Aix*Marseille université

Faculté des Sciences
Licence 2 SPI

Informatique Industrielle

Travaux Pratiques

Julien Marot, Youssef Trardi

Formation au développement d'applications en Assembleur sur un microcontrôleur Microhip PIC 18F4520 en utilisant l'environnement MPLAB IDE.

Contact :

youssefrardi@gmail.com

julien.marot@fresnel.fr

1. Avant de commencer

Ces séances de travaux pratiques (TP) sont destinées à illustrer les notions qui vous ont été présentées en cours. Pour cela, vous aurez à concevoir et à tester un certain nombre de programmes soit en langage assembleur soit en langage C.

Pour chaque tâche que vous aurez à réaliser, nous vous invitons à enregistrer votre programme sous un nom particulier pour garder une sauvegarde de travail. Pour faciliter le débogage et assurer la maintenance et la portabilité de vos codes, il est primordial que vous commenciez par rédiger un algorigramme, que vous commentiez clairement votre programme et que vous pensiez à le rendre le plus clair possible (*ex* : en choisissant des noms appropriés pour les labels).

Concernant l'évaluation de votre travail, nous vous demanderons de nous présenter le programme correspondant à chaque tâche que vous aurez à programmer sur le simulateur ou sur le kit de démonstration (*cf.* plus bas). Nous corrigerons également vos codes en portant une attention particulière à la lisibilité (pensez donc aux commentaires !). Enfin, vous devrez également nous rendre les algorigrammes qui correspondent aux codes.

2. Le Matériel

2.1. Le Kit PICDEM 2 Plus et PIC 18F4520

Le kit PICDEM 2 Plus est un kit simple qui permet de mettre en évidence les capacités de différents micro-contrôleurs du fabricant Microchip à travers divers périphériques et accessoires reproduits sur le schéma de la figure 1.

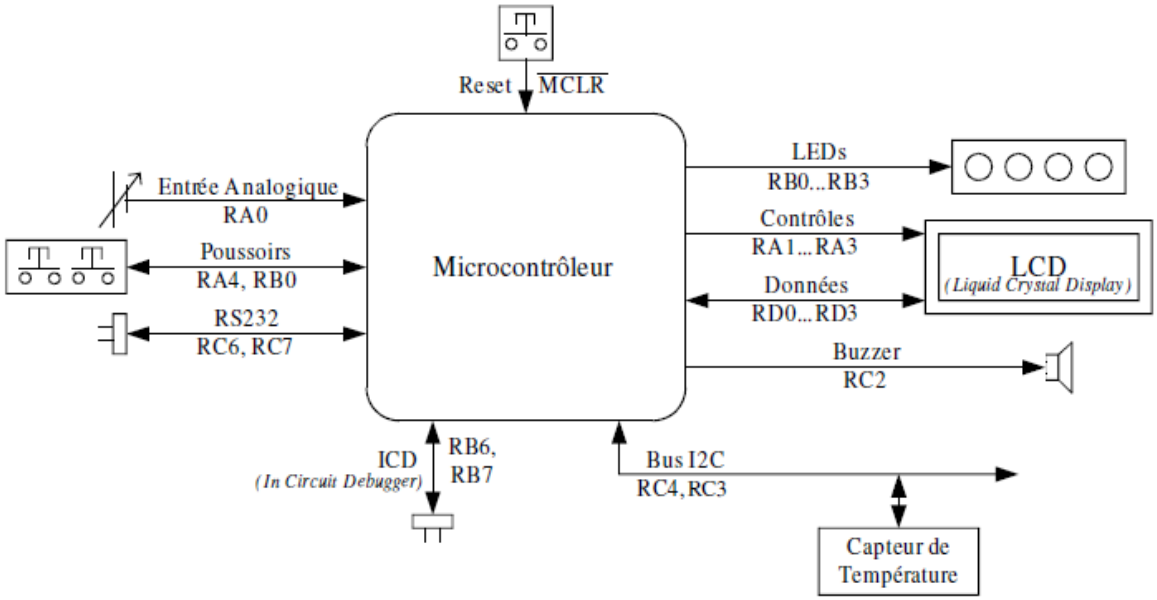


Figure 1 : Le kit PICDEM 2 PLUS, Périphériques et Accessoires

Le micro-contrôleur utilisé est le PIC18F4520 de Microchip. C'est un micro-contrôleur 8 bits disposant d'une mémoire de programme de 32 Ko, de 1536 octets de RAM (Random Access Memory), d'une EEPROM (Electrically Erasable Programmable Read Only Memory) de 256 octets et de 36 entrées/sorties.

Une part importante de la difficulté pour programmer un micro-contrôleur réside dans l'acquisition d'informations. Pour acquérir cette compétence, nous vous invitons à vous référer fréquemment à la documentation technique du micro-contrôleur, ainsi qu'à la documentation technique de la carte d'évaluation, cf. les documentations électroniques.

