IUT de Poitiers – Site de Châtellerault

Départements
Mesures Physiques
Réseaux et Télécommunications

Formation Electronique Numérique 3

Programmation d'un PIC
Les interruptions

Exemples Divers de la programmation du PIC

Objectifs visés

Après avoir d'une part appris à utiliser les fonctions de base de l'outils de développement MPLAB et d'autre part appris à réaliser quelques programmations basique du micro contrôleur, nous allons maintenant approfondir nos connaissances sur l'aspect programmation/matériel du PIC.
Dans un premier temps, nous allons apprendre le fonctionnement des Interruptions, puis nous programmerons divers exemples mettant en œuvre ce procédé.

Travail proposé:

1er programme

1 – Préparation

Un répertoire de travail vous a été attribué. L'ensemble des fichiers nécessaires aux différentes séances de travaux pratiques sera stocké dans le répertoire c:\PICDATA sur le disque dur.

Pour effectuer les opérations suivantes, reportez vous au feuillet "Utilisation MPLAB ".

2 - Fonctions de base

Lancez MPLAB IDE.
A l'aide du « Project Wizard » créez un nouveau projet dans le répertoire c:\PICDATA et ajoutez y le fichier Tempo.asm.
Dans la fenêtre Projet double cliquez sur Tempo.asm pour l'éditer.

Explication du programme :
Le registre _Option est en mémoire FLASH, et a une taille de 14 bits.
Reprendre le synoptique du PIC.
En ce qui concerne l'option _CONFIG
CP1/CP0 (bits 13/12 + 5/4) :Indique quelle zone du PIC sera protégée (_CP_OFF : Aucune)
Debug (bit 11): Debuggage du circuit : Dédicace RB6 et RB7 à la communication avec un debugger
WRT (bit 9) : Autorisation d'écriture en Flash pour sauvegardée des données (dans les zones autorisée ci-dessus)
CPD (bit 8) : Protection en lecture de la mémoire EEPROM (_CPD_OFF =1 Non Protégé)
LVP¨(bit 7) : Utilisation de la broche RB3/PGM comme broche de programmation (si '1').
BODE N (bit 6) : Provoque la ré-initialisation du PIC en cas de chute de tension.
PWRTE (bit 3) Délai de démarrage à la mise en service

WDTE : WatchDog Timer
FOSC1/FOSC2 : Sélection du type d'oscillateur.

L'objectif dans un premier temps est de réaliser une routine d'attente permettant d'allumer une LED pendant une demi seconde, puis de la laisser éteinte pendant une demi seconde.
Nous allons utiliser une routine logicielle d'attente. Cela consiste à occuper le microprocesseur pendant une demi seconde (par des décrémentations) après l'allumage ou l'extinction de la LED.

- Combien de fois utilise t'on la routine temporisation ?
- Sachant que le PIC fonctionne à partir d'un quartz à 8 MHz, combien dénombre t'on de cycle d'horloge en une demi seconde ?
- Un cycle machine a une durée de 4 coups d'horloge. A partir du tableau des instructions (cf. le nombre de cycle machine associé à l'instruction), réaliser une routine pour « occuper » le microcontrôleur pendant une demi seconde ? On utilisera pour cela la fonction décrémentation d'une variable, rappelez vous qu'une variable n'est définie que sur 8 bits.

N'oubliez pas qu'un appel à une sous routine s'effectue par une instruction call et le corps de l'instruction se termine par le code return.

On utilise la même routine pour l'allumage et l'extinction.
L'horloge fonctionne à 8 MHZ, c'est-à-dire 4 000 000 de coups d'horloge en 0.5 seconde.
Nous aurons alors 1 000 000 de cycles machines à ne rien faire.

Nous allons écrire l'ossature du programme

```
;*****************************************
;*              Déclaration de variables              *
;*****************************************

        CBLOCK 0x00C
        Cmpt1 : 1
        ENDC


;*****************************************
;*      Sous routine  de temporisation              *
;*****************************************
; Cette routine introduit un retard de 500 us
; Elle ne reçoit aucun paramètre et ne retourne rien

tempo
        clrf    cmpt1
boucle1
        decfsz cmpt1,f
        goto boucle1
        return
```

Lançons la compilation F10.
Calculer le temps total :

Le temps total est de :
2 cycles pour l'appel de la sous routine
1 cycle pour le reset de la variable
256 cycles pour les 256 décrémentations
510 cycles pour les 255 goto
2 cycles pour le return

Soit un total de 771 cycles. Nous sommes loin des 1 000 000 cycles nécessaires.
Nous allons rallonger notre routine en réalisant une seconde boucle qui va forcer la première boucle à s'éxécuter 256 fois.
Déclarons une nouvelle variable.

