DUPLOUY Corentin
ZEGGAÏ Mehdi

Projet :
Dé Électronique Communiquant

IMA4 2014/2015

Tuteurs projet : M. BOÉ Alexandre, M. VANTROYS Thomas

Table des matières

Introduction 3

1. Cahier des charges 4

1.1. Objectifs : 4

1.2. Solutions matérielles et logicielles 4

1.3. Étapes du projet 5

2. Électronique 6

2.1. Multiplexage des LED 6

2.2. Test des accéléromètres 9

2.3. Partie Sonore 9

2.4. Partie Tactile 9

2.5. Alimentation du cube 10

2.6. Faces du cube 10

2.7. Autres cartes électroniques 13

3. Code Informatique : 14

3.1. Périphériques du microcontrôleur : 14

3.2. Branchement et codage du microcontrôleur 17

4. Problèmes rencontrés 21

Conclusion 22

Introduction

Dans le cadre de notre 4ème année, en Informatique Microélectronique Automatique, nous avions eu un projet à réaliser durant notre semestre 8.

Nous avons porté notre choix sur le projet numéro 24 : Dé Électronique Communiquant. En effet, étant tous les deux dans des filières différentes (Corentin en Systèmes Autonomes, et Mehdi en Systèmes Communicants), nous avons choisi un sujet avec des disciplines permettant d'équilibrer l'apport de nos deux filières : l'électronique, la gestion de l'énergie et la programmation informatique.

Dans un premier temps, nous allons détailler le cahier des charges, pour ensuite expliquer le travail effectué du côté électronique, et du côté programmation, pour enfin relater les problèmes rencontrés.

N.B. : Les codes non-présents dans ce rapport sont trouvables en annexe sur le wiki du projet.

1. Cahier des charges

1.1. Objectifs :

Nous avions pour mission de créer un dé électronique, à la manière de ce qui se fait avec le Futurocube :

Figure 1 : Futurocube

Ce cube pourra avoir les fonctionnalités suivantes, listées par difficulté de conception croissante :

- Génération aléatoire d'un entier entre 1 et 6,
- Éclairage d'ambiance / Animations,
- Jeu du Tic-Tac-Toe (ou morpion),
- Jeu de mémorisation lumière/son (à la manière du jeu Simon)
- Rubik's Cube
- Snake

Les 3 premières étant beaucoup plus simples à programmer que les 3 suivantes, nous réaliserons en priorité ces 3 fonctionnalités.

1.2. Solutions matérielles et logicielles

Nous utiliserons un microcontrôleur de type mbed LPC1768, équipé d'un microprocesseur ARM Cortex M3. Les programmes seront ainsi codés sur la plateforme de développement en ligne mbed, en langage C/C++.

Chaque face de ce cube sera équipée de 9 LEDs RGB en CMS, soit 54 au total. Elles seront montées sur circuit imprimé. Le microcontrôleur est équipé de 26 pins I/O, soit un nombre insuffisant pour allumer chacune de ces LEDs. Des transistors bipolaires et des résistances seront ainsi utilisés pour établir le multiplexage de ces LEDs.

Nous utiliserons un accéléromètre pour détecter les mouvements du cube, notamment pour les fonctions de lancer de dé, voire du Rubik's Cube et du Snake, et nous intégrerons à ce cube une solution tactile.

1.3. Étapes du projet

Ce projet s'organisera sur 4 grandes étapes :

- La recherche de solutions pour l'accéléromètre, le tactile et le multiplexage

- Le test de ces solutions séparément

- La conception des circuits imprimés nécessaires

- La création des programmes nécessaires au fonctionnement du cube

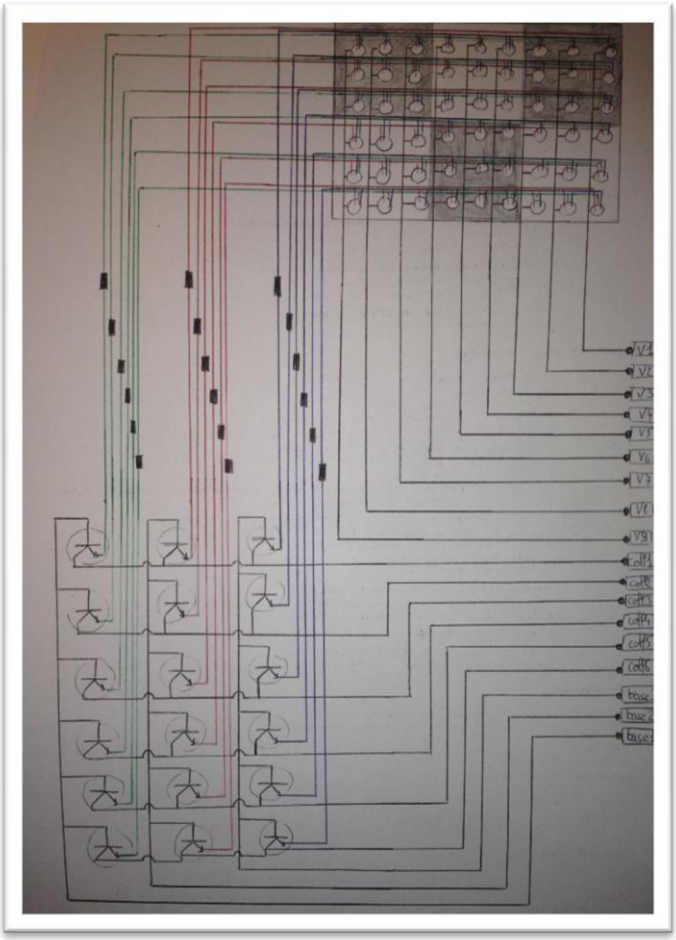
2. Électronique

La partie électronique comprend la création des circuits imprimés ainsi que leur exploitation pour ainsi réaliser un prototype et tester nos différents codes et fonctions.

2.1. Multiplexage des LED

La première chose qui a été réalisée est l'étude de la problématique majeure de notre projet : Comment alimenter et gérer 54 LED RGB, soit un total de 162 LED, en prenant en compte la limite de pins du MBED et sans consommer trop d'énergie ?

Après plusieurs recherches, nous avons décidé d'utiliser un certain type de multiplexage. En effet, nous avons réalisé un quadrillage de 3x6, soit 18 transistors. Cela nous permet, à travers l'activation de la base et la mise à la masse par le collecteur, de contrôler une par une chaque LED. Voici un petit schéma montrant la mise en place d'un tel procédé.

