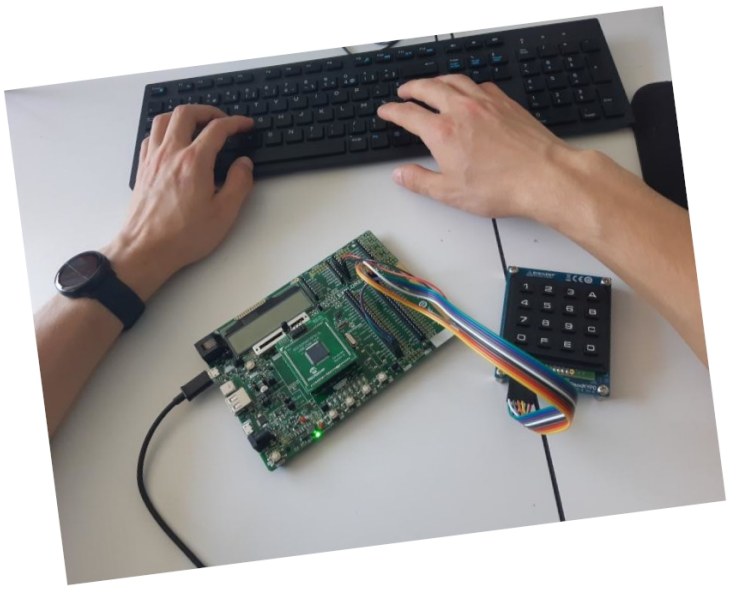
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Projet de Physique P6

STPI/P6/2021 – 42

MICROPROCESSEUR

Projet Clavier



Etudiants :

Pierre BILLAUD — Maxime HUYENH

Ruth LIOTÉ — Aleksi MOUVIER

Adrien PASQUESOONE — Léo ZEDEK

Enseignant-responsable du projet :

Richard GRISEL

Date de remise du rapport : ***12/06/2021***

Référence du projet : ***STPI/P6/2021 – 42***

Intitulé du projet : ***Informatique embarquée : initiation au microprocesseur (clavier)***

Type de projet : ***expérimental, simulation***

Objectifs du projet :

- ***Découvrir les microprocesseurs et leur fonctionnement au travers de la documentation et de simulations réalisées grâce au logiciel MPLAB X IDE .***
- ***Initiation au langage C.***
- ***Réaliser une calculatrice à l'aide d'un clavier et du microprocesseur.***

Mots-clefs du projet: ***microprocesseur, programmation, informatique embarquée, MPLAB***

TABLE DES MATIERES

1. INTRODUCTION

Ce projet, étroitement lié au projet 42, intitulé : “ Informatique embarquée : Initiation au DSP et à la programmation“, a eu pour but de nous faire découvrir l’informatique embarquée par le biais des microprocesseurs. Nous avons donc découvert le fonctionnement des microprocesseurs c'est-à-dire : comment sont stockées les informations dans un microprocesseur, quelles opérations peuvent être effectuées, pour finalement savoir : Que peut-on faire avec un microprocesseur ?

A cause des conditions de travail assez complexe que nous imposait le coronavirus, nous avons pris la décision en début de semestre avec notre professeur référent de faire notre projet sur le PIC32, que l’on soit originellement associé au projet des microprocesseurs ou au projet 42, présentés plus tôt, qui concernait les DSP. Ainsi nous avons pu assister à des cours entiers plutôt que de devoir scinder l'heure en deux, ce qui nous a permis d’approfondir notre travail et de mener à bien ce projet.

Avant de réaliser tout travail, nous devions forcément apprendre comment utiliser un PIC32 et pour cela nous avons lu beaucoup,beaucoup de documentations qui nous ont appris pas à pas les fonctionnalités de ce microprocesseur. Des tests et simulations nous ont ensuite permis de vérifier et d’approfondir notre compréhension du microprocesseur.

Nous allons donc dans ce rapport vous présenter dans un premier temps, les connaissances sur l’informatique embarquée que nous avons obtenues en réalisant ce projet, puis dans un second temps, les résultats obtenus lors de nos simulations, et finalement, le résultat de notre projet.

2. MÉTHODOLOGIE / ORGANISATION DU TRAVAIL

Comme expliqué dans l’introduction, lors des premières séances, les élèves du projet 42 et 41 ont tous travaillé ensemble avec Mr.Grisel afin de découvrir le PIC32. Une fois que nous étions en capacité d’utiliser le PIC32, nous avons été scindé en 2 groupes afin de réaliser chacun un projet.

Au début du semestre, nous nous sommes mis en binôme pour découvrir le PIC32 car nous étions limités par le nombre de microprocesseurs et de postes de travail. Lors de ces séances, des fichiers contenant les programmes nécessaires pour utiliser la carte en elle-même nous ont été fournis. Ceux-ci nous ont permis d’apprendre l’utilisation du PIC32 et d’expérimenter avec celui-ci, notamment avec des options telles que le capteur de température ou le voltmètre. Options qui seront détaillées plus tard dans le rapport.

