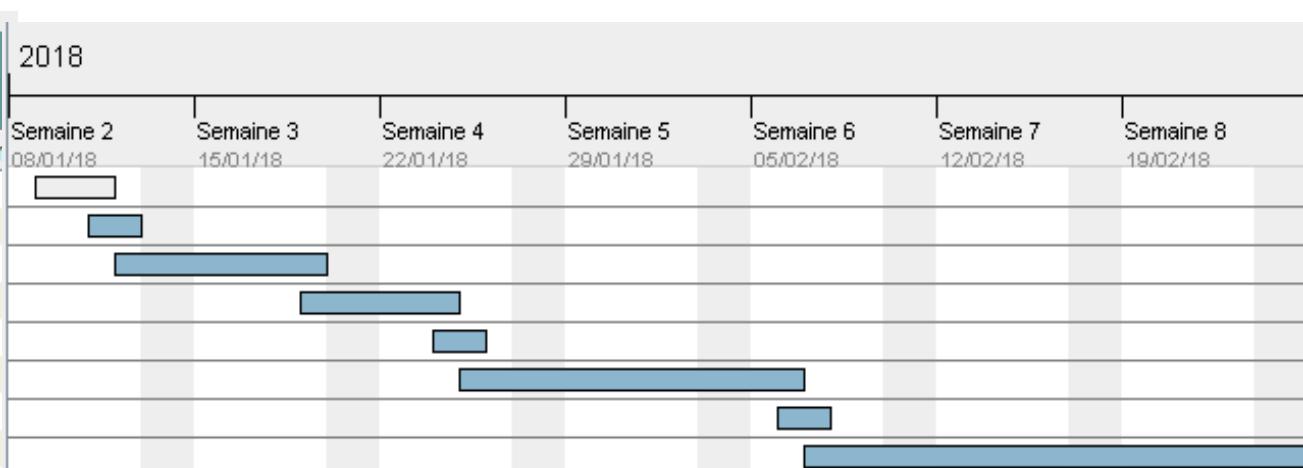
Pour cela j'ai dû configurer un module wifi qui, à partir de données provenant d'une liaison UART série de l'Arduino Pro Mini, génère un signal wifi avec les données à transmettre.

Le module ESP8266-01 correspondait à ce cahier des charges car il est facilement configurable et compatible avec notre microcontrôleur :

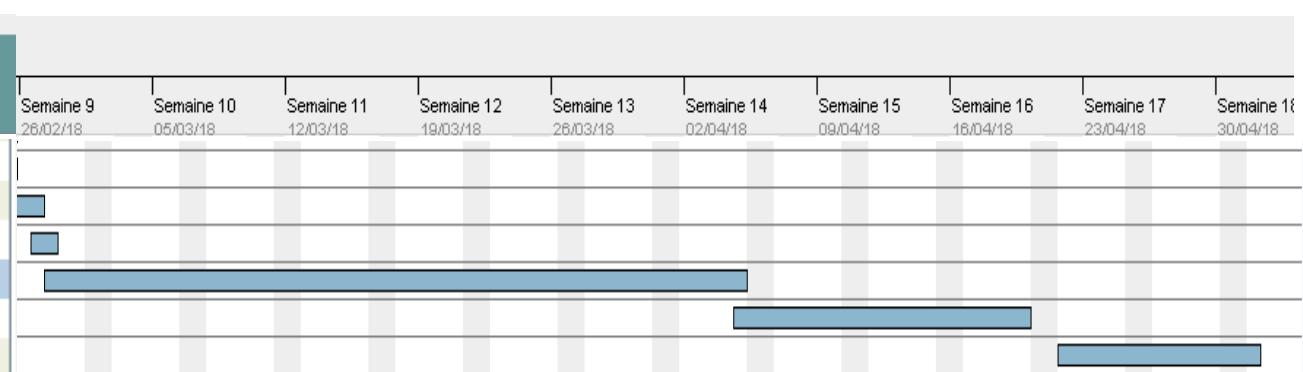


5) Planification

Nom	Date de début	Date de fin
• Répartition des tâches	09/01/18	11/01/18
• Cahier des charges	11/01/18	12/01/18
• Etude composants	12/01/18	19/01/18
• Réalisation IBD détaillée	19/01/18	24/01/18
• Début schématisation carte	24/01/18	25/01/18
• Création librairies EAGLE	25/01/18	06/02/18
• Revue projet 1	06/02/18	07/02/18
• Mis en commun schéma structurel	07/02/18	27/02/18



• Routage EAGLE	27/02/18	01/03/18
• Etude IHM	01/03/18	06/03/18
• Revue projet 2	06/03/18	07/03/18
• Création IHM (Visual Studio)	07/03/18	27/04/18
• Communication WIFI	27/04/18	20/05/18
• Finalisation du projet	21/05/18	04/06/18



6) Conception des composants assistée par ordinateur

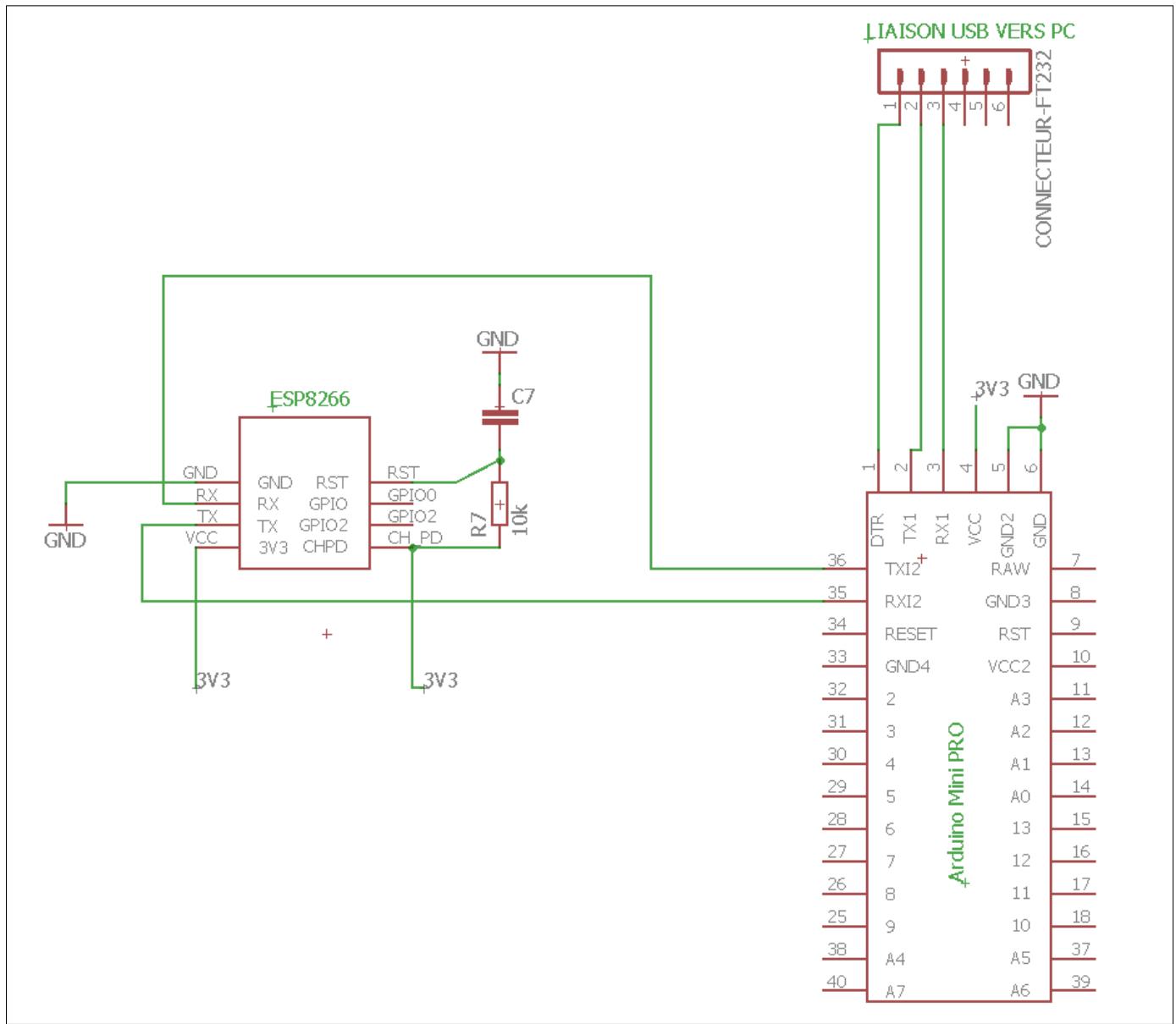
C'est le logiciel de CAO EAGLE qui nous a permis de réaliser toute la partie conception et modélisation des composants. Tout ce travail de conception avait un objectif final, il devait nous permettre de réaliser les schémas structurels électroniques et le routage de notre circuit imprimé.

Pour nous, les trois étudiants affectés au Bloc Maitre du projet, nous devions travailler à l'élaboration d'une seule carte électronique commune. Sur cette carte chacun se devait d'intégrer les composants nécessaires à sa partie.

Pour rappel : L'élément central de la carte est le microcontrôleur Arduino Mini Pro d'où sont reliés les autres composants.

Voici les tâches que j'ai réalisé sur le logiciel EAGLE :

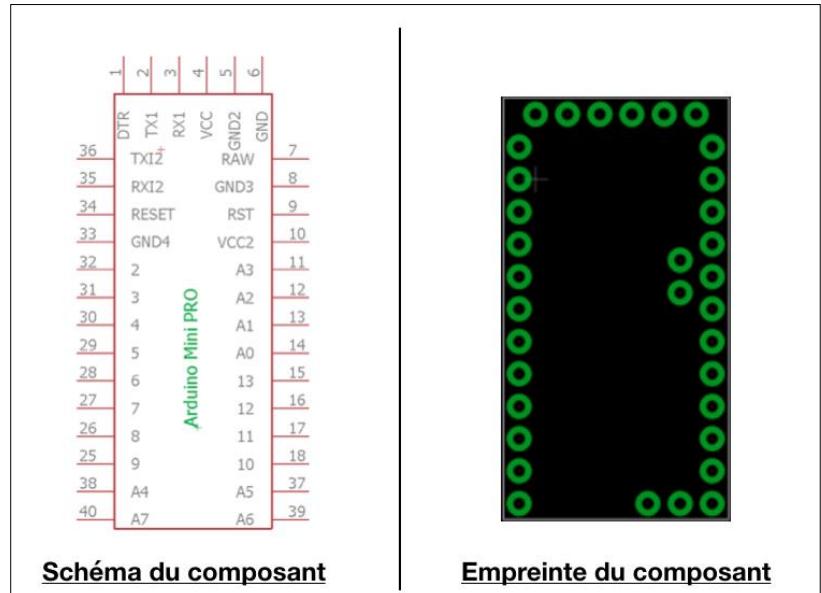
- Librairie du microcontrôleur Arduino Pro Mini.
- Librairie du connecteur FT232RL
- Librairie du module Wifi (ESP8266)
- Schéma structurel électronique local à ma partie.



Explications de la réalisation des librairies :

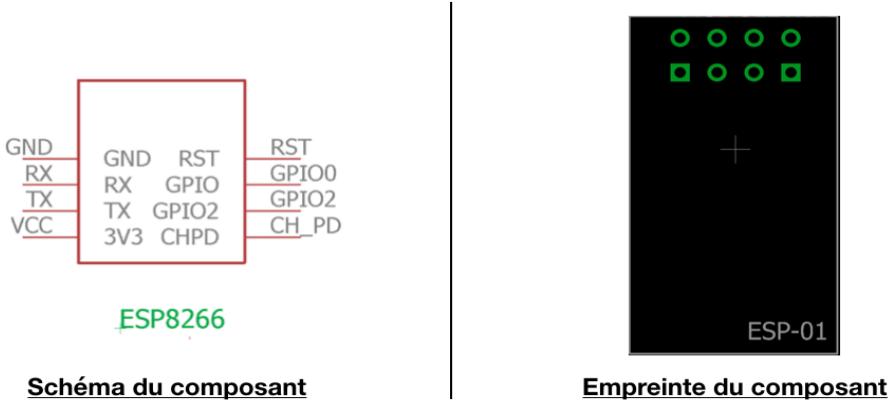
Sur EAGLE, la réalisation des librairies de composants permet de répondre à deux aspects de la conception. Dans un premier temps la librairie d'un composant contient un symbole de ce dernier qui est utilisé dans la partie schématique de EAGLE.. Dès lors que l'on ajoute un composant dans la partie schématique, l'empreinte de ce composant s'ajoute automatiquement dans l'environnement de routage

Pour réaliser une librairie spécifique à un composant il faut donc réaliser une représentation schématique du composant (avec les différentes entrées/sorties) et une empreinte de ce composant (avec les bonnes dimensions, les bonnes cotes etc ..).

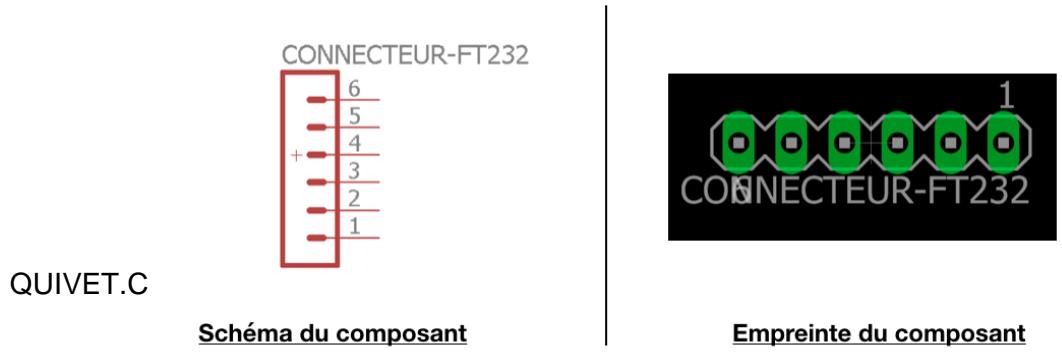


J'ai commencé par créer la librairie de notre Arduino Pro Mini :

Ensuite j'ai commencé la réalisation des librairies nécessaires à la représentation de mes composants. Voici la librairie EAGLE de l'ESP8266 :

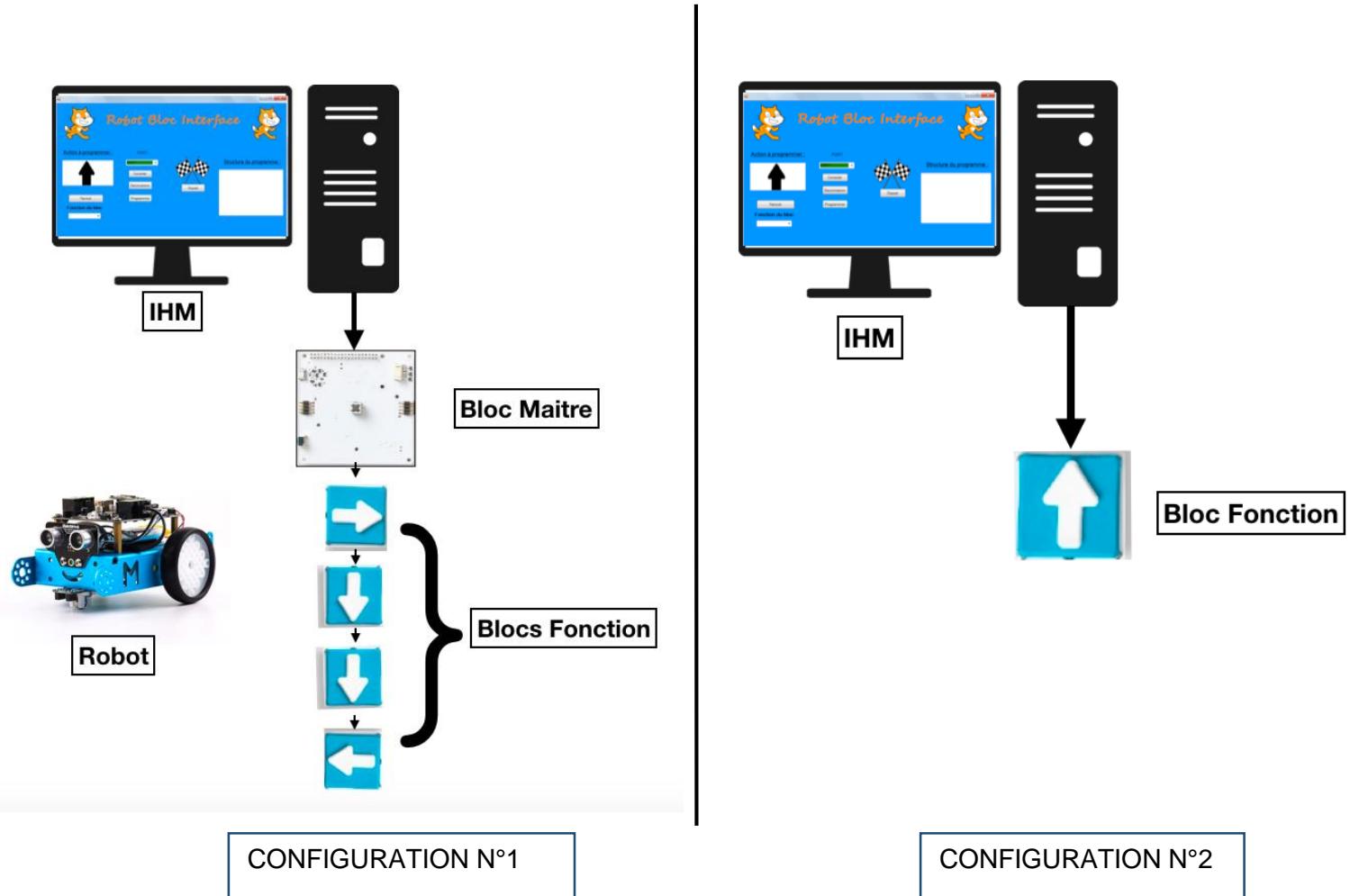


Enfin j'ai réalisé la librairie du FTRL-232, l'implantation de ce composant est différente des autres car on ne fixe pas le module FTRL232 sur la carte à l'horizontal mais à la vertical. Pour ce faire, l'empreinte correspondante à réaliser est donc un simple connecteur 6 broches d'où l'on viendra planter le module à la vertical :



II) L'interface homme machine

1) Présentation



L'interface Homme Machine réalisée peut être utilisée dans les deux configurations ci-dessus :

- Dans la configuration n°1, L'IHM communique avec le bloc maître pour effectuer des actions de contrôles et de visualisation sur le programme construit par les blocs fonctions.
- Dans la configuration n°2, L'IHM communique uniquement avec un seul bloc fonction pour lui affecter une action temporaire.

2) Objectifs

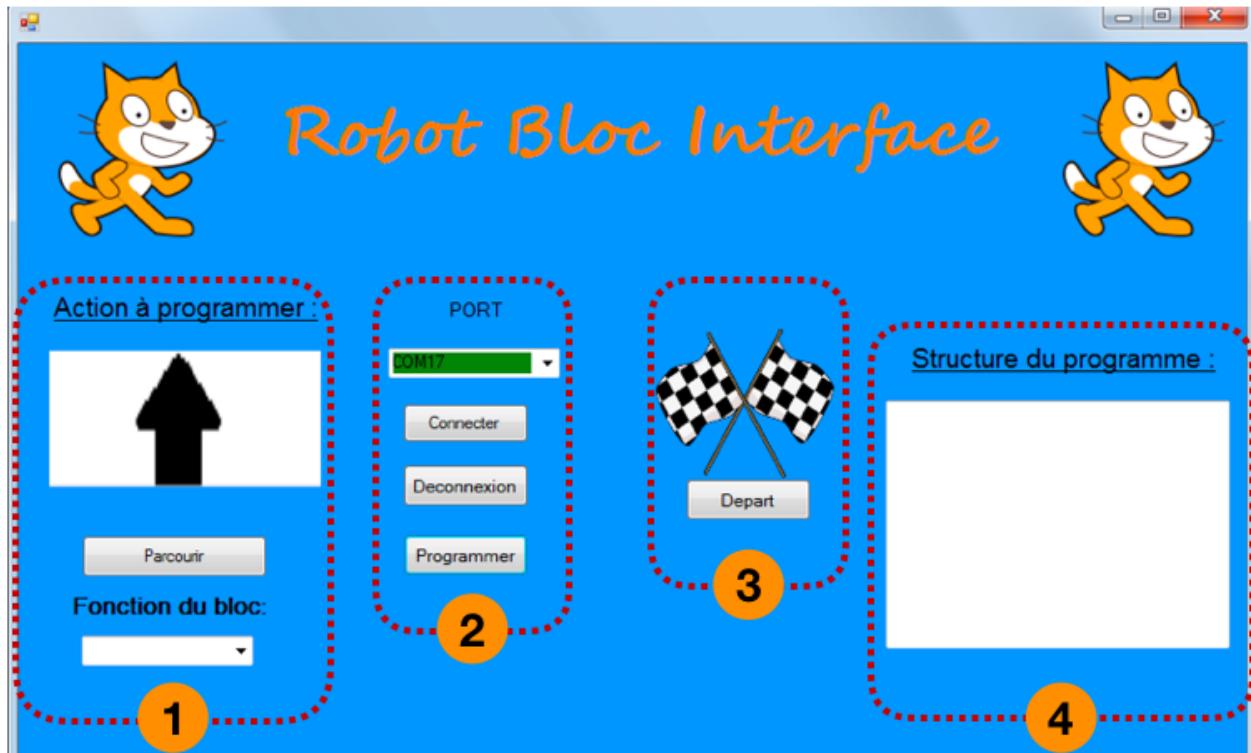
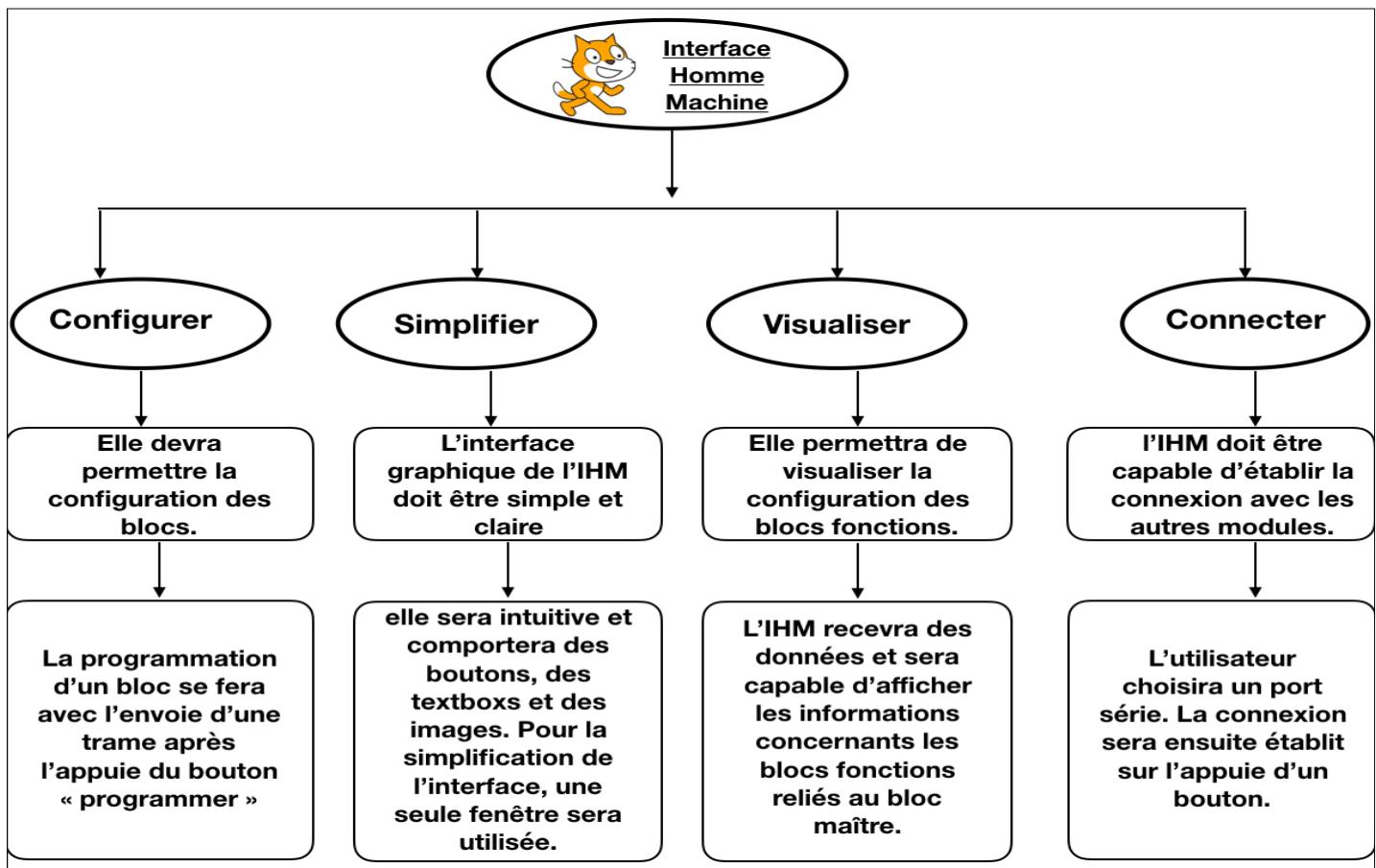
Le choix de réaliser une IHM répondait à plusieurs objectifs du projet :

- Le premier est de pouvoir répondre au critère modulable des blocs fonctions. En effet chaque bloc fonction ne doit pas être contraint à réaliser une action fixe comme avancer, reculer, tourner à droite etc ... Les blocs fonctions doivent être configurés depuis l'IHM et cette dernière affectera une action temporaire à chaque bloc fonction.

Par exemple un bloc fonction peut être configuré comme assurant la fonction « avancer » pendant un certain temps, puis reprogrammer dans une autre fonction plus tard suivant le besoin. C'est grâce à l'interface de l'IHM que ce changement d'attribution est possible.

- Dans un second temps l'IHM devait permettre l'affichage visuel des actions programmées sur les blocs fonctions. Notre projet étant en lien avec la classe de BTS SNIR, l'affichage des actions devait être complètement personnalisable. C'est-à-dire que l'on importe une image au format Bitmap depuis l'IHM, que l'on transmet ensuite sur le bloc fonction, qui affiche cette dernière.
- Le troisième objectif de l'Interface et de pouvoir recevoir les données du bloc maître concernant l'agencement du programme construit et d'afficher ce dernier (afficher la séquence construite par tous les blocs fonctions qui forment une succession d'actions, par exemple «avancer »-« tourner à droite »-« reculer » etc ...).
- Enfin le dernier objectif et de pouvoir lancer le départ du robot qui exécute le programme construit par les blocs fonctions (un bouton physique réalise déjà cette action sur le bloc maître). Le système de lancement du départ est donc présent physiquement et virtuellement.

3) Cahier des charges



- 1 Configuration d'un bloc fonction**
- 3 Départ du robot**
- 2 Connexion à un port série COM**
- 4 Réception du programme assemblé par le bloc maître**

5) Présentation de l'environnement de développement

Pour répondre au cahier des charges de l'interface homme machine, j'ai utilisé la solution de développement Visual Studio. Ce logiciel de développement permet de créer une application exécutable sur n'importe quel PC. Le langage de programmation utilisé est le C# utilisant l'environnement .NET de Microsoft.

Le développement d'une application est divisé en deux parties. Une partie programmation et une partie interface graphique. Le langage de programmation orienté objets C# permet la simplification de la création de la partie graphique. En effet chaque objet visuel (comme un bouton, un champ de texte etc ..) est automatiquement déclaré dans la partie code.

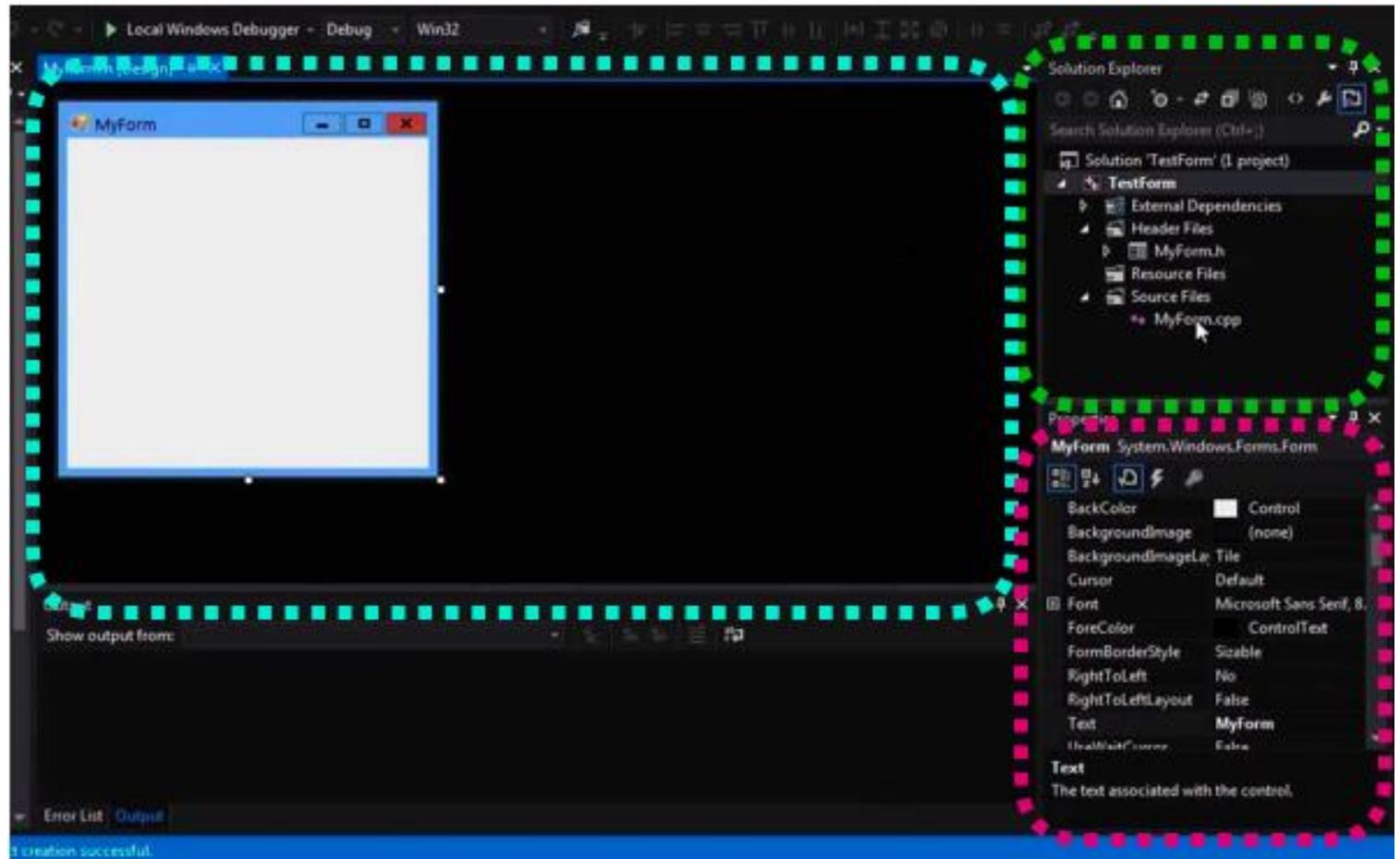
La partie programmation consiste à gérer chaque évènement de la partie graphique et d'attribuer une succession d'actions pour un événement donné. (Par exemple : lors d'un appuie sur un bouton, on envoie une trame de données, puis on affiche une image etc..). Grâce au framework .NET de nombreuses bibliothèques contenant de nombreuses classes sont disponibles et simplifient grandement la programmation des évènements.

Présentation du C# :

- Récent : il a été créé par Microsoft au début des années 2000, ce qui en fait un langage assez jeune (comparé au C, au C++ et à Java).
- Idéal sous Windows & Windows Phone: c'est le langage recommandé pour développer sous Windows aujourd'hui, mais on s'en sert aussi pour créer des applications Windows Phone 7, pour Silverlight, ASP...
- Libre : le langage est ouvert, et on peut s'en servir pour développer des applications sous Linux notamment.
- Inspiré de Java: même s'il a ses spécificités, il ressemble globalement plus à Java qu'au C ou au C++ contrairement à ce que son nom pourrait laisser penser.
- Associé au framework .NET: un langage seul comme le C# ne permet pas de faire grand-chose. On l'associe en général à une boîte à outils que l'on appelle le framework .NET.

Présentation de l'interface d'une application Windows Form (dans Visual Studio) :

L'interface contient trois fenêtres : une **fenêtre principale**, l'**Explorateur de solutions** et la **fenêtre Propriétés** :



Une application Windows Form repose sur la plateforme .NET. A la création d'une application Windows Form, on remarque dans l'explorateur de solution que des fichiers ont été créés :

- Program.cs qui est le point d'entrée de l'application (contient la fonction main).

- Form1.cs qui est le code de la fenêtre principale.

- Form1.Designer.cs qui est le code du designer: c'est le code qui va faire afficher la fenêtre (Form1) et ses composants.

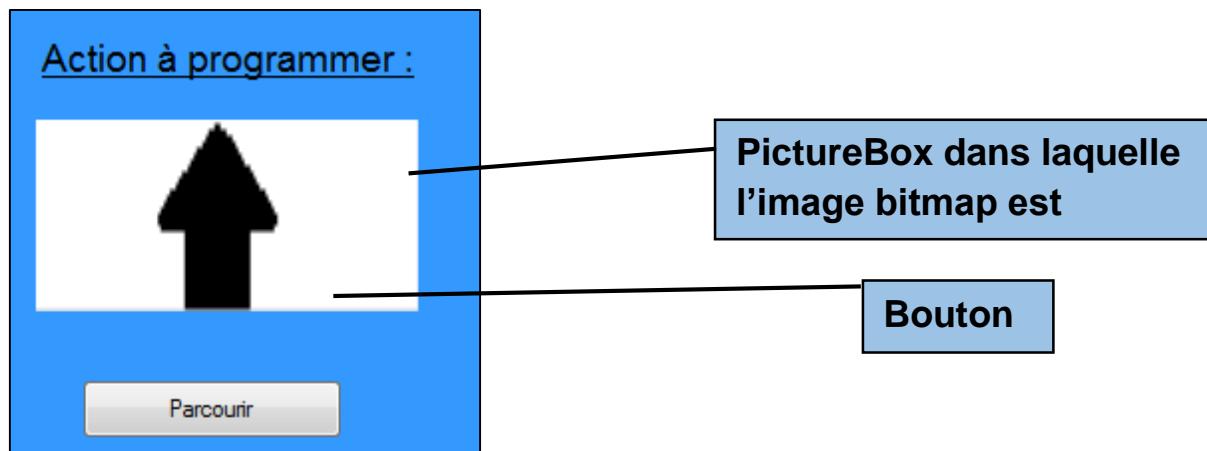
6) Explications et validations

Configuration d'un bloc fonction

Pour la configuration d'un bloc fonction l'utilisateur doit réaliser deux étapes. **La première** est d'importer une image au format Bitmap aux dimensions 128* 32 pixels de couleur monochrome.

Pour ce faire, deux éléments sont nécessaires sur l'interface graphique de l'IHM :

- Un bouton, qui lors d'un clic ouvre une fenêtre dans laquelle l'utilisateur sélectionne l'image avec les bonnes caractéristiques voulues.
- Une PictureBox qui permet de pré-visualiser l'image qui apparaîtra sur le bloc fonction à programmer.



Pour stocker une image dans mon programme, lors d'un clic sur le bouton Parcourir, la fonction « bouton_parcourir » ci-dessous est appelée :

```

private void bouton parcourir(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog1 = new OpenFileDialog();
    openFileDialog1.InitialDirectory = "U:\\\\";
    openFileDialog1.Filter = "BMP (*.bmp)";
    openFileDialog1.FilterIndex = 1;
    openFileDialog1.RestoreDirectory = true;

    if(openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        try
        {
            if ((imagefleche = openFileDialog1.OpenFile()) != null)
            {using (imagefleche)
            {

                imagefleche.Image = Image.FromStream(imagefleche);
                imagefleche.SizeMode = PictureBoxSizeMode.StretchImage;
                imagefleche.Position = 0;

                int nbbits_a_lire = (int)imagefleche.Length;
                int nbbits_lus = 0;
                do
                {
                    int n = imagefleche.Read(tableau_stream, nbbits_lus, 1);
                    nbbits_lus += n;
                    nbbits_a_lire -= n;
                } while (nbbits_a_lire > 0);

                imagefleche.Close();

                Console.WriteLine("number of bytes read: {0:d}", nbbits_lus)
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Impossible d'ouvrir le fichier choisi: " + ex.Message);
        }
    }
}

```

Cette fonction dans un premier temps ouvre une fenêtre de dialogue et affiche toutes les images au format Bitmap se trouvant sur le répertoire « U:// ». Si l'utilisateur choisit et valide l'une d'entre elle, l'image est alors stockée dans une variable de type Stream « imagefleche » (flux de données).

