

INFO2 : INFORMATIQUE EMBARQUEE

TP2 : Mise en œuvre du timer 0 – Simulation

1 Objectifs

Le PIC18F4520 possède 4 **timers**. Ces périphériques permettent, entre autres choses, de générer des durées précises. Lors de cette séance, nous allons nous familiariser avec l'utilisation du timer 0 du PIC18F4520. Nous le configurerons d'abord en **mode 8 bits** puis en **mode 16 bits**.

Afin de valider notre réflexion, plutôt que de programmer directement le composant et de vérifier s'il fonctionne comme attendu, nous effectuerons préalablement des simulations sous MPLAB X. Si les résultats de la simulation sont satisfaisants, alors le projet sera téléchargé dans la mémoire de programme. Simuler est une étape préalable quasi obligatoire lors du développement d'un projet ; grâce à l'absence de manipulation matérielle lors des premières phases de développement, la simulation permet un gain de temps et d'argent important.

Nous utiliserons enfin les fonctions de la bibliothèque de gestion des timers décrites dans **timers.h**.

2 Projet TP2A : prise en main du timer 0 en mode 8 bits. Simulation

2.1 Cahier des charges et objectifs

Toutes les 10 ms, inverser l'état de la LED connectée à RD0. Les aspects temporels seront gérés par le timer 0 en mode 8 bits. Le quartz est à 8 MHz.

Le projet sera uniquement simulé.

Ce projet permet de se familiariser à la génération de durée via un timer. Il sensibilise également à la simulation.

Par rapport à l'utilisation des fonctions **logicielles** de **delays.h** (cf. TP1), on utilise ici un véritable composant **matériel** pour déclencher des événements à intervalle régulier. L'intérêt est le suivant : Pendant qu'une fonction de **delays.h** s'exécute, le processeur ne fait rien (longue suite d'instructions assembleur **NOP** - *No Operation*), ce qui n'est pas très optimisé. Au contraire, quand on utilise un timer et le concept d'**interruption** vu plus tard dans le semestre, le processeur peut faire autre chose que d'attendre la fin de la suite de **NOP**.

2.2 Quelques éléments

- Il y a lieu de créer un projet de nom **TP2A**. Au moment d'éditer les caractéristiques du projet, quand vous en êtes à **Select Tool**, choisissez **Simulator** plutôt que l'habituel programmeur **PicKit3** :



- Le fichier **TP2A.c** aura la structure des fichiers C du premier TP. En voici un exemple possible :

```
#include <p18f4520.h>
#include "../configuration_bits.h"

void main(void)
{
    TRISDbits.TRISD0 = 0;    // RD0 en sortie
    T0CON = 0xC6;            // pré-compteur = 128, mode 8 bits, timer on, horloge interne
    INTCONbits.TMR0IF = 0;    // On met à 0 le drapeau.
    TMR0L = 100;             // On charge la valeur initiale du timer.

    while(1)
    {
        if(INTCONbits.TMR0IF == 1)                // Si la durée s'est écoulée :
        {
            INTCONbits.TMR0IF = 0;                // On remet à 0 le drapeau.
            TMR0L = 100;                          // On recharge la valeur initiale du timer.
            LATDbits.LATD0 = !LATDbits.LATD0;     // On change l'état de la LED.
        }
    }
}
```