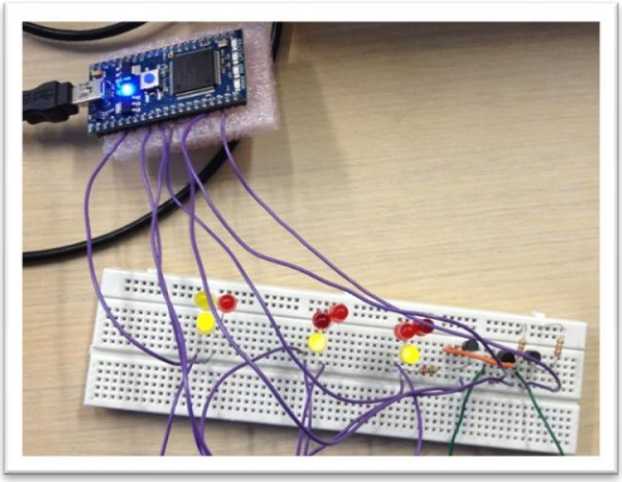


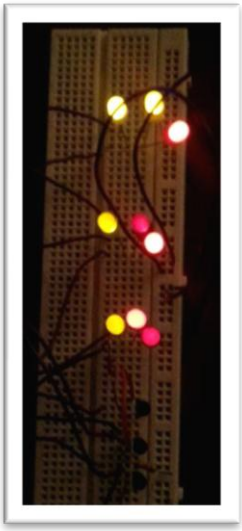
Nous pouvions ainsi contrôler 164 LED avec seulement 18 pins du MBED.

Voici un exemple de fonctionnement : Si nous voulons allumer la LED tout en haut à gauche sur le schéma en vers, il suffit de mettre à 0 les sorties V1 à V8, les bases 2 et 3, et le coll1. On aura donc les autres collecteurs, V9 et base1 à 1.

Ainsi, on alimente toute la ligne de LED grâce à V9 mais seule la LED voulue s'allumera puisque c'est la seule qui est relié à la masse (au 0 fourni le collecteur).

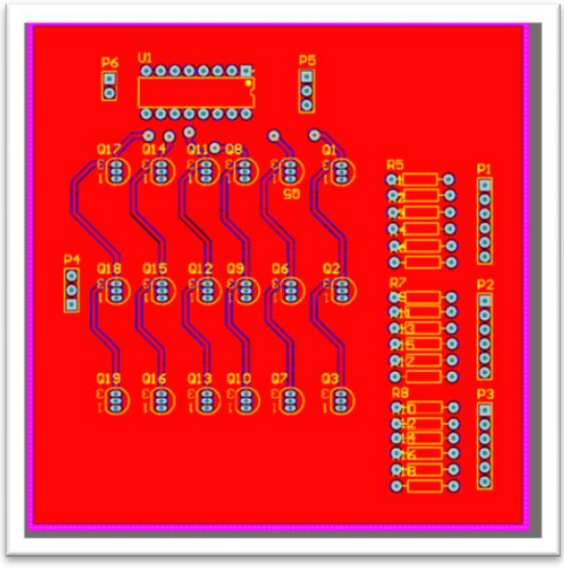
Nous avons réalisé plusieurs tests à petite échelle pour confirmer cette méthode. Voici celui qui montre le test final du multiplexage (les autres sont disponibles sur le wiki).

