Laboratoire d'Électronique numérique:
Utilisation du *PIC16F877*

Année académique 2006-2007

Avant toute chose

1. Créez le répertoire **C :\ELEN033\XX_PIC** où XX est le nom de votre groupe
2. Copiez-y le contenu du repertoire **C :\ELEN033\PIC**

Vous devez faire toujours attention à

1. la mise sous tension de la carte :
   - Alimenter l'ICD2 MAIS PAS la carte.
   - Lancer MPLAB.
   - Choisir *Debugger→Select Tool→MPLAB ICD2*.
   - Dans le menu *Debugger* de MPLAB, choisir l'option *Connect*.
   - Après établissement de la communication, aller dans *Debugger→Settings*.
   - Dans la boîte de dialogue, choisir l'onglet *Power* et vérifier que la case *Power target circuit from MPLAB ICD2* est bien décochée, et appuyer sur OK.
   - Alimenter la carte et choisir *Debugger→Connect*.
2. la configuration du PIC (ligne débutant par __CONFIG), prenez l'habitude de configurer TOUS les bits !
3. la configuration de MPLAB (build options,...)

1 Introduction

Dans ce laboratoire, vous serez amenés à appliquer les concepts développés en répétition et dans l'*Introduction aux microcontrôleurs et à leur assembleur*. Tous les problèmes que vous pourriez rencontrer lors de ces manipulations peuvent être résolus en consultant ces deux références.

RAPPEL important

Tous les composants électroniques doivent être maniés avec délicatesse et précaution. En outre, une attention toute particulière doit être portée à l'électricité statique, qui risquerait d'endommager les circuits. Dans cette optique, nous vous demandons de toujours bien vouloir vous décharger en touchant une prise de terre. Évitez également de toucher les broches des circuits intégrés.

1.1 Sur le PIC utilisé

Le PIC utilisé dans ce laboratoire est un *16F877* (décrit brièvement à la fin de l'*Introduction aux microcontrôleurs et à leur assembleur*). Il s'agit en fait du modèle livré avec la carte de développement. En outre, c'est certainement celui que vous utiliserez dans le cours de Systèmes Programmés Enfouis de M. Boigelot. Il n'est donc pas inutile de déjà s'y familiariser.

Cependant, le niveau des exercices proposés dans ce laboratoire reste assez bas, de sorte que les commandes du *16F84* suffisent amplement pour les réalisé de façon simple.

Il est cependant conseillé de relire la section *7. Evolution : vers le 16F87x...* de l'*Introduction aux microcontrôleurs et à leur assembleur*.

1.2 Sur la carte de développement

Pour ce laboratoire, le PIC est placé sur une carte de développement de type $PICDEM^{TM}$ 2 *plus* de chez *Microchip*.

Une **carte de développement** est un ensemble de circuit intégrés précablés entre eux qui permet de réaliser des montages à diverses fins (pédagogiques, test...) sans devoir concevoir un circuit extérieur.

La carte que vous utiliserez dans ce laboratoire (représentée figure 1) est constituée des éléments suivants :

1. Support DIP[1] 18-, 28- et 40-broches. (Bien qu'il y ait 3 supports, un seul composant peut être utilisé à la fois)
2. Régulateur +5V embarqué pour alimentation direct depuis une entrée 9V, 100mA AC/DC (adaptateur secteur), une pile de 9V.
3. Connecteur RS-232 et hardware associé pour connexion directe à une interface RS-232.
4. Connecteur pour ICD[2].
5. Potentiomètre de 5KΩ pour les appareils avec entrée analogique.
6. Trois boutons-poussoir pour générer des stimuli extérieurs et le RESET.
7. LED verte indiquant la mise sous tension.
8. 4 LEDs rouges connectées au PORTB.
9. Jumper J6 pour déconnecter les LEDs du PORTB.
10. Oscillateur à quartz de 4 MHz (prêt à l'emploi).
11. Trous libres pour connecter le cristal.
12. Cristal à 32.768KHz pour les opérations d'horloge du Timer1.
13. Jumper J7 pour déconnecter l'oscillateur RC intégré (fréquence approximative : 2 MHz).
14. 256K x 8 Serial EEPROM.
15. Écran LCD.
16. Buzzer piezo.

