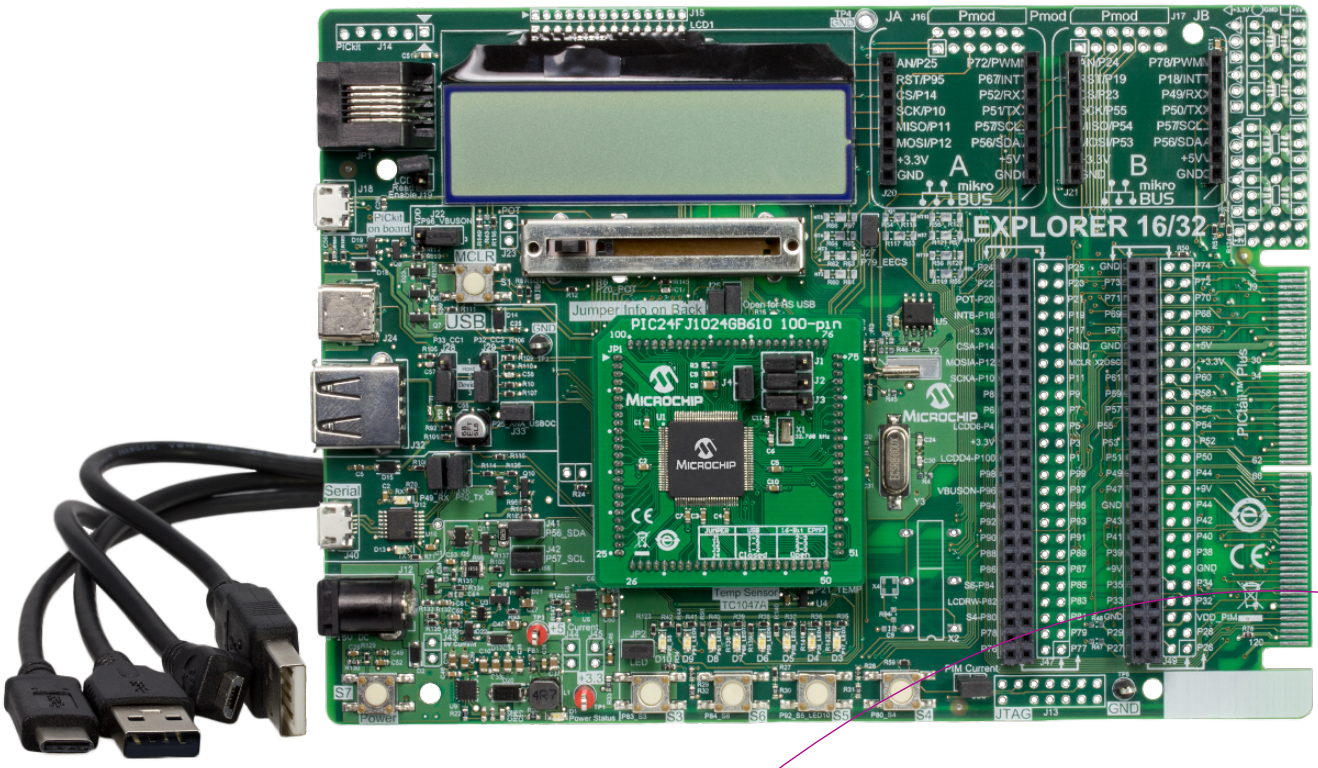
INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
ROUEN NORMANDIE



Ryan JUTEAU
Paul KRAUSKOPF
Adrien QUIBEUF
Elisa RECORDON
Nicolas ROSAL
Tom TESNIERE
Enseignant encadrant
Richard GRISEL

Rapport de projet P6
STPI/P6/2021 - 41

Informatique embarquée : initiation au microprocesseur et à la programmation

P6 – STPI2 – 2020-2021

Date de remise du rapport : 14/06/2021

Référence du projet : STPI/P6/2021 - 41

Intitulé du projet : Informatique embarquée : initiation au microprocesseur et à la programmation

Type de projet : expérimental et simulation

Objectifs du projet :

— Apprendre les rudiments de la programmation embarquée avec une introduction au fonctionnement des microprocesseurs.
— Appliquer ces nouvelles connaissances à la création d'un projet personnalisé, avec une application dans la vie de tous les jours en utilisant la carte de développement et des périphériques fournis.

Mots-clefs du projet : microprocesseur, langage embarqué, initiation, carte de développement

Table des matières

1 Introduction

L'informatique embarquée est très présente dans notre quotidien et ce, dans de nombreux domaines tels que l'automobile, les appareils électroménagers, les téléphones portables et bien d'autres. Ce terme désigne les aspects logiciels que l'on trouve dans des équipements qui n'ont pas pour objectif final d'être purement informatiques.

Ces dernières années, l'informatique embarquée a connu une grande évolution, qui a été rendue possible notamment par la miniaturisation des systèmes. Effectivement, la création des microprocesseurs a révolutionné les systèmes technologiques car ils s'intègrent facilement dans ces derniers grâce à leur petite taille.

Lors de ce projet de physique, nous avons découvert, dans un premier temps, une initiation à cette technologie, réalisée par notre enseignant M. Richard Grisel, puis dans un second temps, la mise en application de nos connaissances théoriques fraîchement apprises. Nous avons eu le choix total de la forme de notre projet, c'est-à-dire des composants utilisés et du besoin auquel répondre.

Concrètement, ce projet nous a permis de comprendre globalement l'utilisation des microprocesseurs, leur fonctionnement ainsi que la programmation informatique avec le logiciel MPLAB.

1.1 Comment fonctionne un microprocesseur ?

En premier lieu, nous allons nous intéresser au fonctionnement global d'un processeur, à savoir les bus de données, l'horloge interne, le pipeline et la pile, qu'il est impératif de comprendre pour la suite de notre projet. Nous découvrirons par la suite le fonctionnement de la carte explorer 16/32 avec les registres, les ports et l'hexadécimal. Enfin, nous étudierons la méthode de fonctionnement d'un IDE, Integrated Development Environment, ici MPLAB, ainsi que le passage de la simulation à la réalité sur ce dernier.

