2012

INSTITUT SUPERIEUR DES ETUDES TECHNOLOGIQUES DE SOUSSE

Fascicule des Travaux Pratiques PIC18f450

Préparé par :

TLILI KAIS

HMIDENE ALI

TP0 : PRISE EN MAIN DU KIT µC-PIC2A

1. Présentation du Kit µc-PIC2A

Le kit **µc-PIC2A** de la société **SET** offre une large gamme d'applications des microcontrôleurs PIC. Ce Pupitre Microcontrôleur "tout-en-un" permet l'étude, la programmation et le test des programmes pour des microcontrôleurs 8-bits à mémoire Flash. Le pupitre permet de tester les programmes développés sur les principaux périphériques d'entrée et de sortie couramment rencontrés dans des systèmes électroniques embarqués. Le microcontrôleur est disposé sur un support à force d'insertion nulle de haute qualité. L'ensemble de ses entrées sorties est déporté sur des douilles de 2 mm et des connecteurs HE10.

2. Caractéristiques principales du Kit µc-PIC2A

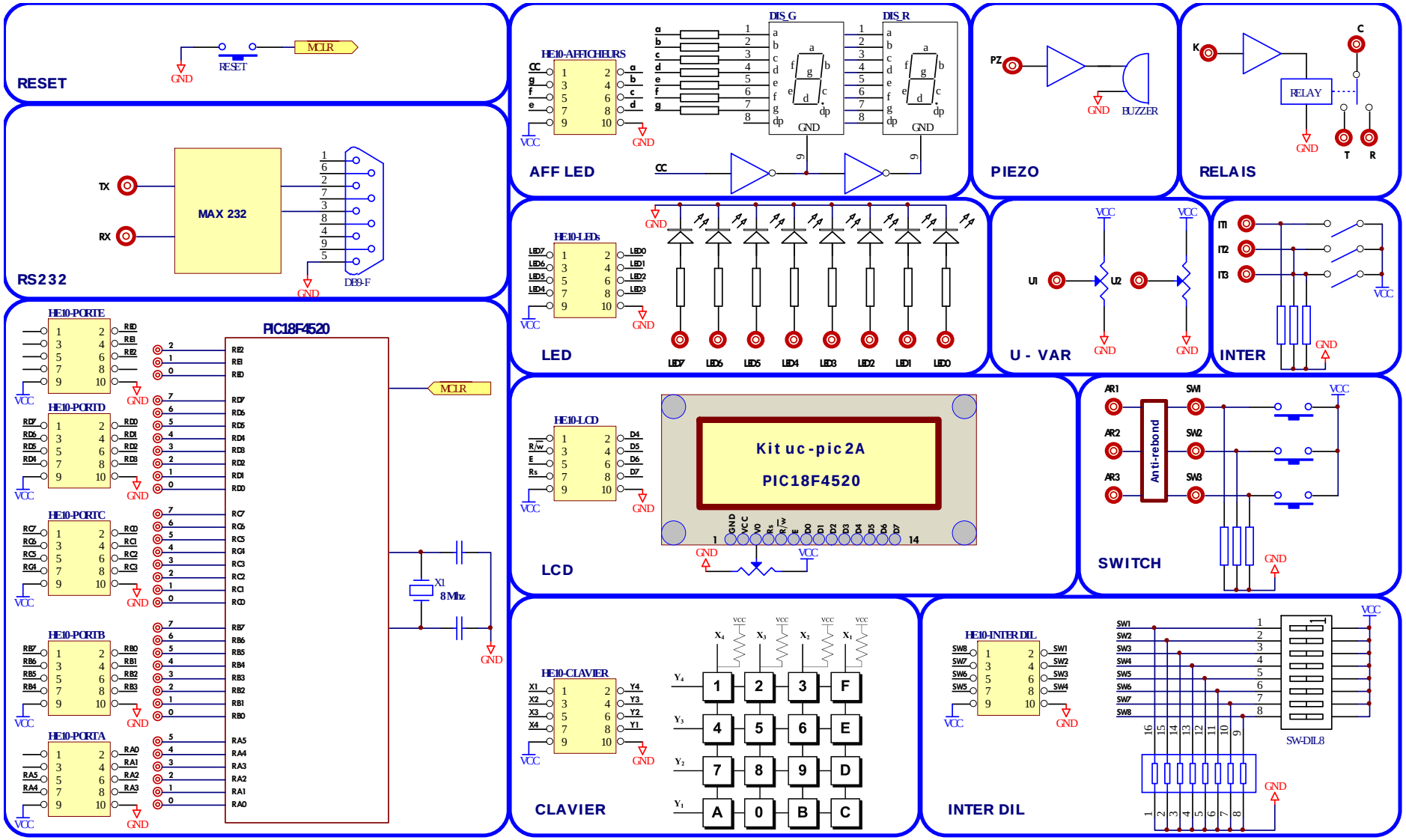
2.1. Caractéristiques matérielles

- ⇨ Un bloc secteur avec interrupteur et porte fusible intégré
- ⇨ Une prise mini-USB pour la programmation
- ⇨ Une sub-D 9 points RS 232 pour la communication
- ⇨ Un support pour microcontrôleur
- ⇨ Un switch reset avec une LED bicolore témoin
- ⇨ Deux douilles Rs232 : Tx, Rx.
- ⇨ Une douille masse et une douille + 4,096 volts
- ⇨ Les ports A, B, C, D et E du microcontrôleur sur douilles 2 mm et connecteurs HE10.
- ⇨ Un afficheur double 7 segments à LED entrées sur connecteur HE10
- ⇨ Un piézo entrée sur douille 2mm PZ.
- ⇨ Un jeu de 7 LED entrées possibles sur douilles ou un connecteur HE10
- ⇨ Un relais avec son voyant d'état.
- ⇨ Un afficheur LCD avec un potentiomètre de réglage du contraste.
- ⇨ Deux potentiomètres, sorties douilles 2mm U1 et U2. Tension de 0 à 4,096 volts.
- ⇨ Un module huit interrupteurs DIL sur sortie HE10
- ⇨ Trois interrupteurs On/Off sur douilles 2mm IT1, IT2 et IT3
- ⇨ Un clavier seize touches sur HE10.
- ⇨ Trois boutons On/Off sur douilles 2mm SW1, SW2, SW3 ou AR1, AR2, AR3 avec anti-rebonds intégrés.

2.2. Caractéristiques logicielles

Le kit est livré avec l'Environnement de Développement Intégré du MICROCHIP comprenant :

- ⇨ L'Environnement de Développement Intégré MPLAB IDE
- ⇨ Le compilateur MPLAB C18.
- ⇨ Le logiciel de programmation PICKit2.


RESET
MCLR
RESET
GND
RS232
TX
RX
MAX 232
DB9-F
GND
HE10-PORTE
HE10-PORTD
HE10-PORTC
HE10-PORTB
HE10-PORTA
PIC18F4520
MCLR
X1
8 Mhz
VCC
GND
HE10-AFFICHEURS
DIS_G
DIS_R
AFF LED
PIEZO
BUZZER
RELAIS
RELAY
HE10-LEDs
LED
U - VAR
INTER
HE10-LCD
Kit uc-pic2A
PIC18F4520
LCD
SWITCH
Anti-rebond
HE10-CLAVIER
CLAVIER
HE10-INTER DIL
SW-DIL8
INTER DIL