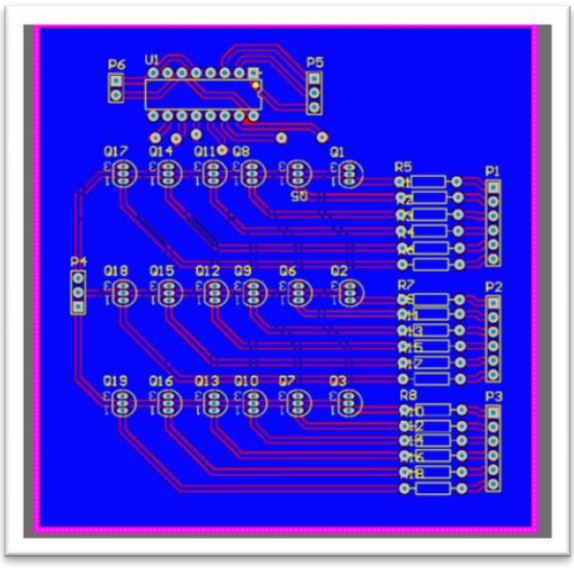


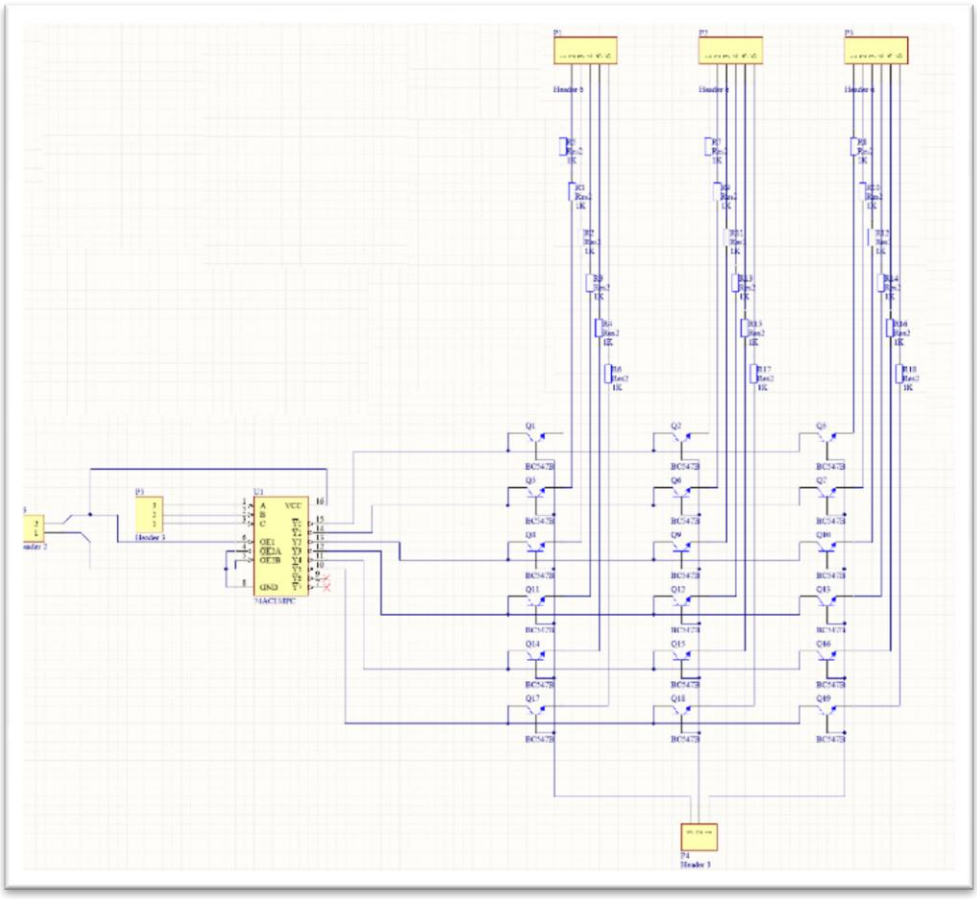


Ici, nous avons 3x3 LED pour représenter 3 LED RGB, 3 résistances pour limiter le courant circulant et 3 transistors NPN branchés selon le schéma plus haut. On peut voir que toutes les LED sont allumées. En réalité, elles ne s'allument qu'une par une mais très rapidement.

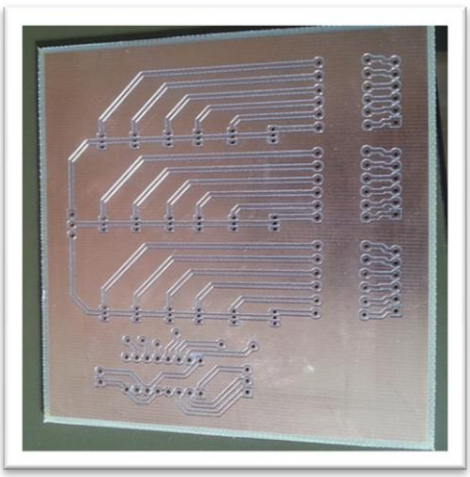
Plus tard, nous avons donc commencé à réaliser ce système de multiplexage sur Altium. Ci-après, nous avons le schematic et le PCB des deux faces. Nous avons dû faire de la double face car certaines pistes été très difficiles à placer. Nous avons donc mis en place des vias. L'amélioration qui a été apporté ici est la présence du décodeur. En effet celui-ci nous permet de piloter les 6 collecteurs avec seulement 3 pins du microcontrôleur, toujours dans l'optique d'en utiliser le moins possible.

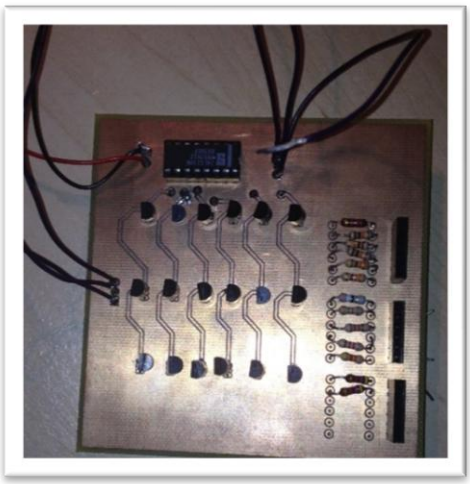






Une fois la carte sortie, les composants ont été soudés.





Cette carte de multiplexage devait être miniaturisée en CMS, mais seul le schematic est réalisé à ce jour. Cependant, nous avions prévu les transistors en CMS ainsi que le décodeur.

2.2. Test des accéléromètres

Pour les détections de mouvement et réveil du cube, nous avons choisi d'utiliser un accéléromètre. Nos premiers tests se sont tout d'abord portés sur l'ADXL335.

Les valeurs des axes X, Y, Z étaient récupérées par les voies analogiques du MBED. Après plusieurs tests (vidéo disponible sur le wiki), et discussions, nous avons décidé de passer à l'accéléromètre ADXL345, qui lui permet de détecter les mouvements soudains. Cela permettait donc d'influer sur la consommation énergétique du MBED. En effet, quand aucun mouvement n'est détecté, le cube se met en veille et s'il y a du mouvement, alors le cube se réveille.

Les tests ont été réalisés en externe et nous n'avons pas encore intégré l'ADXL345 sur un circuit intégré au cube.

2.3. Partie Sonore

Pour la partie sonore, nous avons décidé d'utiliser un simple piezo. Nous avons réalisé plusieurs tests séparément, pour entendre les différentes sonorités, ce qui s'est avéré largement suffisant pour l'utilisation que nous voulions en faire :

- Entendre un « bip » lorsqu'une face est touchée
- Jeu Simon qui produit une suite de différents sons à retenir et à rejouer.

2.4. Partie Tactile

C'est en effet cette partie qui nous a causé le plus de problème et sur laquelle nous avons perdu tant de temps.

Nous étions partis à la base sur des résistances capacitives, ce qui était trop cher. Nous avons ensuite cherché du côté des DIY. Nous avons trouvé plusieurs TUTO, mais pour arduino. Comme nous l'avons explicité sur le wiki, tous nos essais ont été perte de temps et ne nous ont rien apporté au final. Nous nous sommes intéressés également au Sparkfun Touch Shield. Celui-ci était à la base destiné aux Arduino. Nous l'avons testé sous Arduino et cela marchait très bien. Cependant, après discussions, il aurait été trop contraignant à adapté au MBED (6 shields en tout) et n'étant pas transparent, il ne laissait pas la visibilité sur nos LED.

La solution retenue a été apportée par M. Vantroys avec le Capacitive Touch Sensor d'Adafruit. Son fonctionnement a été testé et approuvé séparément. Aujourd'hui il est en cours d'intégration dans le cube. Notre idée (non testée à ce jour) est de le relier directement à chaque face en cuivre du cube (voir image d'une face ci-après). De ce fait, la face entière en cuivre serait la touche capacitive.

Nous espérons fortement pouvoir montrer cette fonctionnalité dans la vidéo que nous allons réaliser.