1.1.1 Fonctionnement général d'un processeur

Schéma d'un processeur L'objectif de ce projet étant une introduction au langage embarqué, il est d'une importance primordiale de comprendre le fonctionnement d'un processeur. Souvent considéré comme une boîte noire, celui-ci a pourtant un rôle capital. Il a pour objectif de calculer des instructions de manière linéaire (les unes après les autres), il est au centre de tout appareil embarqué et permet le fonctionnement de la grande majorité d'entre eux. Nous nous intéresserons ici d'abord au schéma conceptuel d'un processeur, puis nous discuterons rapidement de son fonctionnement dans le cadre de la carte PIC32. Un processeur peut être décrit selon le schéma suivant :

Le processeur est composé d'une unité centrale (U.C.), s'occupant de toute la partie logique (calculs), reliée à deux mémoires différentes. Dans un premier temps, la mémoire DATA, volatile, s'occupe de stocker les données nécessaires à l'exécution du programme (variables, etc...) et la mémoire PROG,

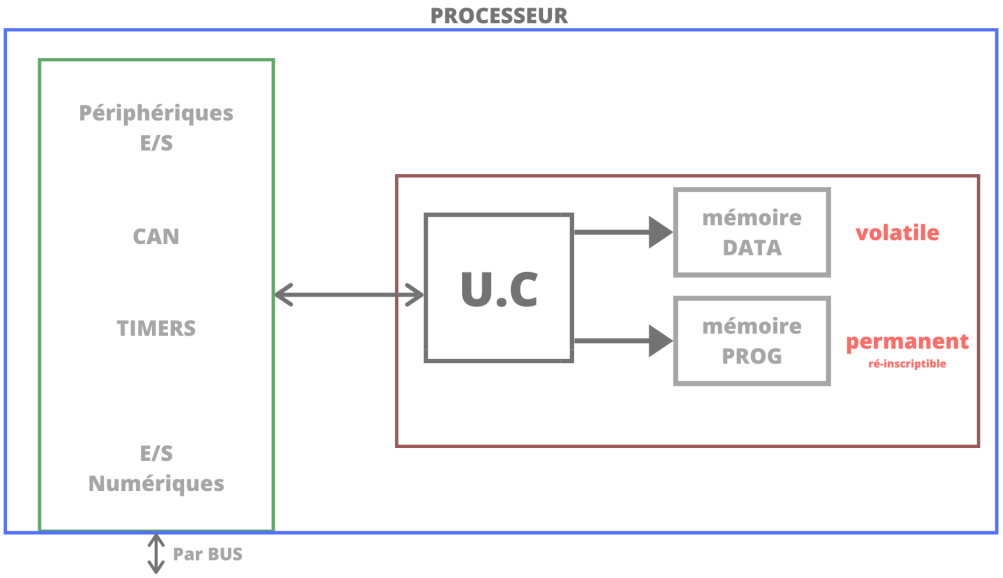


FIGURE 1 – Schéma général d'un processeur

contient le code d'exécution du processeur ; cette dernière est permanente pour pouvoir lancer le processeur de la même manière à chaque fois, mais elle est ré-inscriptible. Cette partie, que l'on pourrait appeler le cœur du processeur, est connectée aux périphériques internes à ce dernier. Nous pourrions parler des "timers" s'occupant, comme leur nom l'indique, d'obtenir une durée précise de temps, ou encore des Entrées/Sorties connectées par BUS aux périphériques de la carte. Nous pouvons aussi noter la présence d'un ou plusieurs Convertisseurs Analogique/Numérique (CAN), s'occupant de convertir un signal discret en signal continu ou inversement.

Fonctionnement interne au processeur Comme écrit précédemment, le rôle du processeur est de "bêtement" prendre et calculer les instructions données les unes après les autres. Celles-ci se trouvent dans la mémoire PROG à une certaine adresse pouvant se trouver dans les registres. À chaque instruction exécutée, le compteur ordinal (Program Counter) s'occupe d'augmenter de 1 l'adresse d'instruction et ainsi passer à la suite du code. Le "reset" permet de remettre le compteur ordinal à 0 (temps 0 du processeur). Même si en informatique et en programmation les langages sont multiples, le processeur, lui, a un langage bien à lui : un langage séquentiel composé d'instructions simples (mov, source, dut, ...). Ces instructions sont exécutées à une vitesse constante, donnée par *l'horloge interne du processeur*, c'est-à-dire combien d'instructions celui-ci est capable d'exécuter à la seconde (en Hz). Dans le cadre de notre projet, le processeur est cadencé à 40MHz. L'exécution de ces instructions se fait de manière particulière, par un schéma nommé le *pipeline*.

Le fait de dire "une instruction = un cycle d'horloge" n'est pas tout à fait exact. En effet, plusieurs instructions annexes à l'instruction principale se greffent à celle-ci, rendant le processus plus lent que