Dans un second temps, toutes les données qui forment cette image sont copiées et écrites dans un tableau de variable de type Bytes nommé « tableau_stream ». Une fois la copie de l'image terminée, le flux de données est libéré.

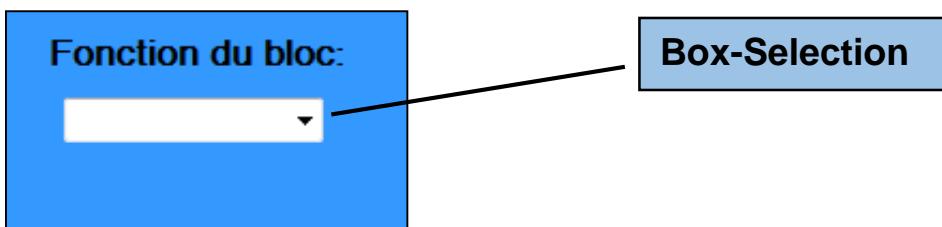
La deuxième partie de la configuration d'un bloc fonction est l'attribution d'une action temporaire au bloc.

Pour ce faire seulement un élément graphique est nécessaire :

- Une BoxSelection qui comporte les 4 actions du cahier des charges, à savoir avancer, reculer, rotation à droite et rotation à gauche.

Le projet étant en lien avec les BTS SNIR, des actions personnalisables doivent aussi pouvoir être configurées, comme tourner à 360 degrés par exemple.

Pour se faire l'utilisateur doit entrer directement l'identifiant de l'action souhaité (par exemple « a » dans le champ de texte).



Ci-dessous, le code répondant à ce cahier des charges :

```
String[] fonctions_blocs = {"avancer","reculer","droite","gauche"};
fonctions_box.Items.AddRange(fonctions_blocs);
String fonctionbloc;

fonctionbloc = fonctions_box.Text;

if (fonctions_box.Text == fonctions_blocs[0])
{
    fonctionbloc = "a";
}
if (fonctions_box.Text == fonctions_blocs[1])
{
    fonctionbloc = "b";
}
if (fonctions_box.Text == fonctions_blocs[2])
{
    fonctionbloc = "d";
}
if (fonctions_box.Text == fonctions_blocs[3])
{
    fonctionbloc = "g";
```

}

La variable de type String « fonctionbloc » contient l'identifiant de l'action choisit par l'utilisateur. Si l'utilisateur entre une action différente (comme le cas des BTS SNIR évoqué précédemment), l'identifiant correspond au texte rentré dans la BoxSelection.

A noter :

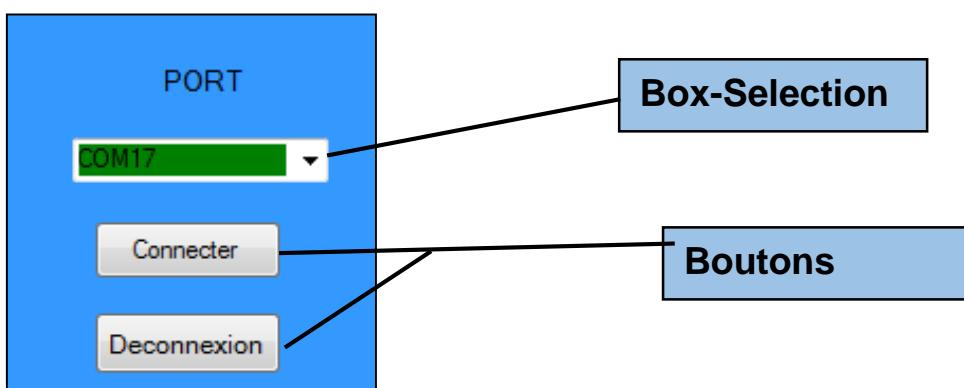
Les identifiants correspondants aux actions ont été choisi en accord avec Yoan GIANINO :

Action	Identifiant
Avancer	a
Reculer	b
Tourner à droite	d
Tourner à gauche	g
Action Personnalisée	Valeur entrée dans la BoxSelection

Etablissement de la connexion à un port COM :

Avant de pouvoir configurer un bloc ou encore de communiquer avec le bloc maître, la première fonction que doit réaliser l'IHM est de se connecter au bon port COM série correspondant. Pour cela il fut nécessaire de consacrer une petite partie graphique de mon interface pour gérer cette connexion.

J'ai utilisé deux boutons et une Box-Selection. La Box-Selection permet à l'utilisateur de visualiser tous les ports COM disponibles et d'en sélectionner un. Pour ce qui concerne les boutons, un est affecté à la connexion et un autre à la déconnexion du port COM choisi. On notera que lors ce que la connexion au port COM est réalisée avec succès, le fond de la Box-Selection devient vert comme nous pouvons le voir ci-dessous :



Code correspondant :

```
private void boutonConexion(object sender, EventArgs e)
{
    try
    {
        serialPort1.PortName = Boxport.Text;
        serialPort1.BaudRate = 1200;

        serialPort1.Open();

        if (serialPort1.IsOpen)
        {
            MessageBox.Show("Connexion réussie", "Port disponible");
            Boxport.BackColor = Color.Green;
        }
    }

    catch (Exception err)
    {
        MessageBox.Show(err.Message, "Echec !");
    }
}

private void boutonDeconnexion_Click(object sender, EventArgs e)
{
    serialPort1.Close();
    Boxport.BackColor = Color.White;
    AffichageReception.Clear();
}
```

Fonction du bouton de Connexion :

- On récupère le nom du port COM choisi par l'utilisateur
- On initialise la vitesse de transmission des données (ici 1200 Bauds).
- On ouvre le port COM serialPort1

Fonction du bouton de Déconnexion :

- On ferme le port COM
- On réinitialise la couleur du fond de BoxPort qui correspond à la BoxSelection

Programmation d'un bloc fonction

La programmation d'un bloc se fait par l'appui sur le bouton « Programmer ». Elle consiste à l'envoie d'une trame sur le port COM connecté vue précédemment. La trame envoyée est divisée en deux parties importantes. La première consiste à l'émission de l'identifiant choisi pour le bloc fonction contenu dans la variable «fonctionbloc». La deuxième consiste à l'envoie de l'image choisi par l'utilisateur comme représentante de l'action à programmer. Le protocole de la trame à lui aussi été mis en place avec Yoan GIANINO, il est le suivant :

Début Trame	Séparateur	Identifiant Action	Séparateur	1ière valeur image ... dernière valeur image
-------------	------------	--------------------	------------	--

Ce qui donne :

{« A » « ; » « fonctionbloc » « ; » « tableau_stream[62] » « ... » « tableau_stream[574] »
.....

Explication du code correspondant :

```
private void boutonprogrammer(object sender, EventArgs e)
{
    if (serialPort1.IsOpen)
    {
        serialPort1.WriteLine("A");
        serialPort1.WriteLine(";");
        serialPort1.WriteLine(fonctionbloc);
        serialPort1.WriteLine(";");
        serialPort1.WriteLine(tableau_stream, 62, 512);
    }
    else
    {
        MessageBox.Show("Aucun port connecté", "Erreur");
    }
}
```

Si le port COM est disponible on écrit le début de trame (« A ») puis le séparateur (« ; »), ensuite la valeur de la variable « fonctionbloc », de nouveau un séparateur (« ; ») puis on envoie la première valeur constituant l'image (« tableau_stream[62] ») jusqu'à la dernière valeur de l'image (« tableau_stream[574] ») soit 512 valeurs après.

A noter : Pourquoi la première valeur de l'image correspond à la 62 ième valeur du tableau contenant l'image ?

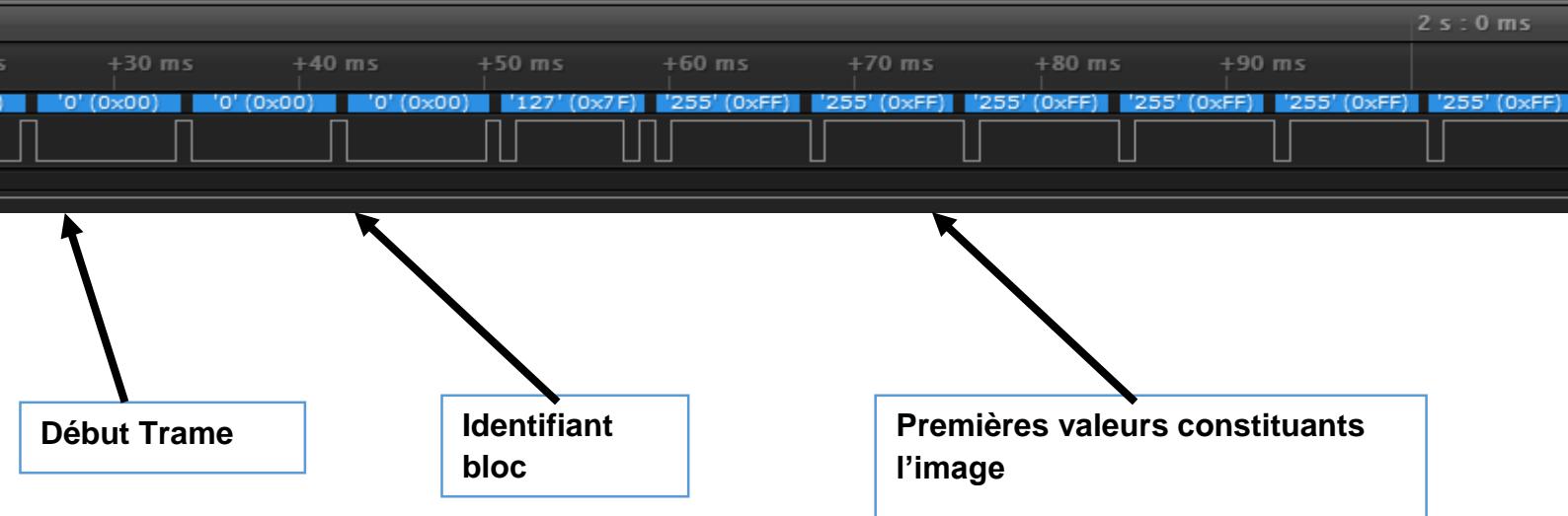
En effet le format Bitmap est constitué d'une en-tête de la manière suivante :

Entête de l'image														
	Largeur image			Hauteur image			Nbre plans			Nbre bits/pixel			Taille entête	
00:	42	4d	5a	00	00	00	00	00	00	36	00	00	28	00
10:	00	00	03	00	00	00	03	00	00	00	01	00	18	00
20:	00	00	24	00	00	00	00	00	00	00	00	00	00	00
30:	00	00	00	00	00	00	00	00	00	00	00	00	80	00
40:	00	00	00	ff	ff	00	00	00	c0	c0	c0	00	00	00
50:	ff	00	ff	00	80	80	80	00	00	00				
	Type compression			Résolution horizontale						Nbre couleurs palette				
	Taille image			Résolution verticale										

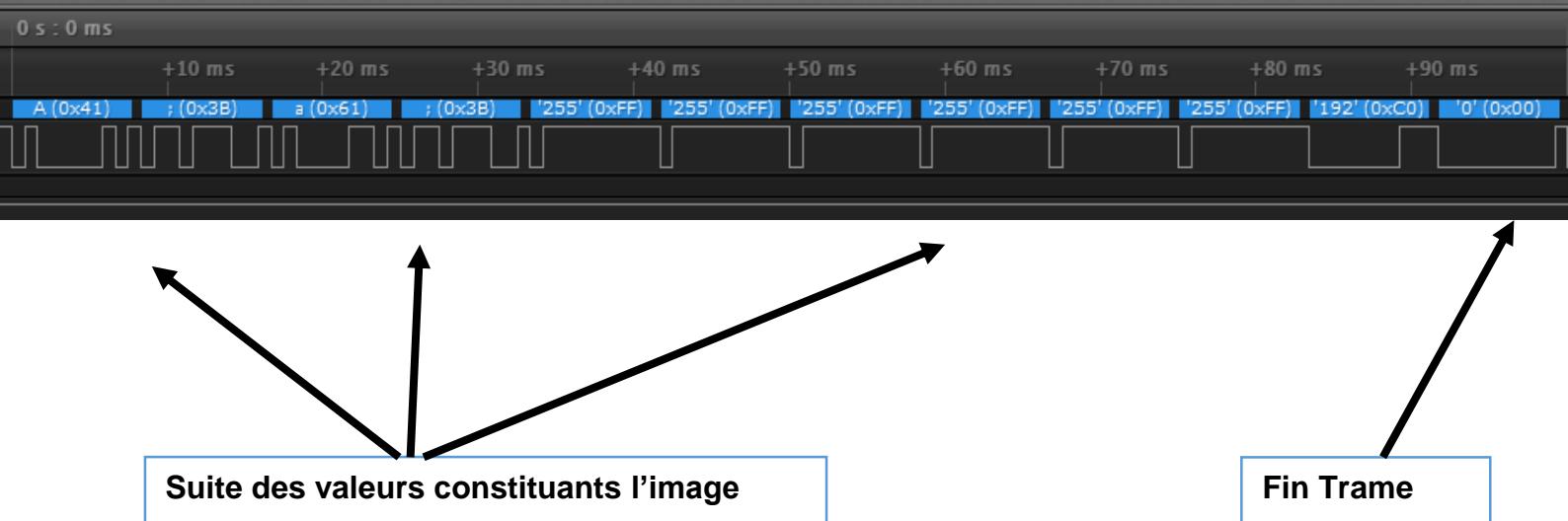
On remarque que cette en-tête prend une place de 62 octets. Dans notre cas cette en-tête nous est inutile et seules les données brutes de l'image nous concernent, c'est pourquoi la première valeur de l'image envoyé correspond à « tableau_stream[62] » .

Validation de l'envoie de la trame par la mesure :

Saleae Logic 1.2.15 – [Connected] – [24 MHz Digital, 2 s]



Saleae Logic 1.2.15 – [Connected] – [24 MHz Digital, 2 s]

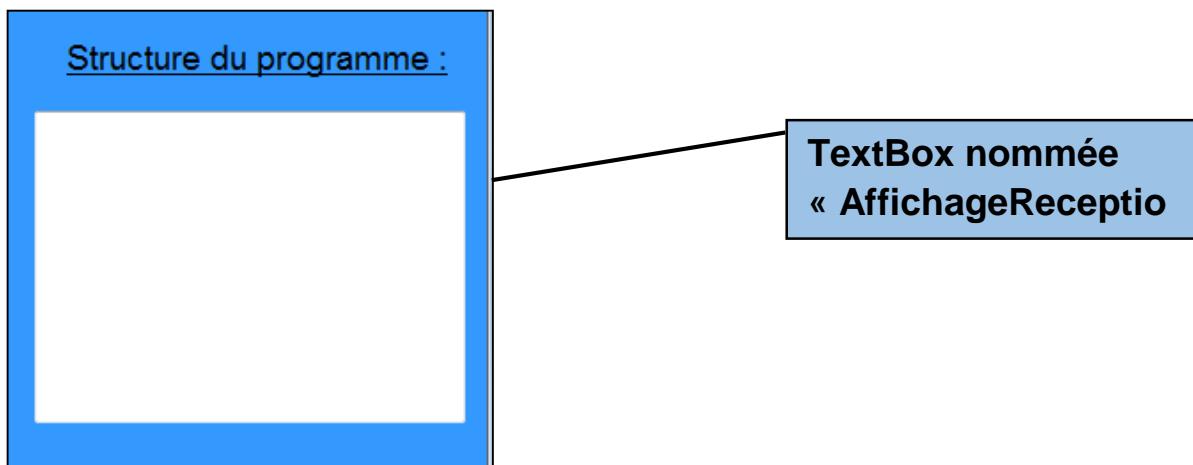


Réception du programme assemblé par le Bloc Maître

Pour valider l'agencement des blocs fonction assemblés, le bloc maître envoie la structure du programme finit à L'IHM. Cette dernière affiche la configuration des différents blocs fonctions à l'utilisateur qui peut ainsi vérifier virtuellement l'assemblage physique des blocs fonction.

Pour ce faire, un seul élément graphique est nécessaire :

- Une textBox qui affiche en lignes de textes, la configuration des blocs fonction assemblés.



Pour la première fois au court du projet, mon IHM devait recevoir une trame et afficher les informations reçues.

Le protocole de la trame reçu par l'IHM est le suivant :

ébut de trame ; Nombre de Blocs assemblés ; Fonction du Bloc 1 ; ... ; Fonction dernier Bloc ; Fin de trame

La fonction ci-dessous est appelée dès qu'un événement entrant sur le port COM est détecté, voici les explications de la fonction « serialPort1_DataReceived » :

```

private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    int longeurchaine,x;
    bool condition;
    string[] tab_blocs_fonctions = new string[30];
    .....
    do
    {
        tramerecu = serialPort1.ReadLine();
        longeurchaine = tramerecu.Length;
        condition = (tramerecu.Substring(0, 1) == "A");
    }
    while (!condition);
    .....
    nombreblocs = tramerecu.Substring(2,1);
    int nombreblocint = Int32.Parse(nombreblocs);
    x = 4;
    AffichageReception.Clear();
    AffichageReception.AppendText("Il y a " + nombreblocs + " blocs connectés : \r\n");
    AffichageReception.AppendText("\r\n");

    for (int i = 0; i < nombreblocint; i++)
    {
        tab_blocs_fonctions[i] = tramerecu.Substring(x, 1);
        x = x + 2;

        if (tab_blocs_fonctions[i] == "d")
        {
            AffichageReception.AppendText("Le bloc n° " + i + " est configuré en
fleche droite \r\n");
        }
        if (tab_blocs_fonctions[i] == "g")
        {
            AffichageReception.AppendText("Le bloc n° " + i + " est configuré en
fleche gauche \r\n");
        }
        if (tab_blocs_fonctions[i] == "a")
        {
            AffichageReception.AppendText("Le bloc n° " + i + " est configuré en
fleche bas \r\n");
        }
        if (tab_blocs_fonctions[i] == "b")
        {
            AffichageReception.AppendText("Le bloc n° " + i + " est configuré en
fleche haut \r\n");
        }
    }
}

```

1 : Déclaration des variables.

2 : Vérification du début de trame. Comme nous l'avons expliqué, la première valeur de la trame doit être un « A ». Tant que cette condition n'est pas validée, la suite de la fonction ne peut être aboutie.

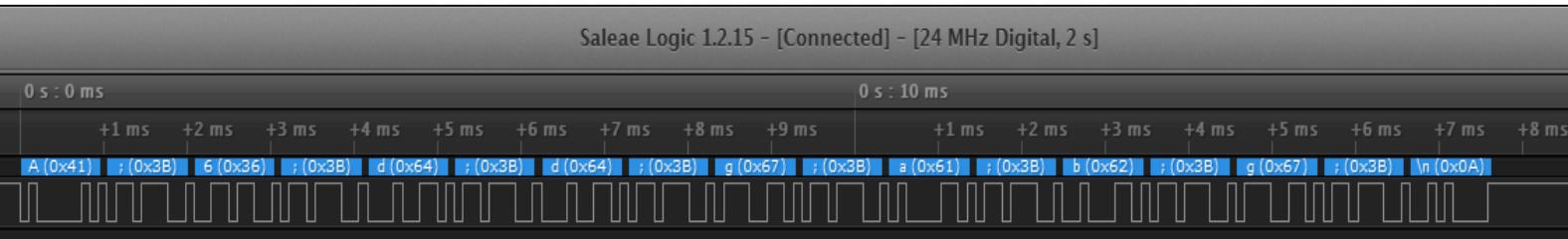
3 : Décomposition de la trame et Affichage. Premièrement on extrait le nombre de blocs assemblés que l'on affiche ensuite dans la TextBox. Puis on récupère chaque fonction des blocs que l'on affiche également dans la TextBox. Pour cela on effectue une succession de tests qui permettent de comparer les valeurs de la trame à notre référentiel («a » ou « d » ou « b » etc ...).

Validation de la réception de la trame :

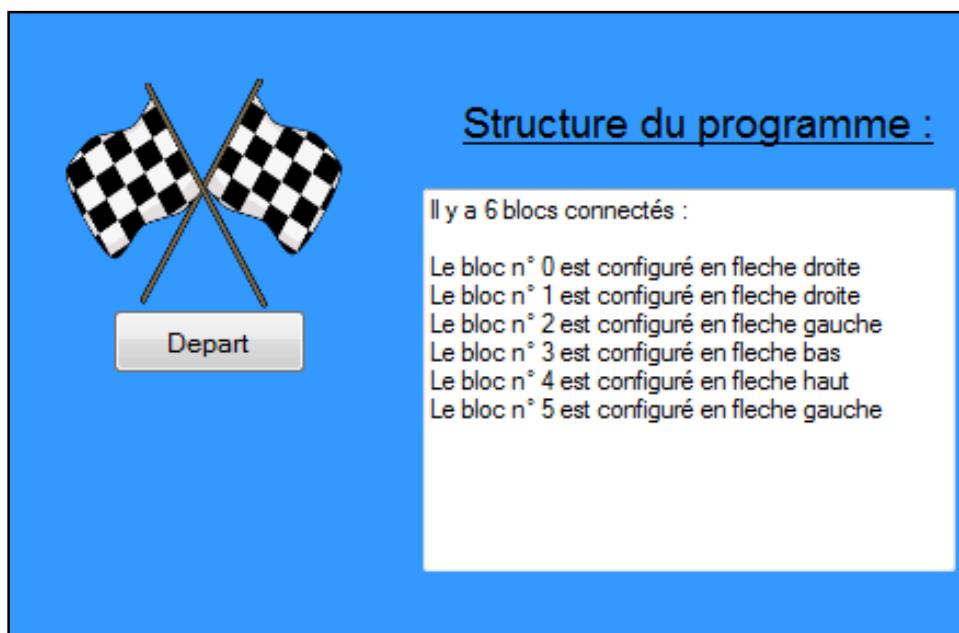
```
void emission()
{
    // transmissions sur le port série des données analogiques
    Serial.print('A'); // début de chaîne
    Serial.print(';'); // séparateur
    Serial.print('6'); // Nombre de blocs fonction
    Serial.print(';'); // séparateur
    Serial.print('d'); // fonction du bloc 1
    Serial.print(';'); // séparateur
    Serial.print('d'); // fonction du bloc 2
    Serial.print(';'); // séparateur
    Serial.print('g'); // fonction du bloc 3
    Serial.print(';'); // séparateur
    Serial.print('a'); // fonction du bloc 4
    Serial.print(';'); // séparateur
    Serial.print('b'); // fonction du bloc 5
    Serial.print(';'); // séparateur
    Serial.print('g'); // fonction du bloc 6
    Serial.print(';'); // séparateur
    Serial.print('\n'); // signal fin de trame
}
```

Pour cela j'ai réalisé un programme Arduino qui permet de simuler l'envoie d'une depuis l'Arduino Mini Pro présent sur le bloc maître. On y retrouve toutes les contraintes, le début de chaîne, les séparateurs, la fonction des blocs et le signal de fin de trame.

Avec l'analyseur logique, on remarque que la trame reçue semble correcte :



Sur l'IHM, on peut voir ci-dessous que la trame reçue est correctement interprétée et on retrouve bien la configuration simulée de tous les blocs fonctions connectés au bloc maître :



III) Configuration Wifi

La configuration de la connexion wifi s'est déroulée en plusieurs étapes :

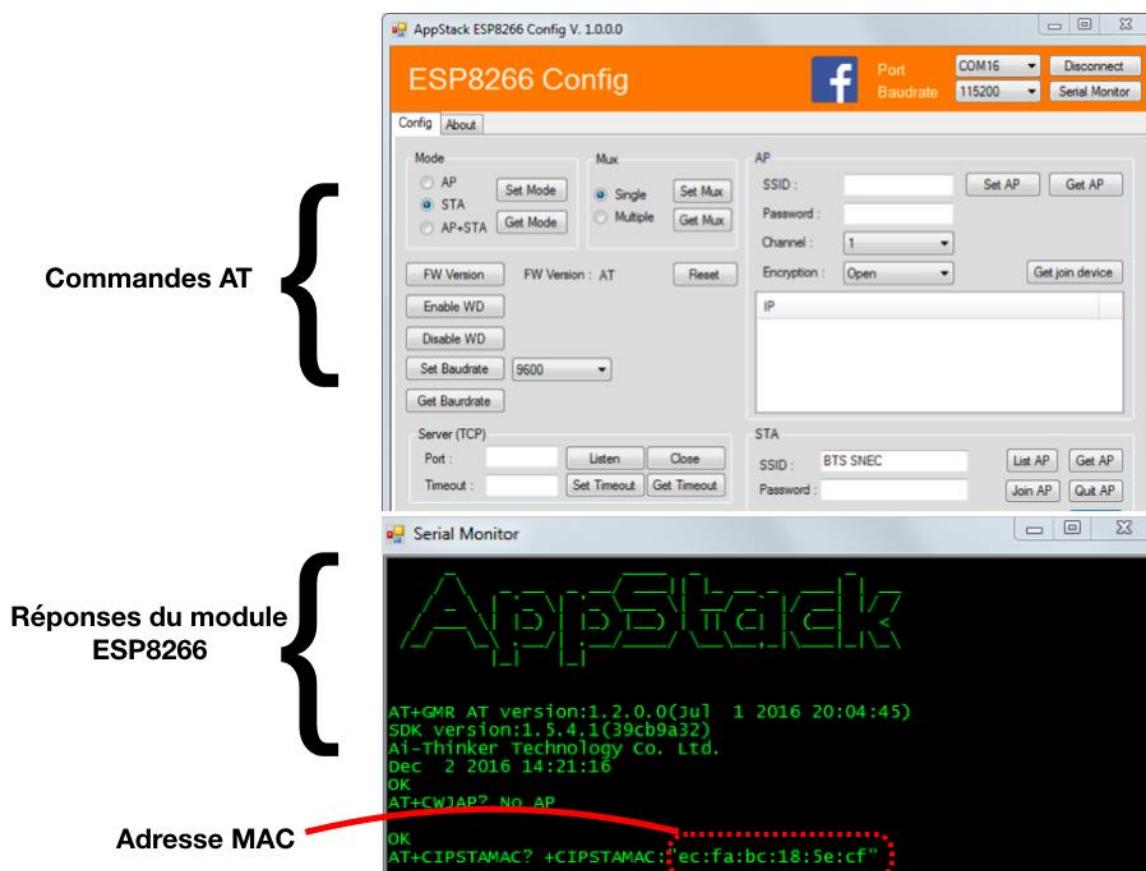
- 1) Configuration du module ESP8266
- 2) Configuration du serveur
- 3) Programmation de la communication Arduino/ESP8266
- 4) Validations

1) Configuration du module ESP8266

La configuration du module ESP8266 consiste à connecter le module à un routeur wifi. Dans notre cas il s'agit du routeur « BTS-SNEC » qui lui-même, est relié au réseau du lycée.

Pour se connecter au point d'accès wifi il est nécessaire de connaître son SSID (c'est à dire le nom du réseau sans fil, dans notre cas « BTS-SNEC ») et son mot de passe. Le routeur ne contient pas de mot de passe. En revanche il accepte ou non les appareils en fonction d'un filtrage MAC (adresse physique matériel). Pour que notre ESP-8266 soit reconnu par le routeur wifi, nous devons donc connaître son adresse MAC.

Pour se faire, j'ai utilisé le logiciel « Appstack ESP8266 Config » qui permet de communiquer avec le module ESP8266 grâce à un port série avec le langage de commande AT (voir annexes).



Une fois l'adresse MAC du module ESP8266 reconnu par le routeur Wifi, il suffit de se connecter à ce dernier (toujours à l'aide d' «Appstack ESP8266 Config ») :

```
AT+CWJAP="BTS_SNEC","" WIFI CONNECTED  
WIFI GOT IP  
OK  
AT+CIFSR +CIFSR:STAIP,"172.16.129.176"
```

2) Configuration du serveur

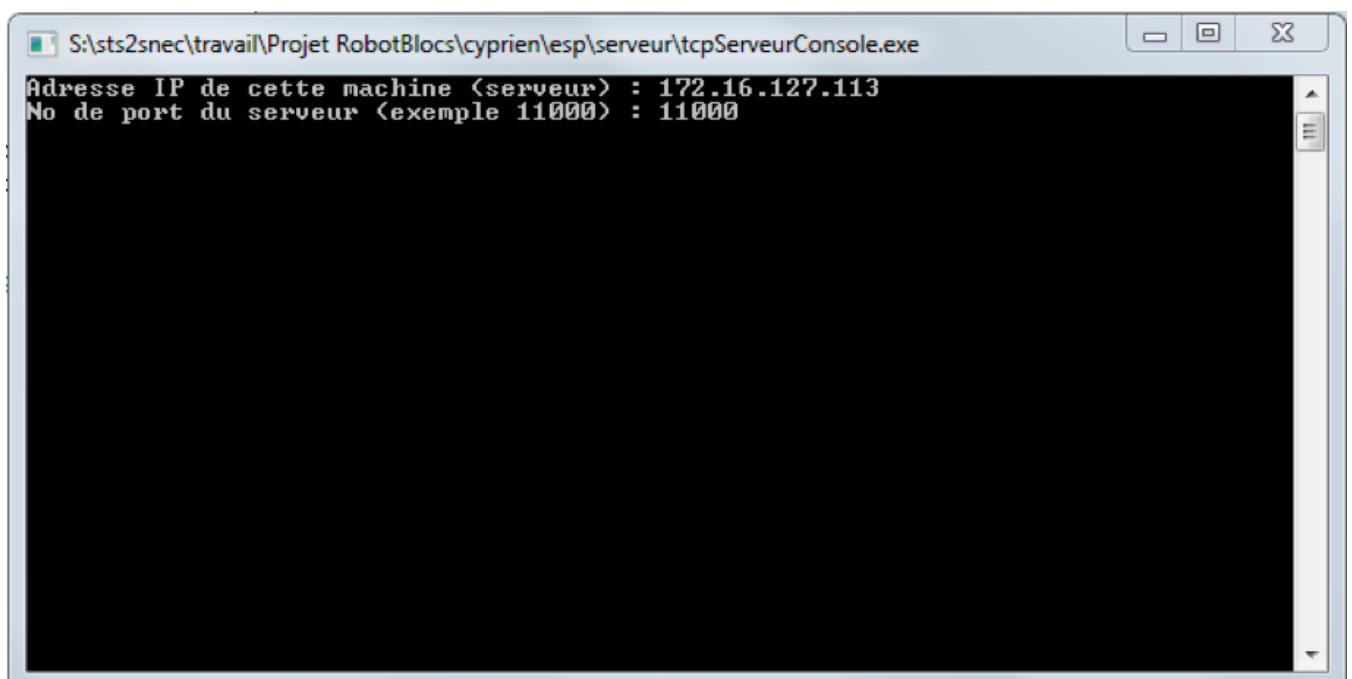
Les BTS SNIR ont créé un serveur pouvant s'exécuter sur n'importe quel poste informatique du réseau du lycée. Un environnement client-serveur est un mode de communication à travers un réseau informatique. Un équipement configuré en client envoie des requêtes. L'autre équipement configuré en serveur attend les requêtes du client, les traite et y répond.

Caractéristiques du serveur :

- Il attend une connexion entrante sur port du réseau local.
- à la connexion d'un client sur le port d'écoute, il établit une session TCP pour recevoir et expédier des données.
- A la suite de la connexion, le serveur communique avec le client.

Ce serveur permet donc de communiquer avec un client lui-même connecté au réseau informatique du lycée.

Par ma part, la configuration de ce serveur fut très simple. Il suffit de renseigner l'adresse IP de la machine où le serveur est exécuté ainsi que le port utilisé. Nous choisirons le port 11000 commun au serveur ainsi qu'à l'ESP8266 (client).



3) Programmation de la communication Arduino/ESP8266

Une fois les configurations du module et du serveur terminé, il faut programmer les instructions qui permettront au module ESP8266 de se connecter au serveur puis de lui envoyer une trame de données. Comme ces instructions viennent du microcontrôleur Arduino Mini Pro, j'ai dû créer un programme Arduino permettant de communiquer avec l'ESP8266.

En voici un extrait :

```
void setup()
{
    String message = "A;6;d;d;g;a;b;b\r\n"; // exemple de trame à envoyer au serveur.
    Serial.begin(9600);
    espSerial.begin(9600);

    Serial.println("Demarrage...");
    Serial.println();

    AT("AT+CWJAP=\"BTS SNEC\",\"\"");
    // connexion en Wifi au SSID du lycée (sans mot de passe).
    AttendConnect("OK");

    AT("AT+CIPSTART=\"TCP\",\"172.16.127.113\",11000");
    //connexion au serveur
    AttendConnect("OK");

    int nbCar = message.length(); // Détermination du nombres de caractères à envoyer
    AT("AT+CIPSEND="+String(nbCar)); // Envoie du nombre de caractères
    AttendConnect("OK");

    espSerial.print(message); //Envoie de la requête au serveur

    recMessage(); // réception de la réponse du serveur
    AT("AT+CIPCLOSE"); // déconnexion
    delay(5000);
}
```

Dans un premier temps on connecte l'ESP8266 au réseau Wifi du lycée nommé « BTS SNEC ». Une fois cette connexion établie, on connecte le module au serveur.

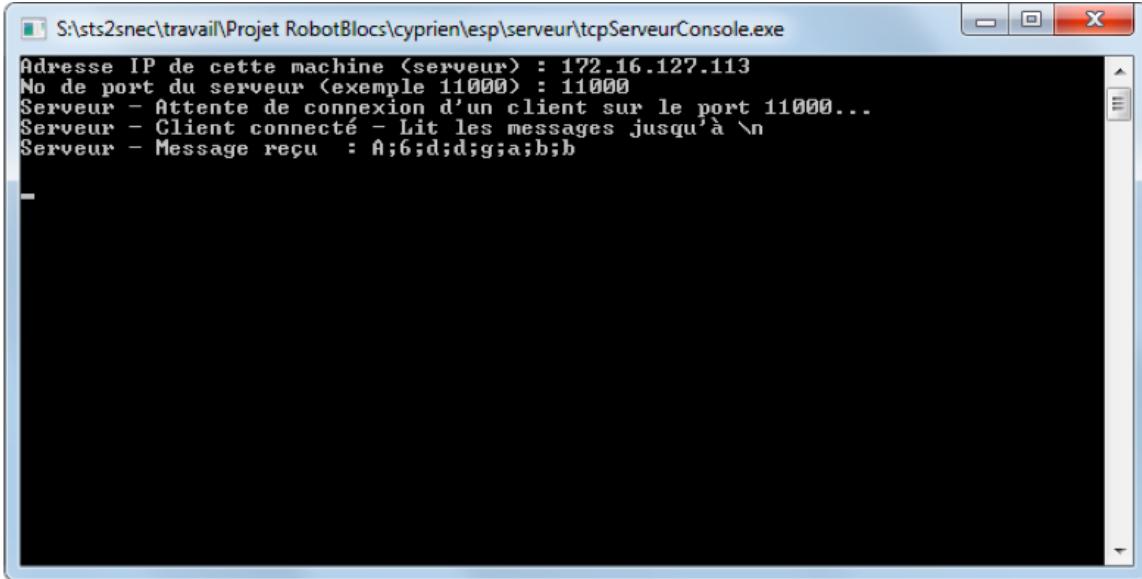
Dès lors que la connexion entre le module et le wifi est établie, il est possible de transmettre une requête. Dans le programme ci-dessous, la requête correspond à un exemple de trame que génère le bloc maître sur la configuration des blocs fonctions « A ;6 ;d ;d ;g ;a ;b ;b ».

Une fois la requête transmise au serveur, on déconnecte l'ESP8266.

A noter : Les lignes de codes « `AttendConnect("OK")` ; » , « `message.length()` ; » et « `recMessage()` ; » correspondent à des appels de fonctions.

