DOSSIER TECHNIQUE

AFFICHEUR DES FLÈCHES DE POSSESSION AU BASKET

BUT3 : GENIE ELECTRIQUE ET INFORMATIQUE INDUSTRIELLE 3EME ANNEE

REALISE PAR : JULES AUGEREAU, DAMIEN CHALOT, EWAN CLEMENT

GROUPE : 3300

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE – UNIVERSITE D'ANGERS

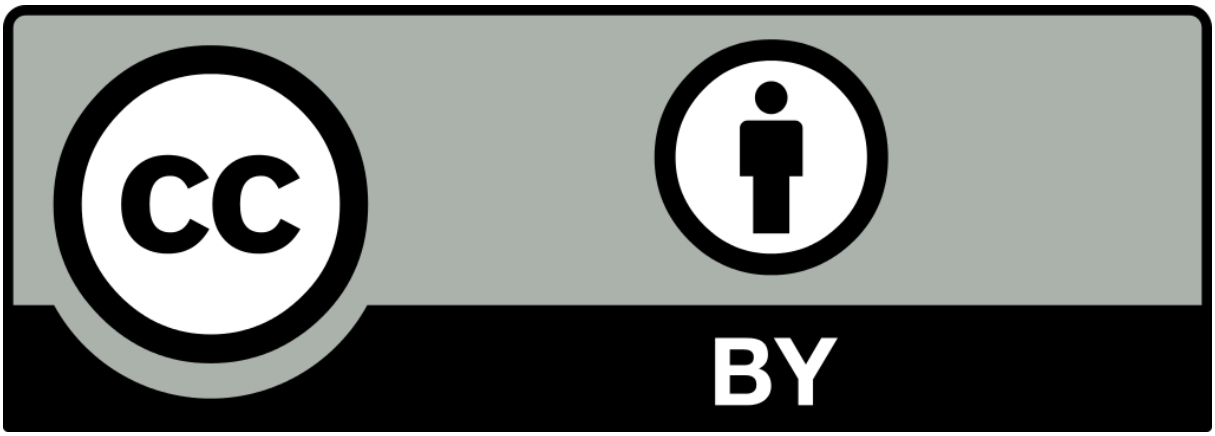
4 BD DE LAVOISIER, 49000 ANGERS

INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
UNIVERSITÉ D'ANGERS

université angers

MENTION LEGALE

Ce rapport présente le travail réalisé par un groupe d'étudiants dans le cadre d'un projet pédagogique. Les auteurs et l'Université d'Angers ne garantissent pas que l'information, les documents, la méthodologie et le matériel présentés dans ce document soient complets, conformes à l'état de l'art et exacts ni n'assurent en toutes circonstances la sécurité des biens, des personnes et des utilisateurs. Les auteurs et l'Université d'Angers ne seront pas tenus responsables des dommages éventuels qui pourraient résulter de l'utilisation du contenu du présent rapport.



Ewan CLEMENT

Jules AUGEREAU

Damien CHALOT

REMERCIEMENT

Avant d'entrer dans le cœur du sujet, nous souhaitons exprimer nos remerciements à l'Université d'Angers, et plus particulièrement l'IUT, qui a généreusement mis à notre disposition le temps et les équipements nécessaires à la concrétisation de ce projet.

Nous tenons également à exprimer notre gratitude envers la responsable du magasin de l'IUT, Madame Denecheau. Sa contribution a été essentielle à divers niveaux : de la soudure des composants à la fourniture du matériel indispensable pour notre projet, en passant par son assistance dans le routage de notre carte.

Enfin, nous remercions tout particulièrement Mr Lucidarme. Son aide nous a été très précieuse dans le déroulement de notre projet. En mettant à notre disposition une salle de travail et les équipements nécessaires, il a grandement facilité la réalisation de notre travail. Ses conseils éclairés et son expertise ont été des atouts majeurs qui nous ont guidés vers la réussite de notre projet.

SOMMAIRE

INTRODUCTION

Un match de basket commence par un "entre-deux". Le principe est simple. Deux joueurs, un par équipe, se mettent face à face. L'arbitre lance le ballon en l'air entre ces deux joueurs qui doivent sauter pour taper le ballon et le transmettre vers un joueur de leur équipe. C'est l'unique « entre-deux » du match qui se joue à partir d'un lancer de l'arbitre. La possession du ballon s'alterne ensuite entre les deux équipes à tour de rôle. Cette possession est matérialisée par une flèche qui indique l'équipe qui bénéficiera du ballon à l'entre-deux suivant.

Entre-deux basket – image issu du site chorale-roanne.com

Pour les clubs de sport avec un budget limité, la flèche est en papier ou en bois.

D'autres ont investi dans une flèche à LEDs.

Table de marque basket

Flèche possession basket

Les différentes occasions d'entre-deux sont les suivantes :

- Au début de chaque nouveau quart-temps, la balle va à l'équipe indiquée par la flèche.
- Si deux joueurs tiennent la balle fermement et qu'aucun ne veut la lâcher, l'arbitre siffle entre-deux.
- Si l'arbitre a un doute à l'issue d'une action car il a mal vu, il donne la possession selon l'indication de la flèche.

Une autre règle est importante à connaître pour la table de marque. Lorsque l'arbitre siffle un entre-deux, il va regarder la flèche et donner la balle à l'équipe en fonction de son indication. La table de marque doit impérativement attendre que l'arbitre ait remis la balle en jeu et que le match ait repris avant de

changer la flèche de sens. Ce laps de temps, pouvant atteindre plus d'une minute, il arrive parfois que la table de marque oublie de changer la flèche de sens au moment de la remise en jeu.

L'objectif de notre projet est d'améliorer l'utilisation de la flèche par l'utilisateur en la modernisant.

La majorité des affichages sportifs se fait avec des panneaux à LEDs. Nous proposons donc d'harmoniser la flèche de possession avec ce type d'affichage en créant un dispositif à LEDs. Il est constitué de deux flèches tournées vers l'arbitre et les joueurs et deux flèches tournées vers la table de marque afin que chacun puisse visualiser la possession. De plus, nous avons intégré un système qui permet d'éviter d'oublier de tourner la flèche.

Tableau de score basket

1. CAHIER DES CHARGES

1.1. DIAGRAMME SYML

Voici le diagramme qui résume les livrables de notre projet. Il permet d'indiquer l'organisation et la priorité de chaque étape.

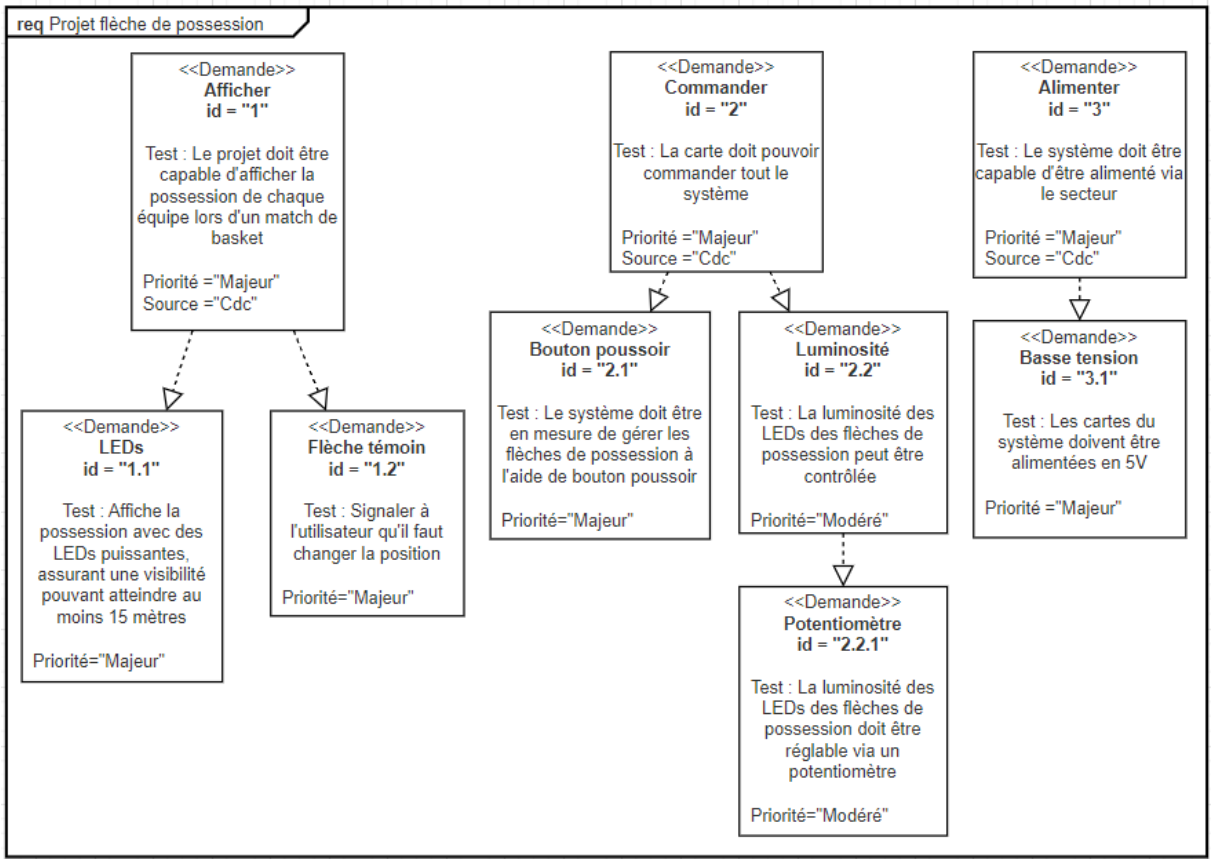


Diagramme sysml

1.2. DIAGRAMME DCU

Comme on peut le voir à travers le diagramme de cas d'utilisation (dcu), l'utilisateur doit pouvoir contrôler l'affichage de la possession afin de le rendre visible pour les joueurs du match de basket. Il peut également contrôler la luminosité de ces flèches, comme il peut contrôler la signalisation du changement de possession.

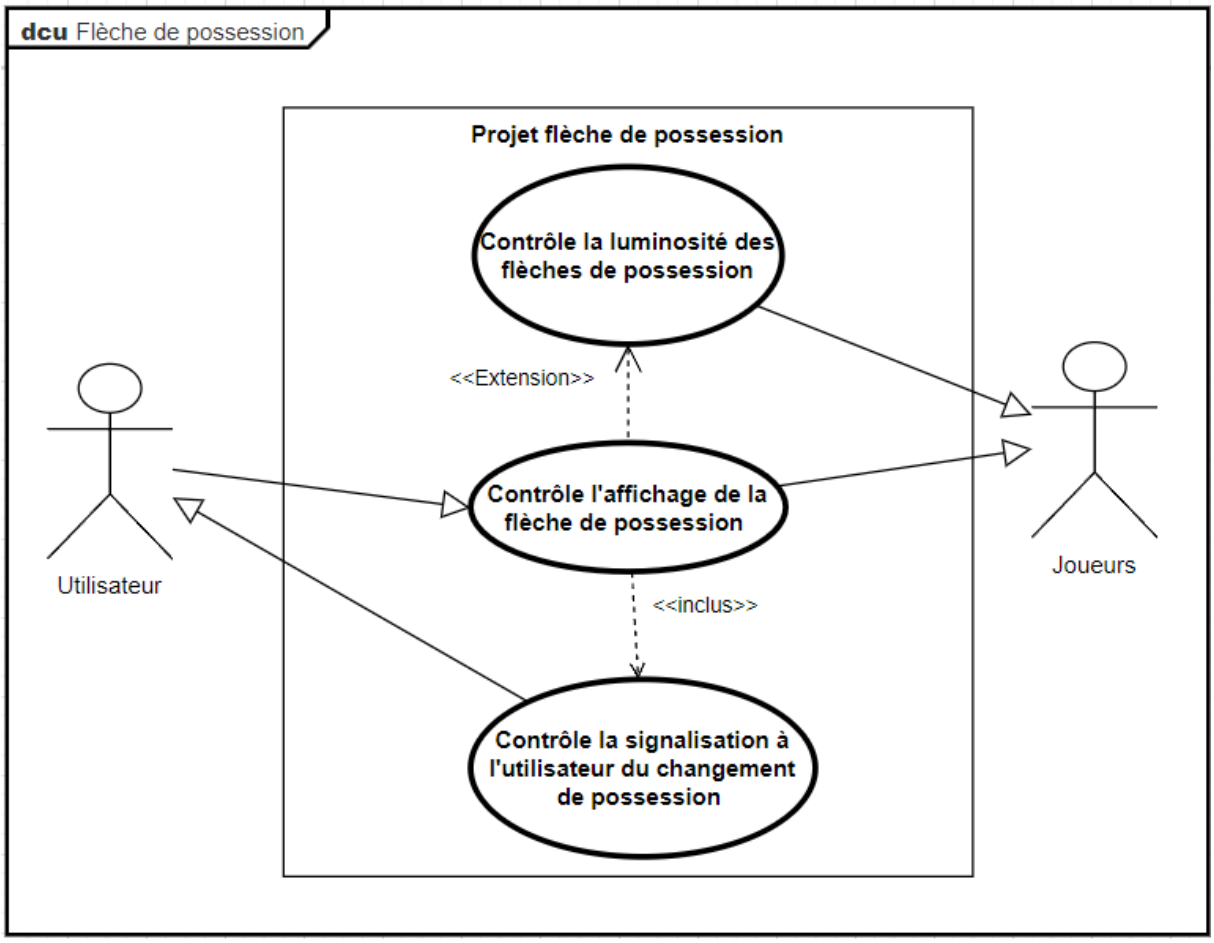


Diagramme dcu

1.3. DIAGRAMME DE GANTT

TÂCHES	Semaine 45	Semaine 46	Semaine 47	Semaine 48	Semaine 49
Réalisation des carte LEDs					
Choix des LEDs					
Réalisation du schéma des cartes					
Réalisation du routage des cartes					
Assemblage des cartes					
Test					
Réalisation de la carte driver					
Analyse de la solution					
Réalisation du schéma de la carte					
Réalisation du routage de la carte					
Assemblage de la carte					
Test					
Programmation					
Recherche et test de la programmation sur bread bord					
Programmation des fonctions séparées					
Programmation du programme principal					
Implémentation et test sur le projet					
Réalisation du dossier technique					

Diagramme de gantt

2. REALISATION

2.1. ORGANIGRAMME GLOBAL DU SYSTEME

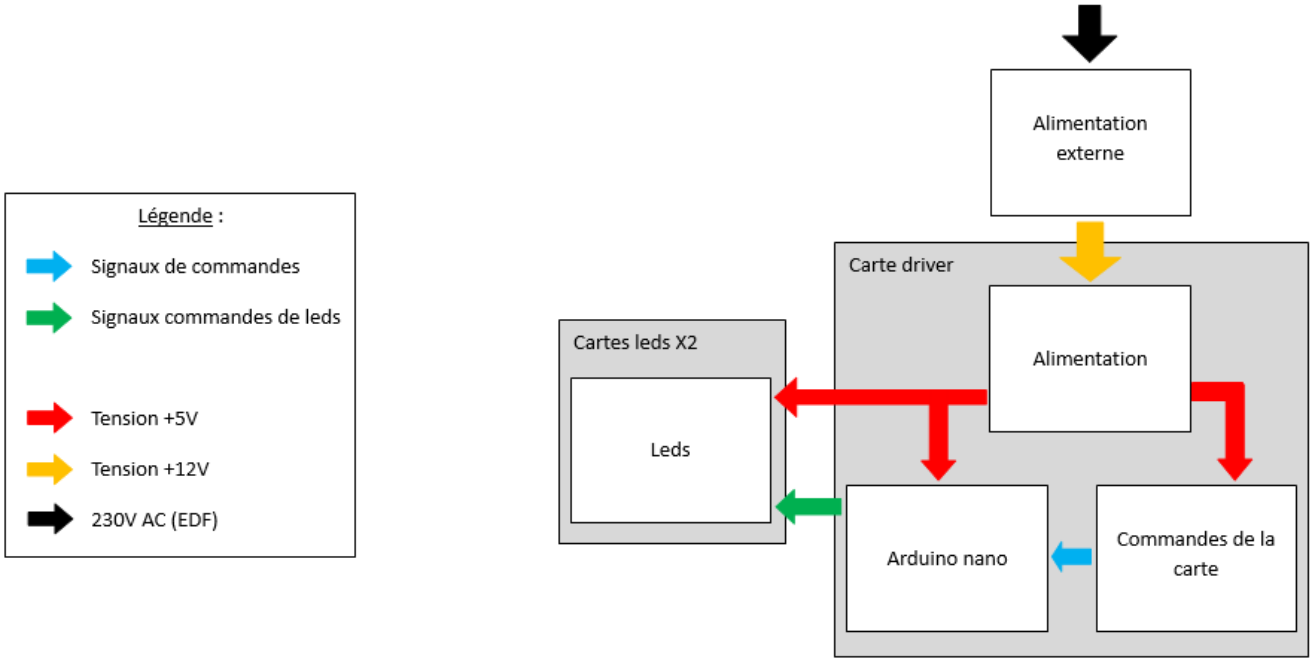


Schéma global du projet

Pour accomplir ce projet, nous avons pensé à cette réalisation. Tout d'abords nous avons la carte driver, qui est alimentée par une alimentation externe qui va délivrer du +12 V à la carte. Cette alimentation externe sera branchée au secteur. Ensuite le circuit d'alimentation de la carte driver va alimenter toutes les parties de la carte (Arduino nano, ainsi que le circuit commande de la carte) mais aussi les 2 cartes LEDs. Puis, nous avons le circuit de commande de la carte qui va servir à l'utilisateur à contrôler le système. Ce circuit va envoyer les signaux de commandes directement au microcontrôleur de la carte driver qui est un Arduino nano. Enfin, cette carte Arduino enverra les signaux de commandes pour piloter les LEDs.

