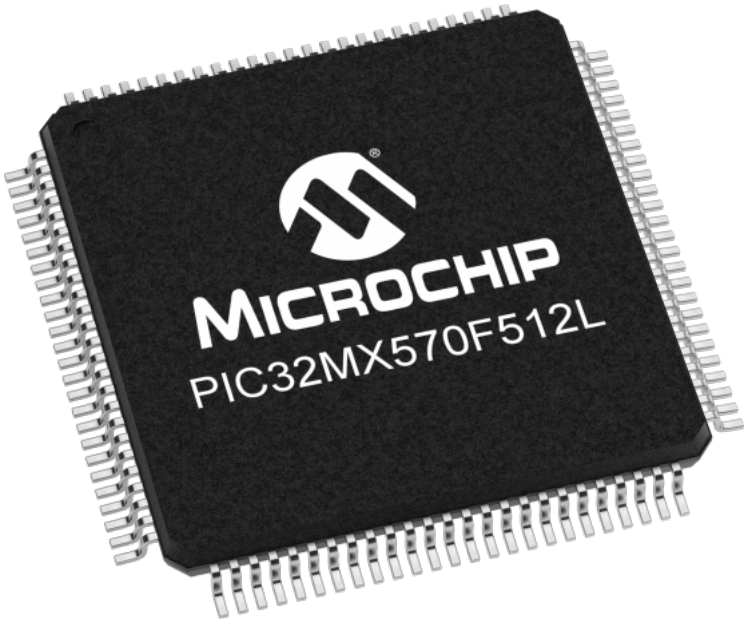
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

Projet de Physique P6
STPI/P6/2020-13

Programmation informatique embarquée : Initiation au Microprocesseur et à la Programmation



Étudiants :

Julie Allais
Yasmine Dubuget
Adrien Neveu
Florin Croitoru
Romain Moussy
Yizhe Wang

Enseignant-responsable du projet :
Richard Grisel

Date de remise du rapport : 15/06/2020

Référence du projet : STPI/P6/2020 – 13

Intitulé du projet :Programmation informatique embarquée : Initiation au Microprocesseur et à la Programmation

Type de projet : *expérimental, simulation*

Objectifs du projet : Les objectifs de ce projet sont la découverte des microprocesseurs ainsi que découvrir comment les programmer. Pour cela nous allons à travers ce projet découvir le language C ainsi que chercher à créer des programmes dans ce language.

Mots-clefs du projet : microprocesseur, programmation

Table des matières

Introduction

Ce projet a eu pour objectif de nous initier aux microprocesseurs ainsi que nous faire découvrir la programmation sur microprocesseur. Ce projet nous a alors aussi permis de nous familiariser avec le language de programmation C ainsi que de découvrir l'environnement de développement de MPLAB.
Aujourd'hui les microprocesseurs sont présents dans une grande majorité de nos appareils éléctroniques tels que les calculatrices, les appareils électroménagers, etc... Ainsi ce projet avait comme objectif de nous permettre de répondre aux questions : Qu'est-ce qu'un microprocesseur ? Comment il fonctionne ? Comment le programmer pour qu'il réalise les tâches que je veux ?
Ainsi dans ce rapport nous allons vous exposer les différentes choses que nous avons appris lors de ce projet et ainsi pouvoir répondre aux différentes questions ci-dessus. Nous allons aussi vous présenter les différentes manipulations que nous avons pu réaliser tout au long du semestre ainsi que le programme que nous avons développé sur notre PIC32.

Chapitre 1

Méthodologie, organisation du travail

Globalement nos séances pouvaient être séparées en deux parties : une partie compréhension, où on essayait de comprendre les différents mécanismes composant un microprocesseur, il s'agissait donc d'une partie théorique, et l'autre partie où on faisait du code en C sur MPLAB, c'est à dire la partie pratique, et on essayait d'appliquer ce que l'on avait appris à l'aide de la théorie.

Durant toute la partie avant le confinement nous avons alors travaillé à deux par ordinateur avec à ce moment là un microprocesseur par binôme. Cela permettait à un maximum de personnes de manipuler un microprocesseur tout en facilitant de l'entraide lors de la compréhension des différents codes en C.

Concernant le confinement, cela a fortement changé notre façon de travailler. En effet, à cause de l'éloignement nous nous sommes mis à travailler essentiellement sur Discord et Zoom. Cela nous permettait alors de pouvoir partager nos codes ainsi que voir à l'aide de nos caméras les résultats sur les cartes pour ceux qui n'en avaient pas.

Ainsi nous avons passé la majorité de nos premières séances à faire de la théorie. Une fois tous les principaux modules de notre carte découverts, nous nous sommes lancés dans la réalisation de notre propre algorithme. Nous avons décidé de faire un jeu et avons donc passé les 5 dernières séances à la réalisation de ce jeu ainsi qu'à l'écriture du rapport.