Ensuite, il a fallu réfléchir à quelles fonctionnalités donner à notre carte; à l’origine nous avions décidé de réaliser un piano à l’aide d’un clavier (ref) et d’une prise jack (ref). Malheureusement par manque de temps et de savoir, nous nous sommes finalement tournés vers la programmation d’une calculatrice de base qui, en plus du clavier ne demandait que la connexion de l’écran LCD au microprocesseur.

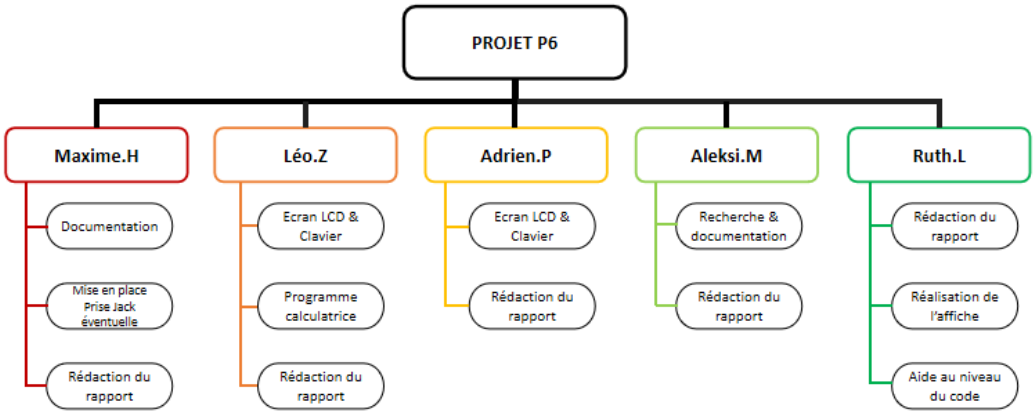
L’organisation du travail au sein du groupe s’est faite naturellement, en effet avec les conditions sanitaires, tous les membres n’étaient pas en présentiel à chaque séance.

Lorsque l'on était en distanciel, nous lisions principalement les documentations et commencions le rapport, le contrôle à distance d’un ordinateur de l’INSA demandant beaucoup de connexion et étant très lent pour suivre convenablement les tests. Et lorsque

l'on était présent nous pouvions procéder aux tests et faire un compte rendu aux élèves en distanciel en fin de séance.

Ainsi, pour ce qui était de la répartition des tâches, lorsque nous étions en train de travailler sur un éventuel piano, ceux en présentiel s'étaient partagés en deux groupes : un s'occupait de la compréhension du clavier et de l'écran LCD tandis que l'autre réfléchissait à comment utiliser la prise jack. Ceux en distanciel, étant eux sur la rédaction du rapport. Néanmoins, suite au changement mentionné plus tôt, l'organisation fut légèrement modifiée. En effet, cette fois-ci un groupe s'occupait de l'implémentation des fonctionnalités de la calculatrice et l'autre de la rédaction du rapport.

Afin d'avoir une vision plus claire et compréhensible de la répartition des tâches, voici un organigramme la représentant :



Organigramme de la distribution des tâches au sein du groupe

3. TRAVAIL RÉALISÉ ET RÉSULTATS

3.1. Travail de documentation

3.1.1. MPLAB

MPLAB est un EDI (Environnement de Développement intégré), créé par la société Microchip qui permet de développer des programmes pour les microprocesseurs PIC et dsPIC. En tant qu'environnement de développement, MPLAB présente une interface graphique avec de nombreuses fonctionnalités comme, bien entendu, créer , éditer compiler et déboguer des programmes en C ou en assembleur, mais aussi diviser/afficher les fichiers sous forme d'arborescence, de suivre les variables en temps réels ou encore l'utilisation d'un mode Simulation, permettant de tester son code dans un environnement virtuel avant de l'envoyer sur une carte, qui pourrait être endommagée.

3.1.2. Le microprocesseur

Créé par l'entreprise Intel en 1971, le microprocesseur a révolutionné le monde de l'électronique. Le rôle d'un processeur dans un ordinateur est d'exécuter les instructions et de traiter les données des programmes. Un microprocesseur est donc un processeur dans lequel tous les composants ont étés soudés à un même circuit imprimé .

Cette miniaturisation a permis :

- d'augmenter les vitesses de fonctionnement des processeurs, grâce à la réduction des distances entre les composants ;
- de réduire les coûts, grâce au remplacement de plusieurs circuits par un seul ;
- d'augmenter la fiabilité : en supprimant les connexions entre les composants du processeur, on supprime l'un des principaux vecteurs de panne ;
- de créer des ordinateurs bien plus petits : les micro-ordinateurs, aussi appelés aujourd'hui, ordinateurs personnels;
- de réduire la consommation énergétique.