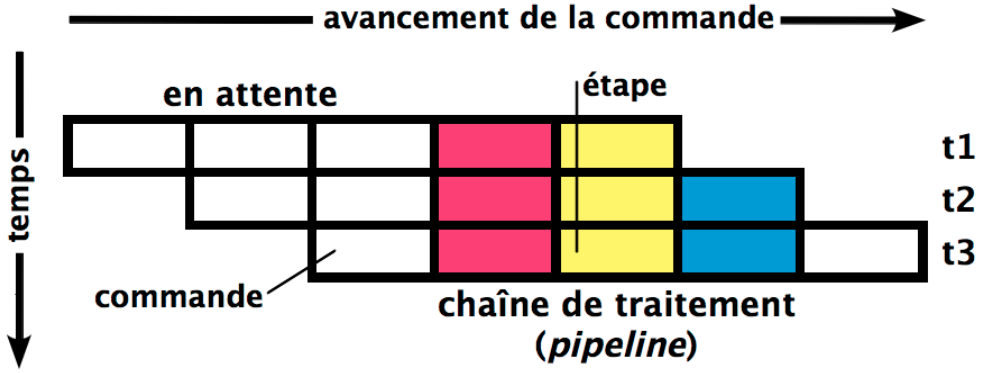


FIGURE 2 – Schéma du pipeline

prévu. Pour accélérer le tout, les informaticiens des années 60 ont inventé une méthode d'exécution nommée Pipeline, qui permet de grandement accélérer l'exécution. Sans rentrer dans les détails, l'idée est de décomposer une instruction en plusieurs sous-parties, qu'on peut empiler et décaler pour les exécuter simultanément. En effet, le processeur peut exécuter ses différentes étapes en même temps, préparant l'instruction suivante avant que la première finisse. Grâce à cette méthode, nous pouvons globalement retrouver l'idée "d'une instruction = un cycle d'horloge" .

Les périphériques E/S Les entrées / sorties sont primordiales pour tout ordinateur. En effet, c'est de cette manière que l'on peut connecter des périphériques à notre machine tel qu'un écran, un clavier, du stockage externe, etc. Cependant, pour une carte de développement, même si elle est équipée de ports USB, on utilise de très fines broches. En effet, la carte Explorer 16/32 est équipée de cent broches afin d'y connecter des périphériques d'Entrée/Sortie en tout genre. Ils sont appelés PMOD et très variés : prise jack, carte bluetooth, moteurs, etc. Dans notre cas, nous avons connecté le moteur et le capteur de luminosité, lui-même relié à un capteur de proximité. La documentation est primordiale pour l'utilisation de périphériques d'E/S. En effet, un port en entrée est lié à un autre pour la sortie, il faut donc les identifier correctement pour ensuite les régler dans le langage embarqué et pouvoir s'en servir.

Les registres et les ports Lors de la communication entre le processeur de la carte et les périphériques d'Entrées/Sorties, plusieurs questions se posent. La première consiste à comprendre le protocole de communication entre les deux éléments (processeur et périphérique) : comment la machine ne se trompe-t-elle pas entre deux périphériques ? Il s'agit d'une question de jonction entre le Hardware et le Software. La deuxième interrogation qui peut se poser concerne l'interprétation et la manipulation des données : comment les retrouver, les modifier, ou encore savoir si l'information doit être lue ou envoyée ?

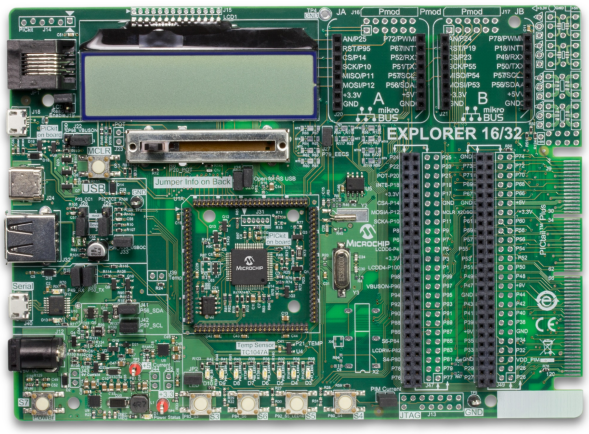


FIGURE 3 – Image de la carte explorer 16/32

Lorsque l'on connecte un périphérique à notre carte mère PIC32 dans le cadre de ce projet, on branche la première à un *port* ou à une partie de ce port (exemple le PORTD pour le PIC32). Chaque port est composé d'un nombre x de broches, dans notre cas un port à 16 broches. Dans la plupart des cas en programmation, les ports sont nommés par des lettres (ici A,B,C,D) et les broches sont numérotées. Ce port, ainsi que chaque broche, a une adresse et un emplacement mémoire attribués dans la carte, ainsi, quand deux périphériques sont branchés en même temps sur la carte, il n'y a pas de confusion possible entre ces deux entités. Chaque broche fait passer une information binaire (0 ou 1), ainsi un port est un ensemble de 16 bits (pour le PIC32), il y a donc 2^{16} états possibles. On pourrait représenter un état en binaire, néanmoins écrire 16 bits est fastidieux, les programmeurs préfèrent ainsi utiliser l'hexadécimal contenant 16 caractères (les caractères 0...9 et A...F). On remarquera alors que $16^4 = 2^{16}$ de ce fait, il ne faudra plus que 4 caractères pour exprimer nos 16 bits.

L'association entre chaque état binaire et hexadécimal n'est rien d'autre qu'une simple transformation de base (on passe de la base 2 à la base 16) les règles de calcul sont donc usuelles. Si je veux, par exemple, éteindre le bit 15 et le bit 0 au lieu d'écrire *0111111111111111110* je préférerai écrire *0x3fffe*.

La question se pose maintenant de savoir comment accéder à ces données et comment les interpréter au niveau Software. Pour cela, les données de communication doivent être stockées et doivent être accessibles au programmeur pour leur lecture/modification. Ces données sont ainsi stockées dans des registres relatifs aux ports physiques comme vu auparavant. Ces registres disposent d'une adresse unique de la forme : *0xFFFF* (hexadécimal 16 bits pour l'exemple de notre projet). Lorsqu'on interagit avec les données, on va modifier les bits se trouvant à l'intérieur du registre à l'adresse enregistrée. Comme les ports sont en 16 bits, il y aura à chaque adresse 16 bits de données correspondant aux valeurs binaires de chaque broche. Ainsi pour reprendre le même exemple que tout à l'heure, si le Port

D est stocké à l'adresse *0xFFFF* et qu'on trouve en lisant les données : *0x3fffe*, cela veut dire que les bits 15 et 0 sont éteints et les autres allumés. Au niveau programmation, l'adresse des ports est déjà enregistrée et nous pouvons donc utiliser des variables (sous le nom de *PORTx* ou x est le nom du port) pour y accéder.