[1] *Dual In-line Package*, technique de packaging de composant microélectronique où les broches de connexion, espacées de 2,54 mm, sont alignées sur deux rangées placées le long de chaque côté.

[2] In-Circuit Debugger

17. Aire de prototypage pour le hardware utilisateur.
18. Senseur thermique Microchip TC74.

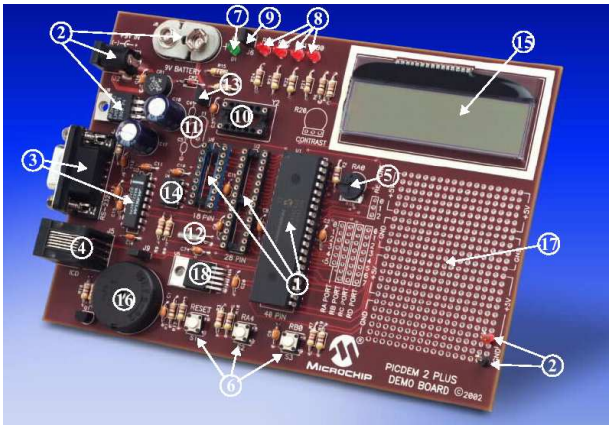

FIG. 1: *PICDEM 2 Plus* Hardware

1.3 Sur l'ICD2

Comme vous pouvez le voir, la carte de développement possède un connecteur pour raccorder un débogueur en circuit (ICD). Dans ce laboratoire, vous utiliserez MPLAB ICD2, un ICD produit également par Microchip ; et dont l'interface software s'intègre entièrement dans MPLAB IDE.

Comme son nom l'indique, un ICD est un composant électronique (hardware) qui permet (entre autre) de réaliser du débuggage (c'est-à-dire de la correction de code) lorsque le PIC est déjà programmé et inséré dans le circuit. Cette fonctionnalité ressemble un peu à la fonction "Debug" de MPLAB ; mais l'utilisation d'un ICD offre d'autres avantages non négligeables.

En effet, si la simulation du code non implémenté est nécessaire dans une première étape du développement d'un programme ; celle-ci ne peut pas prendre en compte les effets du circuit d'encadrement du PIC dans le monde réél. Or, celui-ci a une grande influence sur les fonctionnalités du PIC. Citons par exemple les problèmes de fonctionnement qui peuvent apparaître lorsque la pin RA4 est laissée flottante en sortie et qu'on lui impose un potentiel extérieur. L'ICD2 permet d'intégrer ces effets dans la simulation puisque le PIC est déjà placé dans son circuit.
L'ICD2 possède également les fonctionnalités d'un programmateur, ce qui évite de devoir renvoyer le PIC sur le DATAMAN (le programmateur universel du laboratoire) à chaque modification.

Au cas où le montage (connexions PC/ICD2/carte de développement/alimentation) ne serait pas raccordé, APPELEZ L'ASSISTANT POUR QU'IL VIENNE LE FAIRE ! ! ! La connexion des composants exige un ordre précis qui, s'il n'est pas respecté, pourrait endommager le matériel.

Un débogage avec l'ICD2 se passe en trois étapes :

1. **Programmation du code pour le débogage**
   L'ICD2 doit, avant de déboguer, placer le code du programme dans la mémoire du PIC. Il charge également un léger code de débogage (environ 0x120 mots) au début de la mémoire

programme. Il faut donc que le code du programme n'utilise pas cet espace. Il existe également d'autres restrictions :
- le mode debug doit être activé ;
- le watchdog doit être désactivé ;
- la protection de code dot être désactivée ;
- la protection de lecture en table doit être désactivée.