Les différences notables entre deux microprocesseurs réside dans :

- Le nombre de bits qu'il peut traiter simultanément. Avec maintenant 64 bits, les processeurs sont capables de traiter rapidement de grands nombres avec une grande précision.
- Le nombre de transistors qu'il possède, appelé complexité d'architecture. En effet, plus le microprocesseur contient de transistors, plus il pourra effectuer des opérations complexes, et/ou traiter des nombres de grande taille.
- Les instructions possibles, pouvant être comptées en dizaines ou en milliers d'instructions différentes réalisables, comme les basiques "additionner deux nombres" ou "comparer deux nombres".
- La vitesse du processeur, c'est-à- dire de son horloge interne. Un cycle de l'horloge permet, en général, de faire une instruction (pour les instructions les plus basiques), ainsi plus l'horloge sera rapide plus le processeur pourra effectuer d'instructions en un temps donné, il sera donc plus performant.

3.1.3. Le clavier

Le clavier est composé de 16 boutons poussoirs formant une matrice 4 * 4.

Figure 1: Photo du clavier

Chacun des ces boutons permet de fermer le circuit électrique dans le clavier et permet ainsi à un signal électrique de passer. Ce signal de sortie sera différent en fonction de la touche poussée et du signal d'entrée. C'est en analysant le signal de sortie que l'on peut déduire la touche enfoncée.

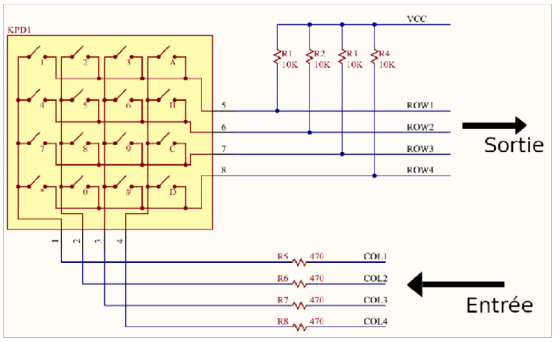


Figure 2: Schéma électrique du clavier

3.2. Initiation à la programmation sur microprocesseur

Pour apprendre à utiliser le microprocesseur nous avons donc commencé par des codes simples et très utiles comme les timers, qui permettent de gérer l'écoulement du temps pendant le programme, si l'on veut par exemple attendre avant de faire clignoter une LED ou lancer une musique, etc.

3.2.1. Allumage de LEDs

Nous avons donc commencé par découvrir comment gérer les LEDs présentes sur notre microprocesseur. Cela nous a appris à utiliser les différents répertoires présents dans le PIC32.

```
int main(void)
{
    int i; // integer local au programme
    // Pour le port A on a les variables d�finies
    // TRISA : Programmation du port, 0xC600 met les Leds en sortie
    // LATA : Latch des sorties
    // PORTA : Idem que LATA pour un port en sortie
    // Instruction pour les ports analogiques Explorer 16
    // ANSELBSET = 0x0230;  // AN9, AN4 et AN5 pour explorer 16 RB4 et RB5 et RB9
    //
    sampleCounter = 0;
    Moyenne = 0;
    truc = 0x12345678;
    i = 0x87654321;
    TRISA = 0xC600; // LEDS en sortie
     while(1)
    {
            LATA = 0x000F;// le programme est une boucle sans fin
            LATA = 0x00F0;

    }

}
```

Figure 3: Code pour allumer/éteindre les LEDS

Nous avons ainsi pu faire s'allumer ou éteindre les LEDs, mais pour aller plus loin nous avions besoin de plus d'outils, nous avons donc commencé à travailler sur les timers.

3.2.2. Nos premiers Timers

Notre but lors de ce cours était d'utiliser ce que nous avions appris sur les LEDs et ce qu'y allait nous être enseigné sur les timers avant de faire clignoter chacune leur tour, ou deux par deux, ou comme nous voulions puisque une fois la démarche acquise il ne nous restait plus qu'à coder.

Un timer nous permet donc d'attendre, un temps voulu, avant que le programme continue, afin d'afficher, dans la majorité des cas, quelque chose ou de faire une animation, ici le clignotement de LEDs. En effet, sans timer pour ralentir le programme, ces indications visuelles seraient bien trop rapides pour que nous puissions vérifier que ce qui est codé fonctionne.

3.2.3. Les Timers

Les interruptions sont une manière bien plus performante de gérer le temps dans son programme. Ce nom vient du fait qu'en effet, une interruption va venir stopper le programme en cours, exécuter son rôle, puis laisser le programme continuer. Ainsi au lieu de faire compter le programme dans le vide en attendant de faire quelque chose, il continuera ce qu'il a à faire et exécutera cette tâche le temps venu.