Se pose maintenant la dernière problématique : comment interpréter les données ? Faut-il les envoyer ? Les recevoir ? Sont-elles analogiques ? Numériques ? Pour cela, il existe deux registres sous le nom de *ANSELx* et *TRISx* (x étant le nom du port) ; chaque registre code une caractéristique pour chaque port et chaque broche. Comme il y a 16 broches, les registres sont sur 16 bits (à chaque broche son bit). Les registres *ANSELx* permettent de définir si une broche est une broche analogique ou numérique (0 pour numérique). Les registres *TRISx* permettent de définir si une broche est une broche d'entrée ou de sortie (1 pour l'entrée). Ainsi si *TRISD* = *0x3fffe* le premier et le dernier bit sont en sorties et les autres sont en entrées.

Le Timer Le Timer est utile pour nous aider à compter précisément le temps avec l'horloge interne. Il existe 3 timers sur notre microprocesseur, correspondant aux registres TMR1, TRM2 et TMR3. Le premier ayant la capacité supplémentaire de fonctionner de façon asynchrone comparé au microprocesseur. Son fonctionnement est le suivant : une fois activé, la valeur dans le registre TMR1 augmente en suivant la fréquence de la Clock entrante, à la façon d'un compteur. Cette horloge peut être mise à échelle "prescaled" selon 1 :1, 1 :8, 1 :64 ou 1 :256 par rapport à la SysClock.

Ainsi nous pouvons suivre le temps écoulé depuis son activation ou depuis que le registre TMR1 a été remis à 0 et nous pouvons faire cette fonction de délai. Le registre du Timer 1 est remis à 0. Nous configurons le Timer sur le registre T1CON en nous assurant que l'horloge entrante soit 1 :1 Sysclock et qu'aucune option supplémentaire ne soit activée. Le seul bit à 1 est le bit "ON" permettant le démarrage du timer. Sachant que la SYSCLOCK est une constante égale à la fréquence, ici 40000MHz. Elle fait 1 tour en 1 seconde, alors si TMR1=SYSCLOCK/1000, il se sera passé 1 milliseconde en 1 tour de timer. Il nous suffit alors de mettre le timer à 0 et d'attendre le nombre de millisecondes voulu. Voici le code associé :

```
1   // Code pour régler le Timer
2   void Delay_ms (unsigned t)
3   {
4           TMR1 = 0;
5           T1CON = 0x8000; // debut timer 1 pré-configuré 1
6       while(t--)            // décrémente t à chaque tour jusqu'à t=0
7                             +// donc si t=10, il y aura 10tours de boucle
8           { TMR1 = 0;
9       while (TMR1 < SYSCLK/1000); // Délai 1ms
10            }
11  }
```

Le convertisseur analogique-numérique Le Convertisseur Analogique-Numérique (CAN), aussi appelé en anglais ADC pour Analog to Digital Converter est un composant très important en électronique qui permet de convertir un signal analogique, comme une tension, en valeur numérique codée sur plusieurs bits. Le conversion s'opère en fragmentant le signal analogique en n valeurs moyennes de son fragment. Le signal analogique correspond visuellement à une courbe sur laquelle on peut zoomer à l'infini. Le signal numérique, lui, correspond à une succession de segments horizontaux, ce qui forme une marche. Le problème de ce procédé réside dans la perte d'information à cause de la compression du signal. Plus l'échantillonnage est grand, c'est à dire le nombre de fragments, meilleure est la qualité.

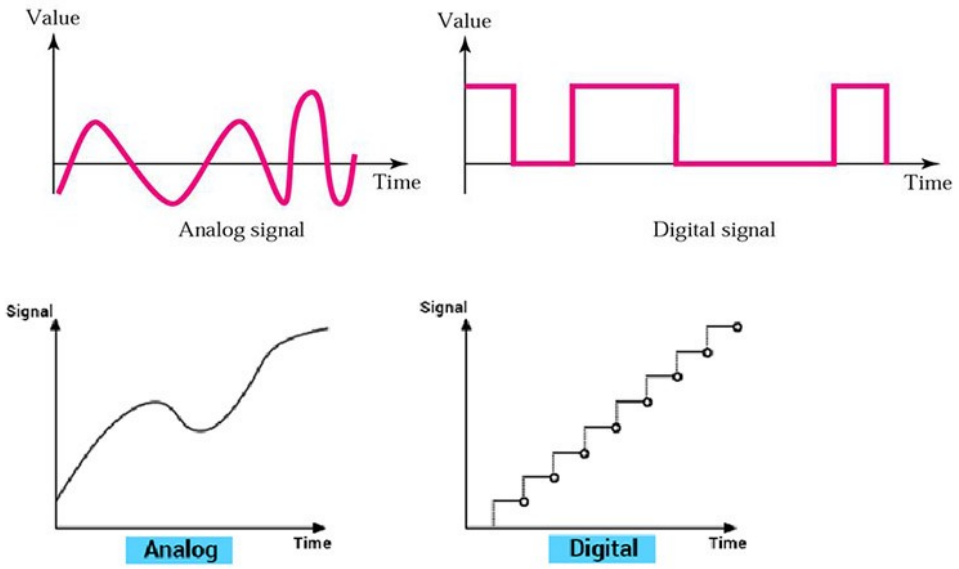


FIGURE 4 – Comparaison visuelle entre un signal analogique et un signal numérique

2 Méthodologie et organisation du travail

Dans cette partie, nous allons décrire l'organisation du cours de P6 ainsi que celle pour le projet en groupe.

2.1 Description de l'organisation adoptée pour le déroulement du travail

2.1.1 Travail commun des groupes

Au début du semestre, nous étions par groupe de 2 ou 3 pour l'initiation à la programmation embarquée et les travaux de découverte de MPLAB. En effet, le fonctionnement d'un microprocesseur et le C embarqué sont des connaissances nouvelles et très techniques. Les applications en live sur les carte Explorer 16/32 nous ont permis de mieux comprendre ce que concrètement le code permettait de faire avec les machines et de communiquer avec elles. Nous avons acquis une base très synthétique et nécessaire à l'élaboration de notre projet spécialisé. Cette phase d'apprentissage et de démonstrations a duré neuf semaines.