Kit µc-pic2A

3. MPLAB

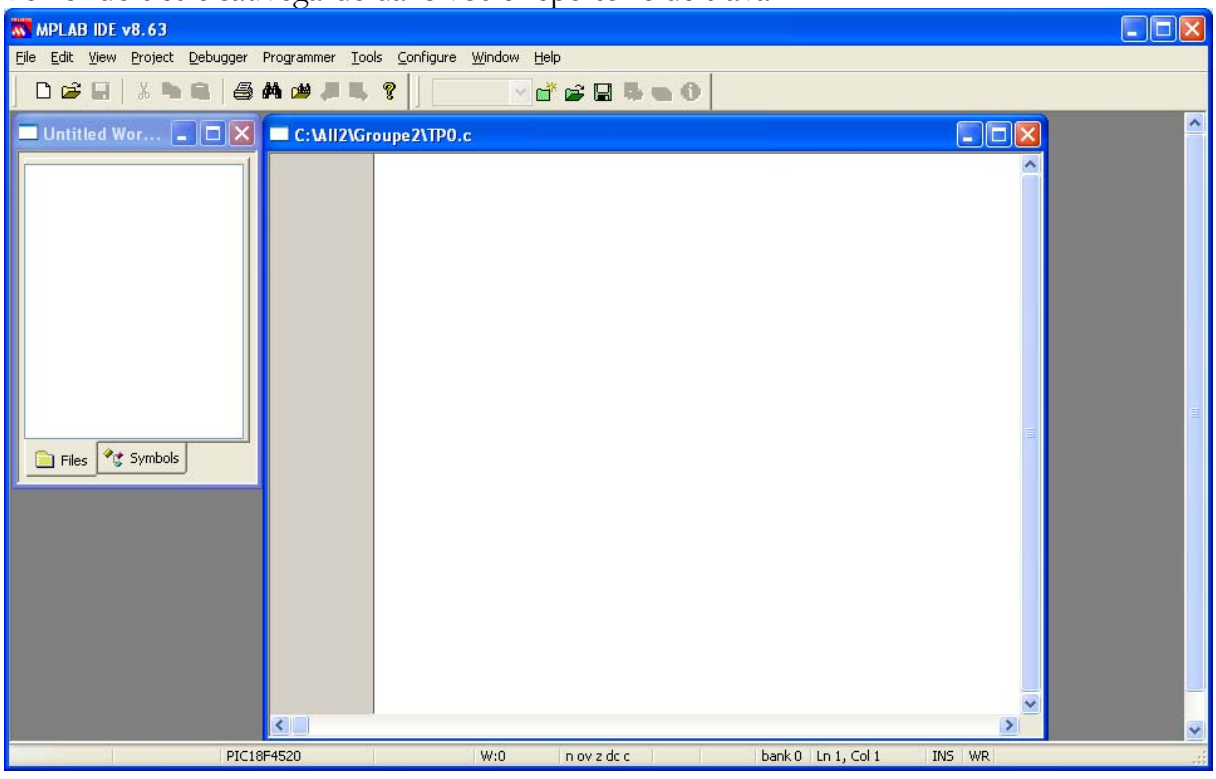
MPLAB IDE est un outil de développement complet pour les microcontrôleurs PIC. Afin de faciliter la programmation et la mise au point, cet environnement peut intégrer les outils de programmation (l'Assembleur, le compilateur C, l'éditeur de lien). De même sous cet environnement on pourra faire la simulation, le débogage et la programmation du circuit.

4. Création d'un projet avec MPLAB C18

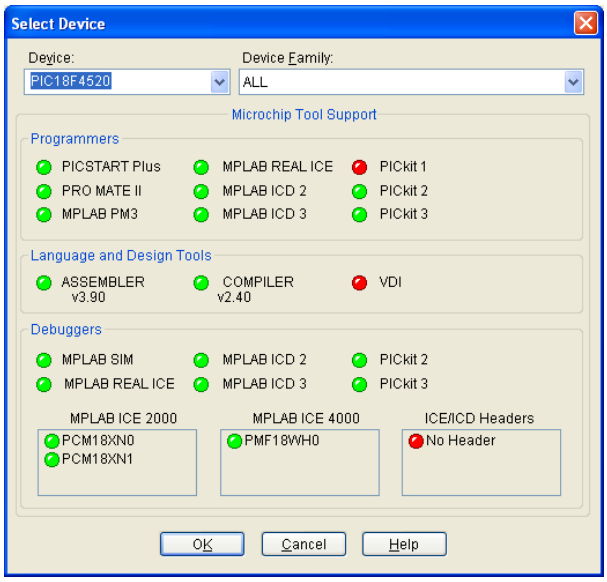
L'environnement **MPLAB C18** sauvegarde vos applications au sein de projets qui s'apparentent à un fichier ***Projet*** unique (avec l'extension .mcp). Ce projet comporte un ensemble de fichiers : fichiers Sources « .c », fichiers Entêtes « .h », fichiers Objets « .o », fichiers librairies « .lib » et fichier Linker Script « .lkr ». Pour créer un projet, il faut suivre les étapes suivantes :

Avant de lancer le logiciel MPLAB IDE, créer votre répertoire de travail : « D:\NomClasse\GoupeNum\ », Vous devez travailler dans ce répertoire le long du semestre.

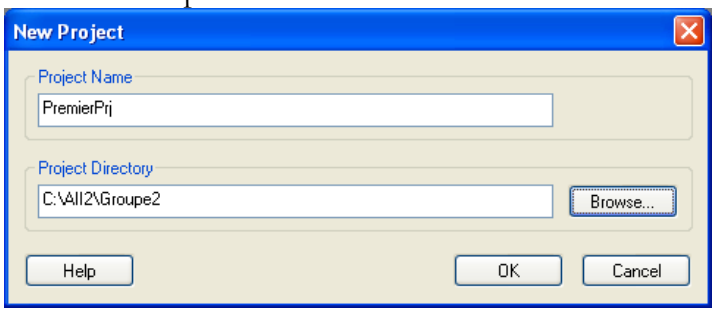
Etape 1 : Lancez le logiciel MPLAB, choisir la commande ***New*** du menu ***File*** pour créer le fichier source. Enregistrez le nouveau fichier vide sous le nom **TP0.c**, Attention, ce fichier doit être sauvegardé dans votre répertoire de travail.



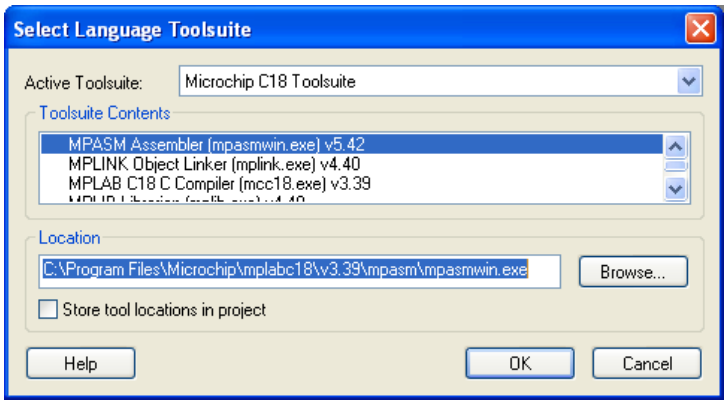
Etape 2 : Utilisez la commande ***Select Device…*** du menu ***Configure*** pour sélectionner le microcontrôleur PIC18F4520 et cliquez sur OK.



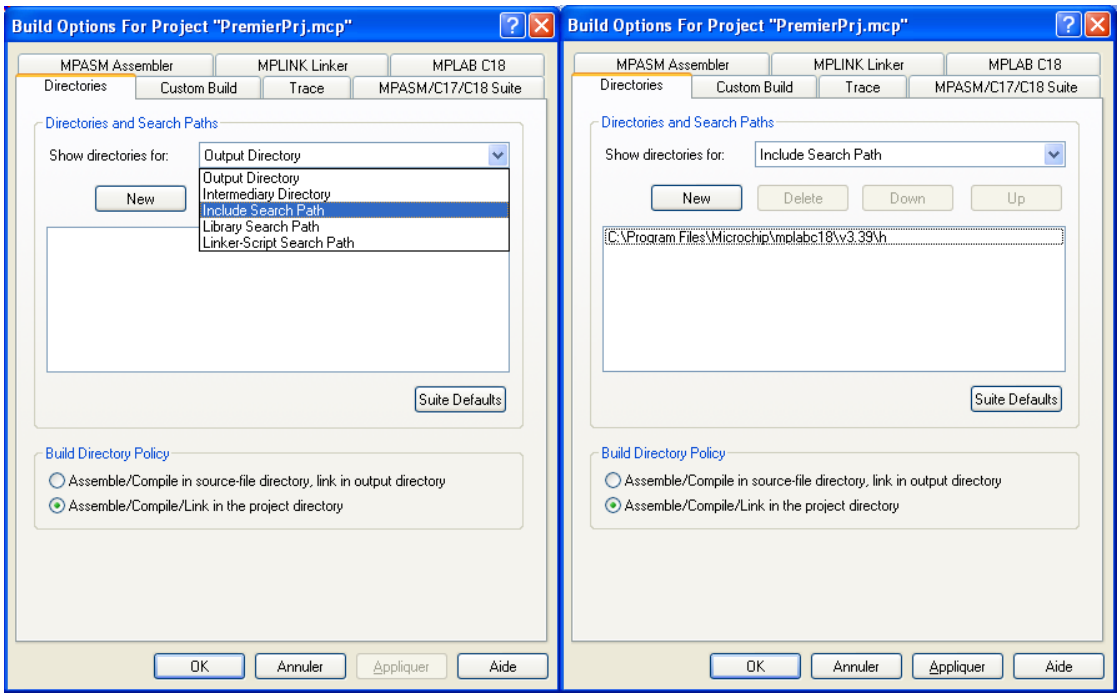
Etape 3 : Choisir ***Project>New…*** pour créer un Nouveau projet. Donnez un nom à votre projet et sélectionnez le répertoire de travail.