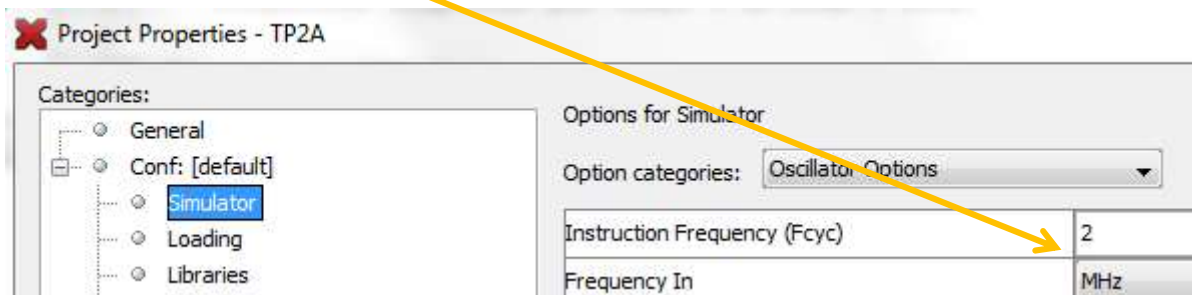
2.3 Simulation

2.3.1 Prise en main

Le but de la simulation est ici de vérifier que l'inversion de l'état de la LED connectée à RD0 s'exécute bien toutes les 10 ms comme l'impose le cahier des charges.

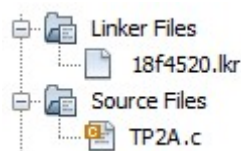
Dans les paramètres du simulateur, il y a lieu de préciser la **fréquence de l'oscillateur** qui sera réellement utilisé ou plutôt la **fréquence d'exécution des instructions**. Il y a un rapport simple entre ces deux nombres : il faut 4 périodes de l'oscillateur pour exécuter chaque instruction (cette durée s'appelle un *cycle*). L'oscillateur de la carte d'évaluation est un quartz à 8 MHz. Les instructions sont donc exécutées à la fréquence de 2 MHz (une instruction toutes les 500 ns). On peut dire qu'ici le microcontrôleur travaille à 2 Mips (*Mega instructions per second*).

- Effectuez le réglage de ce paramètre du simulateur par File / Project Properties et réglez les paramètres comme ci-dessous :




- Quand le programme est saisi, cliquez sur l'icône Build Project . Passez à l'étape suivante s'il n'y a pas d'erreur.

Remarque : Dans la zone Projects, vous devez avoir quelque chose comme ceci :




- Lancez le débogueur en cliquant sur l'icône Debug Project . On peut voir écrit **User program running** dans l'onglet Debugger Console. C'est la boucle infinie commençant par le **while(1)** qui est en train de s'exécuter. Vous


pouvez suspendre l'exécution en cliquant sur Pause . Dans ce cas, on voit une petite flèche verte indiquant l'instruction que le microcontrôleur s'**apprête à exécuter** (pas celle qui vient d'être exécutée !) vue la valeur hébergée dans le PC (*Program Counter*).

```

19     while(1)
20     {
21         if(INTCONbits.TMROIF == 1) // Si la dur
22         {
23             INTCONbits.TMROIF = 0; // On remet
24             TMR0L = 100;           // On recha
25             LATDbits.LATD0 = ! LATDbits.LATD0;
26         }
27     }

```


- Vous pouvez reprendre l'exécution en cliquant sur **Continue** .

- Vous pouvez aussi, quand l'exécution est en pause, cliquer sur **Reset** . Dans ce cas le PC se placera de façon à exécuter la première instruction du programme, c'est à dire celle-juste après **void main(void)** :

```

12     void main (void)
13     {
14         TRISDbits.TRISD0 = 0;

```


- Il est possible de faire du **pas à pas** grâce à **Step Into** . Après un **Reset**, faites plusieurs **Step Into** de suite pour suivre l'évolution du PC. Pour observer le PC, contentez-vous d'observer le SFR de nom PCL, l'octet de poids faible du PC. Les SFR sont visualisables par : **Window / PIC Memory Views / SFRs**.
- Vous allez ensuite rester coincé un bon moment comme ci-dessous. C'est normal, **TMROIF**, le drapeau de fin de comptage du timer 0, ne se levant pas très souvent, la condition du **if** est souvent fausse et, étant dans une boucle infinie, on se retrouve au même endroit.

```

if(INTCONbits.TMROIF == 1)

