La programmation des PIC en C

La gestion du temps

Réalisation : HOLLARD Hervé.
http://electronique-facile.com
Date : 26 août 2004
Révision : 1.1

Sommaire

Sommaire 2
Introduction 3
Structure de ce document...4
Le matériel nécessaire..4
La platine d'essai 4
But à atteindre 5
L'instruction "delay"...5
Le "nop();" 7
La boucle 8
Le timer...11
Récapitulatif 16
Le chien de garde 17
Conclusion 19

Introduction

Les microcontrôleurs PIC de la société Microchip sont depuis quelques années dans le "hit parade" des meilleures ventes. Ceci est en partie dû à leur prix très bas, leur simplicité de programmation, les outils de développement que l'on trouve sur le NET.

Aujourd'hui, développer une application avec un PIC n'a rien d'extraordinaire, et tous les outils nécessaires sont disponibles gratuitement. Voici l'ensemble des matériels qui me semblent les mieux adaptés.

Ensemble de développement (éditeur, compilateur, simulateur) :
MPLAB de MICROCHIP **http://www.microchip.com**
Logiciel de programmation des composants:
IC-PROG de Bonny Gijzen **http://www.ic-prog.com**
Programmateur de composants:
PROPIC2 d'Octavio Noguera voir notre site **http://electronique-facile.com**

Pour la programmation en assembleur, beaucoup de routines sont déjà écrites, des didacticiels très complets et gratuits sont disponibles comme par exemple les cours de **BIGONOFF** dont le site est à l'adresse suivante **http://abcelectronique.com/bigonoff**.

Les fonctions que nous demandons de réaliser à nos PIC sont de plus en plus complexes, les programmes pour les réaliser demandent de plus en plus de mémoires. L'utilisateur est ainsi à la recherche de langages "évolués" pouvant simplifier la tâche de programmation.

Depuis l'ouverture du site **http://electronique-facile.com**, le nombre de questions sur la programmation des PIC en C est en constante augmentation. Il est vrai que rien n'est aujourd'hui disponible en français.

Mon expérience dans le domaine de la programmation en C due en partie à mon métier d'enseignant, et à ma passion pour les PIC, m'a naturellement amené à l'écriture de ce **didacticiel**. Celui-ci se veut **accessible à tous** ceux qui possèdent une petite expérience informatique et électronique (utilisation de Windows, connaissances minimales sur les notions suivantes : la tension, le courant, les résistances, les LEDs, les quartz, l'écriture sous forme binaire et hexadécimale.).

Ce quatrième fascicule vous permettra de réaliser n'importe quelle application, sera la fin de "l'apprentissage indispensable". Tous ce qui suivra vous permettra uniquement de programmer plus rapidement et de façon plus structurée.

Structure de ce document

Ce document est composé de chapitres. Chaque chapitre dépend des précédents. Si vous n'avez pas de notion de programmation, vous devez réaliser chaque page pour progresser rapidement.

Le type gras sert à faire ressortir les termes importants.

Vous trouverez la définition de chaque **terme nouveau** en **bas de la page** où apparaît pour la première fois ce terme. *Le terme est alors en **italique**.*

La couleur **bleue** est utilisée pour vous indiquer que ce **texte** est **à taper** exactement sous cette forme.

La couleur **rouge** indique des **commandes informatiques** à utiliser.

Le matériel nécessaire

Les deux logiciels utilisés lors du premier fascicule.

Un programmateur de PIC comprenant un logiciel et une carte de programmation. Vous trouverez tout ceci sur notre site.

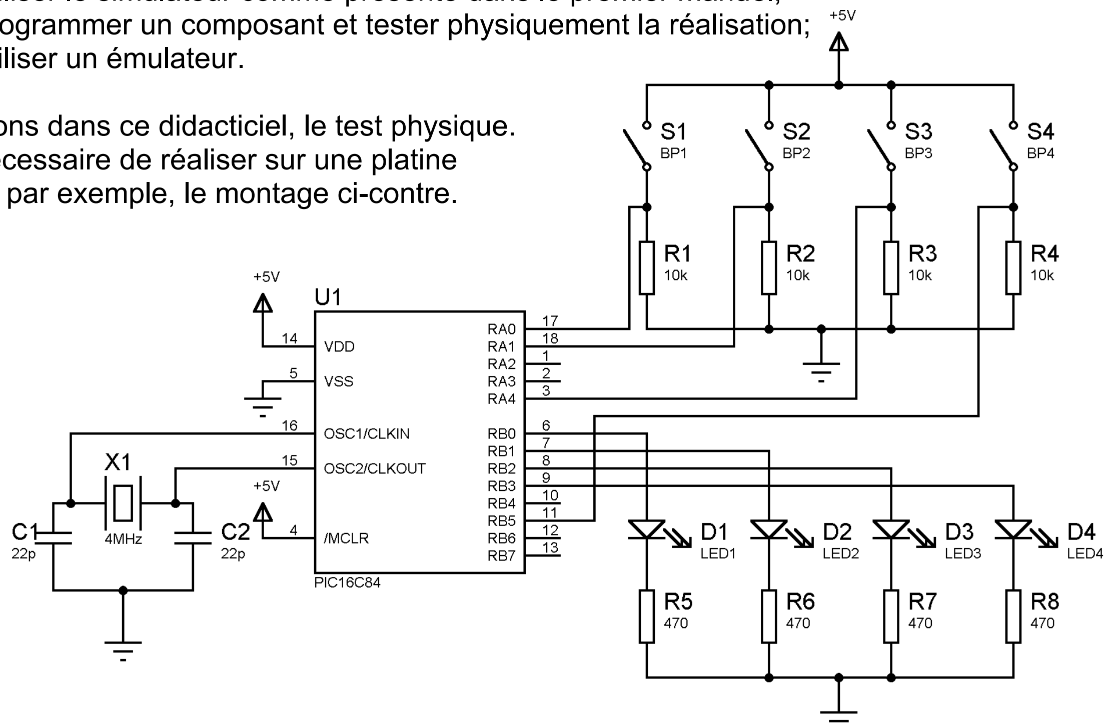
Un PIC 16F84, un quartz de 4MHz, deux condensateurs de 22pF, 4 leds rouges, 4 résistances de 470 ohms, 4 interrupteurs, 4 résistances de 10 Kohms. Une platine d'essai sous 5 volts.

