

INFO2 : INFORMATIQUE EMBARQUEE

TP1 : Prise en main – Entrées/Sorties Tout Ou Rien (TOR)

1 Objectifs des TP du module INFO2 et de ce TP en particulier

L'objectif des TP du module INFO2 est de prendre en main un microcontrôleur 8 bits, le **PIC18F4520** de **Microchip**, à travers deux aspects :

- **Aspect matériel** : le microcontrôleur est testé grâce à la **carte d'évaluation EasyPIC v7** de **MikroElektronika**. Celle-ci comporte l'alimentation et l'oscillateur permettant de faire fonctionner le microcontrôleur. En outre, un grand nombre de composants (leds, boutons-poussoirs, connecteur RS232, connecteurs d'entrées/sorties, écran LCD, potentiomètre, mémoire externe, etc.) sont déjà présents sur la carte, éliminant toute nécessité de CAO et de réalisation dans le cadre de cette prise en main.
- **Aspect logiciel** : l'accent est mis sur la **programmation en langage C** du microcontrôleur puis sur l'utilisation des bibliothèques de fonctions C assurant la gestion des périphériques intégrés (entrées/sorties tout ou rien, liaison RS232, convertisseur analogique/numérique, etc.). Ces fonctions sont soit déjà écrites et documentées par Microchip, soit à écrire par vous-même. Ainsi, on minimisera la manipulation des registres de configuration des périphériques au niveau bit, ce qui permettra un gain de temps et de fiabilité.

Plus particulièrement, cette première séance permet de faire connaissance avec la carte d'évaluation et l'**environnement de développement intégré MPLAB X**, logiciel permettant de développer des projets à base de microcontrôleurs PIC. Pour information, cet IDE (*Integrated Development Environment*) est basé sur la plateforme NetBeans. Le **compilateur C** utilisé est **C18** de Microchip. Il est à destination des microcontrôleurs 8 bits de Microchip type 18Fxxxx.

Les projets à développer permettent d'illustrer la lecture et l'écriture des entrées/sorties numériques (dites aussi « tout ou rien », TOR).

2 Brève présentation du PIC18F4520

Le PIC18F4520 est un microcontrôleur, ce qui signifie qu'il comporte dans une seule puce :

- une partie basique [microprocesseur – mémoire de programme – mémoire de donnée] que l'on appellera « **cœur** » ;
- mais également tout un ensemble de **périphériques**.

Tout étant dans la même puce, il n'y a pas à prévoir l'interconnexion des différents sous-ensembles entre eux comme dans un système plus conséquent (micro-ordinateur par exemple).

2.1 Cœur du microcontrôleur

Source(s) bibliographique(s) de cette partie : **39631E.pdf** (*datasheet* du PIC18F4520), **chapitre 5**.

2.1.1 Microprocesseur

Le microprocesseur est de type 8 bits. Son architecture est de type Harvard.

2.1.2 Mémoire de programme (*program memory*)

La mémoire de programme est organisée en octets chacun référencé par une adresse unique.

Le bus d'adresse de la mémoire de programme est sur 21 bits¹. Par conséquent, 2^{21} octets = 2 Mo sont adressables. Dans les faits, seuls les premiers 32768 octets = 32 Ko sont effectivement implémentés (en technologie *Flash ROM*). Les adresses de stockage des instructions vont donc de 0x0000 à 0x7FFF (=32768 - 1).

Une instruction-machine tient sur deux octets. Par conséquent, 16384 instructions sont possibles dans un programme et les adresses des instructions sont forcément paires : 0x0000, 0x0002, 0x0004, etc.

La cartographie de la mémoire de programme est donnée **Figure 5.1** de **39631E.pdf**.

¹ La taille du registre *Program Counter* est donc aussi de 21 bits.

2.1.3 Mémoire de donnée (data memory)

La mémoire de donnée est organisée en octets chacun référencé par une adresse unique.

Le bus d'adresse de la mémoire de donnée est sur 12 bits. Par conséquent, 2^{12} octets = 4 Ko sont adressables. Dans les faits, seuls les 1536 premiers octets sont effectivement implémentés (en technologie *Static RAM*). Microchip les appelle des *General Purpose Registers* (GPR) ou des *File Registers*. Leurs adresses vont donc de 0x000 à 0x5FF (=1536 - 1). Ces octets servent par exemple à héberger les variables d'un programme en langage C.

A l'autre extrémité des 4 Ko possibles figurent 128 octets que Microchip appelle des *Special Function Registers* (SFR). Leurs adresses vont donc de 0xF80 à 0xFFFF. Ces SFR permettent de configurer le cœur du microcontrôleur et les périphériques, cela sera abordé un peu plus loin.