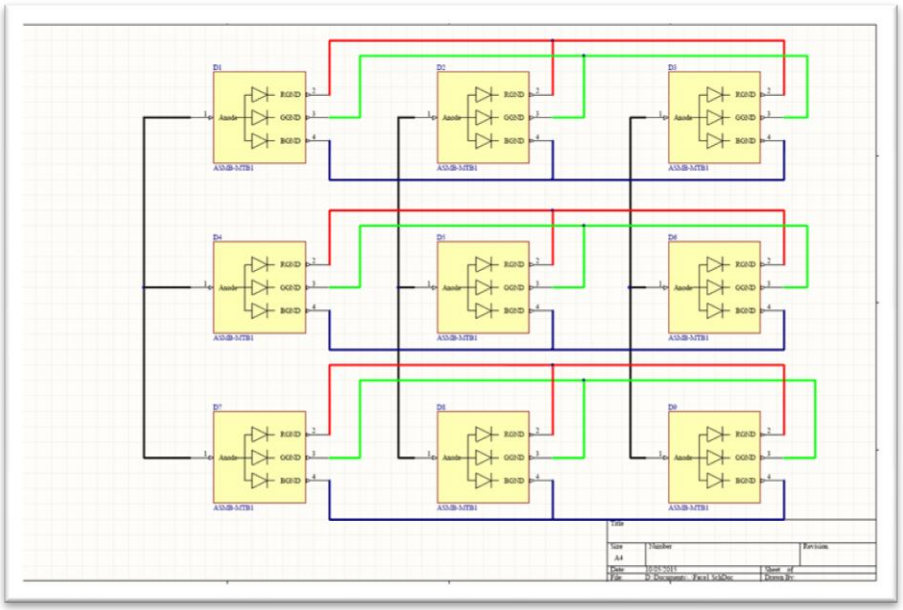
2.5. Alimentation du cube

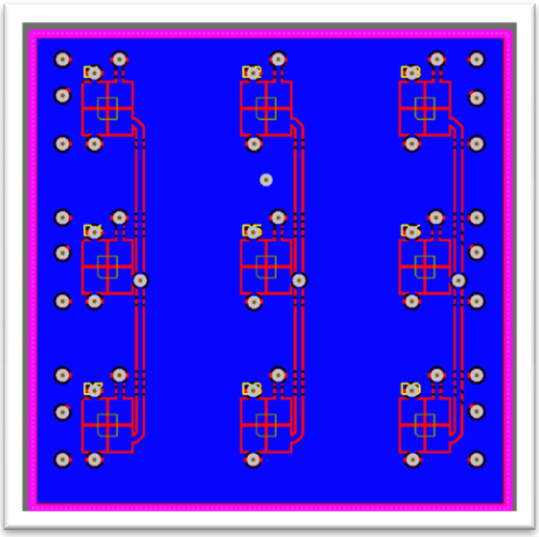
L'alimentation prévue était à la base une batterie, mais cette idée a évoluée en l'utilisation de pile CR2032. Malheureusement, nous n'avons pas eu le temps de réaliser le PCB de la pile pour pouvoir l'intégrer.

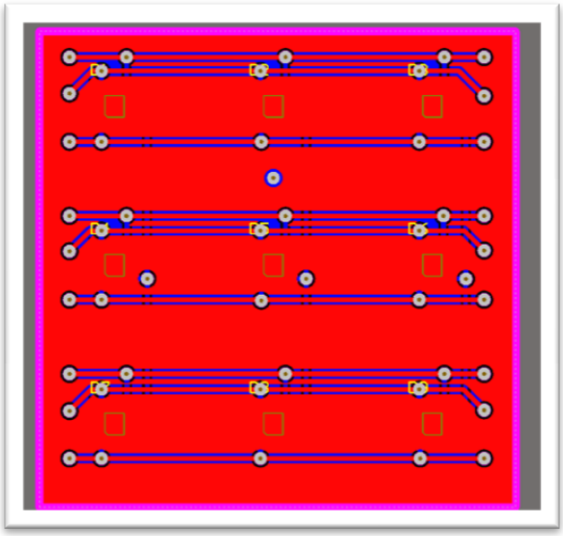
2.6. Faces du cube

Nous avons réalisé les faces du cube à l'aide d'Altium. Les LED CMS n'étant pas encore créées, nous avons élaboré le schematic et les empreintes de celles-ci dans une nouvelle librairie selon un tutoriel de M. Boé.

Voici ensuite les schematic et PCB d'une face :



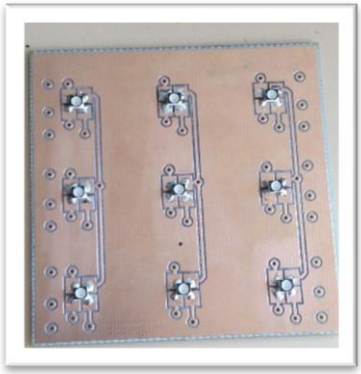


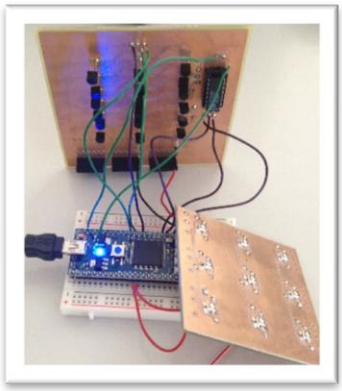


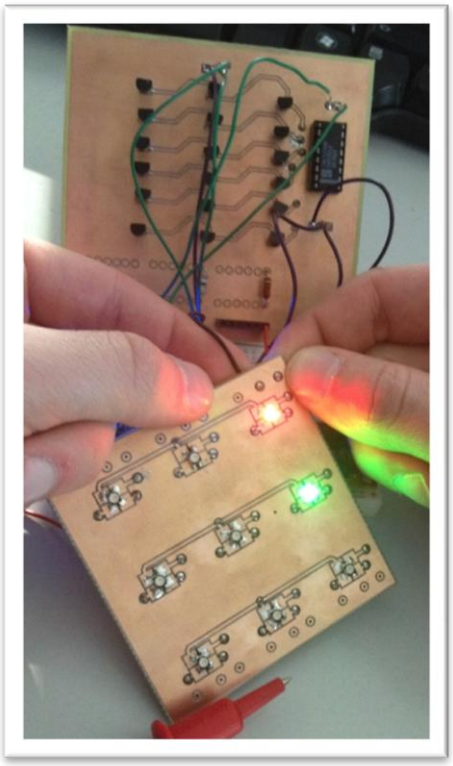
Sur le bottom, on peut voir les 9 empreintes de LED, 3 pistes pour relier les anodes par colonne et les 27 vias qui permettent d'interconnecter les masses de la LED rouge, verte et bleue par ligne sur la couche top. Ces masses sont amenées sur des pads aux deux extrémités de la face pour permettre la connexion avec d'autres faces.