2.2.REALISATION DES CARTES LED

Le but de ces cartes est d'afficher la possession, c'est-à-dire qu'il y aura 2 flèches similaires à réaliser. Nous les appellerons les cartes LED puisque le but est de faire une flèche avec des LEDs afin qu'elle puisse être vue par les joueurs. De plus, elles devront pouvoir communiquer avec la carte driver.

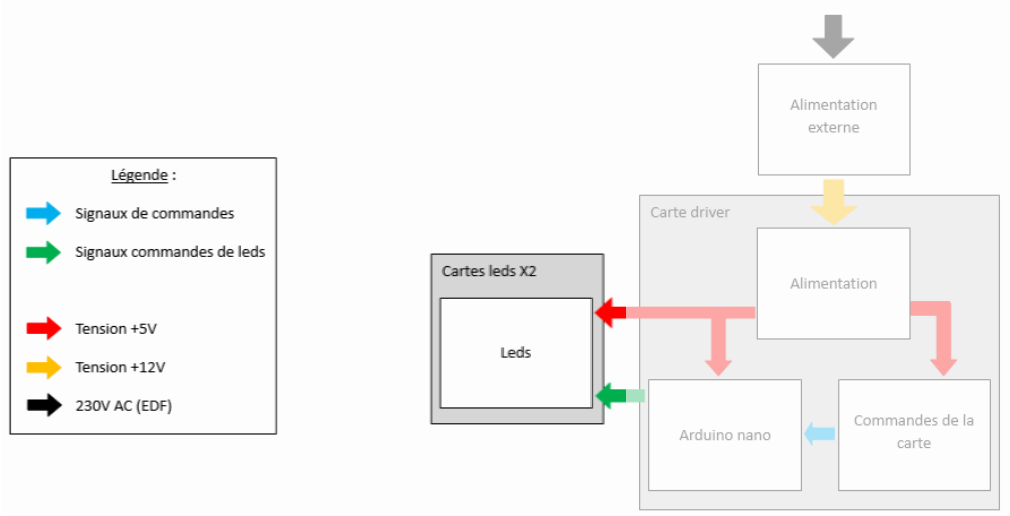


Schéma global du projet

2.2.1. CHOIX DES LEDS

Pour commencer la réalisation des cartes LEDs constituant les flèches de possession au basket, j'ai dû déterminer le choix des LEDs à utiliser :

Type de LEDs	LED standard rouge	LED RGB – WS2812B
Image du composant		
Courant utilisé (mA)	10	15-30
Intensité lumineuse (mcd)	5	390-420 (rouge)
Nombre de broches	2	4
Avantages	- Consommation faible	- Forte luminosité - RGB -> RVB (rouge, vert et bleu) - Programmable
Inconvénients	- Faible luminosité - Non programmable	- Consommation

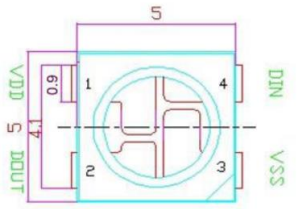
Comme on peut le voir sur ce tableau, la meilleure solution est la LED RGB – WS2812B. En effet, la LED a une intensité lumineuse beaucoup plus élevée que la LED standard rouge, ce qui est intéressant pour un affichage des flèches de possession qui doivent être visibles jusqu'à au moins 15 mètres. Le désavantage de cette forte luminosité est la consommation de courant en raison de la couleur émise par la LED (ex : en blanc, on a une consommation maximum, car association du rouge, bleu et vert). Enfin, cette LED est RGB et surtout programmable ce qui nous épargne un circuit électronique pour piloter la LED.

DESCRIPTION DE LA LED RGB – WS2812B

La LED RGB – WS2812B est une LED RGB programmable, c'est-à-dire que chaque LED peut être pilotée pour prendre une couleur ou une luminosité différente entre chaque LED. Ce genre de LED est très souvent utilisé dans les bandes LEDs que l'on peut retrouver chez nous.

Le fonctionnement de la LED est le suivant : On a 4 broches sur la LED : une correspondant à l'alimentation qui est de +5V (broche 1), deux autres à la communication avec la LED (broches 2 et 3) et la 4ème à la masse (broche 4). De plus, on peut commander la LED pour définir la couleur mais aussi la luminosité.

PIN configuration

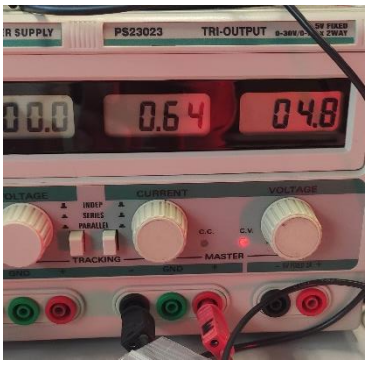


PIN function

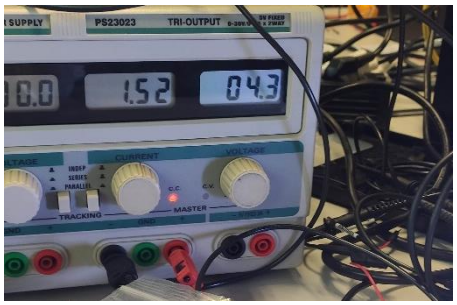
NO.	Symbol	Function description
1	VDD	Power supply LED
2	DOUT	Control data signal output
3	VSS	Ground
4	DIN	Control data signal input

Extrait de la datasheet WS2812B

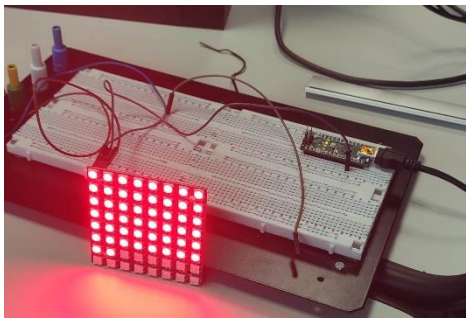
Pour ce qui est de la consommation, celle-ci n'est pas notée et cela est dû aux caractéristiques de cette LED. En effet, comme dit précédemment, le fait de pouvoir commander la couleur, ainsi que la luminosité empêche de définir la consommation de la LED puisque cette dernière varie en fonction de la couleur mais aussi de la luminosité. Pour se donner un ordre d'idée nous avons effectué des tests à l'aide d'une matrice de LED WS 2812B comme vous pouvez le voir ci-dessous :



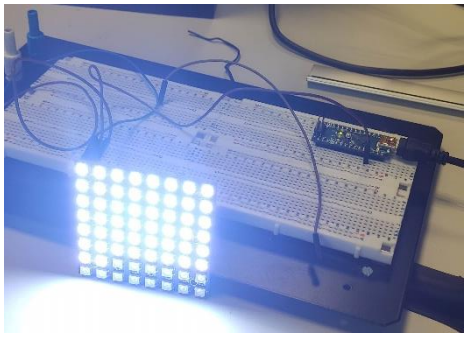
Consommation de courant lorsque 50 LEDs rouges sont allumées



Consommation de courant lorsque 50 LEDs blanches sont allumées



Test 50 LEDs rouges sont allumées sur breadbord



Test 50 LEDs blanches sont allumées sur breadbord

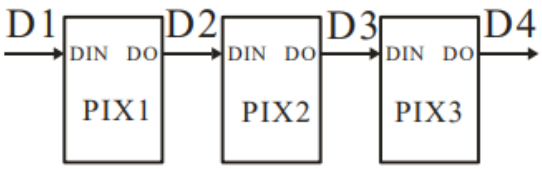
Nous pouvons donc constater que lorsque 50 LEDs s'affichent en rouge, avec un maximum de luminosité, la consommation est de 640 mA, ce qui nous fait 12,8 mA par LED. Avec les LEDs de couleur blanche, toujours avec un maximum de luminosité, la consommation est à de 1,52 A, ce qui nous fait 30,4 mA par LED. Donc la consommation moyenne d'une LED est entre 15 et 30 mA à forte luminosité.

Les LEDs ws2812b étant des LEDs programmables elles nécessitent une programmation particulière.

En effet ces LEDs se codent sur 3 octets, 1 pour chaque couleur primaire soit le bleu, le rouge et le vert. Les valeurs de combinées de ces 3 octets permettent de dire à la LED quelle couleur elle doit afficher.

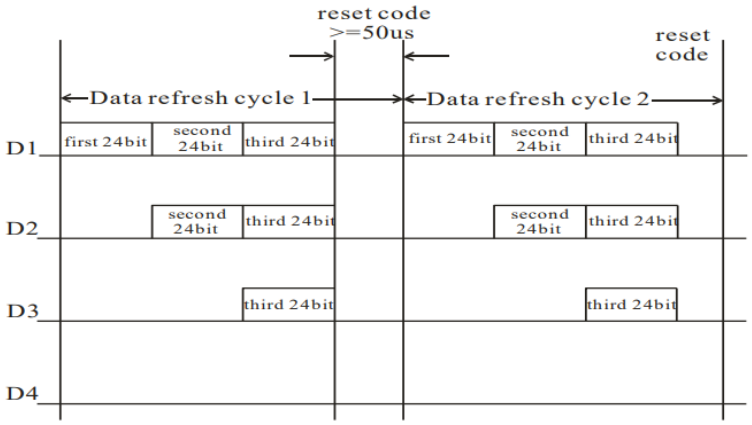
Par exemples pour afficher du blanc les 3 Octets devront être au maximum soit 255 pour chacun. Alors que pour afficher du rouge seul l'octet pour le rouge sera à 255, les autres seront à 0.

Afin de programmer plusieurs de ces LEDs grâce à un seul pin il faut les brancher en série comme sur le schéma ci-contre.



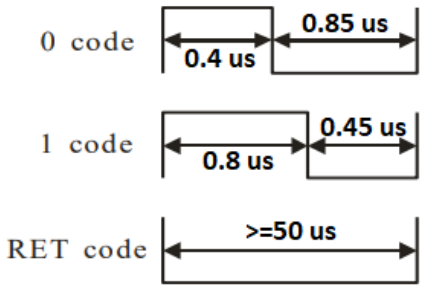
Extrait de la datasheet WS2812B

Pour ce qui est des données lues par les LEDs chaque LEDs va lire les 3 premiers octets soit les 24 premiers bits qu'elle reçoit puis va les supprimer de la trame de données avant de laisser passer le reste de ladite trame. On peut voir le fonctionnement sur le schéma ci-dessous.



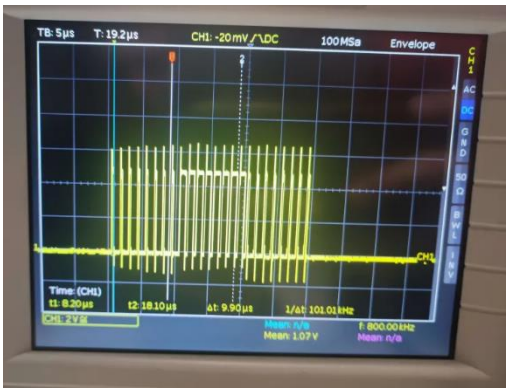
Extrait de la datasheet WS2812B

La dernière particularité de ces leds est la façon dont les données sont envoyées. Afin que la LED puisse lire un bit à 0 il faut envoyer pendant 0.4us 5V puis pendant 0.85 0V et pour lire un bit à 1 il faut envoyer 0.8us à 5v et 0.45us à 0V comme on peut le voir sur le schéma ci-contre. De plus ne rien envoyer pendant 50us ou plus permet de recommencer à écrire sur la première LED.



Extrait de la datasheet WS2812B

Enfin nous avons un exemple de trame qui correspond à la couleur rouge, on a bien le premier et troisième octet à 0 et le seconde à 1 ce bien permet bien d'afficher la couleur rouge.



Trame permettant d'afficher la couleur rouge sur une LED

2.2.2. CONCEPTION KICAD

LEDS

Pour débuter la conception de la carte, j'ai ajouté les LEDs, après avoir déterminé le nombre nécessaire de LEDs par flèche. D'après le schéma ci-dessous, le nombre requis est 50 LEDs par flèche.