Chapitre 2

Outils à notre disposion

2.1 Carte Explorer 16/32

Pour réaliser notre projet nous disposons d'une carte Explorer 16/32 munie d'un microprocesseur PIC32MX570F512L. Cette carte est munie d'un certain nombre d'entrées/sorties que l'on peut programmer. Elle est munie en premier lieu de plusieurs entrées cable pour pouvoir la brancher à des appareils tels que des ports micro USB type B et type C. Elle dispose de d'autres types d'entrées telles que des boutons poussoirs, un capteur de température ou bien un potentiomètre. Elle dispose aussi de sorties telles que des LEDs ainsi qu'un écran LCD.

Voici par exemple une photo de la carte :

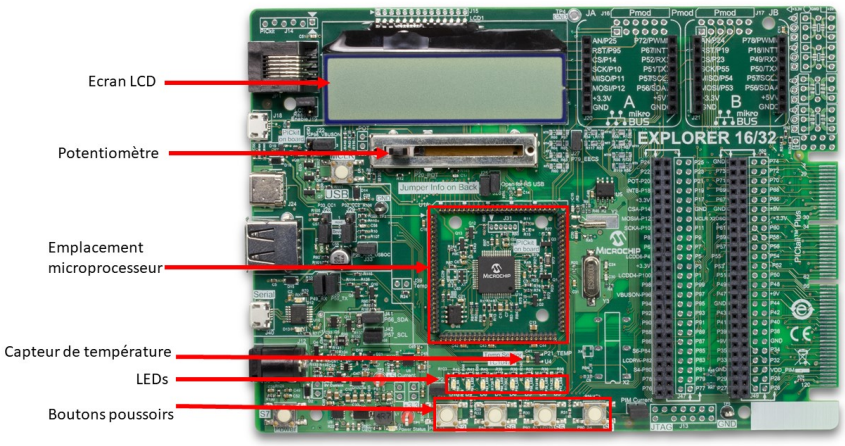


FIGURE 2.1 – image de la carte Explorer 16/32

2.2 MPLAB

MPLAB IDE est un système fonctionnant sur un ordinateur personnel logiciel pour développer des programmes d'application pour les microcontrôleurs Microchip et les contrôleurs de signaux numériques.
Parce qu'il fournit un "environnement" intégré unique pour le code de développement de microcontrôleur intégré, il est appelé un environnement de développement intégré ou IDE. Il permet aux utilisateurs de créer des fichiers source et d'éditer, de compiler, de simuler et de déboguer des programmes, et d'envoyer le code généré au microcontrôleur cible via un programmeur ou un débogueur compatible. Ainsi, comme tout IDE, on peut utiliser l'écriture comme C ou d'autres assembleurs. Son éditeur de programme aide à écrire du code correct avec les outils et le langage de votre choix . Lorsque le code est fonctionnel et compilé, on peut l'exécuter sur la carte.
Cet IDE donne accès à de nombreuses options permettant de faciliter le développement. On a par exemple accès à un simulateur permettant de pouvoir tester le code sans utiliser la carte. Il nous permet aussi d'avoir accès à plusieurs modes d'éxécution : ligne par ligne ou en continu. Pour finir il nous permet d'obtenir de nombreuses informations telles que le pourcentage de mémoire utilisé par le programme, le temps que met une instruction pour s'exécuter, etc...
Voici à quoi ressemble cet IDE :

