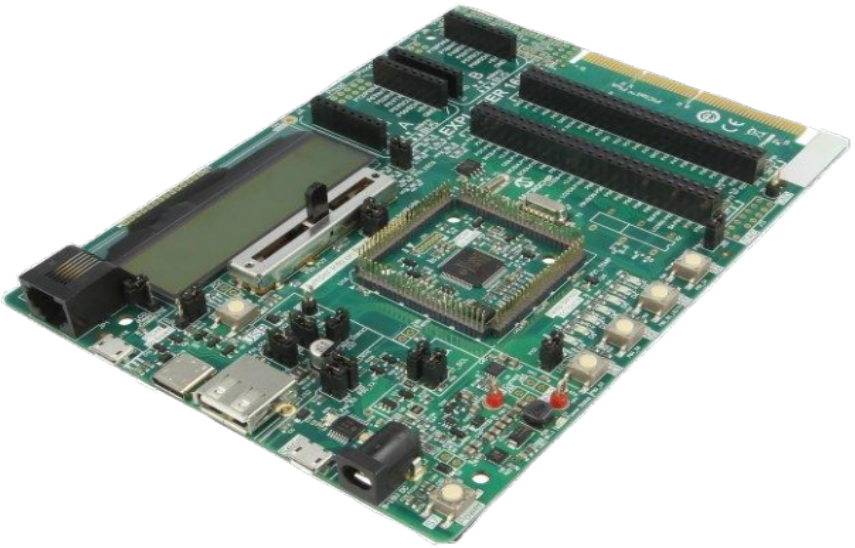
Projet P6
Informatique embarquée : initiation au microprocesseur et à la programmation
Serrure électronique

Julian FERNANDEZ
Kévin GATEL
Damien GENS
Tom SIMON
Timothé VERSTRAETE

À l'attention de Richard Grisel



Résumé

Date de remise du rapport : 15 juin 2020

Numéro de projet : 13

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

Type de projet : Simulation/expérimental

Objectif du projet : Ce projet avait pour but de nous initier au microprocesseur en travaillant avec un modèle et de comprendre comment il fonctionne. Par ailleurs, le projet nous a initiés au langage C que nous avons utilisé pour programmer sur une carte à l'aide de Mplab. Enfin, nous devions mettre en application tout ce que l'on avait vu.

Mots clés du projet : informatique, microprocesseur, programmation

Table des matières

1 Introduction

Ce projet a pour but de nous initier aux microprocesseurs et à leur programmation à travers d'une application. Le microprocesseur est un élément central dans un appareil comme un ordinateur, en effet son rôle est d'exécuter des instructions et de traiter les données des programmes. La miniaturisation des processeurs depuis le début des années 80 a permis de révolutionner le monde de l'informatique et notamment au sujet des problématiques liées à l'informatique embarquée. En effet, cette miniaturisation a permis une augmentation de la vitesse de fonctionnement des processeurs grâce à une réduction de la distance entre les composants qui constituent le processeur car ces derniers se trouvent sur un seul et même circuit imprimé, tout en améliorant grandement la transportabilité du processeur grâce à sa petite taille.

Aujourd'hui, on entend souvent parler des microprocesseurs, en effet ces derniers sont utilisés dans de nombreux domaines liés à l'électronique. Cependant on peut s'interroger sur le fonctionnement général d'un microprocesseur ainsi que l'utilisation concrète de ce dernier. Afin de comprendre le fonctionnement d'un microprocesseur et de se familiariser avec l'utilisation de ce dernier, nous utiliserons une carte de développement PIC32 conçue par l'entreprise américaine Microchip ainsi que l'environnement de développement MPLAB.

Dans un premier temps, nous résumerons les premières séances avec Mr Grisel qui ont permis d'analyser le fonctionnement et les différentes fonctionnalités d'un microprocesseur. Puis, dans un second temps, nous reviendrons sur la réalisation de notre projet codé, c'est-à-dire sur la conception d'une serrure électrique. Enfin nous reviendrons sur le ressenti et l'organisation du groupe vis-à-vis du projet.