2. **Débogage**
   A cet étape on utilise vraiment le débogueur pour vérifier le bon fonctionnement pas à pas ou par étapes du programme et corriger les éventuelles erreurs résiduelles.
3. **Programmation du code final**
   Une fois le débogage terminé et les erreurs corrigées, l'ICD2 permet de programme le PIC avec le programme final (sans le code de débogage).

Nous explorons plus en détails les capacités de l'ICD2 à la fin du laboratoire.

2 Manipulations

2.1 Manipulation 1

Une pile peut facilement être implémentée sur un PIC au moyen des deux registres INDF et FSR. Voici les méthodes ”push” et ”pop”

```
DataPush Macro          ;  "Push" le contenu de "w" sur la pile FSR
  incf   FSR, f
  movwf  INDF
 endm

DataPop Macro           ;  "Pop" le contenu de "w" sur la pile FSR
  movf   INDF, w
  decf   FSR, f
 endm
```

1. Pour cette manipulation il est conseillé de relire les points suivants dans le fascicule *Introduction aux microcontôleurs et à leur assembleur* : 2.3 Registres : W, 3. Programmation des PICs, 4. Instructions
2. Insérer le PIC16F87x sur la carte. Vérifier que le Jumper J6 est branché et le J7 débranché. Lancer *MPLAB IDE*. Dans le menu project, ouvrir le projet *stack* et vérifier que le PIC16f877 et la toolsuite Microchip MPASM sont sélectionnés.
3. Trouvez l’expression mathématique des 3 calculs.
4. Au moyen du débuggeur et ”view — special function registers” et ”view — watch”, regarder l’évolution des variables, des registres STATUS pour le premier calcul et INDF, FSR pour le deuxième calcul.

2.2 Manipulation 2

Ouvrir le projet *error* et vérifier que le PIC16f877 et la toolsuite Microchip MPASM sont sélectionnés.

Cette application est sensée allumer une LED en RB1. Une fois le bouton RA4 appuyé, la LED RB1 est éteinte et RB0 allumée. Toute pression supplémentaire sur RA4 ne produit plus rien.
Quatre erreurs sont glissées dans le code, à vous de les retrouver.

Code

```
 LIST R=DEC
 ifdef __16F84
  INCLUDE "p16f84.inc"
 else
 ifdef __16F877
  INCLUDE "p16f877.inc"
 endif

 __CONFIG _CP_OFF & _WDT_ON & _XT_OSC & _PWRTE_ON
  PAGE
```

```
 org     0
  nop
  movlw  0x02           ;  Led RB1 on, RB0 off
  movwf  PORTB
  bsf    STATUS, RP0          ;  aller en banque 1 pour définir la direction des ports
  movlw  0x0FC          ;  RB0/RB1 en sortie
  movwf  TRISB
  bcf    STATUS, RP0          ;  retour en banque 0

Loop
   btfsc  PORTA, 0          ;  attendre que le bouton ra4 soit pressé
   goto  Loop

   rrf   PORTA          ;  éteindre rb1, allumer rb0
   goto   Loop          ;  boucler indéfiniment
 end
```

2.3 Manipulation 3

Ouvrir le projet *Interrupt* et vérifier que le PIC16f877 et la toolsuite Microchip MPASM sont sélectionnés.

Ce programme allume une LED pendant une seconde quand on appuie et relâche le bouton en RB0. Vous devrez utiliser le chapitre 5 portant sur les interruptions pour vous aider dans cette manipulation.

1. Débranchez le jumper J6 et J7 et branchez une LED sur RB1 (n'oubliez pas la résistance).
2. Complétez les lignes de code manquantes (????) et testez le code.