La cartographie de la mémoire de donnée est donnée Figure 5.6 de **39631E.pdf**. Le détail de cette carte pour les SFR est donné **Table 5.1** de **39631E.pdf**.

2.2 Périphériques

2.2.1 Vue d'ensemble

Les périphériques (*peripherals*) font la richesse d'un microcontrôleur. Les plus connus sont par exemple les ports d'entrée / sortie, les convertisseurs analogiques-numériques, les liaisons série. Voici une liste non exhaustive de ceux qui équipent le PIC18F4520 :

- High-Current Sink/Source 25 mA/25 mA
- Three Programmable External Interrupts
- Four Input Change Interrupts
- Up to 2 Capture/Compare/PWM (CCP) modules, one with Auto-Shutdown (28-pin devices)
- Enhanced Capture/Compare/PWM (ECCP) module (40/44-pin devices only):
 - One, two or four PWM outputs
 - Selectable polarity
 - Programmable dead time
 - Auto-shutdown and auto-restart
- Master Synchronous Serial Port (MSSP) module Supporting 3-Wire SPI (all 4 modes) and I²C™ Master and Slave modes
- Enhanced Addressable USART module:
 - Supports RS-485, RS-232 and LIN/J2602
 - RS-232 operation using internal oscillator block (no external crystal required)
 - Auto-wake-up on Start bit
 - Auto-Baud Detect
- 10-Bit, up to 13-Channel Analog-to-Digital (A/D) Converter module:
 - Auto-acquisition capability
 - Conversion available during Sleep
- Dual Analog Comparators with Input Multiplexing
- Programmable 16-Level High/Low-Voltage Detection (HLVD) module:
 - Supports interrupt on High/Low-Voltage Detection

2.2.2 Cas particulier des ports d'entrées/sorties

2.2.2.1 Introduction

Pour par exemple lire l'état de boutons poussoirs ou imposer des états sur des leds, le PIC18F4520 possède 36 entrées/sorties numériques (ou « tout ou rien », TOR) appelées par Microchip des *General Purpose Inputs/Outputs* (GPIO). Ces GPIO sont regroupées en **ports** de 8 bits. Ces ports s'appellent port A, port B, ..., jusque port E. Le port E ne fait que 4 bits, c'est pour cela que l'on a 36 GPIO et pas 40.

La gestion d'un des cinq ports port x (x = A, B, C, D ou E) se fait par l'utilisation de trois SFR : TRISx, LATx et PORTx.

2.2.2.2 Accès aux registres de contrôle des GPIO en langage C

Prenons l'exemple de TRISD. Celui-ci est un SFR situé à l'adresse 0xF95 de la mémoire de donnée. Il y a deux problèmes :

1. Pour affecter TRISD avec une valeur 8 bits, il faudrait créer un pointeur constant vers la case d'adresse 0xF95, ce qui peut rapidement devenir pénible pour la lisibilité et la maintenance du code (il y a bien d'autres registres que TRISD à manipuler dans un programme).
2. Par ailleurs, si on ne veut modifier qu'un seul des 8 bits de TRISD, on est obligé d'utiliser des masquages. En effet le type de donnée le plus petit en langage C est l'octet.

Pour pallier à cela et donc alléger et fiabiliser l'écriture des programmes, il existe un **fichier d'en-tête** nommé **pic18f4520.h**. Ce fichier, avec l'aide du fichier **p18f4520.asm**, associe le nom symbolique des SFR (par exemple **TRISD**) à leur adresse (par exemple 0xF95) et définit aussi des **structures de bits** (vu en TD). Ainsi on peut écrire des choses comme :

```
TRISD = 0x25; // le registre TRISD prend la valeur 0x25
```

Ou :

```
TRISDbits.TRISD5 = 1 // le bit 5 du registre TRISD prend la valeur 1
```

Ce fichier doit-être inséré en en-tête de tout fichier C voulant utiliser ses services (cf. **partie 3.7**).

2.2.2.3 Comment dire qu'une GPIO est une entrée ou une sortie ?

Avant de travailler avec une GPIO, il faut préciser si celle-ci est destinée à fonctionner en entrée ou en sortie.

Au niveau du port D par exemple, c'est le SFR de nom TRISD qui permet de préciser ces directions. Pour que RD5 soit une sortie il faut que le bit 5 de TRISD soit à 0. Pour que RD5 soit une entrée, il faut que le bit 5 de TRISD soit à 1.