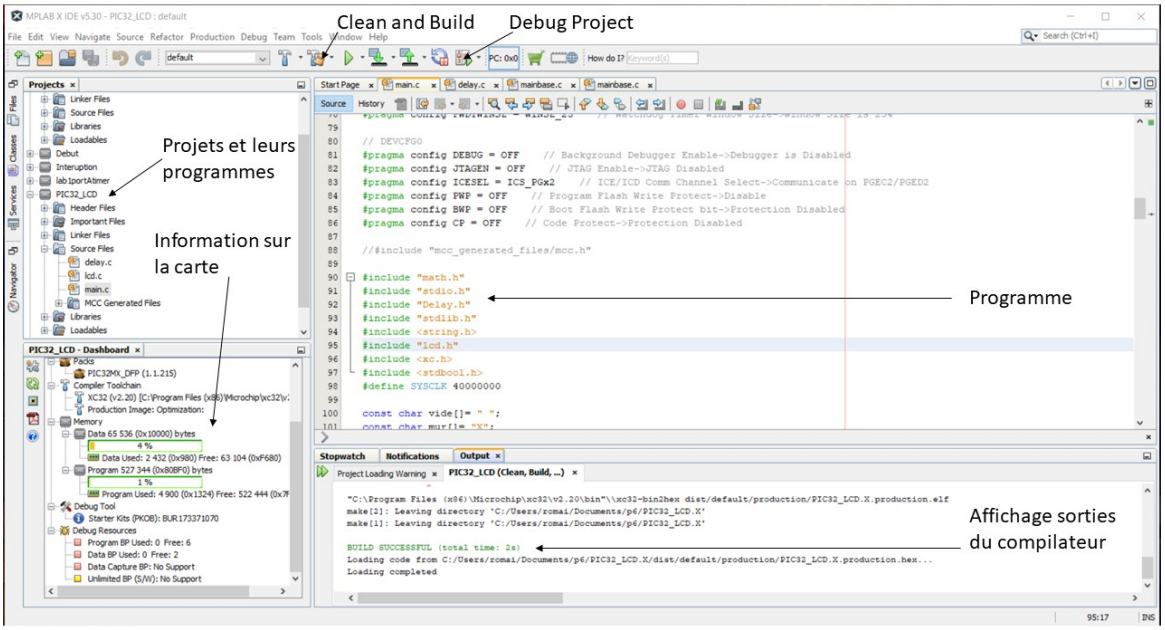


FIGURE 2.2 – Capture d'écran de MPLAB

2.3 Le langage C

Pendant notre projet nous avons utilisé le logiciel MPLAB qui se base sur le langage C. Nous allons donc aborder quelques notions sur le langage C.

Ce langage est très utile parce qu'il nous permet de manipuler la machine par exemple sur la gestion de la mémoire ou pour utiliser et réaliser les « fondations » (compilateurs, interpréteurs...).

Comme tous les langages de programmation compilés il y a quelques étapes nécessaires pour implémenter un code en C. Ces étapes sont les suivantes :
1. Créer
2. Compiler
3. Exécuter
4. Obtenir les sorties

Attention pour créer, compiler, exécuter et obtenir les sorties, il faut d'abord installer un compilateur C sur sa machine Voici quelques commandes et syntaxes de programmation de C :

1.$\#include < stdio.h >$ Il s'agit d'une commande de processeur qui inclut le fichier de sortie d'entrée standard (stdio.h) de la bibliothèque C avant de compiler un programme C

2. $int\ main()$ Il s'agit de la fonction principale d'où commence l'exécution de tout programme C.

3. { Cela indique le début de la fonction

4. $/ * some - comments * /$ permet de commenter son programme

5. Printf ("HelloWorld !") ; La commande printf imprime la sortie sur l'écran : Hello World !

6. Return 0 ; Cette commande termine le programme C (fonction) et renvoie 0.

7.} Cela indique la fin de la fonction principale.

8. Void name() Le mot-clé void peut être utilisé là où se place habituellement le type de retour d'une fonction, comme int pour un entier ou string pour une chaîne de caractères. Lorsque le programmeur écrit void, cela permet d'indiquer que la fonction ne renvoie rien.

Chapitre 3

Travail réalisé et résultats

3.1 Fonctionnement microprocesseur

3.1.1 Qu'est qu'un microprocesseur ?

Qu'est-ce qu'un microprocesseur ?

Un microprocesseur est simplement un processeur dont les composants ont été miniaturisés. Cette miniaturisation fût possible grâce à l'entreprise Intel depuis 1971. Cette invention fût une énorme avancée en électronique car elle permet de pouvoir mettre plusieurs composants électroniques sur un même circuit intégré. Le microprocesseur est encore aujourd'hui un élément présent dans une grande majorité de nos appareils électroniques. Les microprocesseurs, au même type que les processeurs peuvent être caractérisés par :

— Leurs jeux d'instructions
— La complexité de leur architecture
— L'horloge interne
— Le nombre de bits

De manière générale tous les microprocesseurs fonctionnent sur un cycle d'instruction composé de trois étapes :

— récupérer l'instruction (fetch), c'est-à-dire il doit aller dans la mémoire pour chercher l'instruction
— la décoder (decode), l'unité de commande transforme l'instruction en une suite de commandes permettant de la traiter
— l'exécution (execute), où le programme est exécuté. Pour notre part nous avons utilisé un microprocesseur du fabricant Microchip le : PIC32MX570F512L. Il s'agit d'un microprocesseur avec une horloge interne maximale de 50MHz avec 32bits. A noter que nous l'avons utilisée à une fréquence de 40MHz.