Voici un schéma qui décrit la carte :

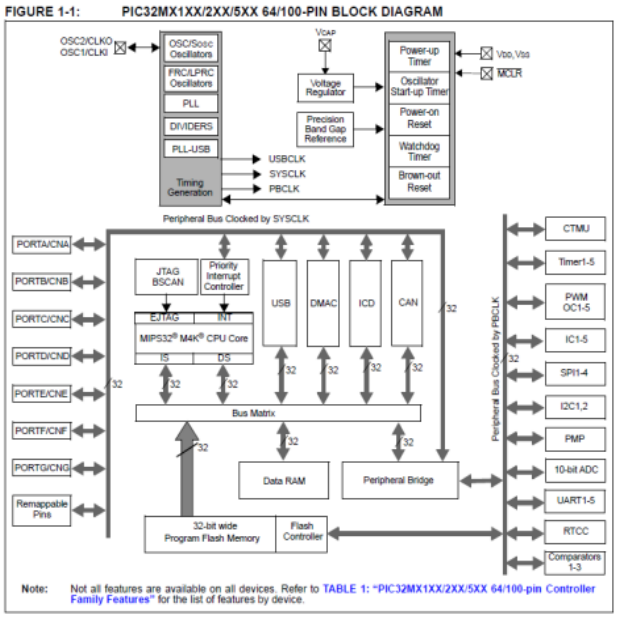


FIGURE 1 – Schéma descriptif de la carte

Et voici une photo légendée de la carte où l'on repère les éléments qui la composent :

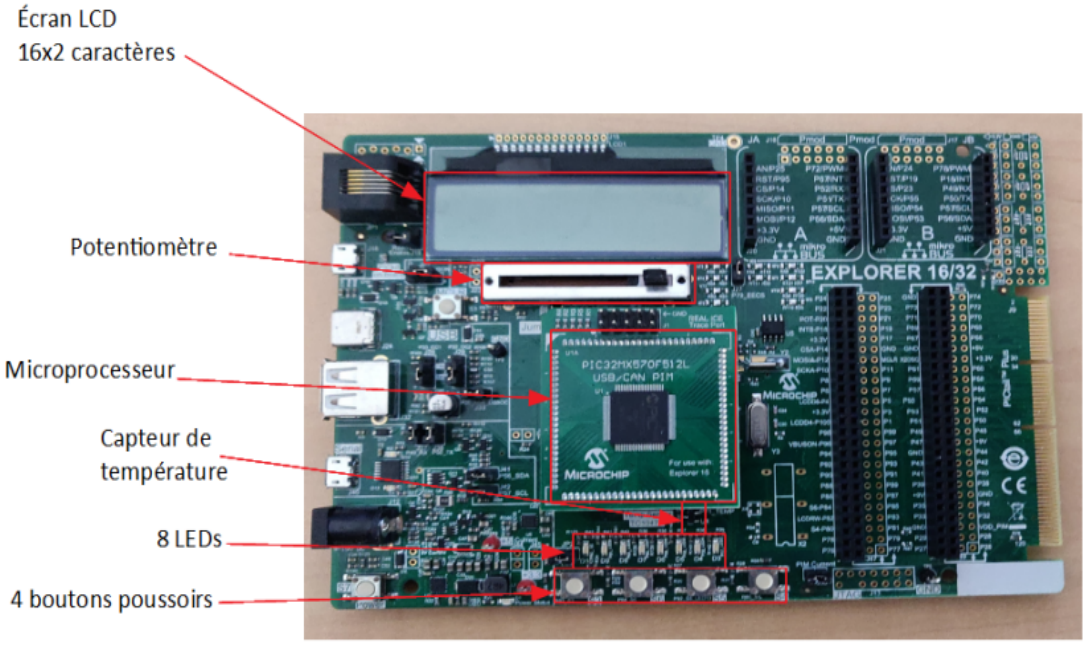


FIGURE 2 – Photo légendée de la carte

Au cours de ce projet, nous avons utilisé le logiciel MPLAB dont voici un aperçu de l'interface au démarrage :