2.2. MPLAB Integrated Development Environment

MPLAB IDE est un environnement gratuit de programmation pour les micro-contrôleurs de la famille Microchip. L'environnement intègre un éditeur d'assembleur, un débogueur complet ainsi que tous les outils permettant

de gérer les programmateurs de composants. La possibilité d'intégrer un compilateur C dans l'environnement de développement MPLAB s'inscrit directement dans la politique du service qui est d'utiliser des langages de programmation évolués dans les développements en électronique.

2.3. Le programmateur ICD 3

Il permet de transférer le programme directement de MPLAB dans le micro-contrôleur via le port USB d'un PC et le port mini-ethernet de la plaquette de test PICDEM2PLUS.

3. Gestion de vos fichiers

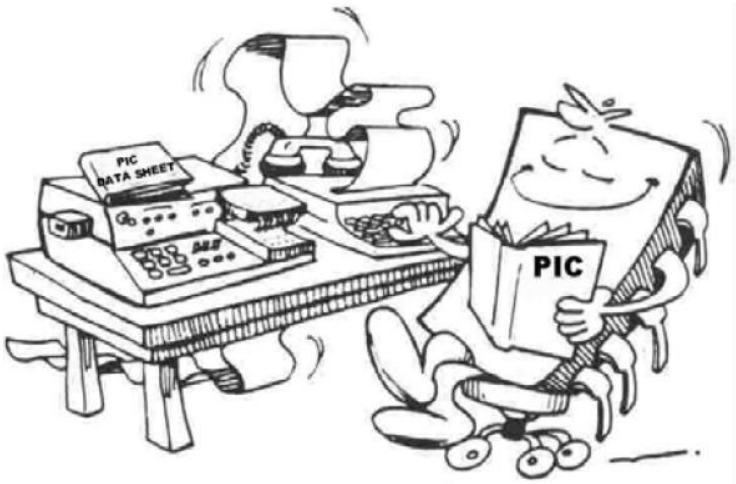
Pour vous logger sur l'ordinateur, utilisez votre login mot de passe habituel 'ENT'.
Vous pouvez le noter ci-dessous. Attention Windows fait la différence entre les majuscules et les minuscules.

Nom d'utilisateur :	.....
Mot de passe :	.....

Vous pouvez rédiger vos programmes soit sur le réseau, soit sur une clé USB. Dans tous les cas, il vous est fortement conseillé de bien ranger vos documents tout au long des séances. pour chacun des TP, faîtes un répertoire associé ex: tp3 pour le tp n°3. Puis pour chaque question si c'est nécessaire faites des sous-répertoires.

Les documentations techniques nécessaires au TP sont disponibles à l'adresse:

https://sites.google.com/site/infoindusamu/



4. TP n°1 : Initiation à MPLAB IDE

Le but de ce TP est de se familiariser avec la suite logicielle « MPLAB Integrated Development Environment » (v8.92) de Microchip afin de pouvoir utiliser les fonctions de bases de cet outil développement à savoir : créer un projet, éditer et compiler un programme en assembleur, simuler son fonctionnement et enfin programmer le microcontrôleur (µC). Les différentes fonctions sont expliquées à travers deux programmes qui vous sont donnés.

4.1. Programme « Interrupteur »

Le programme « Interrupteur » boucle à l'infini sur les tâches ci-dessous afin de reproduire le fonctionnement d'un interrupteur.

- *Tâche 1 :* lire l'état du bouton poussoir S2, broche RA4
- *Tâche 2 :* si le bouton est appuyé, on allume la diode électro-luminescente (LED) L1, broche RB0, sinon on l'éteint.

On notera que les boutons poussoirs sont branchés au microcontrôleur en logique négative. L'état « bouton appuyé » correspond donc à un 0 logique. Les LED sont câblées en logique positive, elles s'allument pour un 1 logique. Le fonctionnement demandé peut être reproduit par l'algorigramme ci-dessous.

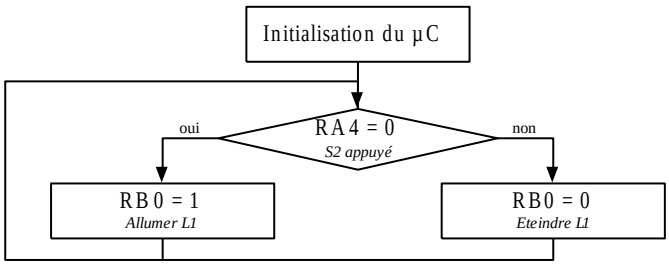


Figure 1: Algorigramme du programme « Interrupteur »

L'algorigramme peut se traduire en assembleur de la manière suivante :

```
        LIST P=18F4520
        #include <P18F4520.inc>
        #include <CONFIG.inc>
```

Définition du µC utilisé, des constantes spécifiques au µC et des paramètres de configuration encodés lors de la programmation.

```
        org h'0000'
        goto init
```

Adresse de départ après un reset du µC.
On place l'instruction goto init *à l'adresse (0000)*

```
init    clrf PORTB
        movlw b'00000000'
        movwf TRISB
        clrf PORTA
        movlw b'00010000'
        movwf TRISA
```

Initialisation :
Remise à zéro des bascules D des différents ports utilisés et configuration des entrées/sorties.

```
boucle  btfsc PORTA,4
        goto eteindre
        bsf PORTB,0
        goto boucle
eteindre bcf PORTB,0
        goto boucle

        END
```

Boucle infini :
Allume ou éteint la LED en fonction de l'état du bouton poussoir.