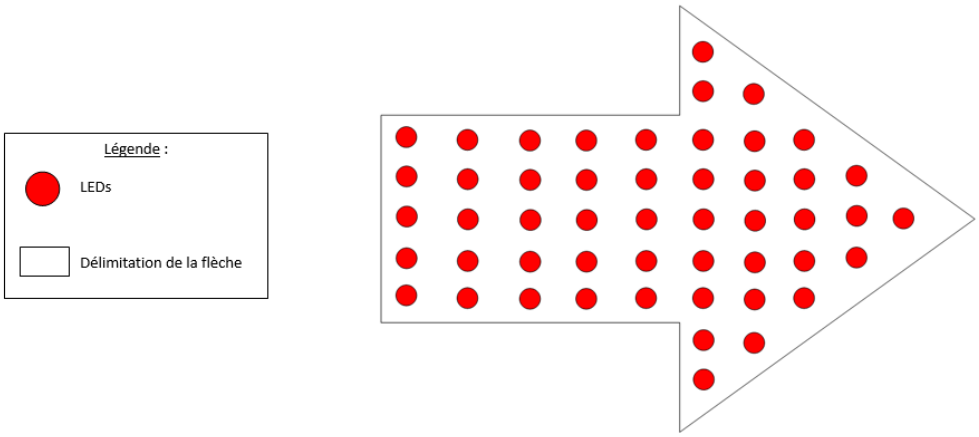


Schéma de détermination du nombre de LEDs nécessaire

Après avoir déterminé le nombre de LEDs nécessaire, j'ai commencé à réaliser le schéma de la carte LED sur Kicad :

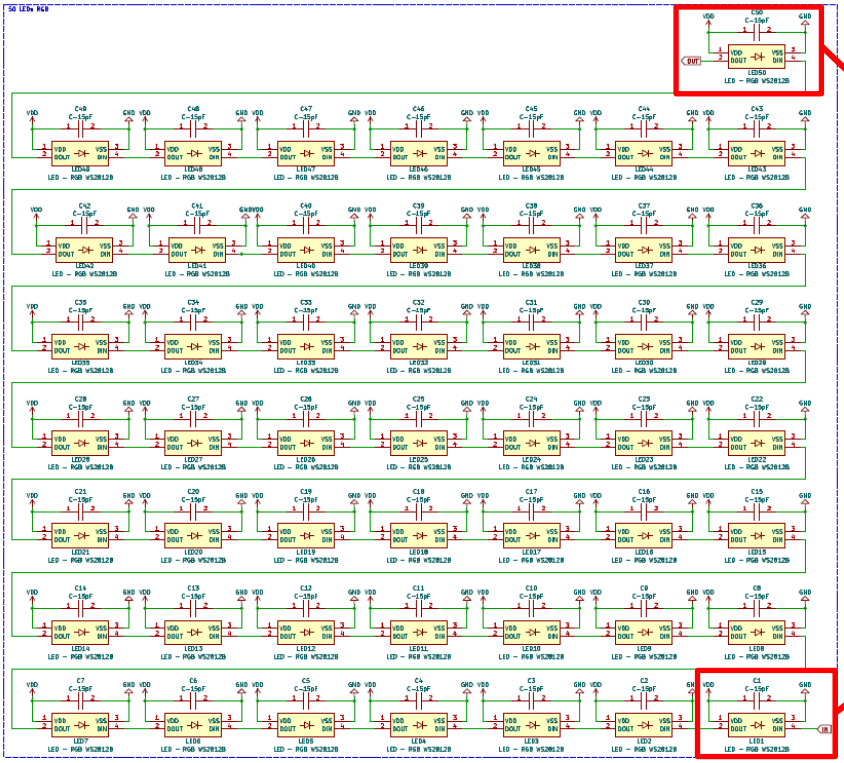


Schéma Kicad des 50 LEDs WS2812B

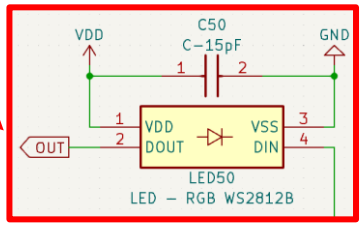


Schéma Kicad de la dernière LED WS2812B

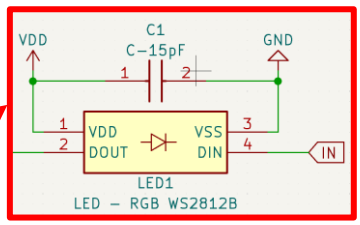


Schéma Kicad de la première LED WS2812B

Comme on peut le voir sur le schéma Kicad ci-dessus, j'ai utilisé 50 LEDs toutes connectées à l'alimentation +5 V en parallèle avec un condensateur de découplage de 15 pF. On peut voir aussi que j'ai connecté la communication des LEDs en série (DIN et DOUT), sauf pour la première et la dernière LEDs qui quant à elles sont reliées au connecteur. Pour faire ce montage, je me suis aidé de la datasheet des LEDs WS2812B, comme on peut le voir ci-dessous :

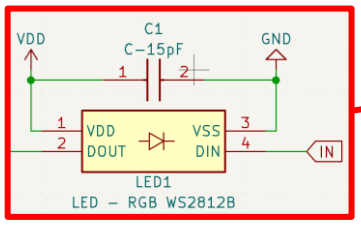


Schéma Kicad de la première LED WS2812B

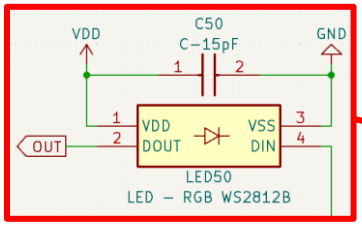
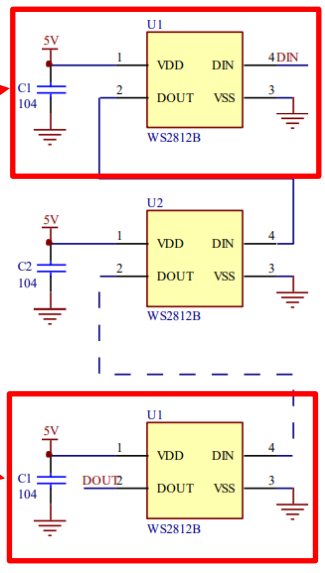


Schéma Kicad de la dernière LED WS2812B



Extrait de la datasheet WS2812B

CONNECTEUR

Après avoir réalisé la partie des LEDs des cartes, j'ai dû créer un connecteur permettant de raccorder les cartes LEDs à l'alimentation +5V, ainsi que la communication des LEDs comme nous le voyons sur Kicad :

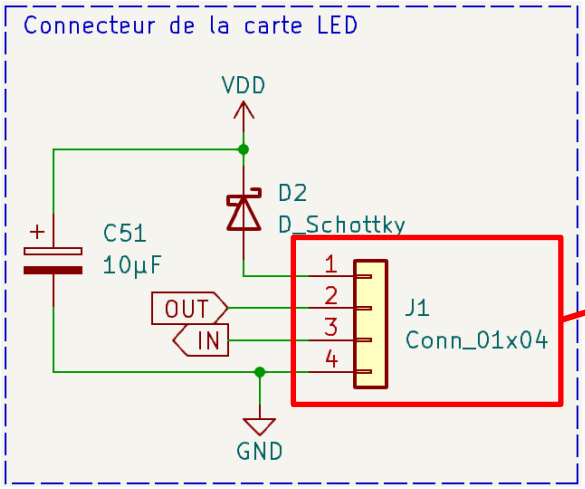
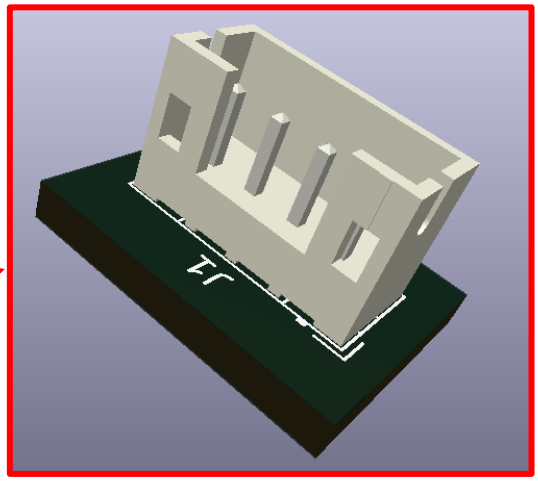
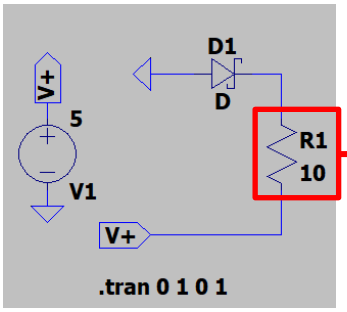


Schéma Kicad du connecteur de la carte LED

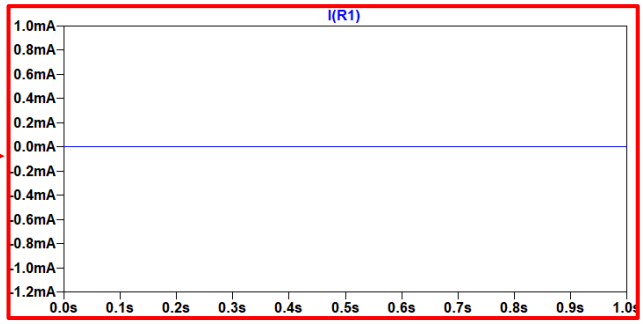


3d du connecteur JST

Comme on peut le voir, nous avons la broche 1 correspondant à l'alimentation +5 V, la broche 2 au signal de communication en sortie de la dernière LED, la broche 3 au signal de communication à l'entrée de la première LED et la broche 4 qui correspond à la masse. De plus, on peut voir que j'ai ajouté un condensateur de découplage de 10 µF afin de filtrer la tension qui arrive en entrée du connecteur. Enfin, on a une diode Schottky qui est connectée en entrée du+5 V afin de protéger la carte d'une potentielle inversion de polarité qui détruirait les 50 LEDs. Son fonctionnement est très simple. En effet, lorsque la polarité est inversée la diode va bloquer le courant traversant comme on peut le voir ci-dessous :



Simulation ltspice de la diode Schottky



Résultat de la simulation ltspice de la diode Schottky

On utilise en particulier une diode de Schottky, car cette diode a une tension de seuil plus basse que les autres diodes, mais aussi une récupération inverse plus rapide et une faible chute de tension.

Enfin, j'ai ajouté une LED, témoin d'alimentation qui pourra être utile pour du débogage afin de déterminer rapidement si la carte est sous tension. Pour cela, j'ai utilisé les LEDs vertes CMS que l'on retrouve à l'IUT et j'ai ajouté une résistance de 330 Ω, ce qui fera un courant de 15 mA pour une LED faible intensité.

$$I = \frac{U}{R} = \frac{5}{330} \approx 15\ mA$$

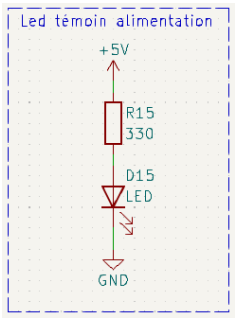
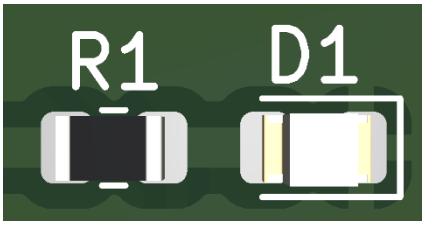


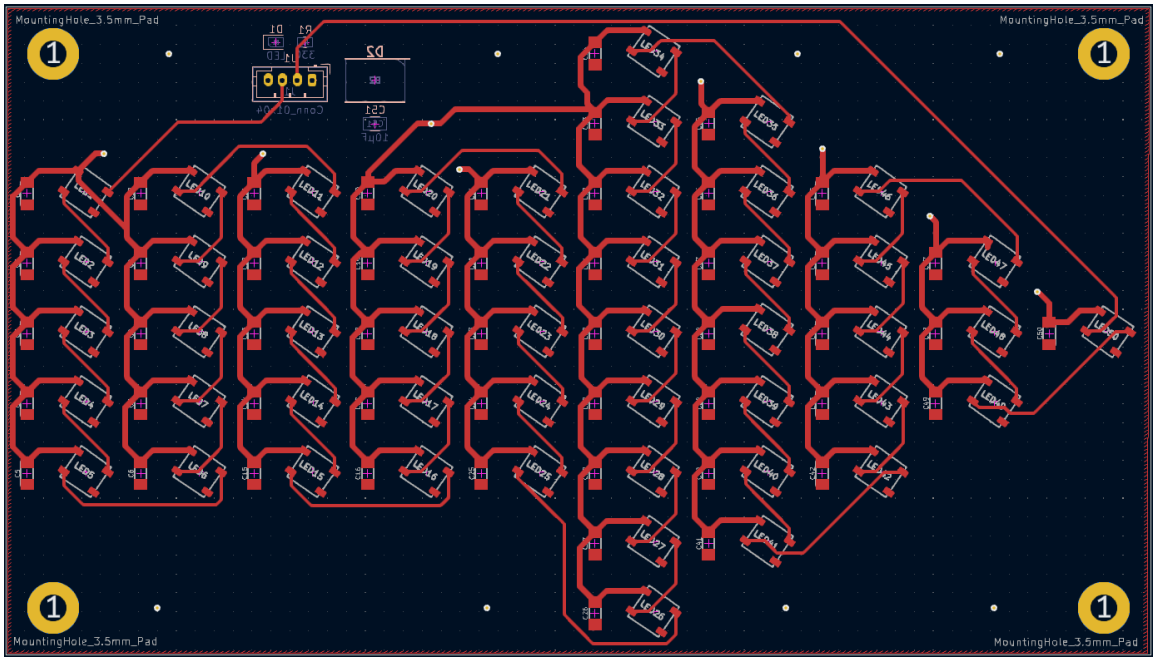
Schéma Kicad de la LED témoin

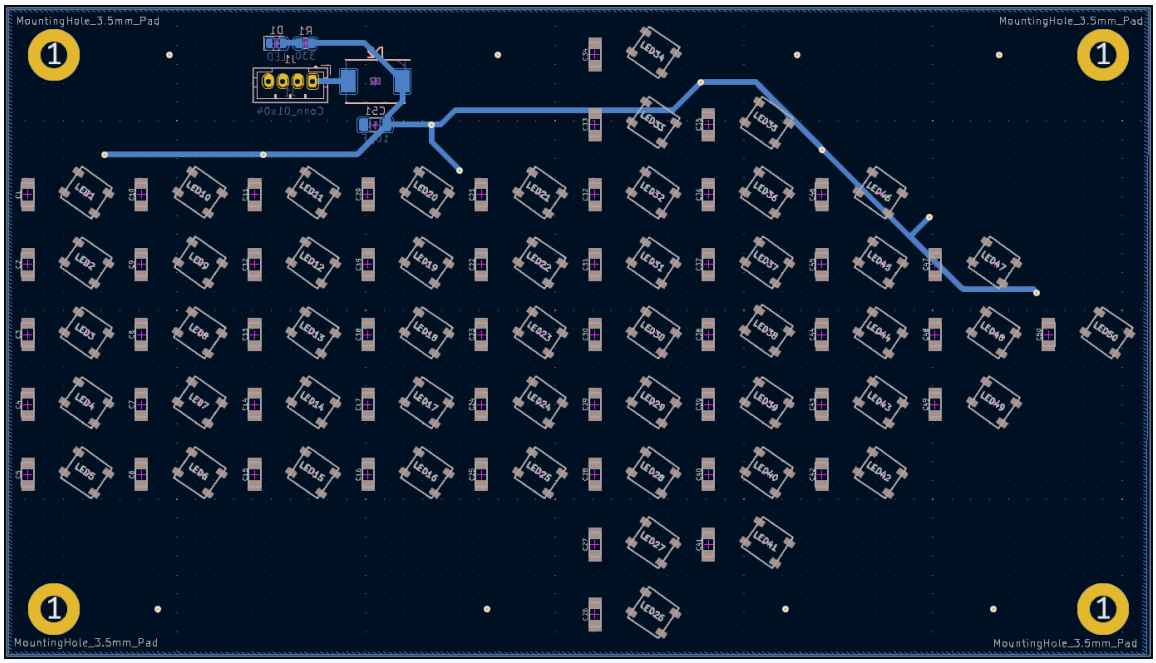


3d de la LED témoin

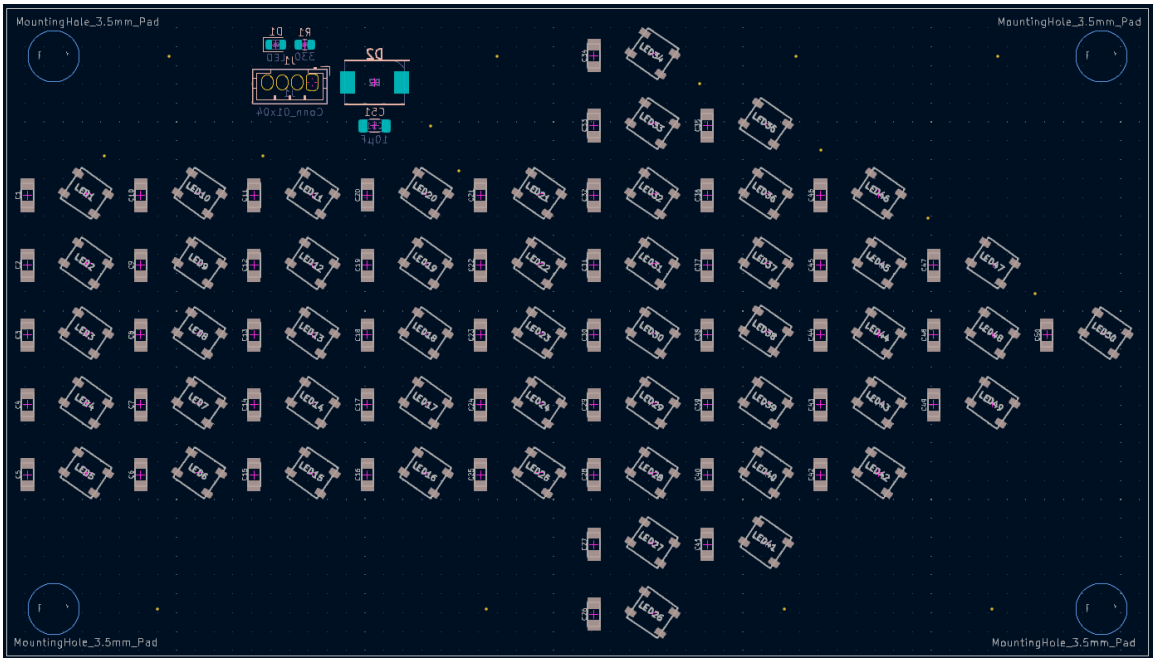
2.2.3. ROUTAGE DE LA CARTE

Pour le routage de la carte, j'ai réutilisé le schéma vu précédemment pour disposer les LEDs (figure XX) tout en ajoutant à côté les condensateurs de découplage de 15 pF. De plus, j'ai pivoté les leds de 30° afin de les router plus facilement (je me suis inspiré de l'Arduino UNO R4 Wifi). Ensuite, côté arrière, j'ai mis le connecteur, la diode de Schottky, ainsi que le condensateur de découplage de 10 µF et la LED témoin pour rendre visible les flèches de possession uniquement. Enfin, sur chaque bord de la carte, j'ai ajouté des trous de fixation pour pouvoir installer la carte dans un futur boîtier sachant que cette carte fait 153,67 mm de largeur et 87,12 mm de hauteur.