4) Validations

Pour valider la communication entre l'ESP8266 et le serveur, on téléverse le programme expliqué ci-dessus et on regarde si le serveur reçoit le message envoyé :



Ci-dessus, on voit que le serveur reçoit bien le message envoyé par l'ESP8266. Aussi pour observer la transmission du message, j'ai utilisé le logiciel Wireshark qui permet d'analyser les paquets transmis au sein d'un réseau informatique :

A screenshot of the Wireshark application. The main pane shows a list of network frames, with the filter set to "ip.addr == 172.16.129.176". The details and bytes panes show the captured data. An arrow points from the bytes pane to a red-outlined section of the frame, highlighting the transmitted message "A;6;d;d;g;a;b;b".

No.	Time	Source	Destination	Protocol	Info
97613	123.664078	172.16.129.176	224.0.0.1	IGMP	V2 Membership Report / Join group 224.0.0.1
97695	124.720071	172.16.129.176	172.16.127.141	TCP	16332 > irisa [SYN] Seq=0 Win=2920 Len=0 MSS=1460
97698	124.722095	172.16.127.141	172.16.129.176	TCP	irisa > 16332 [SYN, ACK] Seq=0 Ack=1 win=8192 Len=0 MSS=1460
97699	124.729348	172.16.129.176	172.16.127.141	TCP	16332 > irisa [ACK] Seq=1 Ack=1 Win=2920 Len=0
97707	124.830748	172.16.129.176	172.16.127.141	TCP	16332 > irisa [PSH, ACK] Seq=1 Ack=1 Win=2920 Len=9
97733	125.033255	172.16.127.141	172.16.129.176	TCP	irisa > 16332 [ACK] Seq=10 Ack=10 Win=64240 Len=0
99023	136.326466	172.16.129.176	172.16.127.141	TCP	cvspserver > irisa [SYN] Seq=0 Win=2920 Len=0 MSS=1460
99024	136.326506	172.16.127.141	172.16.129.176	TCP	irisa > cvspserver [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
99025	136.328651	172.16.129.176	172.16.127.141	TCP	cvspserver > irisa [ACK] Seq=1 Ack=1 Win=2920 Len=0
99027	136.350098	172.16.129.176	224.0.0.1	IGMP	V2 Membership Report / Join group 224.0.0.1
99030	136.436125	172.16.129.176	172.16.127.141	TCP	cvspserver > irisa [PSH, ACK] Seq=1 Ack=1 Win=2920 Len=9
99038	136.632975	172.16.127.141	172.16.129.176	TCP	irisa > cvspserver [ACK] Seq=1 Ack=10 Win=64240 Len=0
101774	160.308484	172.16.127.141	172.16.129.176	TCP	irisa > 16332 [RST, ACK] Seq=1 Ack=10 Win=0 Len=0
101775	160.308508	172.16.127.141	172.16.129.176	TCP	irisa > cvspserver [RST, ACK] Seq=1 Ack=10 Win=0 Len=0
101776	160.308566	172.16.127.141	172.16.129.176	TCP	irisa > 36420 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
102070	165.417634	172.16.129.176	172.16.127.141	TCP	21068 > irisa [SYN] Seq=0 Win=2920 Len=0 MSS=1460
102114	166.247652	172.16.129.176	172.16.127.141	TCP	21068 > irisa [SYN] Seq=0 Win=2920 Len=0 MSS=1460
102173	167.247592	172.16.129.176	172.16.127.141	TCP	21068 > irisa [SYN] Seq=0 Win=2920 Len=0 MSS=1460
102230	168.251498	172.16.129.176	172.16.127.141	TCP	21068 > irisa [SYN] Seq=0 Win=2920 Len=0 MSS=1460
106911	209.850747	172.16.129.176	224.0.0.1	IGMP	V2 Membership Report / Join group 224.0.0.1
106976	210.889906	172.16.129.176	172.16.127.141	TCP	h2gf-w-2m > irisa [SYN] Seq=0 Win=2920 Len=0 MSS=1460
106979	210.891961	172.16.127.141	172.16.129.176	TCP	irisa > h2gf-w-2m [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
106980	210.894359	172.16.129.176	172.16.127.141	TCP	h2gf-w-2m > irisa [ACK] Seq=1 Ack=1 Win=2920 Len=0
106987	210.967788	172.16.129.176	172.16.127.141	TCP	h2gf-w-2m > irisa [PSH, ACK] Seq=1 Ack=1 Win=2920 Len=9
106988	210.968115	172.16.127.141	172.16.129.176	TCP	irisa > h2gf-w-2m [PSH, ACK] Seq=1 Ack=10 Win=64240 Len=9
106998	211.069103	172.16.129.176	172.16.127.141	TCP	h2gf-w-2m > irisa [ACK] Seq=10 Ack=10 Win=2911 Len=0

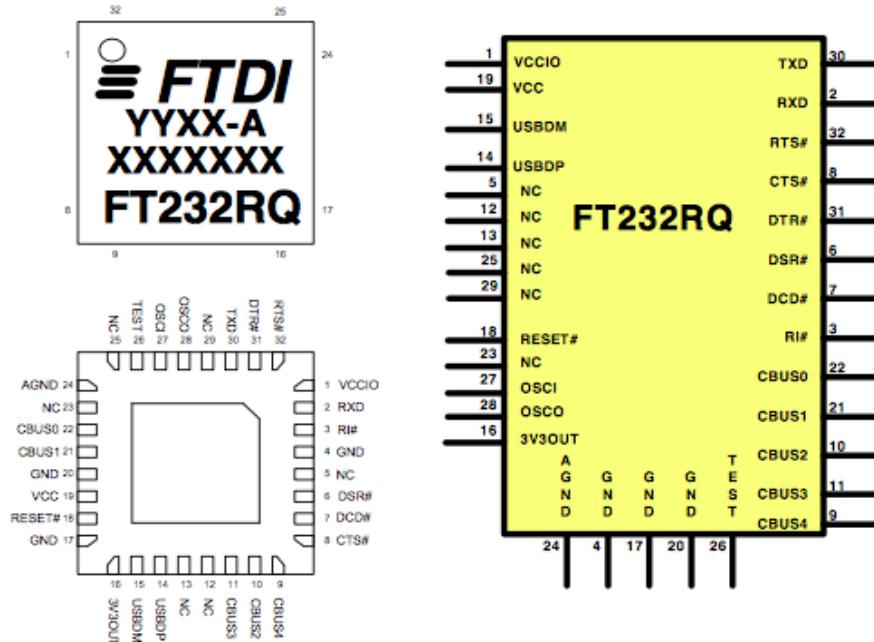
Frame 97707 (63 bytes on wire, 63 bytes captured)
0000 34 64 a9 18 b4 ce ec fa bc 18 5e cf 08 00 45 00 4d.....^.E.
0010 00 31 00 19 00 00 80 06 e1 4f ac 10 81 b0 ac 10 .1.....0.....
0020 7f 8d 3f cc 2a f8 00 00 22 ae bc 6c e4 22 50 18 ..?...."1.P.
0030 0b 68 81 c0 00 00 61 62 63 64 65 66 67 0d 0a .h.....A;6;d;d;g;a;b;b

4d.....^.E.
.1.....0.....
?.*...."1.P.
.h.....A;6;d;d;g;a;b;b

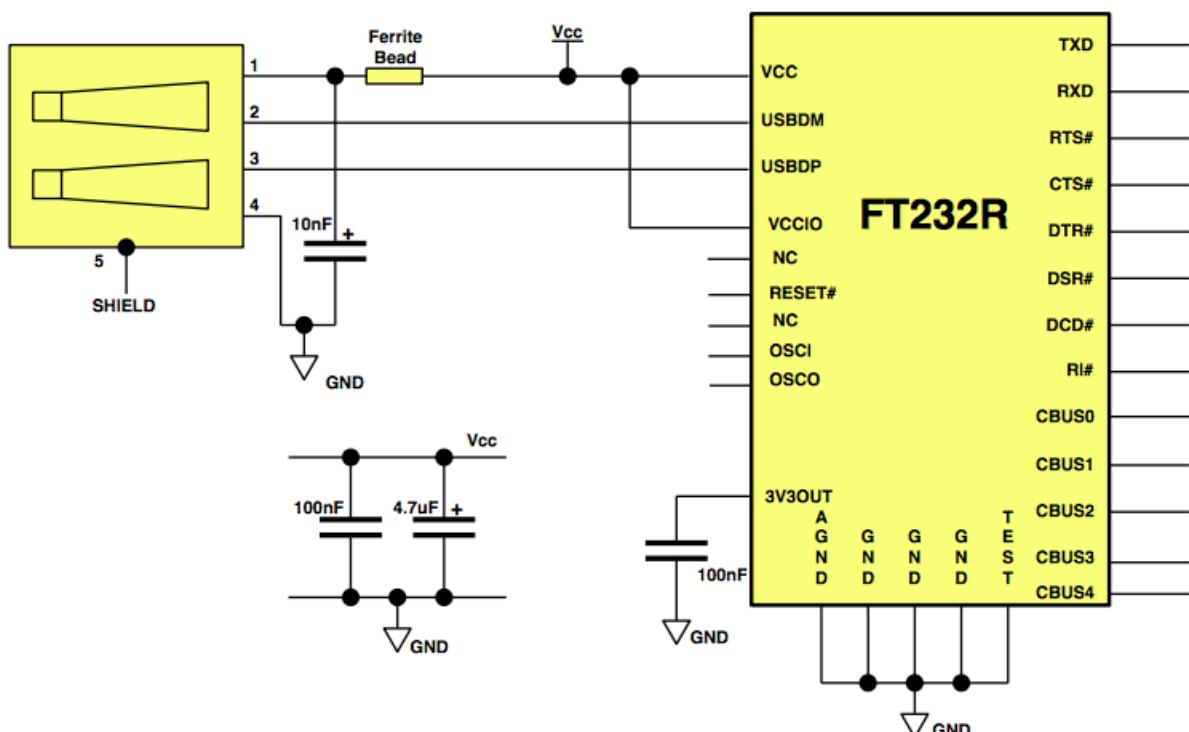
Annexes étudiant 1

Annexe n°1 : Extrait de la documentation FT232RL

3.3 QFN-32 Package



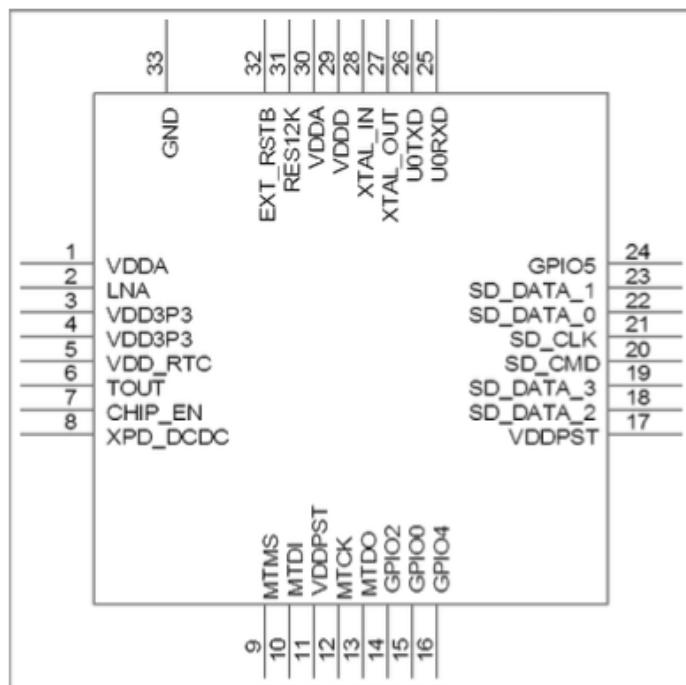
6.1 USB Bus Powered Configuration



Annexe n°2 : Extrait de la documentation ESP8266

Pin Definitions

Figure 2-1 shows the pin layout for 32-pin QFN package.



The functional diagram of ESP8266EX is shown as in Figure 3-1.

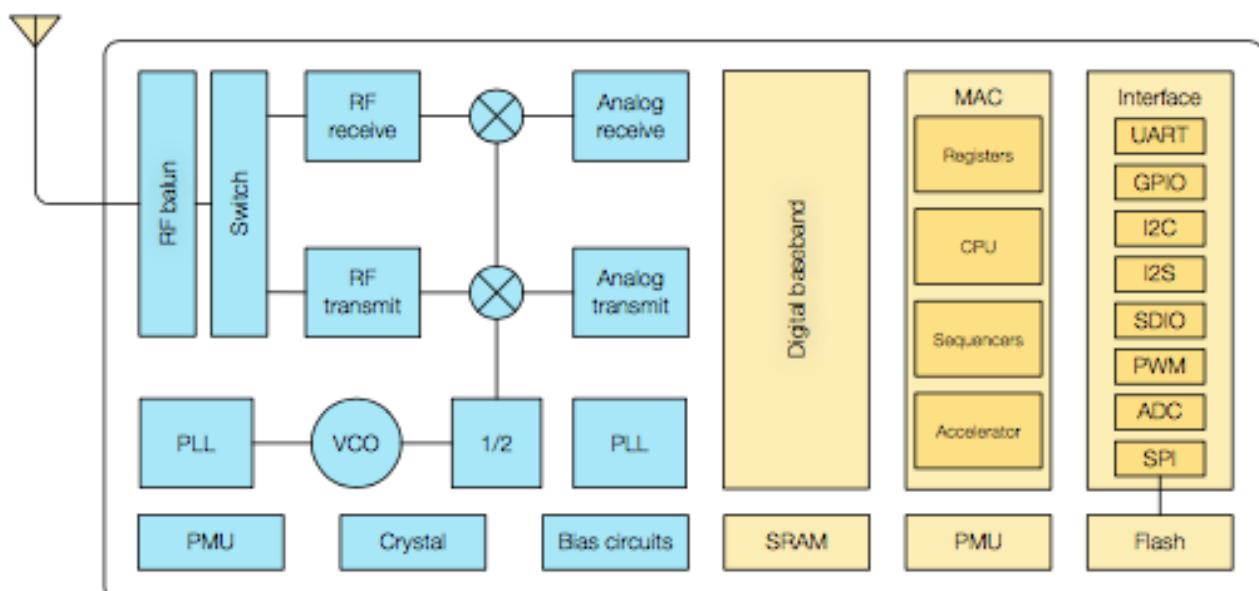


Figure 3-1. Functional Block Diagram

Annexe n°3 : Commandes AT

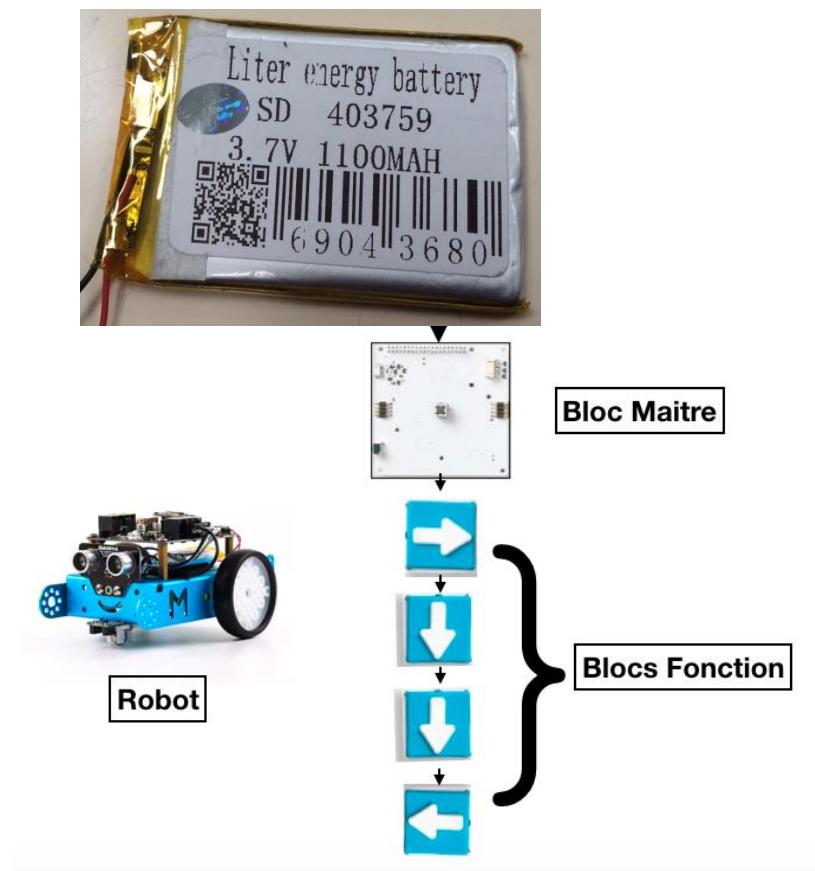
ESP8266 AT Command Set

Function	AT Command	Response
Working	AT	OK
Restart	AT+RST	OK [System Ready, Vendor:www.ai-thinker.com]
Firmware version	AT+GMR	AT+GMR 0018000902 OK
List Access Points	AT+CWLAP	AT+CWLAP +CWLAP:(4,"RochefortSurLac",-38,"70:62:b8:6f:6d:58",1) +CWLAP:(4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1) OK
Join Access Point	AT+CWJAP? AT+CWJAP="SSID","Password"	Query AT+CWJAP? +CWJAP:"RochefortSurLac" OK
Quit Access Point	AT+CWQAP=? AT+CWQAP	Query OK
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105 OK
Set Parameters of Access Point	AT+ CWSAP? AT+ CWSAP= <ssid>,<pwd>,<chl>, <ecn>	Query ssid, pwd chl = channel, ecn = encryption
WiFi Mode	AT+CWMODE? AT+CWMODE=1 AT+CWMODE=2 AT+CWMODE=3	Query STA AP BOTH
Set up TCP or UDP connection	AT+CIPSTART=? (CIPMUX=0) AT+CIPSTART = <type>,<addr>,<port> (CIPMUX=1) AT+CIPSTART= <id><type>,<addr>, <port>	Query id = 0-4, type = TCP/UDP, addr = IP address, port= port
TCP/UDP Connections	AT+ CIPMUX? AT+ CIPMUX=0 AT+ CIPMUX=1	Query Single Multiple
Check join devices' IP	AT+CWLIF	
TCP/IP Connection Status	AT+CIPSTATUS	AT+CIPSTATUS? no this fun
Send TCP/IP data	(CIPMUX=0) AT+CIPSEND=<length>; (CIPMUX=1) AT+CIPSEND= <id>,<length>	
Close TCP / UDP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	
Set as server	AT+ CIPSERVER= <mode>[,<port>]	mode 0 to close server mode; mode 1 to open; port = port
Set the server timeout	AT+CIPSTO? AT+CIPSTO=<time>	Query <time>0~28800 in seconds
Baud Rate*	AT+CIOBAUD? Supported: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600	Query AT+CIOBAUD? +CIOBAUD:9600 OK
Check IP address	AT+CIFSR	AT+CIFSR 192.168.0.106 OK
Firmware Upgrade (from Cloud)	AT+CIUPDATE	1. +CIPUPDATE:1 found server 2. +CIPUPDATE:2 connect server 3. +CIPUPDATE:3 got edition 4. +CIPUPDATE:4 start update
Received data	+IPD	(CIPMUX=0): + IPD, <len>; (CIPMUX=1): + IPD, <id>, <len>; <data>
Watchdog Enable*	AT+CSYSWDTENABLE	Watchdog, auto restart when program errors occur: enable
Watchdog Disable*	AT+CSYSWDTDISABLE	Watchdog, auto restart when program errors occur: disable

RAPPORT:

Projet Robot Blocs

(partie Bloc Maître)

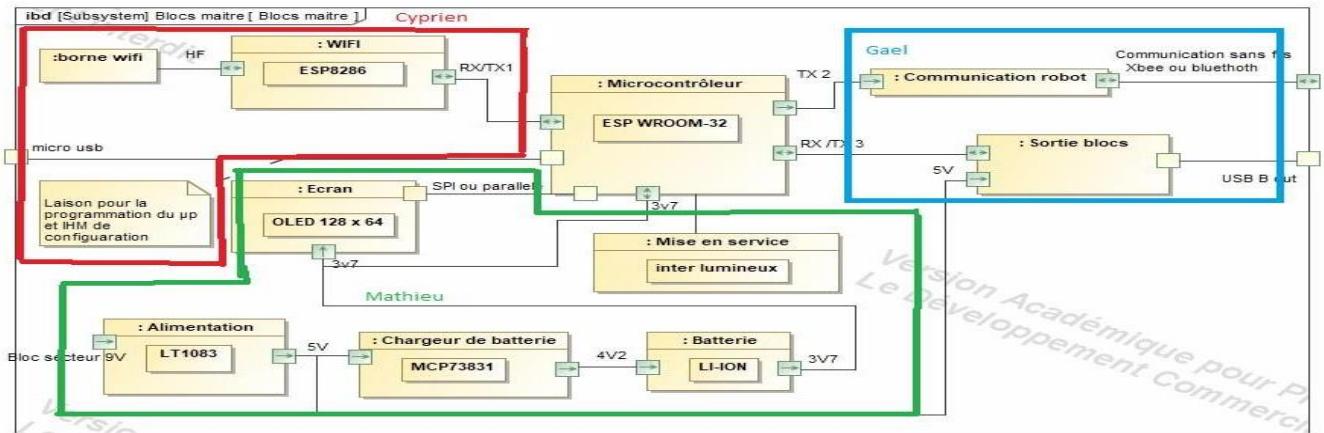


1ère Partie : Objectifs

Le projet bloc consiste en la réalisation et la conception d'un jeu pédagogique destiné aux classes préparatoire (CP) et donc à des enfants âgés de 5 à 6 ans.

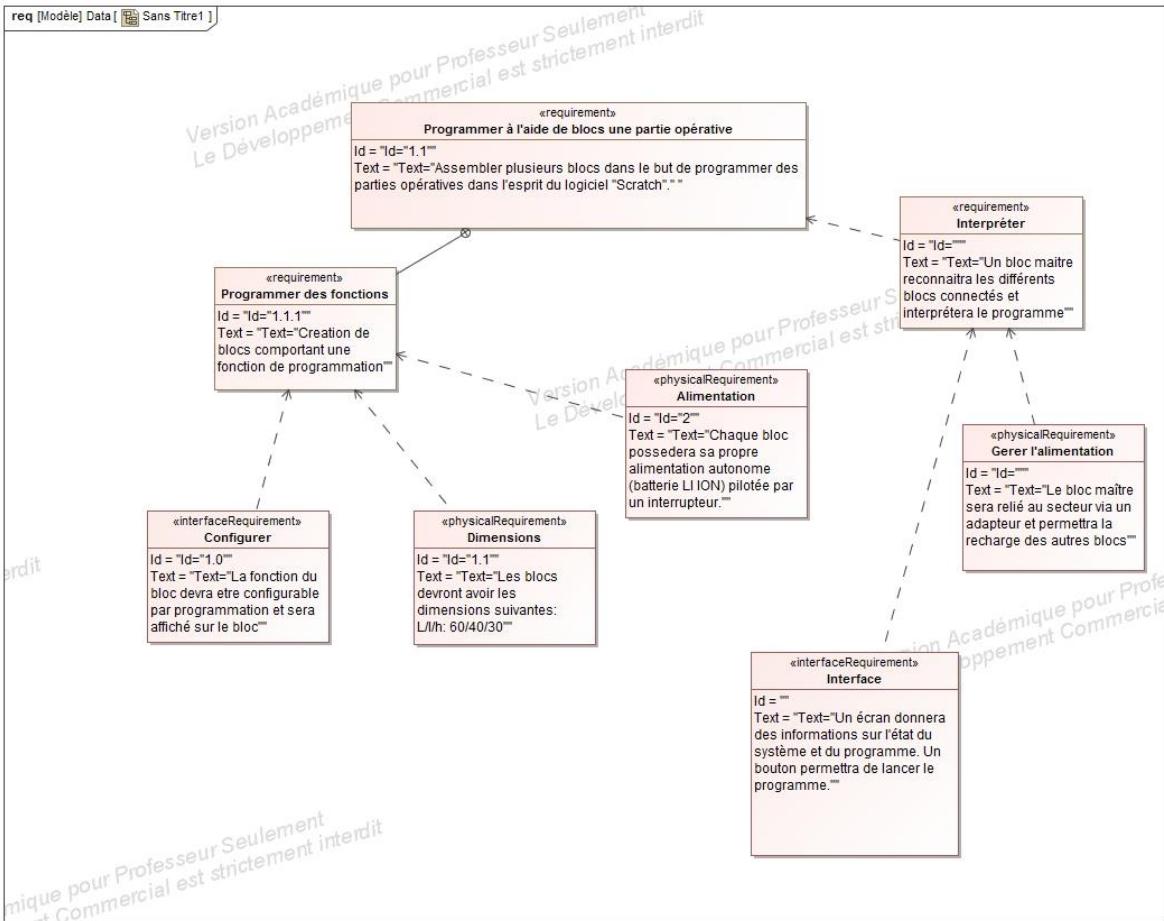
Son objectif est de développer la créativité et la réflexion de l'enfant sous forme d'un jeu à un prix relativement accessible.

Mes attributions :



Comme l'indique le schéma ci-dessus mon rôle consiste à alimenter chaque bloc avec une batterie et un chargeur de batterie et l'affichage de l'écran et produire un inter lumineux

Cahier des charges



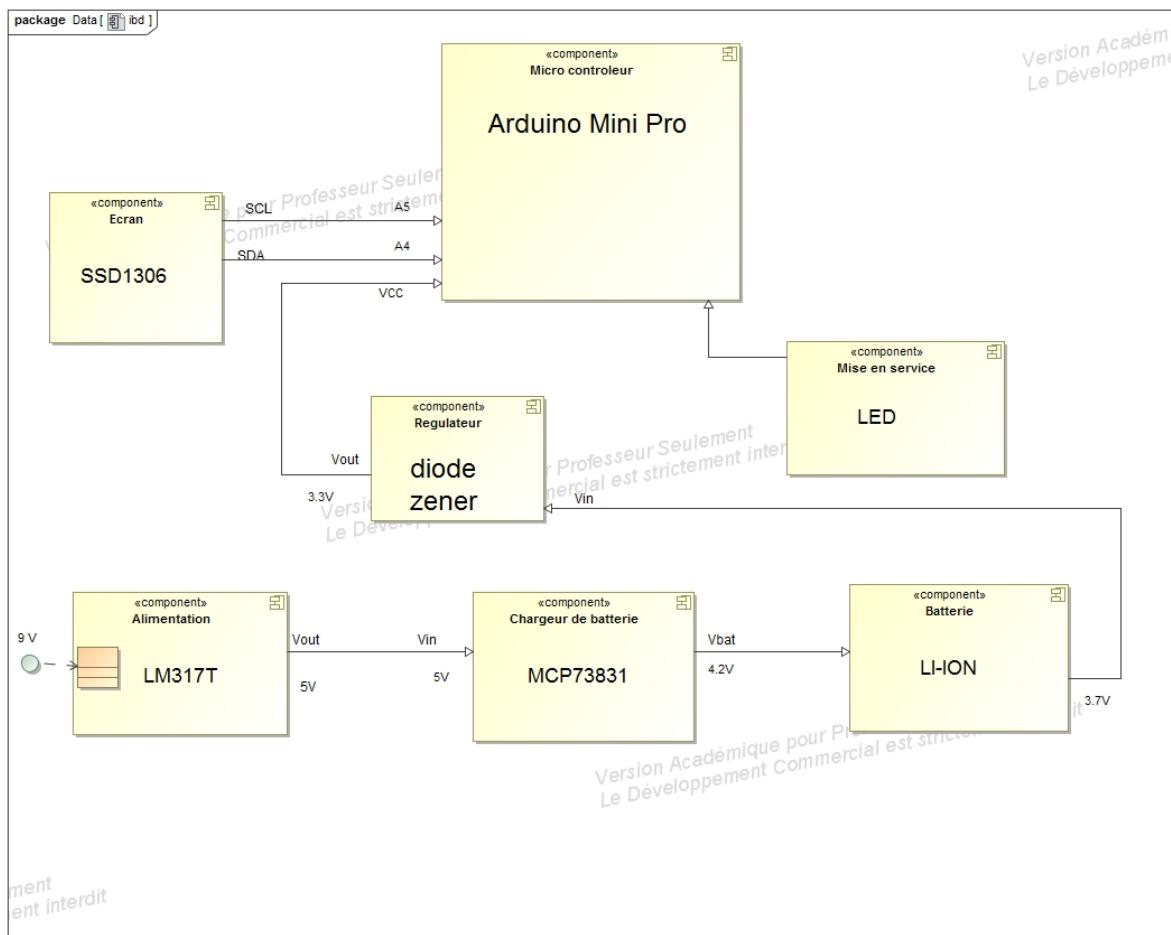
La batterie est de type Lithium-Ion procurant 3.7 V.

Les connecteurs doivent être de type USB2.

Chaque bloc possède sa propre alimentation autonome pilotée par un interrupteur.

- La fonction sélectionnée sera affichée sur l'écran.
- Il faut utiliser des abaisseurs de tensions.
- L'écran doit faire 128x64 pour une question de taille de la carte.

IBD détaillé (Diagramme de bloc interne)



Alimentation

Au vue de l'usage envisagé pour notre projet Robot Blocs à destination des classes de maternelles des écoles, il n'était pas réaliste d'utiliser de simples piles pour alimenter le système.

En effet cela aurait représenté un coût élevé à long terme et un impact négatif en terme de développement durable.

D'autre part les batteries Li-ion employées correctement présentent un niveau de risque moins élevé que les piles classiques.

Ce qui différencie une pile d'une batterie est que la batterie est pas définition rechargeable, si on tente de recharger une simple pile on s'expose à un risque très élevé d'explosion.

L'usage de batteries Li-ion n'est pas exempt de risques surtout si l'on choisit des batteries à forte capacité.

Le risque majeur dans ce cas est le court-circuit qui peut engendrer la projection incandescente des conducteurs fondus.

Notre projet est constitué de cinq blocs reliés entre eux en parallèle grâce à des ports Usb.

Chaque bloc est alimenté par une batterie Li-ion de 3,7 Volt.

L'inconvénient du montage en parallèle est que même sans utilisation il circule entre les éléments des courants.

Cette circulation est due aux résistances internes des éléments et elle contribue à une décharge lente mais progressive de la batterie.

C'est pourquoi il nous a semblé opportun de raccorder le bloc maître au secteur afin de disposer d'un surplus d'alimentation qui permet de compenser la perte de charge des autres blocs et permet d'utiliser la profondeur de décharge à notre avantage.

En effet une batterie lithium ion s'use moins vite lorsqu'elle est rechargée tous les 10% que lorsqu'elle l'est tous les 80%.

Il est bien évident que pour assurer la stabilité et la sécurité du système nous avons fait le choix d'installer des éléments disposant des mêmes caractéristiques.

Caractéristiques de nos batteries Lithium ion :

Capacité nominale : 1050mAh (min) ;

1100 mAh (valeur caractéristique)

Voltage nominal : 3,7 V (tension moyenne)

Tension de charge : 4,2 ± 0,05 V

Courant de charge maximum : 1050mAh

Procédé de chargement : courant constant, voltage constant

Recharge standard : 525mAh (courant constant) pendant 3,5 heures

Recharge rapide : 1050mAh (courant constant) pendant 3 heures

Décharge maximum : 2100mAH

Dimensions :

Epaisseur : 4,8 ± 0,2

Largeur : 35 ± 1

Longueur : 63 ± 1

Poids : 23 ± 1 g

2ème Partie : Choix et présentation des composants

Si notre choix s'est porté sur les batteries Lithium ion c'est qu'elles offrent une densité d'énergie spécifique et volumique élevée (4 à 5 fois plus qu'un accumulateur NiMh (Nickel Métal Hybride)), une charge rapide, elles sont peu polluantes et recyclables (même si ce recyclage est très technique), elles disposent d'une forte tension, n'ont pas d'effet mémoire (effet mémoire = l'incapacité d'une batterie à délivrer la totalité de sa charge en dessous d'un seuil qu'elle a mémorisé lors de recharges successives intervenant trop tôt dans son cycle de décharge) et elles sont d'un rapport encombrement/puissance et poids favorable.

D'autre part leur pourcentage d'auto-décharge est assez faible par rapport à d'autres accumulateurs (5 à 10% par mois) et elles ont une durée de vie raisonnable (500 à 1000 recharges).

Une batterie Li-ion est composée de plusieurs cellules connectées en série et en parallèle en fonction de la tension et des exigences de l'appareil.

Trois types de cellules de batterie sont couramment utilisés :

- cylindrique,
- prismatique,
- polymer.

Nos batteries sont composées de cellules de type polymer, elles sont fines et rectangulaires et n'excèdent pas 10 mm d'épaisseur ce qui joue en faveur d'une carte de petite dimension.

Au sein du projet j'ai réalisé un schéma structurel à l'aide du logiciel Eagle ce qui m'a pris beaucoup de temps, puisqu'il fallait apprendre le fonctionnement de ce logiciel et créer de nouvelles bibliothèques qui permettent d'ajouter de nouveaux composants non connus du logiciel d'origine.

Pour ce faire j'ai du consulter internet afin de trouver les composants adéquats et vérifier leur conformité et leur emplacement ou empreinte sur la carte support.

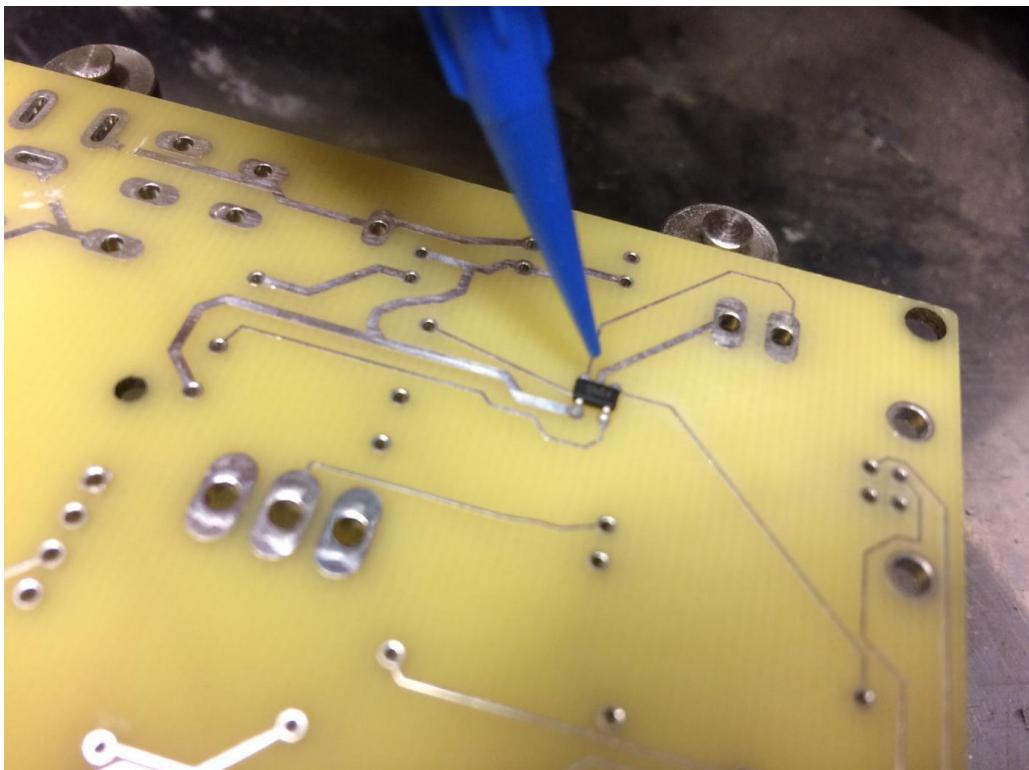
Dans certains cas la bibliothèque souhaitée n'existe pas et il convient alors de la créer.

Ce fut le cas pour notre carte Arduino mini pro qui a nécessité la création d'une bibliothèque par Cyprien puisque notre carte Arduino n'était pas d'origine et ne disposait pas de la même disposition au niveau des ports.

D'autre part il a fallu calculer la valeur de tous les composants à l'aide de lois comme la loi d'Ohm ou la loi des mailles pour calculer l'intensité par exemple.

Une fois ces étapes réalisées nous avons passé commande de la carte auprès du fabricant et nous nous sommes aperçus de quelques erreurs postérieurement ; aussi avons nous dû annuler notre première commande et la renouveler en corrigeant nos erreurs (oubli d'une Led et d'un résistance).