Tout au long des travaux pratiques, nous vous demandons d'établir un glossaire des différentes instructions rencontrées avec leur explication en français. Ce glossaire vous sera très utile pour les différents programmes que vous serez amenés à réaliser.

4.2. Édition et Compilation du programme

Cette partie permet d'apprendre à créer un projet MPLAB, saisir un programme en assembleur et le compiler.

4.2.1. Démarrage

- Avant de commencer, créer le répertoire *tp1* puis dans ce répertoire créer à nouveau un répertoire *interrupteur*.
- Copier dans ce dernier répertoire les fichiers *P18F4520.inc* et *CONFIG.inc*
- Lancer la suite logicielle MPLAB IDE à partir de l'icône qui se trouve sur votre bureau.

4.2.2. Création d'un nouveau projet avec « Project Wizard »

- Cliquer sur *Project >> Project Wizard …*
- Cliquer sur *Suivant* > puis sélectionner PIC18F4520 dans le menu déroulant *Device*.
- Les fichiers asmwin, asmlib, asmlink doivent ensuite être trouvés (cliquer sur Browse à chaque fois) dans le dossier MPASM SUITE en remontant dans l'arborescence des dossiers.
- Cliquer sur *Suivant* > puis sélectionner « *Microchip MPASM Toolsuite* » et « *MPASM Assembler* » afin de pouvoir programmer en Assembleur.
- Cliquer sur *Suivant* > puis dans le champ « *Project Name* » saisir le nom du projet (ex : interrupteur) et dans le champ « *Project Directory* » aller chercher à l'aide du bouton « *Browse* », le dossier créé au début : *tp1/interrupteur*.
- Cliquer sur Suivant > et ajouter les fichiers *P18F4520.inc* et *CONFIG.inc* au projet à l'aide du bouton « *Add* » >>
- Cliquer sur *Suivant* > puis sur *Terminer*.

4.2.3. Edition du programme

Créer un nouveau fichier dans le projet, *File >> Add New Files to Project…*

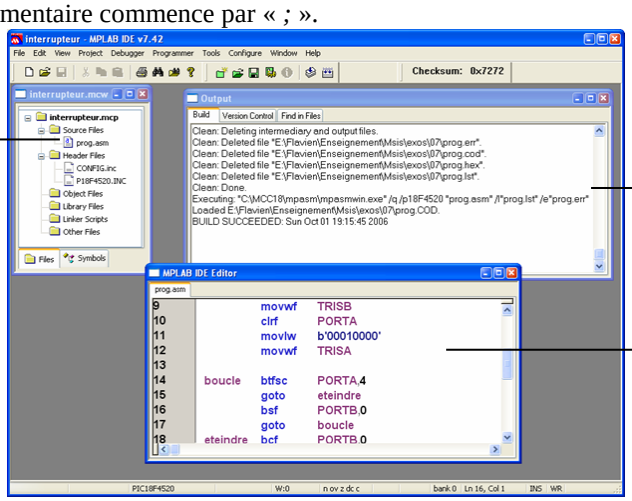
L'enregistrer dans le répertoire projet avec l'extension *asm* (ex : *prog.asm*)

MPLAB Editor se lance

Saisir le programme « Interrupteur » et enregistrer.

Penser à ajouter des commentaires au programme pour montrer que vous avez bien compris les différentes instructions. Une ligne de commentaire commence par « *;* ».

Workspace
Architecture et contenu du projet



Output
Interface MPLAB Développeur

Editor
Edition des fichiers de programme, de configuration

Figure 2: MPLAB IDE, Mode Edition

4.2.4. Compilation

- Cliquer sur *Project >> Build All …* ou sur l'icône de la barre de menu ou encore *Ctrl+F10* pour compiler le projet complet. Erreurs et warnings sont signalés dans la fenêtre *Output* ainsi que le résultat de la compilation *BUILD SUCCEEDED* ou *BUILD FAILED.*
  En double-cliquant sur une erreur ou un warning dans la fenêtre Output, vous serez amener directement à la ligne concernée dans votre programme.

4.3. Simulation

Avant d'envoyer un programme au µC, vous pouvez tester son fonctionnement dans le simulateur (Debugger) MPLAB SIM. Pour lancer le simulateur, cliquer sur *Debugger >> Select Tool >> MPLAB SIM*. Une nouvelle barre d'outils est ajoutée ainsi qu'un onglet « *MPLAB SIM* » dans la fenêtre Output. Le simulateur fonctionne selon trois modes :

- le mode *Step By Step* qui permet de faire une exécution pas à pas du programme (vous êtes alors l'horloge du µC).
- le mode *Animate* qui exécute automatiquement le programme mais à une vitesse réduite pour que l'on puisse suivre l'exécution.
- le mode *Run* où le programme est exécuté automatiquement à la vitesse du PC.