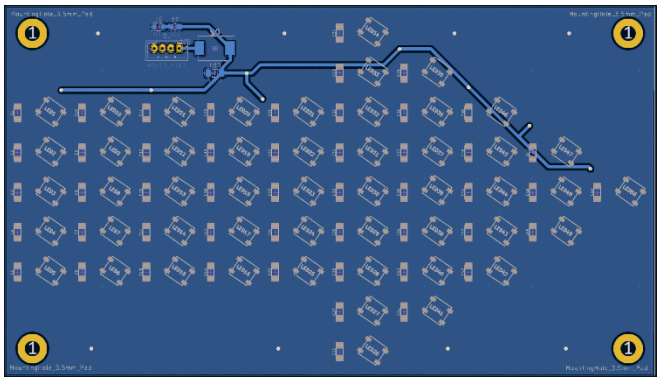




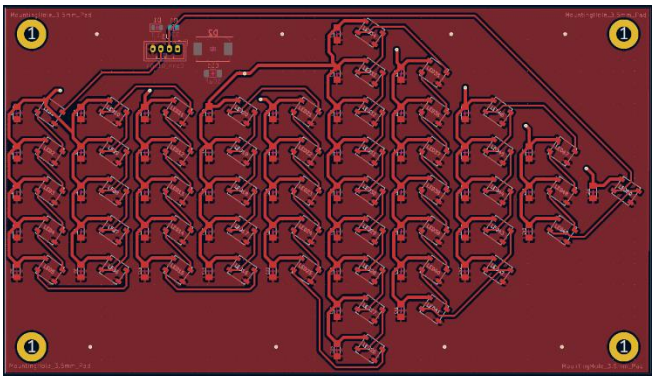
Vision bottom du routage



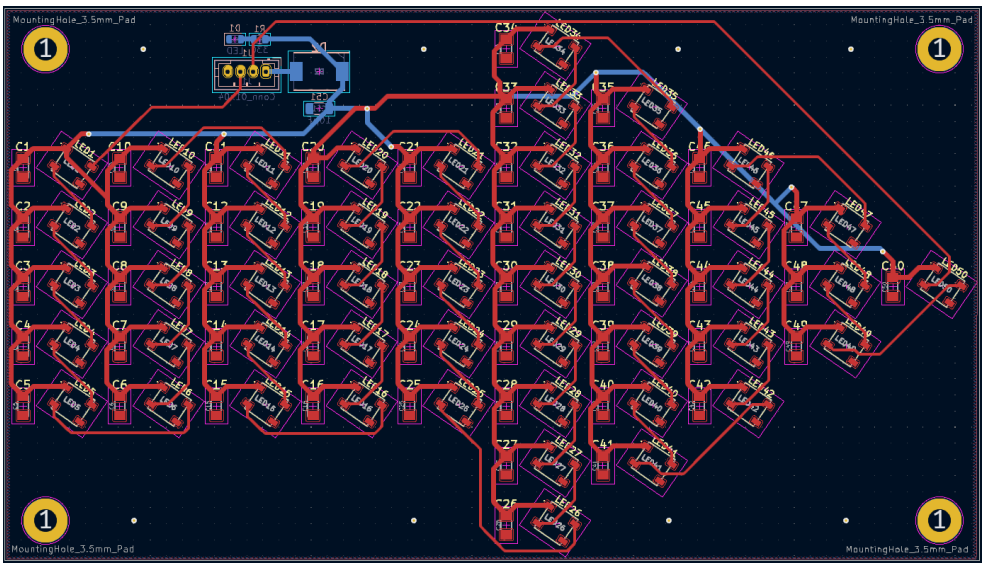
Vision des composants seulement



Vision bottom du routage avec plan de masse



Vision avant du routage avec plan de masse

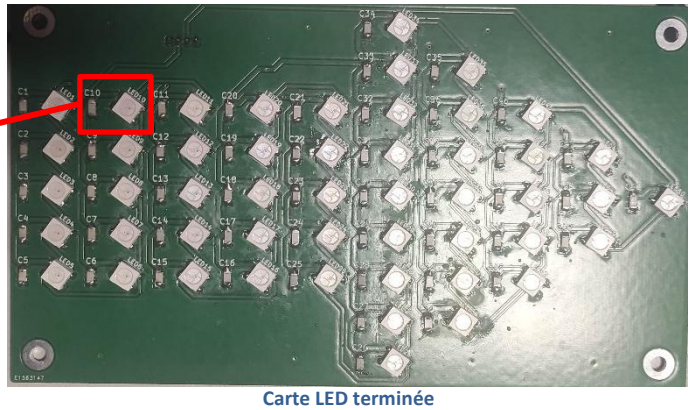


Vision global du routage sans plan de masse

2.2.4. RESULTAT FINAL ET TEST

Finalement, la carte est parfaitement réussie, la soudure des composants s'est bien passée malgré la panne du four à CMS, qui nous a contraint à souder tous les composants à la main. De plus, la LED témoin n'a finalement pas été soudée sur la carte, puisque nous n'en avons pas eu besoin pour du débogage.

LED soudée à la main



Carte LED terminée

Et pour ce qui est des fonctionnalités de la carte, celle-ci fonctionne parfaitement, chaque LEDs est pilotable comme on peut le voir ci-dessous et surtout chaque LEDs fonctionne.

Chenillard de la carte LED

Toutes les LEDs de la carte allumées

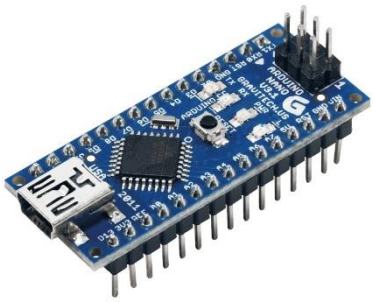
2.3. REALISATION DE LA CARTE DRIVER

L'objectif de la carte driver est de permettre à l'utilisateur de commander les deux cartes LED à l'aide de boutons. Voici les différentes fonctionnalités nécessaires pour la carte driver :

- Un composant programmable pour commander les LEDs RGB.
- Un potentiomètre pour varier l'intensité lumineuse.
- Une LED témoin pour vérifier la présence de tension.
- Un connecteur d'entrée pour l'alimentation et deux sorties pour les cartes LEDs.
- Un régulateur pour s'adapter à la tension des LEDs.
- Des LEDs d'indications pour l'utilisateur. Ces LEDs formeront 2 flèches à l'arrière pour indiquer quelles cartes LEDs sont allumées vers les arbitres.
- 3 boutons-poussoirs : 2 pour allumer chacune des LEDs et un autre pour faire clignoter les LEDs d'indications.

2.3.1. PARTIE MICROCONTROLEUR

La première étape pour la carte driver fut de choisir quel type de composants programmables nous allions utiliser. Deux choix s'offraient à nous : implanter directement le composant sur la carte avec tout ce qui permet de le faire fonctionner (quartz, régulateur de tension, etc.). Cette solution aurait pris du temps car il faut penser à tout ce qui va servir à faire fonctionner le composant programmable ainsi que les différentes connexions possibles et aurait nécessiter des quartz, des ports USB, des adaptateurs de tension, etc. Le choix de commander une carte Arduino nous a donc paru beaucoup plus pertinent. Un simple branchement suffit à la faire fonctionner. Au vu des fonctionnalités dont nous avions besoin (nombre de broches, alimentation, taille, fonctionnalité), une carte Arduino Nano nous suffisait.



Arduino nano

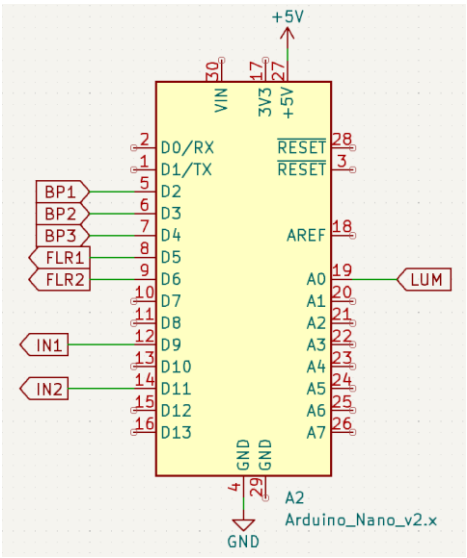


Schéma Arduino nano

2.3.2. PARTIE INTERACTION UTILISATEURS

Les signaux de commande BP1, BP2 et BP3 en entrée sont les signaux provenant des boutons-poussoirs afin d'allumer, éteindre et faire clignoter les LEDs.

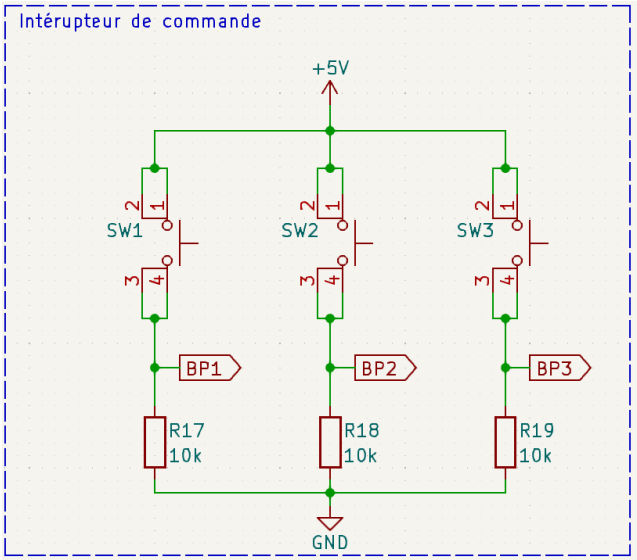


Schéma de câblage bouton poussoir

L'autre signal d'entrée sur la carte Arduino, nommé "Lum", est le signal provenant du potentiomètre pour régler l'intensité lumineuse des LEDs RGB. Nous avons choisi un potentiomètre de 10K Ω car cela nous permet d'avoir une plage de valeurs assez élevée.

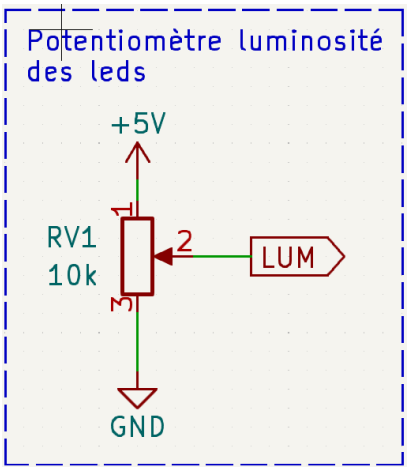


Schéma de câblage potentiomètre

Pour le choix des boutons-poussoirs et du potentiomètre, nous avons décidé de faire avec ceux que nous avions déjà à l'IUT.

Bouton poussoir - site : radiospare

Potentiomètre – site : radiospare

2.3.3. PARTIE LEDS

Dans la partie carte LEDs, nous avons pu voir pourquoi nous avons opté pour des LEDs RGB de type WS2812B. Ici, nous ne sommes pas confrontés à la problématique de la luminosité, car les LEDs ne seront vues que par les utilisateurs qui seront à quelques centimètres. Nous avons donc fait le choix de partir sur l'autre type de LEDs, à savoir les TLHR5400. L'avantage est leur faible consommation, cependant, mais elles ne sont pas programmables.

Pour les piloter, nous avons étudié deux possibilités : utiliser un transistor soit bipolaire, soit un MOSFET.

Transistor bipolaire	Transistor mosfet
Commande par une source de courant	Commande par une source de tension
Accumulation de charge dans la base et le collecteur	Pas d'effet d'accumulation
Vitesse de commutation moyenne	Vitesse de commutation élevée
Impédance d'entrée faible	Impédance d'entrée forte

Au vu des comparaisons ci-dessus, nous avons choisi de prendre un MOSFET. La raison principale est que dans notre montage, nous commandons le transistor avec une tension qui provient de la carte Arduino.

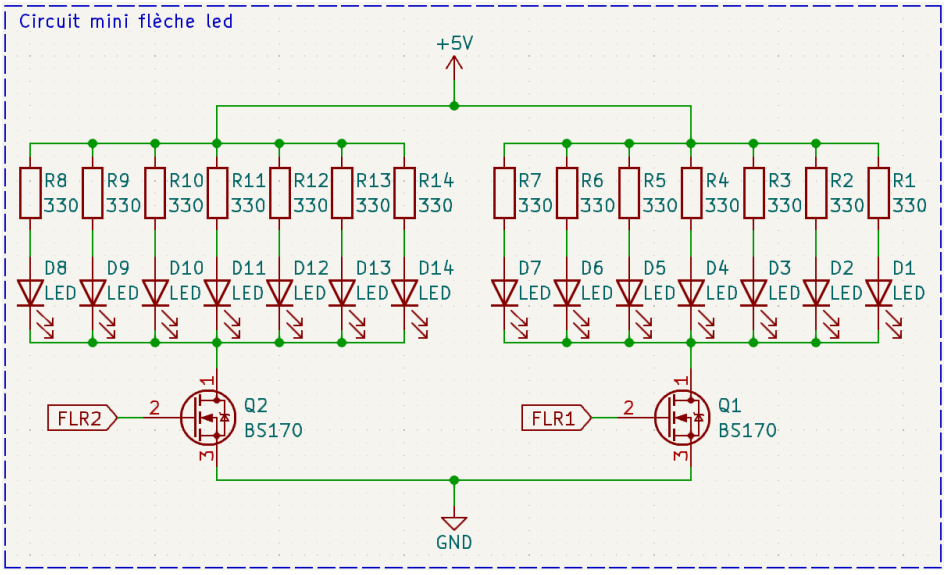
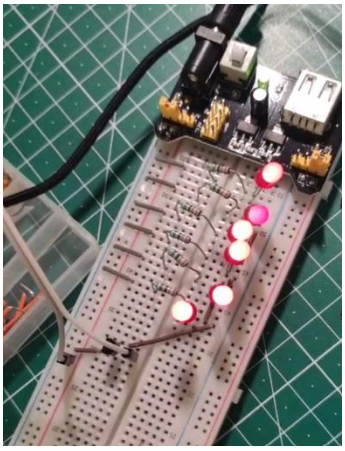


Schéma de câblage LEDs et transistors

Comme on peut le voir, nous utilisons des résistances de 333Ω au-dessus de chaque LED. Cela permet d'avoir une intensité de 15 mA.