La platine d'essai

Pour tester les programmes proposés, il est possible :

- -utiliser le simulateur comme présenté dans le premier manuel;
- -programmer un composant et tester physiquement la réalisation;
- -utiliser un émulateur.

Nous utiliserons dans ce didacticiel, le test physique.
Il est ainsi nécessaire de réaliser sur une platine
de type LAB, par exemple, le montage ci-contre.



But à atteindre

Ce didacticiel vous permettra de **gérer le temps** avec un PIC. Nous verrons 3 niveaux de gestion.

- Le premier niveau nous permettra de consommer du temps grâce à l'instruction nop et aux boucles.
- Le deuxième niveau nous obligera à comprendre de façon détaillée comment fonctionne la partie matérielle du PIC destinée à la gestion du temps afin de créer des temporisations précises.
- Le troisième niveau nous montrera comment gérer le temps tout en effectuant d'autres opérations.

Pour atteindre ces buts, nous utiliserons comme dans les autres didacticiels la platine d'essai de la page 4.

Avant tout ça, afin de rapidement mettre en pratique des temporisations, nous allons écrire un petit programme, qui nous permettra d'utiliser une instruction très appréciable : l'instruction "delay".

L'instruction "delay"

Il serait intéressant d'avoir des instructions qui nous permettraient de consommer du temps. Nous allons créer les instructions suivantes:

delay_10us(temps); temporisation de durée : temps * 10 microsecondes
delay_ms(temps); temporisation de durée : temps * 1 milliseconde
delay_10ms(temps); temporisation de durée : temps * 10 millisecondes
avec temps : entier entre 1 et 255.

Nous allons écrire un fichier contenant 3 fonctions correspondantes aux 3 instructions précédentes. Il suffira de déclarer le nom de ce fichier en début de programme pour pouvoir utiliser ces 3 instructions.

Ne cherchez pas encore à comprendre comment est construit ce fichier. Le but de ce chapitre est de pouvoir **utiliser rapidement des temporisations** sans connaissances sur le fonctionnement du PIC, sur les fonctions.
En allant plus loin dans ce fascicule, vous pourrez comprendre le corps des 3 procédures; leur forme vous sera accessible avec le prochain didacticiel.

1 -Création du fichier contenant le code des instructions

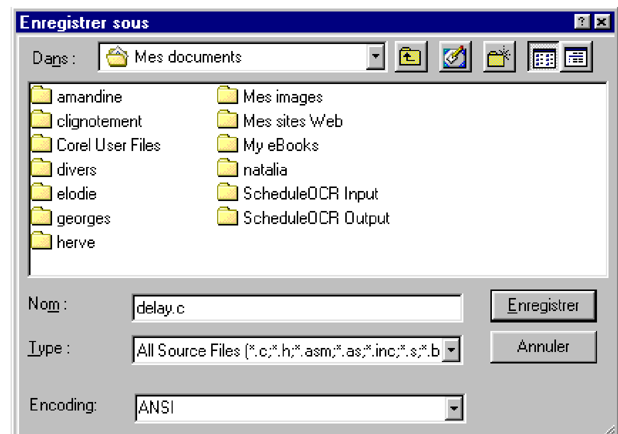
Nous allons créer un fichier dans lequel nous allons mettre le code correspondant à la résolution des 3 nouvelles instructions.

- Dans MPLAB, cliquez sur **New** du menu **File**. Tapez le texte suivant dans la nouvelle fenêtre créée.

```
/*
        delay_10us (char) : delay en multiple de 10 us pour un quartz de 4 MHz
        delay_ms (char) : delay en ms pour un quartz de 4 MHz
        delay_10ms  (char)  : delay en multiple de 10 ms pour un quartz de 4 MHz
*/
//------------------------delay en multiple de 10 us. Quartz de 4 MHz -----------------
```

```
void delay_10us (char usecs)
{
while (-- usecs > 0)
  {
  clrwdt();             // only if necessary,   nop(); a la place
  nop();
  nop();
  nop();
  nop();
  }
}

//------------------------delay en multiple de ms. Quartz de 4 MHz ----------------

void delay_ms (char millisec)
{
OPTION = 2;             // prescaler divide by 8
do
  {
  TMR0 = 0;
  clrwdt();             // only if necessary
  while (TMR0 < 125);          // 125us * 8 =1000us
  }
while (-- millisec > 0);
}

//---------------delay en multiple de 10ms. Quartz de 4 MHz, erreur 0.16 %-------------

void delay_10ms (char n)
{
char i;
OPTION = 7;             //prescaler to 256
do
  {
  clrwdt();             // only if necessary
  i =TMR0 +39;                 // 256us * 39 =10 ms
  while (i != TMR0);
  }
while (--n > 0);
}
```

- Sauvegarder ce fichier grâce à la commande **File>Save As…**



- Fermer ce fichier.