Le délai d'attente fut de 2 semaines et nous avons accumulé du retard car nous ne disposions pas dans notre établissement du matériel nécessaire à la soudure d'un micro composant : le chargeur de batterie MCP73831



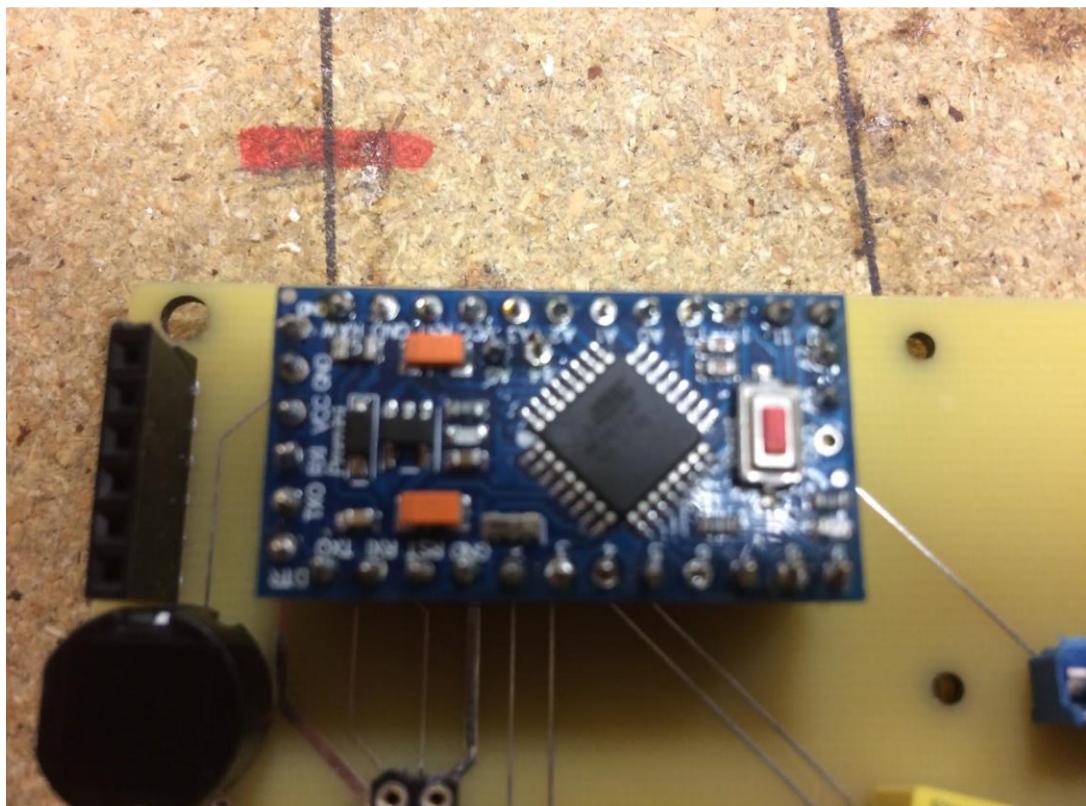
L'équipe pédagogique a donc dû commander une pâte à souder spécifique mais comble de malchance, elle s'est trompée dans les références de la commande ce qui à rallonger d'autant le délai pour que j'entame la soudure des différents éléments.

En attendant je mets à profit ce délai pour préparer notre carte à recevoir les composants.

En effet afin de faciliter la maintenance future de notre système nous avons prévu que certains composants soient amovibles pour faciliter le changement en cas de panne.

Je dessoude donc les pâtes d'origine des composants comme la carte Arduino mini pro pour les remplacer par des pâtes plus fines qui viendront se clipser sur les pâtes de notre carte mère.

Carte Arduino mini Pro



La carte Arduino mini Pro est une carte micro contrôleur. C'est grâce à cette carte que sont créés et stockés les programmes nécessaires au fonctionnement de la carte mère liée à notre projet.

Par exemple c'est dans la carte Arduino que se trouve le programme d'affichage du drapeau sur l'écran.

Elle doit être alimentée en 3.3 V.

Pour la programmer nous utilisons le logiciel Arduino et le composant FT232RL.

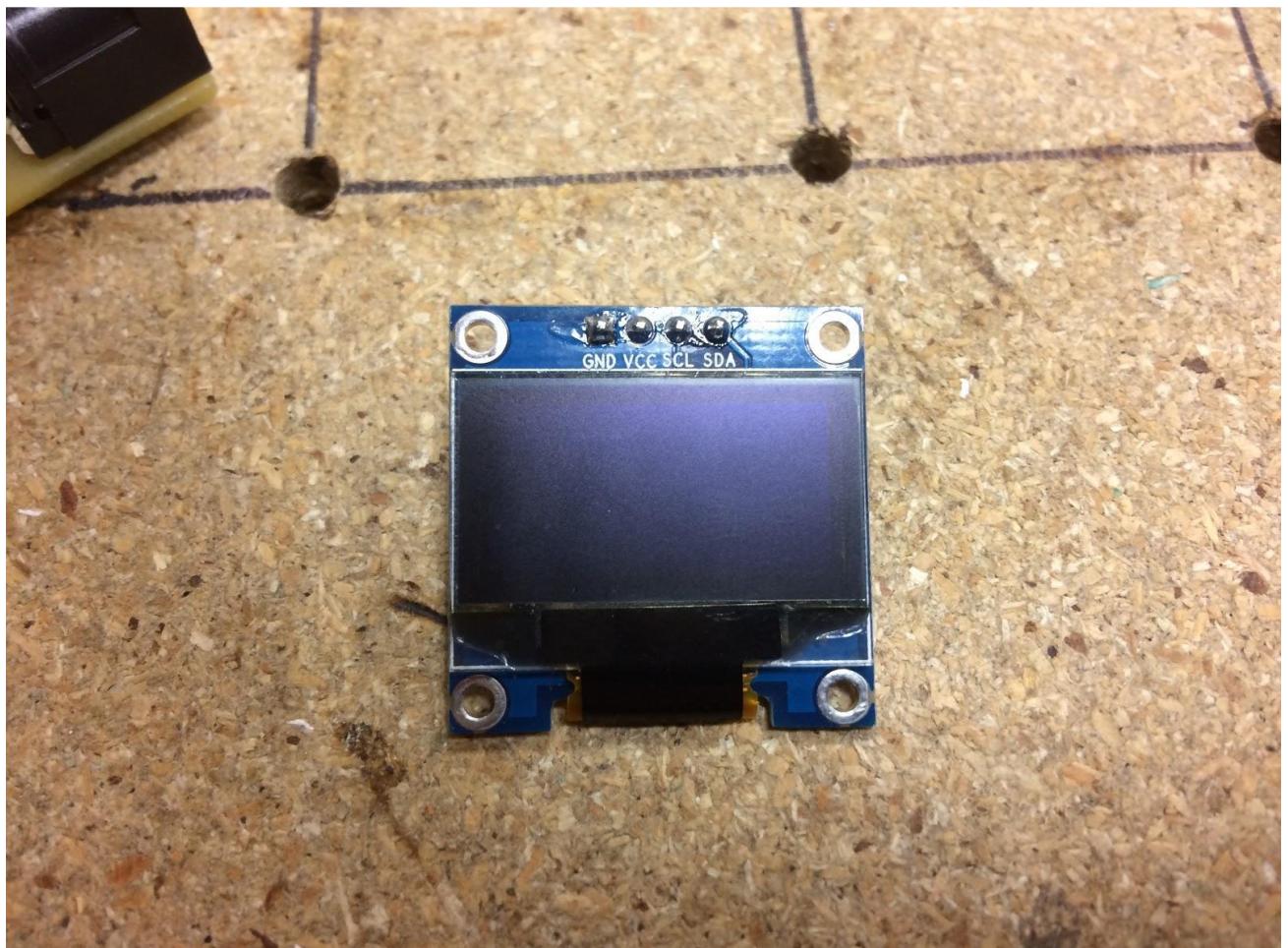
Le module Xbee Pro



Il s'agit d'une carte électronique qui établit une liaison radio permettant de communiquer à distance. C'est cette carte qui va transmettre les actions que doit effectuer le robot.

Ce composant est géré par l'étudiant 3 (Gaël Marguin).

Ecran Oled 128 x 64



Notre choix s'est porté sur cet écran OLED SSD1306 qui dispose d'une définition de 128 x 64 pixels.

Nous l'avons câblé en liaison I2C car cela permet de réduire de façon importante le câblage.

En effet il ne faut que 4 fils : les 2 fils d'alimentation de 3.3 : VCC et la masse GLD , 1 fils SDA et 1 fils SCL qui permettent la liaison I2C et la transmission des données.

De plus il est de petite taille et ne coûte pas très cher.

Le chargeur de Batterie MCP73831



Il s'agit d'un contrôleur de charge de batterie dont le coût est minime.

Il fonctionne en 5 V et son intensité maximale est de 500 mA. Il permet de recharger les batteries et donc de fournir une tension de 4.2 v en sortie.

Il s'agit d'un micro composant CMOS donc minuscule comme le montre la photo ci-dessus.

Pour réaliser une charge rapide il faut mettre une résistance Rprog entre la sortie Vss et Prog .

Voici comment calculer la résistance :

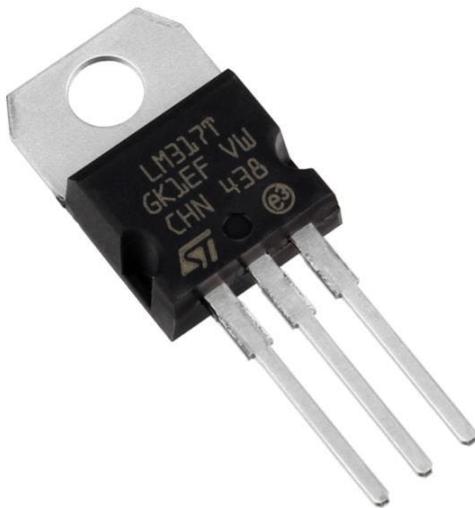
$$I_{REG} = \frac{1000V}{R_{PROG}}$$

Where:

R_{PROG} = kOhms

I_{REG} = milliampere

Le régulateur de tension LM317T



Nous avons dû remplacer le LT1083 par un LM317T.

Ces composants sont des régulateurs de tension, en effet ils permettent de convertir du 9 V en 5 V.

Le LT1083 coûte excessivement cher et nous disposions à l'école du LM317T qui agit de la même manière que le LT1083.

Nous utilisons un potentiomètre qui fait varier la valeur de sortie pour effectuer nos réglages.

3ème Partie : Programmation

Ma participation à ce projet portait également sur l'affichage de la carte mère.

A cette fin il a fallu que je programme un code pour afficher un drapeau de départ sur l'écran OLED SSD1306.

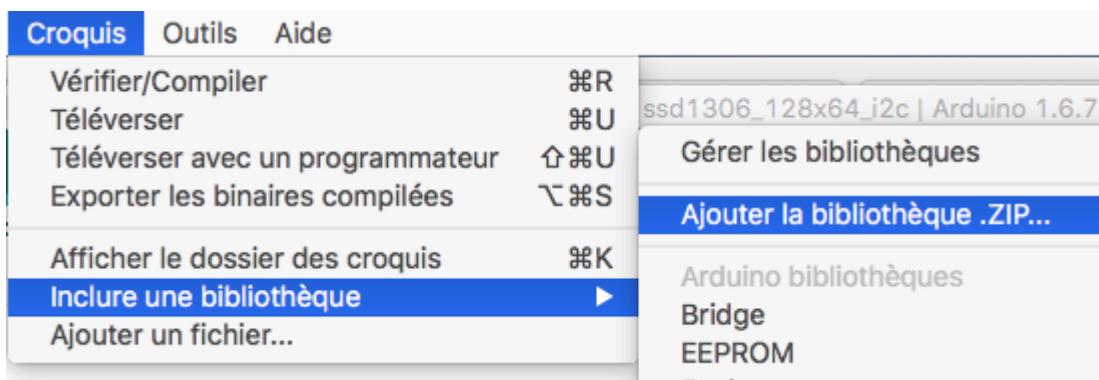
J'ai utilisé Arduino comme outil logiciel.

Je programme en langage C++.

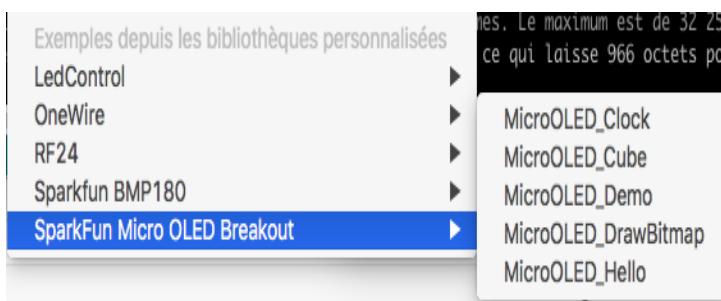
Pour commencer il faut ajouter une nouvelle librairie sur la carte Arduino puisqu'un collègue utilise la librairie Adafruit et donc nous avons voulu tester les deux librairies.

Nous avons donc choisi une librairie Sparkfun qui est plus compacte qu'Adafruit même si les fonctions de tracé sont un peu moins nombreuses.

Il faut commencer par télécharger la librairie mais sans compresser le Zip, puis il faut aller dans Croquis / Inclure une bibliothèque / Ajouter la bibliothèque .ZIP



Enfin il faut choisir le ZIP de la librairie à importer.



Pour afficher une image il faut utiliser le logiciel LCD assistant qui permet de convertir une image au format Bitmap en tableau de caractères affichable sur le mini écran OLED.

Pour pouvoir intégrer une image sur notre écran il faut d'abord trouver une image en noir et blanc (car l'écran n'affiche que du noir ou blanc) sur internet par exemple puis de la convertir en .bmp.

Ensuite il faut redimensionner l'image sur paint (64 pixels par 64 pixels).
Une fois cela fait il faut utiliser le logiciel LCD Assistant qui permet de générer un tableau grâce à l'image.

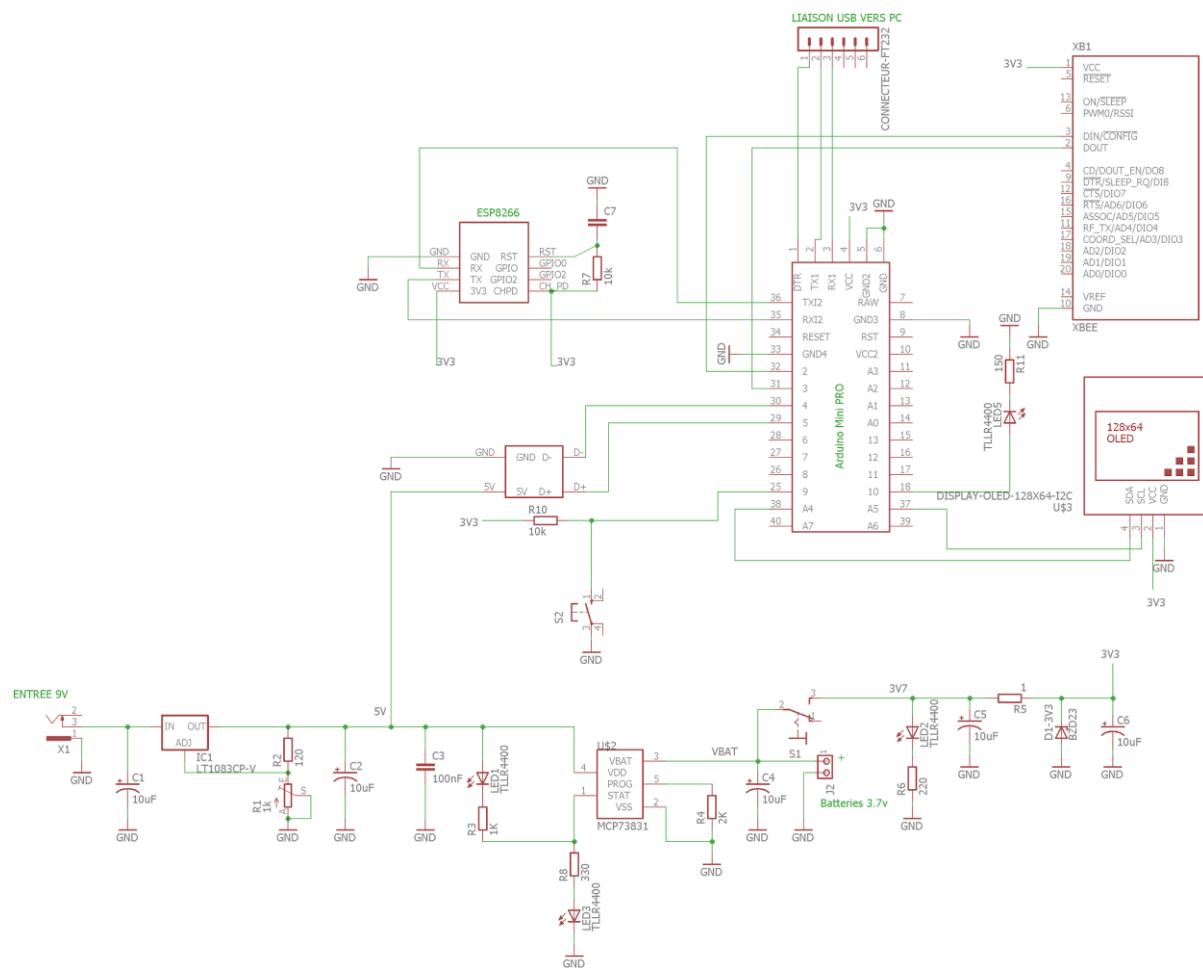
Nous avons juste à copier le tableau généré (en supprimant la première ligne de code) et à le coller à la place du tableau de l'exemple.

4ème Partie : Schéma structurel et Typon

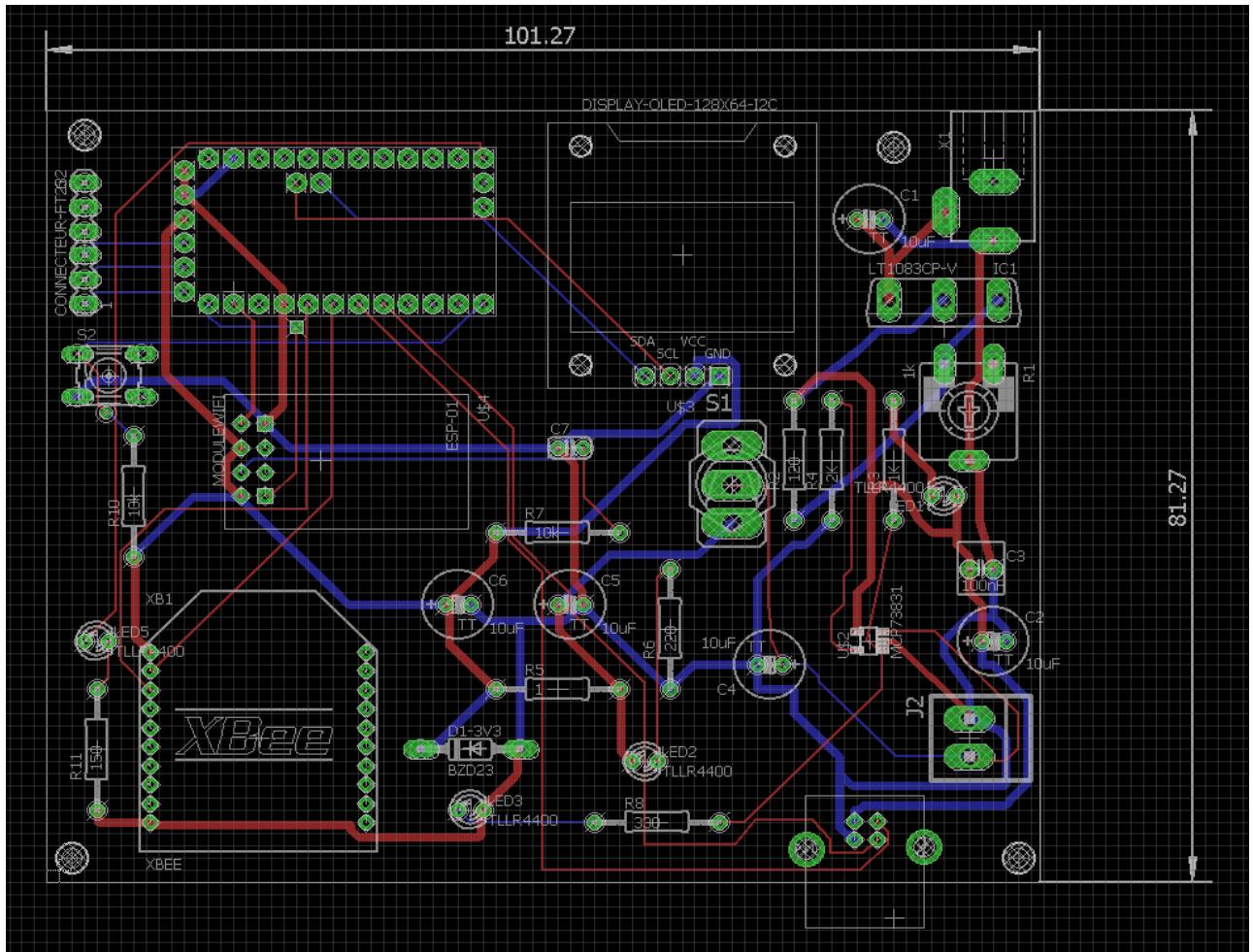
Pour réaliser le schéma structurel et le Typon nous avons utilisé le logiciel Eagle



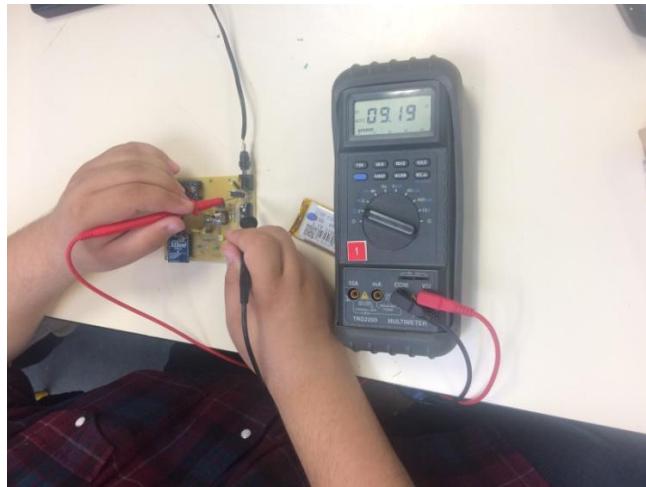
Schéma structurel



Typon



5ème Partie : Mesures et validation



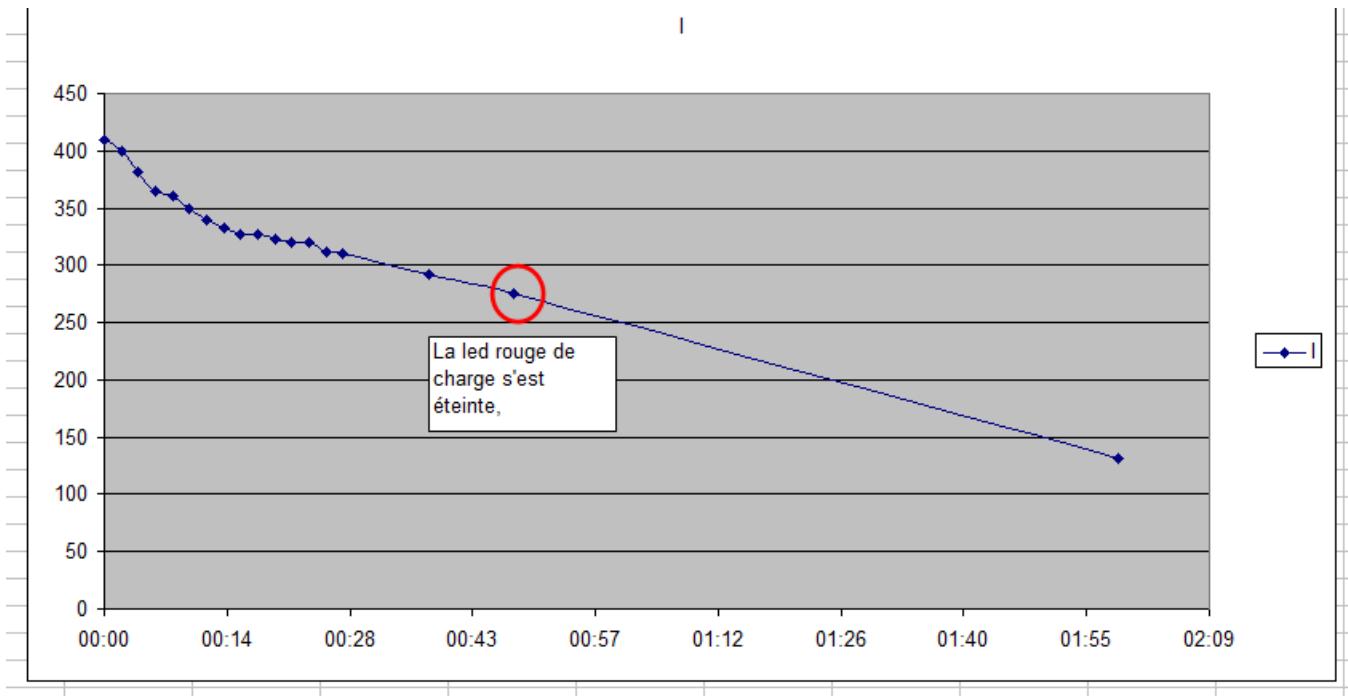
- Mesures juste avec la batterie et interrupteur enclenché

	Valeur attendue	Valeur obtenue	Validation
Batterie	3.7 V	3.8 V	valider
Diode zener	3.3 V	3.52 V	valider
Ecran : Vcc	3.3 V	3.5 V	valider
Sda	3.3 V	3.3 V	valider
Scl	3.3 V	3.29 V	valider
Xbee	3.3 V	3.48 V	valider
Esp	3.3 V	3.57 V	valider
Arduino	3.3 V	3.57 V	valider

Même si les valeurs sont légèrement plus élevées que celles attendues, il est possible de corriger ce détail en changeant la diode zener par une plus puissante pour abaisser la tension.

Nous avons mesuré l'intensité aux bornes de la batterie et nous sommes à 440.27 mA.

- Mesure de l'intensité lors de la charge de la batterie



Il s'agit de la courbe de l'intensité aux bornes de la batterie en fonction du temps.

On remarque que l'intensité baisse en fonction du temps et c'est assez linéaire.

Mesures avec l'alimentation et la batterie mais interrupteur fermé.

		Valeur attendue	Valeur obtenue	Validation
Mcp	Vbat	4.2 V	4.19 V	valider
	Vdd	5 V	5.02 V	valider
Lm317T	In	9 V	9.19 V	valider
	out:	5 V	5.02 V	valider

Toutes les valeurs de tension sont validées, néanmoins la valeur de l'intensité est de 220 mA ce qui est élevé puisque nous devons avoir une valeur maximale d'intensité de 60 mA. Si nous activons à la fois l'interrupteur et l'alimentation alors que le bloc maître est lui-même branché au secteur cela crée une surchauffe sur le chargeur de batteries et également sur le LM317T et cela grille le chargeur de batteries en quelques minutes puisque l'intensité prévue de 500 mA est dépassée.

Pour résoudre le problème nous avons tenté de faire des calculs de dissipation thermique.

Calcul de dissipation thermique pour le chargeur de batterie :

$$P=U \times I$$

$$P_d = (T_j - T_a) / R_{th}$$

Puissance dissipée

$$P_d = P_s$$

$$(125-25)/230 = 0.8 * I_s$$

$$I = 0.543 \text{ A}$$

Calcul de dissipation thermique pour le lm317t :

$$(125-25)/(60+5) = 4.8 * I$$

$$I = 0.320 \text{ A}$$

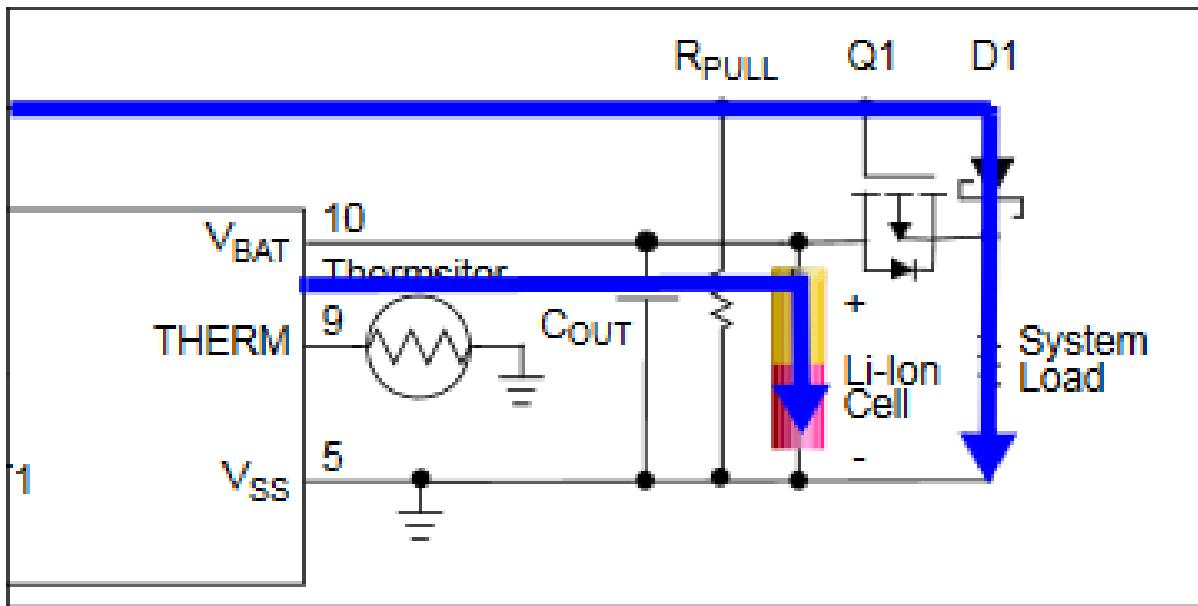
Nous avons donc rajouté un radiateur sur le LM317T. Malheureusement nous ne pouvons pas rajouter un radiateur sur le chargeur de batteries puisque c'est un composant CMOS (Complementary Metal Oxide Semiconductor) et qu'il est trop petit.

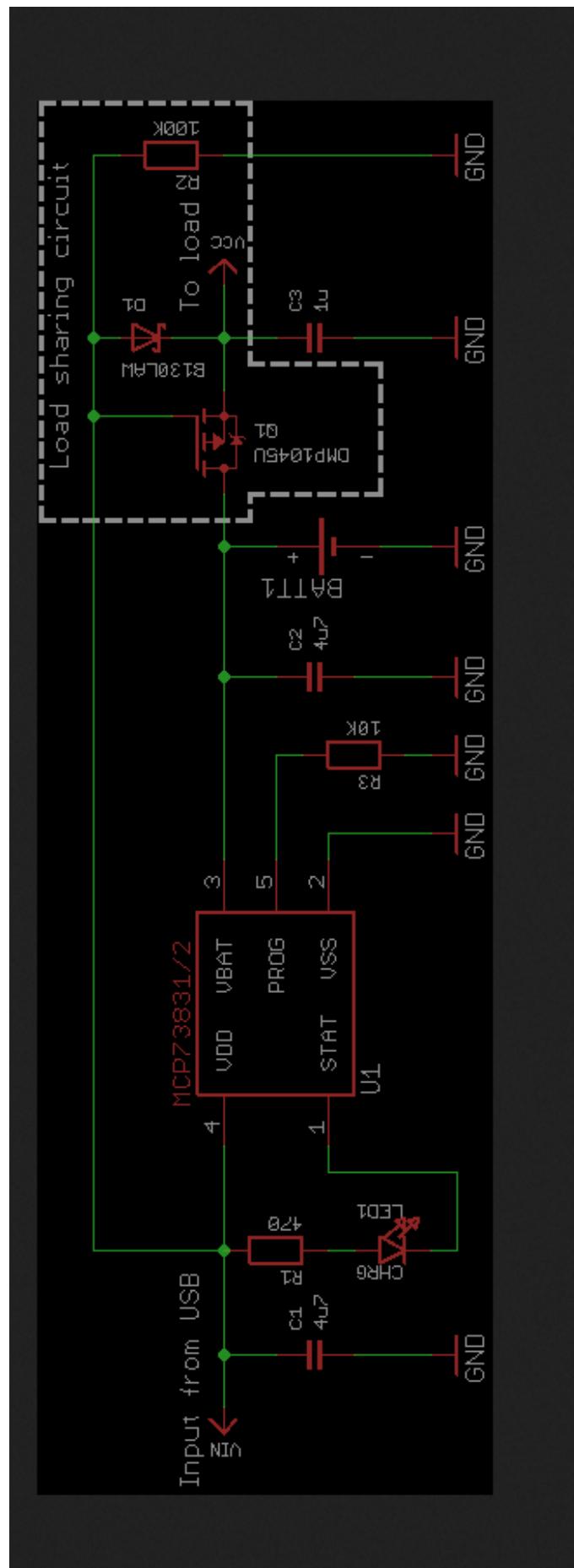
De plus nous avons fait une erreur de conception car le chargeur de batteries ne peut accepter une charge de batterie et une alimentation simultanée.

Nous n'avons malheureusement pas vu sur la documentation technique qu'il fallait agrandir les pistes autour du composant ou bien percer les pistes pour dissiper la chaleur.

Pour tenter de résoudre le problème nous nous sommes appuyés sur un site Internet présentant une solution adaptée à notre cas, le blog : <http://blog.zakkemble.co.uk>

Il faut ajouter un MOSFET en série après le MCP73831 pour créer un circuit qui permettrait d'alimenter le reste de la carte et de charger en même temps la batterie.





6ème Partie : Estimation du coût de ma partie

Fournisseur	Composant	Prix
Conrad—	MCP73831	1.21 €
Sparkfun	XBEE PRO	35 €
ALIEXPRESS	ECRAN	2 €
	ESP8266	1.36 €
	ARDUINO MINI PRO 328	1.72 €
	BATTERIE 3.7 V	4.14 €
	FTDI F232RL	1.64 €
EURO CIRCUIT	TYPON	30 €
TOTAL		77.07 €

Les deux éléments les plus coûteux de notre projet sont le Typon et la carte Xbee Pro, mais en commandant plusieurs exemplaires de Typon le prix devient dégressif et plus attractif.

Réalisation finale



Tout fonctionne correctement sauf si l'on branche l'alimentation et la batterie en même temps comme expliqué précédemment.

Un support de batterie serait envisageable pour améliorer l'esthétique de la carte.