2.1.2 Organisation du travail pour le projet

Une fois l'initiation terminée, nous avons formé deux groupes de 6 dans la classe et réfléchi chacun à un projet concret à entreprendre. Des idées assez simples nous sont venues, comme par exemple une poubelle automatique. Nous nous sommes mis d'accord sur le système suivant : un détecteur qui permet d'activer un moteur. Pour que cela s'accorde avec notre quotidien, nous avons imaginé que ce système pourrait servir dans un distributeur de gel hydroalcoolique.

Pour mener à bien ce projet, nous avons dû combiner le travail en présentiel et en distanciel. Pour ce faire, les personnes présentes en salle s'occupaient de la manipulation du matériel et du codage ; et celles à distance se sont focalisées sur la compréhension et la bibliographie. Nous avons créé un serveur Discord pour pouvoir nous réunir et communiquer au fur et à mesure sur l'avancement du projet. Pendant les dernières semaines, les réunions nous ont permis de faire le point sur ce qu'il restait à faire et de répartir le travail de rédaction. La répartition s'est faite en fonction des connaissances que chacun a acquises et de ses préférences.

2.1.3 Répartition du travail pour le projet

Nom	Travail réalisé
Adrien QUIBEUF	Réalisation du projet (programmation du moteur et de son contrôleur, programme principal), rédaction du rapport (fonctionnement de la partie actionneur), réalisation du poster. Oral : Projet, moteur et programmation.
Paul KRAUSKOPF	Réalisation et programmation du projet (capteur IR, moteur et mise en commun), rédaction du rapport(timer, méthodologie, projet, composants et programmation), plan rapport et poster. Oral : Projet, capteur IR et programmation.
Nicolas ROSAL	Participation à la recherche sur le moteur et les capteurs, plan initial du rapport, rédaction du rapport (fonctionnement général du processeur, Port et registres), participation à la création du poster. Oral : Fonctionnement général du microprocesseur, CAN et E/S.
Elisa RECORDON	Plan initial du rapport, rédaction du rapport (conclusion, travail réalisé avant le projet : LEDs, capteur de température, bouton glissière) Oral : travail réalisé avant le projet
Tom TESNIERE	Rédaction du rapport (introduction, travail en groupe et organisation du projet), écriture du document complet en Latex. Oral : introduction du projet
Ryan JUTEAU	Plan initial, rédaction du rapport, relecture et optimisations LaTeX, parties CAN, E/S, Méthodologie et Organisation du travail. Travail sur le poster et le diaporama. Oral : Fonctionnement de l'Explorer 16/32 et du microprocesseur.

3 Travail réalisé et résultats

Dans cette troisième partie, nous allons découvrir l'IDE utilisé, certains composants de la carte PIC et enfin décrire notre projet et les résultats.

3.1 Introduction au microprocesseur

3.1.1 Découverte de MPLAB

Le logiciel MPLAB est un outil de développement ou IDE, gratuit, qui permet de créer un programme, l'assembler et le simuler. Son interface graphique et ses nombreuses fonctionnalités permettent d'améliorer l'espace de travail.

Cet IDE comporte deux modes distincts. Le premier est classique pour un tel logiciel, il permet à l'utilisateur de coder en C et l'IDE crée une arborescence de fichiers propre à son fonctionnement. Le deuxième mode est son principal intérêt, il consiste en une simulation qui permet de tester son code dans un environnement virtuel sans endommager la carte. Ainsi, le programmeur peut changer des options et tester chaque fois son nouveau code afin de l'améliorer, sans passer par la carte.

Enfin, MPLAB permet de compiler le code une fois terminé et de le charger sur la carte PIC pour l'exécuter à partir de l'ordinateur.

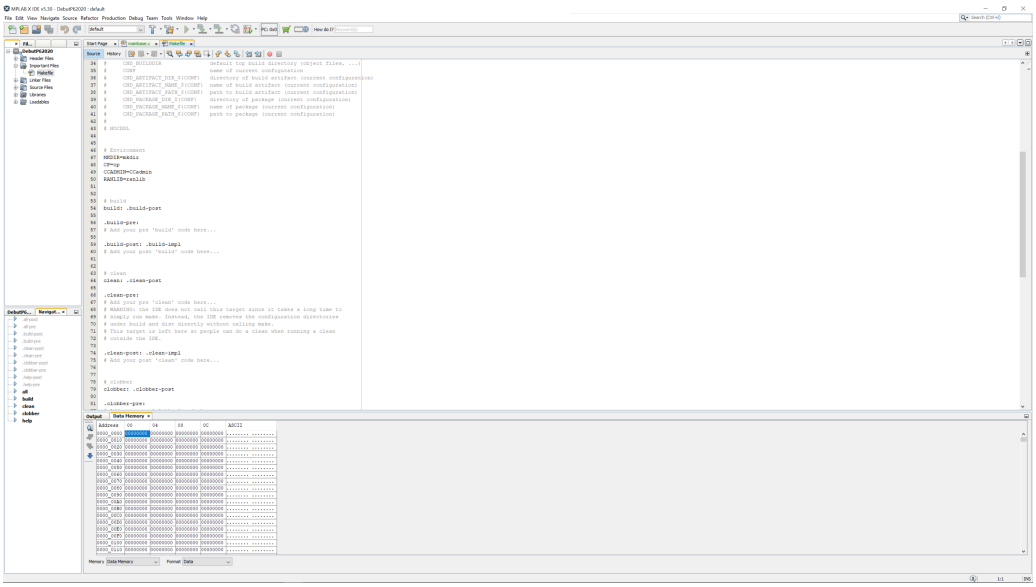


FIGURE 5 – Interface de MPLAB X IDE

3.1.2 LED

Pour nous familiariser avec l'utilisation de MPLAB et du microprocesseur, nous avons d'abord travaillé sur les LEDs présentes sur la carte. L'objectif était d'allumer de façon alternée les 4 premières LEDs et les 4 dernières. Premièrement, nous avons mis les LEDs en sortie : TRISA = 0xC600. Ensuite, dans une boucle infinie, nous avons allumé et éteint les LEDs en affectant à PORTA la valeur 0x000F puis 0x00F0. Afin d'assimiler le fonctionnement de ce programme, nous l'avons exécuté ligne par ligne.