On peut également remarquer la présence d'un pad au milieu qui est relié au plan de masse. Il s'agit de la méthode adoptée pour le tactile en transformant le plan de masse ne une « grosse touche » tactile. Cette solution n'a pas encore été testée.

Une fois imprimée, les LED ont été soudées et nous avons réalisé une première série de tests avant de sortir 5 faces supplémentaires :

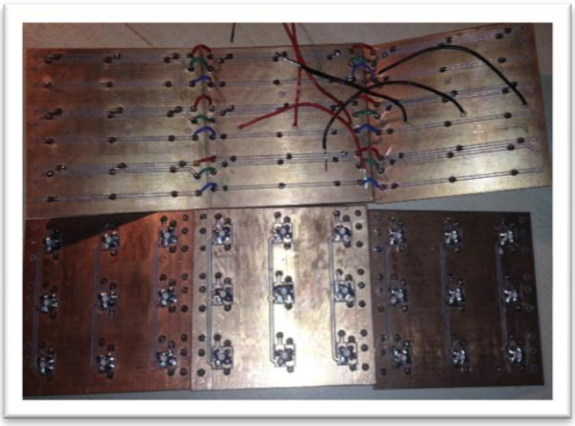






Les tests étant concluants, les faces restantes ont été gravées et ont suivi la même procédure de soudure.

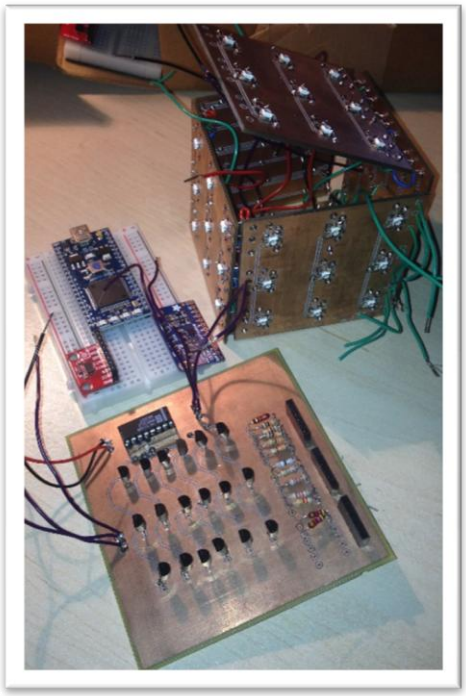
Pour assembler les faces, nous avons utilisé des liaisons filaires, reliant les masses des 3 premières faces d'une part et des 3 restantes de l'autre :



Le prototype final a été obtenu en reliant chaque anode de la première série de trois faces à un autre de la deuxième série de trois faces.

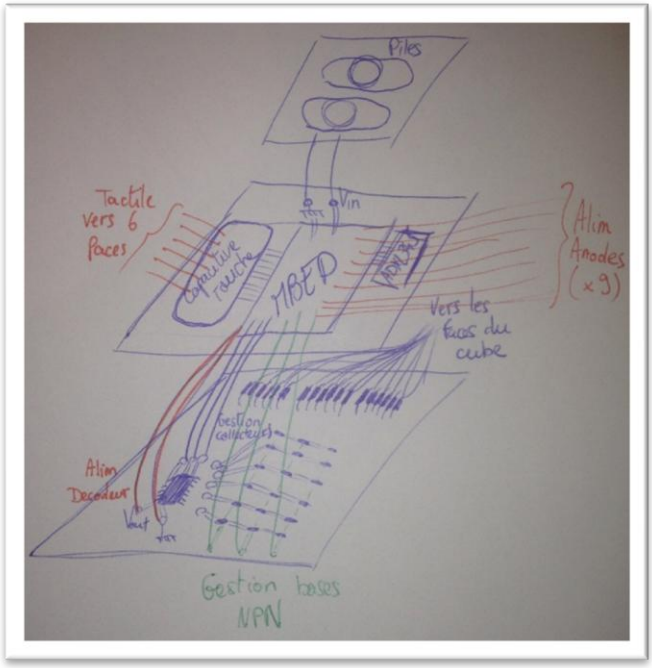
Les autres circuits n'ayant pas encore été réalisé, nous avons décidé de sortir les câbles pour pouvoir les relier au MBED et à la carte de multiplexage à l'extérieur.

Voici le prototype obtenu :



2.7. Autres cartes électroniques

Pour former un cube avec l'électronique totalement embarquée, nous avions prévu de concevoir 3 cartes supplémentaires. Mais nous n'avons pas pris le temps de les concevoir sous Altium. Cependant, voici ce que nous voulions faire :



Une carte pour les piles permettant d'alimenter le MBED.