Ainsi, si on veut que RD7, RD6 et RD5 soient des entrées et que RD4, RD3, RD2, RD1 et RD0 soient des sorties, il faut écrire :

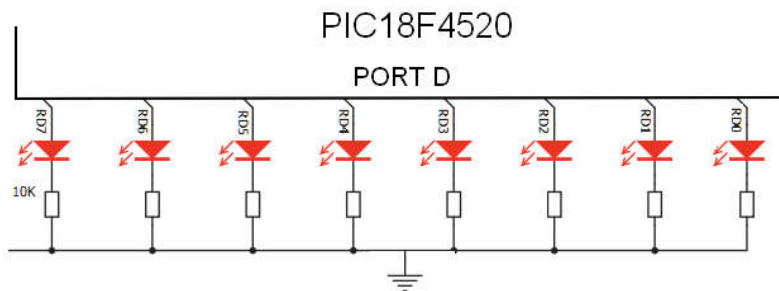
```
TRISD = 0xE0;
```

Il est également possible de préciser la direction d'une seule GPIO :

```
TRISAbits.TRISA5 = 1; // RA5 sera une entrée, le caractère entrée ou sortie des autres broches du port
// n'est pas modifié.
```

2.2.2.4 Ecriture sur un port

Sur le schéma électrique de la carte d'évaluation, on voit par exemple que 8 leds actives hautes sont connectées sur les 8 broches du port D : RD7, RD6, RD5, RD4, RD3, RD2, RD1, RD0 :



Pour imposer un niveau sur le port D, il faut utiliser le registre SFR de nom LATD. Ainsi, pour par exemple allumer les leds connectées à RD7, RD6, RD5 et RD4 et éteindre celles connectées à RD3, RD2, RD1 et RD0, il faut écrire :

```
LATD = 0xF0;
```

Il est également possible de n'imposer qu'un seul bit parmi les 8. Cela est permis au niveau du langage C par l'utilisation d'une **structure de bits**. Ainsi, pour par exemple allumer seulement la led connectée à RD2, on écrit :

```
LATDbits.LATD2 = 1;
```

Remarque : il est bon de savoir câbler également une led active basse à une GPIO.

2.2.2.5 Lecture d'un port

Pour lire l'état des 8 bits d'un port, par exemple le port A, chaque bit étant imposé par un interrupteur par exemple, il faut utiliser le SFR de nom PORTA. Ci-dessous, un exemple :

```
unsigned char valeurPortA; // variable 8 bits non signée qui permettra de stocker l'état du port A
...
valeurPortA = PORTA; // lecture du port A et stockage de sa valeur dans la variable valeurPortA
```

La lecture d'un seul bit du port est également possible. Pour par exemple déclencher une action quand le bit 3 du port A est à 1, on écrit :

```
if(PORTAbits.RA3 == 1)
{
    // Déclencher action
    ...
}
```

Remarque : il est bon de savoir câbler un bouton poussoir actif haut à une entrée GPIO ainsi qu'un bouton poussoir actif bas.

3 Premier projet TP1A : commande des sorties

Attention :

- Cette partie 0 est à voir comme un tutoriel sur lequel il faudra revenir à chaque fois que vous souhaitez créer un projet.
- Choisissez pour l'ensemble du module INFO2 des noms de fichiers et de répertoires « classiques », c'est-à-dire sans espace ni accent.

3.1 Cahier des charges et objectifs

Le projet **TP1A** doit configurer le port D intégralement en sortie (étape 1) puis allumer une led sur deux (étape 2) : les leds connectées à RD6, RD4, RD2 et RD0 sont allumées, les autres sont éteintes.


Ce premier projet a surtout pour objectif de prendre en main l'environnement de développement MPLAB X. Il permet en outre de se familiariser avec les sorties numériques (ou TOR).

3.2 Choix du microcontrôleur, création des répertoires

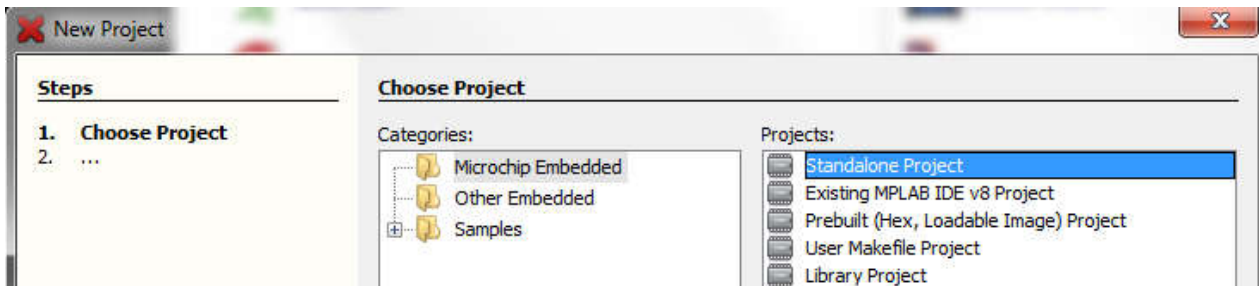
- Allez dans votre disque personnel (Z), et créez-y un répertoire **TPINFO2**.