Une analogie assez simple est celle de la cuisine, l'utilisation d'un décompte revient à rester devant la casserole en comptant chaque seconde pour savoir quand nos pâtes seront prêtes, alors qu'un timer reviendrait à mettre un minuteur et à faire ce que l'on veut jusqu'à la sonnerie du minuteur, on parle ainsi d'attentes passives ou actives.

Pour cela nous disposons donc d'une variable TMR1 qui va s'incrémenter à chaque tour d'horloge. Lorsqu'elle atteindra une valeur définie par PR1, un certain bit, nommé drapeau, s'allumera, donnant le signal au programme qu'il faut réinitialiser TMR1 (pour relancer le timer) et exécuter le code présent dans le timer. La durée entre deux exécution de ce code correspond donc au nombre de tours d'horloge qu'il faut pour que TMR1 atteigne PR1.

```
#define DELAY 65535
void Delay_ms (unsigned t)
{
    TMR1 = 0;
    T1CON = 0x8000; // start timer 1 prescaler 1
    while(t--)
    { TMR1 = 0;
            while (TMR1 < SYSCLK/1000); // Délai 1ms

      }

}
LATA=0x00AA;    // valeur initiale du port A (LATA)
  TMR1=0;
  T1CON = 0x8030;       // prescaler by 256
while (TMR1 < DELAY) // boucle d'attente DELAY*(40Mh/256)
    {
      }
TMR1 = 0;       // remise a zero du registre TIMER TMR1
LATA=0x0055;    // changement de la valeurt du port A  (LATA)
Delay_ms(500);
// si boucle for
LATA=0x0022;    // changement de la valeurt du port A  (LATA)
for (j=0; j <2225187; j++)
{
    }  // duree = 0,5s pour la boucle vide
```

Figure 4: Fonction définissant les timers

3.2.4. Mesure de températures

La carte PIC32 est composée d'un appareil permettant de récupérer la température extérieure. Cependant, la carte renvoie une valeur qui n'est pas la température réelle. Nous avons donc recherché dans la documentation comment convertir cette valeur. Pour nous familiariser avec la carte nous avons donc codé un programme qui récupérait la température en accédant au thermomètre de la carte. Cette fonctionnalité de la carte peut permettre d'effectuer certaines actions une fois une température atteinte par exemple. Cette familiarisation était assez concrète.

3.2.5. Utilisation d'un potentiomètre

Dans la même séance que celle sur la mesure de températures, nous avons appris à manipuler le potentiomètre intégré au PIC32. Ce potentiomètre permet de récupérer une valeur de tension électrique en fonction d'un curseur que l'on peut déplacer. De la même manière, nous avons codé un programme permettant de récupérer cette valeur. Cette fonctionnalité de la carte est très utile pour que les utilisateurs puissent interagir avec la carte.

```
main (void)
{
 LATA=0x0000; //Latch A reset
 TRISA = 0xC600; //Port A output sauf les bits associ�s  d'autres I/O
 ANSELA=0x0600;
 ANSELBSET=0x0030; // voies 4 et 5 en analogique
 ANSELBSET=0x003A; // 0000 0010 0011 1010 voies 4 et 5 et 3 et 1 en analogique test�es
 // AN4 (microchip capteur de temperature TC1047) est multiplexe avec RB4
 // AN5 (potentiometre) est multiplex�e avec RB5 d'ou la mise  1 des bits de selection analogique
 // AN1 est broche analogique du mikroBUS B c'est aussi la broche 50 (P24) des J46 et J47)
 // AN1 est multiplex�e avec RB1
 // AN3 est la broche P22 du connecteur J46/J47 num�ro de pin 48
 // On peut mettre un fil entre POTP20 et les broches des AN1 et AN3 ce qui permet de v�rifier
 // les donn�es de l'ADC

 // ces deux entr�es ont t test�es
adcConfigureManual();
 AD1CON1SET=0x8000;
 while(1)
       {
     valADCvolt = analogRead(5);
     valAN1=analogRead(1);
     valAN3=analogRead(3);
     valADCTemp = analogRead(4);

       }
}
```

Figure 5: Programme principal pour lire la température et la tension modulable

3.3. Notre projet : la réalisation d'une calculatrice

Pour la partie chargée du clavier et de l'écran, plusieurs étapes ont été réalisées :

1- Considérer le clavier comme interrupteurs de LEDs

2- Réussir à afficher des éléments sur l'écran LCD de la carte (PIC 32)

3- Relier des caractères aux touches du clavier pour les afficher sur l'écran

3.3.1. Manipulation d'un clavier

Le clavier est relié au microprocesseur de sorte que les signaux d'entrée et de sortie soient gérés avec le port D.