Dans les deux premiers modes, nous obtenons des résultats au fur et à mesure de la simulation alors que dans le dernier, les résultats ne sont visibles que lorsque la simulation est stoppée. Dans ce paragraphe, nous nous intéressons aux modes *Animate* et *Step By Step* car ils nous permettent de voir qu'elle est l'influence des entrées sur les sorties.

Le réglage de la fréquence de l'oscillateur utilisé se fait dans *Debugger >> Settings …*. Dans notre cas c'est 4 Mhz.

4.3.1. Entrées : Stimulus

Comme son nom l'indique cette fonctionnalité permet de prévoir les événements d'entrées de manières synchrones (les dates sont connues avant la simulation) et asynchrones (les événements sont déclenchés durant la simulation par l'utilisateur). Les différents événements sont enregistrés dans une table (Workbook). Pour en créer une, cliquer sur *Debugger >> Stimulus >> New Workbook*.

Dans le cadre du projet « interrupteur » on souhaite créer les événements suivants :

- à t = 0, initialisation de RA4 à 1 (bouton S2 relâché),
- à t = 10 cycles, on appuie sur S2 (RA4 = 0),
- à t = 25 cycles, on relâche le bouton (RA4 = 1).
- on veut également pouvoir inverser l'état de RA4 quand on le souhaite.

L'onglet « *Pin / Register Actions* » de la fenêtre « *Stimulus* » permet la saisie des événements en programmant leurs dates d'arrivée (événements synchrones).

Démarche de création d'événements synchrones :

- Ajouter une colonne RA4 au tableau d'événement (bouton « *Clic here to Add Signals* »)
- Rechercher RA4 dans la liste « *Available Signals* » et l'ajouter à la liste « *Selected Signals* »
- Remplir la table d'événement (Time, RA4) avec (0,1) (10,0) (25,1)
- Choisir *cyc* (cycle) comme base de temps dans menu « *Time Units* »
- Enregistrer la table (bouton « *Save Workbook* »)

L'onglet « *Asynch* » de la fenêtre « *Stimulus* » permet de pré-programmer des boutons (colonne « *Fire* ») à la génération d'événement (événements asynchrones).

Démarche de création d'événement asynchrone :

- Saisir RA4 dans la colonne « *Pin/SFR* » de la première ligne
- Le menu de la colonne « *Action* » propose cinq types d'action :
  - *Set High* : RA4 = 1
  - *Set Low* : RA4 = 0
  - *Toggle* : RA4 = $\overline{RA4}$
  - *Pulse High* : RA4 = 
  - *Pulse Low* : RA4 = 

  } la largeur de l'impulsion est réglée dans les colonnes « *Width* » et « *Units* ».
- Choisir « *Toggle* » pour inverser RA4
- Enregistrer la table (bouton « *Save Workbook* »)

- Appliquer l'échéancier d'événement (bouton « *Apply* »)

Après chaque changement dans l'échéancier d'événement, cliquer sur « Apply » afin que ces changements soient pris en compte par la simulation.

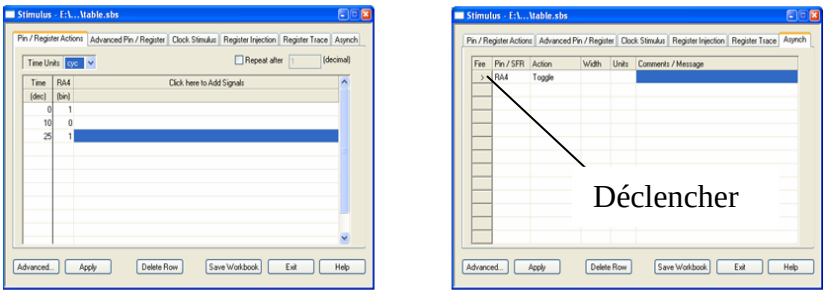


Figure 3: Stimuli synchrone et asynchrone

4.3.2. Sorties : Watch & Logic Analyzer

Afin de suivre et de visualiser les résultats de simulation, MPLAB SIM intègre divers outils dont :

- La fenêtre « *Watch* » (menu *View* >> *Watch*) qui affiche le contenu des différents registres du µC et des différentes variables du programme. La sélection et la validation des registres et variables à afficher se fait à travers les deux menus déroulants et les deux boutons « *Add SFR* » et « *Add Symbol* ». Un clic droit sur un des éléments observés permet l'accès à la boîte de dialogue « *Properties* » afin de régler le format d'affichage de l'élément (ex : Hex, Binary, Single Bit, ...) mais aussi de sauvegarder, exporter, … le tableau « *Watch* ». Il est aussi possible de faire glisser des éléments de la fenêtre « *Editor* » vers la fenêtre « *Watch* ».
- La fenêtre « *Logic Analyzer* » permet l'affichage des éléments logiques (1 bit) en fonction du « *Time Base* ». Le choix des éléments à afficher se fait à l'aide du bouton « *Channels* ».