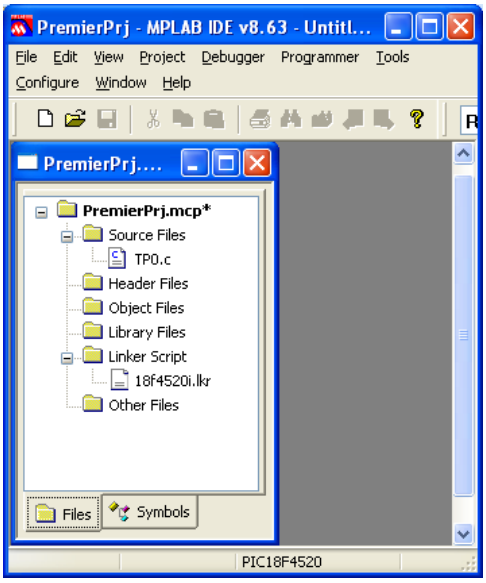
Etape 4 : Choisir la commande ***Project>Select Language Toolsuite*** pour choisir l'outil de développement, (**Microchip C18 Toolsuite**), vérifiez que votre fenêtre rassemble à celui-ci.



Etape 5 : Sélectionnez la commande ***Project>Build Options...>Project,*** dans le volet ***Directories***, donnez les chemins de recherche des fichiers *Include* et *Library*. Dans la case « *Show directories for :* », sélectionnez ***Include Search Path*** puis cliquez sur le bouton « *New* ». Sélectionnez alors le chemin de recherche des fichiers include (*C:\Program Files\Microchip\mplabc18\v3.39\h*). Faites la même chose pour ***Library Search Path*** en sélectionnant le chemin (*C:\Program Files\Microchip\mplabc18\v3.39\lib*).



Etape 6 : Si la fenêtre du Project n'est pas visible, affichez-la, en activant la commande ***Project*** du menu ***View***. Positionnez le pointeur de la sourie sur le Dossier *Source Files*, cliquez sur le bouton droit. Dans la fenêtre qui s'ouvre choisissez la commande ***Add Files…*** et Sélectionnez le fichier Source (**TP0.c** dans cet exemple).
Ajoutez de la même manière le fichier *18F4520_g.lkr* dans le dossier *Linker Script*. Le fichier linker, se trouve dans le répertoire lkr. (*C:\Program Files\Microchip\mplabc18\v3.39\bin\lkr \18F4520_g.lkr*), cette déclaration est devenue inutile à partir de la version 8.50.



Le projet est crée, vous pouvez maintenant saisir votre programme. Ouvrez le fichier source **TP0.c** et tapez le programme suivant :

```
#include <p18F4520.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config PWRT = OFF

unsigned long int count ;
void main()
{     TRISC = 0x00;
      PORTC = 0x00;
      while(1)
      {     PORTC = ~PORTC;
            count = 100000000;
            do{
                  count = count - 1;
            }while(count != 0);
      }
}
```

5. Compilation et chargement du programme

Une fois que vous aurez créé votre projet et écrit le code source, vous pouvez compiler votre programme en sélectionnant la commande ***Build All*** du menu ***Project***, ou cliquez sur le bouton ***Build All***. Vous devez avoir le message « **BUILD SUCCEEDED** », si non corrigez vos erreurs et compilez à nouveau votre programme.
Faites la simulation de votre programme sous ISIS.
Pour tester votre programme sur le kit, suivez les étapes suivantes :

- ⇨ Branchez le kit **µc-pic2A** sur le réseau.
- ⇨ Connectez-le au PC via le port USB
- ⇨ Démarrez le logiciel **PICkit2,** le logiciel doit détecter automatiquement le composant « PIC18F4520 ».
- ⇨ Chargez le fichier « PremierPrj.hex », en utilisant la commande ***Import HEX*** du menu **File**.
- ⇨ Démarrez la programmation du microcontrôleur en cliquant sur le bouton **Write**
- ⇨ Attendez la fin de programmation, puis appuyez sur le bouton **RESET** pour démarrer votre application.

Que constatez-vous ?
Pour augmenter le temps d'attente, remplacez la boucle d'attente par les instructions suivantes :

```
for (count = 0 ; count < 500 ; count--)
      Delay1KTCYx(2) ;
```

N'oubliez pas d'inclure le ficher entête « delays.h » *au début de votre programme.*

Exercice : la librairie « **delays.h** », comporte les fonctions de temporisation suivantes : *Delay1TCY(), Delay10TCYx(), Delay100TCYx(), Delay1KTCYx() et Delay10KTCYx().*
Le Temps de cycle TCY = 4·Tosc, sachant que la fréquence du quartz est Fosc = 8Mhz, D'où TCY = 500ns. En utilisant l'une des fonctions prédéfinies, écrire le code de la procédure d'attente en milliseconde ***delay_ms(int x)***.

TP1 : PROGRAMMATION DES PORTS D'ENTREES-SORTIES

1. Objectifs

- ⇨ Etre capable de gérer les ports parallèles en langage C
- ⇨ Savoir utiliser l'accès individuel aux bits.
- ⇨ Utiliser les fonctions en langage C.

2. Environnement de travail

- Compilateur Mplab C18 et le logiciel de programmation PICkit2.
- Une carte d'étude μc-PIC2A permettant l'implémentation et la vérification pratique des applications.

3. Travail demandé

3.1. PORTS E/S

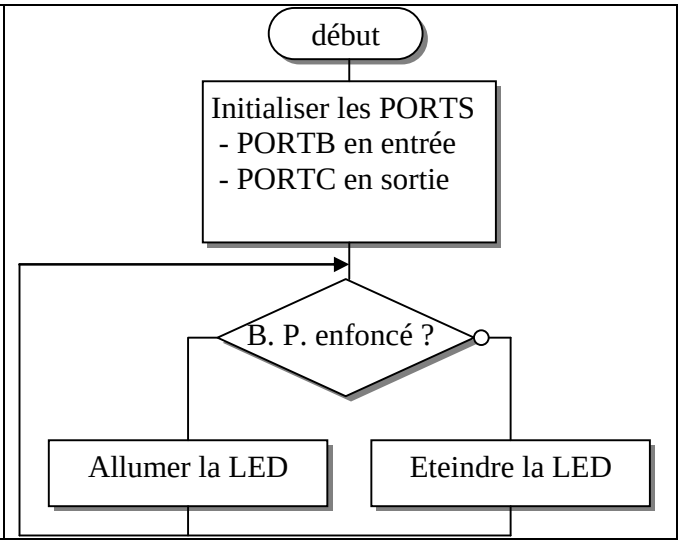
Programmez le microcontrôleur pour réaliser les tâches suivantes :

1. *Compteur simple* : utilisez le PORTC pour réaliser un compteur binaire qui s'incrémente une fois par seconde. Affichez l'état du PORTC sur les LEDs.
2. *Chenillard simple* : faite défiler l'allumage des LEDs de droite à gauche. Fixer le temps de défilement à 200ms.
3. Le PORTC est initialisé à la valeur 55H, faites clignoter la LED connecté à la broche RC2 sans changer l'état des autres bits.

3.2. Scrutation des entrées

1. Complétez le programme suivant afin d'allumer la LED connecté à la broche RC2 lorsqu'on appui sur le Bouton Poussoir RB0.

```
// commande d'une LED
#include <p18F4520.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config DEBUG = OFF
void main()
{     ADCON1 = 0x0F ;
      TRISB = 0xFF ;
      TRISC = 0x00 ;
      PORTC = 0x76 ;
  ……….
     if(……..)
       …… ;
     else
       …… ;
  ……..
}
```



2. dans le fichier p18F4520.h les ports sont déclarés de la manière suivante :

```
extern volatile near unsigned char   PORTC;
```

```
extern volatile near union {
  struct {
    unsigned RC0:1;
    unsigned RC1:1;
    unsigned RC2:1;
    unsigned RC3:1;
    unsigned RC4:1;
    unsigned RC5:1;
    unsigned RC6:1;
    unsigned RC7:1;
  };
  struct {
    unsigned T1OSO:1;
    unsigned T1OSI:1;
    unsigned CCP1:1;
    unsigned SCK:1;
    unsigned SDI:1;
    unsigned SDO:1;
    unsigned TX:1;
    unsigned RX:1;
  };
  struct {
    unsigned T13CKI:1;
    unsigned CCP2:1;
    unsigned :1;
    unsigned SCL:1;
    unsigned SDA:1;
    unsigned :1;
    unsigned CK:1;
    unsigned DT:1;
  };
  struct {
    unsigned T1CKI:1;
    unsigned :1;
    unsigned P1A:1;
  };
} PORTCbits;
```

- La première déclaration spécifie que le PORTC est un octet (volatile unsigned char) défini dans un fichier externe (extern).
- La deuxième déclaration précise que PORTAbits est une union de structures *anonymes* de bits adressables. Ce type de déclaration permet un accès individuel aux bits sans avoir recours aux masques.