Enfin voici l'architecture globale de notre microprocesseur :

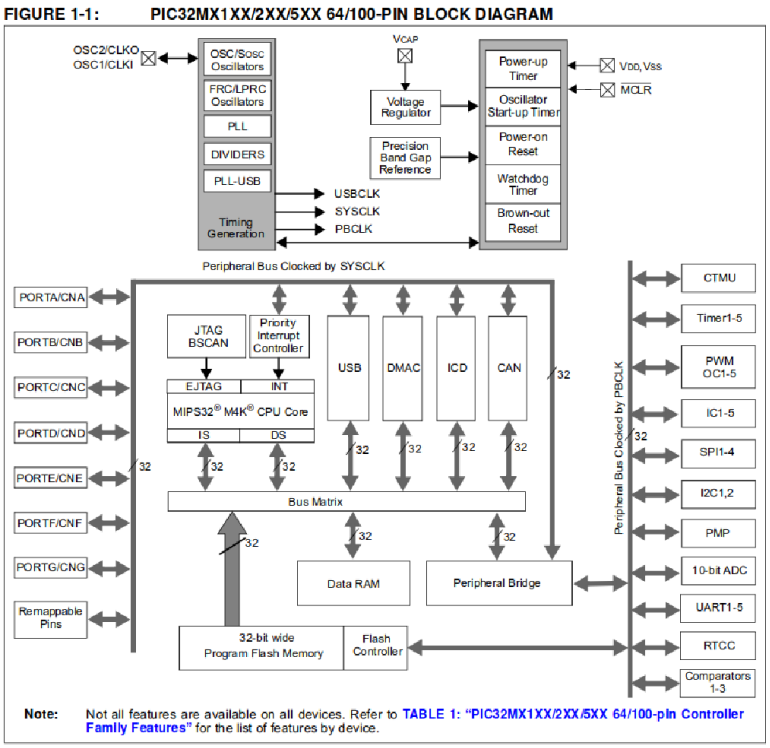


FIGURE 3.1 – Architecture du PIC32MX570F512L

3.1.2 Timers

Le Timer a tout simplement pour but de permettre d'espacer des opérations d'un certain nombre de millisecondes dans le temps. Voici son schéma électronique :

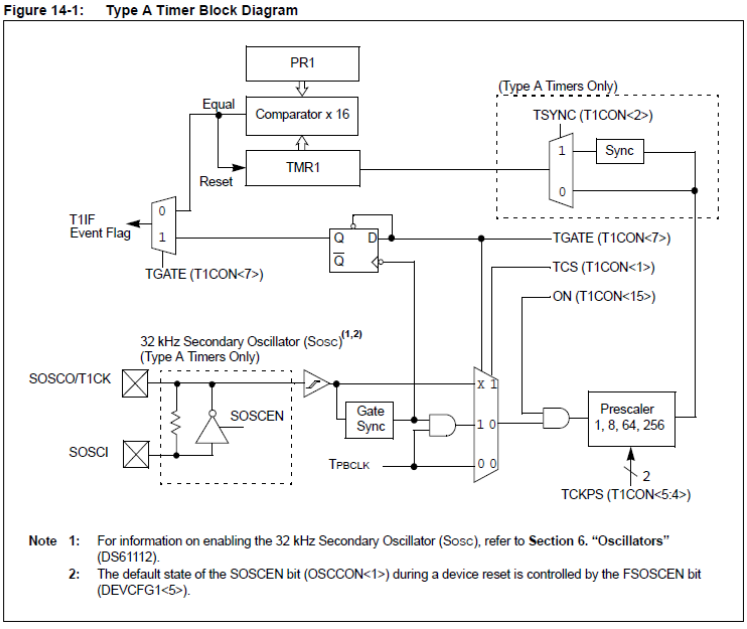


FIGURE 3.2 – Architecture du TIMER

Pour pouvoir utiliser un Timer nous avons à définir la fréquence d'horloge du microprocesseur. L'horloge de notre PIC32MX570F512L est à 40 MHz, ce qui signifie que le microprocesseur exécute une instruction environ toutes les 25ns.

La valeur du registre TMR1 va alors s'incrémenter à chaque instruction du microprocesseur. Cette valeur est récupérée à l'aide d'un comparateur (voir schéma) qui la compare avec PR1. Lorsque le comparateur remarque que les valeurs des registres TMR1 et PR1 sont identiques, celui-ci réinitialise la valeur du registre TMR1 et déclenche le passage du drapeau de 0 à 1. Le temps que prend TMR1 pour s'incrémenter jusqu'à la valeur du PR1 est notre Timer.

A noter que pour influencer le temps du Timer il faut definir ensuite le mode de comptage (d'incrémentation) que l'on souhaite utiliser, donc un prescaler (qui peut être 1, 8, 64 ou 256). La valeur du prescaler est définie en fixant la valeur du registre T1CON (voir schéma précèdent).