Pour le projet « interrupteur », on souhaite afficher :

- les registres TRISA et TRISB sous forme hexadécimale
- les registres PORTA, PORTB et *W* sous forme binaire
- les bits RA4 et RB0

Configurer correctement les fenêtres « *Watch* » et « *Logic Analyzer* ».

4.3.3. Simulation du programme en mode Animate

Maintenant que les entrées et sorties de la simulation sont configurées, on peut lancer la simulation. Avant toutes choses, afin d'initialiser le µC, il est nécessaire d'effectuer un « *Reset* » en cliquant sur le bouton puis cliquer sur le bouton « *Animate* » et observer les fenêtres « *Watch* » et « *Logic Analyzer* ».

Arrêter l'animation à l'aide du bouton , faire un nouveau « *Reset* », relancer l'animation, ouvrir l'onglet « *Asynch* » de la fenêtre « *Stimulus* », agir sur RA4 et observer les résultats.

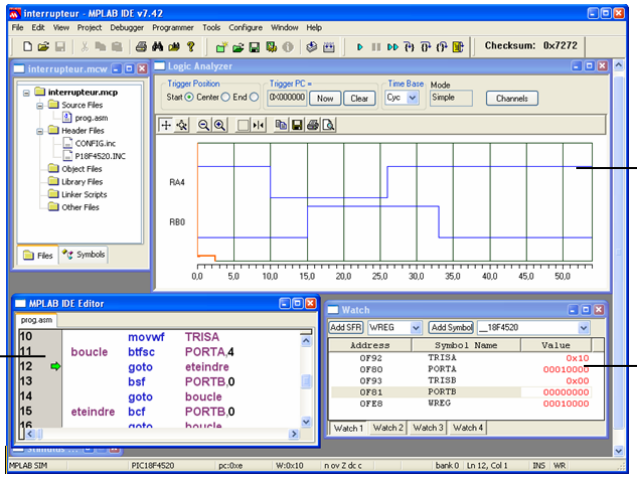


Figure 4: MPLAB IDE, Mode Simulation

Vous pouvez également faire avancer le programme pas à pas (mode « Step By Step ») avec le bouton « *Step Into* » . On notera que les boutons « *Step Over* » et « *Step Out* » permettent respectivement une exécution pas à pas avec une exécution automatique des sous-programmes et une exécution jusqu'à la fin des sous-programmes.

4.4. Programmation du microcontrôleur

Cette phase consiste à envoyer le code hexadécimal du programme dans le µC. Une fois le programme compilé avec succès, le code hexadécimal du programme se trouve dans le fichier avec l'extension *hex* (ex : *prog.hex*), voir le fichier avec la fenêtre « *Program Memory* » du menu « *View* ». Le code de trouve sous l'onglet « *Opcode Hex* ». Si on utilise le programmateur MPLAB ICD3 le programme est injecté dans le PIC qui se trouve sur la plaquette de test.

– Dans MPLAB, sous le menu *Programmer* > *Select Programmer* chosir « MPLAB ICD3 ». Un nouvel onglet est alors ajouté à la fenêtre « *Output* ».

– Envoyer votre programme, menu *Programmer* > *Program* ou en cliquant sur , le transfert est terminé lorsque la LED jaune *Activate* du programmateur est éteinte et que le message « *Programming/Verification completed successfully!* » apparaît dans la fenêtre *Output*

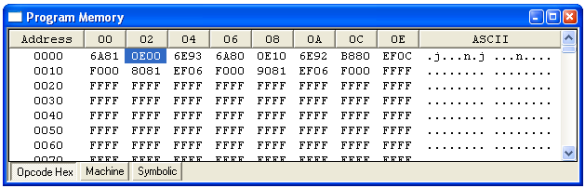


Figure 5: MPLAB IDE, Code hexadécimal du programme

4.5. Programme « Clignotant »

4.5.1. Analyse et compréhension du programme

Le programme ci-après fait clignoter la LED L1, broche RB0, à une fréquence d'environ 1Hz. L'état de la LED change donc toutes les 500ms.

```
        LIST P=18F4520
        #include <P18F4520.inc>
        #include <CONFIG.inc>
```

Définition du µC utilisé, des constantes spécifiques au µC et des paramètres de configuration encodés lors de la programmation.

```
;----- Déclaration de variables
        CBLOCK 0x00
           t1 : 1
           t2 : 1
        ENDC
```

Les variables t1 et t2 sont rangées dans la bank 0 de la ram (adresse 0x00) et ont pour taille un octet.

```
        org h'0000'
        goto init
```

Adresse de départ après un reset du µC.

```
;----- Initialisation
init    clrf    PORTB
        movlw   h'00'
        movwf   TRISB
```

Remise à zéro des bascules D du port B et définition du port B en sortie.

```
;----- Programme Principal
boucle  movlw   h'01'
        xorwf   PORTB
        call    tempo250ms
        call    tempo250ms
        goto    boucle
```

Boucle infini :
Allume et éteint la LED toutes les 500ms.

```
;----- Sous-programme
tempo250ms
        movlw   t10
        movwf   t1
comp1   dcfsnz  t1
        return
        movlw   t20
        movwf   t2
comp2   dcfsnz  t2
        goto    comp1
        goto    comp2

        END
```

Sous programme de temporisation **(à compléter)**
On réalise deux boucles imbriquées qui « occupe » le µC durant environ 250 ms.