Exp : PORTCbits.RC0 = 1, positionne le RC0 à 1 sans changer les autres bits.

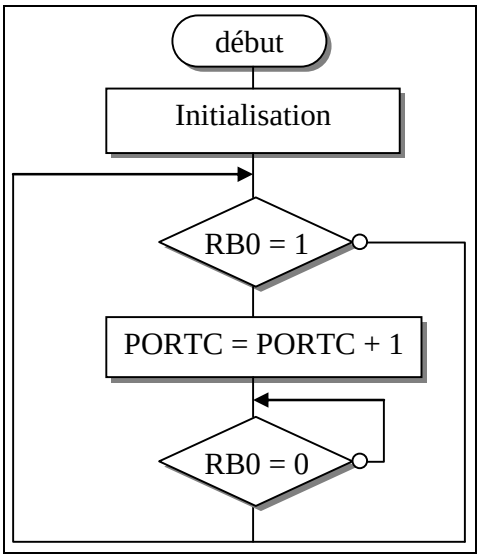
Reprendre la question précédente en utilisant l'accès individuel aux bits.

3. Quels est l'intérêt de faire la scrutation du bit RB0 dans une boucle.

3.3. Détection de front

On se propose ici de compter le nombre de paquets passant devant une cellule photoélectrique. La détection de passage d'un paquet est simulée par l'appui sur le bouton poussoir AR1 qu'on le branche à l'entrée RB0. Chaque appui sur le bouton AR1 incrémente le PORTC d'une unité. L'état du PORTC est affiché sur les LEDs.

1. Ecrire le programme de comptage des paquets.
2. Est-ce que le compteur s'incrémente convenablement ? Pourquoi ?
3. Afin d'éviter l'incrémentation du PORTC avec un pas aléatoire, on vous propose l'organigramme suivant :



Réécrire et tester à nouveau votre programme sur le kit. Que constatez-vous ? Conclure.
Connectez maintenant l'entrée RB0 à la borne « SW1 » au lieu de la borne « AR1 ». Appuyez sur le bouton RESET et exécutez de nouveau le programme. Que constatez-vous ? Apportez les modifications nécessaires à votre programme afin d'éliminer l'effet du rebondissement du bouton poussoir.

3.4. Programmation structurée (utilisation des fonctions)

Le corps d'une fonction est délimité par des accolades { ….. }. Un programme C doit se composer de plusieurs fonctions pour permettre une meilleure lisibilité et maintenance du programme.

La fonction main () est exécutée en premier, elle gère le bon déroulement du programme et l'échange des données entres les autres fonctions.

Exemple d'application : Etant donné le programme suivant :

```
#include <p18F4520.h>
#include <delays.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF
unsigned int i ;
void main()
{     TRISC = 0x00;                        // Initialisation
//-------------------------------------------------------
      while(1)
      {     PORTC = ~PORTC;
//-------------------------------------------------------
            for(i = 0; i < 500 ; i++)     // temporisation
                  Delay1KTCYx(5);
//-------------------------------------------------------
      }
}
```

Réécrire le programme en le décomposant en trois fonctions :

- une fonction « *Init()* » pour l'initialisation des ports
- une fonction de temporisation « *delay_ms(int n)* »
- une fonction principalee « *main()* ».

TP2 : CONVERTISSEUR ANALOGIQUE-NUMERIQUE (C.A.N)

1- Objectifs

Apprendre à programmer le convertisseur analogique numérique du microcontrôleur PIC18F4520.

2- Présentation

Entrées / sorties analogiques

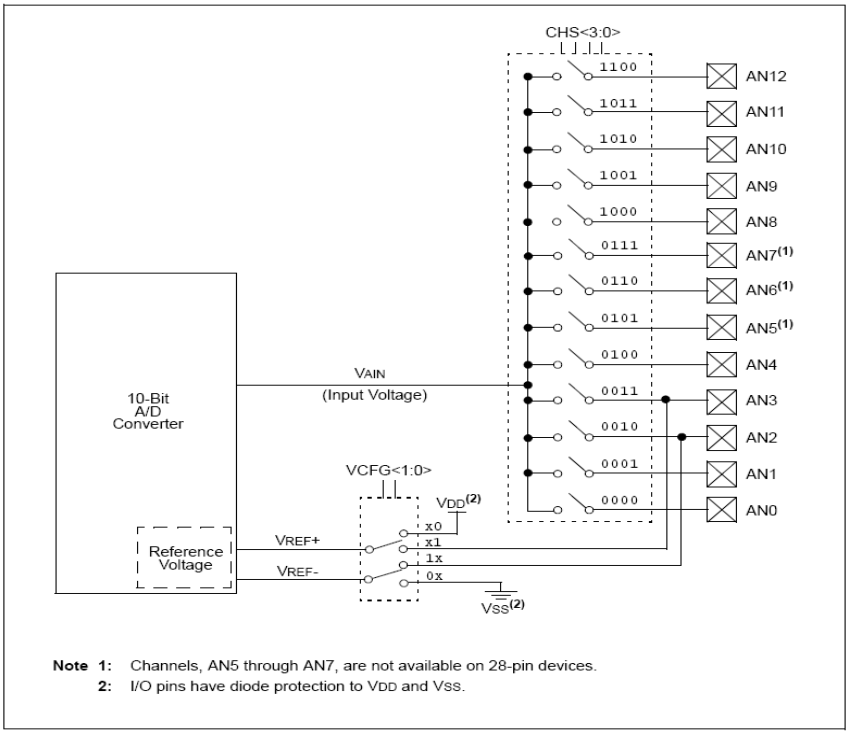
Le monde qui nous entoure est analogique, il faut donc pouvoir interagir avec lui.

— Applications : mesures de vitesse, de température, de pression, de position, régulation d'un processus analogique.

— Principe : utilisation d'un CAN et/ou CNA en interne pour passer du monde analogique au mon numérique et vice-versa.

Le microcontrôleur PIC18Fxx20 dispose d'un module de conversion analogique-numérique de 10 entrées analogiques pour les circuits 28 pins et 13 entrées analogiques pour les circuits 40 broches, le mot numérique convertis est codé sur 10 bits. Ces entrées sont reparties sur les ports A, E et B comme l'indique le tableau suivant :

RA0	RA1	RA2	RA3	RA5	RE0	RE1	RE2	RB2	RB3	RB1	RB4	RB0
AN0	AN1	AN2	AN3	AN4	AN5	AN6	AN7	AN8	AN9	AN10	AN11	AN12



✓ Le module CAN possède 5 registres :

- ADRESH : A/D Result High Register
- ADRESL : A/D Result Low Register

- ADCON0 : A/D Control Register 0
- ADCON1 : A/D Control Register 1
- ADCON2 : A/D Control Register 2

ADCON0: A/D CONTROL REGISTER 0

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CHS3	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bits 7-6 : inutilisé : lu comme « 0 »

bits 5-2 : **CHS<3:0>**: bits de Sélection du canal analogique, ils permettent de sélectionner l'entrée analogique à convertir.

0000 = Canal 0 (AN0)
0001 = Canal 1 (AN1)
0010 = Canal 2 (AN2)
0011 = Canal 3 (AN3)
0100 = Canal 4 (AN4)
0101 = Canal 5 (AN5)(1,2)
0110 = Canal 6 (AN6)(1,2)
0111 = Canal 7 (AN7)(1,2)
1000 = Canal 8 (AN8)
1001 = Canal 9 (AN9)
1010 = Canal 10 (AN10)
1011 = Canal 11 (AN11)
1100 = Canal 12 (AN12)
1101 = inutilisé (2)
1110 = inutilisé (2)
1111 = inutilisé (2)

bit 1 : **GO/DONE :** indique l'état de conversion A/D. La mise à 1 de ce bit démarre la conversion, il se remet automatiquement à 0 à la fin de conversion.