Enfin, pour mieux observer le clignotement des LEDs, nous avons ajouté un délai après chacune des 2 instructions. Nous ne maîtrisions pas encore le Timer, alors nous avons utilisé des boucles for (vides). Connaissant la fréquence (40MHz) et le nombre d'opérations effectuées par le microprocesseur lors de l'exécution d'une boucle for, nous avons pu calculer le temps nécessaire à un tour de boucle. Avec un produit en croix, nous avons obtenu le nombre de tours de boucles à effectuer pour obtenir le délai choisi (en l'occurrence 0,5s). Ainsi, nous avons pu observer le clignotement des LEDs qui restaient allumées pendant 0,5s.

3.1.3 Capteur de température

La carte PIC32 possède un capteur de température *TC1047*, que nous avons utilisé pour mesurer la température de la pièce. Une grande partie du programme nous était fournie, nous pouvions donc directement utiliser valADCTemp. Pour obtenir la température (variable Temp), nous avons effectué ce calcul : $Temp = (valADCTemp * 3.3/1024 - 0.5) * 100$, avec 3.3/1024 comme pas de quantification. Nous avons ensuite regardé la valeur de la variable Temp dans la fenêtre Watches de MPLAB.

3.1.4 Bouton glissière

La première partie du cours de P6 a été guidée par le même objectif de découverte des fonctionnalités du microprocesseur. Dans la continuité des deux éléments précédents, nous avons été amenés à utiliser un autre composant de la carte, le bouton glissière (potentiomètre rectiligne). Celui-ci nous a servi à observer le fonctionnement d'une entrée/sortie.

3.2 Projet

3.2.1 Idée générale du projet

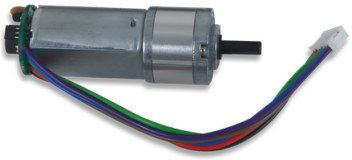
Nous avons voulu trouver un projet qui contienne à la fois une partie “capteur” et une partie “actionneur”, le tout formant un objet technique utile. Ainsi, deux idées ont découlé de ce choix : l'ouverture d'une poubelle ou la distribution de gel hydroalcoolique par détection. Finalement, la deuxième proposition a été celle adoptée car elle nous a semblé plus réalisable techniquement et la plus en accord avec l'actualité, la rendant utile d'un point de vue sanitaire. En effet, la détection évite le contact avec le distributeur et la main. Nous avions donc un objectif : réaliser l'électronique embarquée pour un distributeur de gel hydroalcoolique sans contact.

3.2.2 Composants

Moteur Le moteur est l'actionneur de notre système. C'est lui qui déclenche le distributeur pour qu'une dose de gel soit donnée à l'utilisateur. Afin de simuler la distribution, le moteur sera actionné pendant un temps défini lorsque l'utilisateur passe sa main devant le capteur. Pour contrôler le moteur, nous utilisons un *Pmod* nommé *HB5*, qui comprend un pont en *H*. Il peut contrôler les petits et moyens moteurs DC (à courant continu).

Dans notre cas, nous utilisons un moteur DC qui peut fonctionner jusqu'à 12V. Il est accompagné d'un réducteur 1 :19 qui permet d'avoir un couple important ainsi qu'une vitesse moindre. Le moteur présente aussi un circuit avec capteur de vitesse permettant de connaître la vitesse du moteur à tout moment. Le pont en H du PMOD autorise l'inversion du sens de rotation et le contrôle de la vitesse des moteurs. Dans notre cas, nous garderons un sens de rotation et une vitesse fixes, définis dans la partie configuration de notre code.

Le contrôle de la vitesse du moteur avec la carte se fait grâce à un port *PWM* (*Pulse Width Modulation*). Le *PWM* est une technique couramment utilisée pour synthétiser des signaux pseudo analogiques. Elle sert à générer un signal pseudo analogique à partir d'un environnement numérique ou analogique. Le principe général est qu'en appliquant une succession rapide d'états discrets avec des ratio de durée bien choisis, on peut obtenir, en ne regardant que la valeur moyenne du signal, n'importe quelle valeur intermédiaire. Pour contrôler la vitesse du moteur, il faut donc moduler la largeur de l'impulsion. On alimente le *Pmod* en 3,3V et le moteur en 12V via une alimentation externe.



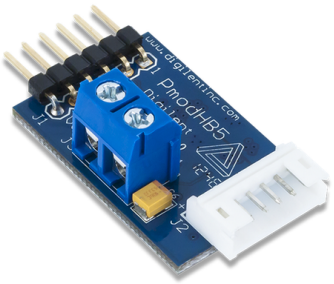


FIGURE 6 – Images du moteur et du Pmod LS1

Capteur infrarouge Seul capteur de notre système, le capteur infrarouge a pour rôle la détection de la main de l'utilisateur. Pour ce faire, nous utilisons un capteur infrarouge de proximité qui sera connecté via la norme de périphérique *Pmod* à la carte *LS1* pour "*Light Sensor 1*". Cette carte fonctionne comme une interface entre les capteurs et le microprocesseur. Elle peut recevoir jusqu'à 4 capteurs différents. Le fonctionnement combiné de la carte et d'un capteur est le suivant :

Une fois alimenté, le capteur émet de la lumière infrarouge via une diode. Une photorésistance de l'autre côté du capteur réagit à la lumière de la diode qui lui est réfléchie. La carte *LS1* reçoit en continu un signal analogique correspondant à la quantité de lumière infrarouge reçue par le capteur. La carte *LS1* compare ensuite la force de ce signal, entre 0V et 3.3V, avec la limite de déclenchement choisie grâce à une résistance réglable. Si la comparaison analogique montre que le capteur reçoit plus de lumière que le seuil prévu, le bit de sortie correspondant à ce capteur prend la valeur logique 1 et le microprocesseur en est donc informé. Sinon, le bit de sortie est à la valeur logique 0.