I=U/R => I=5/333 => I=15mA

Pour être sûr de notre câblage, nous avons testé ce montage en amont. Nous avons placé quelques LEDs sur une table de tests, commandées par le transistor ci-dessus. Ensuite, nous avons envoyé la commande à partir d'une Raspberry Pi.



Test transistor mosfet sur breadbord

2.3.4. LED TÉMOIN

Afin de s'assurer que la carte est alimentée, nous avons ajouté une LED témoin. Nous avons pris une LED verte identique à celles décrites ci-dessus.

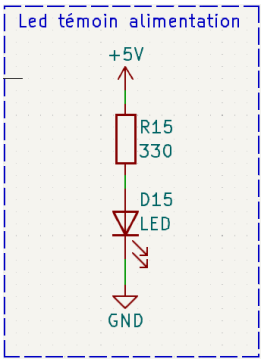


Schéma de câblage LED témoin

2.3.5. PARTIE CONNECTEUR

Afin de connecter la carte driver aux deux autres cartes LEDs, nous avons ajouté des connecteurs permettant d'envoyer les trames générées par l'Arduino.

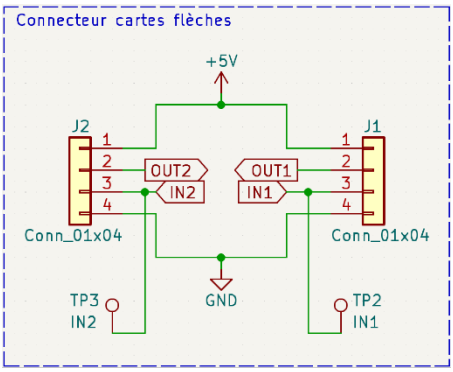
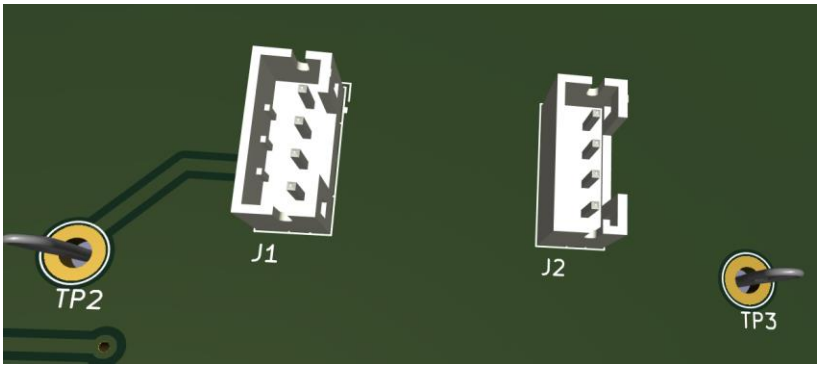


Schéma de câblage connecteur



Vision 3D connecteur

Les signaux IN1 et IN2 sont les trames envoyées, OUT1 et OUT2 sont celles reçues par les LEDs. Le système de renvoi des trames par les LEDs vous sera détaillé plus loin.

Les points de mesure TP2 et TP3 servent à visualiser les trames envoyées.

2.3.6. PARTIE ALIMENTATION

Comme on a pu le voir, nos cartes sont alimentées en 5V. L'objectif de la partie alimentation est donc de passer d'une tension de 230V prise sur le secteur à une tension de 5V.

Pour cela, nous gérons l'alimentation en deux étapes. Nous avons commencé par récupérer une alimentation de PC à l'IUT. Elle délivre du 12V en continu et est capable de fournir jusqu'à 5A. La référence est "PA-1061-0".

Quand le 12V arrive sur notre carte driver, il faut le convertir en 5V. Le plus simple est d'utiliser un régulateur. Il en existe différents types. Nous avons décidé de partir sur un Traco de référence TEN 25-1211 car il est réputé pour sa stabilité électrique, son rendement particulièrement élevé (84%). De plus, celui que nous avons choisi intègre directement des condensateurs de filtrage.

Il nous faut aussi un interrupteur afin d'allumer ou éteindre le système.

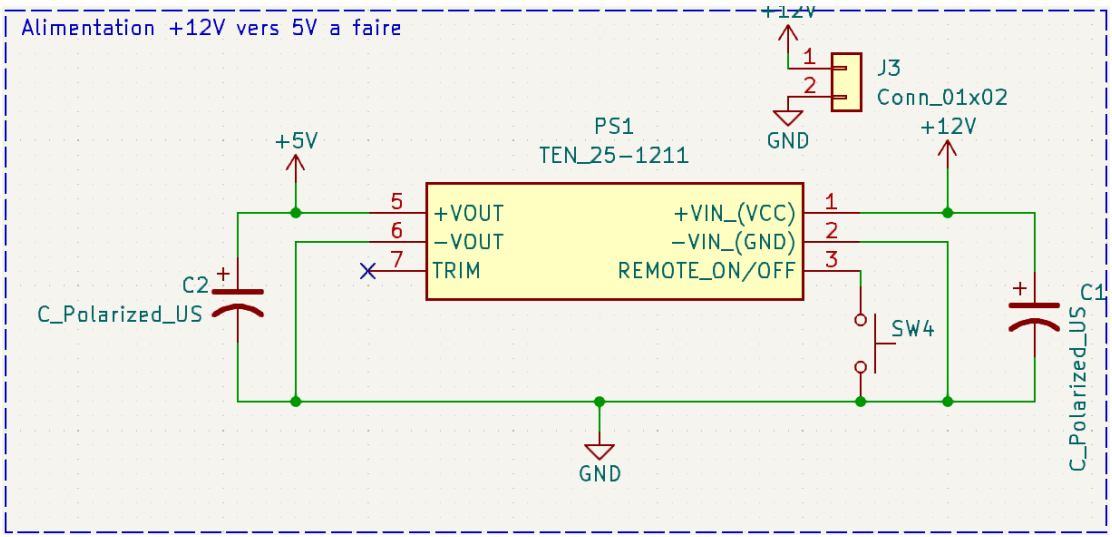
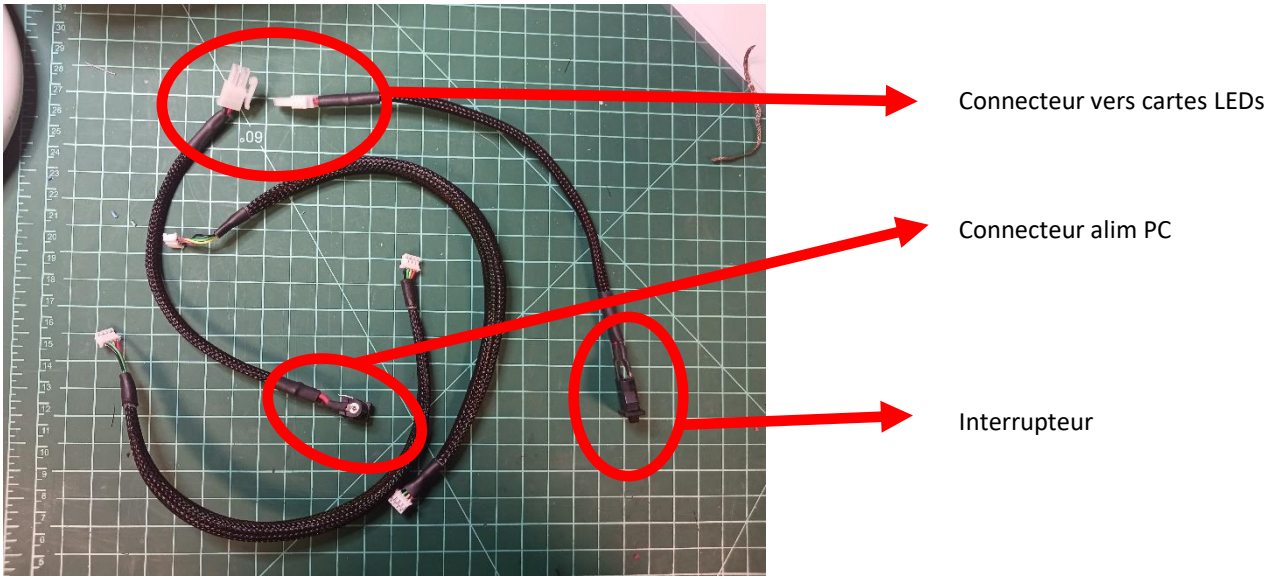


Schéma de câblage régulateur

Comme on peut le voir, le Traco a un mode on/off déjà intégré. Il nous a donc suffi de rajouter un interrupteur sur cette entrée.

Les condensateurs C1 et C2 servent à filtrer le signal. Des condensateurs de filtrage sont déjà présents dans le Traco, mais nous avons voulu en rajouter pour s'assurer d'avoir un signal propre et maximiser la durée de vie des composants.

Pour le connecteur, nous avons pris le vis-à-vis du connecteur PC.



Câbles connectés à la carte driver

2.3.7. SCHEMA COMPLET

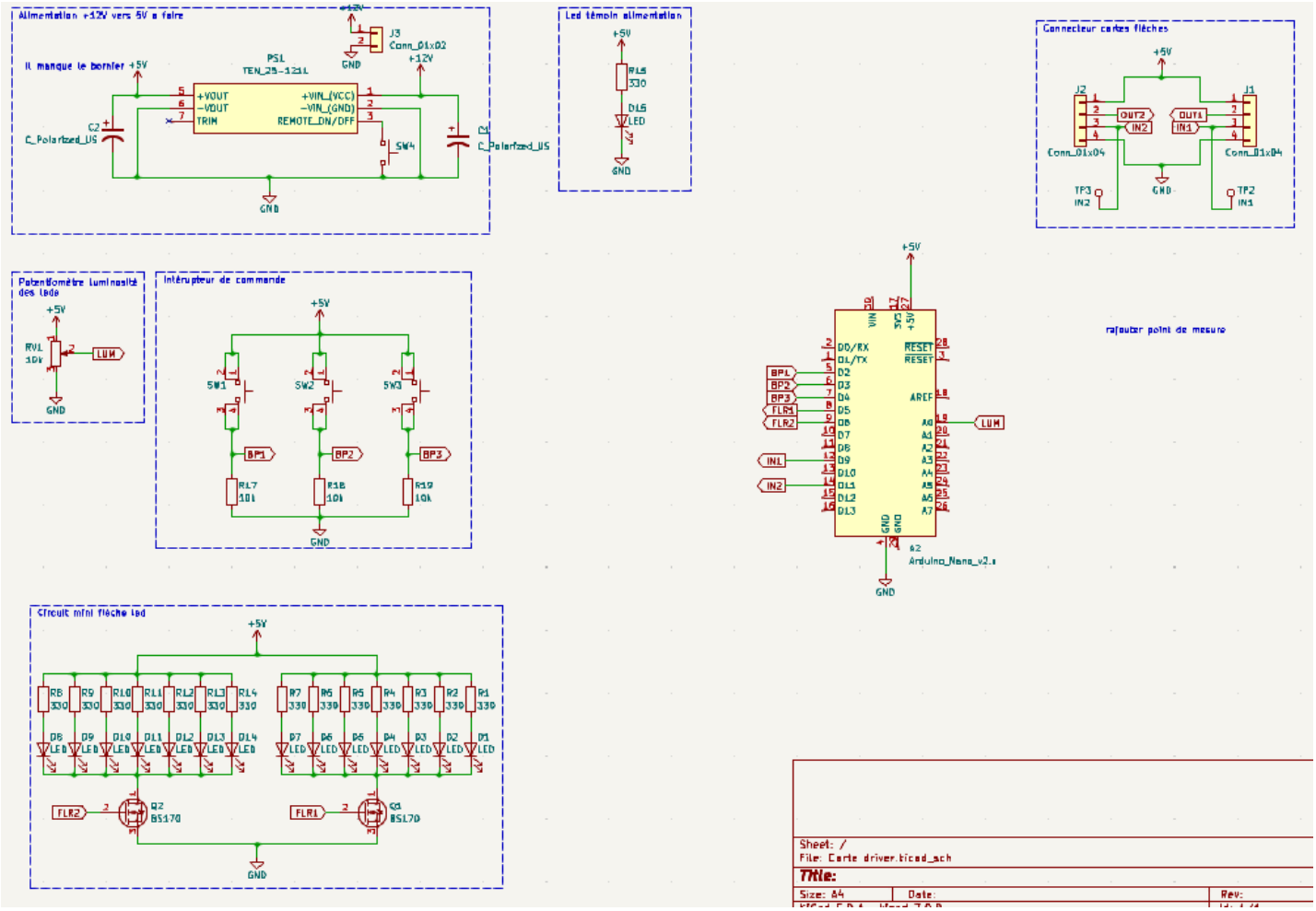
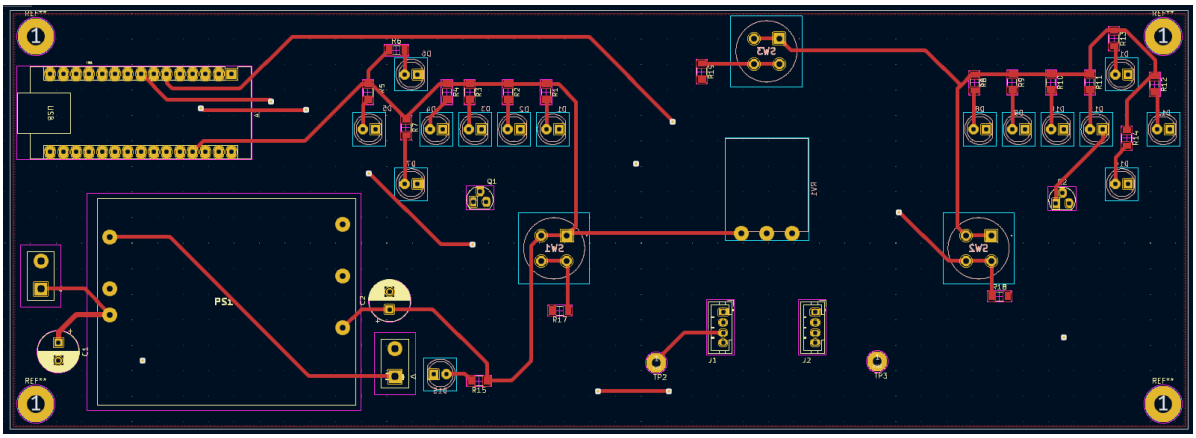


Schéma complet de la carte driver

2.3.8. PARTIE ROUTAGE

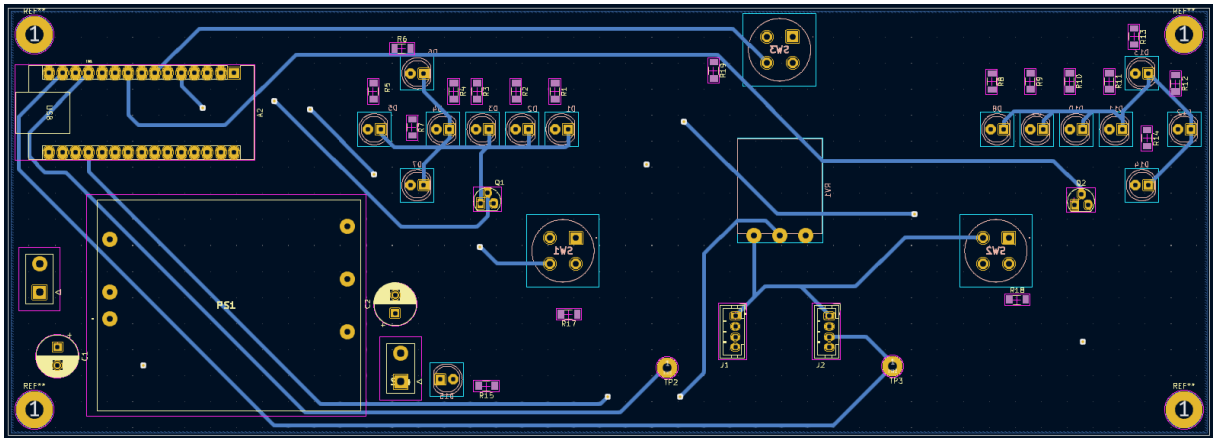
Une fois le schéma complet, il a fallu effectuer le routage. L'enjeu principal était la position des LEDs et des boutons-poussoirs pour faciliter l'utilisation. La place n'était pas une grande contrainte du moment qu'elle ne dépassait pas la taille des deux cartes LEDs réunies. Le bon placement des connecteurs était important afin que les connexions soient le plus simple possible. Ce qui nous donne le routage suivant.