Attention : C'est l'unique fois du semestre où vous avez dû créer ce répertoire **TPINFO2**. Il contiendra tous vos TP du module INFO2.

Au cours de ce premier TP, vous développerez plusieurs **projets** simples de nom **TP1A**, **TP1B**, **TP1C**, etc. Lors du deuxième TP, les projets auront pour nom **TP2A**, **TP2B**, etc. Chaque projet aura son propre répertoire, vous allez voir un peu plus loin comment l'environnement de développement MPLAB X s'en chargera. Pour une vue de ce à quoi doit ressembler votre répertoire **TPINFO2** dans quelques TPs, reportez-vous à la **partie 0**.

- Lancez MPLAB X IDE, l'environnement de développement intégré (*Integrated Development Environment*, IDE).
- Cliquez sur l'icône New Project  de la barre d'outils.
- Choisissez Microchip Embedded dans Categories puis Standalone Project dans Projects. Cliquez ensuite sur le bouton Next>.

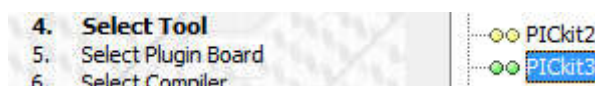
Remarque : Les options qui vont être réglées ici pourront toujours être corrigées par la suite par File / Project Properties.



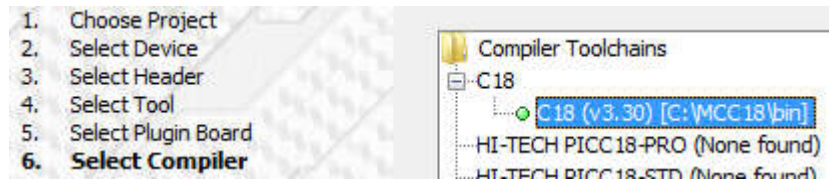
- Choisissez ensuite la famille et le composant comme dans la figure ci-dessous et cliquez sur Next>.



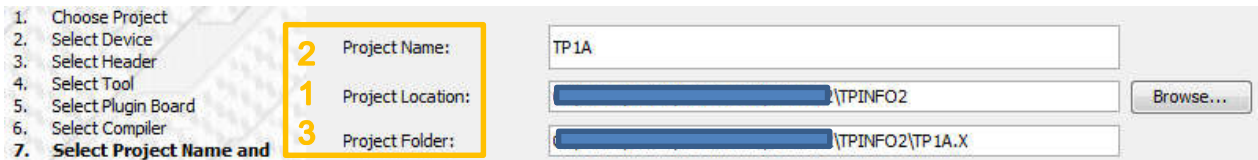
- Choisissez ensuite le programmeur/débogueur, ici un PICkit3 (la petite boîte rouge transparente au dos de la carte d'évaluation), et cliquez sur Next>.



- Choisissez comme compilateur C C18 (v3.30) et cliquez sur Next>.



- A l'étape suivante (photo ci-dessous), veillez à bien être dans **Z:\TPINFO2** au niveau de la rubrique **Project Location**. Au besoin, réglez ceci en cliquant sur le bouton **Browse...**. Si vous n'avez pas **Z:\TPINFO2**, cela ne marchera pas !



- Saisissez enfin le nom du projet à la rubrique **Project Name**, par exemple **TP1A**. On voit alors au niveau de la rubrique **Project Folder** qu'automatiquement un répertoire du projet est créé au sein de **TPINFO2** avec pour nom celui du projet complété par **.X**, c'est-à-dire **TP1A.X**. Cliquez enfin sur **Finish**.

3.3 Ajout du fichier de configuration des bits dans le répertoire des TP

Attention : La manipulation qui suit est à faire une seule fois pour l'ensemble des TP du module INFO2.

- Récupérez sur le disque **partages** dans son répertoire **ressources_info2** le fichier **configuration_bits.h** et placez-le dans le répertoire **TPINFO2**. Ce fichier contient le réglage de ce que Microchip appelle les *configuration bits* du PIC18F4520 : choix de l'oscillateur (horloge), validation ou pas du *watchdog timer*, etc. Si vous êtes intéressé par le détail de leurs valeurs, reportez-vous à la **partie 10.1**.