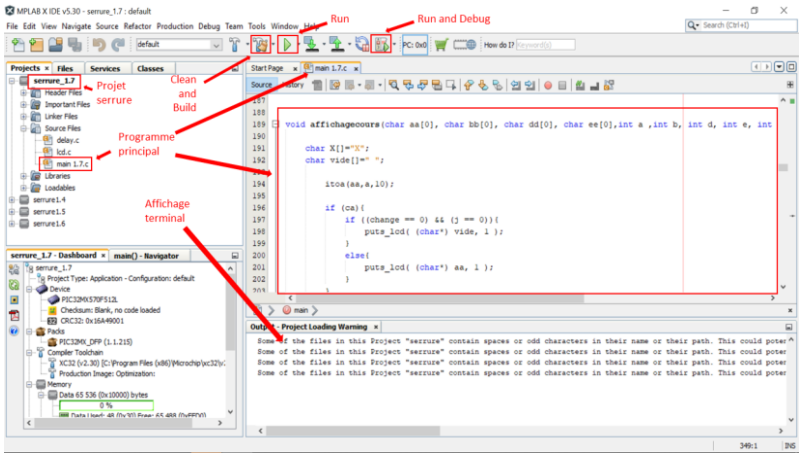


FIGURE 3 – Interface MPLAB au démarrage du logiciel

2 Résumé des premières séances

2.1 Introduction à l'informatique embarquée

Au cours de la première séance nous avons introduit le microprocesseur PIC32Mx570F512L et ses fonctionnalités. Pour cela nous avons vu de manière générale le cycle de développement et d'exécution d'un programme sur le microprocesseur.

Dans un premier temps nous avons évoqué les utilisations du langage C, un langage plus ou moins familier pour les membres du groupe. Ce langage nous servira à coder nos programmes lors des manipulations sur le logiciel MPLAB. Dans un second temps, nous avons abordé le langage assembleur .s qui interagit avec le processeur (l'opération d'assemblage traduit chaque instruction du programme source en une instruction machine).

Par la suite nous nous sommes penchés sur les différents périphériques impliqués, notamment les GPIO (fils transportant les informations binaires), les convertisseurs analogiques/numériques CAN et numériques/analogiques CNA ainsi que les timers orchestrant les interruptions lors de l'exécution du code afin d'exécuter d'autres instructions prioritaires.

2.2 Exemple de Leds avec MPLAB

La deuxième séance avait pour but de nous familiariser avec le logiciel MPLAB. Nous avons alors repris certains concepts du langage C permettant de manipuler : les registres, les adresses, ouvertures fermetures de ports.

Lors de cette séance, nous avons réalisé notre première manipulation sur machine. Celle-ci avait pour but de faire clignoter les LEDs présentes sur la carte de développement contenant le microprocesseur. Pour cela, nous avons réalisé un programme en C : celui-ci changeait alternativement les valeurs du port LATA (port correspondant aux Latch des sorties sur la carte)

```
1 TRISA = 0xC600 ; //On met les LEDs en sortie en donnant cette
      valeur au port TRISA
2 while(1)
3 {
4 LATA = 0x000F ; //Le programme est une boucle sans finLATA =
     0x00F0 ;
5           }
```

Remarque : Il est important de noter que nous réalisons l'exécution des programmes en simulation puis sur la carte pour s'assurer que le programme ne risque pas d'endommager la carte.

Nous avons utiliser les fonctionnalités de l'environnement MPLAB :

— Watches : permet de regarder les Adresses et Valeurs rapidement

— Logic Analyser : permet d'obtenir la vitesse d'exécution d'une instruction.