Routage vision top :



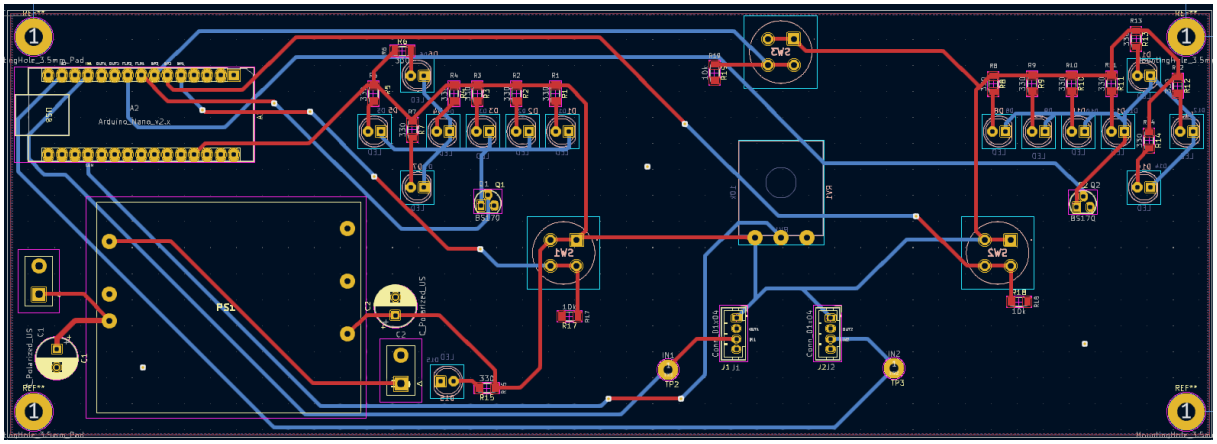
Routage Top

Routage vision bottom :



Routage Bottom

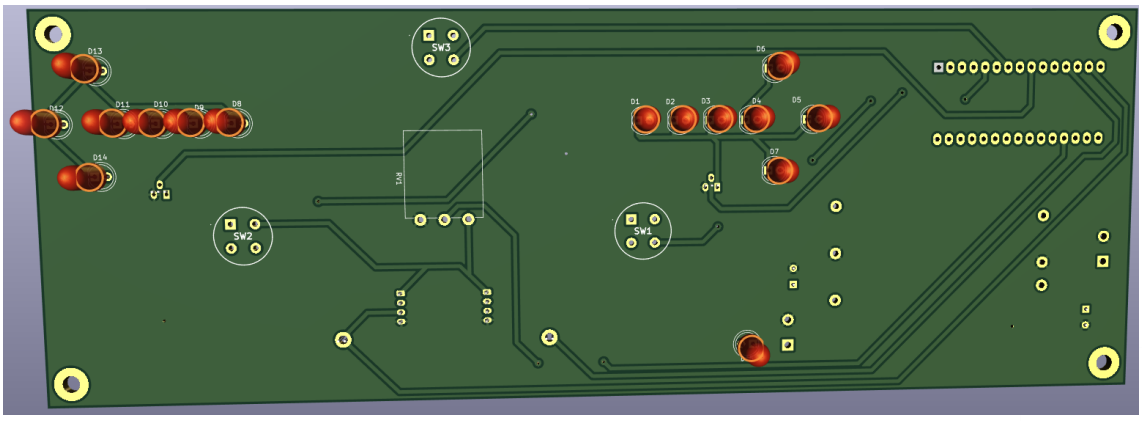
Routage complet (sans plan de masse) :



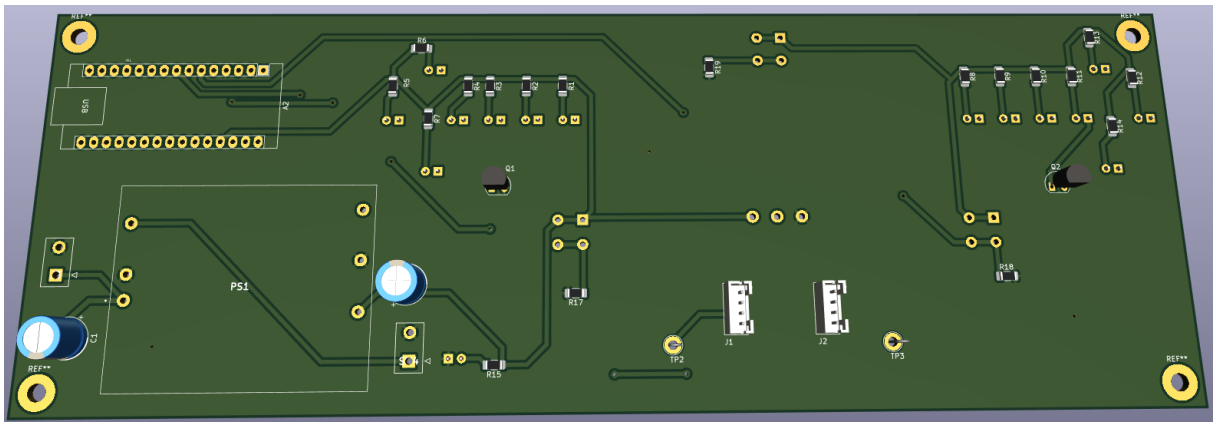
Routage Bottom

On peut voir en haut à gauche la carte Arduino. Elle est placée sur un côté pour que la prise USB soit facilement accessible. On peut également observer les deux flèches ainsi que les boutons-poussoirs. Les boutons-poussoirs situés en dessous des LEDs servent à les allumer, celui du dessus sert à les faire clignoter. Au centre, on retrouve le potentiomètre qui permet d'intensifier ou de diminuer la luminosité émise par les LEDs RGB.

Pour le placement des composants, nous nous sommes aidés de la vision 3D offerte par le logiciel KiCad. Cette vision nous a permis de nous rendre compte de la forme de la carte et de l'emplacement des composants.



Vision 3D de la carte

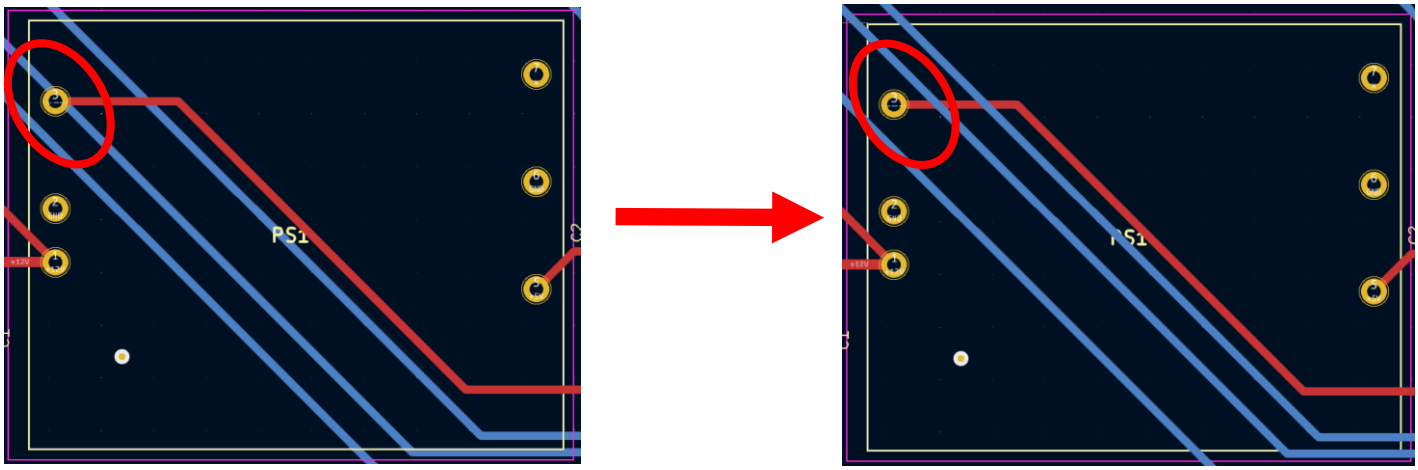


Vision 3D de la carte

Certains composants n'apparaissent pas sur la vision 3D car leurs modèles 3D n'ont pas été associés à leur empreinte.

2.3.9. PARTIE TEST

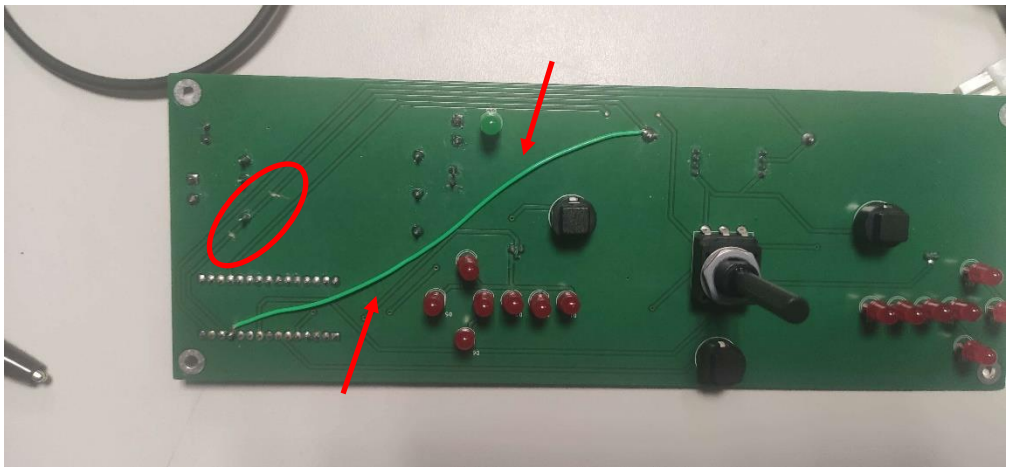
Une fois la carte reçue et soudée, il fallait la tester. Lors de la mise sous tension, rien ne s'est passé. En effectuant des mesures, nous avons constaté que la tension arrivait bien sur la carte à l'entrée du Traco, mais qu'il n'y avait aucune tension en sortie. Nous avons identifié un défaut sur le routage. Initialement, le routage n'était pas comme décrit ci-dessus ; l'une des pistes était en court-circuit avec le Traco.



Erreur de routage

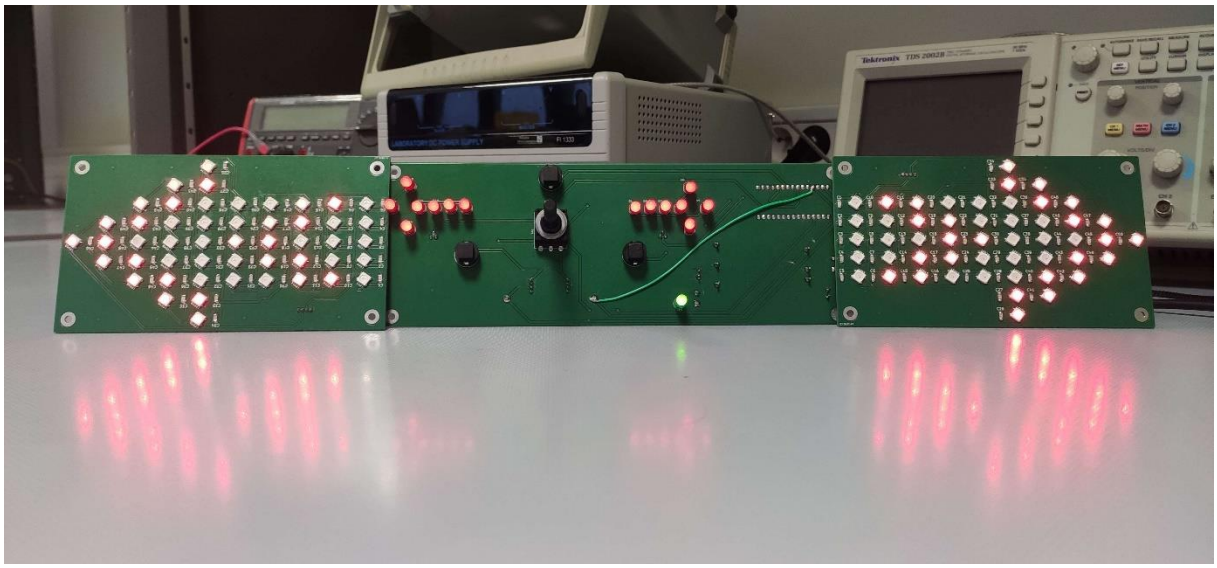
Correction routage

Cette piste relie l'un des signaux de l'Arduino, « IN1 », au connecteur, mais comme on peut le voir ci-dessus, la piste était noyée dans le pin on/off. Nous avons donc coupé la piste pour empêcher le contact, et nous avons effectué une reprise filaire entre ces deux points.



Correction carte driver

Une fois cette modification apportée, notre carte a parfaitement fonctionné.



Projet final

2.4. PROGRAMATION

2.4.1. BIBLIOTHEQUE FASTLED

Afin de contrôler les LEDs ws2812b il existe une bibliothèque Arduino appelée FastLED.h déjà existante et facilitant grandement le contrôle des LEDs. Afin de pouvoir utiliser cette bibliothèque il faut créer un tableau d'objets de type CRGB qui est constitué d'un octet pour chaque couleur primaire et qui symbolise pour chaque entrée du tableau une LED distincte. On assigne ensuite à ce tableau un type de LED ici ws2812b et un pin vers lequel les données sortiront avec la fonction LEDS.addLeds< type de LED, pin des données> (tableaux d'objets CRGB, nombre de LEDs dans le tableau).

On peut ensuite assigner à chaque cellule du tableau une valeur CRGB et envoyer les données sur le pin avec la fonction FastLED.show() ;

On dispose aussi d'une fonction FastLED.setBrightness(valeur) ; qui permet de régler la luminosité des LEDs avec pour valeur 0 étant éteint et 255 étant la pleine luminosité.

2.4.2. BIBLIOTHEQUE LUMI

Cette bibliothèque a pour objectif de régler la luminosité des LEDs grâce au potentiomètre situé sur la broche A0 de la carte Arduino.

Pour cela on lit la valeur du potentiomètre sur la broche A0 et on réinjecte cette valeur divisée par 4 dans la fonction de la bibliothèque FastLED permettant de régler la lumineusité des LEDS.