2 - Création du programme

Nous allons maintenant écrire un programme qui nous permettra de faire clignoter la led1 avec une période de 2 secondes (1seconde allumée et une seconde éteinte).

- Tapez le texte suivant dans le fichier clignotement.C

```
// Attention de respecter les majuscules et minuscules

// -----------------------------Déclaration de fichier externes--------------
#include "delay.c"              // déclaration du fichier contenant les temporisations

//--------------------------------E/S---------------------------------------------
char sortie @ PORTB;
bit led1 @ RB0;
//-----------------------------Variables generales------------------------------
char etat_inters;
char a;
//------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontrôleur
        TRISB = 0b11110000;

        for (;;)  // La suite du programme s'effectue en boucle
                        {
                        led1=!led1;
                        delay_10ms(100);
                        }
}
```

- Essayez ce programme avec la platine d'essai

L'intérêt de cette méthode est de pouvoir utiliser une instruction simple pour effectuer une temporisation. L'inconvénient est le manque de précision de la temporisation, ainsi que l'impossibilité de réaliser une opération pendant la temporisation.

Le "nop();"

Il existe une instruction pour les PIC qui consomme 4 périodes d'oscillation du quartz. Pour notre platine d'essai avec un quartz de 4Mhz, cette instruction consomme donc **1µs**.

En C, il suffit d'écrire **nop ();**

Ex : temporisation de 3µs

```
nop();
nop();
nop();
```

Cette méthode est la plus simple à mettre en œuvre. Elle convient parfaitement pour des petites temporisations (moins de 10µs).

La boucle

Pour réaliser une temporisation un peu plus longue, le plus simple est de réaliser une itération avec rien, un ou plusieurs "nop();" comme corps de la boucle. C'est cette méthode qui a été utilisée pour la création de l'instruction "delay_10us( );"

Attention : la boucle elle-même prend du temps à se réaliser. Il y a ainsi trois étapes pour utiliser cette méthode:

- 1 - écriture d'une boucle simple ;
- 2 - recherche du temps mis par cette boucle ;
- 3 – réglage final de la boucle.

Nous allons chercher à réaliser une boucle d'une durée approximative de 1ms. Pour cela, nous allons réaliser les 3 étapes précédentes.

1 - Ecriture d'une boucle simple

La boucle la plus simple à écrire en C est :

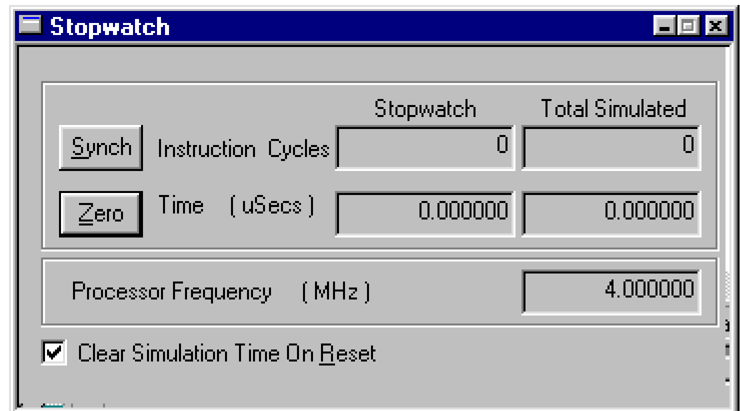
for (temps=0;temps<100;temps++) ;

Le corps de la boucle est vide, car il n'y a rien avant le point virgule. Cette boucle ne va donc rien faire d'autre qu'incrémenter la variable temps de 0 jusqu'à 100.

Ecrire le programme suivant qui est constitué d'un "nop();", d'une boucle de durée encore inconnue, puis d'un autre "nop();"

```
// Attention de respecter les majuscules et minuscules
//--------------------------------E/S et variables generales--------------------
char sortie @ PORTB;
char temps;
//------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;

        for (;;)  // La suite du programme s'effectue en boucle
                        {
                        nop();
                        for (temps=0;temps<100;temps++) ;
                        nop();
                        }
}
```

2 - Recherche du temps mis par cette boucle



- Vérifier que l'horloge est de 4 Mhz (voir fascicule 1)
- Ouvrez la fenêtre Debugger>Stopwatch

La ligne "Instruction Cycles" nous informe sur le nombre de *cycle instruction*[1].
La ligne "Time (usecs) nous informe sur l'écoulement du temps en micro-seconde.
La colonne Total simulated nous informe sur le nombre de cycles d'instruction et le temps passé depuis le début du programme.
La colonne Stopwatch nous informe sur le nombre de cycles d'instruction et le temps passé depuis la dernière action sur le bouton "Zero"

- Cliquez sur l'icône "Build", puis sur l'icône "Reset".
- Faites un Run to cursor sur la boucle " for (temps=0;temps<100;temps++) ;"
  La fenêtre Stopwatch vous indique alors que 8 µs s'est écoulé depuis le début du programme.
- Cliquez sur le bouton Zero de la fenêtre stopwatch afin de remettre à 0 la colonne Stopwatch.
- Faites un Run to cursor sur le deuxième "nop();"
- La fenêtre stopwatch nous indique que 70µs s'est écoulé pour la réalisation de la boucle. Nous avons donc une temporisation de 706µs.