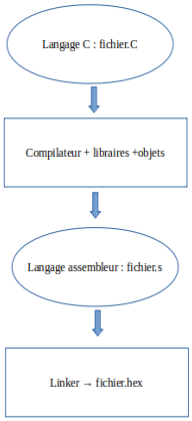


FIGURE 4 – Schéma représentant le fonctionnement des langages C et s

2.3 Mise en place d'un timer

Cette séance avait pour but d'étudier le fonctionnement d'un Timer de type A. Voici le schéma électrique de ce composant :

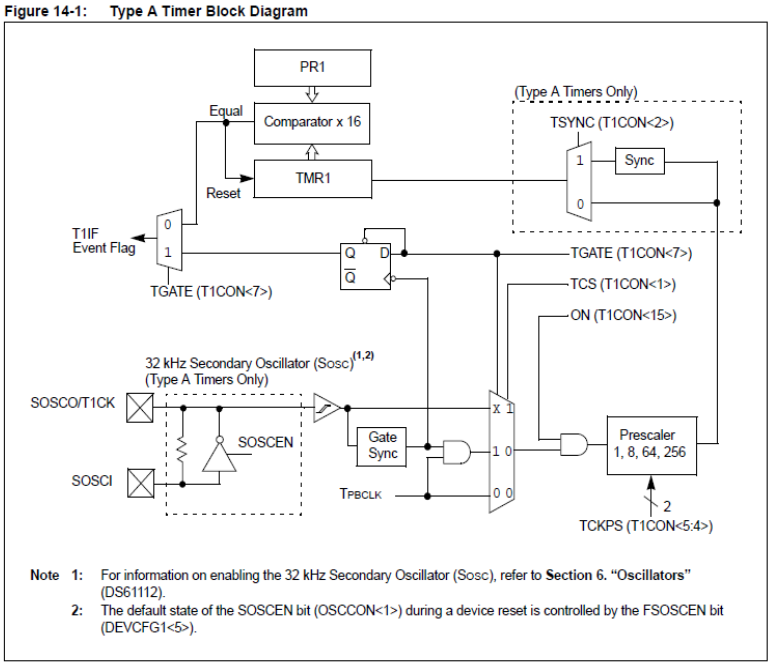


FIGURE 5 – Schéma électrique d'un Timer de type A

Processus d'utilisation du Timer Type A :

Il faut premièrement définir la fréquence d'horloge du microprocesseur. On la définit à 40 MHz, ce qui signifie que le microprocesseur exécute une instruction toutes

les $1/40x10^6$ s ce qui correspond à 25 ns. Il faut ensuite le mode de comptage que l'on souhaite utiliser, soit un prescaler (qui peut être 1, 8, 64 ou 256). La valeur du prescaler est définie en fixant la valeur du registre T1CON. Celui va influencer le temps maximum de comptage du microprocesseur.

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

TABLE 1 – Tableau sur les valeurs du prescaler

On doit ensuite fixer la valeur dans le registre PR1 pour paramétrer le comparateur. La valeur fixée dans PR1 correspond au temps d'attente avant de faire passer la drapeau de 0 à 1 et ainsi exécuter l'instruction.

Remarque : il ne faut pas oublier de remettre le drapeau à 0 après l'exécution de l'instruction afin de permettre l'exécution de l'instruction suivante.

Le comparateur est codé en 16 bits et peut donc comparer jusqu'à la valeur maximale de 0xFFFF = 65535.

La valeur du registre TMR1 est incrémentée à chaque coup d'horloge. Le registre TMR1 permet donc de compter le temps de manière régulière.

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.

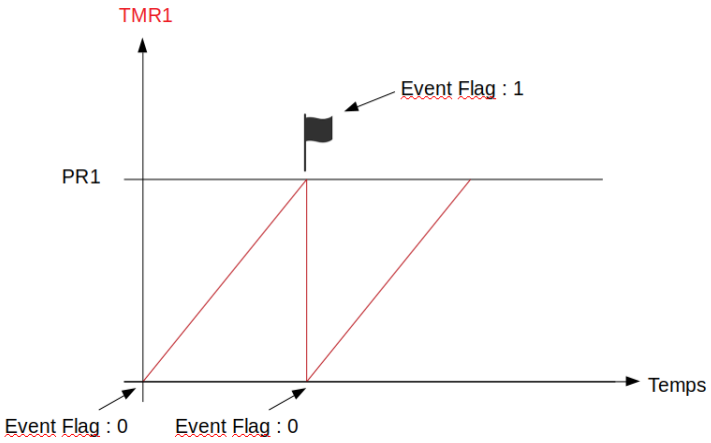


FIGURE 6 – Schéma représentant l'event flag

Manipulations :