ANNEXES

Code Arduino pour afficher le drapeau de départ

```
#include <Wire.h> // Include Wire if you're using I2C
#include <SPI.h>
#include <SFE_MicroOLED.h> // Include the SFE_MicroOLED library

#define PIN_RESET 9 // Connect RST to pin 9 (req. for SPI and I2C)
#define PIN_DC 8 // Connect DC to pin 8 (required for SPI)
#define PIN_CS 10 // Connect CS to pin 10 (required for SPI)
#define DC_JUMPER 0

MicroOLED oled(PIN_RESET, DC_JUMPER); // Example I2C declaration

#include <SoftwareSerial.h>

String trame = "A;4;d;g;a;b;\r\n";
int pinBouton;
int led;
int etatBouton;
int etatantérieur, etat;
int increment;
int nbboucle=0;
SoftwareSerial espSerial(12, 11); // RX, TX

void AT(String mes)
{
    espSerial.print(mes);
    espSerial.write(13); // \n
    espSerial.write(10); // \r
}

void recMessage()
{
    String texte = "";
    do
        if (espSerial.available() > 0)
    {
        char car = espSerial.read();
        if (car == '+')
            do
            {
                if (espSerial.available() > 0) car = espSerial.read();
            }
            while (car != ':');
        else
            texte += car;
    }
    while (texte.indexOf("CLOSE") == -1); //
```

```

        while (espSerial.available() > 0) espSerial.read();
        Serial.print("réponse serveur:");
        Serial.println(texte);
        Serial.println();
    }

void AttendConnect(String mot)
{
    String texte = "";
    do
        if (espSerial.available() > 0)
    {
        char car = espSerial.read();
        texte += car;
        Serial.print(car);
    }

    while (texte.indexOf(mot) == -1);
    while (espSerial.available() > 0) espSerial.read();
    Serial.println();
    Serial.println(".....");
    Serial.println();
}

```

```

uint8_t bender [] = {
0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1C, 0x7C, 0xFE, 0xFE, 0x7E,
0x7C, 0x30, 0x00,
0x80, 0x80, 0x80, 0x00, 0x06, 0x3C, 0xE0, 0x80, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x20, 0xF0,
0xF8, 0xFC, 0xFE, 0xF0, 0x40, 0x00, 0xC0, 0xF0, 0xF8, 0xFC, 0xF0, 0xC0, 0x00,
0x80, 0xC0, 0xC0,
0xE0, 0xE3, 0xCF, 0x0F, 0x0F, 0x1F, 0x1E, 0x1E, 0x3F, 0x3E, 0x70, 0xC0, 0x00,
0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x01, 0x0F, 0x1F, 0x1E, 0x3E, 0x3E, 0x3E, 0x78, 0x60, 0x00, 0x80, 0x80,
0x80, 0xC0, 0xE1,
0xE7, 0x07, 0x03, 0x00, 0x00, 0x00, 0x81, 0xC7, 0x07, 0x03, 0x01, 0x00, 0x00,
0x01, 0x87, 0x1F,
0xF, 0x0F, 0x07, 0x06, 0x00, 0x08, 0x38, 0xF8, 0xF0, 0xF0, 0xF0, 0xE3, 0xEE,
0xF8, 0xE0, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00
}

```

```
0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0xE0, 0xE0, 0xC0, 0xC0, 0xC7,  
0x9F, 0x1F, 0x0F,  
0x0F, 0x07, 0x00, 0x00, 0x1E, 0xFF, 0x7F, 0x3F, 0x1F, 0x04, 0x00, 0x1C, 0xFE,  
0xFF, 0x7F, 0x3F,  
0x0C, 0x00, 0x10, 0xF8, 0xFC, 0xFC, 0xFD, 0x61, 0x01, 0x01, 0x01, 0x03,  
0x03, 0x07, 0x0F,  
0x3C, 0xE0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x03, 0x03, 0x07, 0x07,  
0x07, 0x04, 0x00,  
0x10, 0x78, 0xF8, 0xF8, 0xFC, 0xFC, 0xF0, 0x40, 0x00, 0xC0, 0xE0, 0xF8, 0xFC,  
0xF0, 0xC0, 0x00,  
0x00, 0x00, 0x00, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x01, 0x03, 0x1C, 0x70, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x0E,  
0x3E, 0x3C, 0x7C,  
0x7C, 0x7C, 0x79, 0xC1, 0x01, 0x00, 0x00, 0x00, 0x00, 0x01, 0x0F, 0x0F, 0x07,  
0x03, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x03, 0x0E, 0x78, 0xC0, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x0F, 0x38, 0xE0, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00, 0x00, 0x00, 0x07,  
0x1C, 0x60, 0x00  
};
```

```
void setup()
{
    pinBouton=9;
    led = 10;
    pinMode(pinBouton,INPUT);
```

```

pinMode(led,OUTPUT);

Serial.begin(9600);
espSerial.begin(9600);

Serial.println("Demarrage... ");
Serial.println();

//AT("AT+CWJAP=\\"BTS SNEC\\",\\"\\"); // connexion au Wifi en DHCP au SSID du
lycée sans mot de passe. Les \ permettent la gestion des caractères speciaux
// AttendConnect("OK");
oled.begin();
}

void loop()
{
    while(nboucle > 0)
    {
        etatBouton=digitalRead(pinBouton);
        if((etatBouton == 1)&&(etatanterieur==0))
        {
            if(etat==0){
                oled.clear(PAGE);
                oled.display();
                digitalWrite(led, LOW);
                etat=1;
            }
            else
            {
                oled.drawBitmap(bender);
                oled.display();
                digitalWrite(led, HIGH);

                etat=0;
            }
        }
        etatanterieur= etatBouton;
    }
}

```

```

// AT("AT+CIPSTART=\\"TCP\\",\\"172.16.127.113\\",11000"); //connexion au serveur
// AttendConnect("OK");

// int nbCar = trame.length(); // Détermination du nb de caractères à envoyer
// AT("AT+CIPSEND="+String(nbCar)); // Envoie du nombre de caractères
// AttendConnect("OK");

```

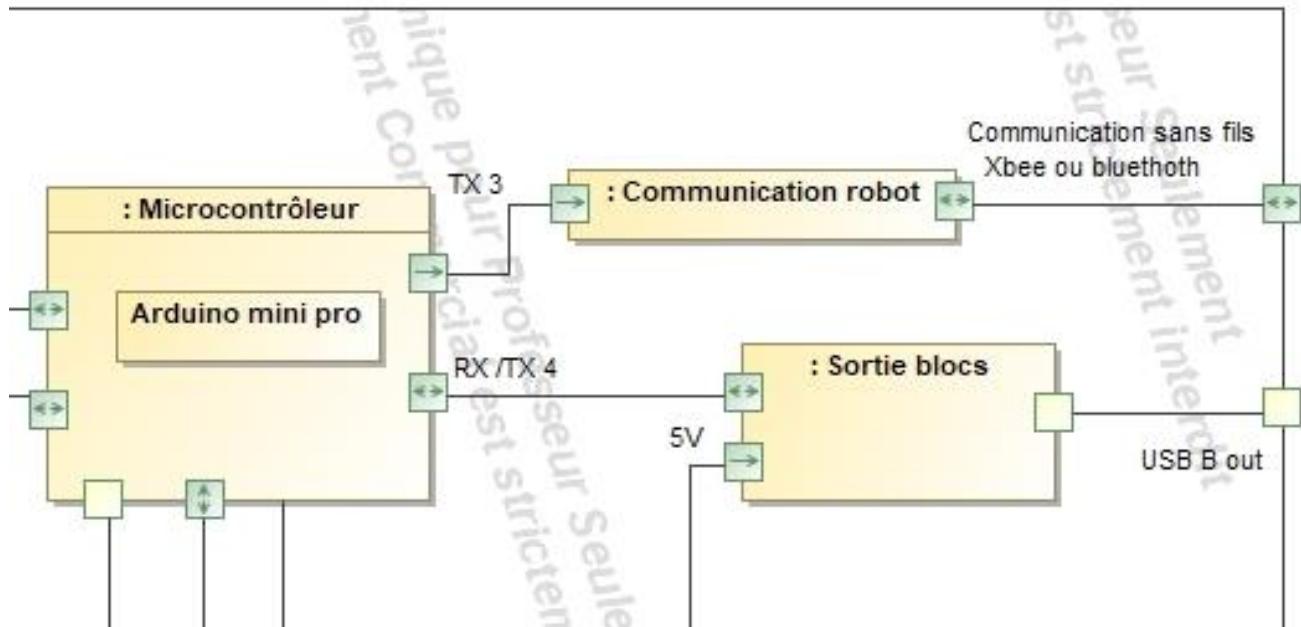
```
// espSerial.print(trame);      //Envoie de la requête au serveur  
// espSerial.write(13); espSerial.write(10);  
  
// recMessage();    // réception de la réponse du serveur  
// AT("AT+CIPCLOSE"); // deconnexion  
// delay(5000);  
nbboucle=1;  
}
```

pour Utilisation de librairies dont celle de Microled Sparkfun et celle de l'I2C communiquer.

drapeau Il s'agit du tableau de valeurs hexadécimales représentant l'image du en BITMAP.

activé Il s'agit de la programmation de l'interrupteur lumineux qui quand il est allume la LED et affiche le drapeau sur l'écran

Étudiant 3 : Lecture du programme construit et communication robot



I) Conception préliminaire :

1.1) Communication sans fils de type Xbee

Le choix a été imposé sur du Xbee

Présentation

XBee est une famille de composants de communication sans-fil facile à mettre en œuvre et "prêts à l'emploi". Par exemple le module XBee 802.15.4 série 1 (voir ci-contre) permet de créer extrêmement facilement une connexion série sans fil. Son coût aux alentours des 20 euros est relativement faible, ce qui a fait son succès.

Les modules XBee sont développés et commercialisés par la société Digi. Ce sont des circuits de communication sans-fil utilisant les **protocoles 802.15.4 et Zigbee**, permettant de réaliser différents montages, d'une liaison série RS232 classique à un réseau maillé (mesh).

Plusieurs produits XBee existent, ce qui peut créer quelques confusions. Il faut retenir qu'il y a deux catégories de XBee : **la série 1 et la série 2**. La différence se fait principalement sur la topologie du réseau que l'on souhaite réaliser :

Série 1 :

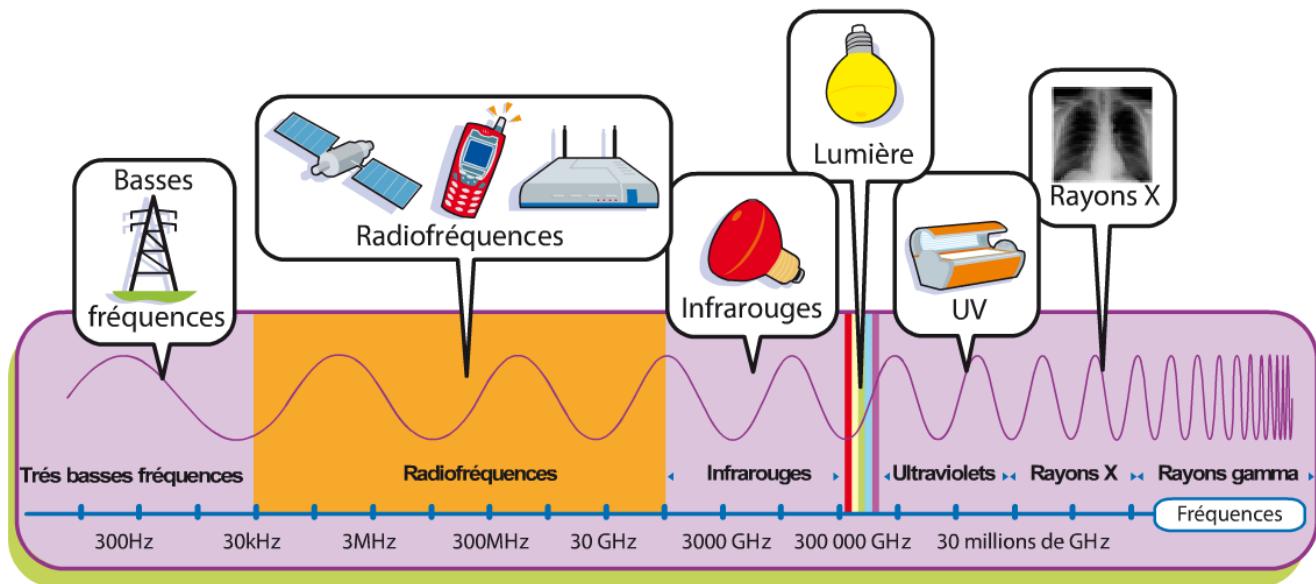
Les modules de la série 1 ont souvent un "802.15.4" qui s'ajoutent à leurs noms. Ils permettent de réaliser des réseaux de topologie : **Point-à-Point, Point-à-Multipoint (Broadcast)**.

Série 2 :

Les modules de la série 2 sont disponibles en plusieurs versions : XBee ZNet 2.5 (obsolète), le ZB (l'actuel) et le 2B (le plus récent). Ils permettent de réaliser des réseaux de topologie : **Point-à-Point, Point-à-Multipoint (Broadcast), Peer-to-Peer, Mesh (Maillé)**.

Vous avez aussi des XBee Pro dans les séries 1 et 2, qui font la même chose mais avec des capacités améliorées, notamment la portée.

Pour en savoir plus, ici vous trouverez le tableau de comparaison des différents modules XBee.



Généralités sur les réseaux sans fil

Bien souvent les ondes électromagnétiques dans le domaine radio sont utilisées pour la transmission des données en milieu aérien. L'onde électromagnétique sert de porteuse à la transmission des données et le codage s'effectue en modulant l'amplitude ou la fréquence de cette porteuse.

Ces ondes ont la particularité de pouvoir pénétrer la matière, mais leur présence en quantité abondante dans l'atmosphère est source de nombreuses perturbations.

Pour éviter les perturbations dans des domaines stratégiques comme l'armée, les secours..., certaines fréquences porteuses sont réservées en fonction des pays. En France l'utilisation des ondes hertziennes comme support de transmission de données est fortement réglementée.

Le protocole 802.15.4

Le protocole [802.15.4](#) de communication défini par l'[IEEE](#) (utilisé par le protocole [ZigBee](#)) est destiné aux réseaux sans fil de la famille des **LR WPAN** (Low Rate Wireless Personal Area Network). Les canaux de fréquences réservés pour cette famille de réseaux sans fils sont les suivants :

- 16 canaux dans la bande de fréquence de 2,4 à 2,4835 GHz
- 10 canaux dans la bande de fréquence de 902 à 928 MHz
- 1 canal dans la bande de fréquence de 868 à 868,6 MHz

Le protocole ZigBee

ZigBee est un protocole permettant la communication entre petits modules radio, à consommation réduite, basée sur la norme [IEEE 802.15.4](#) pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPAN).

Cette technologie a pour but la communication de courte distance telle que le propose déjà la technologie Bluetooth, tout en étant moins chère et plus simple

Comparaison du protocole ZigBee avec d'autres protocoles de communication sans fil tels que Wi-Fi et Bluetooth :

Caractéristiques	Zigbee	Bluetooth	Wi-Fi
IEEE	802.15.4	802.15.1	802.11a/b/g/n/n-draft
Besoins mémoire	4-32 ko	250 ko +	1 Mo +
Autonomie avec pile	Années	Mois	Heures
Nombre de nœuds	65 000+	7	32
Vitesse de transfert	250 kb/s	1 Mb/s	11-54-108-320 Mb/s
Portée (environ)	100 m	10 m	300 m

Caractéristiques des modules XBee série 1

Le choix du modèle XBee peut se faire en fonction de la portée, de la topologie du réseau que l'on souhaite réaliser, du taux de transmission...

Pour l'utilisation que nous en aurons en STI2D, les modèles de la série 1 suffiront normalement, sauf cas particulier. Voici donc ci-dessous les caractéristiques de la version standard ainsi que la version Pro (*Attention les portées indiquées dépendent aussi du type d'antenne choisie*).

Données techniques

Modèle	XBee	XBee-PRO
Portée en intérieur	Jusqu'à 30 mètres	Jusqu'à 100 mètres
Portée en extérieur	Jusqu'à 100 mètres	Jusqu'à 1500 mètres
Puissance de sortie	1 mW	60 mW
Possibilité de régler la puissance de sortie de façon logicielle	Non	Oui
Débit des données RF	250 000 bps	250 000 bps
Débit des données UART	De 1200 bps à 115 200 bps	De 1200 bps à 115 200 bps
Fréquence d'émission	2,4 GHz	2,4 GHz
Sensibilité	-92 dBm	-100 dBm
Tension d'alimentation	De 2,8 V à 3,4 V	De 2,8 V à 3,4 V
Courant consommé lors d'une émission	45 mA à 3,3 V 10 dBm : 137 mA à 3,3 V 12 dBm : 155 mA à 3,3 V 14 dBm : 170 mA à 3,3 V 16 dBm : 188 mA à 3,3 V 18 dBm : 215 mA à 3,3 V	50 mA à 3,3 V
Courant consommé lors d'une attente ou d'une réception	50 mA à 3,3 V	50 mA à 3,3 V
Courant consommé en mode « SLEEP »	Moins de 10 µA	Moins de 10 µA

Types d'antennes

- **Wire** : Simple, radiations omnidirectionnelles.
- **Chip** : Puce plate en céramique, petite, transportable (pas de risques de casser l'antenne), radiations cardioïdes (le signal est atténué dans certaines directions).
- **U.FL** : Connecteur miniature permettant de raccorder des antennes. Une antenne externe n'est pas toujours nécessaire.
- **RPSMA** : Plus gros que le connecteur U.FL, permet de placer son antenne à l'extérieur d'un boîtier.

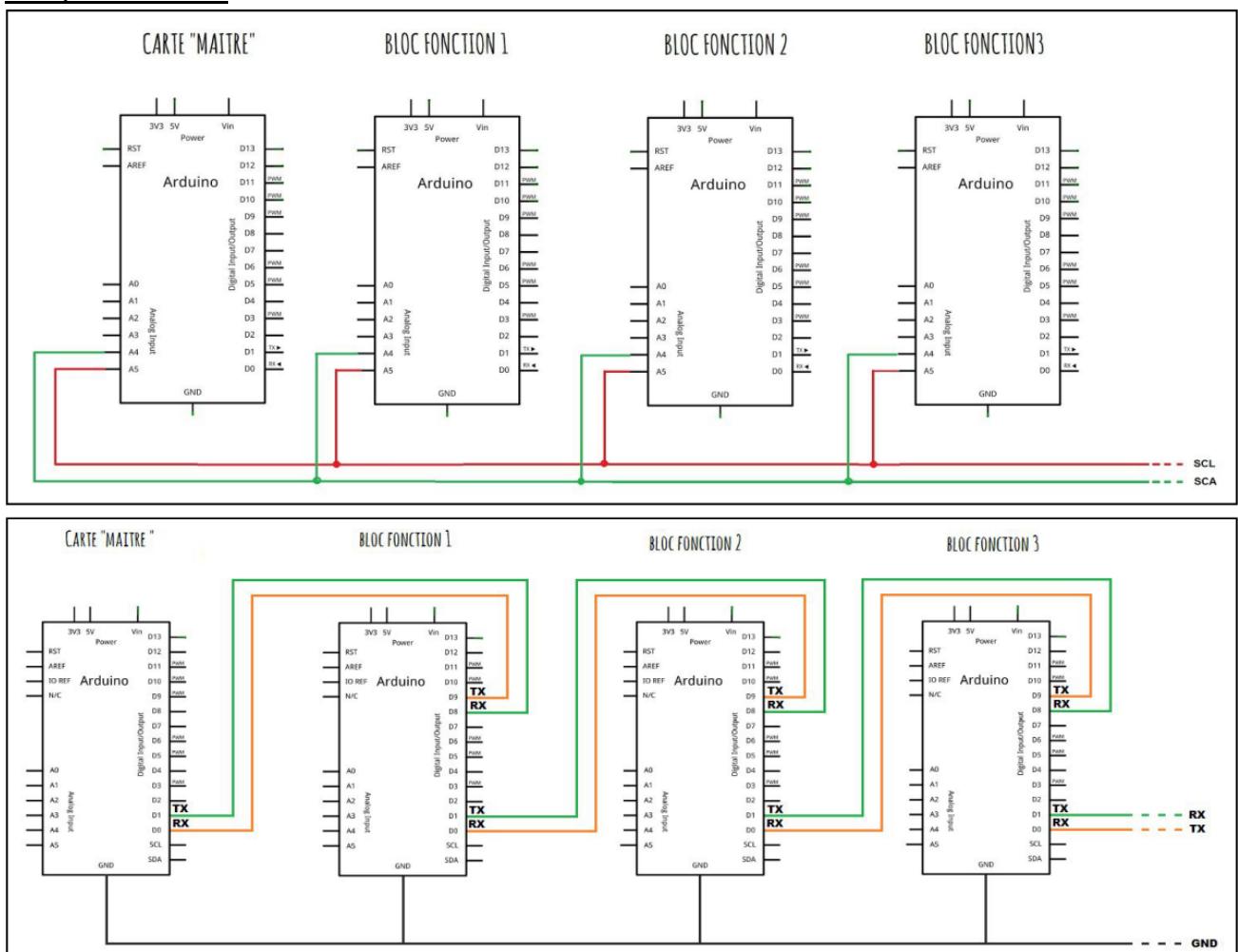
1.2) Communication série :

Ce que l'on veut faire : Faire communiquer les blocs programmables entre eux ainsi qu'avec le bloc maître.

Problématique : Comment savoir quel bloc parle lors de la communication et où il se situe par rapport aux autres blocs ?

Hypothèse : Il faut établir une liaison physique entre les blocs. Il faut également désigner un bloc maître de la discussion qui gère la communication et qu'aucune donnée ne se perde.

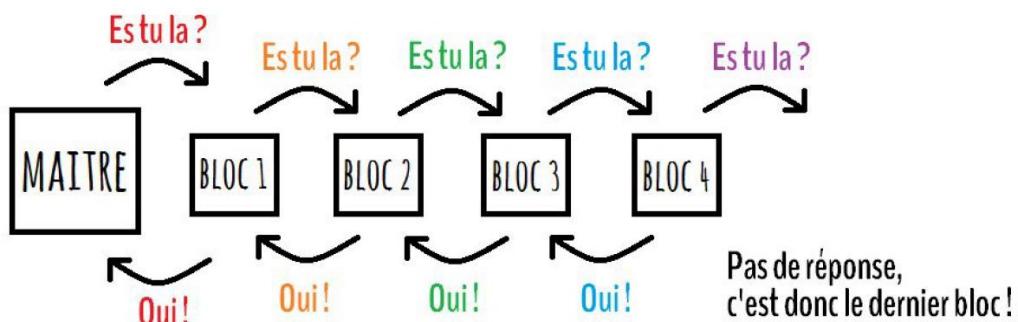
Les possibilités :



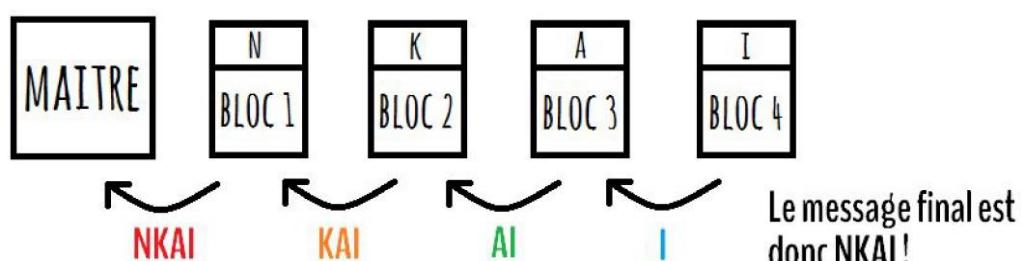
La stratégie :

Une stratégie a été imaginée en deux étapes afin de réaliser la communication :

La première est de compter les blocs. L'informatique n'a pas d'oeil, ou du moins notre système n'en a pas. Pour savoir le nombre de bloc le système va jouer au "téléphone arabe", un message va passer de bloc en bloc à partir du bloc maître jusqu'à arriver au dernier bloc qui n'aura plus personne à qui parler. Pour savoir s'il y a un bloc "2" à côté de lui, le bloc "1" va demander s'il y a quelqu'un à côté de lui. S'il y a un bloc "2" alors la réponse sera "oui" sinon aucune réponse ne sera reçue. Ce bloc sera alors le dernier bloc.



La seconde étape consiste à connaître la fonction de chaque bloc. Le "téléphone arabe" va repartir dans l'autre sens, cette fois ci, du dernier bloc au bloc maître. Chaque bloc va répéter à un autre la séquence qu'on lui aura chuchotée en y ajoutant sa propre fonction (traduit en lettre). De bouche à oreille, l'information finira par arriver au bloc maître qui aura alors toute les informations nécessaires.



II) Réalisation :

2.1) Programmation Xbee :

Algorigrammes :

Le fonctionnement du bloc maître et celui du bloc programmable est légèrement différent. Les deux étapes de chaque bloc ont été représentées en quatre algorigrammes.

Etape 1 - Bloc maître :

La communication se déclenche lors de l'appui sur le bouton central du bloc maître par l'enfant. Cela signifie que la séquence est terminée d'être construite. Dès lors le

bloc maître envoie la lettre "Z" qui est le code simplifié de "Es-tu là ?" en attendant la réponse "Y" qui est le code simplifié de "Oui !" (cf. schéma 1 p11). Cela signifie qu'il y a bien un bloc à côté. L'étape 1 est terminée et on peut passer à l'étape 2.

Cependant si l'on ne reçoit rien, cela signifie qu'il n'y a pas de bloc à côté et donc que le bloc maître est seul. L'appui sur le bouton est sûrement une erreur, on redémarre la carte.

Etape 1 - Bloc programmable :

Le bloc programmable attend qu'on le contacte sur son entrée, c'est à dire par le bloc situé avant lui. Il attend la fameuse question ("Z"->"Es-tu là?") pour y répondre ("Y" -> "Oui !"), avant d'effectuer à son tour la question au bloc situé après lui. S'il reçoit la réponse "Y" cela signifie qu'il y a bien un bloc après lui. L'étape 1 est donc terminée avec la première variante. Cependant s'il ne reçoit rien cela signifie qu'il est le dernier bloc de la séquence. L'étape 1 est donc terminé avec la seconde variante.

Etape 2 - Bloc programmable :

Le programme est adapté aux deux variantes et attend juste de savoir laquelle effectuer. Cette décision est prise lors de l'étape 1.

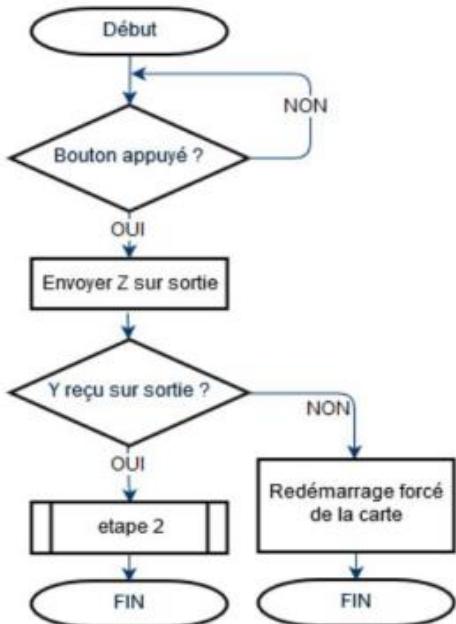
Variante 1 : Lors de la variante 1, cela signifiait que le bloc n'était pas le dernier de la séquence. Il attend donc de recevoir les informations du bloc situé après lui, pour les stocker tant qu'il en reçoit encore puis de les renvoyer au bloc situé avant lui, sans oublier d'ajouter sa propre fonction (cf. schéma 2 p11). Son rôle est à présent terminé et la carte redémarre afin de se réinitialiser.

Variante 2 : Lors de la variante 2, cela signifiait que le bloc était le dernier de la séquence. C'est donc à lui de commencer à envoyer sa fonction au bloc situé avant lui. Il ne recevra pas d'information car il n'y a pas de blocs situés après lui. Son rôle est à présent terminé et la carte redémarre afin de se réinitialiser.

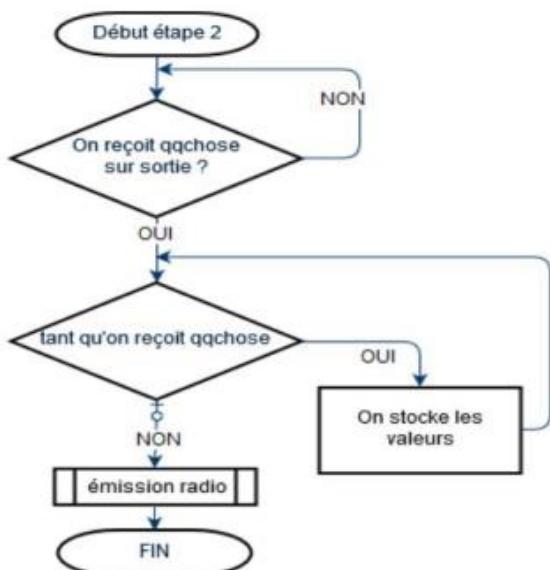
Etape 2 - Bloc maître :

Le bloc maître est dans l'attente de recevoir l'information finale, celle combinant les fonctions de tous les blocs réunis. Il va stocker les valeurs tant qu'il en reçoit puis une fois faite pourra commencer l'émission radio afin de lui transmettre les informations.

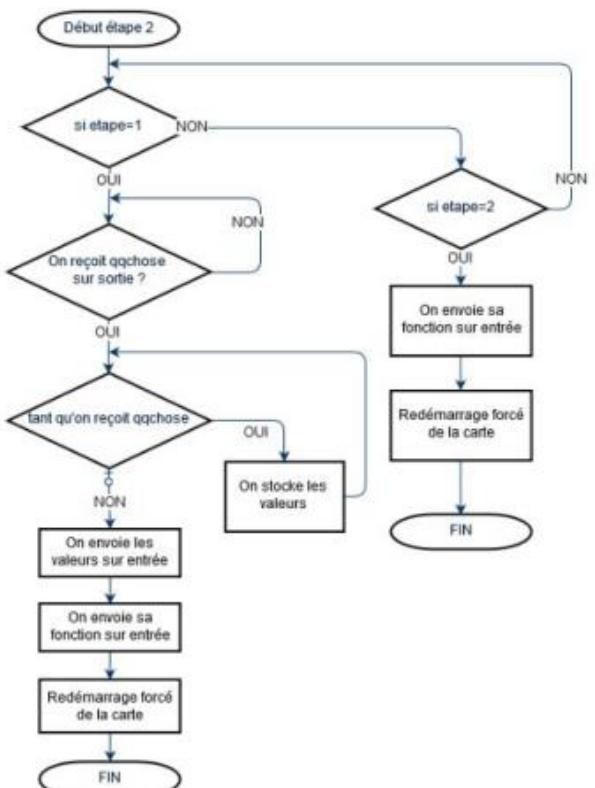
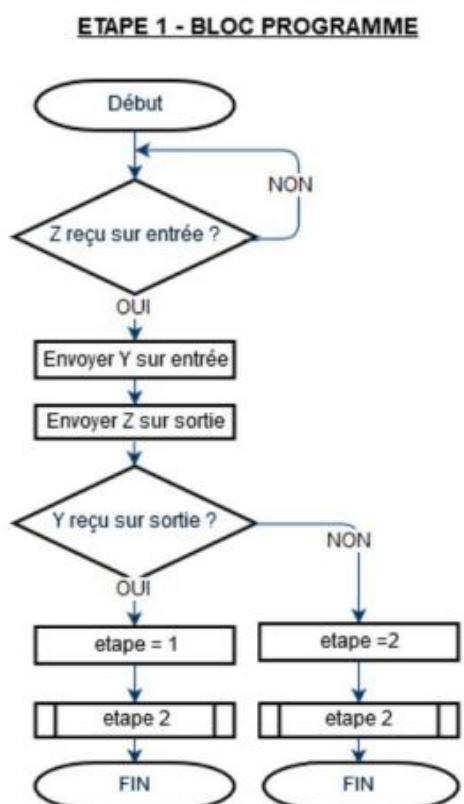
ETAPE 1 - BLOC MAITRE



ETAPE 2 - BLOC MAITRE



ETAPE 2 - BLOC PROGRAMME



Le prototypage :



Les tests :

Le logiciel Arduino embarque un moniteur série qui permet d'interagir avec son programme, notamment afin de surveiller son déroulement. Des lignes de codes ont donc été insérés dans le programme afin de faire apparaître à chaque étape importante, du texte sur le moniteur série permettant ainsi de se situer dans l'exécution du programme.

Ordre de communication :

Étape 1 :

B.M :

“Attente de l'appui sur le bouton” : Le bloc maître attend que l'utilisateur appuie sur le bouton.

B.M :

“Envoie de la lettre Z” : le bouton a été appuyé, la lettre Z a été envoyé au bloc d'à côté (bloc 1).

B.1 :

“On ne reçoit rien” : Le Z n'a pas encore été reçu.

B.1 :

“Lecture de la lettre Z” : La lettre Z a été reçue du bloc précédent pour savoir si un bloc est là.

B.1 :

“Réponse par la lettre Y” : Le bloc est là, on répond par Y pour signaler la présence.

B.M :

“Lecture de la lettre Y” : Un bloc est à côté, en attente.

B.1 :

“Envoie de la lettre Z” : Le bouton a été appuyé, la lettre Z a été envoyée au bloc d'à côté (bloc 2).

// même opération jusqu'à arriver au bloc 3

B.3 :

“Envoie de la lettre Z” : Le bouton a été appuyé, la lettre Z a été envoyée au bloc d'à côté (bloc 4).

B.3 :

“Pas de réponse” : Cela signifie qu'aucun bloc n'est situé à côté.

Etape 2 :

B.3 :

“Etat capteur” : On regarde sa fonction (voir tâche de Sullivan).

B.3 :

“Envoie de ma fonction” : On envoie sa fonction traduit par une lettre comprise par tous les blocs.

B.3 :

“Redémarrage du bloc”

: Le bloc a fini son rôle il peut redémarrer afin de se réinitialiser

// voir bloc 1

B.1 :

“On reçoit quelque chose” : On reçoit les fonctions du bloc précédent.

B.1 :

“Etat capteur” : On regarde sa fonction (voir tâche de Sullivan).

B.1 :

“Envoie du caractère du bloc” : On envoie la fonction des autres blocs.

B.1 :

“Envoie de ma fonction” : On envoie sa fonction traduite par une lettre comprise par tous les blocs.

B.1 :

“Redémarrage du bloc”

: Le bloc a fini son rôle il peut redémarrer afin de se réinitialiser.

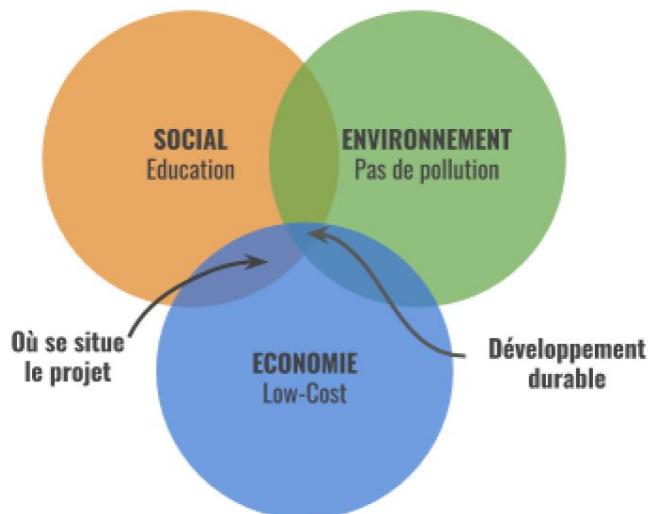
// même opération jusqu'à arriver au bloc maître qui relaie l'information au robot

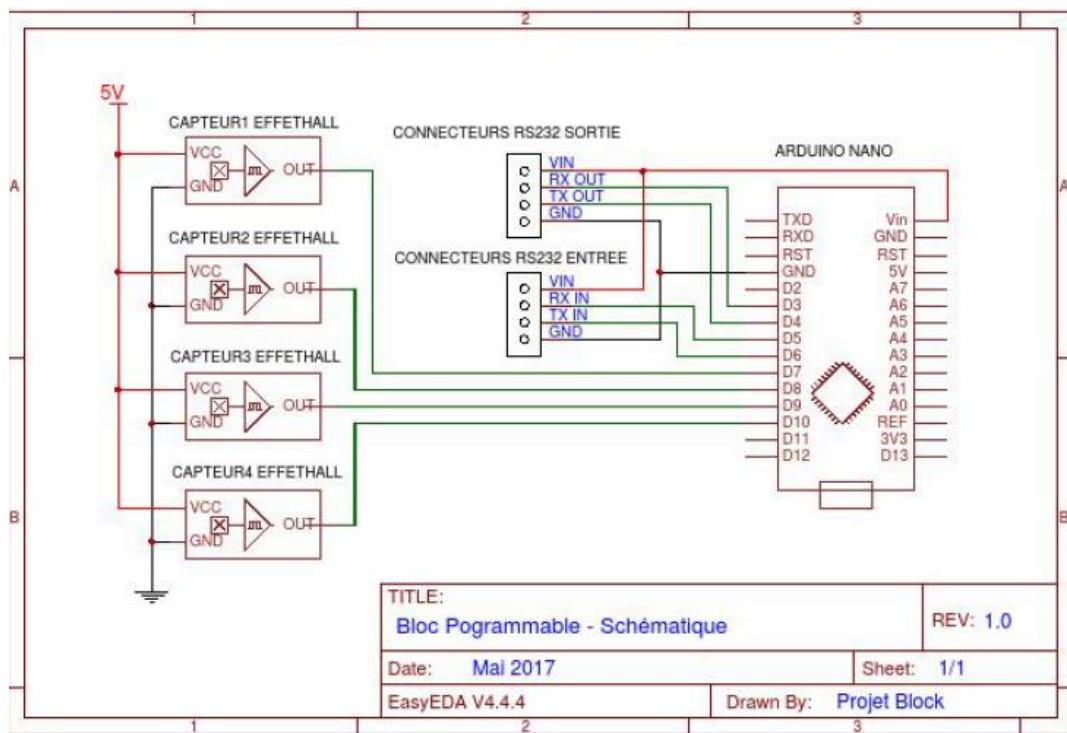
Légende : B.M = Bloc maître ; B.1 = Bloc 1 ; B.2 = Bloc 2 ; B.3 = Bloc 3

Synthèse :

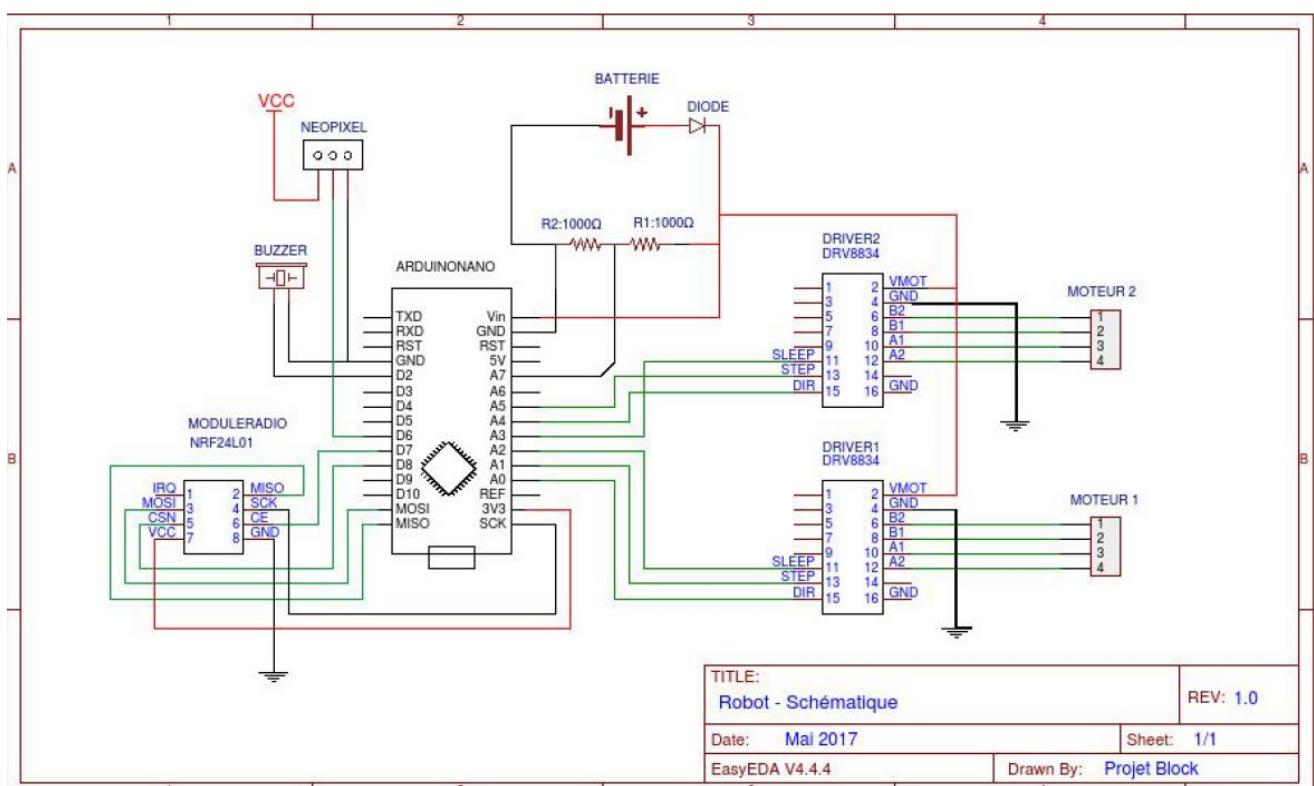
<p><<requirement>> Communiquer</p> <p>text="Le bloc assure deux communication. Il dialogue avec le bloc précédent ou le bloc maître, et le bloc suivant." Id="3.4"</p>	<p><<requirement>> Communiquer avec le BlocProg</p> <p>text="Il communique avec le ou les blocProg " Id="1.2"</p>
---	--

Ainsi l'exigence de départ : Faire communiquer les blocs programmables entre eux ainsi qu'avec le bloc maître a bien été validé et la problématique : « Comment savoir quel bloc parle lors de la communication et où il se situe par rapport aux autres blocs ? » a été résolue. Malgré des problèmes de synchronisation des blocs (l'un envoyait les informations mais l'autre n'était pas prêt à écouter), tout a été résolu et est maintenant fonctionnel.