cmpt1 : 1
cmpt2 : 1

Ecrivons les deux boucles imbriquées

```
tempo
        clrf    compt2
boucle2
        clrf    compt1
boucle1
        decfsz compt1,f
        goto boucle1
        decfsz compt2,f
        goto boucle2
        return
```

Quelle est la temporisation obtenue ?
Durée de la boucle 1 :
256+510 + 1 (decfsz) + 2 (goto boucle2) = 769 cycles.
On répète cette boucle 256 fois avec en plus l'instruction clrf. 256*770. On arrive à 196864 cycles machines. Or, on cherche à avoir 5 fois plus de cycles.
Nous pouvons soit recréer une troisième variable qui permet de réexécuter cette boucle 5 fois, ou on rajoute des instructions nop (5 instructions nop) dans la boucle 1.
Ainsi,

```
tempo
        clrf    compt2
boucle2
        clrf    compt1
boucle1
        nop
```

```
        nop
        nop
        …
        nop
        nop
        decfsz compt1,f
        goto boucle1
        decfsz compt2,f
        goto boucle2
        return
```

Durée de la boucle 1 :
256(1+12 nop) +510 + 1 (decfsz) + 2 (goto boucle2) = 4097 cycles.*
*On répète cette boucle 244 fois avec en plus l'instruction clrf . 244*4098. On arrive à 999912*
La durée correspondante est de 0,5 us à 0,01% près.

```
;*****************************************************************
;                     ASSIGNATIONS                *
;*****************************************************************

#DEFINE Led PORTB,1

;*****************************************
;*              Déclaration de variables              *
;*****************************************

        CBLOCK 0x0020
        cmpt1 : 1
        cmpt2 : 1
        ENDC


;*****************************************************************
;               DEMARRAGE SUR RESET                 *
;*****************************************************************

        org 0x000                               ; Adresse de départ après reset

        goto Init

;*****************************************************************
;               PROGRAMME PRINCIPAL                 *
;*****************************************************************
Init
```

```
        BCF STATUS,RP0                  ;On se place en bank 0, RP0 et RP1 sont
définies dans le Include
        BCF STATUS,RP1                  ;STATUS = IRP|RP1|RP0|!TO|!PD|Z|DC|C|


                                        ;car le Port A est définie en banque 0
        CLRF PORTB
        CLRF INTCON                     ; aucune interruption possible
        BSF STATUS,RP0                  ; BSF : Met le bit RPO de status à 1
                                        ; On passe en bank 1 pour modifier le
TRISA

        clrf TRISB                      ; TRISB est en bank1; tout le port est
configuré en sortie

;       movlw 0x00
;       movwf OPTION_REG                ; On configure le port B en Pull Up pour les
entrées

                                        ; Par précaution, on revient en bank0
        BCF STATUS,RP0




Boucle
        COMF PORTB,F;     bcf Led                       ; pour  eteindre
la led
        call tempo
        COMF PORTB,F;     bsf Led                     ; pour allumer la led
        call tempo
        goto Boucle

;*****************************************
;*      Sous routine  de temporisation   *
;*****************************************
; Cette routine introduit un retard de 500 us
; Elle ne reçoit aucun paramètre et ne retourne rien

                                ; Adresse de départ après reset

tempo
        movlw 0xFF              ; je met compt2 à 255
        movwf cmpt2
boucle2
        movlw 0xFF              ; je met compt1 à 255
```

```
        movwf cmpt1
boucle1
        nop
        nop
        nop
        nop
        nop
        decfsz cmpt1,f
        goto boucle1
        decfsz cmpt2,f
        goto boucle2
        return
        end
```

Nous allons maintenant procéder à une interruption pour piloter l'affichage des LEDS.

Mécanisme général d'une interruption.

Une routine d'interruption peut être considérée comme un sous programme déclenché par l'apparition d'un évènement spécifique.

Le programme se déroule normalement.
L'évènement survient
Le programme achève l'instruction en cours de traitement
Le programme traite l'interruption.

L'évènement qui déclenche l'interruption doit remplir deux conditions :

- l'évènement doit être dans la liste des évènements susceptibles de provoquer une interruption
- l'utilisateur doit avoir autorisé l'interruption, c'est-à-dire avoir signalé que l'évènement en question devait générer une interruption.