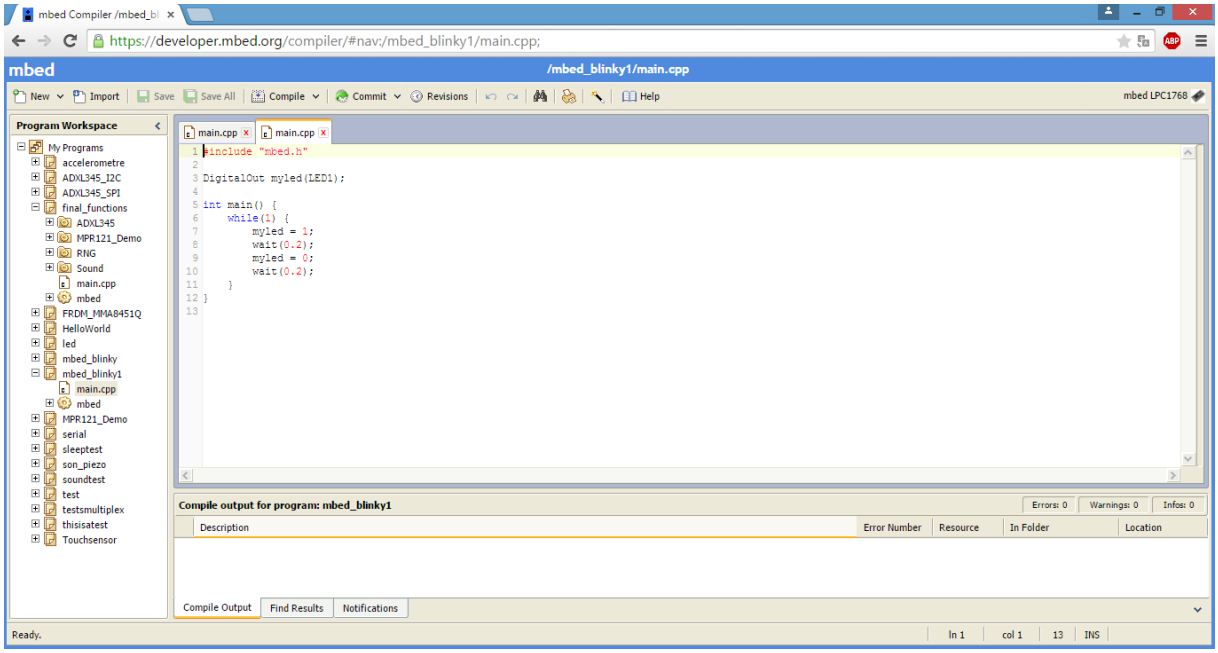
Une seconde comportant le MBED, le capacitive Touch Adafruit et l'accéléromètre.

Une troisième qui correspond à la carte de multiplexage mais en miniature avec des composants CMS.

La liaison entre chaque se faisant par des fils.

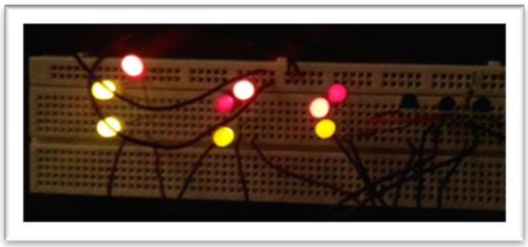
3. Code Informatique :

Nous avons d'abord commencé par découvrir l'interface de programmation en ligne proposée par MBED. Ce kit de développement utilise le langage C/C++.



mbed Software Development Kit (SDK)

L'une des premières fonctions que nous avons développée consistait à tester notre système de multiplexage des LEDs. Nous avons testé cela sur 9 LEDs, (qui représentaient 3 LEDs RGB), ainsi, avec le programme en Annexe , et en agissant sur les différents transistors de notre système de multiplexage, nous avons pu allumer nos 9 LEDs une par une de telle sorte que pour l'œil humain, elles paraissent toutes allumées en même temps :



3.1. Périphériques du microcontrôleur :

Par la suite, nous avons commencé à paramétrer l'utilisation d'un accéléromètre. Nous avons tout d'abord commencé par un accéléromètre analogique ADXL335. Par la suite, pour une gestion plus aisée des valeurs, et pour une éventuelle gestion de l'énergie, il nous a été suggéré d'utiliser un accéléromètre numérique ADXL345. Nous avions deux choix possibles pour la gestion de cet accéléromètre : utiliser les ports I²C, ou bien utiliser les ports SPI. Nous avons choisi ce dernier protocole pour la gestion de l'accéléromètre, et nous l'avons

alimenté en 3.3V. Une bibliothèque disponible sur le SDK nous a permis de paramétrer l'accéléromètre et d'acquérir plus facilement les valeurs des axes X, Y et Z.

La suite de notre projet consistait à trouver une solution tactile à notre cube. Nous étions tout d'abord partis sur des touches capacitives "Do It Yourself", mais nous avons constaté que peu de solutions étaient disponibles sur mbed. Nous avons pu trouver une solution pour Arduino, nous avons essayé de l'adapter pour notre mbed, mais cet essai s'est avéré infructueux, en raison de la difficulté à manipuler les ports du microprocesseur ARM.

Par la suite, M. Vantroys nous a fourni un pad tactile de cette forme, lui aussi physiquement conçu pour être fixé sur Arduino Uno :

SparkFun Touch Shield

Il est équipé du contrôleur de capteur de proximité MPR121. Nous avons pu trouver une bibliothèque spécialement adaptée pour ce contrôleur : MPR121.h. Après avoir étudié les branchements, nous avons essayé de l'adapter pour notre mbed. Les tests ont été moyennement concluants, en raison de la non-solidité des branchements, mais la forme de pad est mal adaptée pour notre cube, puisque nous voulions mettre en place une touche tactile par face. Par la suite, il nous a été proposé une autre solution, plus adapté à notre système :

Adafruit 12-Key Capacitive Touch Sensor Breakout Board

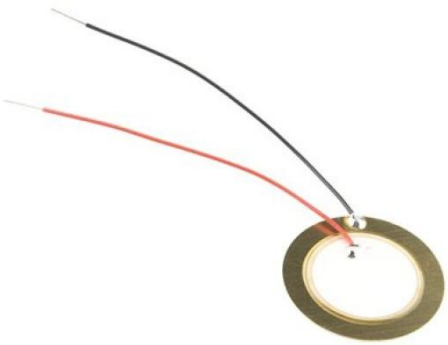
Cette board permet d'utiliser jusque 12 touches capacitives connectées via un fil. Elle utilise elle aussi le contrôleur MPR121, via un branchement I²C; et un régulateur d'alimentation de

3V. On peut ainsi l'alimenter avec 3.3V ou 5V. Avec la bibliothèque MPR121.h du SDK, nous avons pu la tester avec succès, notamment avec la fonction d'interruption suivante :

```
void fallInterrupt() {
    int key_code=0;
    int i=0;
    int value=mpr121.read(0x00);
    value +=mpr121.read(0x01)<<8;
    // LED demo mod by J. Hamblen
    //pc.printf("MPR value: %x \r\n", value);
    i=0;
    // puts key number out to LEDs for demo
    for (i=0; i<12; i++) {
        if (((value>>i)&0x01)==1) key_code=i+1;
        }
    led4=key_code & 0x01;
    led3=(key_code>>1) & 0x01;
    led2=(key_code>>2) & 0x01;
    led1=(key_code>>3) & 0x01;
}
```

Pour chaque touche, les 4 LEDs du LPC1768 s'allument et affichent le numéro de la touche active de façon binaire (de 0 à 12 → de $(0000)_b$ à $(1100)_b$).