Dans un premier temps, nous avons réalisé un programme ayant pour objectif de visualiser le temps d'exécution de boucles d'attentes avec une première fois une boucle while puis une boucle for.

```
1  #define DELAY 65535 // Valeur max sur 16 bits
2  main ()
3  {
4  // init PORTALATA=0x0000;
5  //Latch A reset
6  TRISA = 0x0000; //Port A output
7  // _LATA0= 1; //Led on on startup
8  LATA=0x000F;
9  //init TIMER
10 TMR1 = 0; // clear the timer
11 T1CON = 0x0030; // prescaler by 256
12 T1CON=0X8030; // timer enabled prescaler par 256
13 while (1)
14 {
15 LATA=0x00AA; // valeur initiale du port A (LATA)
16 TMR1=0;while (TMR1 < DELAY) // boucle d'attente DELAY*(40Mh
       /256)
17 {
18 }
19 TMR1 = 0; // remise a zero du registre TIMER TMR1
20 LATA=0x0055; // changement de la valeur du port A (LATA)
21 for (j=0; j <2225187; j++)
22 {
23 } // duree = 0,5s environ pour la boucle vide
24 }
25 } // end program
```

Observations : Lors de l'exécution de la boucle vide *while (TMR1 < DELAY)* on obtient un temps d'exécution de ... ce qui est similaire à la valeur théorique 419ms pour un prescaler = 256.
La boucle vide

```
for (j=0; j <2225187; j++)
```

nous permet d'observer le comptage de 0,5s environ.

Fonctionnalités :

— Stopwatch : renvoie temps d'exécution d'un certain nombre de lignes de code.
— Run to cursor : nous permet d'exécuter un nombre spécifique de ligne.

2.4 Code du timer pour 1 milliseconde

Cette séance s'est déroulée en autonomie sans Mr Grisel.
Dans un deuxième temps, nous avons conçu une fonction ayant pour but de compter un certain nombre de fois 1ms (ce nombre souhaité est passé en entrée de la fonction) dont voici le code :

```
1  #define SYSCLK 40000000 //Cette constante correspond au 40MHz
       de la fréquence de l horloge
```

```
2  void Delay_ms (unsigned t)
3  {
4  TMR1 = 0;
5  T1CON = 0x8000; // start timer 1
6  while(t--)
7  { TMR1 = 0;
8          while (TMR1 < SYSCLK/1000); // Délai 1ms
9  }
10 //T1CONCLR = 0x8000;
11 }
```

2.5 Application du timer aux Leds

Nous avons ensuite utilisé cette fonction pour créer un programme qui éteint et allume les LEDs de la carte avec un intervalle de temps de 0,5s. Pour compter ce délai de 0,5s on utilise la fonction que nous venons de créer en lui passant en entrée la valeur 500. Voici le détail du code de ce programme :

```
1  // Remarque : l'IT timer est de niveau 4 (100) par defaut
       donc si
2  // IPLSR = 100, l'interruption ne prend pas, il faut qu'elles
       soient a 011
3  main ()
4  {
5  // init PORTA
6  LATA=0x0000; //Latch A reset
7  TRISA = 0x0000; //Port A output
8  // _LATA0= 1; //Led on on startup
9  LATA=0x000F;
10 //init TIMER
11 TMR1 = 0; // clear the timer
12 T1CON = 0x0030; // prescaler by 256
13 T1CON=0X8030; // timer enabled prescaler par 256
14 while (1)
15 {
16          LATA=0x00AA; // valeur initiale du port A (LATA)
17          TMR1=0;
18          while (TMR1 < DELAY) // boucle d'attente DELAY*(40Mh
                /256)
19          {
20          }
21          TMR1 = 0; // remise a zero du registre TIMER TMR1
22          LATA=0x0055; // changement de la valeur du port A (
                LATA)
23          Delay_ms(500);
24          // si boucle for
25          LATA=0x0022; // changement de la valeur du port A (
                LATA)
26          for (j=0; j <2225187; j++)
```

```
27          {
28          } // duree = 0,5s pour la boucle vide
29  }
30  } // end program
```