3 - Réglage final de la boucle

- Nous recherchons une temporisation de 1ms, remplaçons la limite supérieure de 100 par le nombre 140. Nous obtenons alors une temporisation de 986 µs, notre but est atteint.

Cette méthode ne nous permet pas de dépasser quelques milli-secondes car nous sommes limités pour la borne supérieure par le nombre 255. En effet, la variable temps est de type char. Nous allons modifier notre type de variable et choisir un type de 16 bits. Ainsi notre variable temps pourra aller jusqu'à 65535. Les différentes variables seront traitées dans le volume 6.

Le programme ci-dessous est constitué d'une boucle possédant comme limite supérieure le nombre 60000.

[1] 1 cycle d'instruction correspond à 4 périodes d'horloge soit ici 1µs.

```
// Attention de respecter les majuscules et minuscules
//---------------------------------E/S et variables generales---------------------
char sortie @ PORTB;
unsigned temps : 16;
//------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;

        for (;;)  // La suite du programme s'effectue en boucle
                        {
                        nop();
                        for (temps=0;temps<60000;temps++) ;
                        nop();
                        }
}
```

Notre nouvelle temporisation fait maintenant 600ms.

Attention : il est difficile de prévoir la durée de réalisation d'une méthode de boucle, car la durée dépend du code final en assembleur et ce code final dépend du type de variable, des bornes de la boucle. Avec une borne supérieure très grande, on peut avoir une temporisation plus courte qu'avec une borne supérieure petite. Observez le tableau pour vous en convaincre.

Borne supérieure de la boucle	Durée avec une variable de 8 bits	Durée avec une variable de 16 bits	Nombre d'instructions en assembleur
100	0.706ms		7
100		1.21ms	13
63000		630 ms	16
65000		524 ms	12

Il existe des variables de 24 et 32 bits, mais elles ne sont pas disponibles avec la version d'évaluation de CC5X.

Il est possible d'augmenter la durée d'une boucle en ajoutant des nop(); dans le corps de boucle. La boucle for (temps=0;temps<63000;temps++) {nop(); nop(); nop(); nop(); nop();} dure 945ms.

Pour finir, créez un programme qui permettra de faire clignoter la led 1 avec une période d'approximativement 1 seconde (0.5 seconde allumé, 0.5 seconde éteinte). Je ne donne pas la solution, à vous de vous débrouiller.

L'intérêt de cette méthode est de ne pas utiliser le *timer*[2] afin de le réserver pour d'autres opérations plus critiques.
L'inconvénient est qu'il est difficile d'obtenir des temporisations précises, longues. De plus, chaque temporisation prend de la place en mémoire, contrairement à l'utilisation du timer. Il est aussi impossible de réaliser des opérations pendant la temporisation puisque le PIC est en train de traiter une boucle.

[2] Ressource du PIC permettant de gérer le temps de façon précise tout en effectuant d'autres opérations.

Le timer

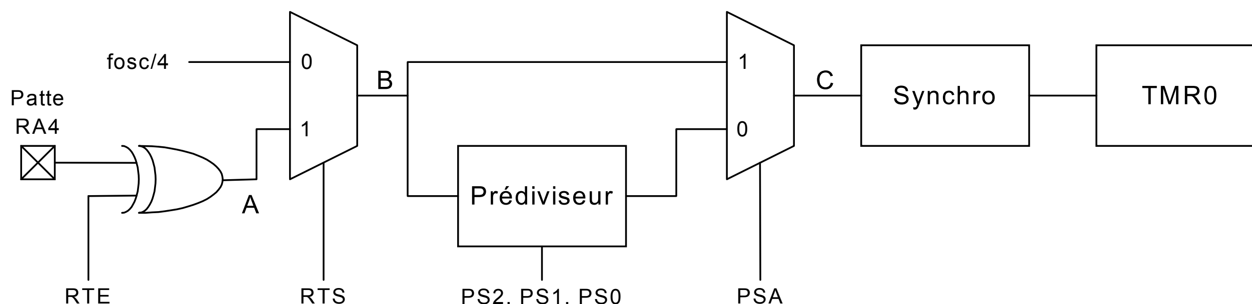
Nous arrivons au traitement de la partie la plus complexe du PIC : le timer. **Le timer est un compteur de cycles d'instructions**. Il suffit de lire régulièrement ce compteur pour évaluer le temps qui s'écoule. Il est également possible d'initialiser ce compteur avec le nombre de notre choix. Il faut aussi noter que ce compteur doit être configuré pour fonctionner.

1 - Constitution du timer.

Le timer est constitué de :

- **le TMR0**. C'est un registre de 8 bits, qui compte les *front montant*[3] du signal qu'on lui fournit. Arrivé à 255, il repasse à 0. Il est accessible en lecture ou écriture de la même façon que le PORTA, le PORTB ou les variables.
- **le prédiviseur**. Il divise la fréquence du signal qu'on lui fournit par une constante qu'il est nécessaire de programmer. Ex : un signal de période 1us divisé par 32 donnera un signal de période 32µs.
- **les aiguillages**. Ils permettent de définir le chemin qu'emprunteront les informations (sortie des aiguillages en B et C).
- **les bits de configuration** (RTE, RTS …). Ils permettent de définir les différentes fonctions.
- **la synchro**. Retarde le début du fonctionnement de 2 cycles d'instruction lorsqu'un bit relatif au timer est modifié. Cette fonction est due à l'architecture du timer et ne peut être contournée.