Comprendre le programme « clignotant » et réaliser l'algorigramme associé.

Expliquer l'utilité de *xorwf* dans le programme principal (utiliser une table de vérité).

Calculer les valeurs d'initialisation t10 et t20 pour que le sous programme de temporisation s'exécute en 250 ms ± 1 ms. Le nombre de cycle requis par chaque instruction est donné dans le jeu d'instruction.

Créer un nouveau projet dans MPLAB et le sauver dans le répertoire *tp1/cligno* puis saisir le programme précédent en remplaçant t10 et t20 par leur valeur (pour informer le compilateur de la base dans laquelle la valeur est écrite on précède la valeur de la première lettre de la base, ainsi 192 en décimal s'écrit d'192', h'C0' en hexadécimal ou encore b'11000000' en binaire).

Compiler votre programme et passer à la partie simulation.

N'oubliez pas de copier les fichiers P18F4520.inc et CONFIG.inc dans le répertoire tp1/clignot

4.5.2. Simulation

Vous pouvez simuler le programme de la même manière que le programme « *interrupteur* » mais vous vous apercevrez très vite que c'est très long du fait de l'animation et de la temporisation. (Vous pouvez tout de même le faire pour bien comprendre mais en initialisant t10 et t20 à 3 et 5 par exemple.) Nous allons donc faire les simulations en mode *Run*, exécution automatique.

Pour notre programme, nous n'avons besoin d'aucun événement d'entrée mais si besoin avait été on aurait pu saisir un échéancier comme précédemment. En mode « *Run* » le programme s'exécute donc de façon automatique et ne s'arrête que lorsqu'il rencontre un point d'arrêt ou à la fin du programme. Il faut donc placer correctement les points d'arrêt afin de valider le fonctionnement de notre programme. Pour observer la durée de la temporisation il faut donc mettre un point d'arrêt sur la ligne d'appel du sous programme de temporisation et un autre sur la ligne suivante. Pour ajouter un point d'arrêt , double cliquer sur la ligne du programme où vous le souhaitez, ou alors utiliser le menu *Debugger >> Breakpoints …* disponible depuis la touche *F2*.

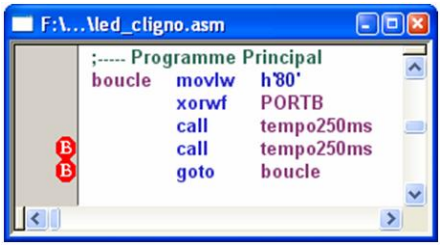


Figure 6: Placement des points d'arrêt

Nous allons suivre l'exécution du programme à l'aide de la fenêtre « *Simulator Trace* » disponible dans le menu « *View* ». Nous allons nous intéresser plus particulièrement à la colonne « *Time* ». Vous pouvez changer l'unité de temps de cette colonne par un clic droit menu « *Display Time* ». Lorsque vous affichez le temps en toutes autres unités que cycle, la valeur affichée dépend de la fréquence de l'oscillateur du µC.

- Lancer la simulation, bouton
- Observer la fenêtre « *Simulator Trace* » lorsque la simulation s'arrête au premier point d'arrêt et noter le temps d'appel de la temporisation (ex : t_i = 250303 cycles)
- Relancer la simulation, bouton
- Noter à nouveau le temps de la fenêtre « *Simulator Trace* » (ex : t_f = 500602 cycles)

Vous pouvez alors calculer la durée de la temporisation d = t_f – t_i = 250299 cycles soit 250,299 ms avec un

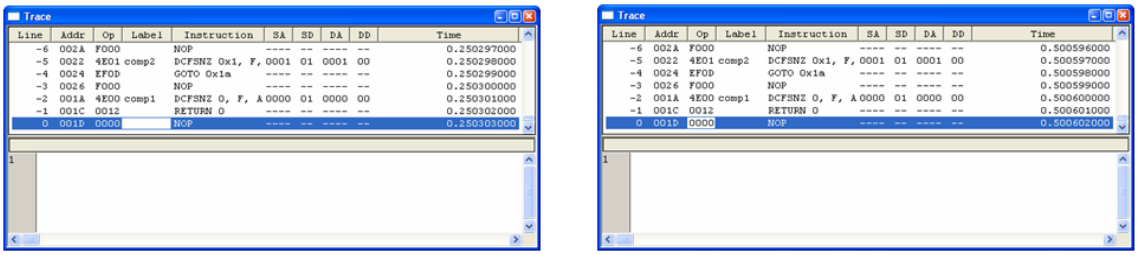


Figure 7: Simulator Trace aux différents points d'arrêt

oscillateur à 4 MHz. Utiliser cette méthode pour valider vos initialisations de t10 et t20.