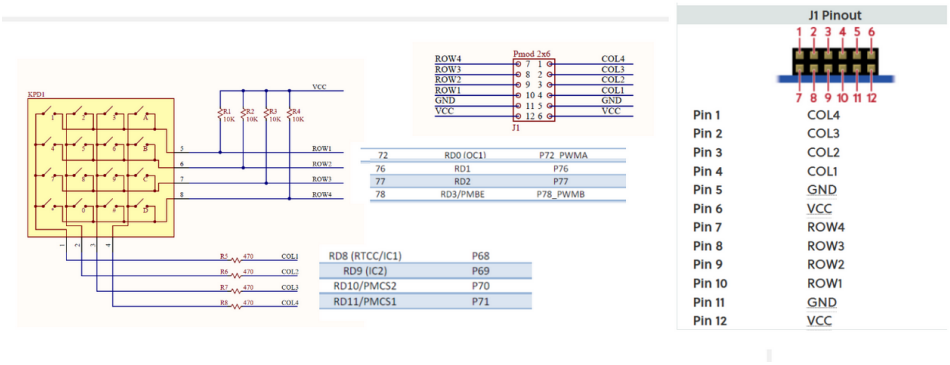


Figure 6: Schéma électrique et broches du microprocesseur utilisées par le clavier

Pour envoyer un signal d'entrée on affecte à **PORTD** une valeur en hexadécimal. Par exemple pour envoyer le signal 0111 en entrée on fixe **PORTD = 0x0700**. Puis on lit le signal d'entrée sur le **PORTD**. Par exemple, si on lit **PORTD = 0x070E**, cela correspond au code d'une touche en particulier.

La détection d'une touche du clavier sera testée constamment. C'est-à-dire qu'il y aura lecture permanente de l'état du clavier. Un signal est donc constamment envoyé en entrée et une lecture de la sortie est constamment effectuée. Lorsque un signal de sortie correspond à une touche (lorsque une égalité est vraie entre le code d'une touche et le signal), on retient en mémoire la touche poussée et lorsqu'on relâche la touche (un peu de programmation pour stopper la suite des instructions tant que la touche n'est pas relâchée), on envoie en sortie la lettre mémorisée.

3.3.2. Configuration de l'écran LCD

Nous avons étudié en cours (lors des séances d'introduction du microprocesseur) une bibliothèque permettant d'utiliser l'écran LCD. Une fonction en particulier ressortait pour afficher une chaîne de caractère sur l'écran LCD. C'est la fonction **puts_lcd** prenant en paramètre la chaîne et sa taille.

Mais avant de pouvoir utiliser cette fonction, il fallait initialiser l'écran avec **Init_lcd**.

3.3.3. Programme de la calculatrice

Nous souhaitions afficher en temps réel sur l'écran LCD le résultat des calculs tapés sur le clavier. Nous devions donc travailler avec des chaînes de caractères pour pouvoir les afficher. Ainsi, le but du programme de la calculatrice était de renvoyer le résultat du calcul sous la forme d'une chaîne de caractères.

Signature de la fonction calcul : **char *calcul(*char chaine)**

Par exemple, la fonction **calcul("123+12")** renvoie la chaîne de caractères "**135**" que l'on pourra ensuite afficher sur l'écran LCD.

Pour ce faire, on utilise tout d'abord une expression régulière pour vérifier si la chaîne de caractères correspond bien à un calcul. **regexp = [[:digit:]]+ [+*/-] [[:digit:]]+** (Ici, nous

avons fait le choix de programmer une calculatrice qui accepte seulement deux facteurs avec un seul opérateur.)

L'utilisation des expressions régulières étant compliquée avec le langage C, nous avons utilisé un algorithme trouvé sur internet que nous avons modifié pour correspondre à nos besoins.

Ensuite, nous avons codé une fonction qui renvoie les deux facteurs et l'opérande contenus dans la chaîne.

Pour faciliter le stockage de ces trois variables, on a construit un type contenant ces 3 variables. La fonction renvoie donc une variable de type **operation**, à voir ci-dessous :

```
struct operation {
  long long facteur1;
  long long facteur2;
  char operande;
};
```

Figure 7: Programme nécessaire pour construire le type operation

Le type de variable utilisé **long long** est un type sur 64 bits dans le but de pouvoir stocker tous les nombres que l'utilisateur peut envoyer.

Une fois que nous avons les deux facteurs (stockés dans un long long int) et que nous connaissons l'opérande, nous utilisons la calculatrice du microprocesseur pour calculer le résultat.

Et enfin, il nous fallait coder une fonction qui convertit un entier en une chaîne de caractères dans le but d'afficher le résultat sur l'écran LCD.