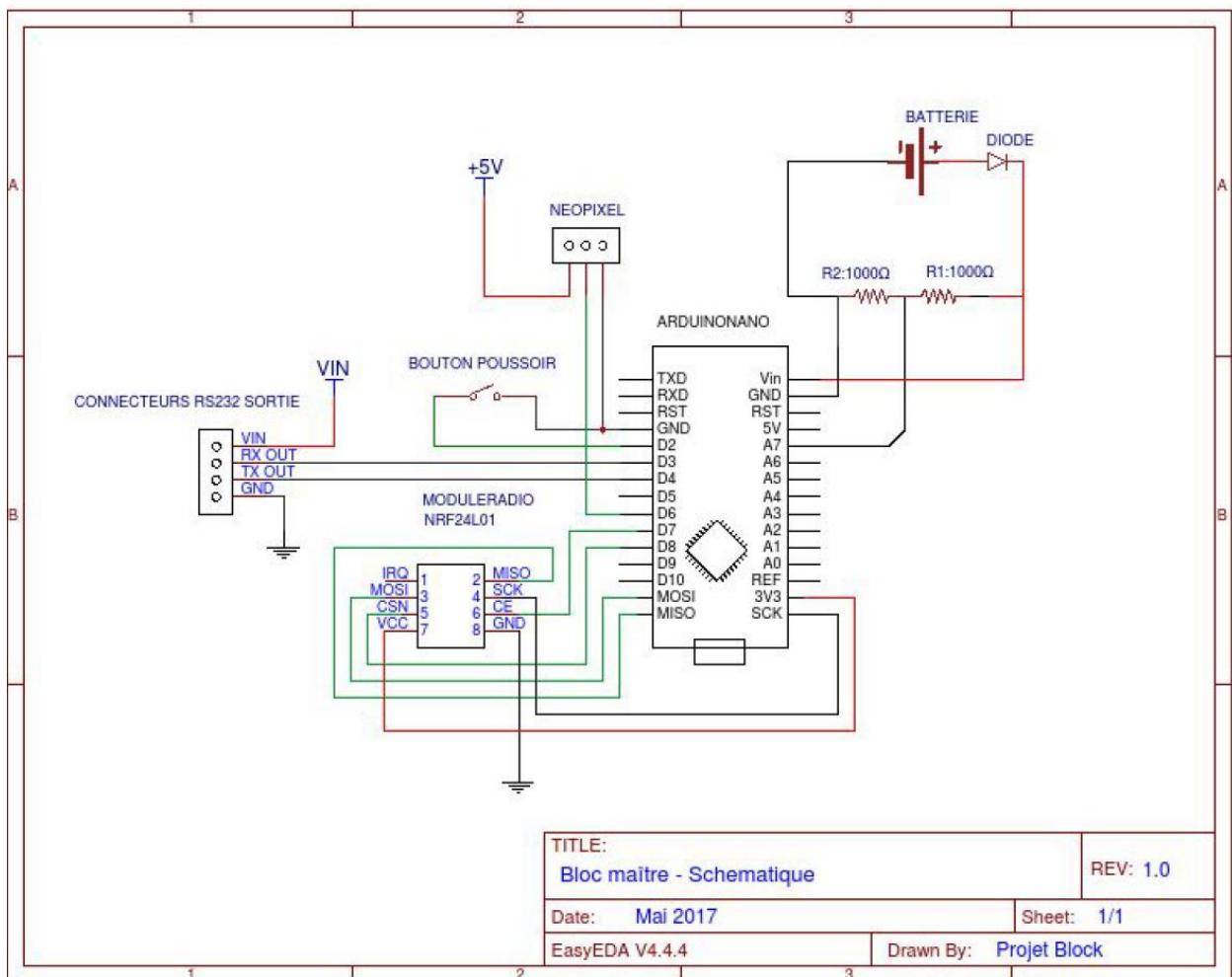




Bloc programmable – Schéma structurel



Robot – Schéma structurel



Bloc maître – Schéma structurel

4. PARTIE INDIVIDUELLE YOAN GIANINO

4.1 Présentation générale du système supportant le projet :

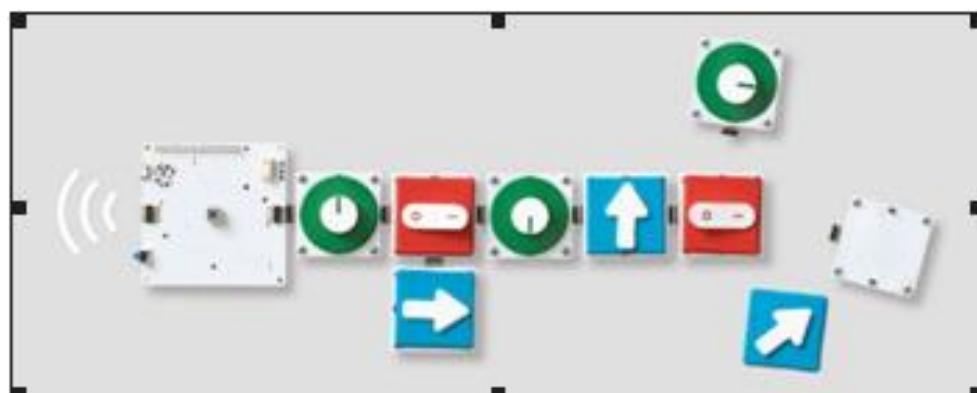
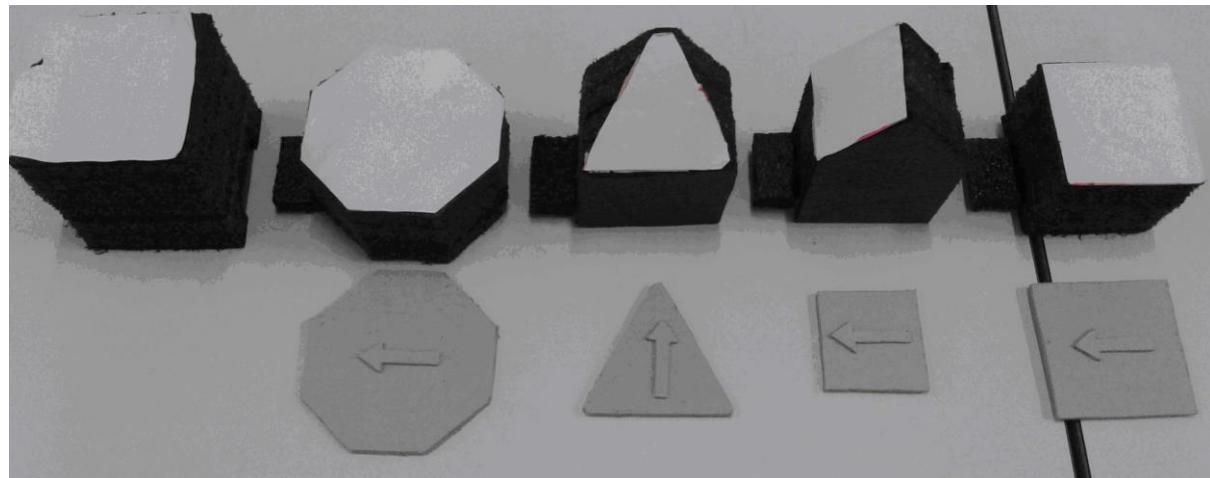
La problématique est la suivante : "concevoir un système permettant à des enfants âgés de 6 ans ou plus, sachant lire ou non, pouvant avoir des difficultés motrices, d'apprendre les bases de la programmation et de l'algorithme".

Expression du besoin :

Créer un système similaire à la programmation sous Scratch, avec des blocs universels pouvant être configurés pour effectuer différentes instructions, par exemple des actions ou des boucles. Les étiquettes de Scratch sont remplacées par des afficheurs LED.

Le système doit pouvoir piloter des parties opératives diverses par l'intermédiaire d'un PC ou d'un Smartphone. C'est un projet existant d'un lycée de Nantes et de Google.

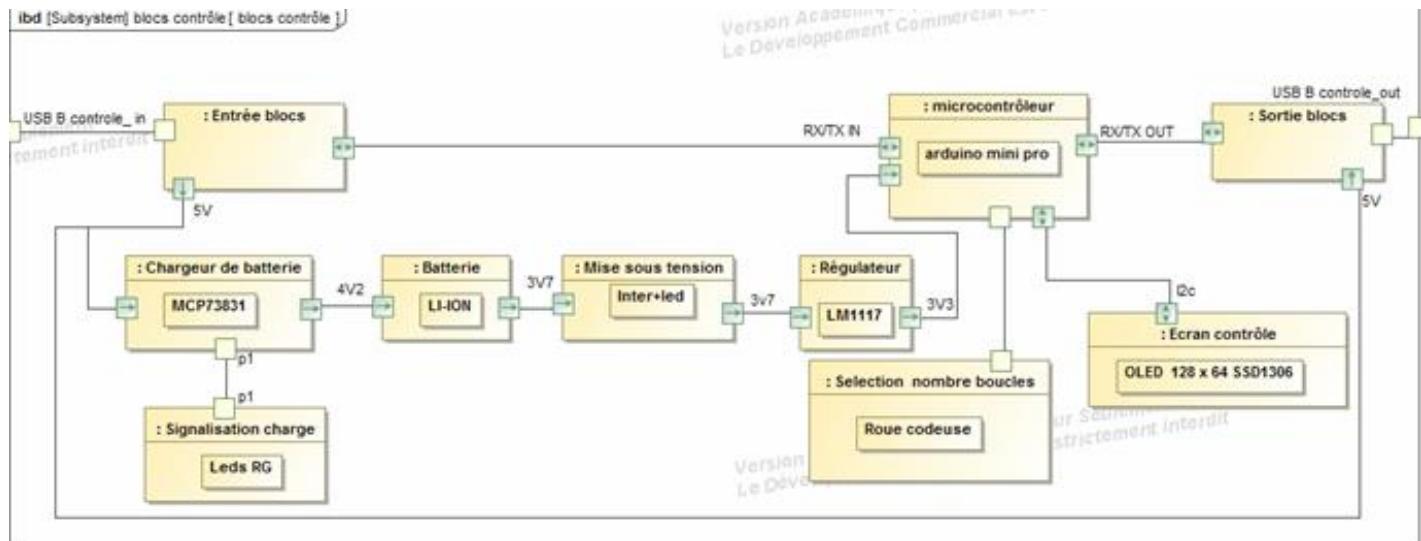
Je devrais créer un équivalent de Scratch programmable avec des blocs physique.



Source : Google

4.2 Étudiant 4 blocs fonction

Structure fonctionnelle



Structure architecturelle

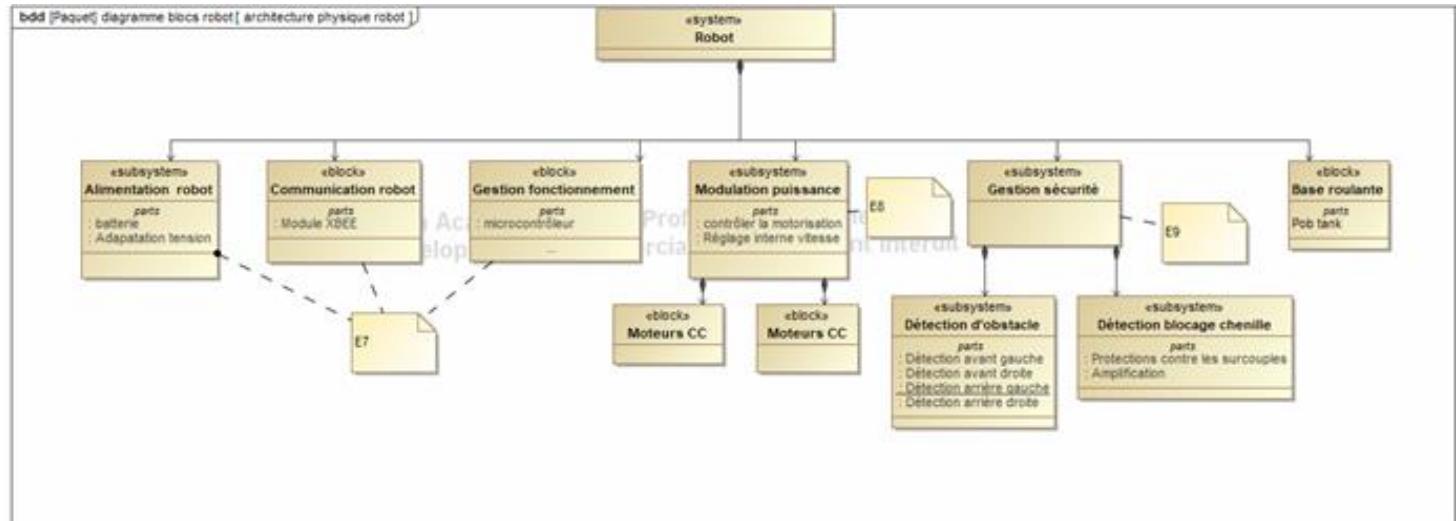
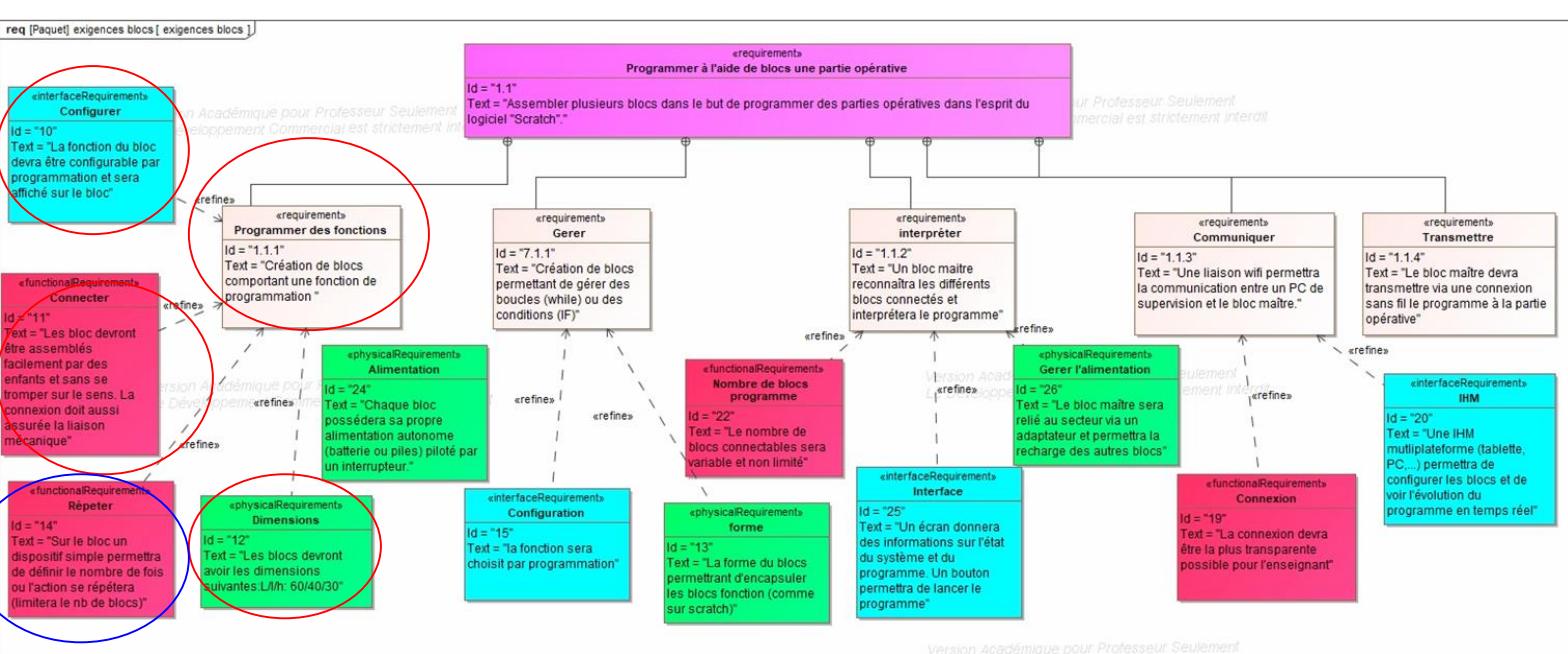


Diagramme donné par les professeurs :



4.3 Cahier des charges

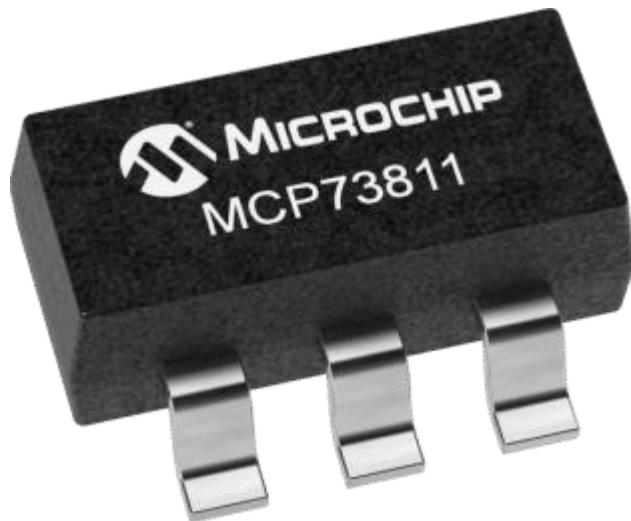
- Les blocs doivent être assemblés facilement par des enfants, sans se tromper sur le sens. La connexion doit aussi assurer la liaison mécanique.
 - Sur le bloc un dispositif permet de définir le nombre de fois où l'action se répétera.
 - Les blocs doivent avoir les dimensions suivantes L/l/h : 60/40/30mm
 - Les blocs possèdent leurs propres alimentations, pilotés par un interrupteur
 - La fonction d'un bloc est choisie par la programmation
 - Le nombre de blocs connectables est variable et non limité
 - La forme des blocs permettant d'encapsuler les blocs fonctions (comme sur scratch)
 - Un écran donne des informations sur l'état du système, un bouton permet de lancer le programme
 - La connexion doit être la plus transparente possible pour l'enseignant

4.4 Planning

PLANNING POUR YOAN GIANINO PROJET ROBOT BLOCS					
mois					
janvier	février	mars		avril	mai
	continuation de Eagle	trouver des solutions pour la programmation sous PIC 16f.			juin
	continuation de Eagle	Changement passe sous Arduino fini modification eagle essais sur platine	travail sur le programme finale	mise en commun avec l'étudiant 1	travail sur le rapport
			travail sur le programme finale	mise en commun avec l'étudiant 1	travail sur le rapport
		(faire des tests de programme pour afficher des images sur ecran)			finit et rend le rapport
	Première revue de projet	(revue de projet 2 + programmer)	travail sur le programme finale	travail sur la carte	
			travail sur le programme finale	travail sur la carte	
répartition des taches	fini eagle	soudure	modification de la carte	travail sur le programme	
		soudure	modification de la carte	travail sur le programme	
élaboration du cahier des charges en accords avec les contraintes du projet					
		soudure	modification de la carte		
		soudure	modification de la carte		
recherches des datasheets					
Etude des composant utilisé à la partie matériel					
cahier des charges sous formes de blocs début de l'IBD		commencement du rapport finale	écriture du rapport finale		
		écriture du rapport	travail sur la carte		
Finition de l'IBD Commencement de Eagle		écriture du rapport	travail sur la carte		
continuation de Eagle				travail sur le rapport	
				travail sur le rapport	
				travail sur le rapport	
		continuation d'écriture de programme (cause IR amélioration code)			
		continuation d'écriture de programme (cause IR amélioration code)			

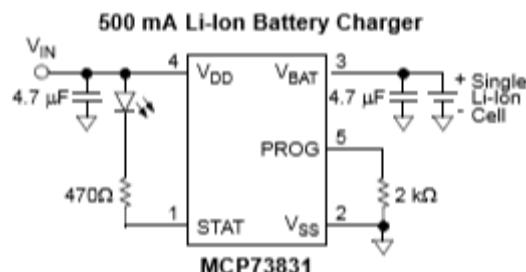
4.5 Choix des composants

Le MCP73811 :

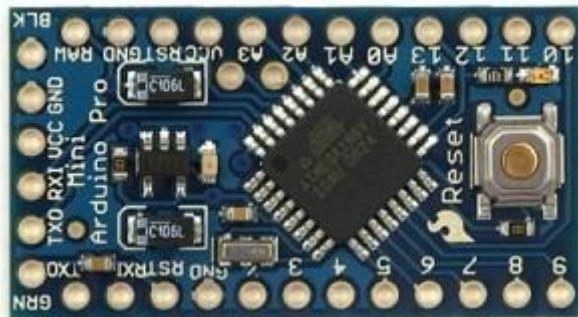


J'ai choisi d'utiliser ce composant car :

Il convertit du 5V en 4.2V, qui est la tension de charge de ma batterie. Cette batterie sert à alimenter mon système.



Carte Arduino mini pro :



J'ai choisi ce composant car :

- Il est de petite taille
- Il est alimenté en 3,3V
- Il possède 2 Ko de RAM afin d'exécuter les programmes

USB-B :



J'ai choisi l'USB-B car il est solide et peut donc résister à des manipulations par de petits enfants. Par ailleurs les enfants ne pourront pas se tromper de sens pour l'emboîter. L'USB-B possède 4 broches, le RX et TX pour le transfert de données, ainsi que le 5V (VCC) et GND pour l'alimentation.

Ecran OLED 128/64 SSD 1306 :



Cet écran possède 8192 pixels.

Ses dimensions sont de 128 par 64 pixels.

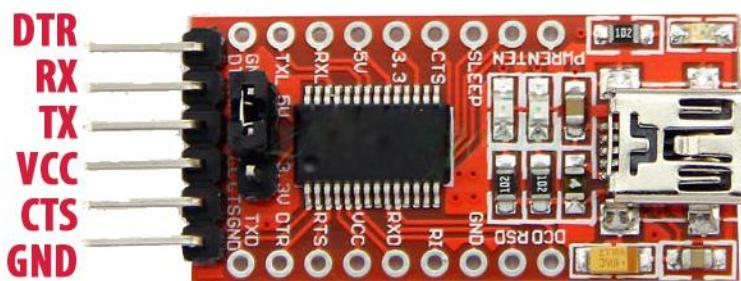
Il est programmable en I2C.

Cet écran, de petite taille, suffit pour afficher ce dont l'on a besoin.

Il possède 4 broches : SDA, SCL pour la communication et enfin VCC et GND pour l'alimentation.

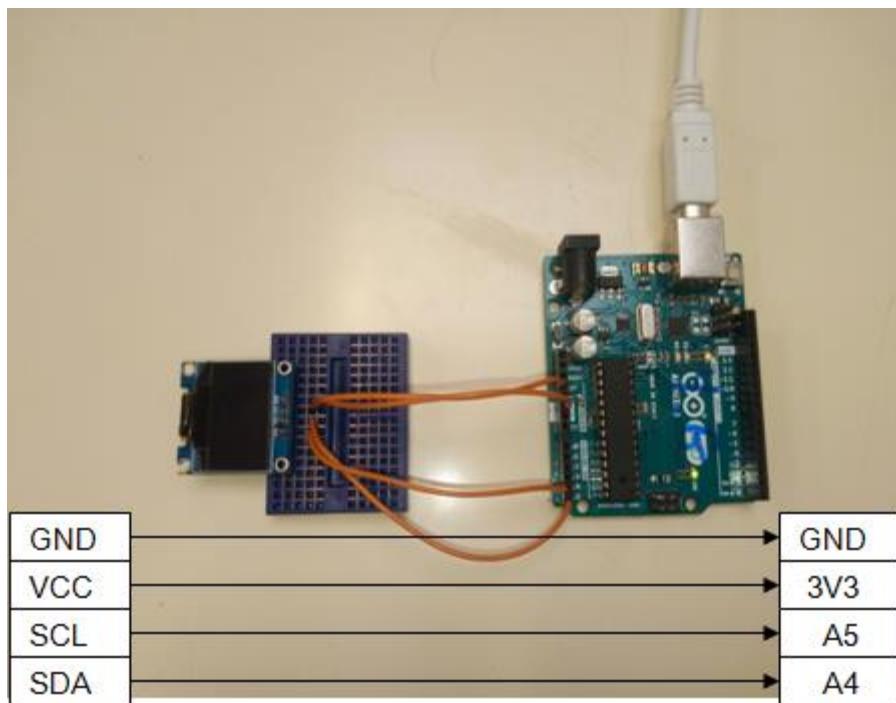
Il est alimenté en 3,3V

FTRL232 :



Ce composant permet de communiquer entre le PC et la carte Arduino. On peut ainsi la programmer.

4.5 Principe et branchement



J'ai réalisé un programme qui me permet d'afficher des flèches sur l'écran en écrivant une lettre dans le moniteur série du logiciel Arduino :

Par exemple la lettre « A » désigne une flèche orientée vers le haut.

```
static const unsigned char PROGMEM logo16_glcd_bmp[] = //nom de la flèche du haut
{B00000000,B00000000, B00011000,B00000000,B00000000,
 B00000000,B00000000, B00111100,B00000000,B00000000, //les bitmap sont fait en
binaire sur plusieurs octets
B00000000,B00000000, B01111110,B00000000,B00000000,
B00000000,B00000000, B11111111,B00000000,B00000000,
B00000000,B00000001, B11111111,B10000000,B00000000,
B00000000,B00000011, B11111111,B11000000,B00000000,
B00000000,B00000111, B11111111,B11100000,B00000000,
B00000000,B00001111, B11111111,B11110000,B00000000,
B00000000,B00011111, B11111111,B11111000,B00000000,
B00000000,B01111111, B11111111,B11111110,B00000000,
B00000000,B11111111, B11111111,B11111111,B00000000,
B00000001,B11111111, B11111111,B11111111,B10000000,
B00000011,B11111111, B11111111,B11111111,B11000000,
B00000111,B11111111, B11111111,B11111111,B11100000,
B00001111,B11111111, B11111111,B11111111,B11110000,
B00000000,B11111111, B11111111,B11111111,B00000000,
```

```

B00000000,B11111111, B11111111,B11111111,B00000000,
B00000000,B11111111, B11111111,B11111111,B00000000,
B00000000,B11111111, B11111111,B11111111,B00000000,
B00000000,B11111111, B11111111,B11111111,B00000000,
B00000000,B11111111, B11111111,B11111111,B00000000,
B00000000,B11111111, B11111111,B11111111,B00000000,
B00000000,B11111111, B11111111,B11111111,B00000000 };

static const unsigned char PROGMEM logo17_lcd_bmp[] = // nom de la flèche de droite

```

//suite des bitmaps

... //voir [annexe](#)

//termine ici

```

#if (SSD1306_LCDHEIGHT != 64)
#endif
int received;

void setup() {
  Serial.begin(9600); // vitesse de transfert en 9600 bauds

  //Utiliser le scanner I2C pour trouver le port série sur lequel se trouve votre écran
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C //initialisation de l'I2C
  display.clearDisplay(); // réinitialisation de l'écran
  display.clearDisplay();
  testscrolltext(); // j'appelle mon programme de test écrit en dessous

}

void loop() {

  reception(); //j'appelle mon programme réception, pour qu'il se lance en boucle

}

int reception()
{

```

```

if (Serial.available()>0)

    received = Serial.read();
    if (received == 'a') // quand on écrit la lettre <a> dans le moniteur série d'Arduino il nous
affiche sur l'écran la flèche du haut
    {
        display.clearDisplay();
        display.drawBitmap(45, 7, logo16_glcd_bmp, 40, 23, 1); //cette ligne sert à afficher un
bitmap (x,y, nom du logo, l,L, couleur)
        display.display();
        delay(200);

    }

    if (received == 'g')//quand on écrit la lettre <g> dans le moniteur série d'Arduino il nous
affiche sur l'écran la flèche de gauche
    {
        display.clearDisplay();
        display.drawBitmap(22, 7, logo18_glcd_bmp, 80, 23, 1);
        display.display();
        delay(200);

    }

    if (received == 'd')//quand on écrit la lettre <d> dans le moniteur série d'Arduino il nous
affiche sur l'écran la flèche de droite
    {
        display.clearDisplay();
        display.drawBitmap(32, 7, logo17_glcd_bmp, 80, 23, 1);
        display.display();
        delay(200);

    }

    if (received == 'b')//quand on écrit la lettre <b> dans le moniteur série d'Arduino il nous
affiche sur l'écran la flèche du bas
    {
        display.clearDisplay();
        display.drawBitmap(39, 7, logo19_glcd_bmp, 40, 23, 1);

```

```

        display.display();
        delay(200);

    }

else
{

    display.clearDisplay();

}

return(0);
}

void testscrolltext(void) {
    display.setTextSize(2); //taille de l'écriture
    display.setTextColor(WHITE); //couleur de l'écriture (blanc)
    display.setCursor(10,0); // (x,y) la position où doit se positionner le curseur
    display.clearDisplay();
    display.println("robot    blocs"); //écriture du texte
    display.display();

}

```

Nous devons brancher la broche SDA de l'écran sur la broche A4 du microcontrôleur, le SCL se branche sur la broche A5, enfin le VCC sur le 3V3 et le GND sur le GND.
(Pour mes tests, j'ai utilisé une carte Arduino uno, qui se comporte de la même manière que la carte Arduino Mini Pro).

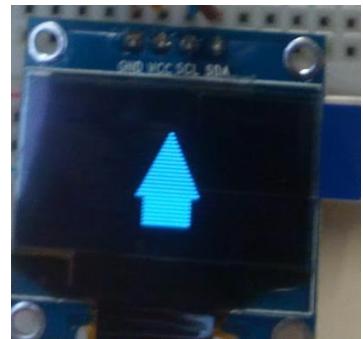
4.6 Relever des tests

Qu'est-ce que un fichier Bitmap ?

Nous allons voir comment est fait un fichier Bitmap et ce qu'il contient.

Je viens de créer le fichier bitmap ci-dessous. Il représente une flèche. Il est composé de 0 et de 1

```
B00000000,B00000000, B00011000,B00000000,B00000000,  
B00000000,B00000000, B00111100,B00000000,B00000000,  
B00000000,B00000000, B01111110,B00000000,B00000000,  
B00000000,B00000000, B11111111,B00000000,B00000000,  
B00000000,B00000001, B11111111,B10000000,B00000000,  
B00000000,B00000011, B11111111,B11000000,B00000000,  
B00000000,B00000111, B11111111,B11100000,B00000000,  
B00000000,B00001111, B11111111,B11110000,B00000000,  
B00000000,B00011111, B11111111,B11111000,B00000000,  
B00000000,B00111111, B11111111,B11111100,B00000000,  
B00000000,B01111111, B11111111,B11111110,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000001,B11111111, B11111111,B11111111,B10000000,  
B00000011,B11111111, B11111111,B11111111,B11000000,  
B00000111,B11111111, B11111111,B11111111,B11100000,  
B00001111,B11111111, B11111111,B11111111,B11110000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000
```



Ma partie est en lien avec l'étudiant 1

L'étudiant 1 m'envoie un fichier bitmap. Je dois le réceptionner, le ranger dans un tableau puis l'afficher sur l'écran de ma carte. Pour cela j'ai utilisé des fonctions bien définies dans mon programme.

Pour réaliser le projet, j'ai dû faire les manipulations suivantes :

Pour la communication entre l'IHM et la carte, on fixe le débit de transmission à 1200 bauds. Nous avons choisi une faible vitesse de transfert afin de réduire les potentiels pertes de données.

Voilà la ligne de code correspondante :

```
void setup() {  
    Serial.begin(1200); // vitesse de transmission de 1 200 bauds pour le port série*  
}
```

Comme dit précédemment j'ai dû réceptionner le fichier bitmap. Pour cela nous avons défini un caractère de « start » dans la trame afin de pouvoir chercher un symbole précis dans la trame.

Début Trame	Séparateur	Identifiant Action	Séparateur	1ière valeur image ... dernière valeur image
-------------	------------	--------------------	------------	--

Le caractère de début de trame est « A ».