Voici le schéma du timer. Pas d'affolement, nous allons l'expliquer.



- A gauche, sont notées les entrées du timer. Elles sont au nombre de deux : le signal du quartz divisé par 4 (soit un signal de période 1us) et la patte RA4.
- L'entrée RA4 attaque une porte "ou exclusif" qui fournit en A un signal dépendant du bit RTE selon le tableau ci-contre.
  Cette fonction permet d'inverser les fronts montant et les fronts descendants[4] si RTE=1.

RTE = 0	RTE = 1
A = RA4	A = !RA4

- Le signal en B dépend de l'aiguillage commandé par RTS. Nous avons donc grâce à RTS le choix de la source d'entrée du timer.

RTS = 0	RTS = 1
B = fosc/4	B = A

- Le signal en C dépend de l'aiguillage commandé par PSA. Nous choisissons donc avec PSA d'utiliser ou non le prédiviseur.

PSA = 0	PSA = 1
C = Prédiviseur	C = B

- Le prédiviseur divise la fréquence du signal présente en B par un nombre calculé grâce à PS2, PS1, PS0. et donné dans le tableau page suivante.
- La synchro retarde de 2 cycles d'instruction le début du comptage.
- TMR0 compte les fronts montants.

[3] Un front montant est le passage de l'état 0 à l'état 1 d'une entité.
[4] Un front descendant est le passage de l'état 1 à l'état 0 d'une entité.

Ex : En positionnant PS2, PS1 et PS0 à 1, le taux de prédivision est de 256. Un signal en entrée du prédiviseur de période 1µs, donnera en sortie un signal de période 256 µs.

PS2	PS1	PS0	Taux du prédiviseur
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 - Configuration du timer

La configuration du timer se fait par la configuration des bits vus précédemment. Ces bits sont accessibles par le registre appelé OPTION.

Registre OPTION

	7	6	5	4	3	2	1	0
			RTS	RTE	PSA	PS2	PS1	PS0
Reset[5]	1	1	1	1	1	1	1	1

Remarques :
- Les fonctions des deux bits de gauche de ce registre ne sont pas données. Ces deux bits sont utilisés pour d'autres ressources du PIC.
- Les états du registre option sont donnés lors du reset (ligne Reset). Dans cette configuration, TMR0 compte les fronts descendants de la patte RA4.

3 - Utilisation du timer.

L'utilisation du timer est assez simple et se fait en 3 étapes.
- **mise à jour du registre OPTION**
- **mise à jour du TMR0**
- **lecture du TMR0**

Remarques :
- A chaque mise à jour du registre TMR0 ou d'un bit de configuration, 2 cycles d'instructions sont nécessaires avant le commencement du comptage.
- La lecture d'un bit par contre ne perturbe pas le comptage.

4 – Exemples de programmes.

Notre premier exemple sera très original puisque nous allons faire clignoter la led 1 avec une période de 2 secondes (1 seconde allumée, 1 seconde éteinte).

Nous configurerons le timer de la façon suivante :
- source : fosc/4
- prédiviseur : 256.

Avec une horloge de 4Mhz, TMR0 s'incrémentera toutes les 256µs. Pour obtenir cette configuration, nous mettrons le registre OPTION à 0b11000111 (le bit 4 peut être indifféremment positionné à 0 ou 1. Les bits 6 et 7 sont laissés à leur état au reset).

Pour réaliser notre temporisation de 1 seconde, nous allons :
- attendre que le registre TMR0 initialement à 0 atteigne 250 (il se sera donc écoulé 256µs*250 = 64 ms) pour incrémenter une variable temps initialement à 0.

[5] Mise sous tension du pic ou action sur la patte MCLR

- attendre que la variable temps atteigne 16 (il se sera écoulé 64ms * 16 = 1.024 seconde).
- mettre à jour la led 1.

```
// Attention de respecter les majuscules et minuscules
//--------------------------------E/S et variables generales--------------------
char sortie @ PORTB;
bit led1 @ RB0;
char temps;
//------------------------------Fonction principale----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;
        OPTION = 0b11000111;       //prediviseur à 256    entrée : clock/4
        temps = 0;
        TMR0 = 0;
        for (;;)                   // La suite du programme s'effectue en boucle
         {
                if (TMR0 == 250) { ++temps; TMR0 = 0; }            // 64ms sont passes
                if (temps == 16) { led1=!led1; temps = 0; }          // 1,024 seconde est passe
         }
}
```

Je vous laisse tester et comprendre complètement ce programme.
Si vous voulez observer la temporisation avec la fenêtre stopwatch et les "run to cursor", il faudra écrire le dernier if de la façon suivante : `if (temps == 16) {`
`led1=!led1; temps = 0; }`
et faire un 'run to cursor' sur la deuxième ligne. En effet, l'écriture sur une ligne provoque l'arrêt de la simulation à chaque exécution du if, soit 16 fois par seconde.

Voici une deuxième configuration du timer qui va répondre au même clignotement sans aucune erreur sur la temporisation. Nous configurerons le timer de la façon suivante :

- source du timer :fosc/4;
- prédiviseur : 64.

Avec une horloge de 4Mhz, TMR0 s'incrémentera toutes les 64µs. Pour obtenir cette configuration, nous mettrons le registre OPTION à 0b11000101 (le bit 4 peut être indifféremment positionné à 0 ou 1. Les bits 6 et 7 sont laissés à leur état au reset).