FIGURE 7 – Image du capteur infrarouge

3.2.3 Programmation

Idée générale Voici le programme principal en pseudo code que nous avons choisi. L'idée générale est la suivante : une boucle continue s'assure que le moteur est à l'arrêt et que les leds sont éteintes, et attribue à la variable 'vrai' la valeur du bit correspondant au capteur. Si le capteur a détecté une main, alors le programme met en route le moteur et les leds pour une durée de 2s, le temps pour recevoir une dose de gel hydroalcoolique.

```
ProgrammePrincipal
Déclarations
    vrai : Boolean

Début
    Configurations
    Tant que 1 Faire
        leds_éteintes()
        PWM(0)
        vrai ← D.bit1

            Si  vrai   Alors
                PWM(40)
                leds_allumées()
                delay_ms(2000)
       FinSi

    FinTantQue
Fin
```

Nous avons de manière générale, grâce à l'IDE, les commandes pour récupérer la valeur d'un bit, la commande pour donner une valeur à un bit et pour les configurations nécessaires pour l'utilisation en entrée/sortie.. Il nous manque alors les différentes procédures 'leds_éteintes()', leds_allumées()', PWM(), delay_ms(). En ce qui concerne 'delay_ms()', nous nous sommes servis de notre précédent travail sur le Timer qui nous a aussi été utile pour l'utilisation des leds qui ne servent qu'à titre d'indication, de débug.

Fonctionnement code du capteur Dans notre montage, le port D était celui choisi pour nos connexions. Nous avons 8 bits disponibles dessus en "GPIO" pour General Purpose Input/Output, soit une utilisation de capteurs et actionneurs externes. Nous avons donc utilisé la broche D1 pour recevoir les données de la carte LS1. Selon la documentation, cette broche D1 correspond à la pin P76. La première étape est de certifier dans le registre que ce second bit du port D sera utilisé en numérique. On doit alors mettre à 0 via la commande ANSELD les bits voulus. De plus, il faut renseigner dans le registre que le bit 1 du port D est en entrée. Pour ce faire, il faut renseigner le bit correspondant à 1 avec

la commande TRISD. Pour lire la valeur de ce bit, il suffit d'utiliser la commande PORTDbits.RD1, à comprendre : valeurs du bit 1 du port D. Tout est représenté dans ce programme, avec en commentaire un "debug" avec les leds pour s'assurer du fonctionnement.

```
1   // Code pour le capteur
2   int vrai;
3
4   int main(void)
5   {
6       //TRISA = 0xC600;    //Debug led
7
8       ANSELD=0xF0C0;              //bit 1 en numérique
9       TRISD = 0xFFFF;              //bit 1 en entrée
10
11      while(1){
12
13          //PORTA = 0x0000;           //Éteint les leds
14          vrai = PORTDbits.RD1;
15
16          if( vrai==1 )
17          {
18          //PORTA = 0x00FF;         //allume les leds
19          Delay_ms(2000);
20          }
21      }
22  }
```

Fonctionnement code du moteur Maintenant que nous savons comment récupérer l'information du capteur, nous devons actionner le moteur lorsque le capteur envoie un signal vrai. Le principal défi du contrôle du moteur est la configuration du pin PWM. Nous réglons la fréquence du PWM à 2000Hz (d'après la documentation du PMOD HB5). Nous définissons la période de travail de chaque cycle, qui correspond au pourcentage de temps de la période où le signal est haut. Par exemple , si on fixe "DUTY_CYCLE" à 50, alors le signal du PWM sera haut pendant 50% (la moitié) du temps de la période.

Pour notre carte, c'est le module Output Compare du PIC32 qui gère le PWM. Il faut donc le configurer. En premier, on configure le port du moteur en tant que sortie avec "TRISD = 0xFFFE ;". On configure l'Output Compare pour associer le port du moteur qui se nomme RB7 avec "RPD0Rbits.RPD0R = 0x000C ;". On définit le mode de fonctionnement standard pour l'Output Compare avec "OC1CON = 0x0006 ;". Il faut maintenant attribuer le temps en seconde de la période, on le calcule grâce à la fréquence de l'horloge interne du processeur et grâce à la fréquence du PWM (2000Hz). La ligne de

code correspondante est "PR2 = (SYSCLK / PWM_FREQ) - 1;". Enfin, on calcule le temps où le signal doit être haut avec "OC1RS = (PR2 + 1) * ((float)DUTY_CYCLE / 100);" et on active les timers et l'Output Compare Module.

Pour contrôler ensuite le démarrage et l'arrêt du moteur, on change la valeur du "duty cycle". On laisse tourner le moteur pendant 2 secondes grâce à la fonction delay(). Tout ceci est représenté dans le programme final présent en annexe.

Mise en commun Maintenant que nous disposons du code pour chacun des composants et que nous avons notre pseudo-code pour le programme final, il suffit de mettre tout en commun en suivant le pseudo-code. Nous avons dû adapter la configuration des ports pour qu'elle prenne en compte chaque composant, c'est-à-dire être sûrs que les ports sont bien en entrée ou en sortie et en numérique ou analogique. Par exemple : le port D est utilisé par le capteur et l'actionneur, donc le bit 0 doit être en sortie et le bit 1 en entrée. Ce qui donne : TRISD=0xFFFE. Il faut faire de même avec ANSELD=0xF0C0.