Une fois les simulations terminées transférer le programme dans le microcontrôleur, et effectuez le test : la LED connectée au port RB0 clignote-t-elle ?

5. TP n°2 : Interruptions du TIMER 0

Dans ce TP, vous ferez clignoter une LED en utilisant un programme d'interruption, au lieu de la temporisation utilisée dans un TP précédent. Nous devons faire clignoter la LED L1, broche RB0, à une fréquence de 0.5Hz sans mobiliser le microprocesseur. Il n'est donc plus question d'utiliser une temporisation pour compter le temps (cf. TP n°1). Nous allons demander à un module externe : le TIMER0, de compter le temps et de signaler la fin du comptage au microprocesseur par une interruption dite interruption de débordement. La figure ci-dessous montre le signal de sortie du TIMER0 et les interruptions de débordement associées. Le TIMER0 compte jusqu'à la valeur *max* en *T* secondes. Toutes les *T* secondes, l'état de la led L1 est inversé.

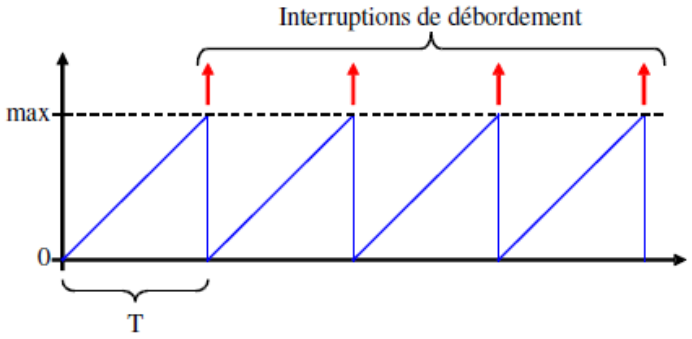


Figure 8: Interruption de débordement du TIMER0

Trouver, en utilisant la datasheet, l'expression de la valeur *max* et de la période de comptage *T* en fonction de la configuration du TIMER0. (*cf datasheet p123*)

Qu'elle est la configuration pour une période de comptage *T* d'environ 1s ?

Quels sont les bits à intialiser pour activer l'interruption du TIMER0 ?

Expliquer la configuration du registre T0CON.

L'énoncé demande de faire clignoter la LED à une fréquence de 0.5Hz donc une période de comptage de 1s. Avec la configuration choisi précédemment, on a une interruption toutes les (1+*t*)s, si on souhaite mesurer le temps précisément cette erreur de *t* seconde n'est pas négligeable.

Pour compenser cette erreur décalage, l'idée est d'initialiser les registres TMR0H TMR0L afin que le comptage ne commence pas à 0 mais à une valeur *min* permettant d'annuler *t* comme le montre la figure 9.

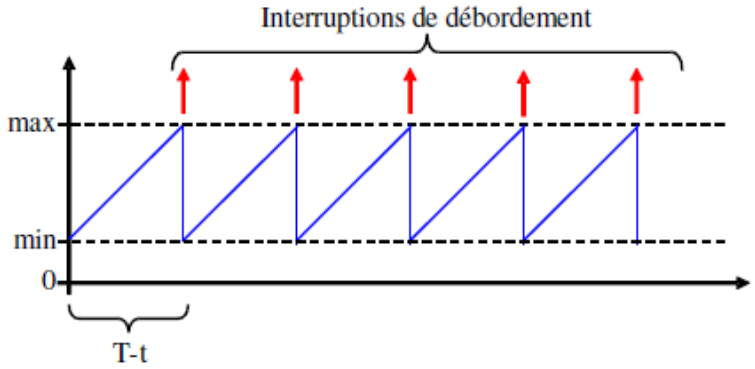


Figure 9: Interruption de débordement du TIMER0

Calculer la valeur *min* pour avoir une fréquence de comptage de 1s exactement.

Réaliser l'algorigramme du programme demandé, écrire et tester ce programme. Utilisez pour le test le logic analyser, en initialisant TMR0 à une valeur proche de la valeur max (par exemple 'FFFD'). Vous pourrez ainsi visualiser les changements d'état de la LED RB0.

6. TP n°3 : Mise en œuvre d'une PWM

Un transducteur piézo-électrique est un composant électronique constitué d'une lamelle de matériau piézo-électrique. Cette lamelle est déformée si une tension alternative est appliquée aux bornes du transducteur piézo-électrique.