Ceci est illustré à l'aide de ce graphique :

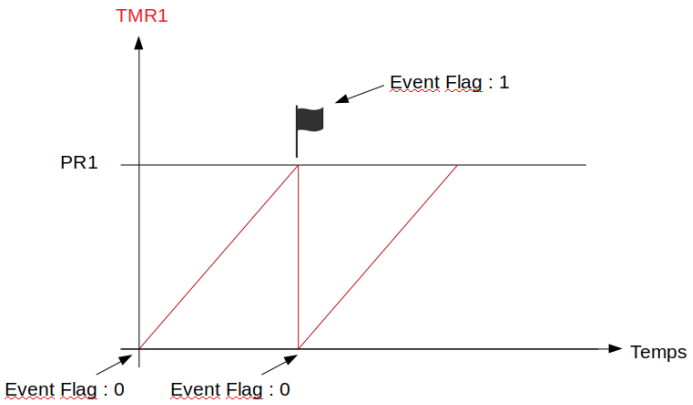


FIGURE 3.3 – TMR1 en fonction du temps

Celui-ci va influencer le temps maximum de comptage du microprocesseur. Le comparateur est codé en 16 bits et peut donc comparer jusqu'à la valeur maximale de 0xFFFF = 65535.
Sur ce graphique ci-dessous on peut voir les différents temps selon le prescaler choisi.

Prescaler	Temps maximal de comptage = 65535 * Δt
1	1,63 ms
8	13 ms
64	104 ms
256	419 ms

FIGURE 3.4 – Temps en fonction du prescaler

3.1.3 Interuption

Qu'est-ce qu'une interruption ?

Pour comprendre les interruptions on va tout d'abord faire quelques rappels sur le fonctionnement d'un microcontrôleur. Nous avons vu précédemment que les instructions d'un microcontrôleur sont exécutées séquentiellement.
En termes de programmation cela veut dire que le déroulement des instructions du programme est toujours le même on parle de ''programmation séquentielle"
Or, un microcontrôleur doit être capable de réagir à d'autres informations en dehors des lignes de code, par exemple recevoir une action d'un autre composant ou faire une action précise dès qu'une entrée de microcontrôleur change d'état.
Périodiquement on regarde l'état de l'entrée. Si entre 2 périodes ou 2 tests le programme détecte un changement d'entrée, l'action nécessaire est exécutée. Pour faciliter la compréhension de ce fonctionnement nous allons donner un exemple. Supposons qu'un client attende son colis, il vérifie toutes les 15 minutes si le livreur est arrivé. (L'état de l'entrée :oui ou non le colis est arrivé et les 15 minutes correspondent à la période entre les 2 tests.
Or il y a une probabilité que le livreur arrive entre les 15 minutes et dans ce cas-là que le client perde le livreur. Pour résoudre ce problème, il suffit que le livreur utilise la sonnette pour prévenir le client de son arrivée. Quand le client entend la sonnette il prend son colis et il revient au point de départ pour attendre son nouveau colis.

Qu'est-ce qu'un bus de communication ?

Un bus informatique est un dispositif de transmission de données partagé entre plusieurs composants d'un système numérique. Il y a 2 types de transmission.
Première transmission est de type parallèle ; sur le circuit les rails sont compacts et disposés en parallèle. Quelques exemples de ce type de bus sont ISA(Industry Standard Achitecture.), PCI(Periphiral Component Interconnect) et AGP(Advanced Graphic Port.).
La deuxième transmission en série est moins efficace car elle coûte plus chère et sa dimension est plus grande. . Des exemples de bus série sont les suivants : USB ,PCI(periphiral Component Interconnect.).

Fonctionnement d'un bus de communication :
Un bus de communication est un canal de transmistion, il reçoit donc un signal. Il existe deux mécanismes de synchronisation de signaux différents :
-protocole synchrone : il est prévu un signal de synchronisation (horloge) qui permet de gérer la synchronisation des communications.
-protocole asynchrone : l'ensemble de la communication est géré par le protocole lui-même par l'échange de messages.

Configurer l'interruption sur un PIC32MX

Une fonction d'interruption est très différente d'une fonction ordinaire. En effet, si celle-ci est bien configurée, elle s'exécute sans avoir été explicitement appelée par le programme principal.
Nous disposons par exemple comme fonction d'interruption de "__ISR()" signifiant "Interrupt Service Routine". Voici le corps de cette fonction :

```
void __ISR(_INT_VECTOR_NUMBER, IPLx[SRS|SOFT|AUTO]) isrName(void)
{
   /* Clear the cause of the interrupt (if required) */
   /* Clear the interrupt flag */
   /* ISR-specific processing */
}
```

Nous allons en premier lieu nous intéresser aux types des variables en entrée de cette fonction.
D'après la page des interruptions de Mircrochip, le type _INT_VECTOR_NUMBER correspond à l'identifiant du vecteur que l'on souhaite utiliser pour l'interruption.
On peut trouver dans la documentation du microprocesseur les différentes sources d'interruption avec leurs identifiants.