3.4 Ajout du fichier de script du linker dans le répertoire des TP

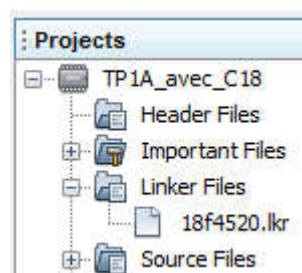
Attention : La manipulation qui suit est à faire une seule fois pour l'ensemble des TP du module INFO2.

- Récupérez sur **ressources_info2** le script du *linker* **18f4520.lkr** et placez-le dans le répertoire **TPINFO2**.

Linker signifie **éditeur de liens**. Celui-ci range de façon cohérente en mémoire de programme le contenu des différents fichiers objet du projet et établit des liens entre eux. Ce type de liens est par exemple nécessaire lorsque l'appel d'une fonction est dans un fichier objet différent de l'implémentation de la fonction elle-même. Le script du *linker* donne des indications sur comment doit être effectué de façon cohérente le rangement en mémoire.

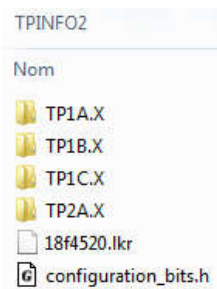
Attention : La manipulation qui suit est à faire pour chaque projet.

- Dans la zone **Projects**, faites un clic droit sur **Linker Files** puis **Add Existing Item...**. Sélectionnez dans le répertoire **TPINFO2** le fichier **18f4520.lkr** :



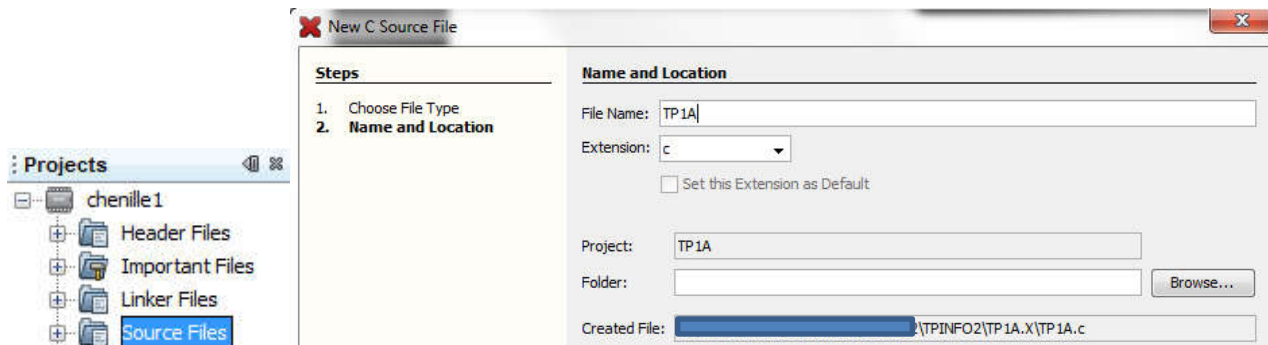
3.5 Arborescence du répertoire TPINFO2

Voici l'arborescence que vous devriez obtenir pour **TPINFO2** au bout de quelques projets développés :



3.6 Ajout d'un nouveau fichier C au projet

- Dans la zone Projects, faire un clic droit sur Source Files puis New puis C Source File....



- Donnez un nom à ce fichier C, par exemple **TP1A**. L'extension ***.c** est ajoutée automatiquement.

Remarque : Il n'est pas obligatoire de donner le même nom que celui du projet.

- Saisissez le code suivant (sans « zapper » les premières lignes de documentation) :

```
/* Auteur      : OL
 * Creation    : 26/05/13
 * Derniere MAJ : 29/05/13
 * IDE         : MPLAB X
 * Compilateur : C18
 * Programme pour la carte EasyPIC v7 avec un PIC18F4520 (boitier DIP40)
 * Configure le port D en sortie et dépose une valeur dessus
 */

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

void main (void) {
    // 1 - Met tout le port D en sortie
    A compléter
    // 2 - Allume une led sur deux du port D
    A compléter
    // 3 - Fige le pointeur de programme (Program Counter) afin de figer le programme
    while(1);
}
```

3.7 Explication des deux directives préprocesseur « #include »

Dans un fichier C, tout ce qui commence par **#** n'est pas une instruction du langage C mais une **directive** à l'attention du **préprocesseur**. Le préprocesseur est un programme qui s'exécute juste avant le compilateur.

#include <p18f4520.h> permet d'inclure le **fichier d'en-tête p18f4520.h** dont l'intérêt a déjà été présenté à la **partie 2.2.2.2**.