Voici le programme principal que nous avons réalisé :

```
#define SYSCLK  40000000
#define PWM_FREQ        2000
int DUTY_CYCLE;
int vrai;
void Delay_ms (unsigned t)
{
        TMR1 = 0;
        T1CON = 0x8000; // start timer 1 prescaler 1
        while(t--)
        { TMR1 = 0;
         while (TMR1 < SYSCLK/1000); // Délai 1ms
        }
}
int main(void)
{
        T2CONSET = 0x8000;                  //Configuration Timer
        OC1CONSET = 0x8000;

        TRISA = 0xC600;                 //Configuration E/S
        ANSELD=0xF0C0;
        TRISD = 0xFFFE;

        DUTY_CYCLE = 100;

```

```
25                  RPDORbits.RPDOR = 0x000C;                   //Configuration PWM
26                  OC1CON = 0x0006;
27                  PR2 = (SYSCLK / PWM_FREQ) - 1;
28                  OC1RS = (PR2 + 1) * ((float)DUTY_CYCLE / 100);
29
30                  // Boucle infinie
31                  while(1){
32                       PORTA = 0x0000;
33                       DUTY_CYCLE = 0;
34                       OC1RS = (PR2 + 1) * ((float)DUTY_CYCLE / 100);
35                       vrai = PORTDbits.RD1;
36
37                       if( vrai==1 )
38                       {
39                          DUTY_CYCLE = 40;
40                          OC1RS = (PR2 + 1) * ((float)DUTY_CYCLE / 100);
41                          PORTA = 0xFFFF;
42                          Delay_ms(2000);
43                         // le programme est une boucle sans fin
44                         }
45                  }
46                  return 1;
47             }
```

4 Conclusion et perspectives

4.1 Conclusion sur le travail réalisé

Le travail réalisé a satisfait notre volonté initiale d'avoir une partie capteur et une partie actionneur. Le capteur infrarouge a été correctement calibré. Ainsi, l'intérêt sanitaire de notre projet est respecté, le dispositif permet bien d'éviter le contact entre le distributeur de gel hydroalcoolique et la main de l'utilisateur.

4.2 Conclusion sur l'apport personnel de cet EC projet

Comme mentionné dans l'introduction de notre rapport, l'informatique embarquée est un domaine aux applications très diverses. En tant qu'ingénieurs, nous serons certainement amenés à rencontrer une de ces applications. La culture scientifique en électronique que nous a fourni cet EC projet se révèlera sûrement très utile à ce moment-là. À cela, on peut ajouter l'acquisition de compétences : nous avons pu appréhender la programmation d'un microprocesseur, en particulier avec le logiciel MPLAB, et l'utilisation du langage C embarqué, qui pourra faciliter l'apprentissage du langage C à l'avenir.

4.3 Perspectives pour la poursuite de ce projet

Pour aller plus loin, nous avions l'intention d'ajouter une fonctionnalité Bluetooth à notre projet. Malheureusement, nous avons manqué de temps, mais c'est tout à fait réalisable. Nous avons retrouvé l'application qui gère le Bluetooth (Serial Bluetooth Terminal). L'idée que nous avions consistait à enregistrer le nombre d'activations journalières du distributeur de gel hydroalcoolique, et de pouvoir consulter les données sur son téléphone, via l'application.

4.4 Retour commun sur le contenu et la mise en place de l'EC de P6

Du fait de la situation sanitaire exceptionnelle, il a été globalement difficile de suivre les cours en distanciel, tant pour la théorie de l'embarqué que pour les manipulations avec la carte de développement. Les notions très théoriques des premiers TP sont assez déroutantes. En effet, nous avons l'impression qu'il nous manque des bases nécessaires pour faciliter leur compréhension. Hormis ces difficultés, nous sommes satisfaits car cet EC est un vrai plus qui nous a permis de découvrir des facettes importantes et intéressantes de l'ingénierie que nous serons amenés à retrouver dans notre formation ou même notre future carrière.

4.5 Remerciements

Nous tenons à remercier M. GRISEL pour nous avoir assistés et guidés durant ce projet. En effet, en totale autonomie et sans des programmes pré-codés pour facilement communiquer avec la carte Explorer16/32, nous aurions été incapables d'appréhender MPLAB et le langage embarqué.

Bibliographie

- **Hexadécimal :** https://fr.wikipedia.org/wiki/Syst%C3%A8me_hexad%C3%A9cimal
- **Pipeline :** https://fr.wikipedia.org/wiki/Pipeline_(architecture_des_processeurs)
- **Bus de données :** https://fr.wikipedia.org/wiki/Bus_de_donn%C3%A9es
- **Horloge et pipeline :** https://www.commentcamarche.net/contents/763-processeur
- **IR Proximity Sensor (2-pack) :** https://store.digilentinc.com/ir-proximity-sensor-2-pack/
- **Pmod LS1 : Infrared Light Detector :** https://store.digilentinc.com/pmod-ls1-infrared-light- https://reference.digilentinc.com/pmod/pmodls1/reference-manual https://microchipdeveloper com/boards:explorer1632
- **CAN :** https://fr.wikipedia.org/wiki/Convertisseur_analogique-num%C3%A9rique https: //miro.medium.com/max/800/1*to1hS64KaO_8awfn_HHIwg.jpeg
- **MPLAB :** https://www.microchip.com/en-us/development-tools-tools-and-software/ mplab-x-ide
- **PWM PIC 32 :** http://umassamherstm5.org/tech-tutorials/pic32-tutorials/pic32mx220-tutoria pwm
- **Liaison série RS232 :** http://www.groupeisf.net/reseaux_informatiques/reseaux_et_ telecommunications/reseaux/liaisons_series/rs232.html?fbclid=IwAR2wZ3IEKfBWabjbqHugMudZY4E
- **Timer** http://ww1.microchip.com/downloads/en/DeviceDoc/61105F.pdf
- **Photo Explorer 16/32 :** https://www.microchip.com/developmenttools/ProductDetails/ dm240001-3