Mécanisme d'interruption sur un PIC

Toute interruption provoque une sauvegarde de l'adresse PC (c'est-à-dire l'adresse en cours du programme au moment ou est intervenu l'interruption) dans une pile interne à 8 niveau 'pile utilisé pour les sous programmes) et l'exécution du programme est routé à l'adresse 0x04 de la mémoire FLASH. Cette adresse est réservée pour les interruptions.
Dans le cas où l'utilisateur utilise plusieurs types d'interruption, il doit définir par une batterie de test quelle est l'interruption qui a déclenchée le déroutement du programme.
Le PIC ne sauvegardant que l'adresse du PC, c'est à l'utilisateur de sauvegarder les registres principaux (Status et W registre).
Enfin, une interruption ne peut pas être interrompu par une autre interruption sauf si le programmateur remet le flag GIE à un trop tôt. Les interruptions sont remises automatiquement en service après le retour dans la boucle principale (retfie).

Sources d'interruption.

A partir du datasheet du PIC, répertoriez les différentes sources d'interruption

Déclencheur	Flag	Registre	Adr	PEIE	Enable	Registre	Adr
Timer 0	T0IF	INTCON	0x0B	NON	T0IE	INTCON	0x0B
Pin RB0 / INT	INTF	INTCON	0x0B	NON	INTE	INTCON	0x0B
Ch. RB4/RB7	RBIF	INTCON	0x0B	NON	RBIE	INTCON	0x0B
Convert. A/D	ADIF	PIR1	0x0C	OUI	ADIE	PIE1	0x8C
Rx USART	RCIF	PIR1	0x0C	OUI	RCIE	PIE1	0x8C
Tx USART	TXIF	PIR1	0x0C	OUI	TXIE	PIE1	0x8C
Port série SSP	SSPIF	PIR1	0x0C	OUI	SSPIE	PIE1	0x8C
Module CCP1	CCP1IF	PIR1	0x0C	OUI	CCP1IE	PIE1	0x8C
Module CCP2	CCP2IF	PIR2	0x0D	OUI	CCP2IE	PIE2	0x8D
Timer 1	TMR1IF	PIR1	0x0C	OUI	TMR1IE	PIE1	0x8C
Timer 2	TMR2IF	PIR1	0x0C	OUI	TMR2IE	PIE1	0x8C
EEPROM	EEIF	PIR2	0x0D	OUI	EEIE	PIE2	0x8D
SSP mode I2C	BCLIF	PIR2	0x0D	OUI	BCLIE	PIE2	0x8D
Port parallèle	PSPIF	PIR1	0x0C	OUI	PSPIE	PIE1	0x8C