1 = En conversion
0 = Fin de conversion

bit 0 : **ADON:** bit d'activation du convertisseur

1 = module de conversion A/D est activé
0 = module de conversion A/D est désactivé (mis hors tension)

ADCON1: A/D CONTROL REGISTER 1

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-q(1)	R/W-q(1)	R/W-q(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bits 7-6 : inutilisé : lu comme « 0 »

bit 5 : **VCFG1:** bit de configuration de tension de référence (Vref – source)
1 : Vref – = AN2
0 : Vref –= VSS

bit 4 : **VCFG0:** bit de configuration de tension de référence (Vref + source)
1 : Vref + = AN3
0 : Vref + = VDD

bits 3-0 : bits de configuration du port A/D

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

ADCON2: A/D CONTROL REGISTER 2

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 : **ADFM:** bit de sélection du format

1 = Résultat justifié à droite
0 = Résultat justifié à gauche

bit 6 : inutilisé : lu comme « 0 »

bits 5-3 : **ACQT<2:0>:** bits de sélection du temps d'acquisition du A/D

111 = 20 TAD
110 = 16 TAD
101 = 12 TAD
100 = 8 TAD
011 = 6 TAD
010 = 4 TAD
001 = 2 TAD
000 = 0 TAD(1)

bits 2-0 : **ADCS<2:0>:** bits de sélection de l'horloge du convertisseur A/D

111 = FRC (clock derived from A/D RC oscillator)(1)
110 = FOSC/64
101 = FOSC/16
100 = FOSC/4
011 = FRC (clock derived from A/D RC oscillator)(1)
010 = FOSC/32
001 = FOSC/8
000 = FOSC/2

Etapes de conversion:

Pour réussir la conversion analogique-numérique on doit suivre les étapes suivantes :

1- Configurer le module A/D :
   - Configurer l'entrée analogique, la tension de référence et les E/S numérique (ADCON1).
   - Choisir le canal d'entrée de conversion (ADCON0).
   - Choisir le temps d'acquisition (ADCON2).
   - Choisir la fréquence de conversion (ADCON2).
   - Activer le module de conversion A/D (ADCON0).

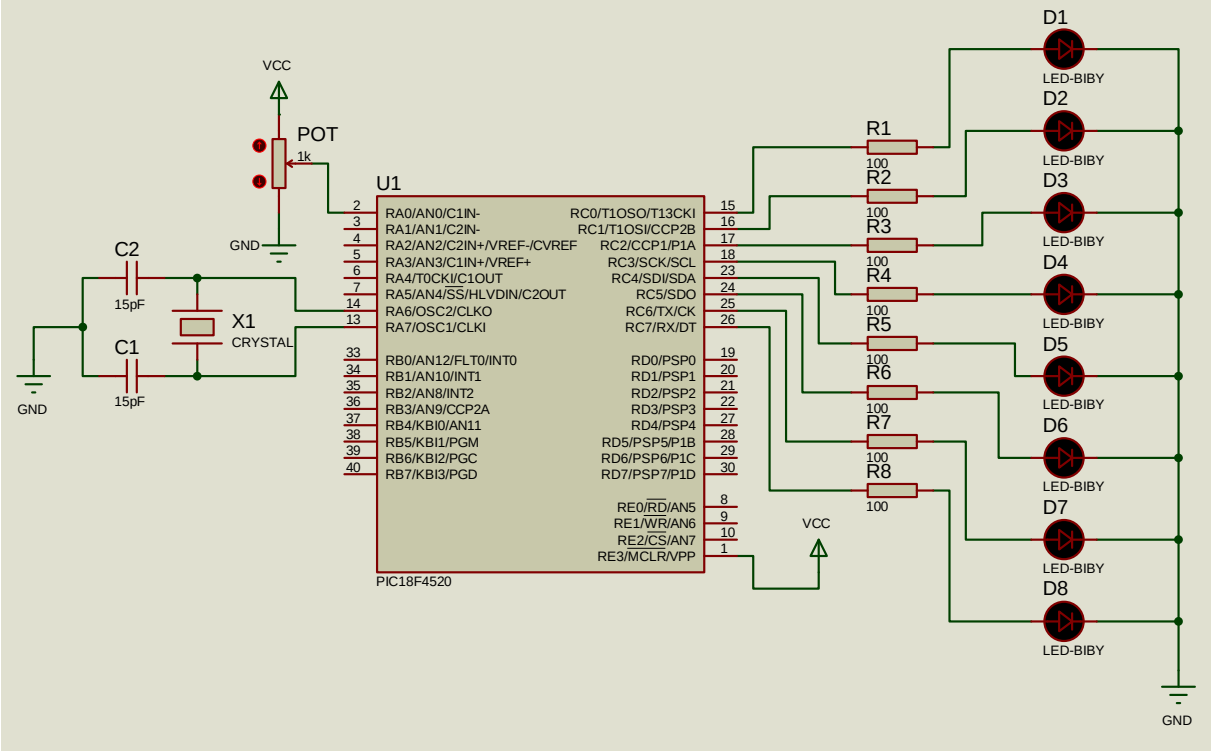
2- Configurer l'interruption de conversion si nécessaire :
   - Clear ADIF bit
   - Set ADIE bit
   - Set GIE bit
3- Attendre le temps d'acquisition demandé
4- Commencer la conversion
   - Activer le bit GO/DONE (ADCON0).

5- Attendre la fin de conversion soit par :
   - Remise à zéro du bit GO/DONE
   - Attendre l'interruption A/D
6- Lire le registre (ADRESH:ADRESL), clear le bit ADIF.
7- Pour la nouvelle conversion, on reprend les étapes de 1 ou 2, le temps de conversion par bit est égale à T_{AD}, une attente minimale de 2 T_{AD} est demandée avant qu'une nouvelle acquisition commence.

3- Travail demandé

1- On veut convertir un signal analogique appliqué à l'entrée AN0 (signal délivré par le potentiomètre POT). La valeur convertie sera envoyée sur le port C pour qu'elle soit affichée sur les diodes LED voir figure ci-dessous.

- En prenant $T_{AD} = 8T_{OSC}$ et un temps d'acquisition $T_{ACQ} = 4T_{AD}$. Donner les valeurs à charger dans les registres ADCON0, ADCON1, ADCON2, TRISC.
- Ecrire le programme correspondant.



2- Charger le programme, mesurer la valeur de tension à l'entrée AN0, Convertir la valeur de sortie en décimale N et complétez le tableau

V_{AN0}(v)									
N(décimale)									

TP3 : LES INTERRUPTIONS

1. Objectifs

- ⇨ Savoir utiliser les interruptions matérielles
- ⇨ Programmer les interruptions avec le langage MPLAB C18
- ⇨ Utiliser les périphériques du microcontrôleur en mode interruption
- ⇨ Traiter la priorité des interruptions

2. Présentation

Les interruptions sont utilisées pour favoriser l'exécution des tâches considérées comme prioritaires par rapport aux autres. La tâche d'interruption (sous programme d'interruption, appelé aussi routine d'interruption) est exécutée à partir d'une adresse fixe. Les microcontrôleurs PIC18 disposent de plusieurs sources d'interruptions. Ces interruptions sont structurées en deux niveaux ; les interruptions hautes priorités et les interruptions basses priorités. Les interruptions hautes priorités sont assignées au vecteur d'interruption d'adresse 0x0008, et les interruptions basses priorités au vecteur d'interruption d'adresse 0x0018.

Chaque source d'interruption est contrôlée par trois bits :

1. un bit ***Flag* (F)**, indique la présence de l'évènement d'interruption.
2. un bit ***Enable* (E)**, autorise la prise en compte de la demande d'interruption.
3. un bit **Priority (P)**, Sélection du niveau de priorité (haute ou bien basse).

Les microcontrôleurs PIC18, utilisent 10 registres pour le contrôle des interruptions :

- RCON
- INTCON
- INTCON2
- INTCON3
- PIR1, PIR2
- PIE1, PIE2
- IPR1, IPR2

Le bit IPEN du registre RCON, active ou désactive la logique de priorité. Lorsque la priorité des interruptions est active (IPEN = 1), le bit GIE/GIEH du registre INTCON permet la validation globale des interruptions hautes priorités et le bit PEIE/GIEL du même registre sert aussi à la validation globale des interruptions basses priorités.

Lorsque le bit IPEN est mis à 0 (état par défaut) la priorité des interruptions n'est pas prise en compte (compatibilité avec la famille mid-runge). Le bit PEIE/GIE permet la validation des périphériques et le bit GIE/GIEH sert à la validation globale de toutes les interruptions. Toutes les interruptions admettent l'adresse 0x0008 comme vecteur d'interruption.

La Figure 1 Montre les deux niveaux de priorité (partie haute et partie basse). De plus les interruptions sont divisées en deux catégories ; la première catégorie comporte les interruptions INT0, INT1, INT2, RB et Timer 0, elles sont contrôlées par les registres INTCONx. La seconde catégorie contient les interruptions dites *périphériques*, celles-ci sont contrôlées par les registres PIRx, PIEx et IPRx.