Enfin, nous avons voulu qu'un son soit émis dans certaines circonstances d'utilisation du dé. Nous utilisons pour cela une pièce piézoélectrique de ce type :



Élément piézoélectrique

Nous avons là aussi pu trouver une bibliothèque "Sound.h". Cette bibliothèque permet l'émission d'une tonalité ou d'une mélodie sur une certaine gamme de notes, et chaque note est émise sur une durée désirée.

3.2. Branchement et codage du microcontrôleur

Après avoir étudié les périphériques utiles à notre cube, nous avons étudié l'allumage des LEDs du cube en lui-même. Nous avons décidé de mettre en place 3 fonctions principales pour notre cube : la génération d'un chiffre aléatoire entre 1 et 6, l'animation (éclairage du cube), ainsi que le jeu du morpion.

Pour commander l'allumage de chaque LED, nous avons tout d'abord décidé de brancher tous les éléments aux I/O du microcontrôleur selon le schéma suivant :

accéléromètre	**(SPI : mosi) p5**			
accéléromètre	**(SPI : miso) p6**			
accéléromètre	**(SPI : sck) p7**			
accél (chip select)	**p8**			
Tactile	**(I²C : sda) p9**			
Tactile	**(I²C : scl) p10**			
	p11		**p30**	base1
anode1	**p12**	**ARM CORTEX M3**	**p29**	base2
anode2	**p13**		**p28**	base3
anode3	**p14**		**p27**	
anode4	**p15**		**p26**	tactile (interrupt)
anode5	**p16**		**p25**	son (piézo)
anode6	**p17**		**p24**	collecteur1
anode7	**p18**		**p23**	collecteur2
anode8	**p19**		**p22**	collecteur3
anode9	**p20**		**p21 (Pwmout)**	son (piézo)

Sachant que notre système de multiplexage est constitué de 6x3 transistors, les branchements s'effectueront comme suit :

- Les pins p28, p29 et p30 seront reliées à chacune des bases des transistors selon 3 colonnes, qui caractériseront chacune des couleurs rouge, vert et bleu.
- Les pins p22, p23 et p24 seront reliées à un décodeur 3 vers 8 de type 74LS138. Ce décodeur active 7 sorties à l'état haut et une sortie à l'état bas en fonction de l'état des entrées. Nous utiliserons 6 de ces 8 sorties, qui seront reliées à chacun des collecteurs selon 6 lignes. La sortie à l'état bas permettra l'allumage d'une ligne de LEDs selon le schéma de multiplexage détaillé en partie électronique.

Aussi, les pins p12 à p20 caractériseront les anodes des LEDs dont le dispositif est détaillé en partie électronique.

Pour notre programme, nous modéliserons nos entrées/sorties de la manière suivante :

```
// Setup LEDs
DigitalOut anode[9]={p12,p13,p14,p15,p16,p17,p18,p19,p20};
BusOut base_couleur(p30,p29,p28);
BusOut collecteur(p24,p23,p22);

// Setup Sound and accelerometer
Sound sound (p21 , p25);
ADXL345 accelerometer(p5, p6, p7, p8);


// Setup the Mpr121:
// Create the interrupt receiver object on pin 26
InterruptIn interrupt(p26);

// Setup the i2c bus on pins 9 and 10
I2C i2c(p9, p10);

// constructor(i2c object, i2c address of the mpr121)
Mpr121 mpr121(&i2c, Mpr121::ADD_VSS);
```

Les anodes des LEDs seront représentées sous la forme d'un tableau de DigitalOut (sorties numériques, 0 ou 1), et les pins caractérisant les bases et les collecteurs des transistors seront représentées sous la forme d'un bus de 3 pins (BusOut, valeurs entre 0 et 7), selon leur type (base ou collecteur). Le son, l'accéléromètre et le contrôleur tactile seront représentées par une classe venant de leurs bibliothèques respectives (respectivement Sound, ADXL345 et Mpr121, la classe InterruptIn est utilisée pour le contrôleur tactile).

D'autres variables globales ont été créées :

- touche1, touche2, touche3, touche4, touche5 et touche6 sont des variables de type int qui valent 1 si la face que chacune désigne est touchée, et 0 dans le cas contraire.
- lancer, anim et morp sont des variables booléennes qui permettent de vérifier si la fonction que chacune désigne a été correctement réalisée.

Avant tout, nous avons créé une fonction led_eteinte() qui permet d'éteindre toutes les pins et ainsi s'assurer que les LEDs soient toutes éteintes.

Ensuite, nous avons créé une fonction allume_LED(numero,face,couleur) permettant l'allumage d'une LED choisie selon sa face (entre 1 et 6), son numéro (entre 1 et 9), et sa couleur.

Nous avons aussi paramétré une fonction permettant l'émission d'un son entré en paramètre (sous le format uint8_t).

Par la suite, nous avons créé une fonction réalisant l'affichage d'un chiffre entré en paramètre pendant une durée elle aussi en paramètre (en secondes), à la manière d'un dé. Cette fonction émet aussi une mélodie en fonction du chiffre.

Cette fonction permet ainsi de simuler le lancer d'un dé, avec la génération d'un chiffre aléatoire. Pour ce faire, on utilise une bibliothèque "Random.h". La fonction getByte() de la classe Random nous permet ainsi d'obtenir un nombre aléatoire entre 0 et 255 :

Dans notre programme main(), cette fonction sera lancée grâce à l'agitation de l'accéléromètre.

Nous avons aussi créé une fonction d'animation : toutes les faces s'allument progressivement de 7 couleurs selon la durée voulue.

A noter que la mesure de la durée se fait grâce à la classe Timer, qui est composée de 4 méthodes principales : read() qui renvoie la durée en s, read_ms() qui renvoie la durée en ms, stop() qui arrête le timer, et reset() qui réinitialise le timer.

La plus grosse partie du développement a été la programmation du morpion. Tout d'abord, nous avons créé une fonction qui permet de vérifier si une partie a abouti sur une victoire/défaite ou un match nul. Elle retourne "true" en cas de victoire/défaite :

Elle prend en paramètre un tableau modélisant la grille du morpion. Ce tableau est créé grâce à la fonction morpion() qui permet le déroulement du jeu.