#include "../configuration_bits.h" permet d'inclure le fichier de réglage des *configuration bits* évoqué en 3.3. **../** indique le répertoire parent du projet donc ici **TPINFO2/**. C'est bien là que l'on a placé **configuration_bits.h**.




Attention : Ces deux **include** devront être présents dans tout programme C écrit.

Remarque : La différence entre `#include <xx.h>` et `#include "yy.h"` : Le premier (avec les chevrons < et >) indique que `xx.h` est un fichier à aller chercher dans le répertoire des fichiers `*.h` logé dans le répertoire d'installation du compilateur. Le deuxième `include` (avec des guillemets ") indique que le fichier `yy.h` est dans le même répertoire que précédemment OU dans le répertoire du projet. Ce type d'`include` est en général relatif aux fichiers `h` créés par le programmeur.

3.8 Complétion du programme

- Compléter le programme de façon à réaliser le cahier des charges (cf. **partie 3.1**).

3.9 Validation du projet


- Cliquez sur l'icône Clean and Build Project  afin de compiler le(s) fichier(s) C et de réaliser l'édition des liens.
- S'il y a des erreurs ou *warnings* signalées dans la zone Output onglet (Build, Load), corrigez-les puis cliquez sur .
- Répétez l'étape précédente tant qu'il y a des erreurs ou *warnings*. Si tout va bien, on doit voir écrit BUILD SUCCESSFULL dans la zone Output onglet (Build, Load).
- Vérifiez que le programmeur est connecté à la carte d'évaluation et que l'alimentation est branchée. Envoyez le programme en mémoire de programme en cliquant sur . Vérifiez que le programme produit bien ce qui est attendu.

3.10 Inspection des mémoires et du code assembleur généré

- Dans la zone **Dashboard**, repérez :
 - combien d'octets de la mémoire de programme occupe ce court projet ;
 - combien d'octets de la mémoire de donnée occupe ce court projet.
- Cela vous paraît-il normal vu l'ampleur du projet (allumer quelques leds) ?

Explication : En fait, quelle que soit l'ampleur du projet, trois bibliothèques sont incluses lors de l'édition des liens (on le voit en éditant `18f4520.lkr`, le script du *linker*) :

- `c018i.o` : contient de quoi initialiser des variables initialisées et appeler la fonction `main()` ;
 - `c1ib.lib` : contient les fonctions standard C ANSI (`strlen()`, `atoi()`, etc.)
 - `p18f4520.lib` : est la bibliothèque spécifique du processeur utilisé, notamment pour gérer les modules périphériques.
- Observez le contenu de la mémoire de programme par Window / Pic Memory Views / Program Memory (ou regardez la figure suivante). Au niveau de la colonne **Label**, repérez `main`. C'est là que commence la traduction en assembleur de votre programme principal, la fonction `void main(void)`. Répondez aux questions suivantes :
 - A quelle adresse commence l'implémentation de votre fonction `main()` ?
 - Repérez les quatre instructions assembleur correspondant à votre fonction `main()`. Essayez de trouver à quoi correspond le `0xF8C` de l'instruction `MOVWF 0xF8C, ACCESS` (cf. copie d'écran ci-dessous).
 - Quelle est l'instruction en langage machine ou *opcode* correspond à l'instruction assembleur `MOVLW 0x55` ?
 - Quelle est la traduction en assembleur du `while(1) ;` ?

Program  Output - TP1A_avec_C18 (Clean, B

Line	Address	Opcode	Label	DisAssy
100	00C6	0E0F		MOVLW 0xF
101	00C8	6AEE	clear_loop	CLRF 0xFEE, ACCESS
102	00CA	62EA		CPFSEQ 0xFEA, ACCESS
103	00CC	D7FD		BRA 0xC8
104	00CE	0012		RETURN 0
105	00D0	6A95	main	CLRF 0xF95, ACCESS
106	00D2	0E55		MOVLW 0x55
107	00D4	6E8C		MOVWF 0xF8C, ACCESS
108	00D6	D7FF		BRA 0xD6
109	00D8	0012	_init	RETURN 0
110	00DA	FFFF		NOP
111	00DC	FFFF		NOP

Memory Program Format Symbol

4 Projet TP1B : clignoteur temporisé par boucle for

Source(s) bibliographique(s) de cette partie : **MPLAB_C18_User_Guide_51288j.pdf** (guide d'utilisation du compilateur C18), **Table 2.1**.

4.1 Cahier des charges et objectifs

La led connectée à RD0 doit clignoter avec une période d'une seconde : 500 ms allumée, 500 ms éteinte. La temporisation sera placée dans une fonction **void tempo(void)** constituée d'une boucle **for** ne faisant que consommer du temps. La temporisation sera ajustée par tâtonnement (10 périodes doivent prendre 10 secondes par exemple).