Les interruptions

Objectif : Nous allons construire un programme qui inverse l'allumage d'une LED à chaque pression sur un bouton poussoir sur l'entrée RB0.

```
;*****************************************************************
;  Ce fichier est la base de départ pour une programmation avec     *
;  le PIC 16F877A. Il contient  les informations de  base pour      *
;  démarrer.                                                        *
;                                                                   *
;  Il contient les routines d'interruptions et les variables        *
;  de sauvegardes w_temp et status_temp                             *
;                                                                   *
;*****************************************************************
;                                                                   *
;   NOM:     Interrup                                              *
;   Date:    09/11/2005                                            *
;   Version: 1                                                     *
;   Circuit: 16F877A                                               *
;   Auteur:  LAUNAY                                                *
;                                                                   *
;*****************************************************************
;                                                                   *
;   Fichier requis: P16F877.inc                                     *
;                                                                   *
;                                                                   *
;                                                                   *
```

```
;*****************************************************************
;                                       *
;   Notes:  Ce programme est destiné à inverser la led          *
;           quand une interruption est générée sur l'entrée RBO         *
;                                       *
;  Les erreurs : Oubli du return dans la routine IntRBO                                    *
; Il ne faut pas oublier de mettre TMR0 à O avant d'enlever le flag
*
; Ne pas oublier de créer la routine d'interruption à l'adresse 0x04
;*****************************************************************


        LIST    p=16F877A               ; Définition de processeur
        #include <p16F877A.inc>         ; Définitions des constantes

        __CONFIG  _CP_OFF & _DEBUG_OFF  &  _CPD_OFF & _LVP_OFF & _BODEN_OFF &
                  _WDT_OFF & _PWRTE_OFF & _HS_OSC

; '__CONFIG' précise les paramètres encodés dans le processeur au moment de
; Nous sommes en mémoire programme, le registre fait donc 14 bits
; la programmation du processeur. Les définitions sont dans le fichier include.
; Voici les valeurs et leurs définitions :
;       _CP_ON                          Code protection ON : impossible de relire
;       _CP_OFF                         Code protection OFF
;       _PWRTE_ON               Timer reset sur power on en service
;       _PWRTE_OFF              Timer reset hors-service
;       _WDT_ON                         Watch-dog en service
;       _WDT_OFF                Watch-dog hors service
;       _LP_OSC                         Oscillateur quartz basse vitesse
;       _XT_OSC                         Oscillateur quartz moyenne vitesse
;       _HS_OSC                         Oscillateur quartz grande vitesse
;       _RC_OSC                         Oscillateur à réseau RC

;*****************************************************************
;                   Définition                  *
;*****************************************************************

INTF EQU 1

;*****************************************************************
;                   ASSIGNATIONS                *
;*****************************************************************

#DEFINE Led PORTB,1                     ; On allume la led1 du port B quand on appuie sur RB0


;*****************************************************************
;                   Mémoire RAM                 *
;*****************************************************************

        CBLOCK      0x70                ; Début de la RAM Commune
```

```
        w_temp:1                        ; Registre de sauvegarde destiné à mémoriser le registre
                                        de travail
        status_temp:1                   ; Registre de sauvegarde destiné à mémoriser le satus
        ENDC

;*****************************************************************
;               DEMARRAGE SUR RESET                     *
;*****************************************************************

        org 0x000                               ; Adresse de départ après reset

        goto Init_reg


;*****************************************************************
;               ROUTINE D'INTERRUPTION                  *
;*****************************************************************

        org 0x004                               ; Adresse d'interruption

; sauvegarde des registres principaux
        movwf w_temp;                   ; Sauvegarde registre W
        swapf STATUS,w                  ; Swap Status et sauvegarde dans W
        movwf status_temp               ; Sauvegarde Status dans la ram

; Test si l'interruption est issue du RBO
; Ce test est "inutile" car nous n'avons autorisé qu'une seule interruption
;       btfss INTCON, INTF              ; Test si le Flag est à 1 (= Débordement)
;       goto restoreg                   ; Si le test est faux, je ne traite pas l'interruption et je restaure les
registres

; traitement de l'interruption
        call intRBO                     ; On doit impérativement faire un call et non un goto
                                        ; car on doit revenir ici en fin de la routine d'interruption
                                        ; restauration du flag
        bcf INTCON,INTF                 ; Je repositionne le Flag informant le dépassement à 0

; restauration des registres
restoreg
        swapf status_temp, w    ; On restaure le registre STATUS
        movwf STATUS
        swapf w_temp,f                  ; Inversion des bits poids forts/poids faible du registre de
                                        sauvegarde w_temp sans modifier le status
        swapf w_temp,w                  ; Re-inversion W <= W avant interruption
        retfie                          ; Retour à l'endroit du programme ou a eu lieu
                                        l'interruption

;*****************************************************************
;               PROGRAMME PRINCIPAL                     *
;*****************************************************************
Init_reg
```

```
        BCF STATUS,RP0                  ;On se place en bank 0, RP0 et RP1 sont définies
                                        dans le Include
        BCF STATUS,RP1                  ;STATUS = IRP|RP1|RP0|!TO|!PD|Z|DC|C|
                                                ;car le Port B est définie en banque 0

        CLRF PORTB

        BSF STATUS,RP0                  ; BSF : Met le bit RPO de status à 1
                                        ; On passe en bank 1 pour modifier le TRISA

        movlw 0x90                      ; Interruption : Validation Interruption Générale et
                                        Interruption TmrO 10010000
        movwf INTCON                    ; INTCON = GIE|PEIE|TOIE|INTE|RBIE|TOIF|
                                        INTF|RBIF|
                                        ; INTCON se trouve dans les 4 banques
                                        ; Nous allons maintenant configurer les ports en
                                        entrée ou en sortie
        movlw 0x01                      ; 01 = 00000001 : 0 Output, 1 : Input
        movwf TRISB                     ; TRISB est en bank1; tout le port est configuré en
                                        sortie sauf RBO

; ATTENTION, le programme ne marche pas si on met un PULL UP avec la carte.
; Il faut impérativement désactivé le pull up, mettre l'interrupteur des BP de la carte en +Vcc
; et mettre sur front montant
        movlw 0xC0                      ; On va pas imposer un PULL UP et une
                                        interruption sur front montant de RBO
        movwf OPTION_REG                ; RBPU|INTEDG|TOCS|TOSE|PSA|PS2|PS1|PS0
                                        : 0000000

        BCF STATUS,RP0

Init
        bcf Led                         ; dans la boucle la led est éteinte
boucle
        nop
        goto boucle

intRBO

        movlw 0x02                      ; On va inverser la valeur du port de sortie par une
                                        astuce en employant un Xor
        xorwf PORTB,w                   ; xor(Led=0,1)=1, xor(Led=1,1)=0        return
        movwf PORTB
        return                          ; Revenir dans la routine ayant appelé intRBO

        end
```