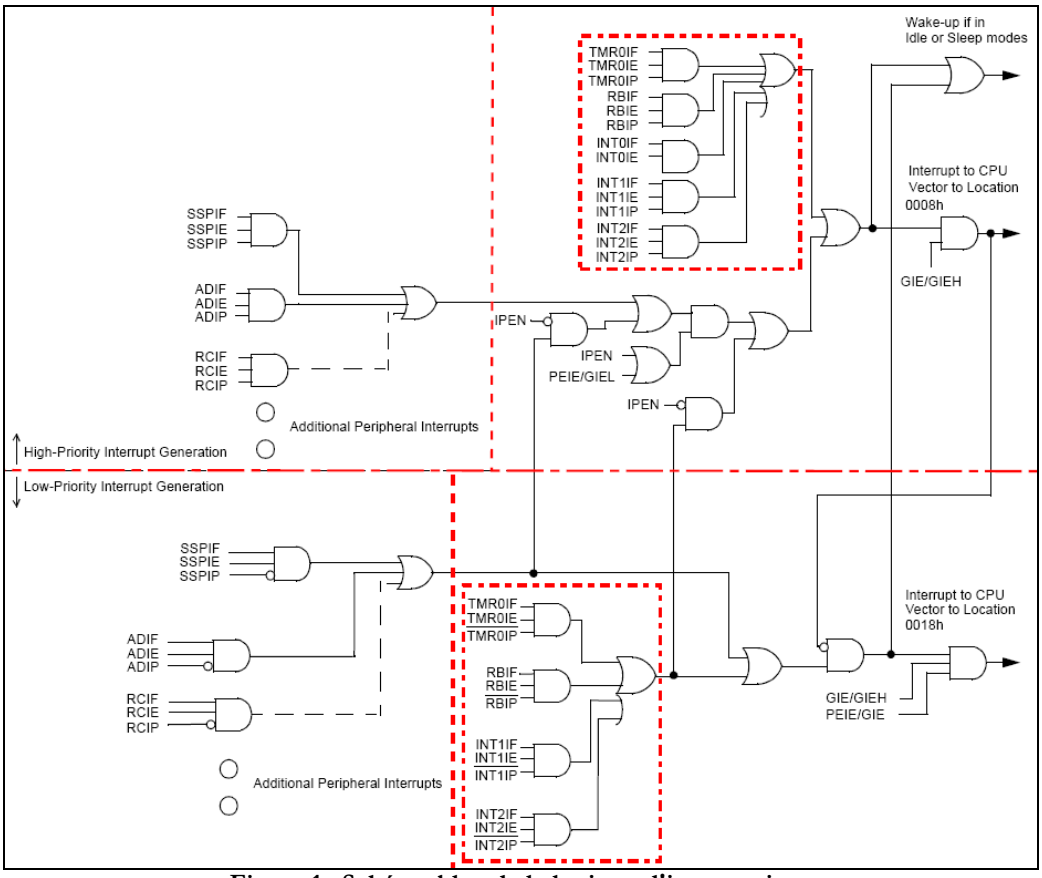


Figure 1 : Schéma bloc de la logique d'interruptions

2.1. Registres des interruptions du CPU

2.1.1. REGISTRE INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : **GIE/GIEH** : Global Interrupt Enable bit

IPEN = 1	**IPEN = 0**
1 : valide les ITs hautes priorités 0 : désactive toutes les interruptions	1 : valides toutes les ITs non masquées 0 : désactive toutes les interruptions

Bit 6 : **PEIE/GIEL :** Peripheral Interrupt Enable bit

IPEN = 1	**IPEN = 0**
1 : valide les ITs basses priorités 0 : désactive les ITs basses priorités	1 : valides les ITs périphériques 0 : désactive toutes les ITs périphériques

Bit 5 : **TMR0IE :** TMR0 Overflow Interrupt Enable bit
1 : valide l'interruption de débordement du Timer 0
0 : désactive l'interruption de débordement du Timer 0

Bit 4 : **INT0IE :** INT0 External Interrupt Enable bit
1 : valide l'interruption externe INT0
0 : désactive l'interruption externe INT0

Bit 3 : **RBIE** : RB Port Change Interrupt Enable bit
1 : valide l'interruption de changement d'état RB

0 : désactive l'interruption de changement d'état RB

Bit 2 : **TMR0IF :** TMR0 Overflow Interrupt Flag bit
1 : indique le débordement du Timer 0 (doit être remis à 0 par le prog.)
0 : pas de débordement du Timer 0

Bit 1 : **INT0IF** : INT0 External Interrupt Flag bit
1 : évènement s'est produit sur l'entrée INT0 (doit être remis à 0 par le prog.)
0 : aucun évènement n'a eu lieu sur l'entrée INT0

Bit 0 : **RBIF** : RB Port Change Interrupt Flag bit
1 : indique le changement d'état des broches RB7..RB4 (doit remis à 0 par le prog.)
0 : pas de changement d'état des broches RB7..RB4

2.1.2. REGISTRE INTCON2

R/W-1	R/W-1	R/W-1	R/W-1	U-0	R/W-1	U-0	R/W-1
$\overline{RBPU}$	INTEDG0	INTEDG1	INTEDG2	-	TMR0IP	-	RBIP
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : $\overline{RBPU}$: PORTB Pull-up Enable bit
1 : les résistances pull-up internes du PORTB sont désactivées
0 : les résistances pull-up internes du PORTB sont activées

Bit 6 : **INTEDG0 :** External Interrupt 0 Edge Select bit
1 : interruption INT0 active sur front montant
0 : interruption INT0 active sur front descendant

Bit 5 : **INTEDG1** : External Interrupt 1 Edge Select bit
1 : interruption INT1 active sur front montant
0 : interruption INT1 active sur front descendant

Bit 4 : **INTEDG2** : External Interrupt 2 Edge Select bit
1 : interruption INT2 active sur front montant
0 : interruption INT2 active sur front descendant

Bit 3 : **Non implémenté**

Bit 2 : **TMR0IP** : TMR0 Overflow Interrupt Priority bit
1 : interruption Timer0 haute priorité
0 : interruption Timer0 basse priorité

Bit 1 : **Non implémenté**

Bit 0 : **RBIP** : RB Port Change Interrupt Priority bit
1 : interruption RB haute priorité
0 : interruption RB basse priorité

2.1.3. REGISTRE INTCON3

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	-	INT2IE	INT1IE	-	INT2IF	INT1IF
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : **INT2IP :** INT2 External Interrupt Priority bit
1 : interruption INT2 haute priorité
0 : interruption INT2 basse priorité

Bit 6 : **INT1IP** : INT1 External Interrupt Priority bit
1 : interruption INT1 haute priorité
0 : interruption INT1 basse priorité

Bit 5 : **Non implémenté**
Bit 4 : **INT2IE** : INT2 External Interrupt Enable bit
1 : valide l'interruption externe INT2
0 : désactive l'interruption externe INT2
Bit 3 : **INT1IE** : INT1 External Interrupt Enable bit
1 : valide l'interruption externe INT1
0 : désactive l'interruption externe INT1
Bit 2 : **Non implémenté**
Bit 1 : **INT2IF** : INT2 External Interrupt Flag bit
1 : évènement s'est produit sur l'entrée INT2 (doit être remis à 0 par le prog.)
0 : aucun évènement n'a eu lieu sur l'entrée INT2
Bit 0 : **INT1IF** : INT1 External Interrupt Flag bit
1 : évènement s'est produit sur l'entrée INT1 (doit être remis à 0 par le prog.)
0 : aucun évènement n'a eu lieu sur l'entrée INT1

2.2. Les interruptions périphériques

Comme on a mentionné plus haut, chaque interruption est contrôle par trois bits : bit ***Flag***, bit ***Enable*** et bit ***Priority***. Pour les interruptions périphériques, ces bits se trouvent respectivement dans les registres PIRx, PIEx et IPRx.