Ce projet permet d'apprendre à changer l'état d'un seul bit d'un port d'entrées/sorties et de réfléchir au choix du type d'une variable.

4.2 Présentation

- Avant de commencer ce nouveau projet, fermez le projet précédent par **File > Close Project**. Recommencez toute la séquence comme pour le premier projet. A chaque nouveau projet vous procéderez ainsi.
- Avec une boucle **for(a = 0; a < N; a++)** ; avec **N** valant par exemple moins de 65000 (cette valeur provoque une attente de plus de 500 ms avec un quartz à 8 MHz comme c'est le cas sur la carte d'évaluation), quel doit être le type de la variable **a** ?

4.3 Validation du projet

- Vérifiez que le cahier des charges est respecté.

5 Projet TP1C : clignoteur temporisé par une fonction de la bibliothèque delays.h

Source(s) bibliographique(s) de cette partie : **MPLAB_C18_Libraries_51297f.pdf** (documentation des bibliothèques du compilateur C18), **Partie 4.5 DELAY FUNCTIONS**

5.1 Cahier des charges et objectifs

La led connectée à RD0 doit clignoter avec une période d'une seconde : 500 ms allumée, 500 ms éteinte. La temporisation sera placée dans une fonction **void tempo(void)** utilisant les fonctions de temporisation explicitées dans **delays.h**.

Ce projet permet de se familiariser avec l'utilisation de fonctions déjà écrites présentes dans une bibliothèque.

5.2 Présentation

La temporisation du projet précédent n'est pas des plus précises. De plus, sa durée effective dépend du compilateur utilisé et de sa façon de coder une boucle **for** en assembleur. Afin d'éviter ce genre de défaut, nous allons utiliser les fonctions de la bibliothèque des temporisations explicitées dans le fichier **delays.h**.

Pour utiliser les fonctions décrites dans **delays.h**, il faut au niveau des **#include** de début de fichier écrire :

```
#include <delays.h>
```

Le quartz de l'oscillateur (horloge) du PIC est à 8 MHz sur la carte d'évaluation. Il faut 4 périodes de cette horloge pour exécuter une instruction. La durée d'une instruction est appelée un cycle (*cycle*) par Microchip.

- Combien de temps dure un cycle ?
- Les durées de temporisations sont donc des multiples de la durée d'un cycle. Trouvez combien de cycles il faut attendre pour réaliser la temporisation demandée (500 ms). Choisissez alors la fonction **Delay***TCYx()** et l'argument qui convient.

5.3 Validation du projet

Ecrivez votre programme et testez-le. Pensez-vous que la période de clignotement est exactement égale à 1 seconde ? Justifiez votre réponse.

6 Projet TP1D : lecture d'une entrée TOR

La led connectée à RD0 doit s'allumer si le bouton connecté à RE0 est appuyé.

Ce projet permet de gérer pour la première fois une entrée TOR afin que l'utilisateur influe sur le déroulement du programme.

6.1 Présentation

6.1.1 Niveau sur une entrée lors de l'appui sur le bouton qui lui est connecté

Etant donné la configuration de la carte d'évaluation, un appui sur un bouton connecté à une broche provoque un 1 logique sur cette broche et un 0 logique lorsque le bouton est relâché.

- Faites un schéma du câblage permettant cela.

6.1.2 Choix de la fonction d'une broche multiplexée

RE0 est appelé une **broche multiplexée** car elle peut jouer plusieurs rôles différents (un à la fois bien-sûr).

Au *reset* du microcontrôleur, la broche RE0 est une entrée analogique, c'est le cas de 13 broches du microcontrôleur. Avant même de configurer TRISE pour mettre RE0 en entrée numérique, il faut configurer RE0 en entrée/sortie numérique (TOR) plutôt qu'en entrée analogique. Pour faire cela sur RE0 et les 12 autres broches ayant un comportement semblable au *reset*, il faut configurer le SFR ADCON1 de la façon suivante :

```
ADCON1 = 0x0F;
```

De plus amples détails seront donnés lors de l'étude de la conversion analogique/numérique, il n'est pas besoin d'en savoir plus pour l'instant.

7 Projet TP1E : chenillard simple, sensibilisation au débogage matériel

7.1 Cahier des charges et objectifs







Réaliser un chenillard sur le port D : RD0 est allumée pendant 250 ms, puis c'est au tour de RD1 pendant la même durée, puis de RD2, etc. Quand RD7 a été allumée, c'est à nouveau au tour de RD0 de l'être. La temporisation sera réalisée à l'aide des fonctions de **delays.h**.