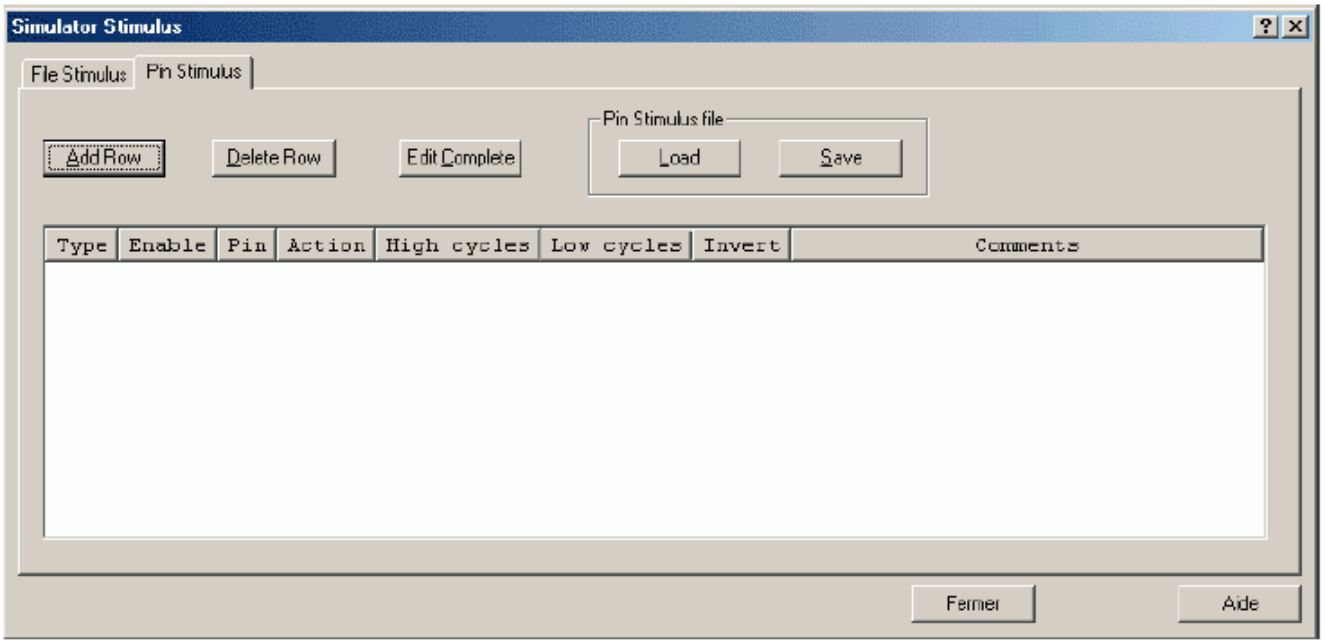
3 – Simulation

Choisissez l'outil de simulation MPLAB SIM comme "debugger".
Effectuez un RESET puis exécutez le programme pas à pas (F6 puis F7) et observez l'évolution des registres PCL, Status (Onglet View -> Special Function Register)

L'interruption doit être activé par un stimulus extérieur. Nous allons voir une autre méthode de simulation

Une fois arrivé dans le programme principal, le programme boucle indéfiniment. En effet, un événement extérieur (bouton-poussoir) est nécessaire pour provoquer le passage dans la routine d'interruption. Je vais maintenant vous expliquer comment simuler un événement extérieur.