Voilà les lignes de code correspondantes, pour la réception du fichier bitmap :

```
while(i<516) // tant i (les donnée du bitmap) est inférieur à 516 il cherche des données
```

```
{  
    if (Serial.available()>0)  
    {  
        if(i<4)  
        {  
            trame[i] = Serial.read(); // Il doit afficher les données reçus du bitmap sur l'écran  
        }  
        else{  
  
            bitmap[y] = Serial.read();  
            y++;  
        }  
    }  
}
```

```
    }  
  
    i++;  
}  
}
```

Pour identifier le début du fichier bitmap, nous avons décidé que ce sera la lettre "A"

Pour afficher le fichier bitmap, nous avons juste besoin d'une fonction qui est la suivante :

```
display.clearDisplay(); //cette ligne permet d'effacer tout ce qu'il a sur l'écran  
display.display();  
display.drawBitmap(x, y, nom, l, L, couleur); // cela permet de positionner l'image sur l'écran  
display.display();
```

Explication de la manipulation à faire :

Il faut téléverser le programme suivant, via le logiciel Arduino, dans la carte, à l'aide du FTRL 232 :

Dans un programme nous trouvons toujours en entête les librairies ainsi que les instructions pour l'écran ssd1306, ici j'utilise quatre librairies :

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

int i,y;           // i est pour le tableau du début de trame (les fonctions)
char fonction_du_block;
byte bitmap[512];    // déclaration du tableau      Byte nom[nombre d'octets]
accueil la trame de la flèche
char trame[4] ;       // déclaration du tableau      Char nom[nombre d'octets]
accueil le début de trame pour les fonctions

void setup() {
```

```

Serial.begin(1200); // vitesse de transmission de 1 200 bauds pour le port série

display.begin(SSD1306_SWITCHCAPVCC, 0x3C); // initialisation de l'I2C
display.clearDisplay(); // reinitialisation de l'écran
display.display();
for (i=0; i<500;i++) bitmap[i] = 0xFF; // mettre un écran noir

i=0;
}

void loop()
{
    i=0 ;
    y=0;

    while(i<516) // tant i (les donnée du bitmap) est inférieur à 516 il cherche des données

    {
        if (Serial.available()>0)
        {
            if(i<4)
            {
                trame[i] = Serial.read(); //il doit afficher les données reçut du bitmap sur l'écran
            }
            else{

                bitmap[y] = Serial.read();
                y++;
            }

            i++;
        }
    }
}

```

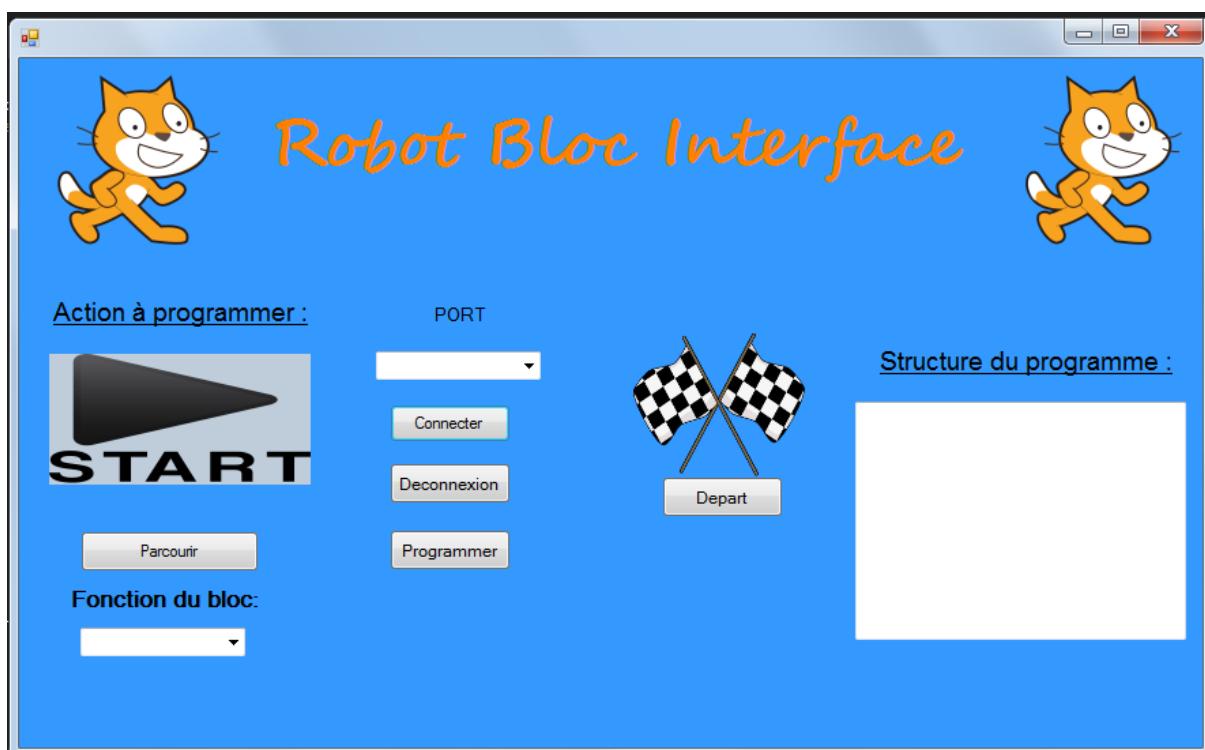
```

fonction_du_bloc = trame[2];
display.clearDisplay();
display.display();
display.drawBitmap(0, 0, bitmap, 128, 32, 1); // c un bitmap (x,y, nom, l,L, couleur)
display.display();

delay(5000);
while(Serial.available()>0)
Serial.read();
}

```

Une fois la manipulation faite nous devons passer sur L'IHM



Nous devons sélectionner un fichier Bitmap à envoyer. Pour cela il faut cliquer sur "parcourir"

J'ai réalisé 4 fichiers bitmap correspondant à avancer, reculer, droite et gauche.

Les dimensions des bitmaps sont 128 par 32 pixels. Il faut savoir que les dimensions de l'écran sont 128 par 64 pixels, mais nous rencontrons un problème. La mémoire RAM de l'arduino mini pro est insuffisante pour accueillir des fichiers bitmap de 128 par 64 pixels, donc j'ai choisi de réduire la dimension de ces fichiers.

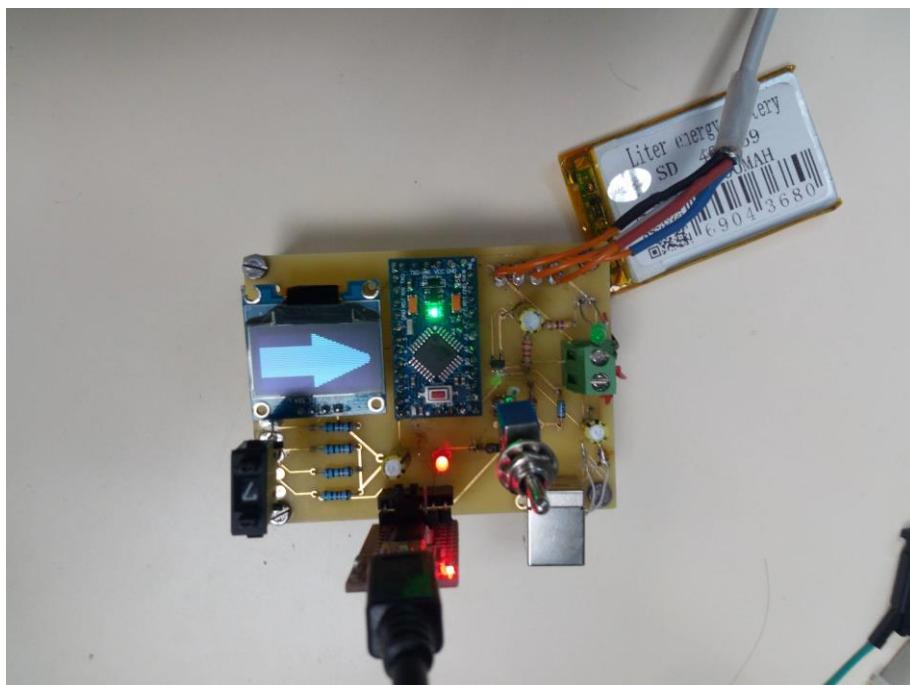
Donc nous choisissons un fichier bitmap par exemple celui-ci



Nous devons choisir la fonction du bloc que l'on est train de créer : sur la case "fonction du bloc" nous choisissons "droite".

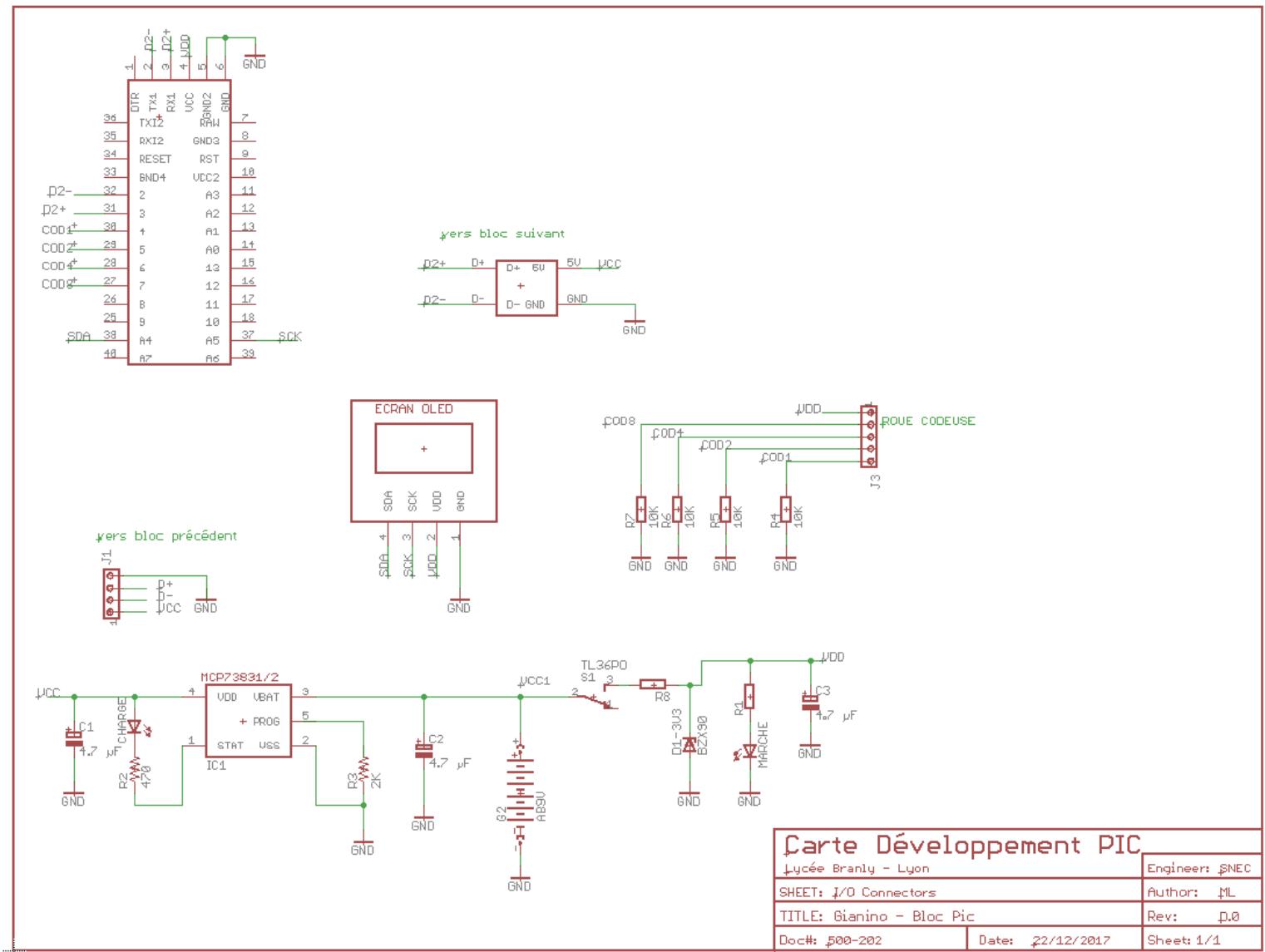
Nous devons dès à présent relier la carte que nous avons réalisée et l'IHM. Pour cela nous choisissons le port du FTRL 232. Une fois la manipulation effectuée nous devons cliquer sur « connecter », puis sur « programmer ».

Voilà le résultat :

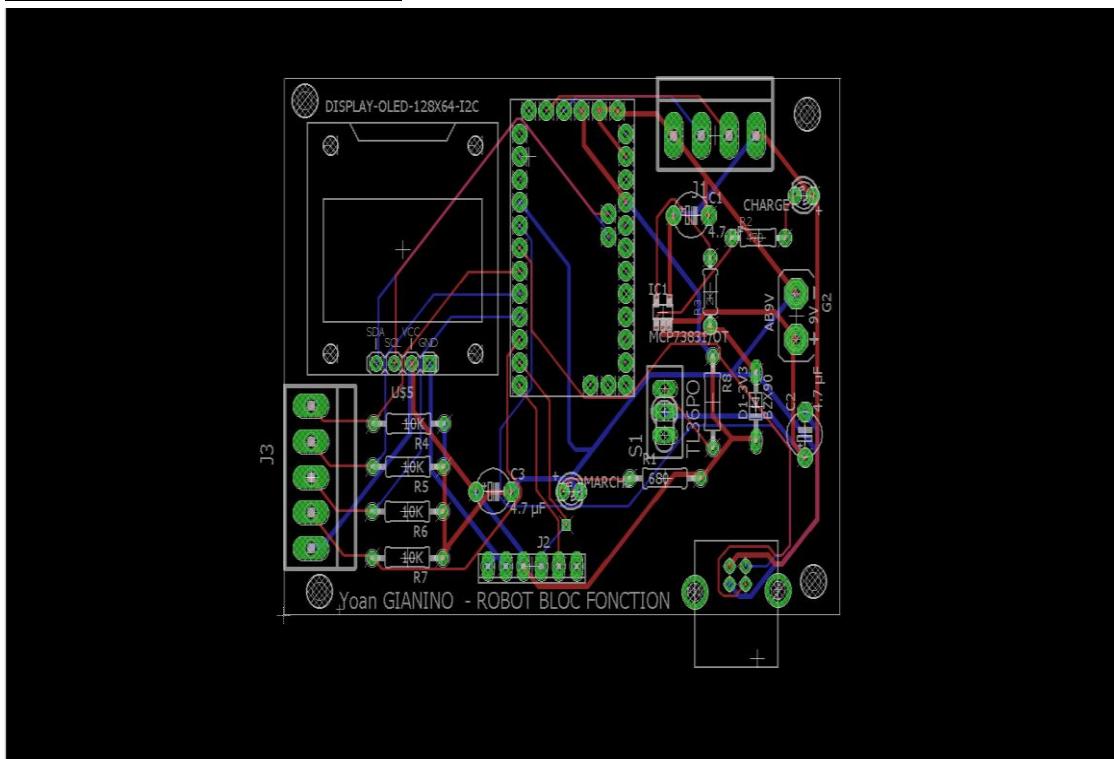


4.7. Document de fabrication

Schéma structurel



Typon de la carte électronique

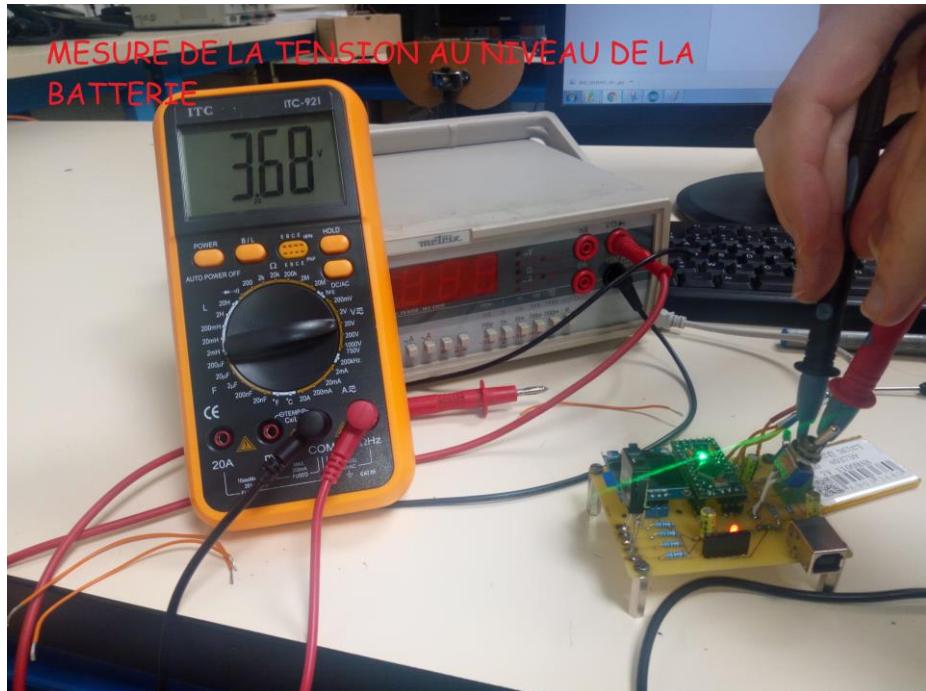


4.8. Coût

typon	34€
Arduino mini pro	1€90
écran oled 128*64	2€14
MCP87361	1€12
LM1117	1€25
USB B	3€40
batterie Li-ion 3.7V 1200mAh	8€98
roue codeuse	6€61
Total	59€61

Note : Certains composants sont empruntés directement à la réserve du lycée.

Test d'alimentation :



Nous pouvons remarquer que la batterie est à 3.68V. D'après la documentation technique, à 3.7V elle devrait être chargée. Donc la batterie est correctement chargée (à la marge d'erreur de mesure près)



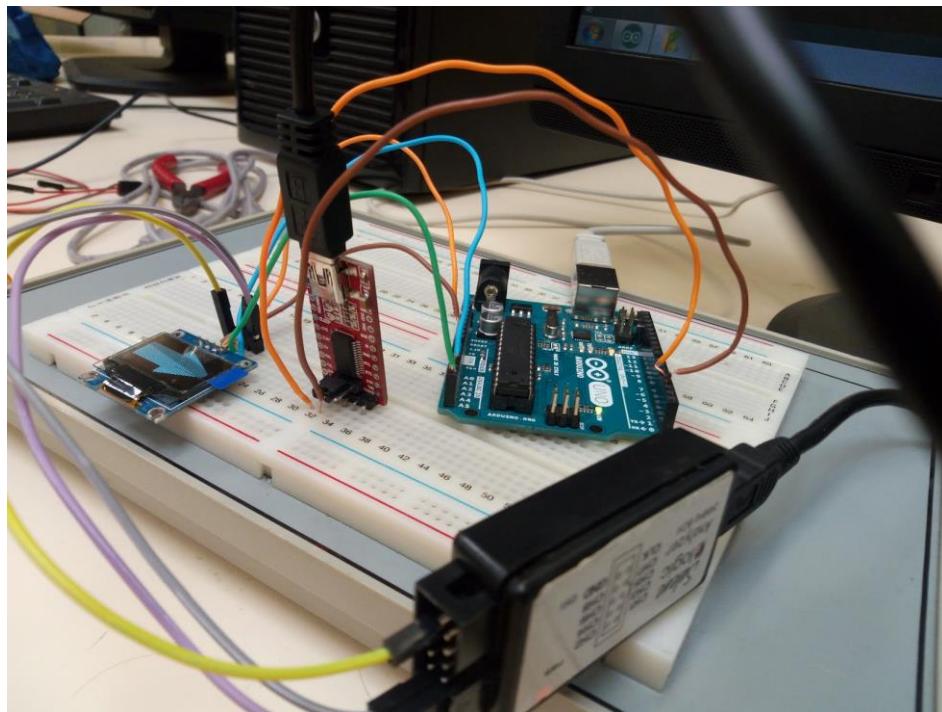
Nous pouvons remarquer que le FTRL232 est alimenté à 3.59 V. D'après le document technique, ce composant doit être alimenté à 3,3V à 10% près.

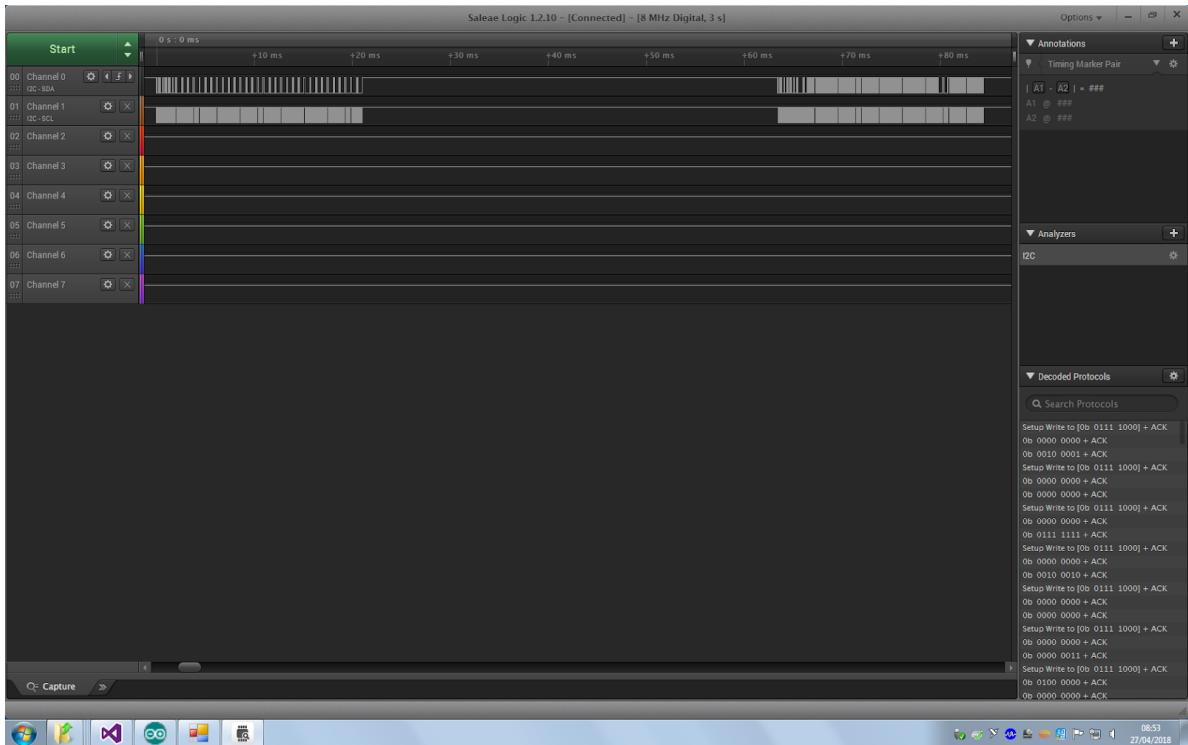


Nous pouvons remarquer que l'écran est alimenté en 3.59V. Ce qui est conforme à la documentation technique.

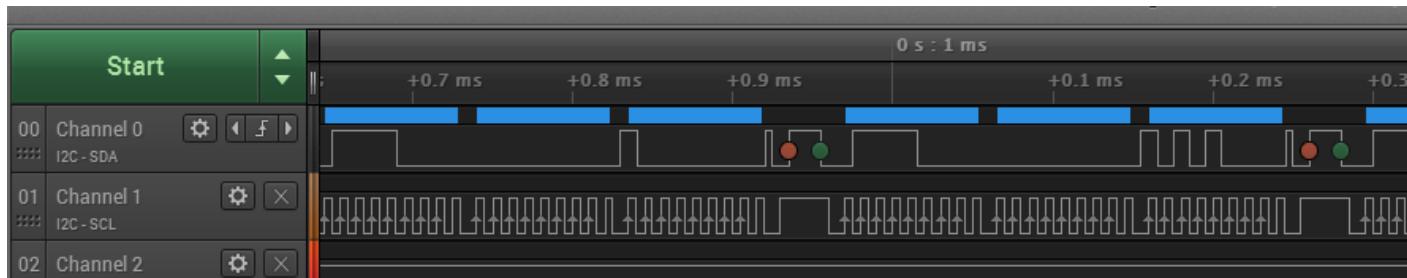
Test de trame :

Pour effectuer ces mesures nous avons utilisé un analyseur logique relié au FTRL 232 :
Câblage de l'intégralité du projet sur une platine d'essais :





Zoomons sur la trame :



Conclusion

Pour conclure, les problèmes rencontrés ont été :

- La mémoire RAM, pour pallier ce programme nous pouvons utiliser soit un Arduino Mega ou un Pic
- Le Pic reviendrait bien moins cher par rapport à l'Arduino MEGA.
- La partie communication qui n'a pas été faite.

Lexique :

IDE : Integrated Development Environment (en français « environnement de développement »), est un logiciel qui rassemble des outils permettant de développer d'autres logiciels tels que des applications mobiles, des logiciels pour ordinateur ou consoles de jeux, des sites web, etc. ainsi que de réaliser des librairies ou des frameworks, c'est-à-dire des morceaux de code qui pourront être sauvegardés et réutilisés dans d'autres programmes.

Scratch : Scratch est un logiciel de programmation visuelle destiné principalement aux enfants, mais aussi aux adolescents.

Programme :

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

#define NUMFLAKES 10
#define XPOS 0
#define YPOS 1
#define DELTAY 2

#define LOGO16_GLCD_HEIGHT 16
#define LOGO16_GLCD_WIDTH 16
```

```
static const unsigned char PROGMEM logo16_glcd_bmp[] = // nom de la flèche du haut
```

```
{B00000000,B00000000, B00011000,B00000000,B00000000,
```

```
B00000000,B00000000, B00111100,B00000000,B00000000,//les bitmap sont fait en binaire sur plusieurs octet
```

```
B00000000,B00000000, B01111110,B00000000,B00000000,
```

```
B00000000,B00000000, B11111111,B00000000,B00000000,  
B00000000,B00000001, B11111111,B10000000,B00000000,  
B00000000,B00000011, B11111111,B11000000,B00000000,  
B00000000,B00000111, B11111111,B11100000,B00000000,  
B00000000,B00001111, B11111111,B11110000,B00000000,  
B00000000,B00011111, B11111111,B11111000,B00000000,  
B00000000,B00111111, B11111111,B11111100,B00000000,  
B00000000,B01111111, B11111111,B11111110,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000001,B11111111, B11111111,B11111111,B10000000,  
B00000011,B11111111, B11111111,B11111111,B11000000,  
B00000111,B11111111, B11111111,B11111111,B11100000,  
B00001111,B11111111, B11111111,B11111111,B11110000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B11111111, B11111111,B11111111,B00000000};
```

```
static const unsigned char PROGMEM logo17_glcd_bmp[] = // nom de la flèche de droite
```

```
{B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B00000000,B00000000,B00000000,B00000000,  
B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B10000000,B00000000,B00000000,  
B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B11000000,B00000000,B00000000,  
B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B11100000,B00000000,B00000000,  
B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B11110000,B00000000,B00000000, B11111111,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111000,  
B00000000,B00000000,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B11111111,B11111100,B00000000,B00000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111110,B00000000,B00000000},
```

```
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B00000000,B00000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B10000000,B00000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B11000000,B00000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B11100000,B00000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B11100000,B00000000,
```

```
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B11000000,B00000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B100000000,B00000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111111,B000000000,B000000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111110,B000000000,B000000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111100,B000000000,B000000000,  
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,  
B11111000,B000000000,B000000000,  
B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,  
B11110000,B000000000,B000000000,  
B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,  
B11100000,B000000000,B000000000,  
B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,  
B11000000,B000000000,B000000000,  
B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,  
B10000000,B000000000,B000000000,  
B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,B000000000,  
B000000000,B000000000,B000000000,};
```

```
static const unsigned char PROGMEM logo18_glcd_bmp[] = //nom de la flèche de  
gauche
```

```
{B00000000,B00000000,  
B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B00000000, B00000000,B00000000,  
B00000001,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B00000000, B00000000,B00000000,  
B00000011,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B00000000,B00000000,B00000000,  
B00000111,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,  
B00000000, B00000000,B00000000,  
B00011111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
```

```

B11111111, B00000000,B00000000,
B00111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000000,
B01111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000000,
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000001,
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000011,
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000111,
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000011,
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000011,
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000000,
B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000000,
B01111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000000,
B00111111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000000,
B00011111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B11111111, B00000000,B00000000,
B00001111,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,
B00000000, B00000000,B00000000,
B00000111,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,
B00000000, B00000000,B00000000,
B00000011,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,
B00000000, B00000000,B00000000,
B00000001,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,
B00000000, B00000000,B00000000,
B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,B00000000,
B00000000, B00000000,B00000000,
B00001111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B00001111, B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,};

```

```

static const unsigned char PROGMEM logo19_glcd_bmp[] = // nom de la flèche du
bas

```

```

{B00000000,B11111111, B11111111,B11111111,B00000000,
B00000000,B11111111, B11111111,B11111111,B00000000,
B00001111,B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,
B00001111, B11111111,B11111111,B11111111,B11111111,B11111111,B11111111,};

```

```
B00000111,B11111111, B11111111,B11111111,B11100000,  
B00000011,B11111111, B11111111,B11111111,B11000000,  
B00000001,B11111111, B11111111,B11111111,B10000000,  
B00000000,B11111111, B11111111,B11111111,B00000000,  
B00000000,B01111111, B11111111,B11111110,B00000000,  
B00000000,B00111111, B11111111,B11111100,B00000000,  
B00000000,B00011111, B11111111,B11111100,B00000000,  
B00000000,B00001111, B11111111,B11110000,B00000000,  
B00000000,B00000111, B11111111,B11100000,B00000000,  
B00000000,B00000011, B11111111,B11000000,B00000000,  
B00000000,B00000001, B11111111,B10000000,B00000000,  
B00000000,B00000000, B11111111,B00000000,B00000000,  
B00000000,B00000000, B01111110,B00000000,B00000000,  
B00000000,B00000000, B00111100,B00000000,B00000000,  
B00000000,B00000000, B00011000,B00000000,B00000000 };
```

```
#if (SSD1306_LCDHEIGHT != 64)  
#endif  
  
int received;  
  
void setup() {  
    // put your setup code here, to run once:  
    Serial.begin(9600);    // vitesse de transfert en 9600 bauds  
  
    //Utiliser le scanner I2C pour trouver le port série sur lequel se trouve votre écran  
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //initialisation de l'écran  
    display.clearDisplay(); // réinitialisation de l'écran  
  
    display.clearDisplay();  
    testscrolltext(); // j'appelle mon programme de texte écrit en dessous  
  
}  
  
void loop() {
```

```
reception(); //j'appelle mon programme réception, pour qu'il puisse le répéter en boucle
```

```
}
```

```
int reception()
```

```
{
```

```
if (Serial.available()>0)
```

```
received = Serial.read();
```

```
if (received == 'a') // nom de la flèche vers le haut
```

```
{
```

```
display.clearDisplay();
```

```
display.drawBitmap(45, 7, logo16_glcd_bmp, 40, 23, 1); //cette ligne sert à afficher un bitmap (x,y, nom du logo, I,L, couleur)
```

```
display.display();
```

```
delay(200);
```

```
}
```

```
if (received == 'g')// nom de la flèche vers la gauche
```

```

{

display.clearDisplay();

display.drawBitmap(22, 7, logo18_glcd_bmp, 80, 23, 1);

display.display();

delay(200);

}

if (received == 'd')// nom de la flèche vers la droite

{

display.clearDisplay();

display.drawBitmap(32, 7, logo17_glcd_bmp, 80, 23, 1);

display.display();

delay(200);

}

if (received == 'b')// nom de la flèche vers le bas

{

display.clearDisplay();

display.drawBitmap(39, 7, logo19_glcd_bmp, 40, 23, 1);

display.display();

delay(200);
}

```

```

}

else

{

    display.clearDisplay(); //réinitialisation de l'écran

}

return(0);

}

void testscrolltext(void) {

    display.setTextSize(2); //taille de l'écriture

    display.setTextColor(WHITE); //couleur de l'écriture

    display.setCursor(10,0); // (x,y) la position du curseur

    display.clearDisplay();

    display.println("robot      blocs"); //écriture du texte

    display.display();

}

```

Annexe:

Datasheet:

LE MCP73831/2



MCP73831/2

Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers

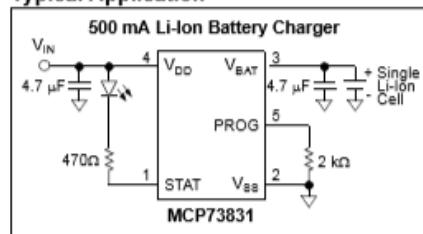
Features:

- Linear Charge Management Controller:
 - Integrated Pass Transistor
 - Integrated Current Sense
 - Reverse Discharge Protection
- High Accuracy Preset Voltage Regulation: $\pm 0.75\%$
- Four Voltage Regulation Options:
 - 4.20V, 4.35V, 4.40V, 4.50V
- Programmable Charge Current: 15 mA to 500 mA
- Selectable Preconditioning:
 - 10%, 20%, 40%, or Disable
- Selectable End-of-Charge Control:
 - 5%, 7.5%, 10%, or 20%
- Charge Status Output:
 - Tri-State Output - MCP73831
 - Open-Drain Output - MCP73832
- Automatic Power-Down
- Thermal Regulation
- Temperature Range: -40°C to +85°C
- Packaging:
 - 8-Lead, 2 mm x 3 mm DFN
 - 5-Lead, SOT-23

Applications:

- Lithium-Ion/Lithium-Polymer Battery Chargers
- Personal Data Assistants
- Cellular Telephones
- Digital Cameras
- MP3 Players
- Bluetooth Headsets
- USB Chargers

Typical Application



Description:

The MCP73831/2 devices are highly advanced linear charge management controllers for use in space-limited, cost-sensitive applications. The MCP73831/2 are available in an 8-Lead, 2 mm x 3 mm DFN package or a 5-Lead, SOT-23 package. Along with their small physical size, the low number of external components required make the MCP73831/2 ideally suited for portable applications. For applications charging from a USB port, the MCP73831/2 adhere to all the specifications governing the USB power bus.

The MCP73831/2 employ a constant-current/constant-voltage charge algorithm with selectable preconditioning and charge termination. The constant voltage regulation is fixed with four available options: 4.20V, 4.35V, 4.40V or 4.50V, to accommodate new, emerging battery charging requirements. The constant current value is set with one external resistor. The MCP73831/2 devices limit the charge current based on die temperature during high power or high ambient conditions. This thermal regulation optimizes the charge cycle time while maintaining device reliability.

Several options are available for the preconditioning threshold, preconditioning current value, charge termination value and automatic recharge threshold. The preconditioning value and charge termination value are set as a ratio or percentage of the programmed constant current value. Preconditioning can be disabled. Refer to [Section 1.0 "Electrical Characteristics"](#) for available options and the [Product Identification System](#) for standard options.

The MCP73831/2 devices are fully specified over the ambient temperature range of -40°C to +85°C.

Package Types

MCP73831/2 2x3 DFN*		MCP73831/2 SOT-23-5	
V _{DD}	1	PROG	STAT 1
V _{DD}	2	EP	V _{BB} 2
V _{BAT}	3	NC	V _{BB} 3
V _{BAT}	4	V _{SS}	V _{BAT} 4
V _{BAT}	5	STAT	V _{DD} 5

* Includes Exposed Thermal Pad (EP); see [Table 3-1](#).