Ce projet permet de réfléchir à un algorithme un peu plus évolué que ceux des projets précédents.

7.2 Présentation

Pour réaliser le chenillard, on a le choix entre : réaliser des décalages sur une variable 8 bits initialisée à 0x01 ou stocker les motifs du chenillard dans un tableau que l'on parcourt. On dépose au moment adéquat la valeur de la variable sur LATD.

- Réalisez le chenillard par la première méthode indiquée.
- Ce projet se prête bien à un **débogage matériel**, surtout si vous avez des « effets de bord », c'est-à-dire des limites mal gérées. On exécute le programme instruction par instruction tout en observant les évolutions de la variable 8 bits et de LATD. Pour cela :

- Au lieu de lancer votre programme par , lancez-le par . Cela active le débogueur. Vous pouvez alors :
 -  : effectuer un reset,
 -  : faire s'exécuter le programme,
 -  : le mettre en pause,
 -  : faire s'exécuter le programme instruction par instruction sans rentrer dans le détail des fonctions,

-  : quitter le débogueur,
 - poser des Watch, par exemple une sur votre variable 8 bits et une autre sur le SFR LATD : Menu Debug/ Add New Watch.... Il est probable que la variable doive être déclarée en global pour pouvoir être observée par une Watch.
- Familiarisez-vous avec le débogueur, *a fortiori* si votre programme fonctionne mal !
 - Si votre programme fonctionne et si vous êtes en avance, implémentez la deuxième méthode (le tableau de motifs).

8 Projet TP1F : chenillard avec marche / arrêt

8.1 Cahier des charges et objectifs

Le cahier des charges est le même que précédemment avec l'ajout de la fonctionnalité suivante : le fonctionnement du chenillard n'a lieu que si le bouton connecté à RE0 est appuyé. Quand celui-ci est relâché, le chenillard se fige dans l'état où il est. Quand celui-ci est réappuyé, le chenillard repart d'où il était.

Ce projet est un peu plus évolué que le précédent au niveau algorithmique.

9 Projet TP1G : chenillard avec marche / arrêt et contrôle du sens de défilement

9.1 Cahier des charges et objectifs

Le cahier des charges est le même que précédemment avec l'ajout de la fonctionnalité suivante : le sens de défilement du chenillard est de droite à gauche si le bouton connecté à RE1 est relâché.

Ce projet est un peu plus évolué que le précédent au niveau algorithmique.

9.2 Présentation

RE1 a la même fonction au *reset* que RE0, c'est-à-dire entrée analogique. La modification faite précédemment sur ADCON1 suffit pour que RE1 devienne aussi une entrée/sortie numérique.

10 Annexes

10.1 Configuration bits

Lors des TP de INFO2, les *configuration bits* du PIC18F4520 sont réglés comme ci-dessous grâce au fichier **configuration_bits.h**.

300001	02	Oscillator	HS
		Fail-Safe Clock Monitor Enable	Disabled
		Internal External Switch Over Mode	Disabled
300002	1F	Power Up Timer	Disabled
		Brown Out Detect	Enabled in hardware, SBOREN disabled
		Brown Out Voltage	2.0V
300003	1E	Watchdog Timer	Disabled-Controlled by SWDTEN bit
		Watchdog Postscaler	1:32768
300005	81	CCP2 Mux	RC1
		PortB A/D Enable	PORTB<4:0> configured as digital I/O on RESET
		Low Power Timer1 Osc enable	Disabled
		Master Clear Enable	MCLR Enabled, RE3 Disabled
300006	81	Stack Overflow Reset	Enabled
		Low Voltage Program	Disabled
		Extended Instruction Set Enable bit	Disabled
300008	0F	Code Protect 00800-01FFF	Disabled
		Code Protect 02000-03FFF	Disabled
		Code Protect 04000-05FFF	Disabled
		Code Protect 06000-07FFF	Disabled
300009	C0	Code Protect Boot	Disabled
		Data EEPROM Code Protect	Disabled
30000A	0F	Table Write Protect 00800-01FFF	Disabled
		Table Write Protect 02000-03FFF	Disabled
		Table Write Protect 04000-05FFF	Disabled
		Table Write Protect 06000-07FFF	Disabled
30000B	E0	Config. Write Protect	Disabled
		Table Write Protect Boot	Disabled
		Data EEPROM Write Protect	Disabled
30000C	0F	Table Read Protect 00800-01FFF	Disabled
		Table Read Protect 02000-03FFF	Disabled
		Table Read Protect 04000-05FFF	Disabled
		Table Read Protect 06000-07FFF	Disabled
30000D	40	Table Read Protect Boot	Disabled