2.2.1. REGISTRE INDICATEURS (FLAGS)

PIR1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
b7	b6	b5	b4	b3	b2	b1	b0

PIR2

R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIF	CMIF	-	EEIF	BCLIF	HLVDIF	TMR3IF	CCP2IF
b7	b6	b5	b4	b3	b2	b1	b0

2.2.2. REGISTRES DE VALIDATIONS (ENABLES)

PIE1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
b7	b6	b5	b4	b3	b2	b1	b0

PIE2

R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
OSCFIE	CMIE	-	EEIE	BCLIE	HLVDIE	TMR3IE	CCP2IE
b7	b6	b5	b4	b3	b2	b1	b0

2.2.3. REGISTRES DE PRIORITE (PRIORITY)

IPR1

R/W-1	R/W-1	R-1	R-1	R/W-1	R/W-1	R/W-1	R/W-1
PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
b7	b6	b5	b4	b3	b2	b1	b0

IPR2

R/W-1	R/W-1	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
OSCFIP	CMIP	-	EEP	BCLIP	HLVDIP	TMR3IP	CCP2IP
b7	b6	b5	b4	b3	b2	b1	b0

3. Programmation des interruptions avec le langage MPLAB C18

3.1. Méthode scrutation

Cette méthode consiste à faire la scrutation en boucle du ***Flag*** d'interruption (*par exemple INT0IF pour l'interruption externe INT0*) jusqu'à où celui-ci passe à 1. N'oubliez pas de remettre le bit flag à 0.

```
If(INTCONbits.INT0IF == 1)
{     INTCONbits.INT0IF == 0 ;
      // Traitement INT0
}
```

3.2. Méthode interruption

Le microcontrôleur exécute la routine d'interruption lorsque le bit ***Flag*** passe à 1 ; celui-ci devra être remis à 0 dans la routine d'interruption. Il faut bien noter qu'aucune interruption ne peut s'exécuter si les bits de validation ne sont positionnés (validation spécifique et globale).

Le compilateur MPLAB C18 utilise les directives « **`#pragma interrupt`** » pour déclarer les routines d'interruptions hautes priorités et, la directive « **`#pragma interruptlow`** » pour déclarer les routines d'interruptions basses priorités. De même il a besoin de la directive « **`#pragma code`** » pour loger le code de l'interruption à l'adresse du vecteur d'interruption 0x0008 ou 0x0018.

L'exemple suivant montre l'utilisation des interruptions avec le compilateur MPLAB C18 :

```
void low_isr(void);
void high_isr(void);


#pragma code low_vector=0x18
void interrupt_at_low_vector(void)
{
      _asm GOTO low_isr _endasm
}
#pragma code /* retoune à la section du code par défaut */
#pragma interruptlow low_isr
void low_isr (void)
{
      /* ... */
}

#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
      _asm GOTO high_isr _endasm
}

#pragma code /* retoune à la section du code par défaut */
#pragma interrupt high_isr
void high_isr (void)
{
      /* ... */
}
```

4. Travail demandé

4.1. Interruption sur l'entrée RB0/INT0

On va reprendre l'exercice 4.2 du TP1, qui consiste à incrémenter le PORTC à chaque appui sur le bouton poussoir connecté à l'entrée RB0.

Méthode scrutation

On vous propose l'algorithme suivant :

1. Si le Flag INT0IF égal à 1
   - Incrémenter PORTC
   - Remettre INT0IF à 0
2. revenir la l'étape 1.

Méthode interruption

Etapes à suivre :

- mettre dans la procédure d'initialisation les bits de validation spécifique (INT0IE) et globale (GIE/GIEH) à 1.
- Dans la routine d'interruption, incrémenter le PORTC et remettre le bit flag INT0IF à 0.

4.2. Conversion Analogique/Numérique par interruption

Reprendre le programme de conversion analogique numérique du TP2 en utilisant la méthode interruption.

L'ADC est contrôlé par les bits :

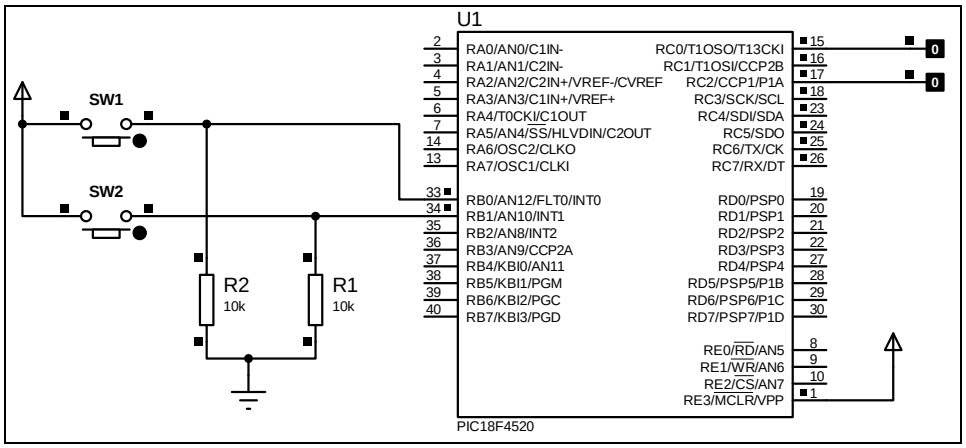
- ADIF du registre PIR1, ce bit indique la fin de conversion
- ADIE du registre PIE1, autorise l'interruption de l'ADC
- ADIP du registre IPR1, permet de choisir le niveau de priorité de l'interruption de l'ADC (haute priorité ou basse priorité).

N'oubliez pas de positionner les bits de validation globale (voir registre INTCON).

4.3. Priorité des interruptions

On donne le programme et le schéma de simulation sous ISIS. Ce programme consiste à :

- changer l'état de la sortie RC0 à chaque appui sur le bouton SW1
- mettre la sortie RC2 à 1 pendant 4 secondes lorsqu'on appui sur le bouton SW2.



On vous propose le code ci-dessous ; faites la simulation sous ISIS. Que constatez-vous ? Apportez les modifications si nécessaires au code proposé.

```
#include <p18F4520.h>
#include <delays.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config PWRT = OFF

void Isr_High ();
void Init();
void delay_ms(int x)
{
      int c_delay;
      for(c_delay = 0; c_delay < x; c_delay++)
            Delay1KTCYx(2);
}
void Init()
{
      TRISC = 0x00;
      TRISB = 0xFF;
      ADCON1 = 0x0F;
      RCONbits.IPEN = 1;
      INTCONbits.INT0IE = 1;
      INTCON3 = 0x48;
      PORTC = 0;
      INTCONbits.GIEH = 1 ;
}
#pragma code HighPriority = 0x08
void INT_ISR_HIGH (void)
{
  _asm
    goto Isr_High
  _endasm
}
#pragma code
#pragma interrupt Isr_High
void Isr_High ()
{
  if (INTCONbits.INT0IF)
    {
      INTCONbits.INT0IF = 0;
      PORTCbits.RC0 = !PORTCbits.RC0;
    }
  if (INTCON3bits.INT1IF)
    {
      INTCON3bits.INT1IF = 0;
      PORTCbits.RC2 = 1;
      delay_ms(4000);
      PORTCbits.RC2 = 0;
    }
}
void main()
{
      Init();
      while(1) continue ;
}
```

TP4 : PROGRAMMATION DES TIMERS

1. Objectifs

- ⇨ Comprendre les modes de fonctionnement des Timers du PIC18.
- ⇨ Mettre en œuvre le Timers0 et le Timer 2.

2. Rappel

2.1. Timer 0

Le timer 0 est formé d'un **pré-diviseur** suivi d'un registre compteur 8/16 bits. Le Timer0 utilise le registre **TMR0L** pour travailler en mode 8 bits, où la paire de registres **TMR0L** et **TMR0** lorsqu'il travaille en mode 16 bits. La configuration du Timer 0 est gérée par le registre **T0CON**. L'interruption du Timer0 est contrôlée par les bits **TMR0IF** et **TMR0IE** du registre **INCON**, ainsi que le bit **TMR0IP** du registre **INTCON2**.