Arduino mini pro:

Microcontroller	ATmega328P *
Board Power Supply	3.35 -12 V (3.3V model) or 5 - 12 V (5V model)
Circuit Operating Voltage	3.3V or 5V (depending on model)
Digital I/O Pins	14
PWM Pins	6
UART	1
SPI	1
I2C	1
Analog Input Pins	6
External Interrupts	2
DC Current per I/O Pin	40 mA
Flash Memory	32KB of which 2 KB used by bootloader *
SRAM	2 KB *
EEPROM	1 KB *
Clock Speed	8 MHz (3.3V versions) or 16 MHz (5V versions)

(*) Older boards were equipped with ATmega 168 with this specs:

- Flash memory: 16 KB
- SRAM: 1 KB
- EEPROM: 512 bytes

USB B:

2411 02

2411 03

**USB-2.0-Steckverbinder
USB 2.0 connectors
Connecteurs USB 2.0**

2411 02

2411 03

USB-Einbaukupplung Typ B, abgewinkelte Ausführung, für Leiterplatten

<p>1. Temperaturbereich</p> <p>2. Werkstoffe</p> <ul style="list-style-type: none"> Kontaktträger Kontaktfeder Lötanschluss Gehäuse <p>3. Mechanische Daten</p> <ul style="list-style-type: none"> Steckkraft Ziehkraft Kontaktierung mit <p>4. Elektrische Daten</p> <ul style="list-style-type: none"> Durchgangswiderstand Bemessungsstrom Bemessungsspannung Prüfspannung Isolationswiderstand 	<p>-40 °C/+80 °C</p> <p>PBT GF CuZn, untermickelt und vergoldet CuZn, untermickelt und verzinkt CuZn, untermickelt und verzinkt</p> <p>≤ 35 N ≥ 10 N USB-Stecker 2431</p> <p>≤ 30 mΩ ≤ 1 A 30 V AC 500 V/60 s ≥ 10⁸ Ω</p>
--	--

2411 02

2411 03

USB chassis socket type B, angular version, for printed circuit boards

<p>1. Temperature range</p> <p>2. Materials</p> <ul style="list-style-type: none"> Body Contact spring Solder pin Shell <p>3. Mechanical data</p> <ul style="list-style-type: none"> Insertion force Withdrawal force Mating with <p>4. Electrical data</p> <ul style="list-style-type: none"> Contact resistance Rated current Rated voltage Test voltage Insulation resistance 	<p>-40 °C/+80 °C</p> <p>PBT GF CuZn, pre-nickelated and gold-plated CuZn, pre-nickelated and tinned CuZn, pre-nickelated and tinned</p> <p>≤ 35 N ≥ 10 N USB plug 2431</p> <p>≤ 30 mΩ ≤ 1 A 30 V AC 500 V/60 s ≥ 10⁸ Ω</p>
--	---

2411 02

2411 03

Embase femelle USB type B, version angulaire, pour cartes imprimées

<p>1. Température d'utilisation</p> <p>2. Matériaux</p> <ul style="list-style-type: none"> Corps isolant Ressort de contact Plot à souder Boltier <p>3. Caractéristiques mécaniques</p> <ul style="list-style-type: none"> Force d'insertion Force de séparation Raccordement avec <p>4. Caractéristiques électriques</p> <ul style="list-style-type: none"> Résistance de contact Courant assigné Tension assignée Tension d'essai Résistance d'isolation 	<p>-40 °C/+80 °C</p> <p>PBT GF CuZn, sous-nickelé et doré CuZn, sous-nickelé et étamé CuZn, sous-nickelé et étamé</p> <p>≤ 35 N ≥ 10 N connecteur mâle USB 2431</p> <p>≤ 30 mΩ ≤ 1 A 30 V AC 500 V/60 s ≥ 10⁸ Ω</p>
--	--

**Bestellbezeichnung
Designation
Désignation**

**Polzahl
Poles
Pôles**

**Verpackungseinheit
Package unit
Unité d'emballage**

2411 02	4	100	
2411 03	4	100	

Verpackung: lose im Karton oder Kunststoffbeutel
Packaging: in bulk in a cardboard box or a plastic bag
Emballage: en vrac dans un carton ou sachet en plastique

www.lumberg.com

06/2006

Ecran 128/64:

1 GENERAL DESCRIPTION

SSD1306 is a single-chip CMOS OLED/PLED driver with controller for organic / polymer light emitting diode dot-matrix graphic display system. It consists of 128 segments and 64 commons. This IC is designed for Common Cathode type OLED panel.

The SSD1306 embeds with contrast control, display RAM and oscillator, which reduces the number of external components and power consumption. It has 256-step brightness control. Data/Commands are sent from general MCU through the hardware selectable 6800/8000 series compatible Parallel Interface, I²C interface or Serial Peripheral Interface. It is suitable for many compact portable applications, such as mobile phone sub-display, MP3 player and calculator, etc.

2 FEATURES

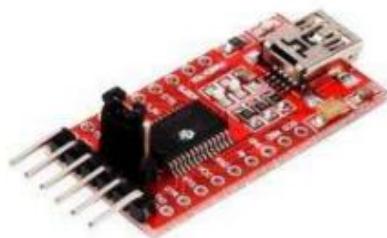
- Resolution: 128 x 64 dot matrix panel
- Power supply
 - V_{DD} = 1.65V to 3.3V for IC logic
 - V_{CC} = 7V to 15V for Panel driving
- For matrix display
 - OLED driving output voltage, 15V maximum
 - Segment maximum source current: 100uA
 - Common maximum sink current: 15mA
 - 256 step contrast brightness current control
- Embedded 128 x 64 bit SRAM display buffer
- Pin selectable MCU Interfaces:
 - 8-bit 6800/8080-series parallel interface
 - 3/4 wire Serial Peripheral Interface
 - I²C Interface
- Screen saving continuous scrolling function in both horizontal and vertical direction
- RAM write synchronization signal
- Programmable Frame Rate and Multiplexing Ratio
- Row Re-mapping and Column Re-mapping
- On-Chip Oscillator
- Chip layout for COG & COF
- Wide range of operating temperature: -40°C to 85°C

3 ORDERING INFORMATION

Table 3-1: Ordering Information

Ordering Part Number	SEG	COM	Package Form	Reference	Remark
SSD1306Z	128	64	COG	8	<ul style="list-style-type: none">◦ Min SEG pad pitch : 47um◦ Min COM pad pitch : 40um◦ Die thickness: 300 +/- 25um
SSD1306TR1	104	48	TAB	11, 56	<ul style="list-style-type: none">◦ 35mm film, 4 sprocket hole, Folding TAB◦ 8-bit 80 / 8-bit 68 / SPI / I²C interface◦ SEG, COM lead pitch 0.1mm x 0.997mm =0.0997mm◦ Die thickness: 457 +/- 25um

FT232RL USB TO TTL 5V 3.3V Convertor



The USB to TTL serial adapter is based on the high quality and very popular FTDI FT232RL chipset and is an excellent way to connect TTL serial devices to a PC through a USB port.

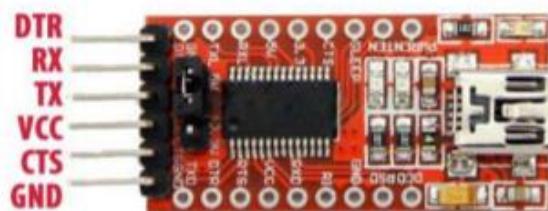
This USB to TTL serial adapter is ideal for many uses, including:

- Programming microprocessors such as ARM, AVR, etc
- Working with computing hardware such as routers and switches
- Serial communication with many devices such as GPS devices
- Serial terminals on devices like the Raspberry Pi

Unlike most USB to TTL serial adapters, this adapter supports both 5V AND 3.3V operation! Simply set the jumper as required to choose between 5V and 3.3V as labelled on the board.

The adapter comes with a right-angle connector fitted allowing you to use it straight away. If you need to access any of the other inputs or outputs of the FT232RL, all the useful signals are provided as through-hole solder pads - ideal for use with straight headers into a breadboard, for example.

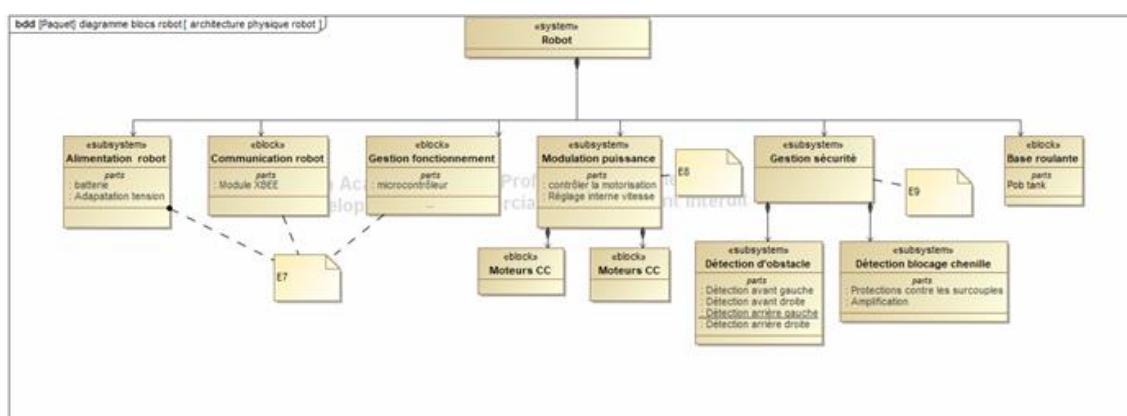
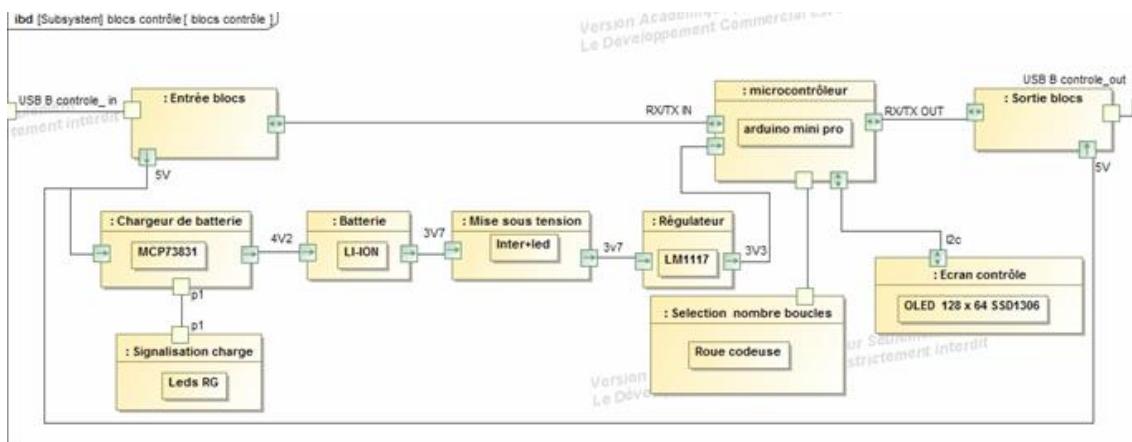
The main connector has 6 pins:



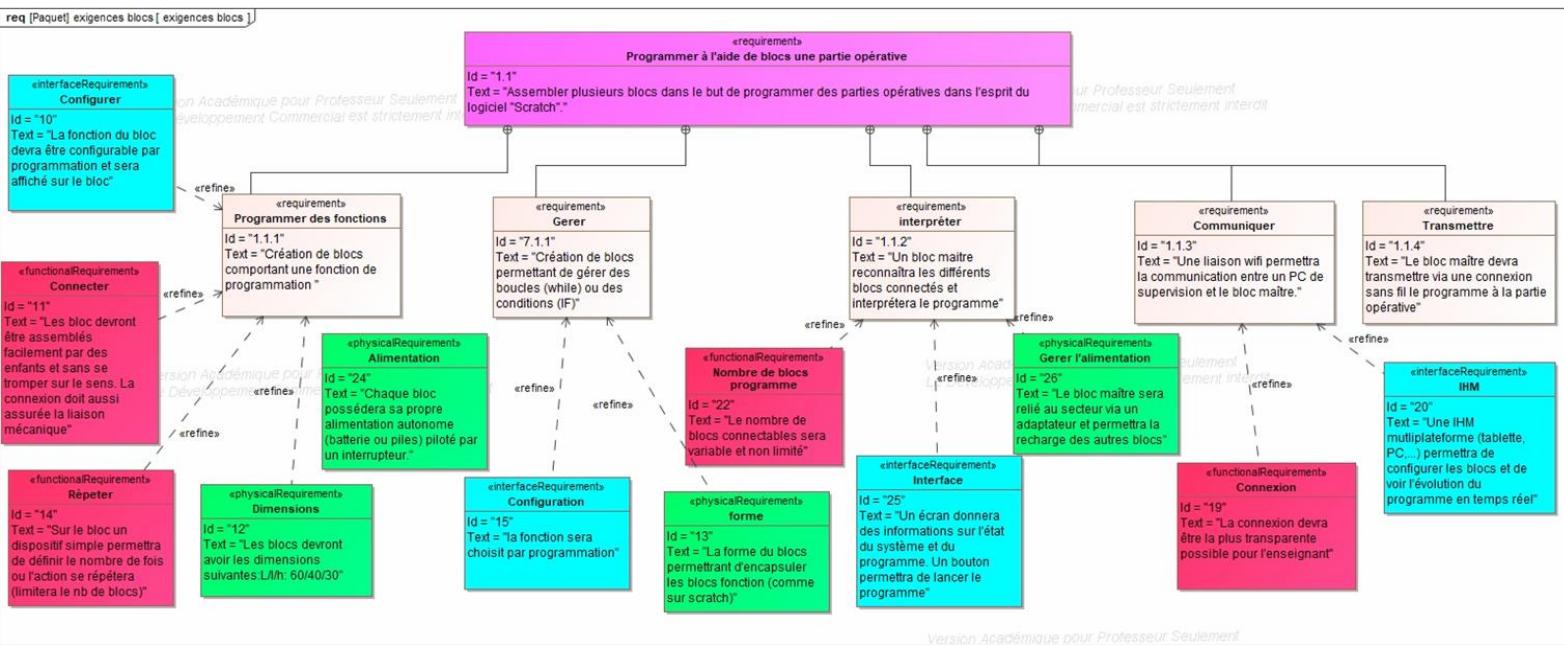
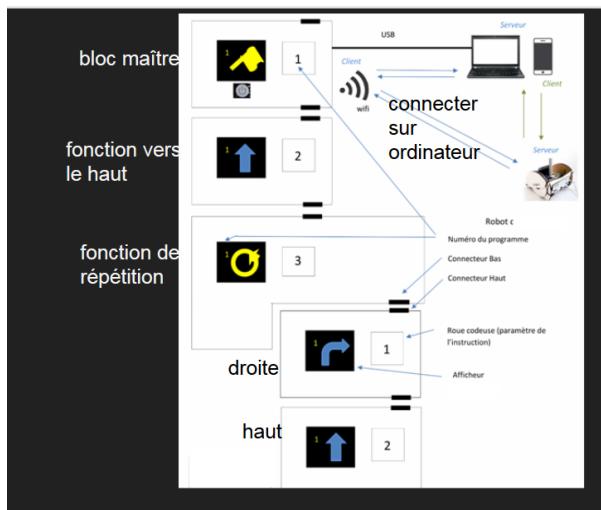
Partie Personnelle Alexandre ANDREUX

2.1 Étudiant 7 bloc Répétition

Structure fonctionnelle



Le bloc répétition a pour but comme son nom l'indique de répéter les actions qui sont coordonnées par les programmes envoyés depuis le bloc maître et suivant de toutes la chaîne de bloc



Cahier des charges

- les blocs doivent être assemblés facilement par des enfants, sans se tromper sur le sens. La connexion doit aussi assurer la liaison mécanique.
- sur le bloc un dispositif permet de définir le nombre de fois où l'action se répètera.
- les blocs doivent avoir les dimensions suivantes L/l/h: 60/40/30
- les blocs possèdent leur propres alimentation, piloté par un interrupteur
- la fonction d'un bloc est choisie par la programmation
- le nombre de blocs connectables est variable et non limité
- la forme des blocs permettant d'encapsuler les blocs fonctions (comme sur scratch)
- un écran donne des informations sur l'état du système, un bouton permet de lancer le programme

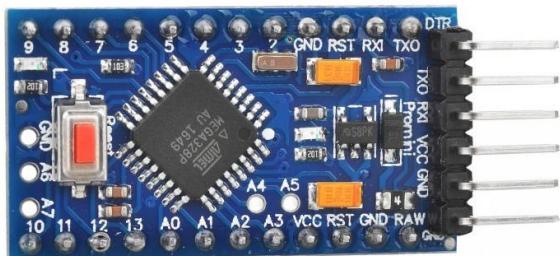
-la connexion doit être la plus transparente possible pour l'enseignant

2.3 Choix des composants

le MCP73831:



Carte Arduino mini pro:



L'Arduino Pro Mini est une carte microcontrôleur basée sur l'ATmega328P.

Il dispose de 14 broches d'entrée / sortie numériques (dont 6 peuvent être utilisées comme sorties PWM), de 6 entrées analogiques, d'un résonateur intégré, d'un bouton de réinitialisation et de trous pour le montage des broches. Un connecteur à six broches peut

être connecté à un câble FTDI ou à une carte de dérivation Sparkfun pour fournir une alimentation et une communication USB à la carte.

-L'Arduino Pro Mini est destiné à une installation semi permanente dans des objets ou des expositions. La carte est livrée sans embases pré montées, permettant l'utilisation de divers types de connecteurs ou la soudure directe des fils. La disposition des broches est compatible avec l'Arduino Mini.

-Il existe deux versions du Pro Mini. L'un fonctionne à 3.3V et 8 MHz, l'autre à 5V et 16 MHz.

USB B:



Débit évalué à 5 Go/s comparativement au débit théorique de l'USB 2.0 qui est de 480 Mo/s.

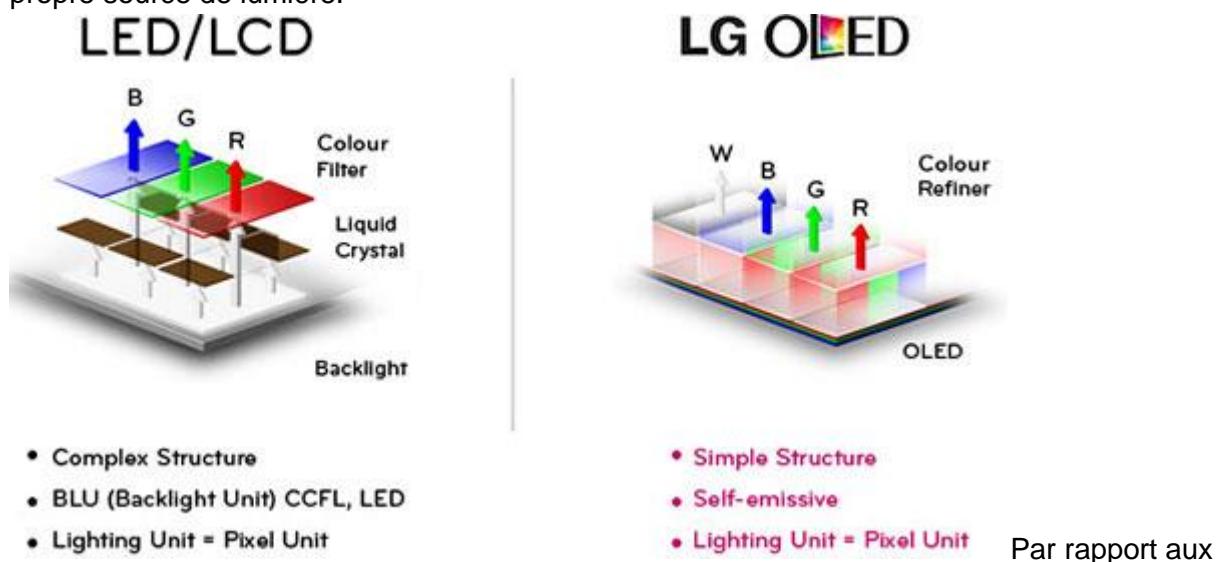
Capacité à être rétro compatible avec les normes précédentes.

Ecran OLED 128/64 SSD 1306:

OLED, pour "Organic Light-Emitting Diode" est un composant capable de produire de la lumière qui peut être utilisé pour la fabrication des TV.

La diode est formée à partir de semi-conducteurs contenant de l'oxygène, du carbone, de l'hydrogène et des atomes d'azote, c'est pour cela que l'on parle de LED organique.

La technologie **OLED** est électroluminescente, c'est à dire qu'elle est capable de générer sa propre source de lumière.



TV utilisant la technologie LCD / LCD LED, un modèle OLED ne nécessite donc pas de rétroéclairage de la dalle.

Ainsi les résultats en contraste et profondeur de noir pourront être supérieurs tout en réduisant l'épaisseur de la TV.

Comparaison entre la structure d'une TV LCD / LCD LED (à gauche) et celle d'une TV OLED (à droite) (crédit : LG)

L'émission de lumière de la technologie **OLED** résulte de l'injection de charges électriques dans une couche de matière organique.

A l'intérieur de la couche émettrice, un couple électron-trou est à l'origine de l'émission d'un photon.

La structure d'un composant OLED se décompose en quatre couches :

- un substrat qui constitue son support, il peut être en verre ou en plastique

- l'anode, électrode en oxyde d'indium-étain, qui est à l'origine des défauts d'électrons (les trous)
- les couches constituées d'éléments organiques qui vont conduire les électrons
- la cathode, électrode principalement faite de métaux, qui émet les électrons.



Cet écran possède 8192 pixels.

Donc 1024 octets.

Ses mensurations sont 128/64.

Il est codé en I2C

Cet écran est petit et peut afficher ce que l'on lui demande.

Il possède 4 pin: SDA, SCL enfin VCC et GND.

Il est alimenté en 3V3

LM 1117 :

Régulateur de tension 3V3



Batterie LI-ION :



Ses principaux avantages sont une énergie massique élevée (deux à cinq fois plus que le nickel-hydrure métallique par exemple) ainsi que l'absence d'effet mémoire (L'effet mémoire entraîne une diminution de la quantité d'énergie que l'accumulateur peut restituer, avec pour conséquence une diminution de la capacité nominale de l'accumulateur). Enfin, l'autodécharge est relativement faible par rapport à d'autres accumulateurs. Cependant, le coût reste important et cantonne le lithium aux systèmes de petite taille.

Avantages :

Ils possèdent une haute densité d'énergie pour un poids très faible, grâce aux propriétés physiques du lithium. Ces accumulateurs sont donc très utilisés dans le domaine des systèmes embarqués.

- Ils ne présentent aucun effet mémoire contrairement aux accumulateurs à base de nickel.
- Ils ont une faible autodécharge (10 % par mois voire souvent moins de quelques % par an).
- Ils ne nécessitent pas de maintenance.
- Ils peuvent permettre une meilleure sécurité que les batteries purement lithium, mais ils nécessitent toujours un circuit de protection.

Inconvénients :

La nature des cycles de décharge : ces batteries préservent mieux leur capacité lorsqu'elles sont rechargées à partir d'un état de décharge partielle que lorsqu'elles subissent des cycles complets de décharge/recharge.

La décharge profonde : La décharge profonde (< 2,5 V par élément ou < 5 % de la capacité totale) est destructrice et peut altérer irrémédiablement l'endurance de ces batteries.

Les éléments lithium-ion sont passivés par construction (par exemple par dépôt d'un mince film de chlorure de lithium sur l'anode) afin de les protéger contre l'autodécharge pendant le stockage et la corrosion. Cependant, cette passivation peut avoir des inconvénients car, en augmentant la résistance interne de l'élément, elle génère une chute de tension lors de l'utilisation (au début de l'application de la charge). Ceci est d'autant plus sensible que le courant demandé par la charge est élevé, ce qui peut conduire à l'intervention du circuit de protection qui coupe alors le circuit si la tension par élément descend en dessous de 2,5 V. Cette résistance de la couche de passivation augmente avec la durée et la température de stockage (les températures élevées augmentent la passivation). D'autre part, cet effet est accentué si la température de décharge est basse et augmente avec les cycles d'utilisation. Mais, l'amplitude du phénomène est aussi fonction de la conception chimique qui n'est pas la même selon les fabricants⁷.

Les courants de charge et de décharge admissibles sont plus faibles qu'avec d'autres techniques.

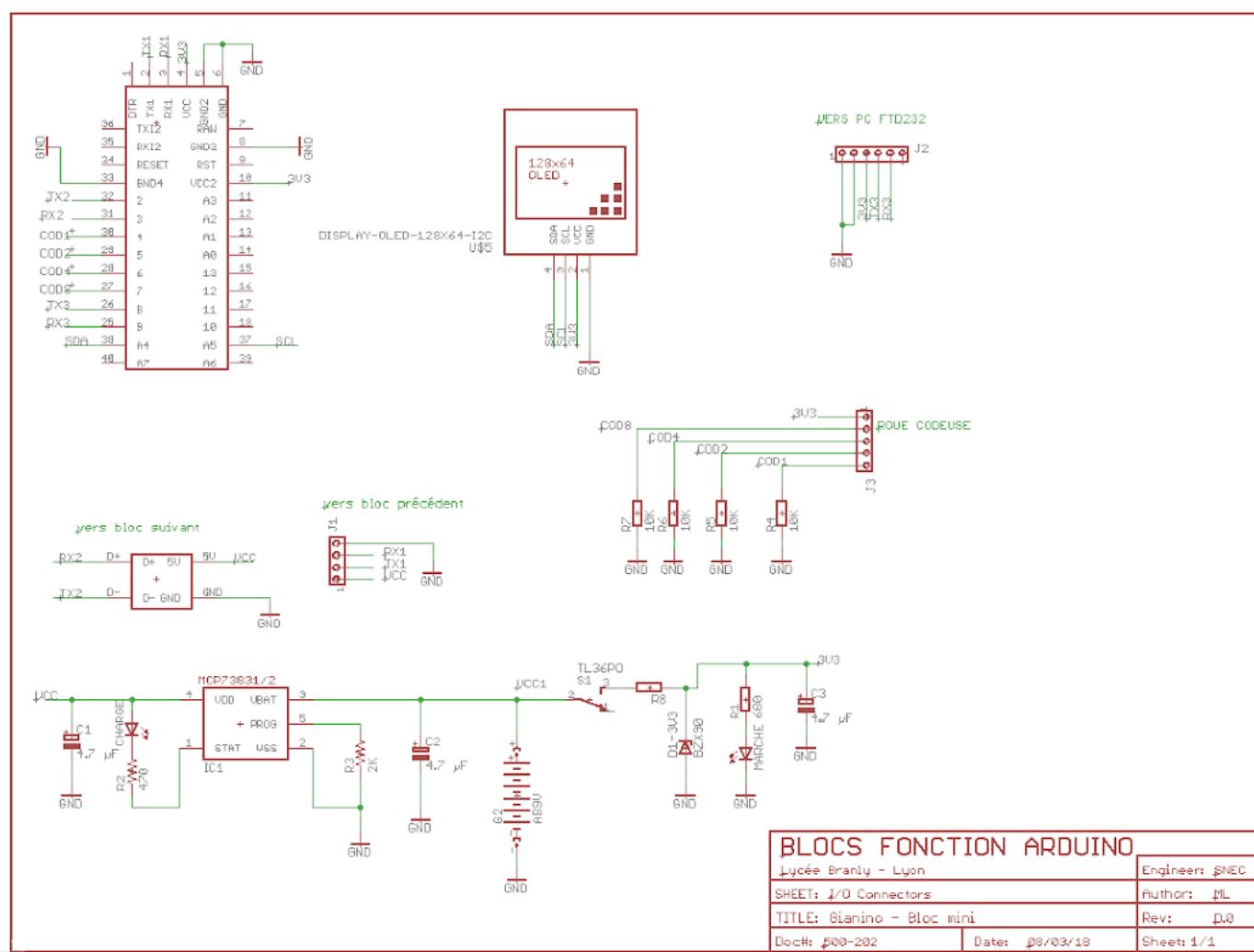
Il peut se produire un court-circuit entre les deux électrodes par croissance dendritique de lithium.

L'utilisation d'un électrolyte liquide présente des dangers si une fuite se produit et que celui-ci entre en contact avec de l'air ou de l'eau (transformation en liquide corrosif : l'hydroxyde de lithium). Cette technique mal utilisée présente des dangers potentiels : elles peuvent se dégrader en chauffant au-delà de 80 °C en une réaction brutale et dangereuse. Il faut toujours manipuler les accumulateurs lithium-ion avec une extrême précaution, ces batteries peuvent être explosives.

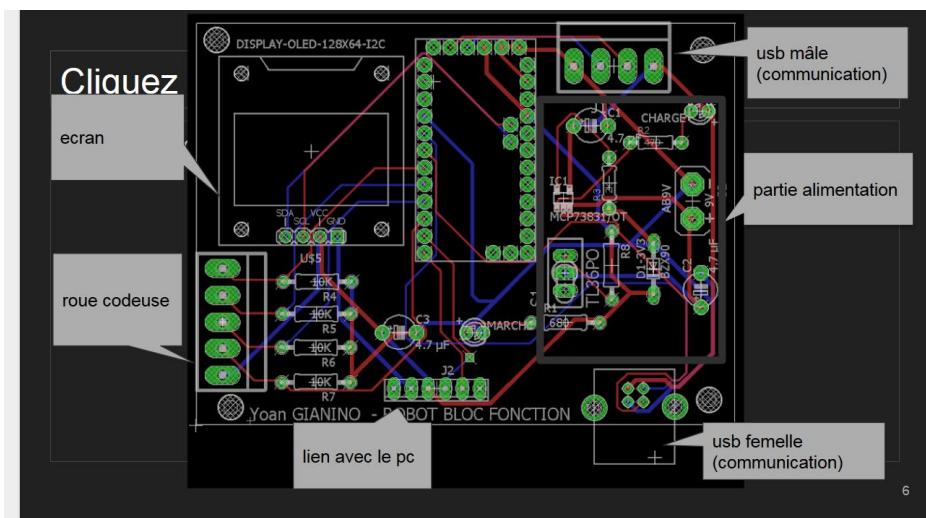
Comme avec tout accumulateur d'électricité ne jamais mettre en court-circuit l'accumulateur, inverser les polarités, surcharger ni percer le boîtier.

2.4 Schéma de construction

Schéma structurel



Typon



2.5 Coût des composants

typon	34€
Arduino mini pro	1€90
écran oled 128*64	2€14

MCP87361	1€12
LM1117	1€25
USB B	3€40
batterie Li-ion 3.7V 1200mAh	8€98
roue codeuse	6€61
Total	59€61

2.6 ANNEXE :

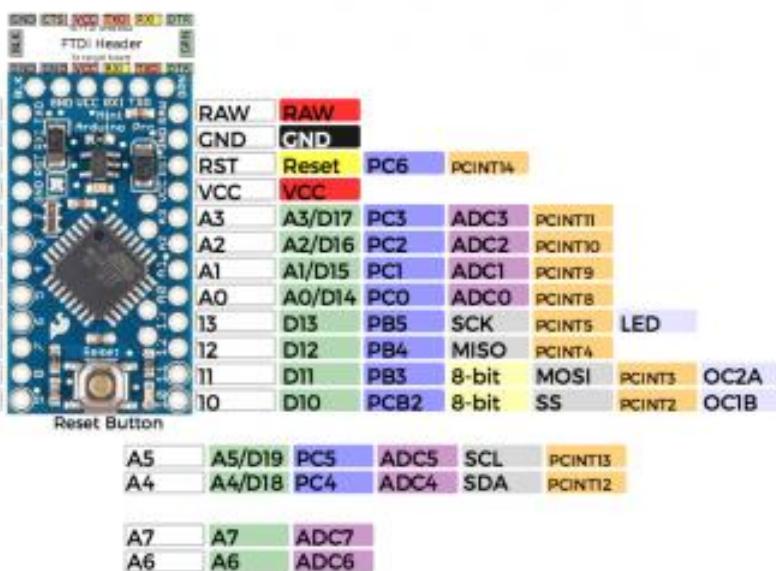
Arduino mini pro

Name	ADC
Power	PWM
GND	Serial
Control	Ext Interrupt
Arduino	PC Interrupt
Port	Misc

Arduino Pro Mini (DEV-11114)

Programmed as Arduino Pro Mini w/ ATMega328
8MHz/ 3.3V

	PCINT17	TXD	PD1	D1	TX0
	PCINT16	RXD	PD0	D0	RXI
	PCINT14	PC6	Reset	RST	GND
	PCINT18	INT0	PD2	D2	2
OC2B	PCINT19	INT1	8-bit	PD3	D3
	XCK	T0	PCINT20	PD4	D4
TI	OCOB	PCINT21	8-bit	PD5	D5
AINO	OCOA	PCINT22	8-bit	PD6	D6
		INT1	PCINT23	PD7	D7
	CLKO	ICP1	PCINT0	PB0	D8
	OCIA	PCINT1	8-bit	PB1	D9



Power
Raw: 3.3V-16V (4V-12V recommended)
VCC: 3.3V
Maximum current: 150mA @3.3V

ATMega328P
 Absolute maximum VCC: 6V
 Maximum current for chip: 200mA
 Maximum current per pin: 40mA
 Recommended current per pin: 20mA
 8-bit Atmel AVR
 Flash Program Memory: 32kB
 EEPROM: 1kB
 Internal SRAM 2kB
 ADC10-bit
 PWM 8-bit

LEDs
Power: Red
User (D13): Green

 sparkfun.com



MCP73831/2

Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers

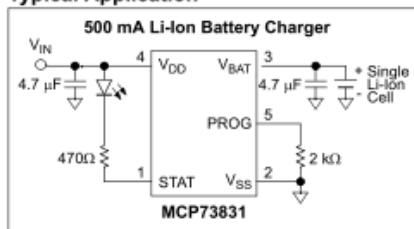
Features:

- Linear Charge Management Controller:
 - Integrated Pass Transistor
 - Integrated Current Sense
 - Reverse Discharge Protection
- High Accuracy Preset Voltage Regulation: $\pm 0.75\%$
- Four Voltage Regulation Options:
 - 4.20V, 4.35V, 4.40V, 4.50V
- Programmable Charge Current: 15 mA to 500 mA
- Selectable Preconditioning:
 - 10%, 20%, 40%, or Disable
- Selectable End-of-Charge Control:
 - 5%, 7.5%, 10%, or 20%
- Charge Status Output
 - Tri-State Output - MCP73831
 - Open-Drain Output - MCP73832
- Automatic Power-Down
- Thermal Regulation
- Temperature Range: -40°C to +85°C
- Packaging:
 - 8-Lead, 2 mm x 3 mm DFN
 - 5-Lead, SOT-23

Applications:

- Lithium-Ion/Lithium-Polymer Battery Chargers
- Personal Data Assistants
- Cellular Telephones
- Digital Cameras
- MP3 Players
- Bluetooth Headsets
- USB Chargers

Typical Application



Description:

The MCP73831/2 devices are highly advanced linear charge management controllers for use in space-limited, cost-sensitive applications. The MCP73831/2 are available in an 8-Lead, 2 mm x 3 mm DFN package or a 5-Lead, SOT-23 package. Along with their small physical size, the low number of external components required make the MCP73831/2 ideally suited for portable applications. For applications charging from a USB port, the MCP73831/2 adhere to all the specifications governing the USB power bus.

The MCP73831/2 employ a constant-current/constant-voltage charge algorithm with selectable preconditioning and charge termination. The constant voltage regulation is fixed with four available options: 4.20V, 4.35V, 4.40V or 4.50V, to accommodate new, emerging battery charging requirements. The constant current value is set with one external resistor. The MCP73831/2 devices limit the charge current based on die temperature during high power or high ambient conditions. This thermal regulation optimizes the charge cycle time while maintaining device reliability.

Several options are available for the preconditioning threshold, preconditioning current value, charge termination value and automatic recharge threshold. The preconditioning value and charge termination value are set as a ratio or percentage of the programmed constant current value. Preconditioning can be disabled. Refer to [Section 1.0 "Electrical Characteristics"](#) for available options and the [Product Identification System](#) for standard options.

The MCP73831/2 devices are fully specified over the ambient temperature range of -40°C to +85°C.

Package Types

MCP73831/2 2x3 DFN*		MCP73831/2 SOT-23-5	
V _{DD}	1	8	PROG
V _{DD}	2	7	STAT
V _{BAT}	3	9	V _{SS}
V _{BAT}	4	6	V _{SS}
V _{BAT}	5	5	V _{DD}
			V _{BAT} [3]

* Includes Exposed Thermal Pad (EP); see [Table 3-1](#).

LM1117 :

Product Folder Sample & Buy Technical Documents Tools & Software Support & Community Reference Design

TEXAS INSTRUMENTS

LM1117
SNOS412N – FEBRUARY 2000 – REVISED JANUARY 2016

LM1117 800-mA Low-Dropout Linear Regulator

1 Features

- Available in 1.8 V, 2.5 V, 3.3 V, 5 V, and Adjustable Versions
- Space-Saving SOT-223 and WSON Packages
- Current Limiting and Thermal Protection
- Output Current 800 mA
- Line Regulation 0.2% (Maximum)
- Load Regulation 0.4% (Maximum)
- Temperature Range
 - LM1117: 0°C to 125°C
 - LM1117I: -40°C to 125°C

2 Applications

- Post Regulator for Switching DC–DC Converter
- High Efficiency Linear Regulators
- Battery Chargers
- Portable Instrumentation
- Active SCSI Termination Regulator

3 Description

The LM1117 is a low dropout voltage regulator with a dropout of 1.2 V at 800 mA of load current.

The LM1117 is available in an adjustable version, which can set the output voltage from 1.25 to 13.8 V with only two external resistors. In addition, it is available in five fixed voltages, 1.8 V, 2.5 V, 3.3 V, and 5 V.

The LM1117 offers current limiting and thermal shutdown. Its circuit includes a Zener trimmed bandgap reference to assure output voltage accuracy to within $\pm 1\%$.

A minimum of 10- μ F tantalum capacitor is required at the output to improve the transient response and stability.

Device Information⁽¹⁾

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM1117, LM1117I	SOT-223 (4)	6.50 mm × 3.50 mm
	TO-220 (3)	14.986 mm × 10.16 mm
	TO-252 (3)	6.58 mm × 6.10 mm
	WSON (8)	4.00 mm × 4.00 mm
	TO-263 (3)	10.18 mm × 8.41 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Adjustable Output Regulator

$$V_{OUT} = 1.25 \left(1 + \frac{R_2}{R_1} \right)$$

* C_{Adj} is optional, however it will improve ripple rejection.