Lors de cette séance nous nous sommes aussi intéressés au fonctionnement des interruptions de timer.
Une fonction de gestion des interruptions est différente d'une fonction ordinaire en ce sens qu'elle gère l'enregistrement et la restauration du contexte pour s'assurer qu'au retour de l'interruption, le contexte du programme est maintenu.

Pour réaliser une interruption de timer, il faut alors utiliser la fonction __ISR permettant définir l'ISR signifiant « Interrupt Service Routine ». Voici le code général 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 */
}
```

Décrivons maintenant les différentes variables prises en entrée de cette fonction : La première variable _ INT_ VECTOR _ NUMBER correspond à l'identifiant du vecteur que l'on souhaite utiliser pour définir l'interruption de routine.

Dans la documentation, on peut retrouver une table de correspondance entre les différentes sources d'interruptions possibles et l'identifiant du vecteur correspondant.

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 7 – Extrait de la documentation sur les timers

Ici, on retrouve la ligne qui nous intéresse (encadrée en bleue) car c'est le Timer 1

qui est la source de l'interruption de la route. On repère que l'identifiant du vecteur correspondant est 4.

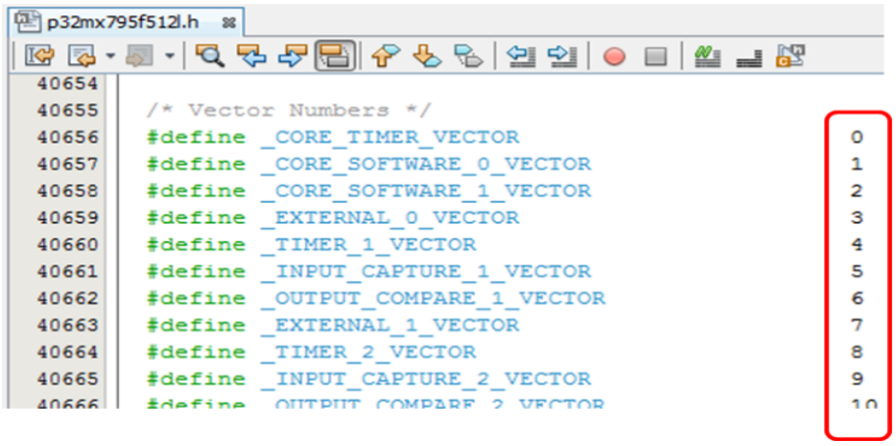


FIGURE 8 – Extrait du code sur les timers

Sur la page du site de Microchip correspondant à notre microprocesseur, on retrouve une correspondance entre les identifiants des vecteurs et le nom des variables à utiliser. Ici, on peut donc voir que notre première variable à passer en entrée de la fonction d'interruption est _ TIMER _ 1_VECTOR.

La deuxième variable de cette fonction est IPLx[SRS|SOFT|AUTO]. IPL signifie « Interruption Priority Level ». Ici, on choisit une priorité de niveau 4 (valeur qui va remplacer le x) car c'est la valeur qui est indiquée dans le tableau présenté ci dessus (case en rouge). Enfin, on doit préciser comment le contexte est préservé lors de l'interruption. Sans rentrer dans les détails, on choisit le paramètre AUTO pour que la sauvegarde du contexte (avant interruption) ainsi que sa restauration (après interruption) soient faites automatiquement.

Enfin on doit préciser isrName(void) qui correspond au nom (unique) que l'on donne au service d'interruption de routine.

3 Projet réalisé : serrure électronique

3.1 Code de la serrure

Nous avons décidé d'appliquer toutes les connaissances acquises sur le microprocesseur sur un exemple concret de programme. Nous avons ainsi créé un programme de serrure électronique, ce programme permet de rentrer un code à 4 chiffres et de vérifier si celui-ci est le même que celui sauvegardé dans le programme au préalable. Il existe plusieurs particularités à l'exécution du programme : le chiffre en cours d'écriture ainsi que les LEDS correspondantes clignotent, à la vérification du code, les LEDS clignotent différemment et le programme redémarre si le mot de passe est incorrect. Nous allons donc expliquer comment ont été codées plusieurs fonctionnalités.

3.2 La rentrée des chiffres à l'aide des boutons

Pour rentrer les chiffres pour le code nous utilisons 3 boutons : le premier sert à décrémenter la valeur, le deuxième l'incrémente et le dernier permet de passer au chiffre d'après ou de demander la validation du mot de passe quand tous les chiffres ont été saisis. Voici le code associé :