Pour réaliser notre temporisation de 1 seconde, nous allons :

- incrémenter une variable temps à chaque incrémentation du TMR0 (donc toutes les 64µs);
- attendre que la variable temps atteigne 15625. Il se sera écoulé exactement 1 seconde;
- mettre à jour la led 1.

Afin d'éviter une quelconque erreur, il ne faudra jamais mettre à jour TMR0 ou un bit de configuration (voir le problème de la fonction synchro).

Nous avons donc besoin d'une variable temps de 16 bits et d'une autre variable afin de détecter l'incrémentation de TMR0.Nous l'appellerons newtmro.

```
// Attention de respecter les majuscules et minuscules
//--------------------------------E/S et variables generales--------------------
char sortie @ PORTB;
bit led1 @ RB0;
unsigned temps : 16;
```

```
char newtmro;
//------------------------------Fonction principale-----------------------------
void main(void)
{
  sortie = 0;                   // Initialisation des pattes du microcontroleur
  TRISB = 0b11110000;
  OPTION = 0b11000101;     //prediviseur à 64    entrée : clock/4
  temps = 0;
  newtmro = TMR0+1;
  for (;;)                             // La suite du programme s'effectue en boucle
   {
      if (TMR0 == newtmro) { ++temps; ++newtmro; }              // 64us sont passées
      if (temps == 15625) {led1=!led1; temps = 0; }              // 1 seconde est passée
   }
}
```

Je vous laisse tester et comprendre complètement ce programme.

Voici un dernier exemple qui met en avant l'intérêt du timer sur la fonction delay.

- Le bouton poussoir 1 allume la led 1 et éteint les autres.
- Le bouton poussoir 2 allume la led 2 et éteint les autres.
- Le bouton poussoir 3 allume les leds 1, 2 et 3.
- Le bouton poussoir 4 éteint les leds. 1, 2 et 3.
- Dans le cas où aucun ou plusieurs interrupteurs sont actionnés, rien ne se produit.
- La led 4 clignote avec une période de 2 secondes.

Nous avons donc le clignotement de la led 4 en même temps que d'autres actions.

Voici le programme correspondant.

```
// Attention de respecter les majuscules et minuscules
//--------------------------------E/S et variables generales--------------------
char sortie @ PORTB;
bit inter1 @ RA0;
bit inter2 @ RA1;
bit inter3 @ RA4;
bit inter4 @ RB5;
bit led1 @ RB0;
bit led2 @ RB1;
bit led3 @ RB2;
bit led4 @ RB3;

//----------------------------Variables generales-------------------------------
char etat_inters;
char temps;
//------------------------------Fonction principale-----------------------------
void main(void)
{
        sortie = 0;                     // Initialisation des pattes du microcontroleur
        TRISB = 0b11110000;
        OPTION = 0b11000111;  //prediviseur à 256    entrée : clock/4
        temps = 0;
        TMR0 = 0;
        etat_inters=0;
        for (;;)  // La suite du programme s'effectue en boucle
          {
```

```
        etat_inters.0=inter1;  // Mise de l'etat des inter dans la variable etat_inters
        etat_inters.1=inter2;
        etat_inters.2=inter3;
        etat_inters.3=inter4;
        switch (etat_inters){
            case 1:           //action sur inter1 uniquement
                led1=1;
                led2=0;
                led3=0;
                break;
            case 2:           // action sur inter2 uniquement
                led2=1;
                led1=0;
                led3=0;
                break;
            case 4:           // action sur inter3 uniquement
                led1=1;
                led2=1;
                led3=1;
                break;
            case 8:           // action sur inter4 uniquement
                led1=0;
                led2=0;
                led3=0;
                break;
                }
        if (TMR0 == 250) { ++temps; TMR0 = 0; }        // 64ms sont passés
        if (temps == 16) { led4=!led4; temps = 0; }    // 1,024 seconde est passé
    }
}
```

Pour réaliser un programme avec cette méthode, il faut faire attention à plusieurs choses.

- Le programme doit obligatoirement exécuter la ligne if (TMR0 == … régulièrement.
- Il est donc interdit d'utiliser une boucle longue.
- La durée maximale de scrutation de tout le programme autorisée est de 64 ms. En effet, toutes les 64 ms, il est indispensable de réaliser la ligne if (TMR0 == …. La précision de la temporisation est donc ici au minimum de 64 ms.

Reprenez maintenant le code correspondant à la résolution des 3 instructions delay en début du fascicule. Vous êtes enfin capable de le comprendre. Nous aborderons l'instruction clrwdt(); un peu plus loin.

L'intérêt du timer est d'être précis, de permettre d'effectuer des opérations pendant la temporisation. Son inconvénient est d'être difficile à programmer, nécessite la réalisation de comparaison régulièrement pendant le programme. Il rend donc difficile l'utilisation de boucle longue. Pour palier au dernier inconvénient, il sera nécessaire d'avoir recours aux interruptions, notion qui sera abordée dans le prochain fascicule.

Récapitulatif

Le tableau ci-dessous récapitule les utilisations classiques de chaque méthode.

	nop();	boucle	delay	timer	timer + interruption
Temporisation très courte	X				
Temporisation sans le timer		X			
Temporisation peu précise, facile à mettre en œuvre			X		
Temporisation précise				X	
Temporisation précise, accompagnée de boucles longues					X

Le timer du PIC 16F84 correspond à la structure la plus répandue chez les PIC. Il existe aussi des timers 8 bits plus complexes, des timers 16 bits. Certains pic ont plusieurs timers. Il est donc nécessaire en fonction de la complexité de l'application à réaliser de choisir le pic le plus approprié. Il sera alors indispensable de bien lire la documentation du pic afin de pouvoir le programmer.