L'appel de la fonction lumi permet donc de régler directement la lumineusité des LEDs.

```
//Lumi.h

#ifndef Lumi_h
#define Lumi_h

#define Pot_lum A0

#include <Arduino.h>
#include <FastLED.h>

void lumi(void);

#endif
```

Header de la bibliothèque Lumi.h

```
//Lumi.cpp

#include <Lumi.h>

void lumi(void){
  static int sensorValue = 0; //variable stockant la valeur du potentiomètre initialisée à 0
  sensorValue = analogRead(Pot_lum);//lit la valeur du potentiomètre comprise entre 0 et 1023
  FastLED.setBrightness(sensorValue/4);//Affecte la valeur du potentiomètre à la lumineusité.(divisé par 4 pour avoir 255 max)
}
```

Code de la bibliothèque Lumi.h

2.4.3. BIBLIOTHEQUE APPUI

Cette bibliothèque permet de détecter un front montant sur les 3 boutons poussoirs de la carte driver

Dans le header on définit les 3 pins utilisés pour les boutons poussoirs ainsi que la fonction, permettant d'initialiser les boutons en entrées ainsi que les 3 fonctions permettant de détecter le front montant sur chacun des boutons.

```
//Appui.h

#ifndef Appui_h
#define Appui_h
#include "Arduino.h"
#define Bp_d 2
#define Bp_g 3
#define Bp_m 4

void Appui_init(void);//Initialise les 3 pins en entrées
bool appui_d(void);//renvoi 1 si front montant sur pin 2 sinon 0
bool appui_g(void);//renvoi 1 si front montant sur pin 3 sinon 0
bool appui_m(void);//renvoi 1 si front montant sur pin 4 sinon 0

#endif
```

Header de la bibliothèque Appui.h

Dans la partie code on retrouve l'initialisation des 3 pins reliés aux boutons poussoirs dans la fonction Appui_init. Ensuite on a la fonction pour détecter le front montant sur le bouton poussoir du milieu grâce à la lecture de la valeur du bouton poussoir et au stockage de l'état précédent dans la variable etat_precedent. On ajoute un délai de 10 ms afin d'éviter les rebonds après chaque changement d'état du bouton.

Les deux autres fonctions appui_g et appui_d reprennent exactement le même code à part la lecture des données qui se font sur les respectivement sur les pins 3 (Bp_g) et 2 (Bp_d).

Cette fonction retourne donc 1 quand elle détecte un front montant et 0 sinon.

```
//Appui.cpp

#include "Appui.h"

void Appui_init(void){//Initialise les 3 pins en entrées
  pinMode(Bp_d, INPUT);
  pinMode(Bp_g, INPUT);
  pinMode(Bp_m, INPUT);
}

bool appui_d(void){//renvoi 1 si front montant sur pin 2 sinon 0
  static unsigned char etat_precedent = 0;//variable etat précedent propre à chaque fonction
  if(digitalRead(Bp_d)==1 && etat_precedent==0 ){//si bouton droit appuyé et n'était pas appuyé avant
    etat_precedent=1;//état précédent à 1 pour éviter de re-rentrer dans cette condition
    delay(10);//délai anti-rebond
    return 1;// retourne 1
    }
  else if (digitalRead(Bp_d)==0 && etat_precedent==1){//si bouton relâché
        etat_precedent=0;//permet de redétecter un front montant
        delay(10);//délai anti-rebond
        return 0;//retourne 0
  }
  else return 0;// si état précédent=état actuel retourne 0
}
```

Code de la bibliothèque Appui.h

2.4.4. BIBLIOTHEQUE CHENILLARD

La bibliothèque Chenillard contient la fonction du même nom permettant de créer un chenillard.

Le paramètre step correspond à l'étape dans lequel le chenillard est. Cette étape va de 0 à 9 et le tableau LEDs correspond au LEDs de la flèche avant pour laquelle on veut créer le chenillard.

```
//Chenillard.h

#ifndef Chenillard_h
#define Chenillard_h

#include <Arduino.h>
#include <FastLED.h>

void chenillard(int step, struct CRGB leds[]);
//fonction permettant de créer un chenillard sur le tableau leds[] en incrément step de 0 à 9
#endif
```

Header de la bibliothèque Chenillard.h

Afin de créer ce chenillard on doit allumer et éteindre les LEDs manuellement lors de chaque état.

On retrouve ci-dessous le code permettant de paramétrer les 3 premiers états. Dans celui-ci on utilise la structure CRGB afin d'assigner directement la couleur voulue à une LED distincte.

Il faut noter que la couleur noire correspond à l'état éteint de la LED.

```
//Chenillard.cpp

#include<Chenillard.h>

void chenillard(int step, struct CRGB leds[]){
//fonction permettant de créer un chenillard sur le tableau leds[] en incrément step de 0 à 9
  switch(step){//switch comprant les 10 états du chenillard
    case 0 ://éteint les LEDs de l'état 9 et allume la troisième LED
      leds[25]=CRGB::Black;
      leds[33]=CRGB::Black;
      leds[34]=CRGB::Black;
      leds[40]=CRGB::Black;
      leds[41]=CRGB::Black;
      leds[45]=CRGB::Black;
      leds[46]=CRGB::Black;
      leds[48]=CRGB::Black;
      leds[49]=CRGB::Black;

      leds[2]=CRGB::Red;
    break;

    case 1 ://éteint la LED de l'état 0 et allume des LEDs afin de créer une flèche
      leds[2]=CRGB::Black;

      leds[1]=CRGB::Red;
      leds[3]=CRGB::Red;
      leds[7]=CRGB::Red;

    break;

    case 2 ://éteint la flèche précédente et allume des LEDs pour créer la flèche suivante
      leds[1]=CRGB::Black;
      leds[3]=CRGB::Black;
      leds[7]=CRGB::Black;

      leds[0]=CRGB::Red;
      leds[4]=CRGB::Red;
      leds[6]=CRGB::Red;
      leds[8]=CRGB::Red;
      leds[12]=CRGB::Red;
    break;
```

Code de la bibliothèque Chenillard.h

2.4.5. BIBLIOTHEQUE ETATS

La bibliothèque Etats a pour but de gérer les sorties pour les différents états du code principal. On a donc en premier la fonction Etats_init() permettant d'initialiser les tableaux de LEDs pour réaliser les flèches du devant et configure les pins 5 et 6 en sorties, ceux-ci sont reliés aux flèches situées à l'arrière. Enfin on initialise la luminosité à 25% de la luminosité max dans le cas où la fonction lumi n'est pas utilisée.

```
//Etats.h

#ifndef Etats_h
#define Etats_h

#include <Arduino.h>
#include <FastLED.h>
#include <Chenillard.h>

#define FASTLED_ALL_PINS_HARDWARE_SPI
#define LED_PIN 9 //Broche du tableau de LEDs gauche
#define LED_PIN2 11//Broche du tableau de LEDs droite
#define NUM_LEDS 50//définit le nombre de LEDs dans le tableau
#define Led_d 5 //Broche de la flèche arrière droite
#define Led_g 6 //Broche de la flèche arrière gauche

void Etats_init(void);//Initialise les sorties et Les tableaux de LEDS

void switch_code(unsigned char Etat);
//Fonction gérant les sorties en fonction de l'état dans lequel la variable Etat se trouve

#endif
```

Header de la bibliothèque Etats.h

Nous avons ensuite la fonction switch_code qui a pour but d'écrire les valeurs sur les Flèches en fonction de la variable Etat que l'on lui donne en paramètre.

Ci-dessous on retrouve un tableau qui synthétise les sorties en fonction de l'état.

Etats\sorties	Flèche arrière gauche	Flèche arrière droite	Flèche avant gauche	Flèche avant droite
0	Eteint	Eteint	Eteint	Eteint
1	Clignote	Clignote	Chenillard	Chenillard
2	Eteint	Allumé	Eteint	Allumé
3	Allumé	Eteint	Allumé	Eteint
4	Eteint	Clignote	Eteint	Allumé
5	Clignote	Eteint	Allumé	Eteint

Tableaux des sorties en fonction des Etats

Le Chenillard correspond à l'utilisation de la fonction chenillard provenant de la bibliothèque du même nom. Il faut noter que la fréquence du clignotement et du changement d'état du chenillard s'effectue toutes les 250 ms tant que l'on est dans un état qui le commande. Ci-dessous, on retrouve le code commenté des fonctions de la bibliothèque Etats. Il faut noter que la fonction FastLED.show est nécessaire car elle permet d'envoyer les valeurs des tableaux Fleche_d et Fleche_g vers leurs flèches correspondantes sur la maquette.

De plus, dans l'état 1, on utilise un modulo 10 sur la variable "e" afin que celle-ci reste entre 0 et 9, permettant ainsi à la fonction chenillard de tourner en boucle. Enfin, on utilise une boucle for dans tous les états afin d'assigner à toutes les LEDs d'un tableau la même valeur.

```
//Etats.cpp

#include <Etats.h>
CRGB Fleche_d[NUM_LEDS];//Tableau de LEDs de la flèche droite
CRGB Fleche_g[NUM_LEDS];//Tableau de LEDs de la flèche gauche


void Etats_init(void){//Initialise les sorties et Les tableaux de LEDS

  LEDS.addLeds<WS2812B, LED_PIN, GRB>(Fleche_g,NUM_LEDS);//Initialise le tableau de LEDs de la flèche gauche sur broche 9
  LEDS.addLeds<WS2812B, LED_PIN2, GRB>(Fleche_d,NUM_LEDS);//Initialise le tableau de LEDs de la flèche droite sur broche 11
  pinMode(Led_d, OUTPUT);//Initialise la Broche 5 en sortie
  pinMode(Led_g, OUTPUT);//Initialise la Broche 6 en sortie
  FastLED.setBrightness(64);//Met la lumineusité des LEDs à 25% (64/256=25%)
}

void switch_code(unsigned char Etat){//Fonction gérant les sorties en fonction de l'état dans lequel la variable Etat se trouve

  static unsigned char e=0;//Variable stockant l'état du chenillard
  static bool clignote=0;//Variable pour alterner l'état des LEDs afin de les faire clignoter
  switch (Etat){//Attribue les sorties en fonction de l'état actuel
  case 0 ://Etat dans lequel tout est éteint
    digitalWrite(Led_d,0);//Eteint flèche arrière droite
    digitalWrite(Led_g,0);//Eteint flèche arrière gauche
    for(int i=0; i<50; i++){//Permet de modifier les 50 LEDs du tableau
      Fleche_d[i]=CRGB::Black;//Eteint le tableau de droite
      Fleche_g[i]=CRGB::Black;//Eteint le tableau de gauche
    }
    FastLED.show();//Met à jour les tableaux de LEDs
  break;

  case 1 ://Etat dans lequel les flèches avants font un chenillard et celles arrières clignotent
    digitalWrite(Led_d,clignote);//Fait clignoter flèche arrière droite
    digitalWrite(Led_g,clignote);//Fait clignoter flèche arrière gauche
    clignote=!clignote;//alterne l'état des LEDs à chaque exécution de l'état
    e=(e+1)%10;//boucle les états du chenillard
    chenillard(e,Fleche_g);//Fonction chenillard pour fleche gauche
    chenillard(e,Fleche_d);//Fonction chenillard pour fleche droite
    chenillard((e+5)%10,Fleche_g);//Fonction chenillard pour flèche gauche
    chenillard((e+5)%10,Fleche_d);//Fonction chenillard pour flèche droite
    FastLED.show();//Met à jour les tableaux de LEDs
    delay(250);//Délai de 250 ms entre clignotement et changement d'état du chenillard
  break;

  case 2 ://Etat dans lequel les flèches gauches sont allumées
    digitalWrite(Led_d,0);//Eteint flèche arrière droite
    digitalWrite(Led_g,1);//Allume flèche arrière gauche
    for(int i=0; i<50; i++){//Permet de modifier les 50 LEDs du tableau
      Fleche_d[i]=CRGB::Black;//Eteint le tableau de droite
      Fleche_g[i]=CRGB::Red;//Allume le tableau de gauche en rouge
    }
    FastLED.show();//Met à jour les tableaux de LEDs
  break;
  case 3 ://Etat dans lequel les flèches droites sont allumées
    digitalWrite(Led_d,1);//Allume flèche arrière droite
    digitalWrite(Led_g,0);//Eteint flèche arrière gauche
    for(int i=0; i<50; i++){//Permet de modifier les 50 LEDs du tableau
      Fleche_d[i]=CRGB::Red;//Allume le tableau de droite en rouge
      Fleche_g[i]=CRGB::Black;//Eteint le tableau de gauche
    }
    FastLED.show();//Met à jour les tableaux de LEDs
  break;

  case 4 ://Etat dans lequel la flèche gauche avant est allumé et celle à l'arrière clignote
    digitalWrite(Led_d,0);//Eteint flèche arrière droite
    digitalWrite(Led_g,clignote);//Fait clignoter flèche arrière gauche
    clignote=!clignote;//alterne l'état des LEDs à chaque exécution de l'état
    delay(250);//Délai de 250 ms entre clignotement
    for(int i=0; i<50; i++){//Permet de modifier les 50 LEDs du tableau
      Fleche_d[i]=CRGB::Black;//Eteint le tableau de droite
      Fleche_g[i]=CRGB::Red;//Allume le tableau de gauche en rouge
    }
    FastLED.show();//Met à jour les tableaux de LEDs
  break;

  case 5 ://Etat dans lequel la flèche droite avant est allumé et celle à l'arrière clignote
    digitalWrite(Led_d,clignote);//Fait clignoter flèche arrière droite
    digitalWrite(Led_g,0);//Eteint flèche arrière gauche
    clignote=!clignote;//alterne l'état des LEDs à chaque exécution de l'état
    delay(250);//Délai de 250 ms entre clignotement
    for(int i=0; i<50; i++){//Permet de modifier les 50 LEDs du tableau
      Fleche_d[i]=CRGB::Red;//Allume le tableau de droite en rouge
      Fleche_g[i]=CRGB::Black;//Eteint le tableau de gauche
    }
    FastLED.show();//Met à jour les tableaux de LEDs
  break;

  default ://Etat par défaut dans lequel tout est éteint
    digitalWrite(Led_d,0);//Eteint flèche arrière droite
    digitalWrite(Led_g,0);//Eteint flèche arrière gauche
    for(int i=0; i<50; i++){//Permet de modifier les 50 LEDs du tableau
      Fleche_d[i]=CRGB::Black;//Eteint le tableau de droite
      Fleche_g[i]=CRGB::Black;//Eteint le tableau de gauche
    }
    FastLED.show();//Met à jour les tableaux de LEDs
  break;
  }
}
```

Code de la bibliothèque Etats.h

2.4.6. MAIN

On arrive finalement dans le programme principal. Dans celui-ci, on inclut les bibliothèques "Etats.h", "Appui.h" et "Lumi.h" vues précédemment. On initialise d'abord les sorties avec "Etats_init()" et les entrées avec "Appui_init()".