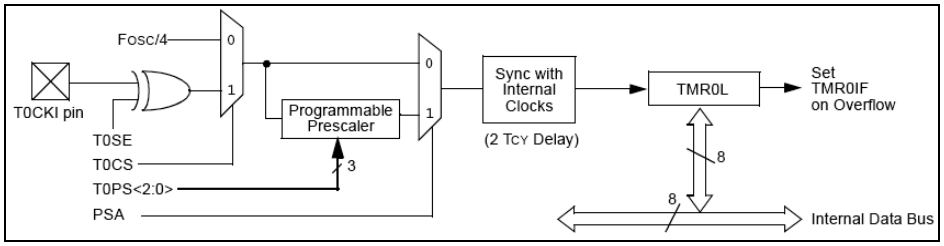


Figure 4-1 : Schéma synoptique du timer0

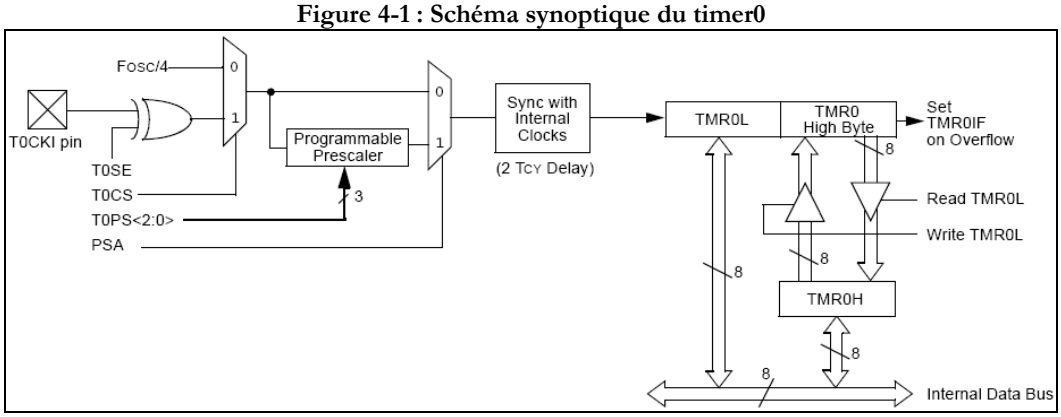


Figure 4-2 : Timer 0 en mode 16 bits

Registre T0CON

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	**T08BIT**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : **TMR0ON: Timer0 On/Off Control bit**
TMR0ON = 1 : validation du Timer 0.
TMR0ON = 0 : Timer 0 stopé
Bit 6 : **T08BIT: Timer0 8-Bit/16-Bit Control bit**
T08BIT = 1 : mode 8 bit (par défaut).
T08BIT = 0 : mode 16 bits.
Bit 5 : **T0CS** : **TMR0 Clock Source Select bit**
T0CS = 1 : fonctionnement en mode compteur (horloge externe)

T0CS = 0 : fonctionnement en mode Timer (horloge interne Fosc/4)
bit 4 : **T0SE** TMR0 Source Edge Select bit.
T0SE = 1 : incrémentation sur transition positive de l'horloge externe
T0SE = 0 : incrémentation sur transition négative de l'horloge externe
Bit 3 : **PSA** : Prescaler Assignment bit.
PSA = 1 : Prédiviseur non assigné
PSA = 0 : Prédiviseur assigné au Timer0
Bit 2..0 : **PS2, PS1, PS0** : Prescaler Rate Select bits.

PS2	PS1	PS0	Prédiv Timer0
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

2.2. Timer 2

Ce timer est formé d'un prédiviseur, d'un registre compteur 8bits **TMR2** et d'un postdiviseur. Contrairement aux autres timers, le registre **TMR2** déborde chaque fois que son continu dépasse la valeur chargée dans le registre **PR2** ; de même le bit ***T2ON*** ne bloque pas le comptage comme dans le cas du timer1, mais il le met hors service. Ce Timer est configuré par le registre **T2CON.** En mode interruption, il est contrôlé par les bits ***TMR2IF*** du registre **PIR1**, ***TMR2IE*** du registre **PIE1** et ***TMR2IP*** du registre **IPR1**

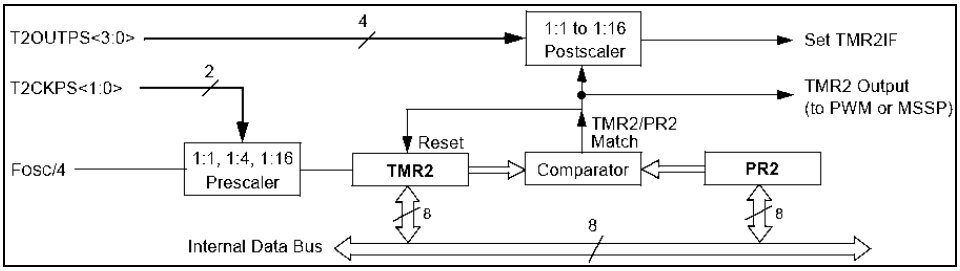


Figure 4-3 : Schéma synoptique du timer 2

Registre T2CON

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	**TOUTPS3**	**TOUTPS 2**	**TOUTPS 1**	**TOUTPS0**	**TMR2ON**	**T2CKPS1**	**T2CKPS0**
b7	b6	b5	b4	b3	b2	b1	b0

Bit 7 : **Non implementé .**
Bit 6..3 : **TOUTPS3..TOUTPS0: Timer2 Output Postscale Select bits**

TOUTPS3	TOUTPS 2	TOUTPS 1	TOUTPS0	PREDIVISION
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
		:		:
		:		:
1	1	1	1	16

Bit 2 : **TMR2ON: Timer2 On bit**
TMR2ON = 1 : Timer 2 ON (mise en service)
TMR2ON = 0 : Timer 2 OFF (mise hors service)
Bit 1..0 : **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits

T2CKPS1	T2CKPS0	Prédivision
0	0	1
0	1	4
1	x	16

Registres associés

INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF
INTCON2	*RBPU*	INTEDG0	INTEDG1	INTEDG2	-	TMR0IP	-	RBIP
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
IPR1	PSPIP	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP

2.3. Le mode scrutation

Nous savons que tous débordement du timer, entraîne le positionnement du bit indicateur de débordement (TMR0IF dans le cas du timer0 par exemple). Vous pouvez utiliser cet indicateur (*flag*) pour déterminer si le timer a débordé ou non.
Exp : `while(INTCONbits.TMR0IF == 0)` ; *// Attendre le débordement du timer0 pour sortir.*

```
INTCONbits.TMR0IF = 0 ;      // N'oubliez pas de remettre ce bit à 0
```

2.4. Le mode interruption

C'est le mode principal d'utilisation des timers. En effet, pour qu'une interruption se déclenche à chaque passage du bit indicateur de débordement à 1 ; il faut positionner à l'avance les bits de validation de l'interruption (voir TP3).

- Pour que l'interruption timer 0 se déclenche lorsque TMR0IF passe à 1, il faut positionner les bits TMR0IE et GIE/GIEH ou GIEL/PEIE selon le niveau de priorité.
- Pour que l'interruption timer 2 se déclenche lorsque TMR2IF passe à 1, il faut positionner les bits TMR2IE, PEIE et GIE

Dés que le timer déborde, La CPU se branche automatiquement à une routine d'interruption.

3. Travail demandé

3.1. Préparation

On veut configurer l'un des timers pour qu'il déborde toutes les 20ms. La fréquence du microcontrôleur F_{osc} = 8 Mhz.

1. Quel est le mode de fonctionnement à choisir (mode timer ou mode compteur) ?
2. Donner pour chacun des timers, les valeurs à charger dans les registres qui lui sont associés.
    a. Pour le Timer 0 : Prediv = ?; TMR0 = ?; T0CON = ?
    b. Pour le Timer 2 : Prediv = ?; PR2 = ?; Postdiv = ?; T2CON = ?
3. Quel est le timer qui convient mieux.

3.2. Manipulation

Le mode Timer

1. Saisir et tester le programme suivant :

```
#include <p18F4520.h>
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
#pragma config DEBUG = OFF
#pragma config PWRT = OFF
void Isr_High ();
void Init();
void Init()
{     TRISC = 0x00;
      PORTC = 0;
      RCONbits.IPEN = 1;
      INTCONbits.TMR0IE = 1;
      INTCON2bits.TMR0IP = 1;
      T0CON = 0x86;
      TMR0H = 0xA3;
      TMR0L = 0x03;
      INTCONbits.GIEH = 1 ;
}
#pragma code HighPriority = 0x08
void INT_ISR_HIGH (void)
{
  _asm
    goto Isr_High
  _endasm
}
#pragma code
#pragma interrupt Isr_High
void Isr_High ()
{     TMR0L = 0x03;
      INTCONbits.TMR0IF = 0;
      PORTCbits.RC0 = !PORTCbits.RC0;
}
void main()
{
      Init();
      while(1) continue;
}
```

a) Décrire selon le contenu de T0CON le mode de fonctionnement du timer 0. Déterminez alors sa période de débordement.
b) Que peut-on faire pour allonger la période de débordement à 10s ?

2. On veut faire l'acquisition d'un signal analogique appliqué à l'entrée AN0, la valeur convertie est envoyée au PORTC. La fréquence d'échantillonnage étant fixée à 20ms par le timer2.

a) Faite les initialisations de tous les registres à utiliser.
b) Ecrire le code correspondant

Le mode Compteur (faite la simulation des prog. Avec ISIS)

Ecrire un programme qui incrémente le contenu du registre TMR0 à chaque appui sur le bouton RA4. Afficher le compte sur les LEDs connectées au PORTC.