TABLE 7-1: INTERRUPT IRQ, VECTOR AND BIT LOCATION

Interrupt Source(1)	IRQ Number	Vector Number	Interrupt Bit Location			
			Flag	Enable	Priority	Sub-Priority
Highest Natural Order Priority						
CT – Core Timer Interrupt	0	0	IFS0<0>	IEC0<0>	IPC0<4:2>	IPC0<1:0>
CS0 – Core Software Interrupt 0	1	1	IFS0<1>	IEC0<1>	IPC0<12:10>	IPC0<9:8>
CS1 – Core Software Interrupt 1	2	2	IFS0<2>	IEC0<2>	IPC0<20:18>	IPC0<17:16>
INT0 – External Interrupt 0	3	3	IFS0<3>	IEC0<3>	IPC0<28:26>	IPC0<25:24>
T1 – Timer1	4	4	IFS0<4>	IEC0<4>	IPC1<4:2>	IPC1<1:0>
IC1 – Input Capture 1	5	5	IFS0<5>	IEC0<5>	IPC1<12:10>	IPC1<9:8>
OC1 – Output Compare 1	6	6	IFS0<6>	IEC0<6>	IPC1<20:18>	IPC1<17:16>
INT1 – External Interrupt 1	7	7	IFS0<7>	IEC0<7>	IPC1<28:26>	IPC1<25:24>
T2 – Timer2	8	8	IFS0<8>	IEC0<8>	IPC2<4:2>	IPC2<1:0>
IC2 – Input Capture 2	9	9	IFS0<9>	IEC0<9>	IPC2<12:10>	IPC2<9:8>
OC2 – Output Compare 2	10	10	IFS0<10>	IEC0<10>	IPC2<20:18>	IPC2<17:16>

FIGURE 3.5 – Les différentes sources d'interruption

On retrouve ces indentifiants du header à la position :
/xc32/<version>/pic32mx/include/proc/p32mx795f512l.h

p32mx795f512l.h

```
40654
40655  /* Vector Numbers */
40656  #define _CORE_TIMER_VECTOR              0
40657  #define _CORE_SOFTWARE_0_VECTOR         1
40658  #define _CORE_SOFTWARE_1_VECTOR         2
40659  #define _EXTERNAL_0_VECTOR              3
40660  #define _TIMER_1_VECTOR                 4
40661  #define _INPUT_CAPTURE_1_VECTOR         5
40662  #define _OUTPUT_COMPARE_1_VECTOR        6
40663  #define _EXTERNAL_1_VECTOR              7
40664  #define _TIMER_2_VECTOR                 8
40665  #define _INPUT_CAPTURE_2_VECTOR         9
40666  #define _OUTPUT_COMPARE_2_VECTOR        10
```

FIGURE 3.6 – Les différentes sources d'interruption dans .h

Par exemple pour une interruption de Timer nous utilisons le vecteur avec l'identifiant 4, ou encore pour une interruption avec un bouton poussoir on utilise l'identifiant 33.

Le type IPLx[SRS | SOFT | AUTO] sert à définir le niveau de priorité de l'interruption entre 0 et 7, permettant ainsi de pouvoir utiliser plusieurs interruptions.

3.1.4 I/O Ports

Les échanges d'informations avec les périphériques du microprocesseur et leur contrôle se fait par le biais des entrées-sorties du microprocesseur, appelées I/O ports. En figure 1 se trouve la représentation générale d'une structure de port partagé.