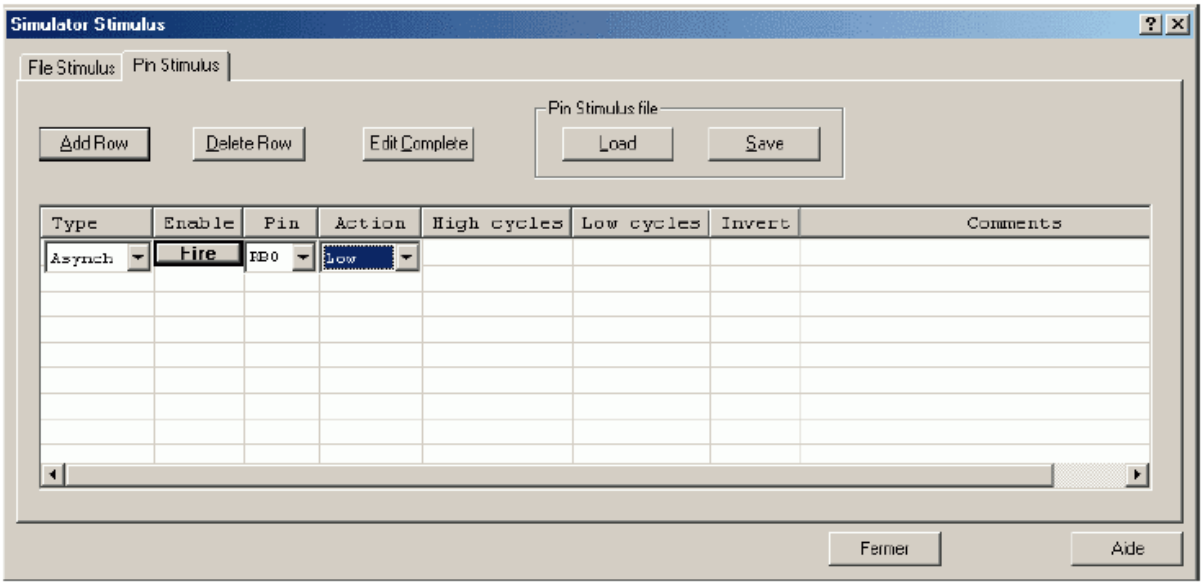
Allez dans le menu « debugger -> stimulus ». Si par hasard vous avez des messages d'erreur concernant un fichier, ignorez-les. Sélectionnez l'onglet « pin stimulus ».



Cliquez sur « add Row » pour obtenir un bouton d'action. Une nouvelle ligne est créée dans la fenêtre, avec un bouton intitulé « Fire ». Cliquez une fois dans la colonne « pin » juste à côté du bouton. Les cases « pin » et « action » se remplissent. Elargissez les colonnes à la souris pour mieux voir.
Nous allons maintenant préciser l'action de notre bouton. Dans la case « type », nous sélectionnons « asynch » pour «asynchrone ». En effet, l'événement pourra intervenir à n'importe quel moment de l'exécution du programme. Ne cliquez pas sur le bouton « fire » pour l'instant.

A l'aide du menu déroulant de la case « pin », nous allons déterminer quelle pin sera stimulée par l'action sur le bouton. Comme notre bouton se trouve sur la pin RB0, sélectionnez cette pin.
Si vous regardez dans la case « action », vous voyez que vous avez accès au mode de fonctionnement du bouton. Vous avez le choix entre Pulse (génère une impulsion), Low (place un

niveau 0), High (place un niveau 1), ou Toggle (inversion du niveau à chaque pression). Nous choisirons la moins pratique pour cet exemple, mais la plus explicite. Choisissez donc « Low ». Votre bouton de stimulation est maintenant configuré. Vous devriez obtenir une fenêtre du style suivant :



Créez maintenant une nouvelle ligne, avec « add row », et créez un second bouton, mais avec le paramètre « High » dans la case action.

Pressons <F7> pour avancer d'un pas et valider la modification de niveau. Examinez PORTB : RB0 est maintenant passé à 1. Notre bouton-poussoir n'est pas enfoncé. Pressez quelques fois <F7> pour vérifier que rien d'autre ne s'est passé.

Nous allons maintenant simuler la pression du bouton-poussoir :

- Pressez le premier bouton pour envoyer 0 sur RB0.
- Revenez dans l'éditeur et pressez une seule fois sur <F7>. L'instruction qui suit l'événement est alors l'instruction située à l'adresse 0x04, car le passage de 1 à 0 sur RB0 a provoqué notre interruption.
- Avancez lentement par pressions de <F7> dans la routine d'interruption. Examinez l'effet des différentes instructions vues

4- Programmez le composant

A partir de l'exécutable PicFlash2, programmer votre carte. Vous devez réaliser une lecture du fichier .hex, fichier qui a été compilé précédemment.

3ème programme : utilisation d'une interruption par Timer0

Dans cet exemple, nous allons parler de temporisation et comptages. Si on examine le fonctionnement du TimerO, on voit qu'il s'agit en fait d'un compteur.

On peut lui faire compter :

- Le nombre d'impulsions reçues sur la PIN RA4/TOK1. Nous sommes en mode Compteur.
- Comptez les cycles d'horloges du PIC lui-même.

La sélection de l'un de ces deux modes de fonctionnement s'effectue par le bit 5 du registre OPTION : TOCS pour Tmr0 Clock Source select bit.

Si celui-ci vaut 1, on travaille en mode compteur (sur les fronts montants ou descendants en fonction du bit 4 du registre OPTION : TOSE) et on vient lire la valeur du registre tmr0.