Code

```
 list p=16f877
 #include p16f877.inc

 CBLOCK 0x020                  ;  Commencer les registres a la fin des valeurs
  Dlay:2
 ENDC

;****** CONFIGURATION *******

 __config H'3779'

;***************************

 ORG 0
 goto init

;****** Le programme d' interruption se déclenche ***
;****** lorsque l' entrée RB0 passe de 0 à 1          ***
```

```
 ORG 4

;*********** Programme d 'interruption *********************

 bsf PORTB,1            ; on allume la led connectée sur rb1
 call Delay
 call Delay
 call Delay
 call Delay
 call Delay
 bcf PORTB,1
 ????

 RETFIE                  ; retour d 'interruption

;************* Programme d' INIT *************************

init
 bsf STATUS,5              ; on met à 1 le 5eme bit du registre status pour accéder
                           ; à la 2eme page mémoire ( pour configurer trisa et tr
                           ; -> broches en entrée ou en sortie )
 MOVLW B'10000001'  ; rb0,rb3,rb4 en entrée ( rb0 sera la broche utilisée
                           ; pour l ' interruption )

 MOVWF TRISB

 bcf STATUS,5               ; on remet à 0 le 5eme bit du registre status pour accéder
                              ; à la 1ere page mémoire

 ????  ; Le passage de 0 à 1 sur RB0 provoque une IT
                                                  ; sur un  front montant

 ???? ; autorise les interruptions sur RB0
 ???? ; autorise toutes les interruptions

 clrf PORTB

;************ Programme principal en boucle *******************

main

 sleep                ; mise en sommeil du PIC conso : 3.2 mA, attente impulsion sur RB0

 GOTO main

Delay               ;  Délai de 1/5 secondes
```

```
  clrf   Dlay
  clrf   Dlay + 1
  decfsz Dlay, f
   goto  $ - 1
  decfsz Dlay + 1, f
   goto  $ - 3

  return
 end
```

2.4 Manipulation 4

Pour cette manipulation, il faut à nouveau débrancher le jumper J6 et connecter une LED et une résistance sur RB1.
Ouvrez le projet *eep*. Celui-ci est identique au projet *Manip2a* vu dans ELEN040 sauf qu'au lieu d'allumer une LED sur une pression de RB0, il faut N pressions, où N est une valeur inscrite en EEPROM.

Commencez pas **lire le code** et essayez d'en **comprendre** les mécanismes (principalement ceux liés à l'**utilisation de l'EEPROM**). Pour mieux comprendre l'EEPROM, référez-vous au 2^e exemple du manuel, ainsi qu'à la section qui lui est dédiée.

Maintenant, nous allons un peu manipuler l'ICD2 pour mettre en évidence ses possibilités. Tout d'abord, branchez le jumper J7. Ensuite, dans MPLAB, sélectionnez *Programmer→Select Tool→None*, puis *Debugger→Select Tool→MPLAB ICD2*. Cochez également la case *EEPROM* dans le menu *Debugger→Settings→Program*. Les diverses fenêtres de débogage sont accessibles via le menu *View*.

- Commencez par lancer le débogage via *Debugger→Run*, ou F9. Pour arrêter l'exécution du programme, utiliser *Debugger→Halt*, ou F5.
- Il est également possible de faire tourner le programme pas à pas (*Debugger→Step into*, ou F7), ou en exécution animée (*Debugger→Animate*).

2.5 Manipulation 5

Prérequis

Pour cette manipulation il est conseillé de relire les points suivants dans le fascicule *Introduction aux microcontôleurs et à leur assembleur* :

- 2.2.1 Mémoire RAM
- 2.3 Registres : W, TMR0, OPTION, STATUS, PORTA/PORTB, TRISA/TRISB
- 2.4 Timer 0
- 2.5 Entrées/sorties
- 3. Programmation des PICs
- 4. Instructions
- 7. Évolution vers le 16F87x

Manipulations

1. Insérer le PIC16F87x sur la carte. Vérifier que le Jumper J6 est branché et le J7 débranché. Lancer *MPLAB IDE*. Dans le menu project, ourvir le projet *bincout*.