Je vous conseille de regarder la doc du PIC16F627 ou 16F628. Ces deux pic sont compatibles broche à broche avec le 16F84, les programmes du 16F84 sont utilisables directement avec les 16F627 et 16F628 à condition de rajouter une instruction supplémentaire en début de programme de façon à désactiver la ressource appelée "comparateurs analogiques".

L'avantage de ces PIC est de posséder un timer 8 bits pour les petites temporisations, un timer 16 bits pour les longues temporisations, un timer 8 bits complexes pour des horloges particulières. Il est aussi possible de générer automatiquement grâce aux deux derniers timer des signaux avec des périodes variables, des *rapports cycliques*[6] variables.

[6] Temps à l'état haut divisé par la période (permet de quantifier combien de temps le signal reste à l'état haut pendant la durée d'une période).

Le chien de garde

On retrouve le chien de garde sous le nom **watchdog timer** dans toutes les documentations de PIC. Le chien de garde est utilisé pour faire un reset (le programme recommence au début, certaines ressources sont mises à des états déterminés) du PIC. Cette fonction, à première vue sans intérêt pour un débutant, est en réalité principale.

Dans un microcontrôleur, donc un PIC, les causes de dysfonctionnement d'un programme sont nombreuses même lorsque celui-ci est sans erreur :

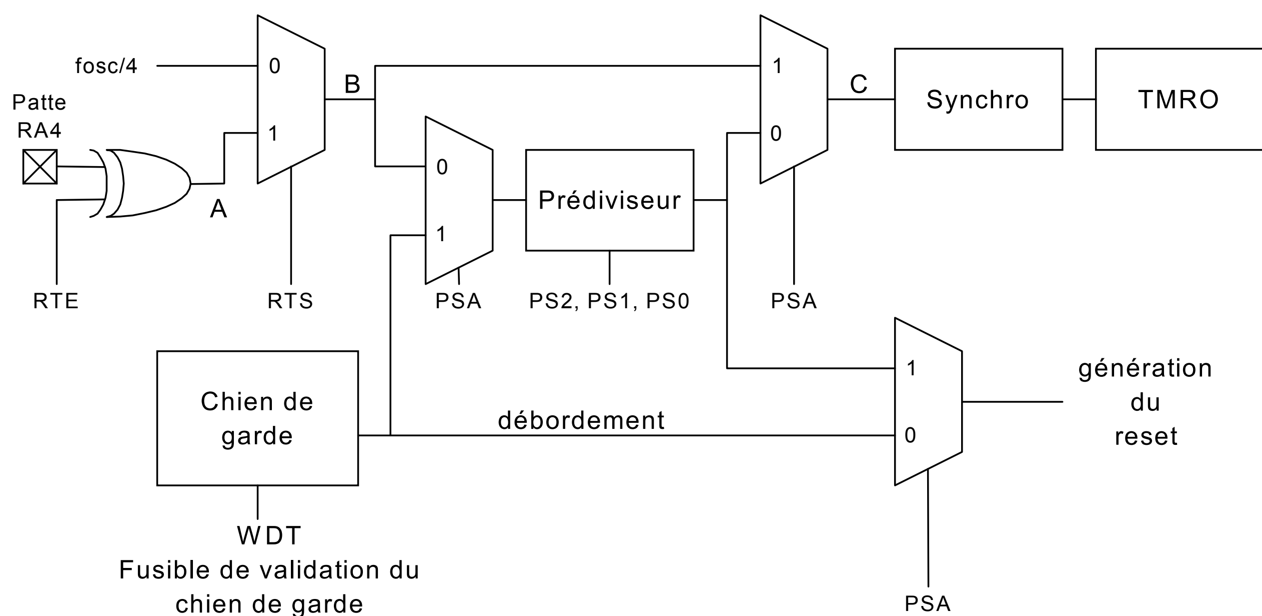
- coupure courte d'alimentation modifiant un registre,
- champs électromagnétique modifiant un registre,
- action imprévue sur la patte reset (cause mécanique électromagnétique),
- choc mécanique, modifiant un registre,
- surtension due à la mise en fonction d'un appareil proche, d'un composant …

Vous savez bien que lorsqu'un appareil électronique ne fonctionne pas correctement, il suffit de le débrancher, puis de le remettre en fonction pour obtenir le fonctionnement attendu. Personne ne s'explique le disfonctionnement, et bien il en est de même pour les PIC. Le chien de garde permet un reset du PIC lorsque certains défauts apparaissent.

Le chien de garde correspond à un **timer particulier**.

- Celui-ci est connecté à une **horloge interne**, indépendante du quartz.
- Il **compte** en permanence et **au moment du *débordement***[7], **fait un reset du microcontrôleur**.
- Afin d'**éviter le débordement** il est indispensable de le **remettre à 0** grâce à l'instruction **clrwdt();**.
- Le **temps de débordement est typiquement de 18ms**. Il est **possible d'allonger ce temps** en utilisant le prédiviseur du timer. Le prédiviseur n'est alors plus utilisable par le timer.
- **Il est aussi possible de désactiver cette fonction** grâce à un fusible accessible au moment de la programmation.

Ci-dessous le schéma du timer et du chien de garde.



[7] Passage de FF à 0 du chien de garde.