Si celui-ci vaut 0, on compte à partir de la valeur qui est sauvegardée dans le registre tmr0 située à l'adresse 0x01.

Les bits de poids faible du registre OPTION représentent les valeurs du prédiviseur.

PSA	PS2	PS1	PS0	/tmr0	/WD	Temps tmr0	Temps typique Watchdog (minimal)
0	0	0	0	2	1	512 µs	18 ms (7ms)
0	0	0	1	4	1	1024 µs	18 ms (7ms)
0	0	1	0	8	1	2048 µs	18 ms (7ms)
0	0	1	1	16	1	4096 µs	18 ms (7ms)
0	1	0	0	32	1	8192 µs	18 ms (7ms)
0	1	0	1	64	1	16384 µs	18 ms (7ms)
0	1	1	0	128	1	32768 µs	18 ms (7ms)
0	1	1	1	256	1	65536 µs	18 ms (7ms)
1	0	0	0	1	1	256 µs	18 ms (7ms)
1	0	0	1	1	2	256 µs	36 ms (14 ms)
1	0	1	0	1	4	256 µs	72 ms (28 ms)
1	0	1	1	1	8	256 µs	144 ms (56 ms)
1	1	0	0	1	16	256 µs	288 ms (112 ms)
1	1	0	1	1	32	256 µs	576 ms (224 ms)
1	1	1	0	1	64	256 µs	1,152 Sec (448 ms)
1	1	1	1	1	128	256 µs	2,304 Sec (996 ms)

Programme :

Nous souhaitons faire clignoter une LED à la fréquence approximative de 1Hz en utilisant le tmr0.