2. Ajouter les instructions manquantes *? ? ? ? ; selection de la banque*
   – Remarque : Il existe une instruction ”banksel registre” mais on demande d'utiliser le registre STATUS.
   Exemples d'utilisations de banksel : *banksel TRISA* ou *banksel PORTA*
3. Dans ”configure — configuration bits”, desactiver ”low voltage program”et activer ”background debug”.
   Compiler le code, programmer le PIC et lancer le debuggage avec l'ICD2.
   Expliquez l'intérêt de TOIF dans le programme principal. Comment est configuré le registre OPTION_REG ?
   Que constatez vous au niveau des LEDs ? Quelle est la fréquence du compteur binaire sur LEDs ? (la fréquence du compteur TMR0 avant prédiviseur est le quart de celle du cristal utilisé)
4. Donner une nouvelle valeur de OPTION_REG plus adéquate.
   Recalculer la nouvelle fréquence du compteur binaire.
5. ajouter ce code pour déclarer un nouveau registre *monregistre* :

```
CBLOCK H'20'          ;  registres principaux commencant a l'adresse 12
  monregistre         ;  un registre general de comptage
ENDC                  ;  fin des définitions de registres
```

   (avant la ligne : `ORG 0 ; debut du programme`) et remplacer le programme principal par :

```
again:   movlw D'7'          ;
         movwf monregistre
         incf PORTB, f       ;  ajouter 1 à portB


loop:    btfss INTCON, T0IF  ;  attendre TOIF
         goto $ -1           ;  revenir a la ligne preced.
         bcf INTCON, T0IF    ;  mettre le flag d'interruption à 0
         decfsz monregistre, f    ;  decremente monregistre, skip si 0
         goto loop
         goto again

         end                 ; fin du programme
```

   Décrire ce que font ces changements.
6. Expliquer les changements à faire pour avoir un ”décompteur”.
7. Comment commencer à compter à partir de 11 ? (pour rappel, utiliser W)
8. Expliquer les changements à faire pour compter jusqu'à 16 puis décompter jusque 0. De nouveau arrivé à 0, utiliser Z pour recommencer à compter.
9. Quels sont les changements à faire pour ne compter que les nombres impairs ?
10. Créer un afficheur qui déplace une seule LED allumée de droite à gauche puis de gauche à droite. (utiliser l'instruction *rlf registre,f*)

Code

```
;-------------------------------------------------------------------------;
; BINCNT.ASM         compteur binaire sur LEDs ( RB0 - RB4 )              ;
;-------------------------------------------------------------------------;
;-------------------------------------------------------------------------;
;         lignes utiles pour l'assembleur                                 ;
;-------------------------------------------------------------------------;
        LIST P=16F877           ;  modèle de pic utilisé
        INCLUDE "p16f877.inc"   ;  include qui définit divers registres
        ;ERRORLEVEL -224        ;  pour supprimer les warnings
        __CONFIG _PWRTE_ON & _HS_OSC & _WDT_OFF   ;  switchs de configuration

          ORG 0              ; début du programme
;-------------------------------------------------------------------------;
;       mettre tous les bits de portA et portB en sortie
;       et donner spécifier la valeur du registre option
;-------------------------------------------------------------------------;
         clrf   PORTA       ; Sorties portA à 0
         clrf   PORTB       ; sorties portB à 0
         ????  ; selection de la banque 1
         clrf TRISA           ; portA en sortie
         clrf TRISB           ; portB en sortie
         movlw B'00000001'    ;
                              ;
         movwf  OPTION_REG    ; écriture du registre option
         ????  ; sélection de la banque 0

;-------------------------------------------------------------------------;
;                   Programme Principal                                   ;
;-------------------------------------------------------------------------;

loop:
         incf PORTB, f        ; ajouter 1 à portB
         btfss INTCON, T0IF   ; attendre TOIF
         goto $ -1            ; revenir à la ligne preced.
         bcf INTCON, T0IF     ; mettre le flag d'interruption a 0
         goto loop

         end                  ; fin du programme
```