Malgré le bon fonctionnement de la calculatrice individuellement (cf figure en dessous) , lors de l'insertion de celui-ci au niveau de notre programme principal, les ordinateurs d'INSA ne possédaient pas toutes les bibliothèques essentielles au bon fonctionnement de celui-ci. Ainsi, par manque de temps et de connaissances, nous n'avions pas pu ajouter les bibliothèques nécessaires et faire fonctionner notre projet.

```
Veuillez appuyer sur un bouton :5
Vous avez appuyer sur 5
5
Veuillez appuyer sur un bouton :8
Vous avez appuyer sur 8
58
Veuillez appuyer sur un bouton :6
Vous avez appuyer sur 6
586
Veuillez appuyer sur un bouton :-
Vous avez appuyer sur -
586-
Veuillez appuyer sur un bouton :5
Vous avez appuyer sur 5
586-5
Veuillez appuyer sur un bouton :6
Vous avez appuyer sur 6
586-56
Veuillez appuyer sur un bouton :E
Vous avez appuyer sur E
586-56
530
[Inferior 1 (process 5731) exited normally]
(gdb)
```

Figure 8: Démonstration du fonctionnement du code

3.3.4. Perspective sur la prise jack

Pour pouvoir produire du son, l'élément PmodI2S, compatible avec le PIC32 fut envisagé. Toutefois, celui-ci fut trop compliqué à intégrer au projet. Néanmoins, des recherches concernant son implémentation ont été faites.

Le PmodI2S est un module audio stéréo qui convertit les données numériques (digital en anglais) en données analogiques, d'où le nom DAC (Digital to Analog Converter).

Celui-ci est composé de 6 broches, que nous allons numéroter de 1 à 6 pour les énumérer plus facilement, la première étant la plus haute.

- La première broche s'appelle MCLK ce qui veut dire "Master Clock", c'est l'horloge interne. Celle-ci aura la plus haute fréquence, elle est utilisée pour synchroniser les autres horloges.
- La deuxième broche s'appelle LRCK pour "Left Right Clock", cette horloge est utilisée pour contrôler ce qui sort des parties droite et gauche, notamment pour les écouteurs par exemple. Cette broche nous permet de définir la vitesse d'échantillonnage. Le format standard est de 48,8 kHz soit 48 800 échantillons par seconde. Lorsque le signal est au niveau bas, le canal gauche est transféré alors que lorsque le signal est au niveau haut, c'est le canal droit.
- La troisième broche s'appelle SCK pour "Serial Clock", c'est l'horloge "série". Elle est utilisée pour transférer les données transmises par LRCK.
- La quatrième s'appelle SDIN pour "Serial Data INput", cette broche permet de transférer des données analogiques pour qu'elles puissent être converties en données numériques et traitées par le processeur.
- La cinquième et la sixième broche sont utilisées pour l'alimentation de la prise jack, la cinquième étant la masse (GND pour "ground") et la sixième est réservée à l'alimentation fournie par le processeur, celui-ci doit être de 3,3 V (VCC) même si cette broche peut être branchée à 5V, la tension nominale est de 3,3V.

La fréquence du microprocesseur est de 48MHz. Toutefois l'horloge interne ne peut pas fonctionner à cette fréquence, il nous faut donc un timer qui divise par 4 cette valeur pour avoir la fréquence de l'horloge interne. L'horloge série doit être divisée en 16 x 2 = 32 parties pour encoder chaque bits, 16 pour la gauche 16 pour la droite, ce qui nous fait une fréquence de 1,5 MHz. En fin SDIN doit avoir avoir la même fréquence que l'horloge interne soit 32 x 1.5 = 48 MHz.

Nous n'avons pas pu implémenter ce module car la présence de ces timers pourrait gêner notre code principal. La fréquence est aussi trop haute et la prise jack ne pourrait émettre que des sons à une fréquence supérieure à 20 kHz ce qui est inaudible pour l'oreille humaine.

3.4. Difficultés rencontrées

3.4.1. Généralités

Lors de la réalisation de ce projet, nous nous sommes heurtés à de nombreuses difficultés. En effet, tout d'abord un manque de temps crucial s'est fait sentir, l'introduction générale aux microprocesseurs ayant duré très longtemps, nous n'avons pas pu compléter notre application personnelle et ainsi réaliser le piano désiré. Une autre perte de temps que nous avons pu rencontrer fut la durée de compilation du programme qui durait 1 minute à chaque fois. Ainsi à chaque cours, nous perdions dans les alentours de 15 à 20 minutes

pour essayer de faire compiler notre code. Un autre obstacle auquel nous avons fait face, fut l'accès à la salle et au matériel. Effectivement, continuer notre projet en dehors des plages horaires de P6 était impossible car nous ne pouvions pas emmener la carte PIC32 chez nous. De plus, travailler ensemble sur notre sujet était particulier car l'organisation dûe au COVID-19 demandait l'alternance distanciel/présentiel de certains membres. Et enfin, un des problèmes les plus importants, fut la complexité du sujet étudié. Malgré le temps que nous avons passé à étudier de la documentation et écouter les explications, il nous était toujours difficile de comprendre tous les aspects des microprocesseurs en si peu de temps.