- Calculer le nombre de débordement de tmr0 nécessaire pour une temporisation de 500 ms.
- Choisir la pré-division adaptée à notre programme
- Réaliser le programme.

```
;*****************************************************************
;  Ce fichier est la base de départ pour une programmation avec      *
;  le PIC 16F877A. Il contient  les informations de  base pour       *
;  démarrer.                                                         *
;                                                                    *
;  Il contient les routines d'interruptions et les variables         *
;  de sauvegardes w_temp et status_temp                              *
;                                                                    *
;*****************************************************************
;                                                                    *
;   NOM:     Interrup                                               *
;   Date:    09/11/2005                                             *
;   Version: 1                                                      *
;   Circuit: 16F877A                                                *
;   Auteur:  LAUNAY                                                 *
;                                                                    *
;*****************************************************************
;                                                                    *
;   Fichier requis: P16F877.inc                                      *
;                                                                    *
;                                                                    *
;                                                                    *
;*****************************************************************
;                                                                    *
;   Notes:  Ce programme est destiné à réaliser une                  *
;                          temporisation à partir du timer 0                    *
;                  pour allumer/eteindre la led sur une durée de 500 us
*
;                                                                    *
;Il ne faut pas oublier de mettre TMR0 à O avant d'enlever le flag
*
;*****************************************************************


        LIST      p=16F877A                 ; Définition de processeur
        #include <p16F877A.inc>             ; Définitions des constantes

        __CONFIG   _CP_OFF & _DEBUG_OFF & _CPD_OFF & _LVP_OFF & _BODEN_OFF &
_WDT_OFF & _PWRTE_OFF & _HS_OSC

; '__CONFIG' précise les paramètres encodés dans le processeur au moment de
; Nous sommes en mémoire programme, le registre fait donc 14 bits
; la programmation du processeur. Les définitions sont dans le fichier include.
; Voici les valeurs et leurs définitions :
;       _CP_ON                          Code protection ON : impossible de relire
;       _CP_OFF                         Code protection OFF
;       _PWRTE_ON                   Timer reset sur power on en service
```

```
;       _PWRTE_OFF              Timer reset hors-service
;       _WDT_ON                 Watch-dog en service
;       _WDT_OFF                Watch-dog hors service
;       _LP_OSC                 Oscillateur quartz basse vitesse
;       _XT_OSC                 Oscillateur quartz moyenne vitesse
;       _HS_OSC                 Oscillateur quartz grande vitesse
;       _RC_OSC                 Oscillateur à réseau RC

;*****************************************************************
;               Définition              *
;*****************************************************************

OPTIONVAL EQU 0x87              ; OPTIONVAL prend pour valeur 10000111
                                ; Valeur que l'on prendra pour le registre OPTION
                                ; OPTION = RBPU|INTEDG|TOCS|TOSE|PSA|
PS2|PS1|PS0
                                ; RBPU = Tension Pull Up sur l'entrée B, PSA=0
Préscaler sur le Tmr0
                                ; PS2=PS1=PS0=1 => Freq utilisée = Fquartz/256
=> 32,768 ms
Val_compteur EQU 15         ; Valeur du compteur pour avoir une boucle de 500 ms
TOIF EQU 2

;*****************************************************************
;               ASSIGNATIONS            *
;*****************************************************************

#DEFINE Led PortB,0


;*****************************************************************
;               Mémoire RAM             *
;*****************************************************************

        CBLOCK      0x70                ; Début de la RAM Commune
        w_temp:1                        ; Registre de sauvegarde destiné à mémoriser le
registre de travail
        status_temp:1                   ; Registre de sauvegarde destiné à mémoriser le satus
        ENDC

        CBLOCK 0x20                     ; Debut de la RAM en banque 0
        cmpt : 1                        ; Nombre de boucle pour assurer une tempo de
500 ms avec un tmr0
        ENDC


;*****************************************************************
;           DEMARRAGE SUR RESET             *
;*****************************************************************

        org 0x000                       ; Adresse de départ après reset
```

```
        goto Init_reg


;*****************************************************************
;               ROUTINE D'INTERRUPTION                  *
;*****************************************************************

        org 0x004                       ; Adresse d'interruption

; sauvegarde des registres principaux
        movwf w_temp;                   ; Sauvegarde registre W
        swapf STATUS,w                  ; Swap Status et sauvegarde dans W
        movwf status_temp               ; Sauvegarde Status dans la ram

; Test si l'interruption est issue du TIMER0
; Ce test est "inutile" car nous n'avons autorisé qu'une seule interruption
        btfss INTCON, TOIF              ; Test si le Flag est à 1 (= Débordement)
        goto restoreg                   ; Si le test est faux, je ne traite pas l'interruption et je restaure les
registres

; traitement de l'interruption
        call inttmr0                    ; Un call est une routine qui permet de revenir à l'emplacement du
programme
                                                ; un goto ne revient pas
        clrf TMR0                       ; TMR0 vaut 255, je l'annule sous risque de remettre mon
flag à 1
; restauration du flag
        bcf INTCON,2                    ; Je repositionne le Flag informant le dépassement à 0
; restauration des registres

restoreg
        swapf status_temp, w    ; On restaure le registre STATUS
        movwf STATUS
        swapf w_temp,f                  ; Inversion des bits poids forts/poids faible du registre de
sauvegarde w_temp sans modifier le status
        swapf w_temp,w                  ; Re-inversion W <= W avant interruption
        retfie                          ; Retour à l'endroit du programme ou a eu lieu
l'interruption



;*****************************************************************
;               PROGRAMME PRINCIPAL                     *
;*****************************************************************
Init_reg

        ; PASSAGE EN BANK 0
        BCF STATUS,RP0                  ;On se place en bank 0, RP0 et RP1 sont définies
dans le Include
        BCF STATUS,RP1                  ;STATUS = IRP|RP1|RP0|!TO|!PD|Z|DC|C|
```

```
                                        ;car le Port A est définie en banque 0
        clrf TMR0
        CLRF PORTB

        ; PASSAGE EN BANK 1
        BSF STATUS,RP0                  ; BSF : Met le bit RPO de status à 1
                                        ; On passe en bank 1 pour modifier le
TRISA
        movlw 0xA0                      ; Interruption : Validation Interruption Générale et
Interruption TmrO 10100000
        movwf INTCON                    ; INTCON = GIE|PEIE|TOIE|INTE|RBIE|TOIF|
INTF|RBIF|
                                        ; INTCON se trouve dans les 4 banques

                                        ; Nous allons maintenant configurer les
ports en entrée ou en sortie
        clrf TRISB                      ; TRISB est en bank1; tout le port est configuré en
entrée

        movlw 0x07                      ; 07 = 00000111 : Prédiviseur par 256 sur Tmr0
        movwf OPTION_REG                ;
        ; PASSAGE EN BANK 0
        BCF STATUS,RP0                  ; Par précaution, on revient en bank0


Init
        movlw Val_compteur              ; J'affecte mon nombre de boucle
        movwf cmpt


Boucle                                  ; Boucle infini car je n'ai pas de tache de fond

        nop
        goto Boucle

inttmr0
        decfsz cmpt,f           ; Décrémente le compteur de passage
        return                          ; si le cmpt est non nul, nous ne sommes pas arrivé à la fin
de la tempo on ne fait rien
        comf PORTB,f                    ; xor(Led=0,1)=1, xor(Led=1,1)=0
        movlw 7                         ; on réassigne le compteur
        movwf cmpt
        return

        end
```