Puis, la fonction fallInterrupt(), permettant le fonctionnement des touches tactiles, a été modifiée de telle sorte que les variables globales touche1, touche2, touche3, touche4, touche5 et touche6 soient modifiées en conséquence.

Maintenant que toutes les fonctions nécessaires ont été élaborées, nous avons alors pu établir notre main(), conçu comme suit :

```
int main(){
    int datax, datay, dataz;
     /** Préparation Accéléromètre **/
    int readings[3] = {0, 0, 0};

    //pc.printf("Starting ADXL345 test...\n");
    //pc.printf("Device ID is: 0x%02x\n", accelerometer.getDevId());

    //Go into standby mode to configure the device.
    accelerometer.setPowerControl(0x00);

    //Full resolution, +/-16g, 4mg/LSB.
    accelerometer.setDataFormatControl(0x0B);

    //3.2kHz data rate.
    accelerometer.setDataRate(ADXL345_3200HZ);

    //Measurement mode.
    accelerometer.setPowerControl(0x08);

    interrupt.fall(&fallInterrupt);
    interrupt.mode(PullUp);
```

```
    accelerometer.getOutput(readings);
    while (1) {
        //13-bit, sign extended values.
        //pc.printf("%i, %i, %i\n", (int16_t)readings[0],
(int16_t)readings[1], (int16_t)readings[2]);
        lancer = false;
        anim = false;
        morp = false;
        //secouer le dé pour lancer
        if (touche1==1){
            while (!lancer){
                wait(0.1);
                datax=readings[0];
                datay=readings[1];
                dataz=readings[2];
                wait(0.01);
                accelerometer.getOutput(readings);
                if ((abs(datax-readings[0]) > 100) || (abs(datay-
readings[1]) > 100) || (abs(dataz-readings[2]) > 100)) lancer_de();
            }
        }
        //animations
        else if (touche2==1){
            while (!anim) animations(60);
        }
        //morpion
        else if (touche3==1){
            while (!morp) morpion();
        }

    }
}
```

Les fonctions printf() servaient à debugger notre programme grâce à l'affichage des valeurs sur le PC.

Après la préparation de l'accéléromètre, la boucle principale du programme est lancée. Dans cette boucle, les 3 fonctions principales sont lancées :

- Lancer du dé : après avoir touché la touche correspondante, le programme lancer_de() est activé lors d'une forte agitation de l'accéléromètre : si la valeur retournée par chacun des axes connait une forte variation (on pose un changement de valeur de 100).
- Animation : après avoir touché la touche correspondante, le programme animations(60) est activé. Ici, les différentes couleurs se succèdent sur les 6 faces pendant 60 secondes.
- Morpion : après avoir touché la touche correspondante, le jeu du morpion est lancé via le programme morpion().

4. Problèmes rencontrés

L'un des plus gros problèmes que nous avons rencontré a été le manque de temps, dû probablement à une mauvaise gestion du temps. Nous n'avons notamment pas pu aborder la gestion d'énergie, nous avions initialement prévu d'alimenter notre cube avec des piles bouton de type CR2032. Notre projet reste donc à l'état de prototype, le microcontrôleur s'alimente alors via le port Mini USB. Aussi, nous aurions voulu produire une carte de multiplexage plus petite, ainsi qu'un circuit imprimé spécial pour notre mbed LPC1768, afin de regrouper tous ces éléments dans le même cube.

De plus nous n'avons pas pu réaliser tous les tests finaux sur ce prototype. En effet, nous avons pu monter celui-ci lors de la dernière semaine du projet seulement.

Nous avons pu programmer les 3 fonctions prioritaires (lancer de dé, animations, morpion), mais nous aurions tout de même voulu coder les trois autres programmes (Rubik's Cube, Snake, Simon), pour produire le cube que nous avions en tête avant de commencer notre projet.

Conclusion

Ce projet nous aura permis d'approfondir nos connaissances dans deux principaux domaines : la conception de circuits électroniques, ainsi que le développement d'une application orientée microcontrôleur. Ainsi, nous avons appris à connaître un nouveau langage, le C++, reprenant le style de programmation du C en y incluant la notion de programmation orientée objet.

D'autre part, ce projet nous auras permis d'exercer la gestion de projet et de se rendre compte de nos erreurs. La principale erreur a été de ne pas se projeter assez vite vers un premier prototype. Au lieu de cela, nous sommes restés un moment sur le test de chaque composant séparément en essayant de prévoir son introduction à l'électronique du cube. Nous avons pris du retard en négligeant un peu la partie électronique et création de carte et nous nous sommes rendus compte que celle-ci n'était pas des moindres et prenant un certain temps.

Annexe 1 : Test du multiplexage sur 9 LEDs simples (ou 3 RGB)

```
#include "mbed.h"

DigitalOut anode1(p5);
DigitalOut anode2(p7);
DigitalOut anode3(p9);
DigitalOut collecteurs(p11);
DigitalOut base1(p13);
DigitalOut base2(p15);
DigitalOut base3(p17);

int main() {
    while(1) {
        //seq 1
        anode1 = 1;
        anode2 = 0;
        anode3 = 0;
        collecteurs = 0;
        base1 = 1;
        base2 = 0;
        base3 = 0;
        //seq 2
        wait(0.0022);
        anode1 = 0;
        anode2 = 1;
        anode3 = 0;
        //seq 3
        wait(0.0022);
        anode1 = 0;
        anode2 = 0;
        anode3 = 1;
        wait(0.0022);
        //seq 4
        anode1 = 1;
        anode2 = 0;
        anode3 = 0;
        collecteurs = 0;
        base1 = 0;
        base2 = 1;
        base3 = 0;
        //seq 5
        wait(0.0022);
        anode1 = 0;
        anode2 = 1;
        anode3 = 0;
        //seq 6
        wait(0.0022);
        anode1 = 0;
        anode2 = 0;
        anode3 = 1;
        wait(0.0022);
        //seq 7
        anode1 = 1;
        anode2 = 0;
        anode3 = 0;
        collecteurs = 0;
        base1 = 0;
        base2 = 0;
        base3 = 1;
        //seq 8
        wait(0.0022);
        anode1 = 0;
        anode2 = 1;
        anode3 = 0;
        //seq 9
        wait(0.0022);
        anode1 = 0;
        anode2 = 0;
        anode3 = 1;
        wait(0.0022);
    }
}
```