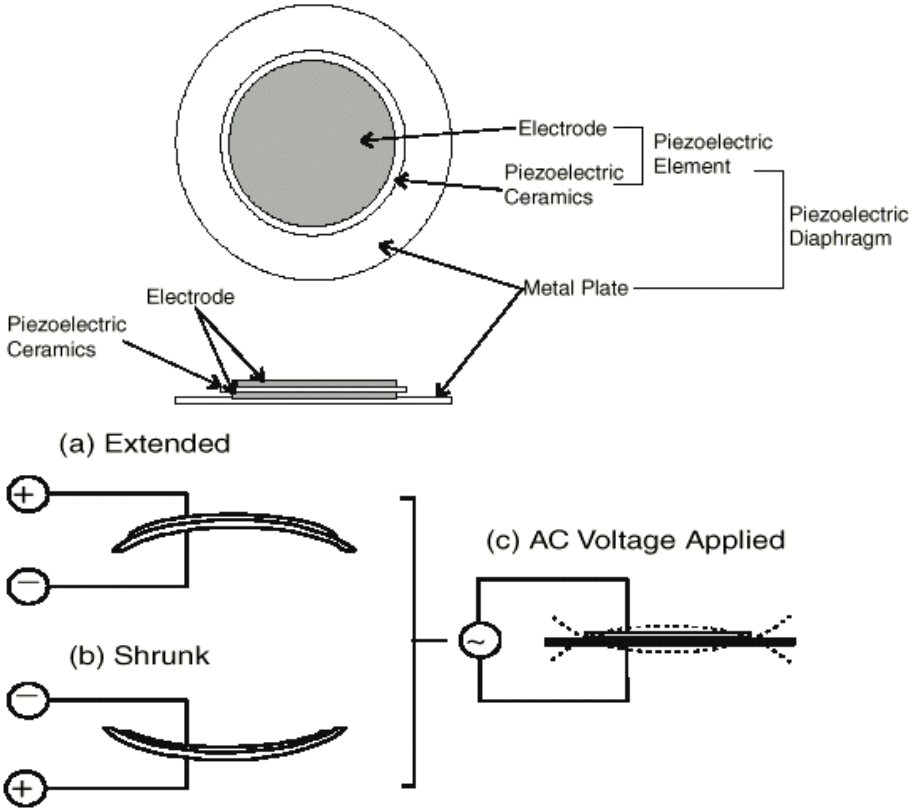


Figure 10: Le transducteur piézo-électrique

Le mouvement de la lamelle dans l'air génère un son. La fréquence du son émis est directement liée à la nature et à la fréquence du signal appliqué aux bornes du transducteur piézo-électrique.

Sur la carte d'évaluation PICDEM 2 Plus, le transducteur piézo-électrique peut être commandé grâce à la broche RC2 du micro-contrôleur. Le montage électronique utilisé pour commander le transducteur piézo-électrique permet de l'alimenter avec une tension bipolaire (+5V, -5V).

Pour générer un son, on doit donc générer un signal rectangulaire (grâce au module PWM) que l'on appliquera au transducteur piézo-électrique. Il faudra initialiser tous les modules nécessaires à la génération d'un signal rectangulaire (module CCP1, TIMER2) (voir p.159 du « *data-sheet* ») et initialiser le *PORTC.*

La période de la PWM est donnée par :

PWM Period = (PR2+1) x 4 x TOSC x (TMR2 Prescale Value)

La durée de l'état haut de la PWM, exprimée en sec. est donnée par :

PWM Duty Cycle = (CCPR1L:CCP1CON<5:4>) x TOSC x (TMR2 Prescale Value)

Trouvez la relation qu'il doit exister entre les valeurs des registres PR2 et CCPR1L:CCP1CON<5:4> pour que le rapport cyclique soit égal à 0.5.

Dans le programme d'initialisation :

- Configurer le PORTC en entrée excepté la broche RC2 en sortie,
- Configurer le module PWM, placer la valeur 103 (en décimal) dans le registre PR2 (fixe la fréquence du signal rectangulaire), régler le Duty Cycle à 0.5 x (PWM Period),
- Placer la valeur $(0F)_h$ dans le registre CCP1CON,
- Activer le TIMER2, placer le « *postscaler* » à 1:1, placer le « *prescaler* » à 16.

dans le programme principal : on implantera une boucle d'attente simple (avec les instructions goto, et nop).

Une partie du programme vous est fournie en page suivante. Créez un nouveau projet sur MPLAB, simuler son fonctionnement et validez-le sur la carte d'évaluation. Le buzzer de la carte doit émettre un son.

Comment faire pour que le son émis soit plus grave ?

```
      LIST P=18F4520
      #include <P18F4520.inc>
      #include <CONFIG.inc>


init
       clrf PORTC
       movlw h'00'
       movwf TRISC ; RC2 en sortie

;---Configuration du module PWM---
       movlw d'_ _ _'
       movwf PR2 ; fréquence 600Hz

       movlw b'00110100' ; D0
       movwf CCPR1L ; Rapport cyclique 0.5

       movlw h'0F'
       movwf CCP1CON;
;Rapport cyclique 0.5, mode PWM tout en active low, single output


;---Configuration du Timer2---
       movlw h'_ _ _'
       movwf T2CON ; TMR2 ON, postscaler=1, prescaler=16

;---Programme principal---
main
       nop
       goto main


END
```

En partant du programme précédent, on souhaite maintenant ajouter une fonctionnalité : à chaque appui sur le bouton poussoir connecté à RA4, on souhaite changer l'état du transducteur piézo-électrique.

Utilisez pour cela les 4 bits de poids faible de CCP1CON.