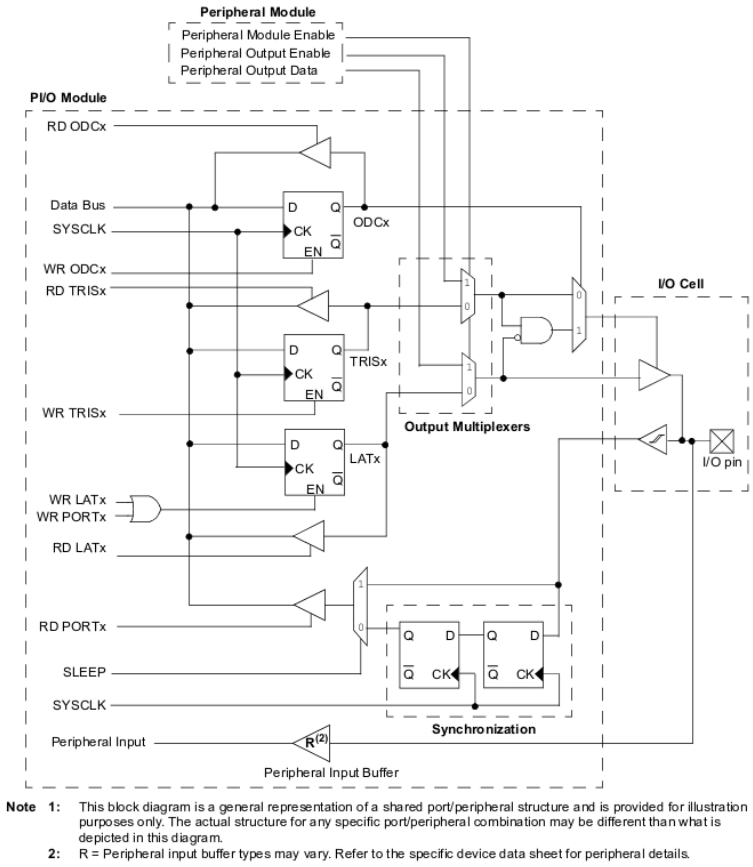


FIGURE 3.7 – Architecture I/O ports

3.2 Manipulations

3.2.1 Clignotement des LEDs

Notre première manipulation consistait à faire clignoter des LEDs. Pour cela nous avons modifié alternativement le port LATA en 0x000F puis 0x00F0 ce qui allume d'abord les LEDs de droite puis les éteints et allume les LEDs de gauche. On observe ainsi un clignotement de nos LEDs.

TRISA = 0xC600; //On met les LEDs en sortie en donnant cette valeur au port TRISA
while(1) //Le programme est une boucle sans fin
(
LATA = 0x000F;
LATA = 0x00F0;
)

3.2.2 Capteur de température

Sur la carte PIC32 se trouve un capteur de température. Nous avons testé son fonctionnement à l'aide d'un programme récupérant les données analogiques du capteur et les convertissant en données numériques. Ensuite la température est affichée sur l'écran LCD de la carte. Ainsi en posant un doigt sur le capteur, on observait un changement de température par rapport à celle précédemment affichée qui était celle de la pièce.

3.2.3 Delay

Nous avons créé une fonction permettant de faire un delay du nombre de millisecondes voulu. Ainsi en appelant la fonction, on donne un certain nombre de millisecondes. Le programme lance alors un timer d'une milliseconde autant de fois que nécessaire pour atteindre le temps voulu.

```
void delay_ms( unsigned t)
{
    T1CON = 0x8000;     // enable Timer1, source PBCLK, 1:1 prescaler
    while( t--)
    {
        TMR1 = 0;
        while( TMR1 < SYSCLK/1000); // wait 1ms
    }
    // disable Timer1
    T1CONCLR = 0x8000;
}
```

FIGURE 3.8 – Fonction Delay

3.2.4 Projet de jeux sur le microcontrôleur

Nous avons eu l'idée de coder un petit jeu avec nos connaissances sur le sujet, donc sur MPLAB en C embarqué. Nous avons voulu intégrer l'écran LCD ainsi que les boutons poussoirs afin d'en faire un jeu jouable par tous.

Voici donc quelques explications sur le code que nous avons écrit sur MPLAB.

Premièrement nous avons repris la procédure de délai (décrite dans la partie Delay).

Ensuite nous avons divisé le jeu en plusieurs parties : nous avons fait la partie algorithmique, puis géré l'interface graphique avec l'écran LCD, ajouté les boutons poussoirs , et enfin nous avons implémenté les interruptions pour améliorer la jouabilité.

Pour la partie algorithmique, nous avons décidé de se réunir tous ensemble pour faire le pseudo code avant de le traduire en C. Notre jeu fonctionne avec la gestion d'un personnage dans un tableau, qui se déplace en même temps jusqu'à collision entre le personnage et un mur (représenté par une croix) . Nous avons aussi décidé d'implémenter un système de score qui s'affiche à l'écran quand le jeu est perdu.

Nous disposions d'un écran 16x2 sur la carte que nous avons utilisé pour afficher notre jeu. Sur cet écran apparaissaient donc le personnage et les obstacles. Nous avons aussi ajouté des messages pour l'utilisateur, un écran d'accueil ainsi qu'un écran de fin de jeu avec affichage du score.
Ci-dessous l'affichage de l'accueil :

```
const char Accueil[] = "Press S4 to play";
 puts_lcd( (char*) &Accueil, sizeof(Accueil)-1 );
```

FIGURE 3.9 – Affichage Accueil

Celui du score :

```
const char fin_jeu[]="GAME OVER";
const char score_fin[]="Score :";
puts_lcd( (char*) &fin_jeu, sizeof(fin_jeu)-1 );
delay_ms(500);
line_2();
puts_lcd( (char*) &score_fin, sizeof(score_fin)-1 );
char buffer[3];
itoa(buffer,score,10);
 puts_lcd( (char*) &buffer, sizeof(buffer)-1 );
```

FIGURE 3.10 – Affichage Score

On remarque qu'il a fallu 3 lignes de code pour afficher le score, en effet le score étant un entier, il est plus difficile de l'afficher en langage C (plus de précisions dans la partie « IO Ports/ Difficultés rencontrées »).