3.4.2. Un problème en particulier : relier le clavier à l'écran LCD

Lors de la confection de la calculatrice, nous avons rencontré un gros problème. La gestion du clavier et de l'écran LCD paraissait incompatible. En effet, lorsque que l'on implantait ces deux fonctionnalités, l'écran LCD se mettait soudainement à ne plus marcher. Le fait d'initialiser PORTD sur une certaine valeur faisait bugger l'écran. La solution était de mettre la fonction init_lcd après chaque utilisation du clavier. Mais avant d'arriver à trouver cette solution, nous avons perdu beaucoup de temps à chercher à déboguer le programme.

La construction du code pour la calculatrice a aussi été difficile car la manipulation des chaînes de caractères en langage C est très différente de celle en langage Pascal (seul langage que nous avions étudié).

Après avoir retravaillé sur le code, nous avions finalement réussi à mettre en place un semblant de fonctionnement, le clavier répondait et affichait des valeurs à l'écran. Néanmoins, ces valeurs ne concordaient pas du tout à ce que l'utilisateur rentrait.

Les deux principaux problèmes reliés furent :

- La première ligne du clavier qui ne répondait pas et ne renvoyait aucun élément sur l'écran lors de l'exécution,
- Les valeurs du clavier étaient inversées par rapport à la diagonale 1-5-9-D. Par exemple, le 2 devenait un 4 ou encore le 6 devenait un 8 et inversement.

Toutefois, ces deux problèmes n'apparaissaient pas lors du débogage en step by step, seulement lors de l'exécution globale du code.

4. CONCLUSIONS ET PERSPECTIVES

Nous avons beaucoup appris grâce à ce projet, notamment le travail d'équipe. Cette année a été particulièrement difficile pour se retrouver afin de travailler en équipe. En effet, à cause de la crise sanitaire et l'alternance présentiel / distanciel nous avions du mal à nous rencontrer. Nous avons de plus appris le langage de programmation C embarqué. Ce langage a été intéressant à coder, en effet la plupart d'entre nous n'étions pas familier avec ce langage et donc ce fût un challenge de réussir. L'utilisation du microprocesseur a été satisfaisante lorsque cela devenait concret, particulièrement lors de la séance de l'allumage des LEDs. Nous sommes donc assez fiers de notre projet même si celui-ci n'est pas finalisé.

Comme expliqué plus tôt, nous aurions préféré coder un piano qui aurait été un programme assez différent de ce que nous avons pu voir durant nos deux années de cursus ingénieur, programme qui n'a pu se faire.

Pour notre calculatrice, à cause des difficultés rencontrées pendant ce projet nous avons perdu beaucoup de temps nous n'avons de ce fait pas pu implémenté toutes les fonctionnalités que l'on aurait aimé présenter.

En imaginant un clavier plus grand, et beaucoup plus de temps,nous aurions aussi probablement pu ajouter d'autres possibilités comme un calcul d'exponentielle ou de fonctions trigonométriques .

5. BIBLIOGRAPHIE

Consulté le 19/04/2021 :
https://reference.digilentinc.com/reference/pmod/pmodkypd/start

Consulté le 19/04/2021 :
https://reference.digilentinc.com/pmod/pmodi2s/reference-manual

Consulté le 20/04/2021 :
https://tutorial.cytron.io/2017/08/13/i2s-pic32mxmz-introduction/

Consulté le 10/05/2021 :
https://fr.wikipedia.org/wiki/Microprocesseur

Consulté le 11/05/2021 :
http://nalhossri.free.fr/Librairies%20pour%20CODE%20WARRIOR/clavier_matriciel/clavier_matriciel/decodage_clavier_matriciel.htm

Cours de Richard GRISEL à propos du microprocesseur et du langage C.

6. ANNEXES

6.1. Programme de la calculatrice

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <regex.h>

struct operation {
  long long facteur1;
  long long facteur2;
  char operande;
};

long long strToInt(const char *ma_chaine)
{
  long long result = 0;
  int taille = strlen(ma_chaine);

  for(int i = 0; i<taille; i++){
    result = result * 10 + (ma_chaine[i] - '0');
  }

  return result;
}

//ma_chaine doit être au plus grand de taille 16
int estValable(const char *ma_chaine)
//Cette fonction retourne 1 si la chaine correspond à un calcul.
{
  int err;
  regex_t preg;
  const char *regex = "[[:digit:]]+[+*/-][[:digit:]]+"; //Utilisation d'expression régulière.