```
if ((PORTDbits.RD7==0) && (c!=9))
{
    c = c + 1;
}
else{
        if ((PORTDbits.RD7==0) && (c==9))
{
    c = 0;
}

}

if ((PORTDbits.RD6==0)&&(c!=0)) //Si le bouton poussoir S6 est à 1 (enfoncé)
{
    c = c - 1;
}
else{
    if ((PORTDbits.RD6==0)&&(c==0)) //Si le bouton poussoir S6 est à 1 (enfoncé)
    {
        c = 9;
    }
}


if (PORTDbits.RD13==0) // passe au chiffre suivant
{
   j = j + 1;
   c=0;
   delay_ms(500);
}
```

FIGURE 9 – Code associé aux boutons pour la serrure électronique

Lorsque le « PORTDbits » associé au bouton passe à 0, le bouton à été enfoncé, alors dans ce cas il faut modifier la valeur de c, qui sert de compteur, lorsque le premier ou deuxième bouton est enfoncé on peut alors augmenter la valeur de j, qui sert à connaître le nombre en cours de saisie, lorsque le dernier bouton est enfoncé.

Il existe des cas particuliers pour les deux premiers boutons. En effet si la valeur de c est à 9 et que l'on augmente celle-ci de 1, il faut la passer à 0, de même lorsque la valeur est à 0 et que l'on diminue celle-ci de 1, c doit passer à 9.

3.3 Mot de passe en écriture

Pour conserver les valeurs du mot de passe en cours d'écriture, nous concevons les chiffres indépendamment les uns des autres. En effet, à chaque boucle du programme, nous conservons le chiffre en cours d'écriture dans une valeur. Par exemple pour le premier chiffre :

```
if (j==0) //sauvergarde les valeur du compteur dans les int correspondant
{
    if ((c == 1)&&(ca == 0)){
        ca = 1;
        c = 0;
    }
    a=c;


}
```

FIGURE 10 – Code pour la conservation du premier chiffre pour la serrure électronique

Ici, le j est à 0, nous sommes donc en écriture du premier chiffre, la valeur de c est donc sauvegardée dans la variable a. Au final, nous avons 4 variables pour conserver les 4 chiffres. Pour afficher le mot de passe en cours d'écriture, il suffit donc d'afficher les chiffres un à un dans le bon ordre.

Afin de vérifier que le mot de passe est le bon, il suffit de l'écrire sous forme décimale et de la comparer avec la constante définie au préalable :

```
mdp_rentre = 1000*a+100*b+10*d+e; // ecrit le mdp sous forme de int
if ((mdp_rentre==mdp_int)&&(ca)&&(cb)&&(cd)&&(ce)) //test le mdp
```

FIGURE 11 – Code pour comparer le mot de passe rentré et le bon mot de passe

3.4 Utilisation des LEDS

Afin de faire clignoter les LEDS, nous avons créé une procédure qui prend j et une variable change qui passe de 0 à 1 à chaque boucle de programme :