Le bit PSA détermine si le prédiviseur est affecté au timer ou au chien de garde

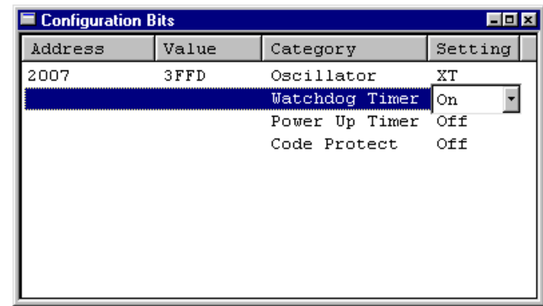
- PSA = 0, le prédiviseur est utilisé pour le timer, le chien de garde génère un reset au bout de 18ms.
- PSA = 1, le prédiviseur est affecté au chien de garde, le taux de prédivision est donné dans le tableau ci-dessous (différent du taux pour l'affectation au timer).

PS2	PS1	PS0	Taux du prédiviseur
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

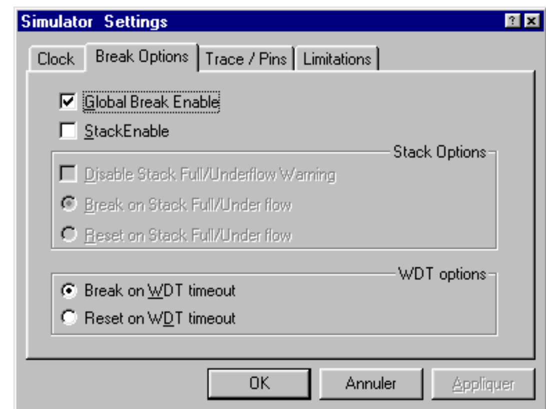
Reprenez maintenant le code correspondant à la résolution des 3 instructions delay en début du fascicule. Vous pouvez maintenant comprendre l'utilité des instructions clrwdt();

L'exemple ci-dessous est identique à celui permettant le clignotement de la led 1 avec une période exacte de 1 seconde, plus l'utilisation du chien de garde en sécurité. Si pour une raison inexpliquée l'instruction clrwdt(); ne se réalise pas toutes les 18ms au maximum, le programme recommence à 0.

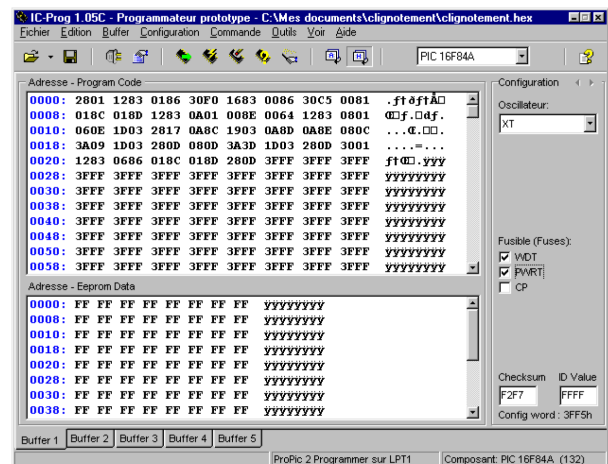
```
// Attention de respecter les majuscules et minuscules
//--------------------------------E/S et variables generales--------------------
char sortie @ PORTB;
bit led1 @ RB0;
unsigned temps : 16;
char newtmro;
//-----------------------------Fonction principale----------------------------
void main(void)
{
  sortie = 0;                  // Initialisation des pattes du microcontroleur
  TRISB = 0b11110000;
  OPTION = 0b11000101;     //prediviseur à 64    entrée : clock/4
  temps = 0;
  newtmro = TMR0+1;
  for (;;)                              // La suite du programme s'effectue en boucle
   {
       clrwdt();
       if (TMR0 == newtmro) { ++temps; ++newtmro; }             // 64ms sont passés
       if (temps == 15625) {
               led1=!led1; temps = 0; }                  // 1 seconde est passé
   }
}
```



- Tapez ce programme sous MPLAB.
- Ouvrez la fenêtre Configure>Configuration Bits.
- Mettre watchdog Timer à 'on' pour la simulation.
- Simuler ce programme, la led1 clignote.
- Enlevez l'instruction clrwdt();
- Compiler, faites un run to cursor sur la ligne led1=!led1; temps = 0; }.La simulation n'arrive jamais sur cette ligne puisque toutes les 18ms le chien de garde déborde.



- Il est possible d'arrêter la simulation au moment du débordement du chien de garde. Pour cela ouvrez la fenêtre Debugger>Settings…, cliquez sur l'onglet Break Options et configurez comme ci-contre.
- Simuler votre programme. Celui-ci s'arrête au moment du débordement, donc au bout de 18ms.
- Rajoutez l'instruction clrwdt();



- Pour programmer le PIC il est obligatoire d'autoriser le watchdog timer. Pour cela, avant de lancer la programmation, validez la coche "WDT".
- Tester votre PIC sur la platine LAB.

Conclusion

Avec ce quatrième didacticiel, vous êtes enfin capable de parfaitement gérer le temps. Vous pouvez maintenant concevoir n'importe quel programme. avec un PIC 16F84.

Afin de programmer efficacement, il vous manque toutefois quelques notions. La notion de fonction, primordiale en C, sera abordée dans le prochain fascicule. Nous verrons aussi comment générer des interruptions afin d'utiliser, entre autres, le plus efficacement possible le timer.