  err = regcomp (&preg, regex, REG_NOSUB | REG_EXTENDED); //Compilation de l'expression regulière

  if (err == 0) //Si il n'y a pas d'erreur
  {
    int match;

    match = regexec (&preg, ma_chaine, 0, NULL, 0);

    regfree (&preg); //Libération de mémoire
```

```
    if (match == 0)
    {
        return 1;
    }

    else if (match == REG_NOMATCH)
    {
        return 0;
    }

    else
    {
        char *text;
        size_t size;


        size = regerror (err, &preg, NULL, 0);
        text = malloc (sizeof (*text) * size);
        if (text)
        {

            regerror (err, &preg, text, size);
            fprintf (stderr, "%s\n", text);
            free (text);
        }
        else
        {
            fprintf (stderr, "Memoire insuffisante\n");
            exit (EXIT_FAILURE);
        }
    }
  }

  return (EXIT_SUCCESS);
}
```

```c
int estOperande(char charactere)
{
  return (charactere == '+') || (charactere == '-') || (charactere == '*') || (charactere == '/');
}

struct operation chaineToOperation(const char *ma_chaine)
{
  struct operation result;

  int taille = strlen(ma_chaine);
  int indexOperande = 0;

  while ( ! estOperande(ma_chaine[indexOperande]) ){ //Recherche l'index de l'operande
    indexOperande++;
  }

  result.operande = ma_chaine[indexOperande];

  char nombre1[15]; //14 est la taille max des nombres

  for(int j = 0; j < indexOperande; j++ ){ //On remplie la chaine des charactere du premier facteur
    nombre1[j] = ma_chaine[j];
  }
  nombre1[indexOperande] = '\0'; // On termine la chaine.

  result.facteur1 = strToInt(nombre1);

  char nombre2[15];

  for(int j = indexOperande + 1; j < taille; j++){ //On met dans nombre2 la chaine qui suit l'opérande.
    nombre2[j - (indexOperande + 1)] = ma_chaine[j];
  }

  nombre2[taille - (indexOperande+1)] = '\0';

  result.facteur2 = strToInt(nombre2);

  return result;
}
```

```c
long long calcul(struct operation mon_op){
  switch(mon_op.operande) {
    case '+' :
      return mon_op.facteur1 + mon_op.facteur2;
      break;

    case '-' :
      return mon_op.facteur1 - mon_op.facteur2;
      break;

    case '*' :
      return mon_op.facteur1 * mon_op.facteur2;
      break;

    case '/' :
      return mon_op.facteur1 / mon_op.facteur2;
  }
}

void videBuffer(void) { // À retirer.
    int c;
    while ( (c=getchar()) != '\n' && c != EOF );
}
```

```c
void saisieTexte(char chaine[17])
{

  char bouton = ' ';
  int i = 0;

  while (bouton != 'E'){

    printf("Veuillez appuyer sur un bouton :"); //À retirer
    scanf("%c", &bouton); //À remplacer par bouton = toucheClavier(); (celui du microprocesseur)
    videBuffer(); //À retirer.
    printf("Vous avez appuyer sur %c\n", bouton); //À retirer.

    if (bouton == 'R') { //Si on presse la touche pour retour
      if (i > 0) {
        chaine[i] = ' ';
        i--;
      }
    }

    else if (bouton != 'E' && bouton != ' ') { //Si on presse autre chose que le bouton entrée ou un bouton inactif.
      if (i < 16){
        chaine[i] = bouton;
        i++;
      }
    }
    //On ne prend pas en compte tous les charactere car le clavier ne pourra donner que les chiffres et les lettres E et R.

    printf("%s\n", chaine); //À remplacer par puts_lcd.

  }
  chaine[i] = '\0'; //On termine la chaine.


}
```

```c
void saisieTexte(char chaine[17])
{

  char bouton = ' ';
  int i = 0;

  while (bouton != 'E'){

    printf("Veuillez appuyer sur un bouton :"); //À retirer
    scanf("%c", &bouton); //À remplacer par bouton = toucheClavier(); (celui du microprocesseur)
    videBuffer(); //À retirer.
    printf("Vous avez appuyer sur %c\n", bouton); //À retirer.

    if (bouton == 'R') { //Si on presse la touche pour retour
      if (i > 0) {
        chaine[i] = ' ';
        i--;
      }
    }

    else if (bouton != 'E' && bouton != ' ') { //Si on presse autre chose que le bouton entrée ou un bouton inactif.
      if (i < 16){
        chaine[i] = bouton;
        i++;
      }
    }
    //On ne prend pas en compte tous les charactere car le clavier ne pourra donner que les chiffres et les lettres E et R.

    printf("%s\n", chaine); //À remplacer par puts_lcd.

  }
  chaine[i] = '\0'; //On termine la chaine.


}
```

```c
int main (void)
{

  char ma_chaine[17] = "                "; //On le remplie d'espace.
  saisieTexte(ma_chaine);

  if (estValable(ma_chaine) && strlen(ma_chaine) <= 16){

    struct operation mon_operation = chaineToOperation(ma_chaine);

    printf("%lli\n", calcul(mon_operation)); //À remplacer par put_lcd.
  }

  else{
    printf("Mauvaise saisie "); //À remplacer par puts_lcd.
  }


  return 0;
}
```