Ensuite, on retrouve les conditions des changements d'états qui sont résumées dans le diagramme d'états ci-dessous. Il faut noter que la fonction "lumi" se situe en dehors de la machine à états, la luminosité pouvant donc être changée dans tous les états.

```
#include <Etats.h>
#include <Appui.h>
#include <Lumi.h>

void setup() {//Initialisation
Etats_init();//Initialise les Tableaux de LEDs et les LEDs arrière
Appui_init();//Initialise les boutons en entrées
}

void loop() {//Boucle principale
  static unsigned char Etat =0;//Etat initial dans lequel tout est éteint
    if(appui_m()){//si front montant sur Bouton milieu
      if(Etat==0)Etat=1;//et si état 0 état devient 1
      if(Etat==2)Etat=4;//et si état 2 état devient 4
      if(Etat==3)Etat=5;//et si état 3 état devient 5
    }
  if(appui_d() && (Etat!=0))Etat=3;//si front montant sur Bouton droit et pas dans étant 0
  if(appui_g() && (Etat!=0))Etat=2;//si front montant sur Bouton gauche et pas dans étant 0
  switch_code(Etat);//applique l'état correspondant
  lumi();// permet de régler la lumineusité avec le potentiomètre
}
```

Code principal

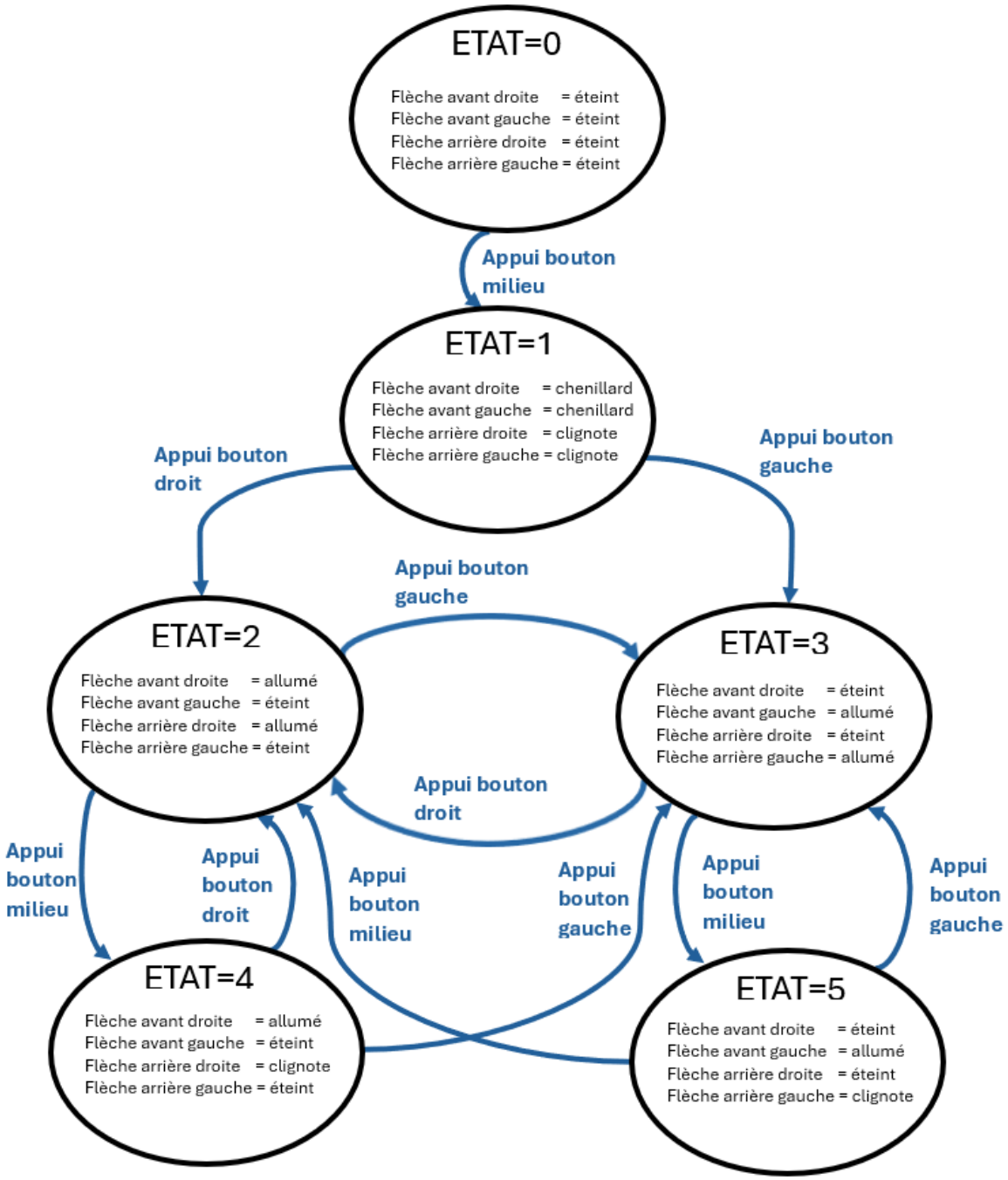


Diagramme d'état du code

L'état initial est donc l'état 0, dans cet état tout est éteint, on sort de cet état avec un appui sur le bouton milieu pour arriver dans l'état 1. Dans l'état 1 les tableaux de LEDs à l'avant réalisent un chenillard et les flèches à l'arrière clignotent.

On a alors le choix d'appuyer sur le bouton de droite ou celui de gauche qui respectivement allument les flèches avant et arrière droite ou les flèches avant et arrière gauche. Enfin un appui sur le bouton milieu lors des états 2 ou 3 permet de faire clignoter la flèche arrière qui était allumée dans l'état précédent et d'ensuite retourner dans les états 2 ou 3 en fonction du bouton appuyé.

4. CONCLUSION

En conclusion, notre projet s'est bien déroulé. Dès le début, nous avons compris les attentes qui nous étaient données et les avons intégrées à notre démarche. Nous avons rapidement identifié les premiers enjeux de notre projet afin de progresser de manière efficace. Nous avons réparti la charge de travail en trois pour avancer rapidement, tout en collaborant pour nous entraider et assurer la cohérence de chaque partie. Lors de l'assemblage de notre travail, nous avons obtenu un résultat très satisfaisant. Après une légère correction sur notre carte driver, nos flèches ont fonctionné et ont répondu aux objectifs fixés en début de projet.

Cependant, certains points pourraient être améliorés si nous devions refaire ce projet. Tout d'abord, nous avons mis un peu trop de temps avant de commencer le routage de la carte driver. Nous avons donc été pressé par le temps par la suite. En conséquence, une erreur s'est glissée sans que nous nous en rendions compte, et le routage n'était pas optimisé, pouvant être grandement amélioré. En ce qui concerne la programmation, notre code manquait de structure. Bien qu'il soit fonctionnel, nous aurions dû prendre plus de temps pour le segmenter en différentes bibliothèques afin de faciliter sa lecture et sa compréhension.

En conclusion, ce projet nous a beaucoup plu. Il nous a permis d'appliquer les connaissances acquises en cours tout en développant de nouvelles compétences.

5. PERSPECTIVE D'EVOLUTION

5.1. BOITIER

Cet objectif était déjà dans notre cahier des charges initial, cependant par manque de temps nous n'avons pas pu le réaliser. Il serait très intéressant de créer un boitier afin de pouvoir regrouper toutes les cartes en un seul ensemble et d'avoir une version du projet terminé et utilisable en situation réelle.

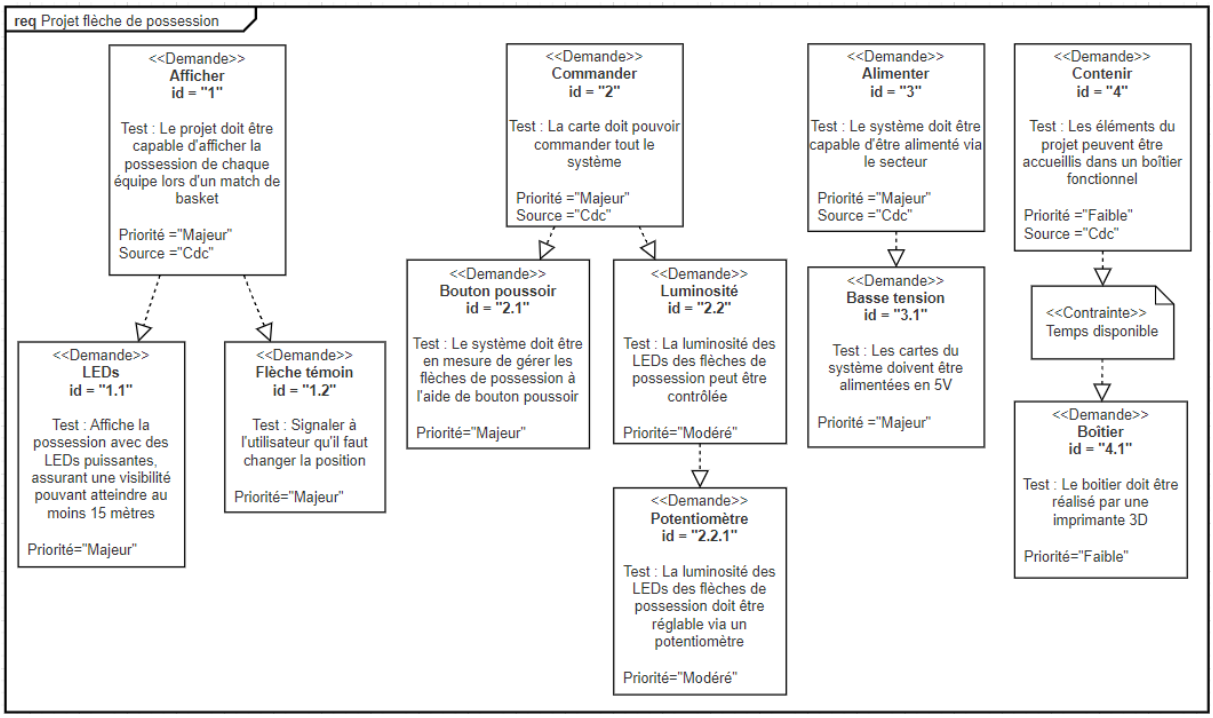


Diagramme sysml

5.2. AFFICHAGE SCORE ET TEMPS DE JEU

Nous avons pensé à créer au niveau de l'interface utilisateur, un affichage permettant de visualiser le score et le temps de jeu. Pour cela on pourrait utiliser un afficheur LCD ou pour changer un afficheur OLED. Cette perspective d'évolution pourrait être intéressante à réaliser, car elle ajoute de la plus-value à notre projet en ajoutant des fonctionnalités supplémentaires qui pourrait être utile à l'utilisateur.

Afficheur LCD - Site : Amazon

Afficheur OLED - Site : Amazon

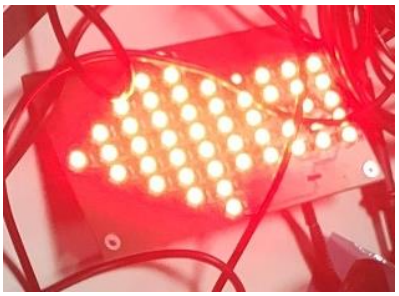
5.3. COMMUNICATION BLUETOOTH + APPLICATION

Cette évolution intéressante permettrait à l'utilisateur, de contrôler à distance les flèches de possession. De plus, comme ce qui a été dit précédemment on pourrait ajouter sur l'application, le temps de jeu et le score de chaque équipe.

5.4. ALIMENTATION

Enfin, nous avons pensé à deux évolutions possibles pour l'alimentation. Premièrement, créer nous-même l'alimentation permettant de convertir le 230 AC du secteur en +12V. Ce projet entièrement fait main est à étudier car nous ne savons pas le temps et la difficulté que cela peut prendre de créer cette alimentation.

Deuxièmement, nous avons mesuré réellement le courant utilisé par le projet. Comme on peut le voir ci-contre, nous avons une consommation de 380 mA au niveau de l'entrée 12 V du Traco, lorsque le système est au maximum de sa consommation (une flèche de possession allumée avec luminosité maximum + composants de la carte driver allumés).



Test consommation maximum du projet

Si l'on calcule le courant sous 5 V en prenant en compte le Traco qui a un rendement de 86% :

$$Pentrée = U * I = 12 * 0.38 = 4{,}56\ W$$

$$Psortie = Pentrée * \eta = 4.56 * 0.86 \approx 9.12\ W$$

$$Isortie = \frac{Psortie}{U} = \frac{9{,}12}{5} \approx 1{,}82\ A$$

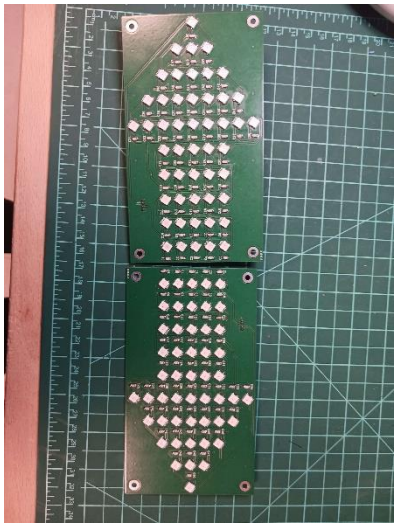
Nous avons un courant de 1,82 A sur 5V. Au vu du courant et de la tension utilisés, nous pourrions remplacer l'alimentation externe de 12 Volts et un Traco par un port USB qui peut délivrer jusqu'à 5 V sous 3 A maximum, ce qui est suffisant.

6. RESUME

Le projet "Flèche de Possession au Basket" a été réalisé par des étudiants en troisième année de Génie Électrique et Informatique Industrielle (GEII) à l'IUT d'Angers. L'objectif majeur de ce projet consiste à moderniser une flèche utilisée pour indiquer la possession de balle lors d'un match de basket. L'enjeu principal était de concevoir des flèches équipées de LEDs et de rendre le système électronique, permettant ainsi un contrôle direct de l'utilisateur.

Pour atteindre cet objectif, nous avons élaboré deux cartes électroniques intégrant des LEDs, auxquelles s'ajoute une troisième carte driver responsable de la gestion des deux premières. L'utilisateur dispose de quatre boutons devant lui pour interagir avec le système. Deux boutons servent à allumer ou éteindre les flèches, un troisième active un signal visuel rappelant de changer la direction des flèches, et enfin, un dernier bouton permet d'ajuster la luminosité en le tournant.

L'objectif final de ce projet est son installation dans une salle de basket, offrant ainsi une application concrète et fonctionnelle.



Carte LEDs

The "Basketball Possession Arrow" project was carried out by third-year students in Electrical Engineering and Industrial Computing (GEII) at the IUT of Angers. The primary goal of this project is to modernize an arrow used to indicate ball possession during a basketball game. The main challenge was to design arrows equipped with LEDs and to make the system electronic, thus allowing direct user control.

To achieve this objective, we developed two electronic cards incorporating LEDs, with an additional third driver card responsible for managing the first two. The user has four buttons in front of them to interact with the system. Two buttons are used to turn the arrows on or off, a third activates a visual signal reminding to change the direction of the arrows, and finally, a last button adjusts the brightness by turning it.

The ultimate goal of this project is its installation in a basketball court, providing a concrete and functional application.

Projet final

7. BIBLIOGRAPHIE

7.1. SOURCES RECHERCHES ALIMENTATIONS

- https://fr.rs-online.com/web/p/regulateurs-a-decoupage/6664379
- https://www.tracopower.com/fr/fra
- https://www.sonelec-musique.com/electronique_bases_transistor.html

7.2. SOURCES POUR RECHERCHE MOYEN COMMUTATION LEDS

- https://www.mouser.fr/ProductDetail/Omron-Electronics/G6E-134P-US-DC5?qs=JK6Bpmia%2FmsLYo4XagKASw%3D%3D&gad_source=1&gclid=Cj0KCQiAmNeqBhD4ARIsADsYfTccCEEy1oSZLnUKDBA6GSv-Asrntz0Yy89Jw9LqCNOwzf1SZDvE9ysaAriwEALw_wcB
- https://fr.rs-online.com/web/p/relais-de-puissance/6803593

7.3. SOURCES RECHERCHES LEDS

- https://fr.rs-online.com/web/c/afficheurs-et-optoelectronique/led-diodes-electroluminescentes/leds/?applied-dimensions=4293014978
- https://fr.aliexpress.com/item/1558427079.html
- https://www.youtube.com/watch?v=DWdNKsDpvqs&ab_channel=RenaudBauer-Createur
- https://fr.aliexpress.com/item/32452753190.html?spm=a2g0w.search0104.3.96.30055b6dJvLBFt&ws_ab_test=searchweb0_0,searchweb201602_3_10882_10065_10068_204_5727215_10843_318_10059_10884_5727315_10887_10696_100031_10084_10083_10103_452_10618_10304_10307_10820_532_10821_10302,searchweb201603_60,ppcSwitch_0&algo_expid=ba981cb9-679d-4078-9c84-4ca535b3ae33-14&algo_pvid=ba981cb9-679d-4078-9c84-4ca535b3ae33&priceBeautifyAB=0
- https://basket-market.fr/fr/270-fleche-de-possession-alternee-electronique.html
- https://www.lextronic.fr/matrice-a-leds-rouges-32-x-16-40440.html
- https://www.ledcontrollercard.com/french/p10-exterieur-smd3535-1-2-duty-led-module-d-ecran-32x16-dot.html

7.4. PROGRAMMATION

- https://www.arduinolearning.com/code/ws2812-8x8-64-led-matrix-arduino-examples.php
- https://github.com/FastLED/FastLED
- https://fastled.io/
- https://docs.arduino.cc/hardware/uno-rev3
- https://docs.arduino.cc/hardware/nano