```
if (j == 0){
    if (change == 0 )
    {
        LATA=0x00C0;
    }
    else
    {
        LATA=0x0000;
    }
    }
```

FIGURE 12 – Code pour le clignotement des LEDS

On vérifie l'étape en cours à l'aide de j pour qu'en fonction de la valeur de change, on allume ou pas la LEDS, ici pour le premier chiffre soit on allume les 2 premières LEDS (0x00C0) soit on n'en allume aucune (0x0000). Les valeurs de LEDS changent en fonction du chiffre en cours d'écriture afin de toujours faire clignoter au bon endroit.

Les LEDS clignotent aussi lorsque le code est correct ou non, nous avons créé 2 procédures exécutant ces clignotements sous un principe de boucle et de changement

des valeurs associées aux LEDS. Si le code est correct, les LEDS clignotent 2 à 2 en alternant, sinon elles clignotent toutes ensembles.

4 Organisation du travail

Le projet s'est très bien déroulé malgré les bouleversements liés à la crise du COVID-19. Lors de la première phase du projet, c'est-à-dire les premières séances destinées aux explications des différents points importants à connaître sur les microprocesseurs grâce aux informations apportées par M. Grisel, une bonne ambiance et une bonne dynamique de groupe se sont installées avec beaucoup d'interactions entre les membres du groupe et M. Grisel pour permettre la compréhension des différents points par tous les membres du groupe. La majorité de cette période s'est déroulée dans les locaux de l'INSA, ce qui a facilité la compréhension des différents éléments avant un éloignement imposé par la situation sanitaire mondiale.

Par la suite, nous avons dû nous séparer des locaux de l'INSA cependant, M. Grisel nous a accordés sa confiance et nous a laissés emprunter une carte de développement chacun pour pouvoir comprendre les fonctionnalités, effectuer des tests et réaliser notre projet codé depuis notre lieu de confinement. Nous le remercions chaleureusement pour sa confiance et sa bienveillance tout au long de cette période difficile pour chacun. Lors de cette deuxième phase du projet, nous avons donc dû réaliser un projet codé autour du microprocesseur et des différentes fonctionnalités proposées par la carte de développement. Nous avons eu carte blanche pour choisir l'application de notre choix, ce qui a été apprécié par l'ensemble des membres du groupe.

Malgré le contexte, nous avons su garder une réelle organisation avec une réunion hebdomadaire avec M. Grisel pour faire le point sur l'avancée de notre projet, accompagnée de réunions sur Discord avec les membres du groupe dédiés au développement informatique en lui-même. Nous avons apprécié les précieux conseils de M. Grisel lors de l'élaboration de notre serrure électronique. Enfin, nous nous sommes répartis le travail afin de réaliser les différents formats de rendu attendus.

5 Conclusion

Pour conclure, ce projet est un véritable avantage dans notre formation d'ingénieur. En effet, ce dernier nous aura permis de découvrir de nouveaux concepts liés à l'informatique ce qui constitue un apport culturel pour les personnes souhaitant poursuivre leur formation dans une voie différente de celle de l'informatique et un véritable apport de connaissances utiles pour la suite d'une formation dans la voie de l'informatique pour ceux qui souhaitent poursuivre dans cette voie.

Cette initiation aux microprocesseurs et à leur programmation nous aura permis de mettre en œuvre des compétences en codage acquises depuis le début de notre formation d'ingénieur au service d'un projet destiné à utiliser du matériel électronique. Ce projet se place parfaitement dans la continuité des différents projets d'informatique que nous avons réalisés au cours de notre formation. En effet, celui-ci nous a permis de découvrir un nouveau langage, le langage C, tout en appliquant des méthodes de travail de projet informatique déjà rencontrées. Ceci constitue alors un avantage

pour une poursuite dans l'informatique car c'est un langage que nous utiliserons très prochainement dans la suite de notre cursus.

Enfin, le microprocesseur étant un composant d'une importance capitale pour le bon fonctionnement d'une machine informatique, cette initiation à travers ce projet nous aura permis d'en apprendre un peu plus sur les enjeux de l'amélioration continue de ce composant au cœur des problématiques d'optimisation de la vitesse de calcul et de gestion des ressources informatiques dans la course à l'amélioration de l'efficacité des algorithmes.

Table des figures

Liste des tableaux