Le personnage peut se déplacer verticalement avec les boutons poussoirs, le bouton S3 pour monter et le S4 pour descendre, ce qui permet au joueur d'éviter les obstacles avançant vers lui en haut ou en bas. Nous avions d'abord opté pour un test régulier sur la pression des boutons par l'utilisateur, mais nous nous sommes vite aperçus que ca n'était pas la meilleure manière de gérer ces boutons. En effet, il fallait appuyer au bon moment pour que le test détecte la poussée, ce qui résultait à devoir appuyer longtemps (car il est impossible de savoir quand le programme fait le test).

Nous avons donc choisi d'utiliser les interruptions, en changeant le système de déplacement vertical du personnage. Les interruptions ont été une très nette amélioration de ce petit jeu : nous sommes passés d'un appui testé presque aléatoirement qui forçait l'appui long à une version du jeu où l'appui pendant une très courte durée du bouton entrainait un déplacement vertical instantané ! Le code ci-dessous est un exemple pour le bouton S3 (pour descendre) lors du front descendant (lors de l'appui du bouton et non le relâchement), y étant la hauteur du personnage.

```
if (lectstatd & 0x40)
    {// cas appui S3 prioritaire
        lectportd = PORTD & 0x40; // lecture portD 0 si 0; 40 si 1
        if ((etatS3==1) && (lectportd == 0) ) // frontdescendant ? //
        {
                                    y=1;
                                    etatS3=0;  // front descendant on change
```

FIGURE 3.11 – Interruption bouton S3

Finalement, ce mini jeu nous a appris à manipuler un langage de programmation différent, et un peu plus en autonomie ainsi que différents aspects d'un microcontrôleur : Boutons, Ecran LCD, Interruptions...

Difficultés rencontrées

Lors de ce projet, certaines difficultés ont été rencontrées par notre groupe, et nous avons donc dû nous adapter et nous former pour pouvoir continuer la progression de notre travail.
Tout d'abord, la documentation sur le PIC32 que nous avions à disposition était quelque peu ardue. En effet, elle est écrite en anglais, avec un vocabulaire technique que nous ne connaissions pas, et est assez conséquente. Un certain travail a donc dû être fourni par le groupe pour traduire et comprendre les informations nécessaires à la réalisation du projet. De plus, certains membres du groupe n'étaient à l'origine pas familiers avec l'utilisation des bases binaire et hexadécimale, et ont donc dû travailler davantage pour les maîtriser.
Une autre difficulté est venue du fait que personne ne connaissait le langage C, dont nous avons pourtant eu besoin pour coder ce projet. Il a donc fallu apprendre un nouveau langage de programmation, ce qui prend du temps. L'utilisation du langage C a également fait apparaître certaines difficultés, notamment la gestion de pointeurs pour utiliser des chaînes de caractères, ou encore le fait que les messages d'erreur fournissaient peu d'informations sur l'origine des problèmes lorsque le code ne compilait pas.
Enfin, la situation inattendue du confinement nous a amené à devoir changer d'organisation. En effet, à cause de l'éloignement, il a fallu utiliser différents outils informatiques pour communiquer malgré parfois des problèmes de connexion. De plus, tout le monde n'a pas pu avoir accès à une carte, et donc les tests étaient parfois plus compliqués à réaliser.

Conclusion et perspectives

Ce projet nous a donc permis de découvrir ce qu'est un microprocesseur ainsi que son fonctionnement à travers la carte PIC32, et plus généralement d'acquérir des notions de base dans le domaine de l'électronique. Nous avons ainsi découvert différents périphériques, et appris à coder dans un nouveau langage et à utiliser notamment des délais ou des thermomètres. Pour mettre en application ces nouvelles compétences de façon ludique, nous avons donc créé un jeu. Pour aller plus loin dans ce projet et améliorer ce jeu, nous pourrions encore améliorer notre utilisation des délais. Le tableau permettant de générer des obstacles pourrait aussi être rendu aléatoire, afin de ne plus avoir un jeu répétitif. L'esthétique de l'affichage pourrait également être davantage travaillée.

Bibliographie

[1] LIEN INTERNET, https://microchipdeveloper.com/32bit:mx-arch-exceptions-usage (Valide à la date du 21/05/2020)

[2] LIEN INTERNET, http://ww1.microchip.com/downloads/en/DeviceDoc/61132B.pdf (Valide à la date du 25/05/2020)

[3] LIEN INTERNET, https://www.microchip.com/DevelopmentTools/ProductDetails/DM240001-2#additional-summary (Valide à la date du 04/06/2020)

Table des figures