```

2.3.2 Mesure de la durée séparant deux changements d'état de la led

- Quittez le mode débogage / simulation en cliquant sur .
- Pour voir combien de temps se passe entre deux instructions commandant le changement d'état de la led :


```
LATDbits.LATD0 = ! LATDbits.LATD0;
```

placez un point d'arrêt (*breakpoint*), juste avant son exécution en cliquant (simple clic) sur le numéro de ligne lui correspondant. Vous devez avoir quelque chose comme ceci :


```


24     TMR0L = 100;           // On recha
25     LATDbits.LATD0 = ! LATDbits.LATD0;
26 }


```

- Relancez le débogueur en cliquant sur l'icône **Debug Project** . Effectuez un **Reset** et un **Continue**. Le programme démarre depuis le début et se fige à la rencontre du premier point d'arrêt :

```
LATDbits.LATD0 = ! LATDbits.LATD0;
```

- Mettez en place un chronomètre (*stopwatch*) par **Window / Debugging / Stopwatch** et remettez-le à zéro en cliquant sur l'icône **Clear stopwatch**  de l'onglet **Stopwatch**.

- Faites **Continue** . Le programme reprend jusqu'à rencontrer à nouveau le point d'arrêt. La *stopwatch* indique la période du changement d'état de la led. Vérifiez que celle-ci est cohérente avec le cahier des charges.

Remarque : Quand vous avez un programme comportant des appels à des fonctions (ce n'est pas le cas ici) et que vous ne souhaitez pas rentrer dans l'exécution pas à pas de cette fonction, vous pouvez remplacer le **Step Into** par un **Step Over** . L'appel de la fonction est vu alors comme un seul pas.

Une temporisation de 10 ms pour faire clignoter une led est trop brève. Dans le projet suivant, nous allons voir comment obtenir des durées plus longues.

3 Projet TP2B : prise en main du timer 0 en mode 16 bits.

3.1 Cahier des charges et objectifs

Toutes les 500 ms, inverser l'état de la LED connectée à RD0. Les aspects temporels seront gérés par le timer 0 en mode 16 bits. Le quartz est à 8 MHz.

Le projet sera uniquement simulé.

Ce projet permet de maîtriser le timer 0 également en mode 16 bits.

- Quelle durée maximale peut être réalisée avec le timer 0 en mode 8 bits (oscillateur à 8 MHz) ? Justifier alors l'utilisation du mode 16 bits.
- Donner les valeurs des registres T0CON, TMR0H et TMR0L répondant au cahier des charges
- Créer un nouveau projet **TP2B** afin de répondre au cahier des charges.
- Simuler la durée de la période générée. Est-ce satisfaisant ?

3.2 Utilisation des fonctions de la bibliothèque timers.h

Quand on regarde le code qui permet de configurer le timer 0, notamment l'affectation des registres T0CON, TMR0H et TMR0L, celui-ci n'est pas des plus explicites. En outre, dès que l'on veut changer les caractéristiques du timer (pré-compteur, mise en marche, etc.), il faut recalculer la valeur des registres.

Avec le compilateur C18 est fournie une bibliothèque (*library*) de fonctions de gestion des timers. Ouvrir le document **MPLAB_C18_Libraries_51297f.pdf** et trouver les pages concernant les 4 fonctions relatives au timer 0.

- Remplacer dans le code (en mettant en commentaires l'ancien code), les lignes d'affectation de T0CON, TMR0H et TMR0L en utilisant des fonctions de la bibliothèque timers.
- Ajouter en en-tête de fichier l'inclusion permettant d'utiliser les fonctions de la bibliothèque timers.
- Simuler la durée générée. Est-ce bien la même qu'auparavant ?
- Par Window / PIC Memory Views / SFRs, vérifier que le registre T0CON a la même valeur quand il a été initialisé par la fonction de la bibliothèque **OpenTimer0()** que quand vous l'aviez initialisé par vous-même directement.
- Quitter le mode simulation et programmer une carte EasyPIC v7 avec le programme développé. Dire si tout fonctionne correctement.

4 Projet TP2C : Chronomètre avec les afficheurs 7 segments

- Élaborez un chronomètre au centième de seconde qui ne retarde pas comme indiqué à la fin de l'énoncé du TD4.